GUILHERME C. JANUÁRIO

ASLI schemes as a kernel convolved way to optimize stencil computation

Esquemas ASLI para optimização de computação stencil através de convolução do núcleo computacional

São Paulo
2021

GUILHERME C. JANUÁRIO

ASLI schemes as a kernel convolved way to optimize stencil computation

Esquemas ASLI para optimização de computação stencil através de convolução do núcleo computacional

Versão Revisada

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Doutor em Ciências.

Área de Concentração:

Engenharia de Computação

Orientador:

Profa. Dra. Tereza C. M. B. Carvalho

São Paulo
2021

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, _____ de _____ de _____

Assinatura do autor: _____

Assinatura do orientador: _____

Catalogação-na-publicação

Januário, Guilherme
ASLI Schemes as a Kernel Convolved Way to Optimize Stencil Computation / G. Januário -- versão corr. -- São Paulo, 2021.
195 p.

Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.CE625.6.1.1.3 - SUPERCOMPUTADORES 2.CE610.1.3 - ARQUITETURAS PARALELAS 3.CE610.1.2 - ANÁLISE DE DESEMPENHO 4.CE550.43.20 - OTIMIZAÇÃO MATEMÁTICA 5.CE610.6.1 - COMBINATÓRIA I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

AGRADECIMENTOS

Meus profundos agradecimentos a minha orientadora, Professora Doutora Tereza C. M. B. Carvalho, e a meus pais, pelo apoio durante todo o processo. Diferentes amigos encorajaram-me e deram suporte na discussão de ideias sobre o trabalho em diferentes fases. Nesse sentido, sou muito grato a Ricardo e Letícia, Leo, Koga, e em especial a minha querida Gabriela, que até mesmo colaborou durante a dedução de uma fórmula. Este trabalho contou em fase inicial com o apoio infraestrutural dos departamentos *Data Centric Systems* e *Systems Acceleration and Memory* do IBM Thomas J. Watson *Research Center*. Também sou grato a todas as pessoas com quem tive contato no LaSSu, o Laboratório de Sustentabilidade da USP durante esses anos.

$\sim \cdot \sim \cdot \sim \cdot \sim \cdot \sim \cdot \sim \cdot \sim \cdot \sim \cdot \sim \cdot \sim$

My deepest appreciation to my advisor Prof. Dr. Tereza C. M. B. Carvalho Tereza and to my parents. Different friends helped and fostered the discussion of ideas through different steps of my work. In this regard, many thanks to Ricardo and Leticia, Leo, Koga, and special thanks to my dearest Gabriela, who also contributed to deduce a formula. This work enjoyed in the early days the infrastructural support of both the Data Centric Systems, and Systems Acceleration and Memory departments of IBM Thomas J. Watson Research Center. Many thanks also to everyone at LaSSu, the Laboratory of Sustainability of USP, with whom I interacted during all these years.

RESUMO

Computação do tipo estêncil é notória por ter o desempenho computacional limitado pela capacidade da memória de acesso rápido (RAM). Nos computadores atuais, isso implica subutilização da unidade central de processamento nesse tipo de computação. Para buscar amenizar a limitação, diversas abordagens de reordenação da computação foram propostas na literatura, notoriamente subtipos de *space-blocking* e *time-blocking*. Objetiva-se neste trabalho introduzir uma nova técnica para optimização de computação estêncil, diferente de *space-blocking* e *time-blocking*. Computação estêncil implica várias iterações de travessia por todos os pontos de um domínio, com cada iteração atualizando cada ponto com base no valor prévio dos pontos vizinhos. A técnica introduzida, ASLI (*Aggregate Stencil-Loop Iteration*, Iteração Agregada do Laço Estêncil), funciona atualizando os valores dos pontos do domínio com o operador estêncil original convoluído consigo uma ou mais vezes. Ela implica percorrer o domínio dos dados menos vezes que em uma implementção mais direta, do estado da arte, sendo que cada travessia efetua mais computação com os dados carregados nos registradores. Este operador mais complexo cria novas oportunidades de reúso de valores presentes nos registradores, e aumenta a razão de FLOPs por carregamento de dados da memória (*load*). Esquemas de reúso de computação e de dados são desenvolvidos para os casos de 1-, 2-, e 3- dimensões. A Tabela de Influência é apresentada como meio de auxiliar no cálculo de coeficientes convoluídos e deriva-se uma sequência numérica relacionada. Para operadores estêncil 2D e 3D com formato estrelar, a quantia total de FLOPs aumenta, mas uma melhor interação com o subsistema de memória torna a abordagem benéfica em comparação a implementacões não-ASLI. ASLI possui implementação relativamente simples, permitindo que mais cientistas aproveitem da capacidade de seus conglomerados de supercomputação com mais facilidade. Monstram-se resultados de desempenho para uma variedade de plataformas, provando-se a viabilidade da abordagem e que esta pode ser aplicada junto a técnicas e solucões correntes, ajudando a aumentar o desempenho de outros métodos já existentes na literatura. Para melhor exibição de ASLI e de sua comparação com outras abordagens, este trabalho esboça uma metodologia e novas métricas para avaliação de computação estêncil, e talvez também de escalabilidade de acesso à memória de computadores. Pode-se entender ASLI como a aplicação de um princípio mais amplo, a Convolução de Núcleo de Computação, ao caso particular de computação estêncil. Desse ponto de vista a Tabela de Influência poderia colaborar na disseminação da Convolução de Núcleo a outras aplicações.

Palavras-chave: computação estêncil; supercomputação; Convolução de Núcleo; ASLI; iteração agregada do laço estêncil.

ABSTRACT

Stencil computation is notorious for having the performance limited by the main memory access. In current computers it implies underutilization of the central processing units. To cope with this limitation, multiple approaches relying on reordering the computation have been proposed, most notably variations of space-blocking and time-blocking. This work introduces a technique to speed up stencil computation, which is not based on space-blocking or time-blocking. Stencil computation implies multiple iterations of traversals through every domain point, with each iteration updating every point based on the previous values of the neighboring points. The technique introduced, named Aggregate Stencil-Loop Iteration (ASLI), works by updating the value of each domain point using the original stencil operator convolved with itself one or more times. The approach implies traversing the data domain fewer times than a straightforward iterative stencil implementation would, with each traversal performing more computation per data item fetched into registers. This more complex operator creates new opportunities for in-register data reuse and increases the FLOPs-to-load ratio. Computation and data reuse schemes are developed for its application to 1, 2, and 3-dimensional stencils. The Influence Table is presented to assist in the calculation of convolved coefficients. An integer sequence is derived. For 2D and 3D star-shaped stencils, the total number of FLOPs increases, but better interaction with the memory makes it beneficial even when compared with optimized non-ASLI implementations. ASLI is relatively easy to implement, allowing more scientists to productively extract better performance from supercomputing clusters. Performance results are shown for a variety of platforms, proving the soundness of the approach and exemplifying how it can be straightforwardly applied with existing techniques and solutions, helping to increase the performance of existing optimization methods. In order to better express ASLI and to enable comparison with other approaches, a methodology is outlined and new metrics are set forth for evaluating stencil implementations, and perhaps the scalability of memory access in a machine. ASLI can be regarded as the application of a broader principle, namely, Kernel Convolution, to the particular case of stencil computation. From this perspective, the Influence Table could promote the use of Kernel Convolution in other applications.

Keywords: stencil computation; supercomputing; kernel convolution; ASLI; aggregate stencil-loop iteration.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

AR	Associative Reorder
ASLI	Aggregate Stencil-Loop Iteration
CSE	Common Subexpression Elimination
CWR	Computation Wavefront Reuse
EFLOPS	Equivalent FLOPS
ES	Equivalent Stencil
FLOPs	Floating Point Operations
FLOPS	Floating Point Operations per Second
GLUPS	Giga-Lattice Updates per Second
GPU	Graphics Processing Unit
GS	Giga-Stencil Point Updates per Second
IT	Influence Table
KC	Kernel Convolution
NTK	Naivest Type of Kernel
NSK	Naive-Smart Type of Kernel
NUMA	Non-Uniform Memory Access
PWC	Pair-Wise Composition
RAM	Random Access Memory
RIS	Reduction in Iteration Space

TB	Time-Blocking
SIMD	Single Instruction, Multiple Data

CONTENTS

1 INTRODUCTION

Stencil computation emerges in a gamut of scientific applications, such as image processing and geometric modeling (ROTH *et al.*, 1997), solving partial differential equations (PDEs) through finite-difference methods (KRUEGER *et al.*, 2011), and seismic modeling, common in oil-industry practices (SHIMOKAWABE; AOKI; ONODERA, 2014). Stencils of size comparable to the ones analyzed in this work have also been used in the game-of-life and in the pricing of American put stock options (BANDISHTI; PANANILATH; BONDHUGULA, 2012; ODEGAARD, 2003). The importance and ubiquity of stencil computations have led companies to build special-purpose compilers (ROTH *et al.*, 1997), a trend that has its counterpart in efforts to develop auto-tuners and compilers, such as in (DATTA *et al.*, 2008). Christen et al. (CHRISTEN; SCHENK; CUI, 2012) show that stencil-codes usually perform poorly, relative to peak floating-point performance. Roth et al. (ROTH *et al.*, 1997) show that stencils must be handled efficiently to obtain high performance from distributed machines. All of these activities confirm the need of novel approaches, such as ASLI, to optimize stencil computation.

In a stencil computation, the values of a set of domain points is iteratively updated with a function of the recent values of neighboring points. It updates the value of an element usually with a sum of products (ROTH *et al.*, 1997) or with a sum of products of sums, if the symmetry allows factorization. The outermost loop in a stencil kernel is always an iteration through the time domain (BANDISHTI; PANANILATH; BONDHUGULA, 2012), sometimes for tens of thousands of time steps (CHRISTEN;

SCHENK; CUI, 2012). This work explores stencils that need just the values of the previous iteration, which is very common. Most often, the domain is represented by a rectilinear grid (CHRISTEN; SCHENK; CUI, 2012) or an equivalent Cartesian description. Constant-coefficient stencil operators are commonly found in the literature and in practice. In this case, the stencil operator, comprised by the structure of the neighborhood and coefficients applied to each neighbor, is the same for all domain points.

The here described approach to optimize stencil computations is centered on the data and is designed to exploit in-register data reuse. It was dubbed Aggregate Stencil-Loop Iteration (ASLI), and it can be considered an application of kernel convolution (KC) in the context of stencil computation. The concept and exemplification of kernel convolution is an incidental contribution brought up by this work. Instead of traversing the domain points for T iterations and applying a given stencil operator to every point, ASLI takes as input the desired Reduction in Iteration Space (RIS) and traverses the domain points for T/RIS iterations, applying an operator that is obtained by convolving the original operator with itself RIS-1 times. This reduces the *FLOPS-to-load from main memory* ratio, setting ASLI apart from techniques such as unrolling and jamming. ASLI performs more computation on the loaded data than a naive or non-ASLI (henceforth, straightforward) stencil implementation. This aspect creates new opportunities for in-register data reuse and thereby increases the *FLOPs-to-load from cache* ratio, setting ASLI apart from time-blocking techniques. Most of the exposition of concepts focus on star-shaped stencils of radius 1, symmetric and asymmetric, which are the 1D 3-point, the 2D 5-point, and the 3D 7-point. The experimentation chapters also include results from the technique applied to higher-order 1D, 2D and 3D cases, even when the stencil operator has asymmetric coefficients.

1.1 Motivation

The idea that later evolved into ASLI first emerged as a solution to another problem in another context. Once identified that stencil computation could be suitable for that same approach, intellectual curiosity motivated early sketches for the bidimensional case, as this case would be more intellectually demanding than the unidimensional problem. Moreover, if no solution could be readily developed for 2D stencils, then perhaps the technique could still be lacking wider application and momentum. Once an investigative path had been determined to approach 2D stencils, the author set forth to learn whether stencil was itself useful. Indeed, it seemed to be the case, as Section 1.1.1 comments, which gave final motivation for the research here presented.

1.1.1 A Basket of Applications and Optimization Approaches

Mattson et al. (MATTSON; WIJNGAART; FRUMKIN, 2008) used a star-shaped 2D, 5-point stencil to solve the steady-state of an anisotropic heat equation. The authors of (ROTH *et al.*, 1997) focused on 2D stencils of 5 and 9 points. Early authors have also studied how to optimize 2D, 9-point stencils in the context of seismic modeling (MYCZKOWSKI; STEELE, 1991), using accuracy of fourth order in space and second order in time (that is, they use the values of iterations N-1 and N when calculating for N+1). To simplify the exposition, they presented their technique for the 1D case. Hoefler and Schneider (HOEFLER; SCHNEIDER, 2012) focused on a 2D, 5-point stencil, then extended their analysis to a 4D, 9-point stencil. Stencils that need the value of more than one previous iteration, and stencils with more than three dimensions are out of the scope of this work.

Maruyama et al. (MARUYAMA *et al.*, 2011) analyze their techniques when applied to a 3D, 7-point stencil operator for diffusion. A 3D stencil of radius 2 (13 points) is applied to a real world application in (SHIMOKAWABE; AOKI; ONODERA,

2014), which uses the third-order Runge-Kutta scheme. Another work (DATTA *et al.*, 2008) focused on the 3D, 7-point stencil, analyzing performance on a single node. Bandishti et al. (BANDISHTI; PANANILATH; BONDHUGULA, 2012) analyzed 1D, 2D, and 3D operators of radius 1, among others. They investigated the following benchmarks: 1d-heat, 2d-heat, 3d-heat, game-of-life, apop, and 3D, 7-point. Wave2D is a 5-point stencil that applies finite difference in a 2D discretized grid (SAROOD; MENESES; KALE, 2013), whereas the canonical Jacobi2D iteratively applies a 5-point stencil (SAROOD; MENESES; KALE, 2013). The work (DURSUN *et al.*, 2009) teaches that stencil computation is also used in flow simulations, oceanic modelling, multimedia processing, quantum dynamics, and computational electromagnetics. The majority of the work in the literature focuses on rectangular and structured grids, but King and Kirby (KING; KIRBY, 2013) investigated optimizations for 2D stencils on unstructured meshes. This doctoral thesis does not explore unstructured meshes. Neither does this work analyze GPU's. For evaluations and strategies of various sizes of stencil in GPU's, see (NASCIUTTI; PANETTA, 2016; MACHADO; NASCIUTTI; PANETTA, 2018).

Basu et al. (BASU *et al.*, 2015) introduce partial sums, a technique otherwise similar to the computation wavefront reuse introduced here, but developed for only originally symmetric operators and just effective for large enough operators. Thus, when applied to their investigated stencils in isolation, partial sums is beneficial to the cube-shaped 3D, 27- and 125-point stencils, with very small improvements on smaller patterns. The interesting technique of semi-stencil (CRUZ; ARAYA-POLO, 2014) complicates the structure of the computation, something also achieved in a different way by the unrelated ASLI technique explored here, but semi-stencil is more effective for larger sizes, not that much for the 3D, 7-point, among other stencils. In the future it would be interesting to assess how semi-stecil interacts with the larger convolved operators generated by kernel convolution.

With multi-threaded wavefront diamond blocking, Malas et al. (MALAS *et al.*, 2015) achieve for the 3D, 7-point a speed-up of 2.6 over their spatial blocked implementation that nearly reaches the memory-bound performance limit. Theirs is a time-blocking technique. In this scenario, they claim that the intra-cache data transfer and the core performance are the bottleneck. A side effect of ASLI is to help with intra-cache latency and core performance. Finally, MODESTO (GYSI; GROSSER; HOEFLER, 2015) combines loop tiling and fusion. The technique applied is different from ASLI, for it does not convolve the necessary computation. At most it can be regarded as a trivial application of the concept of Kernel Convolution.

The vast majority of this thesis had been written up to 2016, with Sections 6.5 – 6.8 and 6.10 written in March, 2019. Since then, little novelty has been added to stencil optimization, with most work in the area usually exploiting new architectures, minor changes, or investigating larger operators. A more recent work is discussed in Section 2.5, since it requires more familiarity with stencil. That work points to the relevance of ASLI, Kernel Convolution, and other concepts and formulas developed here.

1.2 Objectives

The main and more general objectives of this work are:

- To propose a technique (ASLI) to speed up stencil computation that better exploits values already in cache or registers.
- How to apply varying degrees of the technique to the most common 1D, 2D, and 3D operators.

The following specific goals have been devised:

- Schemes to apply computation reuse with ASLI for 1D, 2D, and 3D stencils.

- How to work with asymmetric and high order operators.
- How to derive the stencil coefficients of the technique.
- How to work with the boundary conditions.
- Metrics to analyze the optimization achieved with ASLI.
- To show results from a multitude of machines and situations.

1.3 Method

The prototype of the idea that later became ASLI first emerged as an attempt to reduce the amount of divisions inside the main loop of a code for simulation. Knowingly, divisions take much longer than additions and multiplications to execute inside a common computer, so the idea was to increase a variable representing a denominator through multiple time steps, or iterations of the time domain. With the approach, during the succession of time steps the code would no longer solve the original problem, but a scaled version thereof. Only after a given amount of iterations the denominator would be applied, thus scaling the problem back to the original size. Dealing with the actual application at that time was more complex than the description above, and was harder than with the 2D stencil. This is discussed here because perhaps the least reproducible aspect of a method is the generation of the core idea itself. Although stencils commonly do not accompany a significant amount of divisions, by the time the proto-idea of ASLI was conceived the presence of divisions was a significant factor. The fact that a competition was involved might have stoked some creativity as well. When the approach was to be applied to stencil, the following assumptions and steps were taken:

1. Due to the nature of the computation it would be possible to merge (convolve) the expressions of multiple time steps, the exact way how to do it not being

important at this point in time. There would be no reason to know precisely how to merge the expressions in case the technique could not yield a speed-up.

2. Assessment that it would be possible to determine the 2D convolved coefficients, by a proto-idea of influence tables.

3. The same proto-idea showed that it would be possible to reduce computations in the convolved forms of 2D and 3D.

4. Development of a simple 2D code for GPU, quickly showing that the approach worked. It used mock coefficients, as by the time it was not clear how to derive them.

5. Simplifying assumption that using the correct convolved coefficients would not alter the performance results in the tests. Neither would altering the computations of the border points.

6. Development of 3D codes for GPU and CPU, both of which achieved interesting speed-up. From here on the focus would be solely on CPUs.

7. Search in the literature whether the technique had already been attempted or hinted at.

8. Study of the properties of asymmetric and high-order operators with the aid of influence tables, which turned out to be rather unexpected.

9. Study of 1D kernels as a basis for the systematization of the properties and alterations induced by ASLI.

10. Another round of search through the literature showed that some works had enough information in themselves to show that ASLI would accelerate even codes already highly optimized.

11. Simplifying assumption, commonly made on the literature, that the performance would not be hurt by how the value of the domain points ended up arranged in the memory in a way suitable for SIMD instructions. This way it was possible to start from the fact that it indeed was possible to arrange the values in such a manner, ignoring how to do it.

12. Development of the final codes and execution of experiments.

13. Systematization of how to determine convolved coefficients and how to update elements close to the borders.

Interestingly, items 10 and 11 above represent a dialogue with the literature, with item 10 helping to prove the viability of ASLI in various scenarios, and items 11 and 5 helping to focus on the steady-state part of the computation. Proving the general validity of ASLI with these assumptions would allow other researchers and developers to implement ASLI or similar techniques according to their very specific needs.

1.4 Organization

Chapter 2 introduces the reader to computations in stencil pattern and summarizes the main techniques to optimize it. Chapter 3 introduces the main technique of this work, which is Aggregate Stencil-Loop Iteration (ASLI), and some ancillary techniques. Chapter 4 and Chapter 5 delve deeper into the generation of convolved operators and theoretical analysis of ASLI. Chapter 6 further develops the concept of Influence Tables and how to derive convolved coefficients. Aggregate Stencil-Loop Iteration is put to practice through many experiments discussed in Chapters 7, 8, and 9. Chapter 10 concludes this work and discusses some implications of kernel convolution.

2 BACKGROUND ON STENCIL OPTIMIZATION

To make the discussion throughout this work more readily accessible, an implementation of stencil code and the related equations are shown in this chapter, followed by an exemplification of how to apply time-blocking to it. Variations of time-blocking are arguably the most advantageous techniques to speed-up stencil computation in the state-of-the-art. Therefore, some of the experiments evaluated for this work employ time-blocking approaches.

2.1 Stencil Computations

To introduce stencil computations and some well-known optimization techniques, first consider a 1D, symmetric, 3-point stencil. The underlying mathematics of this stencil is explored here and also used in Chapter 3 to introduce ASLI. A 3D example is then used to show an optimization that ASLI allows and benefits from. This introductory work describes and focus on the core of such techniques, which can be represented by the steady-state of the presented computations, that is, this work focus on the computation involved in the vast majority of the points. Therefore boundary conditions are often not considered here, as they have but a marginal effect on the performance of the computation. That said, a technique is also described herein, which enables researchers to deal with their specific border functions.

In straightforward stencil implementations, the new values of the domain points for time $T=t+2$ are calculated during a traversal through the same domain points. The

inputs are the stencil operator (consisting of the coefficients k_0 and k_1 in Figure 1a) and the values of the domain points for time $T=t+1$ as shown in Equation 2.1. Before this traversal, the values for time $T=t+1$ had been calculated through a similar traversal using the values of the domain points when time $T=t$. Equations (2.1)–(2.4) summarize the formulas used in these two sweeps through the stencil domain, and Figure 1a helps to visualize the operator. Figure 1b offers a pseudo-code implementation of the 1D, 3-point, executed for T time-steps. This code uses two arrays to hold values of two consecutive time iterations. For small-sized stencils, the performance of the code is typically memory-bound. This is especially true for the 1D example shown, that performs just 4 operations per calculated stencil output.

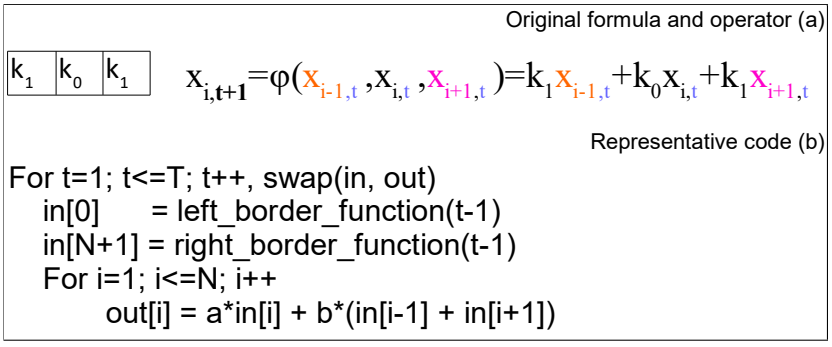
$$\begin{aligned} X_{i,t+2} &= \phi(X_{i-1,t+1}, X_{i,t+1}, X_{i+1,t+1}) \\ &= k_1 X_{i-1,t+1} + k_0 X_{i,t+1} + k_1 X_{i+1,t+1} \end{aligned} \tag{2.1}$$

$$X_{i-1,t+1} = k_1 X_{i-2,t} + k_0 X_{i-1,t} + k_1 X_{i,t} \tag{2.2}$$

$$X_{i,t+1} = k_1 X_{i-1,t} + k_0 X_{i,t} + k_1 X_{i+1,t} \tag{2.3}$$

$$X_{i+1,t+1} = k_1 X_{i,t} + k_0 X_{i+1,t} + k_1 X_{i+2,t} \tag{2.4}$$

Figure 1: Representative elements of a 1D-3point stencil computation.



Source: Author.

In Figure 1, the function *swap* merely changes what is being named *in* and *out* through the multiple iterations of the outermost loop. If *in* and *out* are vectors of dou-

ble precision in C language, this can be accomplished with the following preprocessor directive: *#define swap(x,y) {double* temp=x;x=y;y=temp;}*. The usual stencil implementation for a domain of size N makes use of two memory regions, each of size N. Initially one of the regions is filled with the initial values of the domain. The first iteration sweeps through this initialized memory region, reading the values of its points and writing the result onto the other memory region. The second iteration then reads this second region, writing the computation onto the other memory region, effectively overwriting the previous content. This process repeats for T steps. The *swap* operation does not involve exchanging the values written in each memory region, rather just selecting which region will be read from and written to during each sweep.

2.2 Definitions and Notations

Given the above considerations, it is possible to formalize stencil computation and related concepts that will be used throughout this work. Without loss of generality, Definition 1 focus on 1D symmetric stencil. This definition couples the concepts of an operator and an update equation. The idea of formalizing an update equation is to have it tightly linked to the implementation of the operator under analysis. For example, Line 5 of Algorithm 1 represents the update equation illustrated in Definition 1, whereas the commented-out Line 6 represents an update equation with two extra multiplications. Depending on the metric used to compare the performance of different implementations of a state-of-the-art operator, it is important to have in mind what was the exact update equation employed. Also note that in the stencil code representation of Algorithm 1 enough space is allocated to hold the values of the left and right borders, with $A[0]$ and $A[N-1]$ pointing to the leftmost and rightmost domain elements, so that $A[-1]$ and $A[N]$ can contain the values of border functions when appropriate.

Definition 1 (Stencil Operator (O), Update Equation)**.** *A symmetric* operator O *of radius* r *is determined by the* $r+1$ *coefficients* $c_0,...,c_r$ *and the* Update Equation

ALGORITHM 1: Stencil Computation

Input: T, the number of iterations. D, a pointer to the first element of the domain, where D[-1] is also allocated and represents the border condition.

Output: A vector with the domain points after T iterations of the stencil operator applied to every point.

```
1  pointer B = D;
2  pointer A = &(allocate(N + size_borders))[size_left_border];
3  for t=1; t <= T; t++ do
4      B[-1] = left_border_function[t-1];
5      B[N] = right_border_function[t-1];
6      for i = 0, i < N, i++ do
7          A[i] = c0 * B[i] + c1 * (B[i-1] + B[i+1]);
8          //A[i] = c0 * B[i] + c1 * (B[i-1]) + 1 * c1 * B[i+1];
9      end
10     temp = A; A = B; B = temp;
11 end
12 return B;
```

$X_i^{t+1} = c_0 X_i^t + \sum_{j=1}^{r} c_j (X_{i-j}^t + X_{i+j}^t)$.

The application of an operator to a point or to a domain is expressed in Definition 2, whereas Definition 3 formalizes the key concepts of a stencil computation.

Definition 2 (Application of an Operator ($O(D^t)$))**.** *The* application $O(X_i^t)$ *of O to the domain point X_i^t generates a new version of (or value for) the point through the Update Equation. The* application $O(D^t)$ *of the operator O to a domain version D^t is the generation of the domain version D^{t+1}, each element of which is obtained by $X_i^{t+1} = O(X_i^t)$, the application of O to every element of D^t.*

Definition 3 (Stencil Computation (S))**.** *A stencil computation $S = \langle O, T, N, D \rangle$ receives as input a domain D (or D^0) comprised of N successive elements and, for T iterations, generates the versions $D^1, D^2, ..., D^T$ of the domain, where each D^t is generated through the application $O(D^{t-1})$ of the stencil operator O to D^{t-1}.*

When applied to a practical problem, Definition 3 is joined with border functions. In 1D cases, theses functions calculate the values in the left and right borders, which are inputs to the update equation when applied to the first and last points, respectively. The

time spent with the idiosyncrasy of updating the border elements is usually irrelevant in face of the amount of internal elements. Therefore this work considers constant border functions in the experiments and in the majority of discussions. Section 6.10 shows how to incorporate different border functions into the technique discussed in this work.

2.3 Common Limitations to Stencil Performance

The majority of work on stencil optimization focus the study cases on small stencils, such as 1D of radius up to 4, 2D of radius up to 2, or 3D of radius 1 (BANDISHTI; PANANILATH; BONDHUGULA, 2012), being the versions of unitary radius the most common. All of these have in common the small code balance, or compute to memory access ratio. As a consequence of this and the gap between core frequency and memory bandwidth, such stencils are memory-bound and could not achieve but a small fraction of the peak floating point performance. In this scenario, many space-tiling techniques have been developed to help a kernel achieve its memory-bound roofline (e.g., in (RIVERA; TSENG, 2000)). Additionally, time-blocking techniques have been developed and studied for years to help with cache reuse, thus shifting said roofline (e.g., see (FRIGO; STRUMPEN, 2005) for a ramification of time-blocking). However, even with these approaches the fraction achieved of a machine's peak floating point performance is relatively small. Kernel Convolution comes to the aid of space- and time-tiling, leveraging their result. Sections 5.2 and 5.7 help to understand the constraints imposed to memory-bound kernels and how Kernel Convolution helps to circumvent then.

2.4 Space-Tiling and Time-Blocking

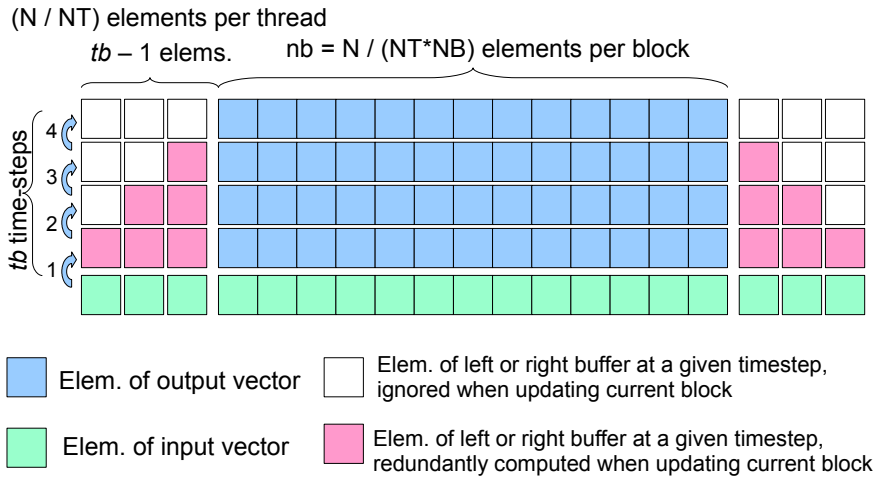
Space-tiling and time-blocking are the main techniques to speed-up stencil codes with low ratio of compute to memory access, thus memory-bound. They accelerate stencil computations by providing better reuse of data already cached. These approaches exploit the fact that the points $X_{i,t}$ do not necessarily have to be computed in the order implicit in Figure 1b. Time-blocking is especially interesting for the heavily memory-bound 1D, 3-point stencil. How to best apply time-blocking and space-tiling has for long been a subject of research (DATTA; YELICK, 2009; MALAS *et al.*, 2015), and many variations and flavors of time-blocking have been proposed. To further understand how to apply time-blocking and space-tiling, consider the following example, which also describes how both techniques were implemented in the 1D kernels evaluated in this work:

Divide a domain of N elements among N_T threads, each being responsible for updating N/N_T successive elements. The work of each thread is then split into NB blocks of size nb. Figure 2 depicts the work of one such thread. To be effective, nb must fit the cache (usually the last-level cache). A thread just processes the elements of the next block after every element of the previous block has been updated. The foregoing steps account for space-tiling. However, for 1D this is still not particularly interesting. In this scenario, the real effectiveness of space-tiling comes when time-blocking is further employed.

There are many approaches to apply time-blocking. The method employed by this work was not based on other methods of the literature. Additionally, no claims of novelty are made with regard to this approach, which has been named *boxed time-blocking*. It was developed to leverage the large level 3 memory cache of the machines where experiments with time-blocking were run. With this approach, the cores synchronize only after tb time steps, and each core divides its own work into smaller chunks,

to seek to benefit from the other cache levels as well. The approach employed defines a *boxed time-blocking* code of tb steps as a code that calculates the tb $\times$ nb values inside the space-time box composed by $X_{t+\theta,i} \forall \theta \in \{1,\ldots,\text{tb}\}, i \in \text{BLOCK}(k)$ before proceeding to the block $k+1$. There are NT threads and each thread updates the value of NB blocks. Therefore, a domain that contains N points implies that each block involves $\text{nb} = \frac{N}{\text{NT}\times\text{NB}}$ domain points.

Figure 2: Using boxed time-blocking to update one block of nb=13 elements with a time-blocking degree of tb=4.



Source: Author.

A direct consequence of this boxed time-blocking is that points in the borders of each block depend on points associated with the neighboring blocks. A naive approach to deal with the dependencies in the borders of each boxed block being processed is to process along with it the elements in its left and right borders, while keeping these extra elements in temporary arrays. Such arrays are necessary as, ultimately, the corresponding border elements will be (on the right) or have been (on the left) processed along with the corresponding neighboring block. Figure 2 depicts this process: The thread responsible for the output elements (blue) also calculates the border elements (pink), saving them in local arrays. The process is repeated for tb $= 4$ time-steps, until the fourth step when just the desired output elements are calculated. Such boxed strategy for time-blocking performs more calculation than necessary, as it calculates

the pink and blue values in Figure 2, instead of just the blue ones. For this and other reasons, this might be a naive approach when compared to literature.

2.5 An Approach Similar to ASLI

A recent innovation in stencil optimization is DCMI (KORAEI; FATEMI; JAHRE, 2019), which is an FPGA implementation of ASLI (JANUARIO *et al.*, 2016) that does not benefit from all computation reuse that ASLI makes possible in 2D and 3D. If also implemented in FPGA, perhaps the extra reuse could enhance their power-efficiency results even more. Still, the authors claim to obtain speed-ups of up to 7.7× compared to carefully optimized state-of-the-art accelerators, hinting at how widespread and beneficial ASLI could be. The authors also go to great lengths to show that the technique is applicable to various codes of the SPEC 2017 benchmark suite, hinting at the widespread applicability of Kernel Convolution. Interestingly, Koraei et al. are dismissive of the similarity of their work to ASLI, while at the same time misstating some key concepts. Some key passages of (KORAEI; FATEMI; JAHRE, 2019) are discussed in Appendix C, the understanding of which deepens the knowledge of kernel convolution and of how ASLI relates to the literature. Therefore, it seems that the most recent advancements in optimizing stencils of sizes similar to the ones explored in this work have been made by the very application of ASLI, which is this thesis' subject, to a different technology (FPGA). It also seems that some aspects of ASLI need to be further clarified, which is hopefully accomplished here, and also that ASLI can be beneficial even when not all the possible reuse it generates is employed. Hopefully other researchers will use the technique to investigate other ways of saving time and power consumption, contributing to stencil literature and to a more efficient use of the world's resources.

2.6 Main Points of the Chapter

Before the notion of kernel convolution (JANUARIO *et al.*, 2016) every approach to optimize stencil has calculated new values of the N points of the domain for T iterations, totalling, NT applications of the stencil pattern. Stencil codes usually have the performance limited by the main memory access. Space-blocking helps the code to approach this limit. Time-blocking helps to surpass this specific limitation. The literature of stencil optimization seemed to be stalled with trying different flavors of time-blocking and publishing results on the most recent machine architecture, including FPGA's.

3 AGGREGATE STENCIL-LOOP ITERATION (ASLI)

Aggregate Stencil-Loop Iteration (ASLI) is based on the insight that *it might be faster to compute the successive application of a given stencil for a given number of steps by, instead of naively following the specification, applying a different yet equivalent stencil pattern, for a smaller number of steps*. This is achieved by applying a convolved version of the original stencil. The convolved operator looks at a larger neighborhood than the original and should be applied fewer times to generate the same result, in machine precision. *ASLI thus leaves the underlying mathematical model untouched*. That is to say, the convergence of both operators is the same after both calculate the same number of equivalent steps (the concept of equivalence in this context will be detailed later). To this author's knowledge, every technique to speed-up stencil computation in the literature explicitly computes every value $X_{i,t}$. Whereas **spatial and temporal blocking** just benefit from the fact that such values do not have to be computed in a naive order, *ASLI takes a different direction and realizes that actually not every value has to be computed*.

ASLI works by convolving Equations (2.1)–(2.4) into Equation 3.1, where the formula for a domain value at time $T{=}t{+}2$ is expressed as a function of the values at time $T{=}t$ and of the coefficients e_0, e_1, e_2 obtained as shown in Figure 3a. This way, the sweep through the data points needed to compute the values of time T=t+1 is avoided. To achieve this, a new operator should be applied in lieu of the original operator, the new one looking at a larger neighborhood and performing a computation equivalent to

two iterations of the original operator. For this reason, the general concept was named **Kernel Convolution**, being ASLI its application to stencil computations. Figures 3 and 4 show that ASLI's new computation pattern completely skips the computation of values for time *T=t+1*. This halves the number of sweeps, as can be seen in Figure 4. Figure 3 also shows that, in just T/2 iterations, this code yields the same result the code of Figure 1 and Algorithm 1 calculate in T iterations. Equation 3.1 represents the update equation of internal domain elements, corresponding to the most time-intensive part of the computation. The equations for points close to the borders are represented in Figure 3b, and they require additional border convolved coefficients. This will be discussed later.

$$\begin{aligned} X_{i,t+2} &= \omega(X_{i-2,t}, X_{i-1,t}, X_{i,t}, X_{i+1,t}, X_{i+2,t}) \\ &= e_0 X_{i,t} + e_1(X_{i-1,t} + X_{i+1,t}) + e_2(X_{i-2,t} + X_{i+2,t}) \end{aligned} \tag{3.1}$$

As can be appreciated in the innermost loop of Figure 3b, applying Equation 3.1 to calculate two time iterations with a single pass through the domain requires three coefficients (e_0, e_1, and e_2) and the knowledge of more points at each calculation (5, as opposed to the 3 points needed in a straightforward implementation). But these 5 loads take the place of 6 loads in the straightforward implementation (3 loads for each of the two passes – compare the amount of black arrows of both patterns between $T = 0$ and $T = 2$ in Figure 4). Additionally, there are now more intermediate computations that can be reused; i.e., *ASLI creates opportunities for computation reuse*. For example, the product $e_2 X_{(i+2,t)}$ necessary to compute $X_{(i,t+2)}$ is also necessary to compute $X_{(i+4,t+2)}$. If kept in registers or cache, this intermediate product can be reused. *This reuse that just exists because of the kernel convolution is especially useful for 2D and 3D cases, where line and plane computations are reusable*. Section 3.1 offers some notation and terminology.

Figure 3: Representative elements of O^2 obtained by convolution with RIS=2 of O^1 of Figure 1.

Operator O^2 and related formulas (a)

e_2	e_1	e_0	e_1	e_2

$$e_0 \equiv k_0^2 + 2k_1^2$$
$$e_1 \equiv 2k_0k_1$$
$$e_2 \equiv k_1^2$$

$$x_{i,t+2} = \omega(x_{i-2,t}, x_{i-1,t}, x_{i,t}, x_{i+1,t}, x_{i+2,t})$$
$$= e_0 x_{i,t} + e_1(x_{i-1,t} + x_{i+1,t}) + e_2(x_{i-2,t} + x_{i+2,t})$$

Representative code for O^2 (b)

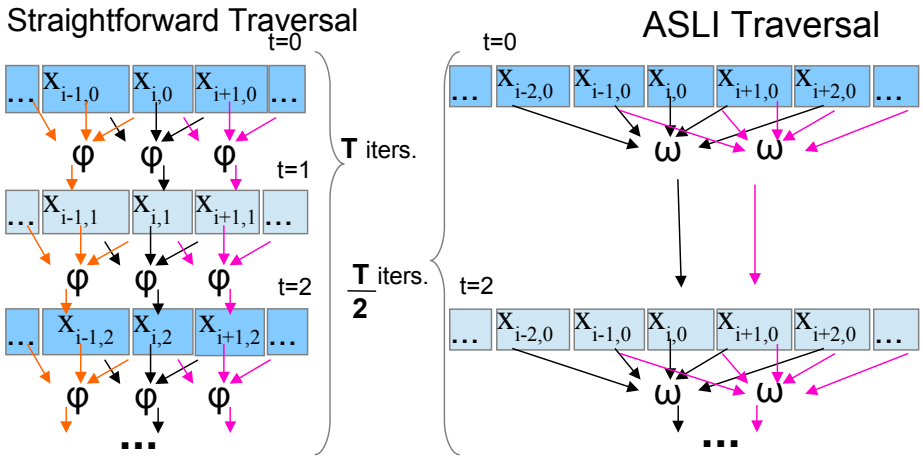
```
For t=2; t<=T; t+=2, swap(in, out)
    out[1] = e0_l*in[1] + e1*in[2] + e1_l*left_border_function(t-2)
                        + e2*in[3] + e2_l*left_border_function(t-1)
    out[2] = e0*in[2] + e1*(in[1]+in[3])
                      + e2*(left_border_function(t-2)+in[4])

    For i=3; i<=N-3; i++
        out[i] = e0*in[i] + e1*(in[i-1]+in[i+1]) + e2*(in[i-2]+in[i+2])

    out[N-1] = e0*in[N-1] + e1*(in[N-2]+in[N])
                          + e2*(right_border_function(t-2)+in[N-3])
    out[N] = e0_r*in[N] + e1*in[N-1] + e1_r*right_border_function(t-2)
                        + e2*in[N-2] + e2_r*right_border_function(t-1)
```

Source: Author.

Figure 4: Comparison of the dynamics of a straightforward traversal and a traversal according to ASLI.



Source: Author.

3.1 Definitions and Notation

Definition 4 formalizes kernel convolution applied to stencil, where **RIS** stands for Reduction in Iteration Space. For nomenclature or formulaic purposes, in many contexts O^1 can be named *original* or *simple* operator, and **RIS** might be denoted as Δ, designating the convolution degree.

Definition 4 (Kernel Convolution, RIS, simple stencil operator)**.** *Kernel Convolution is the process of solving a stencil computation* $S = \langle O, T, N, D \rangle$ *by, alternatively, computing* $S\prime = \langle O^{\text{RIS}}, \frac{T}{\text{RIS}}, N, D \rangle$*, where* O^{RIS} *is said to be the* convolved operator *of* degree RIS*, for which it holds that* $O^{\text{RIS}}(D^t) \triangleq \underbrace{O(O(...O}_{\text{RIS } times}(D^t) = D^{t+\text{RIS}}$*. The operator O of the original computation S can then be interpreted as* O^1 *and dubbed* original *or* simple *operator or form.*

The following notations help to understand the processes of Computation Wavefront Reuse and Pair-Wise composition described in the next sections. Later, these processes are used in the next chapter to analyze how to save FLOPs in convolved kernels even in case the original operator has radius greater than 1 or is asymmetric.

- $C_\Delta(K)$ denotes the transposed K-th column of the stencil operator O^Δ, so that, e.g., for Figure 5c, we have the transposed $C_3(2) = [3k_1^3, 3k_0k_1^2, 3k_1^3]$ and $C_3(7) = [k_1^3]$.

- $L_{t,(i,j)}(r)$ denotes the sub-column of $2r+1$ elements of the domain at time t, centered at point (i, j).

- $P_\Delta(K, r)$ denotes the transposed pair of coefficients along column K of O^Δ, r cells away from the central row of the operator O^Δ. For example (Figure 5c), $P_3(5,1) = [6k_0k_1^2, 6k_0k_1^2]$ and $P_3(6,1) = [3k_1^3, 3k_1^3]$.

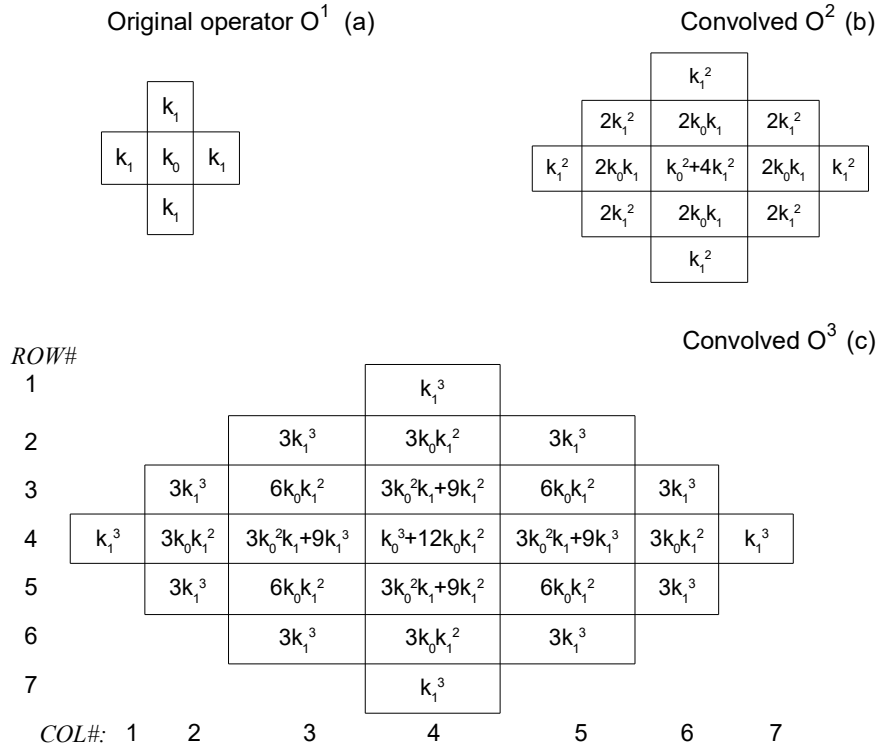
- Similarly, for a domain and time of interest, $Q_{t,(i,j)}(r)$ denotes the pair of

points along column j of the domain, apart r points from (i, j). For example, $Q_{t,(i,j+2)}(r) = [X_{t,(i-r,j+2)}, X_{t,(i+r,j+2)}]$.

3.2 Computation Wavefront Reuse (CWR)

Figure 5 depicts convolved versions of a symmetric 2D, 5-point stencil. Coefficients on the left and right side of the convolved operators are the same; that is, the original symmetry is kept. This allows for computation reuse of whole columns (or planes, if 3D) as the computation wavefront moves, e.g., to the right. This section shows how to benefit from such reuses that just emerge with ASLI.

Figure 5: Forms and literal coefficients of convolved operators for a symmetric 2D stencil.



Source: Author.

With the notation of Section 3.1, one can express the new value of a point being updated as a summation of inner products. For example, the update equation relative

to the application of O^1 (Figure 5a) to a domain D at time t can be expressed as:

$$\begin{aligned} X_{t+1,(i,j)} &= k_1 X_{t,(i,j-1)} + (k_1 X_{t,(i-1,j)} + k_0 X_{t,(i,j)} + k_1 X_{t,(i+1,j)}) + k_1 X_{t,(i,j+1)} \\ &= C_1(1) L_{t,(i,j-1)}(0) + C_1(2) L_{t,(i,j)}(1) + C_1(3) L_{t,(i,j+1)}(0) \end{aligned} \tag{3.2}$$

The update equation of O^3 has seven inner products:

$$\begin{aligned} X_{t+3,(i,j)} =& C_3(1) L_{t,(i,j-3)}(0) + C_3(2) L_{t,(i,j-2)}(1) + \\ & C_3(3) L_{t,(i,j-1)}(2) + C_3(4) L_{t,(i,j)}(3) + \\ & C_3(5) L_{t,(i,j+1)}(2) + C_3(6) L_{t,(i,j+2)}(1) + \\ & C_3(8) L_{t,(i,j+3)}(0) \end{aligned} \tag{3.3}$$

Equation 3.3 shows the update equation for point (i, j) when the convolved operator O^3 is applied to the domain at time $T = t$. One iteration of O^3 yields the same value of three iterations of O^1. Note that one of the necessary inner products to calculate $X_{t+3,(i,j+2)}$ is $C_3(3) L_{t,(i,(j+2)-1)}(2) = C_3(3) L_{t,(i,j+1)}(2)$, which, due to the symmetry, equals $C_3(5) L_{t,(i,j+1)}(2)$. Now, this last quantity had been previously computed when calculating $X_{t+3,(i,j)}$ (fifth member in the right-hand side of Equation 3.3). Thus, if saved in registers or cache, this inner product can be reused. This reuse enabled by kernel convolution is named ***Computation Wavefront Reuse (CWR)***. With one extra multiplication, *CWR* is also possible for asymmetric stencils (Section 4.1).

3.3 Pair-Wise Composition (PWC)

To present the technique of *Pair-Wise Composition (PWC)*, the notations of Section 3.1 are used. Recall that $P_3(5,1) = [6k_0k_1^2, 6k_0k_1^2]$ and $P_3(6,1) = [3k_1^3, 3k_1^3]$. As k_0 and k_1 are constants, $P_3(5,1) = hP_3(6,1)$ for some constant h. In fact, if O^1 is star-shaped this happens for varying values of h between any $P_\Delta(x,r)$ and $P_\Delta(y,r)$ of any pair of columns $C_\Delta(x)$ and $C_\Delta(y)$. Writing the update equations of $X_{t+3,(i,j)}$ and $X_{t+3,(i,j+1)}$

(similar to Equation 3.3), but focusing on selected inner products, we have:

$$X_{t+3,(i,j)} = ... + C_3(6)L_{t,(i,j+2)}(1) + ... \tag{3.4}$$

$$X_{t+3,(i,j+1)} = ... + C_3(5)L_{t,(i,(j+1)+1)}(2) + ... \tag{3.5}$$

Expanding the members shown in the right-hand side and using the pair-notation of Section 3.1, it comes:

$$\begin{aligned} C_3(6) \cdot L_{t,(i,j+2)}(1) =& 3k_1^3 X_{t,(i-1,j+2)} + 3k_0 k_1^2 X_{t,(i,j+2)} + 3k_1^3 X_{t,(i+1,j+2)} \\ =& 3k_0 k_1^2 X_{t,(i,j+2)} + P_3(6,1) Q_{t,(i,j+2)}(1) \end{aligned} \tag{3.6}$$

$$\begin{aligned} C_3(5) \cdot L_{t,(i,j+2)}(2) =& (3k_0^2 k_1 + 9k_1^2) X_{t,(i,j+2)} + P_3(5,1) Q_{t,(i,j+2)}(1) \\ &+ P_3(5,2) Q_{t,(i,j+2)}(2) \end{aligned} \tag{3.7}$$

Recalling that $P_3(5,1) = hP_3(6,1)$ for some constant h, Equation 3.7 can be rewritten as Equation 3.8 so that the quantity $P_3(6,1) \cdot Q_{t,(i,j+2)}(1)$ computed when updating $X_{t+3,(i,j)}$ can be reused when updating $X_{t+3,(i,j+1)}$. This is the *Pair-Wise Composition (PWC)*, applicable separately or in conjunction with *CWR* to save FLOPs in kernel convolved schemes.

$$\begin{aligned} C_3(5) \cdot L_{t,(i,j+2)}(2) =& (3k_0^2 k_1 + 9k_1^2) X_{t,(i,j+2)} \\ &+ hP_3(6,1) \cdot Q_{t,(i,j+2)}(1)) + P_3(5,2) Q_{t,(i,j+2)}(2) \end{aligned} \tag{3.8}$$

For concreteness, formulas involved in reusing one specific pair of coefficients were shown. It is clear that, as the computation wavefront moves (in this case towards higher values of j), more pairs emerge allowing more reuse in other columns (up to and including the central column of the operator – the columns on the left side benefit from *CWR*). In symmetric cases, with no gain or loss the programmer or dedicated compiler could choose to save only the summation of the members of $P_{1,(i,j+2)}$ and change the constant h accordingly, as $P_3(6,1) = [3k_1^3, 3k_1^3] = 3k_1^3[1,1]$. But this option does not work for asymmetric operators, whereas reusing the product $P_3(6,1) \cdot Q_{t,(i,j+2)}(1)$

does work, as discussed in Chapter 4

3.4 Basis of the Influence Table for Algorithm Designers

The computation of convolved coefficients is not always straightforward. To tackle this issue, the Influence Table (I.T.), first introduced in (JANUARIO *et al.*, 2016), is explored here. Influence Table is a tool the programmer can use to analyze the geometry and the coefficients of convolved operators. Similarly, a dedicated compiler could use I.T.'s instead of operating with symbolic logic. Finally, it allows researchers to explore convolution in other applications, neighborhoods, and perhaps in other computations as well.

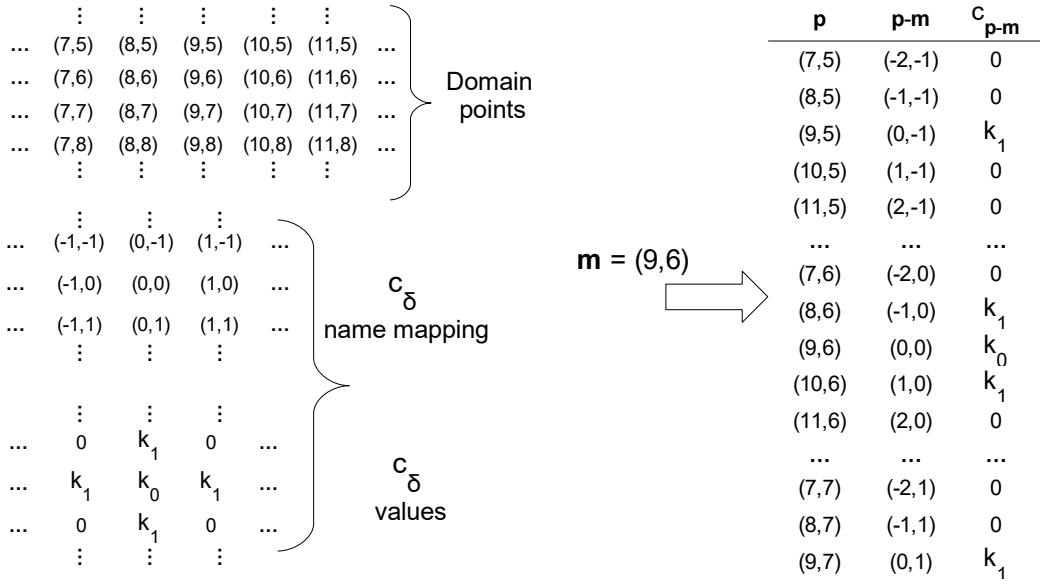
Theorem 3.4.1 (Isolation Principle)**.** *It is possible to isolate the contribution of the value $X_{\boldsymbol{q},t}$ of any point $\boldsymbol{q}$ at time-step $T = t$ to the value $X_{\boldsymbol{m},t+\mathrm{RIS}}$ of any point $\boldsymbol{m}$ at time-step $T = t + \mathrm{RIS}$.*

Theorem 3.4.1 is proven here for $RIS = 2$. A more general proof by induction is then presented. First, consider the original stencil operator O^1. It is possible to understand this simple or naive form as having coefficients c_δ, for every displacement vector $\delta = \mathbf{m} - \mathbf{n}$ that exists between any pair of domain points represented by the coordinate tuples $\mathbf{m}$ and $\mathbf{n}$. This way, the representation contains many more coefficients than just the traditional ones, but the vast majority of the coefficients are null. For example, in Figure 5c, O^3 has $c_{(-2,1)} = 3k_1^3$, $c_{(-3,1)} = 0$ (omitted in the figure), and $c_{(-3,0)} = k_1^3$. When applied to the domain at time $T = t$, O^1 has as update equation the Equation 3.9:

$$X_{\mathbf{m},t+1} = \sum_p c_{\mathbf{p}-\mathbf{m}} X_{\mathbf{p},t}, \forall \mathbf{m}, \mathbf{p} \in D. \tag{3.9}$$

Figure 6 exemplifies some of the variables of Equation 3.9 when a hypothetical

Figure 6: How to interpret the coefficients c_δ of a stencil in terms of displacement vectors, and how to apply it to update the value of point **m**, of coordinates (9, 6).



Source: Author.

domain point $\mathbf{m} = (9,6)$ undergoes the process of being updated with the original stencil O^1 of Figure 5a. Figure 6 also conveys that out of every possible c_δ of this O^1 solely five elements are not null. The claimed isolation can thus be proven for RIS $= 2$ as follows:

Proof. By the definition of the stencil operator, $X_{\mathbf{m},t+2} = \sum_{\mathbf{p}} c_{\mathbf{p}-\mathbf{m}} X_{\mathbf{p},t+1}, \forall \mathbf{m}, \mathbf{p}$ in the domain D. Expressing $X_{\mathbf{p},t+1}$ as function of the values at $T = t$, it comes:

$$X_{\mathbf{m},t+2} = \sum_{\mathbf{p}} c_{\mathbf{p}-\mathbf{m}} (\sum_{\mathbf{q}} c_{\mathbf{q}-\mathbf{p}} X_{\mathbf{q},t}) \tag{3.10}$$

$$= \sum_{\mathbf{q}} (\sum_{\mathbf{p}} c_{\mathbf{p}-\mathbf{m}} c_{\mathbf{q}-\mathbf{p}}) X_{\mathbf{q},t} \tag{3.11}$$

Finally, by regarding the term in parentheses in Equation 3.11 as the convolved coefficients e_δ, defined by Equation 3.12, the contribution of any $X_{\mathbf{q},t}$ to $X_{\mathbf{m},t+2}$ can be isolated as in Equation 3.13.

$$e_{\mathbf{q}-\mathbf{m}} \triangleq \sum_{\mathbf{p}} c_{\mathbf{p}-\mathbf{m}} c_{\mathbf{q}-\mathbf{p}} \tag{3.12}$$

$$X_{\mathbf{m},t+2} = \sum_{\mathbf{q}} e_{\mathbf{q}-\mathbf{m}} X_{\mathbf{q},t} \tag{3.13}$$

□

The same reasoning can be used to prove Theorem 3.4.1 by induction. Here, the above proof corresponds to the base step of induction. It is then necessary to show that, if the theorem holds true for $T = t + k$, then it is also valid for $T = t + k + 1$.

Proof. By renaming e in Equation 3.13 to accommodate multiple time-steps, it becomes Equation 3.14, which is true by hypothesis:

$$X_{\mathbf{m},t+k} = \sum_{\mathbf{q}} e^{k}_{\mathbf{q}-\mathbf{m}} X_{\mathbf{q},t} \tag{3.14}$$

Analogously to Equation 3.10, the expression for $X_{\mathbf{m},t+k+1}$ can be written as

$$X_{\mathbf{m},t+k+1} = \sum_{\mathbf{p}} c_{\mathbf{p}-\mathbf{m}} (\sum_{\mathbf{q}} e^{k}_{\mathbf{q}-\mathbf{p}} X_{\mathbf{q},t}) \tag{3.15}$$

$$= \sum_{\mathbf{q}} (\sum_{\mathbf{p}} c_{\mathbf{p}-\mathbf{m}} e^{k}_{\mathbf{q}-\mathbf{p}}) X_{\mathbf{q},t} \tag{3.16}$$

$$= \sum_{\mathbf{q}} e^{k+1}_{\mathbf{q}-\mathbf{m}} X_{\mathbf{q},t}, \text{ with} \tag{3.17}$$

$$e^{k+1}_{\mathbf{q}-\mathbf{m}} \triangleq \sum_{\mathbf{p}} c_{\mathbf{p}-\mathbf{m}} e^{k}_{\mathbf{q}-\mathbf{p}} \tag{3.18}$$

□

Theorem 3.4.2 (Influence Principle)**.** *If at time $T = t$ a given domain point $\boldsymbol{m}$ has value $X_{\boldsymbol{m},t} \neq 0$ and every other point has value 0, then the value $X_{\boldsymbol{p},t+RIS}$ of any point $\boldsymbol{p}$, $\boldsymbol{m}$ inclusive, at time $T = t + RIS$ is only dependent at time $T = t$ on the value $X_{\boldsymbol{m},t}$. Therefore, $X_{\boldsymbol{p},t+RIS} = e_{\boldsymbol{m}-\boldsymbol{p}} X_{\boldsymbol{m},t}$.*

Proof. In the theorem, $e_{\mathbf{m}-\mathbf{p}}$ denotes the coefficient $c_{\mathbf{m}-\mathbf{p}}$ of the operator O^{RIS}, obtai-

ned by the convolution of degree RIS of O^1, that is, the symbol e denotes a convolved coefficient. A convolution of degree 1 is defined as resulting in the same operator O^1. Theorem 3.4.2 is a direct consequence of the definitions and stems from the fact that the values that change at time $T = t+1$ result from a direct influence of $\mathbf{m}$, the only point with non-null value at time $T = t$. The values at time $T = t+2$ result from the direct influence of the values that had previously suffered direct influence from the point $\mathbf{m}$ and up to $T = t + \text{RIS}$. This way, every $X_{p,t+\text{RIS}}$ can be traced back to a multiplication of $X_{m,t}$, so that there exists a number $w_{\mathbf{m},\mathbf{p}}$, for which it holds that $X_{\mathbf{p},t+\text{RIS}} = w_{\mathbf{m},\mathbf{p}} X_{\mathbf{m},t}$.

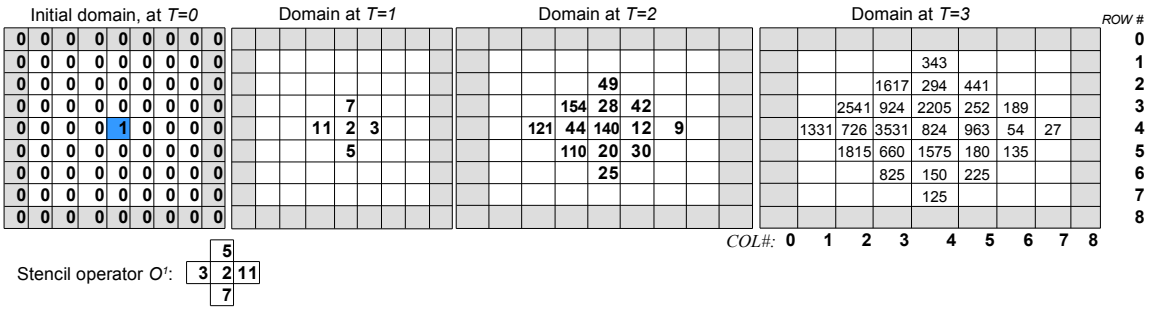
On the other hand, we know by the Isolation Principle (Theorem 3.4.1) that $X_{\mathbf{p},t+\text{RIS}} = \sum_{\mathbf{q}} e_{\mathbf{q}-\mathbf{p}} X_{\mathbf{q},t} \forall \mathbf{p}, \mathbf{q} \in D$. By construction, $X_{\mathbf{m},t}$ is the only non-null point at time t. Thus the previous summation can be reduced to $X_{\mathbf{p},t+\text{RIS}} = e_{\mathbf{m}-\mathbf{p}} X_{\mathbf{m},t}$, and $w_{\mathbf{m},\mathbf{p}} = e_{\mathbf{m}-\mathbf{p}}$. □

The Influence Principle (Theorem 3.4.2) states that when the original operator is applied **RIS** times to a domain that has only one value different from zero, then the resulting values can be traced back to a convolved coefficient. This outcome is explored in Theorem 3.4.3 to yield the Influence Table, which conveys practicality to the study and application of kernel convolution, thus being paramount to the understanding of the technique. Although the Influence Table is named after a bi-dimensional application of the technique, Theorems 3.4.1— 3.4.3 bear no restriction with respect to the dimensionality, thus being n-dimensional.

Theorem 3.4.3 (Influence Table)**.** *If $X_{\boldsymbol{m},t} = 1$ and $X_{\boldsymbol{p},t} = 0, \forall \boldsymbol{p} \neq \boldsymbol{m}$, then the values of the domain points at time $T = t + \text{RIS}$ represent the operator O^{RIS}, flipped in every dimension and centered at $\boldsymbol{m}$ (i.e., with $e_{\boldsymbol{0}} = X_{\boldsymbol{m},t+\text{RIS}}$).*

Proof. To understand Theorem 3.4.3, first note that in this scenario $X_{\mathbf{m},t} = 1$ implies through Theorem 3.4.2 that $X_{\mathbf{p},t+\text{RIS}} = e_{\mathbf{m}-\mathbf{p}}$, $\forall \mathbf{p}$, $\mathbf{m}$ inclusive. Consider a point $\mathbf{q}$ and a displacement vector ρ such that $\mathbf{q} = \mathbf{m} - \rho$. Then, by Theorem 3.4.2,

Figure 7: Influence Tables of a stencil operator for RIS=1, 2, and 3.



Source: Author.

$X_{\mathbf{q},t+\mathrm{RIS}} = X_{\mathbf{m}-\rho,t+\mathrm{RIS}} = e_{\mathbf{m}-(\mathbf{m}-\rho)} = e_\rho$. By definition (Equation 3.13), when applying the operator O^{RIS} to m at time $T = t$, e_ρ is the coefficient that should be applied to the point $\mathbf{g}$ for which $\mathbf{g}-\mathbf{m} = \rho$. That is, the current value of point $\mathbf{q} = \mathbf{m}-\rho$ at time $t+\mathrm{RIS}$ is the coefficient that should be applied to point $\mathbf{g} = \mathbf{m}+\rho$ when the operator O^{RIS} is applied to $\mathbf{m}$. In other words, the current values of domain points represent the convolved stencil operator O^{RIS}, centered at $\mathbf{m}$ and flipped in every dimension, which enables one to obtain the values at time $T = t+\mathrm{RIS}$ straight from the values at time $T = t$, skipping the explicit computation of values at any intermediate time-step $t < \mathrm{ts} < t+\mathrm{RIS}$. □

3.5 Influence Table Exemplified for 2D Asymmetric Stencil

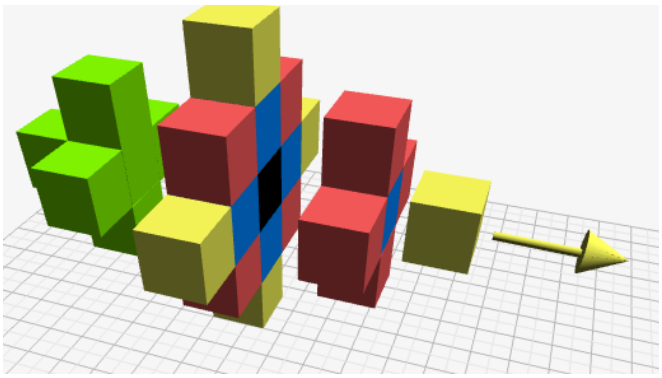
Theorems 3.4.1 and 3.4.3 are the basis of the Influence Table, explained below. The Influence Table (I.T.) is a useful way to calculate the coefficients of convolved operators and can be used either by compilers or programmers, who can implement it manually or with a spreadsheet processor. To obtain an I.T., start with a representation of a domain where just one element has value 1 (e.g., the blue point in the domain represented in Figure 7). Then, apply the operator under analysis to every domain point to get the I.T. relative to RIS=1. The successive application of the original operator to every point generates successive Influence Tables. Figure 7 shows the I.T.'s relative

to RIS=1, 2, and 3 for a hypothetical, completely asymmetric, 2D stencil of 5 points. Note that the I.T. relative to RIS=1 is the original stencil operator O^1, flipped in both dimensions and centered at that only point that had value 1 at $T = 0$, as could have been predicted by Theorem 3.4.3. Likewise, the I.T. relative to $T = 2,3$ is a flipped representation of O^2, O^3. Gray cells represent the points outside the domain borders, or ghost regions. Zeros are omitted in the figure for clarity. Influence Tables can also be used for 1D and 3D stencils. Chapter 4 further explores its worth and some insights it allows.

3.6 Computation Reuse and Intrinsics of 3D ASLI

ASLI was introduced with a 1D stencil and the Influence Table with a 2D stencil. Now a 3D stencil is used to explore new possibilities of data reuse and an implementation in C. First, consider a symmetric 3D stencil of 7 points, such as $\text{out}_{x,y,z} = k_0 \times \text{in}_{x,y,z} + k_1 * \times(\text{in}_{x+1,y,z} + \text{in}_{x-1,y,z} + \text{in}_{x,y+1,z} + \text{in}_{x,y-1,z} + \text{in}_{x,y,z+1} + \text{in}_{x,y,z-1})$.

Figure 8: Updating one domain point (central, black) by applying ASLI of RIS=2 to an originally 3D, 7-point stencil.



Source: Author.

According to the notation used in Chapter 3, the stencil above has $c_{(0,0,0)} = k_0$ and $c_{(\pm1,0,0)} = c_{(0,\pm1,0)} = c_{(0,0,\pm1)} = k_1$. When computed with kernel convolution through ASLI, the originally 3D, 7-point stencil requires information from 25 points per calculation as shown in Figure 8. However, the technique also brings the opportunity to

reuse intermediate computation. In Figure 8, points of the same color (yellow, red, and blue) share the same convolved coefficient and represent computation still to be done. The convolved coefficients are $e_{00} = k_0^2 + 6k_1^2$ (for the one black point), $e_{01} = 2k_0k_1$ (for the five blue points), $e_{11} = 2k_1^2$ (for the eight red points), and $e_{02} = k_1^2$ (for the five yellow points). The green points represent multiplications and additions already performed, in the current time-step iteration, when the computation wavefront reaches the current point (the black point in Figure 8). According to the principles of CWR (Section 3.2), the result of such computations may reside in an intermediate variable, at the discretion of the programmer. Additionally, as the stencil domain is being traversed in the arrow direction, just 13 points need to be explicitly loaded by the kernel, namely the red and yellow points.

To understand every surface reuse that ASLI enables in 3D stencils, it helps to see the computation of Figure 8 as composed of five slices or planes. It has been commented above how to reuse the computation of the two planes on the left (green). Now, when calculating $X_{t+2,(k,i,j)}$, the contribution of the four blue points in the mid-slice is of the form $A_{k,i,j} = e_{01} \times (X_{t,(k,i-1,j)} + X_{t,(k-1,i,j)} + X_{t,(k,i+1,j)} + X_{t,(k+1,i,j)})$. The contribution of the four red points in the second to the right slice is $B_{k,i,j} = e_{11} \times (X_{t,(k,i-1,j+1)} + X_{t,(k-1,i,j+1)} + X_{t,(k,i+1,j+1)} + X_{t,(k+1,i,j+1)})$. Therefore, if $B_{k,i,j}$ is saved, e.g., by residing in registers, when it is time to calculate $X_{t+2,(k,i,j+1)}$ the contribution of the four blue points in the then mid-slice can be obtained inexpensively with $A_{k,i,j+1} = (e_{01}/e_{11}) \times B_{k,i,j}$. This reuse is reflected in Line 25 of Listing 3.1. In the same line, the variable *ring[0]* deals with the reuse of four of the green points. Additionally, this division is not supposed to be implemented as depicted in Listing 3.1, as in this case significant slow down would accrue. The division e_{01}/e_{11} is a problem constant and should be pre-computed once before even the first time iteration.

Listing 3.1 shows a possible way to implement the innermost loop of the stencil, according to an ASLI approach. In Line 2, W is the width or size of the dimension

Listing 3.1: Innermost loop of 3D 7-point ASLI with RIS of 2. The division in Line 25 is actually supposed to be implemented as a problem constant.

```
1 int col = ghost_radius;
2 for(w = 0; w < W; w++, col++){
3
4   // 1 READ, right-hand yellow point
5   axis[0]=axis[1];  axis[1]=axis[2];
6   axis[2]=axis[3];  axis[3]=axis[4];
7   axis[4]= in[d][row][col+2];
8
9   // 4 READS, four right-hand, red points
10  ring[0]=ring[1];  ring[1]=ring[2];
11  ring[2]=e11*(
12  in[d-1][row][col+1]+in[d][row+1][col+1]
13       +in[d][row-1][col+1]+in[d+1][row][col+1]);
14
15  // 8 READS, outermost ring, four remaining yellow and red points
16  om = e02*(in[d+2][row][col]+in[d][row+2][col]
17           +in[d-2][row][col]+in[d][row-2][col]);
18  om+= e01*(in[d+1][row+1][col]+in[d+1][row-1][col]
19     +in[d-1][row+1][col]+in[d-1][row-1][col]);
20
21  // Final Computation
22  out[d][row][col] = e00*axis[2]
23     + e01*(axis[1]+axis[3])
24     + e02*(axis[4]+axis[0])
25     + ring[0]+ring[2]+(e01/e11)*ring[1];
26     + om;
27 }
```

being traversed. Note the variables necessary to implement the data reuse. The variables are the vectors *axis*, of five elements and *ring*, of three elements. This vector *ring* is not to be confused with the outermost ring (the 8 red and yellow points in the mid-slice), which offers no computation reuse. In this symmetric case, the surface computation could be reused as is. *In asymmetric cases the surface reuse is still possible, at the inexpensive addition of one multiplication*, as presented in Chapter 4 for the 2D case.

3.7 Main Points of the Chapter

A kernel convolution approach to optimize stencil computation attacks the problem internally by changing the update equation of the stencil and how many times it should be applied to each point, rather than passively only working with the immediate implications of the pre-defined stencil specification. The requirement is that the arithmetic result be deterministically maintained. After convolution, the new stencil pattern offers reuse of intermediate computation. It is expected that such reuses be exploited, helping to reduce the total amount of operations in 2D and 3D kernels. The convolved stencil operator increases the memory-bound limit that previous techniques took for granted, and previously existing optimizations are applicable to this stencil as well. The Influence Table can help to deduce the convolved coefficients during design or run time.

The main ways how kernel convolution alters the kernels are:

1. By decreasing the amount of computation needed (or FLOPs) in 1D, and increasing in 2D and 3D.
2. By increasing the maximum theoretically achievable compute rate (or FLOPS).
3. By shifting the operational intensity, thus the memory-bound roofline.
4. By changing the amount of reuse possible from the caches and register files.
5. By changing how easily the inefficiencies can be hidden.

4 MORE PROPERTIES OF ASLI OPERATORS

A striking consequence of convolving the original kernel is that the emerging operator offers computation reuse even in unexpected places. Some possibly unexpected reuses are explored here.

4.1 Patterns in Asymmetry

The previous chapters discussed the benefits of applying ASLI to symmetric cases. The question that arises is whether the approach would be beneficial with asymmetric stencil operators. This section proves that in such a case ASLI can be even more advantageous than when applied to symmetric operators. It considers the Influence Table (I.T.) of ASLI with RIS = 3 applied to a 2D, 5-point completely asymmetric stencil, both represented in Figure 7. The striking property that can be seen in O^3 is that column 2 is a multiple of column 6, that is, $C_3(2) = [2541, 726, 1815] = 13.44 \times [189, 54, 135] = 13.44 \times C_3(6)$. The same holds, with a different multiplier, between the pair of columns $C_3(3)$ and $C_3(5)$. This is a general and useful property of star-shaped stencils (even if 3D), allowing *CWR* also for asymmetric stencils. Recalling the pair notation for operators (Section 3.1), by which $P_3(6,1)$ denotes the pair of coefficients, along column $C_3(6)$ of O^3, away 1 cell from the central row (row r_4, in Figure 7), we can write $P_3(6,1) = [189, 135]$. Then we see that $P_3(5,1) = 1.33 \cdot P_3(6,1)$ and $P_3(4,1) = 11.66 \cdot P_3(6,1)$. Similarly, for the coefficients away 2 cells from the central row, $P_3(4,2) = 0.66 \cdot P_3(5,2)$. That is, not just *CWR*, but also *Pair-*

Wise Composition (PWC) is still applicable when the original stencil has asymmetric coefficients.

Each of these last three identities allows to reduce, for each pair, 4 FLOPs (one Mul and one Add for each element of the pair) to 2 FLOPs (one Mul of the summation, previously computed, of the elements of the original pair, followed by an Add). Additionally, the fact that $C_3(2)$ and $C_3(3)$ can be obtained from $C_3(6)$ and $C_3(5)$ allows us to reduce what would be, respectively, 6 and 10 FLOPs (Mul and Add for every element in the column) to 2 and 2 FLOPs (Mul and Add of the previous summation of every element in each column). Now, it can be seen in Figure 7 that the kernel convolved operator O^3 has 25 points (the non-blank cells) accounting for potential 49 FLOPs. It has been just shown that for this completely asymmetric case 18 FLOPs can be saved with *Computation Wavefront Reuse* and *Pair-Wise Composition*. Then, with $\Delta = 3$, the total FLOPs for asymmetric 2D 5-point kernel convolved stencil is 31. The straightforward stencil would require, for 3 iterations, $3 \times 9 = 27$ FLOPs which produces a FLOPs ratio of $27/31 \approx 0.87$.

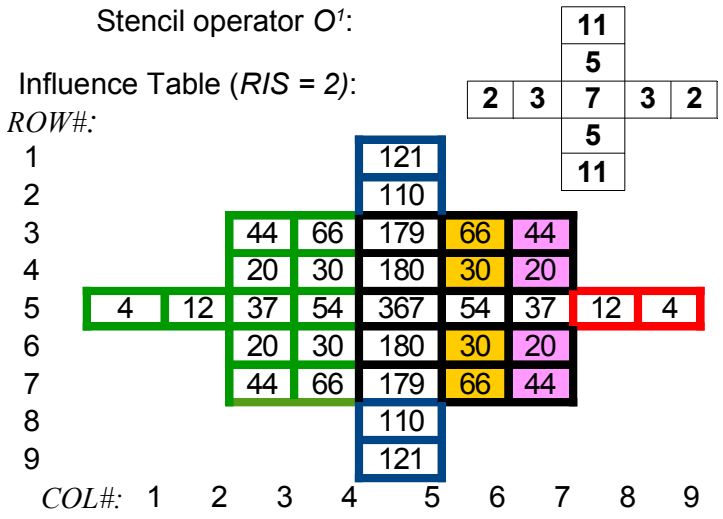
At first glance, a FLOPs ratio smaller than 1 would reflect a "speed-down". However, as ASLI shifts the computing algorithm to a higher computation-to-load ratio, the transformed kernel in reality executes faster, even if no SIMD instruction is used to benefit from this new ratio.

4.2 High-Order Operators in the 2D Case

This section exemplifies the application and dynamics of an ASLI scheme that further applies CWR and PWC to a high-order, 2-way symmetric, stencil of radius W. To do so, the section keeps track of the necessary FLOPs and compares it among different cases. Let us define a baseline operator by considering a completely symmetric (or 4-way symmetric) stencil: $\text{out}_{x,y} = k_0 \times \text{in}_{x,y} + \sum_{R=1}^{W} k_R \times (\text{in}_{x+R,y} + \text{in}_{x-R,y} +$

$\text{in}_{x,y+R} + \text{in}_{x,y-R}$), where k_R is the coefficient applied to the four neighboring points at distance R from the central point. The operator is said to have radius W and a total of 1+4W points. This operator has W quadruplets, each requiring 3 Adds among the four elements, a following Mul, and a final Add. The central point just requires one Mul, as the other intermediate calculations are added to it. After T iterations, the amount of FLOPs performed is $F_{sym}(T) = (1+5W)T$ FLOPs. Now let us consider an asymmetric operator, the operator in Figure 9, which we will classify as having half symmetry. It has W vertical pairs of coefficients (11 and 5) and W horizontal pairs. Each of the 2W pairs requires 1 Add between its elements, 1 Mul of this partial result, and another 1 Add. The central element requires 1 Mul. Thus $F_{\text{half-sym}}(1) = 1 + (2W \text{ pairs} \times 3 \text{ FLOPs per pair})$, and $F_{\text{half-sym}}(T) = (1+6W)T$.

Figure 9: 2D, radius W = 2, half-symmetric operator and its I.T. for RIS=2.



Source: Author.

To study and exemplify the application of kernel convolution to this high-order and asymmetric stencil, we start noting that the convolved operator[1] O^2 of Figure 9 can be divided into four parts: (a) the four left columns (green borders), (b) the upper and lower arms (blue, with coefficients 121 and 110), (c) the right arm (red), and (d) the right main body (black) composed of the fifteen cells delimited by rows 3 and 7

[1] Although an Influence Table (I.T.) is a flipped representation of the convolved operator, if there is symmetry along the axes of O^1, as is the case here, then the I.T. also matches the corresponding convolved operator.

and columns 5 and 7. Using *CWR*, the twelve elements in (a) are reduced to a sum of columns, needing just $2W-1$ Adds. The next contributions will be added to this partial sum. Other $3W$ FLOPs are needed to compute the W pairs in (b) (Add, Mul, Add, per pair). The right arm (c) demands $2W$ FLOPs (Mul, Add, per coefficient). By now, just (d) remain to be accounted for.

The *PWC* technique can be employed to reduce the FLOPs of this computation. In (d) the three coefficients along row 5 (in the example, 367, 54, and 37) contribute $2+2W$ FLOPs. Each of the W rightmost pairs of the computation wavefront, i.e., the two pairs $P_2(7,1)$ and $P_2(7,2)$ (pink cells), contribute 3 FLOPs, for a total of $3W$ FLOPs. Along the central column there are other W pairs remaining. With *PWC* they require only a Mul and Add, totaling $2W$ FLOPs. In (d), only $W-1$ columns remain to be considered, minus the axial elements in row 5. That is, just the four orange cells of $C_2(6)$ remain to be analyzed. Note in Figure 9 that the multiplier h from $P_2(6,1) = hP_2(7,1)$ is the same multiplier from $P_2(6,2) = hP_2(7,2)$. Thus in this case we can extend the notion of *PWC* to benefit from the equality $[66,30,30,66] = 1.5 \times [44,20,20,44]$, with $h = 1.5$. This is the manifestation of *PWC* for high-order stencils. Except for coefficients in the central row, the contribution of each of those $W-1$ columns can be calculated, with *extended PWC*, with a Mul and Add, making $2(W-1)$ FLOPs. Without reuse, this would have required quadratic $3W(W-1)$ FLOPs.

By summing up the individual contributions of parts (a) to (d), we get $F_{\text{sym},\Delta=2,r=W} = F_{\text{half-sym},\Delta=2,r=W} = 16W-1$. Table 1 summarizes the information. There is a limit to the ratio of FLOPs performed by a naive implementation and the corresponding convolved version. The table also shows that for the half-symmetric operator of Figure 9, kernel convolution can be beneficial if it helps improve the interaction of the program with the memory subsystem, registers, and other inefficiencies by a factor S for which $S \times 0.84 \geq 1$, or $S \geq 1.19$. The experimental results show that the strategy yields better time to solution even though it increases the total FLOPs in

Table 1: FLOPs to calculate 2 equivalent stencils, varying the symmetry. See Section 5.3 for the notion of equivalent stencils.

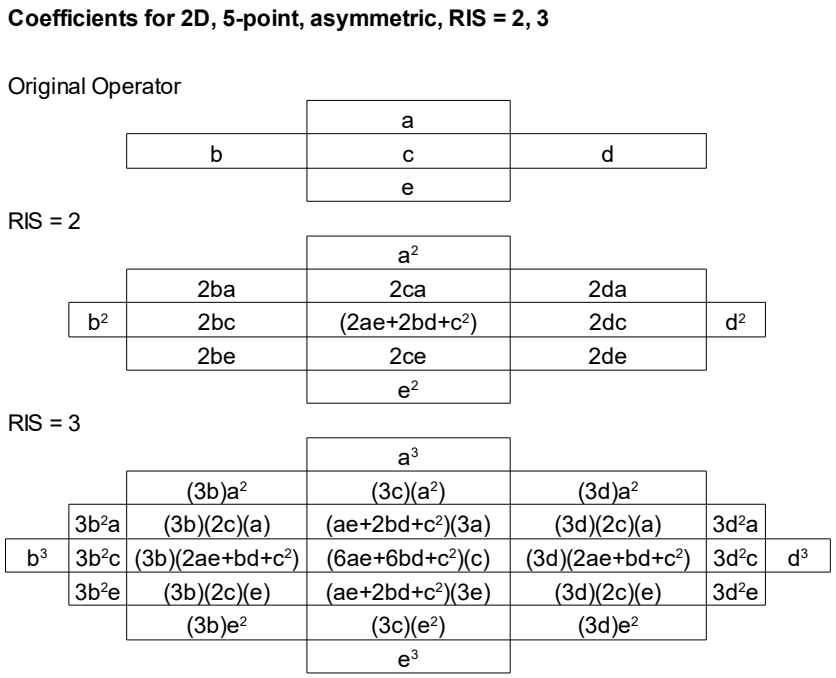
Radius:		2	3	4	5	6
FLOPs	O^1, sym.	22	32	42	52	62
	O^1, half-sym.	26	38	50	62	74
	O^2, both	31	47	63	79	95
FLOPs Ratio	sym.	0.71	0.68	0.67	0.66	0.65
	half-sym.	0.84	0.81	0.79	0.78	0.78

2D.

4.3 Putting It All Together

Figure 10 shows convolved versions of an asymmetric 2D, 5-point stencil expressed in literal variables. If we assume without loss of generality the original coefficients as $a = 2, b = -0.71, c = 3, d = -0.2, e = -1.1$, we obtain O^1 and a convolved O^3 as illustrated in Figure 11.

Figure 10: Depiction of convolved operators of an asymmetric original operator.

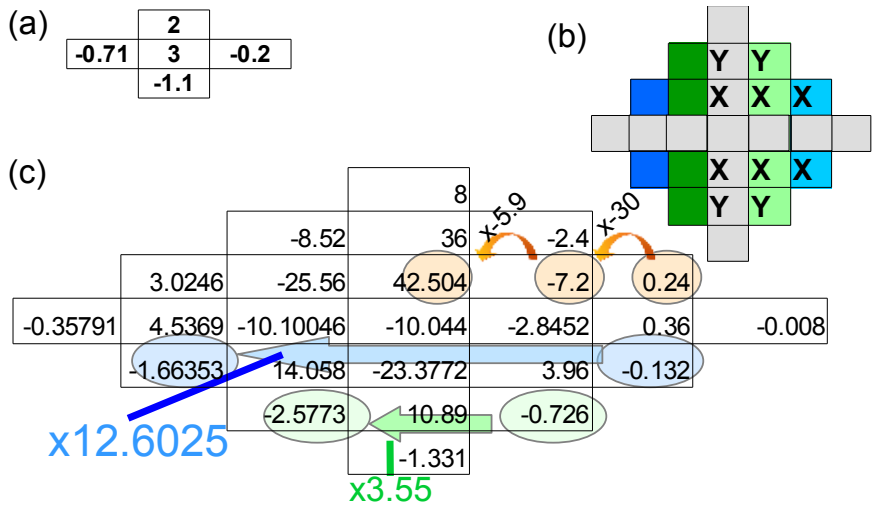


Source: Author.

Figures 10 and 11 illustrate that computation reuse exists also for asymmetric ope-

rators. This property happens because O^1 is star-shaped, and is not a consequence of the particular coefficients selected. In Figure 11, columns of same color and different shades are multiple of each other, including the corresponding axial coefficient (gray). The computation of the vertical axis (gray cells) offers no further reuse. Computation can also be reused while composing the columns: both elements of pairs marked with X or Y in a column are multiple, by the same factor, of the corresponding pair of any other column. Pair-wise reuse to compose columns have been introduced in Section 3.3. For example, it is possible to compute the summation of the individual contribution of the three elements in the dark blue column on the left side of O^3 as a multiple of the individual contribution of the three coefficients on the light blue column: $3.0246 = 12.6025 \times 0.24, 4.5369 = 12.6025 \times 0.36, -1.66353 = 12.6025 \times 0.132 \implies (3.0246 + 4.5369 - 1.66353) = 12.6025 \times (0.24 + 0.36 - 0.132)$.

Figure 11: Example of computation reuse for the convolved O^3 of an asymmetric operator O^1.



Source: Author.

4.4 On SIMD, and on Square and Cube Stencils

The convolution of square (2D) and cube (3D) operators results in still a square and cube stencil, but larger. If the original operator has symmetry, which is very com-

mon for such stencils, CWR and PWC can be applied. If the original operator has no symmetry, than, as a consequence of the diagonal coefficients, CWR and PWC cannot be applied as described here.

There are two immediate ways how ASLI offers a better exploitation of SIMD units. First, by changing the FLOPs-to-byte ratio, ASLI helps to keep the SIMD units busier, thus reducing the time wasted waiting for new data from memory. Second, as ASLI increases the stencil neighborhood, it creates more work to be done with neighboring values. This potentially could be exploited with ASLI to increase the speed-up already achievable with the ancillary techniques introduced here (CWR and PWC).

4.5 Main Points of the Chapter

The Influence Table helps designer to visualize properties of the computation being convolved. Once convolved, asymmetric stencils which originally offered no reusable computation at all start to offer reuse. Kernel convolution is applicable to high-order stencils as well, and also offer otherwise unexpected reuse.

5 FURTHER ANALYSIS OF ASLI

Having acquired a better understanding about the intrinsics of ASLI, it is now possible to make a deeper qualitative analysis of how this technique relates to other optimization techniques in the state-of-the-art. This chapter also introduces theoretical notions necessary to quantitatively compare ASLI implementations among each other and with more conventional approaches.

When introducing ASLI, it was said that kernel convolved operators were defined to produce the *same arithmetic result* as the simple operator. In other words the operations involved in solving a problem with a convolved stencil are rearrangements of the operations already entailed by the simple operator, obtained through manipulation of additions, subtractions, multiplications, and divisions. To compare the properties of different possible kernel versions, or more broadly to differentiate between approaches, it is paramount to assess whether they at least attempt to perform some common tasks equitably. The following concept is hence put forward:

Definition 5. *[Necessary FLOPs(f)] The necessary or useful FLOPs per application of the update equation of a stencil operator is the amount of FLOPs entailed by an update equation that (i) has no void arithmetic operation, such as a multiplication by 1 or an addition of 0, and (ii) has the smallest number of operations possible resulting from a combination of factorization and computational reuse*[1].

[1] Since division instructions generally take longer to complete, in case they are present it is useful to separate between Necessary Div FLOPs and Necessary Non-Div FLOPs. None of the kernel codes discussed in this work contain divisions within a significant loop.

The requirement (i) from Definition 5 ensures that the compute rate, usually expressed in GFLOPS, of a machine code that can be represented in C as `out[i] = in[i-1] + 1.*[i-1]` is not conflated with another represented as `out[i] = in[i-1] + [i-1]`. In a strictly memory-bound situation, both codes would take the same amount of time to calculate the same number of `out[]` elements, but the first version would have performed two operations per `out[]` element (one addition and one multiplication by 1), against one operation in the second case. This difference would artificially rig the performance indicator GFLOPS in favor of the first code, whereas in this case the indicators Giga-Stencil per Second and time-to-solution would be the same. The requirement (ii) helps with two tasks. First, it prevents one from falling in the same mistake as when conflating versions of `out[i] = K*in[i-1] + K*[i-1]` with `out[i] = K*(in[i-1]+[i-1])`, where factorization saved one FLOP. Here, the indicator GFLOPS could be rigged by 50% in favor of the first example. Second, it reminds that sometimes a factorization might be left out or performed just partially, in case some sub-products of the computation might be reused to save more FLOPs. CWR and PWC are good examples of this scenario.

5.1 Speed of light — ς

The *speed-of-light* performance is the best performance that a code with some properties reaches in a machine. This is a concept already present in the stencil-related literature, which this work promotes as being more relevant in the field of kernel convolution. Let ς denote the speed-of-light. As a rule of thumb, this indicator of upper bound performance can be found by measuring the performance a code achieves on domains big enough to benefit from pre-fetchers and other cache hacks, but smaller enough to prevent harmful data movement from main memory. Additionally, the measurement of ς is usually taken when just one core is employed, and the domain size is comparable to the resources available to that core, actually meaning that multiple tenta-

tive values will be measured for ς with varying domain sizes, and the best reproducible measurement will then become ς. It is good practice to choose a speed-of-light reproducible to avoid indicators coming from spikes in performance — unless, of course, the spikes themselves are deterministically reproducible.

To understand the significance of this speed-of-light, perhaps it is enough to realize that the scaling factors found in the literature are always (much) smaller than 100% for stencil computation, when the single-core performance is the baseline. In other words, if P_n denotes the best performance when n cores are employed, then $\varsigma = P_1$ and usually $\varsigma > P_n/n$. The speed-of-light is the Holy Grail of the scalability of a technique in a machine. Conversely, if two distinct techniques have different enough speed-of-light indicators, then it might be easier for the fastest technique in single core to be the fastest as well in multiple cores, even if this very technique has worst scalability. Once defined and measured as above, the value of ς lies somewhere between the per core memory-bound limit performance and the machine's peak floating point performance.

The importance of the speed-of-light can be appreciated nearly ubiquitously: in Figure 3 of (CHRISTEN; SCHENK; CUI, 2012)[2], in Figure 4 of (KAMIL *et al.*, 2010), in Figure 4 of (DATTA *et al.*, 2008) whose sub-figure 4b explicitly shows that the per-core performance decreases with increasing cores. The same holds for Figure 4 of (DURSUN *et al.*, 2009), where wall-clock time increases with increasing number of processors. Interestingly, there are two cases in Figure 5 of the same work where superlinear speed-ups are achieved. Dursun et al. say it may result from increasing aggregate cache size, whereas this author suggests it might point out to possible opportunities of speeding-up their original or baseline code. In any case, these superlinear cases seem to be the ones standing out, pointing to a general validity of the speed-of-light approach. As a final example, Figure 3 of (TANG *et al.*, 2011) brings results from ten pairs of single core and 12-core performance with Pochoir and from ten pairs of

[2]For example, from "Lapl GSRB" of their Figure 3a, $P_2 = 1.37 \times P_1 < 2 \times P_1$ and $P_{16} = 13.33 \times P_1 < 16 \times P_{16}$.

baseline codes. For the ten pairs of baseline codes and for nine pairs of Pochoir codes, $P_{12} < 12 \times P_1$. Once again, the one case in which the superlinear scalability is attained (12.3 instead of the expected maximum of 12) is the outlier. As commented earlier, it might point out to possibilities of enhancing the single-threaded code. For example, is not the threading infrastructure hurting the single core performance, although such infrastructure is pointless to optimize a single-threaded experiment? This author suggests to take the speed-of light as the best compute rate attained in a variety of domain sizes, with and without the extra overhead of the technique under analysis. It is also noteworthy in Figure 3 of (TANG *et al.*, 2011) that the scalability in the experiments with Pochoir is higher than the scalability in the experiments without Pochoir. The same will be observed in the experimental chapters of this doctoral work, where ASLI increases both the speed-of-light and the scalability.

In the 2D case, kernel convolution increases the FLOPs to **equivalent stencil** ratio (Section 5.3). This should mean that no kernel convolution version would perform better than an approach that just iterates naively through the time domain, provided all latency and register dependency are hidden and provided a perfect use of SIMD in the naive implementation. However, this work shows that kernel convolution does pay off for 2D and 3D operators as well, what is patent from the fact that it has higher **speed-of-light** than non-convolved stencils. The next sections help to better understand the technique, and show how to compare it with naive, or non-kernel convolved, stencil computations.

5.2 Operational Intensity and the Memory-Bound Roofline — μ

Another quantity sometimes employed to characterize stencil computations is the operational intensity, used to set an expected limit to the performance. This limit results from the interaction of the main memory with the computational power (compu-

ting units) of the system when both are trying to execute the designated stencil computation. Nowadays, in stencil codes such interaction is commonly unbalanced due to the memory not keeping up with the computing units, a situation in which the performance limit becomes *memory-bound*.

A set of cores can compute stencil points at a rate just as high as the memory subsystem can provide input points and store output points, in a producer-consumer relationship. Suppose a memory bandwidth of a total B Gigabytes per second for read and write, meaning that in a given second the aggregate amount of bytes written and bytes read is limited to B Gigabytes. Then suppose that the inputs to each application of the update equation are allocated close enough in the memory, residing in the same cache-line, or even contiguous cache-lines, a case where the data pre-fetchers usually work well. Additionally, suppose that the outputs reside close to each other, but far from the inputs. In this case, on average each application of the update equation entangles the traffic of two points: one read and the other written. If each point has size s bytes, then the memory can handle the generation of just as much as $\frac{B}{2s}$ Gigapoints per second. Finally, if the update equation requires f FLOPs to calculate each output stencil point, in other words, if f denotes the *necessary FLOPs*, then the maximum achievable rate of floating point operations per second that the set of cores could sustain is given by Equation 5.1.

$$\mu = \frac{fB}{2s} \tag{5.1}$$

A similar way to obtain Equation 5.1 is to regard the quantity $f/(2s)$ as a *code balance* or an *operational intensity* per by byte transitioned with the main memory. A setback of the formula is that it mingles the read bandwidth B_r with the write bandwidth B_w, where $B = B_r + B_w$, thus not being precise in case $B_r \neq B_w$, but rather an approximation.

In Equation 5.1, regarding optimization, B is a given circumstance of the environment (machine), s reflects an implementation decision about precision (4 bytes for single precision, 8 bytes for double), and f has been so far treated as an immutable

aspect of the desired stencil computation. If the architecture employs write-allocate cache policy, the denominator should be adapted to 3s.

In the state-of-the-art, space-tiling is paramount for the stencil kernel to reach the memory-bound roofline performance. Time-tiling is a very interesting approach that builds on top of space-tiling to help stencil-kernels to even surpass this rate. Section 2.4 comments more about this technique while introducing the approach of the 1D experiments. Both can be boosted if applied together with Kernel Convolution.

5.3 Equivalent Metrics — EGFLOPS, EGS, EFLOCC

The two most common dimensional units used to analyse the performance of stencil implementations and optimization techniques are *GFLOPS*, standing for giga-floating point operations per second, and *GS*, standing for giga-stencil point updates per second, occasionally also referred to as *GLUPS*, giga-lattice updates per second. How to convert among these units is specific to the operator, and care should be taken. Accordingly, given a stencil pattern or operator, the conversion among these units is given by the following equation:

$$P_{gf} = f \times P_{gs}, \tag{5.2}$$

where P_{gf} is expressed in GFLOPS, P_{gs} in GS, and f are the necessary FLOPs, as in Definition 5.

But extra care should be taken when comparing stencil codes that underwent kernel convolution. The above cited metrics count how many times a stencil output is computed, disregarding whether the counted stencil point is the final output or an intermediate output which is a mere sub-product of the chosen implementation. Indeed, GFLOPS is intimately related to how many applications of the *update equation* of the original, non-convolved stencil occur during execution.

The update equation of a convolved stencil is larger than that of the original stencil. Not just that, but it is applied fewer times as well. When it comes to the original operator, the update equation is employed for T iterations, whereas a kernel of convolution degree Δ executes only T/Δ iterations of an arithmetically equivalent equation to obtain the same final product. These factors hinder a direct comparison of simple and kernel convolved operators if the comparison relies on terms such as GFLOPS or GS. Instead, the following *equivalent metrics* should be used to compare kernel convolved stencils among themselves and with the simple form:

Definition 6 (EGFLOPS, EGS)**.** *Equivalent GFLOPS and Equivalent GStencils per Second (EGFLOPS and EGS) take into account that a convolved form is designed to yield mathematically the same output as the simple form.* Both perform the same semantic work, but through distinct realizations in the machine. *The calculation is simple:* $Work_{simple}/T_{convolved}$*, where* $Work_{simple}$ *is the amount of FLOPs or Stencil Updates a simple implementation would perform for the same problem.*

Definition 7 (FLOCC, EFLOCC)**.** *Floating Point Operations per Core per Cycle (FLOCC) and its* equivalent-form *(EFLOCC) indicate how a computation rate, usually expressed in (E)GLOPS, relates to the per-core capability of the machine. It can be obtained by dividing the performance of the code by the product of the machine's clock times the number of cores used in the computation.*

The metrics FLOCC and EFLOCC offer a more palpable view of how the computational units are being used and what goes on in a cycle by cycle perspective. As kernel convolution allows for better exploitation of such units, this kind of fine-grained concern becomes more insightful than in a scenario where just simple stencil forms are used, guiding the kernel engineering process. EFLOCC is also a convenient way to compare the code quality when the number of used cores varies.

Let us assume that T time-steps must be performed in a 1D domain of N elements with a Reduction in Iteration Space of 2 (RIS=2). The related ASLI kernel has to go

through the domain $NT/\text{RIS} = NT/2$ times and performs 7 FLOPs at each point, as $x_{i,t+2} = e_0 \times x_{i,t} + e_1 \times (x_{i-1,t} + x_{i+1,t}) + e_2 \times (x_{i-2,t} + x_{i+2,t})$, where the e_a's are the convolved stencil coefficients. The total FLOPs through time is now $7NT/2 = 3.5NT$. A straightforward implementation, which does not apply ASLI or kernel convolution, requires 4 FLOPs per point, or $4NT$ in total. Therefore, there is a possible $4NT/3.5NT = 1.14$[3] speed-up over the best possible straightforward implementation. This means that if both kernels achieve peak floating-point performance ASLI is faster, as it has fewer computation to perform.

In this regard, the notions of **Equivalent Stencils (ES)** and **Equivalent FLOPs per Second (EFLOPS or EFlop/s)** allow to compare the performance of kernel convolved and straightforward implementations. For example, if both kernels are given the same problem input (domain and amount of iterations), a convolved kernel issues fewer intermediate stencil outputs in total but both compute the same amount of ES, as they solve the same problem. Thus, if both kernels have the same time to solution, they perform the same ES per second and EFLOPS, but ASLI might have performed fewer (in the 1D case) or more (2D, 3D) actual FLOPs. From Definition 6, it is clear that the ratio between equivalent metrics corresponds to the speed-up, or the inverse of the ratio between time to solution.

5.4 Behavioral Aspects of ASLI

For kernel-convolved schemes, more data points are needed per iteration. As ASLI always restructures the innermost loop, when designing ASLI kernels it is recommended to traverse the fastest varying dimension applying computation reuse to foster in-register data reuse. For example, 1D ASLI kernels should have just *one explicit load*, that of the rightmost point.

[3] The theoretical limit of this speed-up for large enough values of RIS of the 1D, 3-point, symmetric stencil is 1.33, and 1.25 if asymmetric.

ASLI can change the computation characteristics of stencil in three areas: (a) the amount of *equivalent stencils* per iteration, (b) the FLOPs to bytes ratio, and (c) the FLOPs to *equivalent stencils* ratio. By design, ALSI increases (a). For 1D, 2D, and 3D, ALSI always increases (b). For 1D, it decreases (c). However, for 2D and 3D star-shaped stencils, ALSI actually increases (c). In a machine with perfect memory subsystem, where not just main memory latency can be hidden but also the latency associated with the various levels of memory cache, or in the case all latency and register dependency is hidden, ASLI would not be beneficial for 2D and 3D. Literature shows that this scenario is far from being true.

5.5 On the Interaction with the Memory Subsystem

To characterize Aggregate Stencil-Loop Iteration in best case scenarios, it is profitable to distinguish between two scenarios with respect to interaction with the memory subsystem:

Scenario 1 — the whole domain fits L1 cache.

Scenario 2 — the domain is much larger than the last level cache.

Additionally, to exercise how different approaches influence different performance aspects, let us consider two kernels implemented naively. Listings 5.1 and 5.2 convey in an assembly-like language two straighforward kernels implemented naively, which do not apply neither kernel convolution nor any other main optimization. Even in this case one can discern a simple optimization, by performing fewer *explicit loads*. These kernels help one to begin discerning what tradeoffs are at stake in a variety of optimization techniques. Note in Listing 5.1 that the Naivest Type of Kernel contais “three explicit loads”, whereas the Naive-Smart Type of Kernel (5.2) has just one explicit

load, in line 7.

Listing 5.1: Naivest Type of Kernel (NTK).

```
1 For each timestep:
2   For each x[i] in Domain:
3     // innermost loop
4     // get data - three explicit loads
5     load x[i-1]
6     load x[i]
7     load x[i+1]
8
9     // compute y[i] = a*(x[i]) + b*(x[i-1] + x[i+1])
10    temp1 = a*(x[i])
11    temp2 = x[i-1] + x[i+1]
12    temp2 = b*temp2
13    temp1 = temp1 + temp2
14
15    // store the new point
16    store temp1 at y[i]
17  next x[i]
18  swap(x, y)
19 next timestep
```

Listing 5.2: Naive-Smart Type of Kernel (NSK).

```
1 For each timestep:
2   For each x[i] in Domain:
3     // innermost loop
4     // get data - one explicit load
5     left_x = middle_x
6     middle_x = right_x
7     right_x = load x[i+1]
8
9     // compute y[i] = a*(x[i]) + b*(x[i-1] + x[i+1])
10    temp1 = a*(middle_x)
11    temp2 = left_x + right_x
12    temp2 = b*temp2
13    temp1 = temp1 + temp2
14
15    // store the new point
16    store temp1 at y[i]
17  next x[i]
18  swap(x, y)
19 next timestep
```

5.5.1 Scenario 1: Whole Domain Fits L1 Cache

Still in this case ASLI is usually advantageous, as it enhances instruction level parallelism and also helps to hide latency of first-level cache access.

5.5.1.1 Naivest and Naive-Smart Types of Kernel

To achieve the best performance possible with the *Naivest Type of Kernel (NTK)*, the stencil program should minimize the impact of three main forces while traversing the domain:

- back to back register dependencies,
- the latency of the three L1 accesses, and
- the book keeping operations that control the loops.

With hand-optimization and loop unrolling, it is possible to significantly reduce latency and register dependency in the *Naivest Type of Kernel*, in part because in this scenario the domain fits the L1 cache, which provides relatively low latency. If we now compare lines 5 through 7 of NTK with lines 5 trough 7 of the *Naive-Smart Type of Kernel (NSK)*, we see that in NSK two load instructions were changed in favor of two register moves. But NSK still has no time-blocking, let alone kernel convolution. In fact, its innermost loop has the same amount of instructions as that of the Naivest-Type of Kernel, for which it was argued it is possible to develop a version that significantly hides latency and register dependency. The same can be done for the Naive-Smart Kernel, in which case both kernels would perform similarly. But in this author's experience the Naive-Smart Kernel still has two practical advantages, at least with off-the-shelf compilers: (1) it is more likely that compilers will generate a machine code for NSK better than for NTK, and (2) with loop unrolling and variable remapping, many instances of lines 5 and 6 can be suppressed in NSK. Therefore, it is actually more feasible to

obtain a faster program for the Naive-Smart Type of Kernel than for the Naivest Type of Kernel.

5.5.1.2 Time-Blocking

The general reasoning carried out in Section 5.5.1.1 applies here as well. It can be added that the time-blocking version would not be any better. Both a time-blocking version with *three explicit loads* or a version with *one explicit load* would perform similar to the NTK or NSK counterpart. Time-blocking helps to bring the data from the main memory closer to the register files less often, but in Scenario 1 everything fits the cache. Therefore, as time-blocking does not change the dynamics of the innermost loop, be it in the case of one explicit load or three explicit loads, no real advantage exists here. A side advantage in relation to plain NTK or NSK could come when multi-threading is employed, as in this case the time-blocking implementation could apply concurrent start. It is important to note that, although in the case of multi-threading the time-blocking program could be faster than multi-threaded NTK or NSK, the maximum achievable performance per-thread would not be better than a single-threaded NTK or NSK, or for that matter time-blocking kernel. In other words, time-blocking scales better, but does not enhance what is being scaled, namely, it does not help much with the speed-of-light performance (Section 5.1). A practical disadvantage is that time-blocking requires many more copies of the domain points if applied blindly or naively, so potentially it is no longer true that everything fits the cache. In Scenario 2 this technique will be very beneficial. Just like the naive NTK and NSK kernels, time-blocking kernels execute the innermost loop $N \times T$ times, each time performing 4 FLOPS, for a total of 4NT FLOPs.

5.5.1.3 ASLI

For kernel-convolved schemes, more data points are needed per iteration. Because ASLI always restructures the innermost loop, when designing ASLI kernels it is recommended , while traversing the fastest varying dimension, to apply the reuse of values previously computed or loaded to the register files. That is, 1D ASLI kernels should have just *one explicit load*, of the rightmost point, thus being more similar to the Naive-Smart Type of Kernel than to the Naivest Type of Kernel.

Recall from Section 5.3 the 1D example where the kernel-convolved and non-kernel-convolved codes have $3.5NT$ and $4NT$ total FLOPs, respectively. Because the problem fits L1 cache, it is often argued in the literature that the tradeoff between memory access and computation is negligible. Accordingly, Section 5.3 argued that if both a naive and a kernel-convolved code achieve peak floating-point performance, what is more likely to happen when the domain fits the memory cache, then the kernel-convolved code could potentially be $4NT/3.5NT = 1.14$ times faster. Of course, this assumes cache latency, register dependencies, and other undesired inefficiencies can be hidden. The exploitation of such speed-up might be constrained by potential imbalance between add and multiply instructions, e.g., in the case the architecture provides fused multiply-add or two separate units for addition and multiplication (further discussed in Section 5.7.2). This phenomenon gives rise to two concepts explored under Section 5.7 and empirically evaluated in experimental chapters, namely, **intrinsic speed-up**, which can then be used as a benchmark, and **hyper performance**, of which there are two types, **soft and hard** (Section 5.7.3.1).

If kernel convolution reduces the amount of computation for 1D stencils, thus helping to increase in-cache performance, does it mean that it reduces the performance of in-cache 2D and 3D stencils, where it increases the amount of computation? It is hardly so, as the experiment section shows. This happens because the performance of 2D and 3D stencils is commonly far from achieving peak floating point performance,

and kernel convolution helps to reduce (i) the impact of back to back register dependency, as more independent sub-expressions are created, and to reduce (ii) the *impact of accessing not only the main memory, but, what is more central to Scenario 1, the memory cache as well*. This seems to tie well with the observation in (MALAS *et al.*, 2015) that the intra-cache data transfer and the core performance are the bottleneck for good enough time-blocking techniques, such as the ones that work analyzes.

5.5.2 Scenario 2: Domain is Much Larger than the Last Level Cache

ASLI was originally designed with this scenario in mind, where the latency of accessing main memory becomes a significant bottleneck to performance. The already present in the literature concepts of operational intensity and compulsory bytes (Section 5.2) exchanged between main memory and cache offer valuable insight to understand how different computer architectures limit the performance of stencil and kernel convolution. For a more fluent exposition, these quantities will be revisited while NTK and NSK are discussed below.

5.5.2.1 Naivest and Naive-Smart Types of Kernel

To make a complete sweep or traversal through the domain of N points, every point has to travel from the main memory to the caches (and then from the caches to the processor, but this concerns Section 5.5.1). Because in Scenario 2 the domain is much larger than the largest cache level, by the completion of a whole traversal through the domain points the first points needed in the next traversal necessarily no longer reside in cache, so that they need to travel from the main memory to the caches in order to ultimately be reused. When counting the **compulsory bytes** exchanged between the main memory and the caches, it does not matter whether a point went to the caches because it was explicitly loaded, in the sense of Listings 5.1 and 5.2, or because it belongs to the same cache line of a previously requested point and still resides in cache

by the time the point is necessary and explicitly loaded.

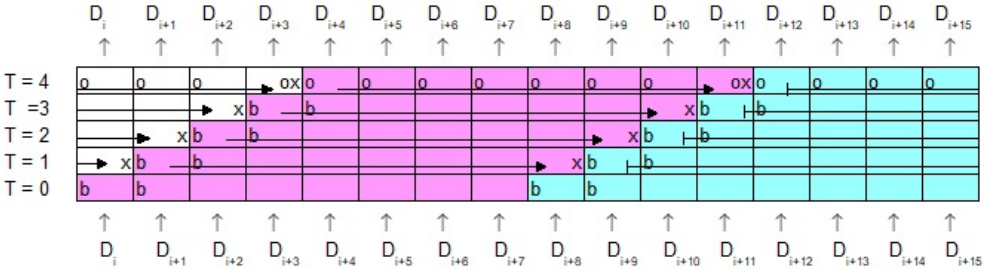
Also, because the stencil kernels are updating the value of the domain points (e.g., line 16 of Listings 5.1 and 5.2), it also means that each of the N points will be written back to the main memory once per sweep through the domain. Thus, if *double precision* is assumed, per sweep the code necessarily generates the read traffic of at least $8N$ bytes and the write traffic of $8N$ bytes. Therefore, the so-called **compulsory bytes** are $16N$ per sweep or $16NT$ total, after every time iteration is performed. For the naive implementations a total of $4NT$ FLOPs are executed. The **operational intensity** is then defined as the ratio between the FLOPs and the compulsory bytes, or $4NT/16NT = 0.25$ in this case. In write-allocate architectures there are extra 8 bytes generated per double precision domain point. Nonetheless, it is possible to eliminate this extra traffic with special instructions (write-through).

5.5.2.2 Time-Blocking

Time-blocking acts to increase the amount of work performed on the data found in the cache(s). This approach can shift the code from being memory bound to computation bound. However, on page 29 of (DATTA *et al.*, 2009) the authors note that "another surprising result of the study is the lack of correlation between main memory traffic and wallclock run time", an observation which coalesces with the already discussed observation in (MALAS *et al.*, 2015). This is specially applicable to cache-oblivious approaches. The lack of correlation also ties to the discussion of Section 5.5.1, where it was argued that time-blocking is bound to perform close to the naivest and naive-smart types of kernel. Figure 12 helps to analyze the time-blocking approach:

Figure 12 depicts three space-time blocks (white, pink, and blue) of a time-bocking approach that calculates 4 time steps by each time-blocking iteration. With multithreading, different threads would be used for different blocks. The row identified

Figure 12: Example of the dynamics of a 1D stencil code implemented with time-blocking.



Source: Author.

with $T = 0$ is the input for the first time step of the depicted time-blocking iteration. The top row is both the result of the 4^{th} time step of the current time-blocking iteration, and the input to the first time step of the next time-blocking iteration.

Cells marked with *x* lie in the boundary of a space-time block. As such, two of the values necessary to calculate each *x* cell reside in an adjacent space-time block, in the underneath row. These necessary values are represented in the boundary cells marked with *b*. For this reason, the value of these domain points are more likely to end up consuming main memory traffic. Cells marked with *o* are input to the next iteration of the time-blocking algorithm, and need to be stored in the main memory. In practice, the whole cache line of the cells marked with *o* and *b* will interact with the main-memory, and threads writing in neighboring *o* cells will compete for resource. All this activity consumes memory bandwidth and generates latency.

5.5.2.3 ASLI

As commented in Section 5.4, for 2D and 3D stencils ASLI increases the total amount of computation. But the main benefits from kernel convolved stencils actually come from reducing the *operational intensity*, that is, the ratio of FLOPs to bytes exchanged with the main memory. Notwithstanding, experimental results show that even the 3D ASLI, which could exhibit the most impact from extra computation added by kernel convolution, has significant speed-up compared to the literature. ASLI

works to cut the total main memory traffic by a factor of RIS, being most beneficial in applications that are heavily memory-bound. This fact allows us to draw two related conjectures, the practical proof of which remain as future work:

Conjecture 1 — It follows that for *strided access* and for *languages with more overhead* like Python and Java kernel convolution might pay off more naturally and the speed-up achievable through kernel convolution could be more likely to approach RIS. In such scenarios more unnecessary bytes are brought from the main memory to cache, as the stride transfers unnecessary data in the same cache-line as necessary data.

Conjecture 2 — In such scenarios, it is also more likely that the convolution of more complex calculations pays off more naturally.

Conjecture 1 has two parts to it, concerning inefficiencies from memory access with stride pattern and from slower languages. Regarding strided access, what it does memory-wise is to increase the value of s in Equation 5.1, effectively decreasing the theoretical peak of memory-bound codes. This in turn leaves more space for ASLI to be effective. There is, however, a limit to the negative effects of this access pattern, or perhaps multiple limiting plateaus. In the case of multiple limits, perhaps higher values of s would steadily reduce performance, until one of the limits in reached. From there one, higher values of s would not induce performance reduction for a while, until another threshold is reached and performance starts to degrade again. Perhaps these limits would be related to the sizes of cache lines of the various cache levels. Another practical limitation of having strided access is that this would usually imply that the SIMD units are not used with its full capacity. Even though there are SIMD instructions specifically designed to gather data from a stride pattern, perhaps they are not so widely employed. This would mean a reduction to the peak compute-bound performance. Finally, note that if the SIMD units are still used in its full width, then ASLI has room to be relatively more beneficial, as in this case the peak compute-bound performance is not reduced, whereas the memory-bound performance is.

Table 2: Characteristics of some ASLI schemes for 1D, 2D, and 3D completely symmetric stencils.

									Machine 1		Machine 2	
					Peak DP (GFlops/s):				75		74	
					Bandwidth (GB/s):				5.9		16.6	
D	r	RIS	FLOPs	Naive FLOPs	FLOPs Ratio	Cache Reads	Arith. Intensity	Opera. Intensity	Roofline	Max Speed-up	Roofline	Max Speed-up
1	1	1	4	4	1	3	0.17	0.25	1.48	1	4.15	1
		1	4	4	1	1	0.5	0.25	1.48	1	4.15	1
		2	7	8	1.14	1	0.88	0.44	2.58	2	7.26	2
		3	10	12	1.2	1	1.25	0.63	3.69	3	10.38	3
		20	61	80	1.31	1	7.63	3.81	22.49	20	63.29	20
	2	1	7	7	1	5	0.18	0.44	2.58	1	7.26	1
		2	13	14	1.08	1	1.63	0.81	4.79	2	13.49	2
		3	19	21	1.11	1	2.38	1.19	7.01	3	19.71	3
		12	74	84	1.15	1	9.13	4.56	26.92	12	74	11.72
2	1	1	6	6	1	5	0.15	0.38	2.21	1	6.23	1
		2	15	12	0.8	5	0.38	0.94	5.53	2	15.56	2
		3	26	18	0.69	7	0.46	1.63	9.59	3	26.98	3
3	1	1	8	8	1	7	0.14	0.5	2.95	1	8.3	1
		2	24	16	0.67	13	0.23	1.5	8.85	2	24.9	2

Regarding slower languages, some key factors to stencil performance seem to be the representation of the domain points on the memory, the calculation of the indexes, and the retrieval of the point values. Usually the representation of the domain points would imply stride access, thus effectively increasing s of Equation 5.1 and lowering the memory-bound limit. The calculation of the indexes might become a complication to such languages, as it might be the case that the related compiler is not as advanced as common of-the-shelf C compilers in this regard, and the programmer definitely has less power to induce the use of more effective machine instructions for data addressing. This could be detrimental to Conjectures 1 and 2 in case RIS is elevated enough or the underlying simple operator is complicated enough, because of the multitude of indexes necessary. Finally, once the indexes of the domain points are determined, the language has to translate it into a memory index that points to the location in memory where the value of the point resides, and then retrieve it. This is also the kind of operation where higher level languages would include some logical checks, adding overhead. This access is the equivalent of the *explicit load* in the sense of this Section 5.5.

5.5.3 Characterization for 1D

Naive and ASLI versions of stencil kernels are represented in each row of Table 2. The first three columns define the kernel configuration, with respect to dimension,

radius, and reduction in iteration space or convolution degree. Speed-ups achievable in two hypothetical machines are shown. The operational intensities assume a compulsory main memory traffic of 16B generated by each double precision (DB) calculated. The highlighted row shows a kernel version that would be compute bound. Any row with RIS=1 represents a naive, straightforward kernel, and therefore the corresponding kernel has FLOPs Ratio = 1. The information to calculate the FLOPs Ratio is found in the 4^{th} and 5^{th} columns. These columns show that the 3D ASLI of RIS=2 performs 24 FLOPs to obtain the same result that a straightforward implementation would get with 16 FLOPs, resulting in $\textit{FLOPs Ratio} = 16/24 \approx 0.67$.

The 7^{th} column tells that the kernel of the 1^{st} row has 3 cache reads (or *explicit loads*). This row represents the most straightforward kernel, which is NTK, the Naivest Type of Kernel. The 2^{nd} row features 1 *explicit load*. This row represents a straightforward kernel that, although not through ASLI, applies the principle of *Computation Wavefront Reuse (CWR)* or a primitive variation thereof. This is the *Naive-Smart Type of Kernel (NSK)*. Such kernel brings just one new data value from the caches to the register files per iteration of the innermost loop. Recall that in the 1D case such reuse is that of individual points. The kernel with wavefront reuse (2^{nd} row) has higher *arithmetic intensity*[4] than the most straightforward (1^{st} row), as it diminishes the interaction between processors and caches. Both schemes have the same operational intensity, as both traverse the whole domain the same amount of time, with the same update equation. When analyzing domains larger than the last cache level, operational intensity and related compulsory bytes exchanged with the main memory should be considered.

We are now ready to compare ASLI vs. **Naive**: One iteration of ASLI with RIS=2 performs a work equivalent to two iterations of a naive implementation. In doing so, ASLI needs 7 FLOPs, whereas a naive implementation requires $2 \times 4 = 8$ FLOPs. Therefore, if both kernels achieve peak performance, ASLI is still faster, as it has fewer

[4] Arithmetic intensity in this work denotes the FLOPs per data loaded from caches into register, differing from operational intensity (Section 5.2)

computation to do. The 6th column, *FLOPs Ratio*, reflects this information. An ASLI of RIS $= 3$ requires 10 FLOPs to compute one iteration. In doing so, ASLI computes *three equivalent stencils*. A naive implementation would require $3 \times 4 = 12$ FLOPs. The information necessary to calculate the *FLOPs Ratio* of the kernels is found in the 4th and 5th columns, respectively labeled *FLOPs* and *Naive FLOPs*. Before exploring the next columns, let us compare the time-blocking and ASLI versions.

Time-blocking kernels (not singled-out in the table) in the state-of-the-art have the same *arithmetic intensity* of their **NK** or **NSK** counterpart. Their relevance stems from changing the operational intensity, so much so that they may shift the program from being memory-bound to compute-bound. Implicit in this statement is the fact that memory-boundedness usually refers solely to main memory, overlookig the effects of the other memory levels. Such kernels would not have the extra speed-up that could come from reducing the total amount of FLOPs, so that their *FLOPs Ratio* (6th column) would be 1. In practice such approaches will be parametrized to fit the last level cache and, because the corresponding latency is still much higher than L1 latency, a blind application of this technique will incur many cycles wasted with L2 stalls, specially for 2D and 3D cases. This is obviously the case for ASLI as well, but note that ASLI works in the direction of cutting such wasted cycles by a factor of RIS. Such practical vicissitudes are usually overlooked when discussing "best case scenarios". To conclude, ASLI can be time-blocked.

The last four columns exercise the concepts and trade-offs. These columns represent a limit to achievable performance on two types of hypothetical machines, specified by the *double precision peak performance (DP)* and the *bandwidth (BW)*. The two columns labeled Roofline show for each machine the minimum between $B \times$ OI, which is the product of bandwidth by the operational intensity, and DP, the peak performance in double precision. This minimum represents the peak performance achievable by the related kernel in best case scenarios. The columns labeled Max Speed-up show the

product of *FLOPs Ratio* by the ratio *Roofline(kernel) / Roofline(straightforward)*. Before the breaking of the memory-bounded zone, the free lunch a kernel would get with the FLOPs Ratio is actually irrelevant, and the combined speed-up comes solely from the degree of the kernel convolution (i.e., RIS). This phenomenon is better appreciated when analyzing the last row of 1D kernels in the table (gray row), as in this case the kernel would be able to achieve peak DP in Machine 2, but not in Machine 1.

5.5.4 Characterization for 2D and 3D

For the 2D and 3D cases, ASLI results in FLOPs Ratio < 1. This means that the 2D, radius 1, RIS = 2 kernel, which has FLOPS Ratio of 0.8, would never perform better than a given straightforward implementation that reaches $12/15 = 80\%$ of peak performance. However, in practice 2D stencils are far from achieving 80% of peak performance. In the memory-bounded zone, the FLOPs Ratio is irrelevant for the theoretically achievable speed-up, and a kernel would benefit from ASLI. There are some well-known approaches to tackle the low operational intensity of stencil codes. Time-blocking is one interesting approach and especially useful for 1D stencils. For 2D and 3D cases such approaches are usually parametrized to fit the last level cache. Because of the latency disparity among cache levels, a blind application of time-blocking still incurs many wasted cycles with cache stalls. As commented earlier, this could happen with a careless implementation of ASLI as well, but ASLI by itself works to reduce the amount of such wasted cycles, in addition to being suitable to be applied with time-blocking

5.6 On Rounding Errors

It was said that, by construction, ASLI yields a result mathematically equivalent to the state-of-the-art implementations. But the finite representation of the intermediate results of both methods can make their final result diverge, so that we face the question

of asserting which method is more precise, and by how much.

First, let us introduce the notion of **incorporated FLOPs**: it is the total amount of FLOPs performed to issue a given *floating point* number. Let us work with a 2D asymmetrical case, as for this case counting the FLOPs is simpler. In a naive implementation, each point results from 9 FLOPs ($1Mul + 4 \times (1Mul + 1Add)$). On its turn, each of the five input float points of the current calculation, at iteration T, was the result of other 9 FLOPs, so that the current output is the result of $9 + 5 \times 9 = 44$ FLOPs, with respect to the domain at timestep $T - 2$. For the 2D ASLI with RIS=2, recall that the *input constellation* (Figure 13) has 13 points. Also, the output of the current iteration, with respect to the domain at timestep $T - 2$, requires 25 FLOPs ($1Mul + 12 \times (1Mul + 1Add)$).

Mathematically put, let the immediate FLOPs $ImF_{x \leftarrow a,b}$ designate the number of FLOPs required to yield output x having as input a and b. Let the incorporated FLOPs ($IncF_x$) designate the number of FLOPs involved in having output x, accounted for through an interval of interest. Then:

$IncF_x = ImF_{x \leftarrow a,b} + IncF_a + IncF_b$, or, more generally,

$IncF_x = ImF_{x \leftarrow I} + \sum_{i \in I} IncF_i$, where I is the set of inputs to calculate x.

Figure 13: Example of *input constellation*.

		5		
	2	6	10	
1	3	7	11	13
	4	8	12	
		9		

Source: Author.

At first glance, more incorporated FLOPs would propagate more error, what favors ASLI as a more precise approach. However, this assumes Gaussian error distribution, which might not be the case in errors arising from truncation or computational rounding in the scenario of stencil computation. Another possible root of systematic error might come from the process of calculating the convolved coefficients. The silver li-

ning comes from the fact that convolved coefficients come from original coefficients, which come from mathematical models and, therefore, might have much better precision than the original domain points, which usually represent measures from the physical world. Additionally, depending on the application and hardware, the coefficients could be stored with higher precision than the domain points. This could offset any truncation error generated when deriving the convolved coefficients. Finally, note that from the first iteration onwards, the input points to the update equations of the stencil are no longer completely independent.

The discussion is not complete and the definite conclusion remains to be taken. Ultimately, it can be settled by running both versions on an input representative of a real world scenario but for the result of which a known formula exists. Some approximate computing library might be used to simulate reduced precision, thus enabling errors to be detected early.

Something that might weigh against the accuracy of ASLI, albeit negligibly, is the fact that coefficients smaller than one would imply smaller and smaller convolved coefficients. These would lead to numbers of much different magnitudes being added and subtracted to each other, thus inducing rounding errors. This could be a systematic non-Gaussian source of rounding errors. This is a possible topic of future work, with the above reasoning perhaps working as guidelines or ideas for future researchers.

5.7 Determining New Performance Boundaries for Stencils

Kernel Convolution transforms a stencil pattern into another stencil pattern with higher arithmetic intensity. The resulting code increases the register pressure and, if the code is implemented with necessary care, say, with a simple time-blocking approach, then the in-register interactions can indeed become a bottleneck. Additionally, as the new pattern is more compute intense, there is an extra reason to analyze the interaction

of stencils with the floating point units. This section investigates the involved trade-offs, which could be used by compilers as part of cost models for Kernel Convolution.

5.7.1 FLOPs Ratio – ϕ

The formerly alluded to notion of FLOPs Ratio is formalized and further exemplified here. To begin the investigation, let us consider a 1D-3point symmetric stencil, with update equation $X_i^{t+1} = c_0 X_i^t + c_1(X_{i-1}^t + X_{i+1}^t)$, where c_0 and c_1 are constant coefficients. This operator requires 4 FLOPs (floating point operations) per updated domain point. Thus, through time, i.e., after the T iterations, the total computation of the stencil problem $S = \langle O, T, N, D \rangle$ requires $4NT$ FLOPs. Let us now consider a kernel convolved approach to compute S. Without loss of generality, it is possible to create a kernel that uses a convolution degree of 2 and use it to calculate the same result (in machine precision) of P by, instead, computing $S' = \langle O^2, T/2, N, D \rangle$. In this case, the update equation has the form $X_i^{t+2} = e_0 X_i^t + e_1(X_{i-1}^t + X_{i+1}^t) + e_2(X_{i-2}^t + X_{i+2}^t)$, with $e_{0,1,2}$ constant. This requires 7 FLOPs per updated domain point when computing P', or a total of $7N\frac{T}{2} = 3.5NT$ FLOPs through time. The FLOPs Ratio can now be defined:

Definition 8 (FLOPs Ratio (ϕ))**.** *The FLOPs Ratio ϕ_{RIS} of O^{RIS}, with respect to the original operator O^1, is the ratio between the FLOPs necessary to compute $S = \langle O, T, N, D \rangle$ and the FLOPs necessary to compute $S' = \langle O^{\text{RIS}}, T/\text{RIS}, N, D \rangle$.*

Therefore, if the necessary FLOPs (Definition 5) in the update equations of the original and convolved operators are respectively f_1 and f_Δ, the FLOPs ratio can be calculated with Equation 5.3, where RIS is represented by Δ.

$$\phi_\Delta = \Delta f_1 / f_\Delta \tag{5.3}$$

The FLOPs Ratio of 1D stencils is greater than 1, meaning that a convolved operator executes fewer FLOPs than the original, simple form. The savings of FLOPs result

from computation saved in the central element. With O^1, the central element entangles RIS multiplications every RIS steps. With O^{RIS}, these multiplications merge into just one multiplication of the central element. The radial elements offer no savings. The FLOPs Ratio also reflects the speed-up that would be experienced when solving a problem with kernel convolution, if both the original and convolved computations performed equally, i.e., with same floating point operations per second (FLOPS). In other words, if both versions performed at 100% peak floating point performance, the kernel convolved computation would still be faster. This *opens up the opportunity of having over 100% equivalent performance*, a theoretical result further discussed in Section 5.7.3.1.

5.7.1.1 Generalization

A symmetric stencil of radius r has update equation of the form $X_i^{t+1} = c_0 X_i^t + \sum_{j=1}^{r} c_j (X_{i-j}^t + X_{i+j}^t)$, requiring $1+3r$ flops per updated point, totalling $(1+3r)NT$ through time. If the problem were to be solved by applying a kernel convolution of degree RIS, the update equation would be $X_i^{t+\text{RIS}} = e_0 X_i^t + \sum_{j=1}^{r\text{RIS}} e_j (X_{i-j}^t + X_{i+j}^t)$, for a total of $(1+3r\text{RIS})N\frac{T}{\text{RIS}}$ FLOPs through time. For asymmetric stencils, the respective quantities are $(1+4r)NT$ and $(1+4r\text{RIS})N\frac{T}{\text{RIS}}$. With the information above, one can express the FLOPs ratio ϕ_{RIS} as in Equation 5.4.

$$\phi_{\text{RIS}} = \begin{cases} \dfrac{\text{RIS}(1+3r)}{1+3r\text{RIS}} & \text{if symmetric} \\ \dfrac{\text{RIS}(1+4r)}{1+4r\text{RIS}} & \text{if asymmetric} \end{cases} \tag{5.4}$$

5.7.2 Exploitation of Fused Multiply-Add Units – η

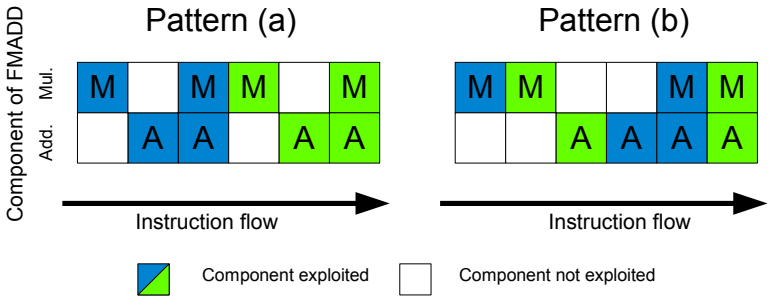
It is a well know fact in the literature that the mix of instructions, in this case specially of multiplications and additions, can be detrimental when it comes to achieving a machine's peak floating point performance. But stencils are far from achieving

peak performance, being usually heavily main memory-bound or, in the case of some time-bocking approaches, as it seems, cache-level memory-bound. Therefore possible imbalances between multiplications and additions have been left aside in the literature. With the advent of ASLI, a few reasons to be more attentive to the mix of adds and muls emerge: kernel convolution changes the instruction mix, the code gets closer to peak performance, and the mix might impact whether it is beneficial or not to pursue further convolution.

Although the 1D-3point symmetric stencil has perfect balance of multiplies and additions, as it requires two additions and two multiplications to update each point, its computation through the simple form (to the best of the author's knowledge, the only form found in the literature before ASLI) can never achieve peak floating point performance on fused multiply-add architectures. This is a remarkable fact. Figure 14 helps to understand this physical limitation. It shows two instruction scheduling possibilities for the 2×4 FLOPs necessary to update two domain points, each represented in different colors, with a 1D-3point stencil. For both patterns, just 8 FLOPs are performed (colored cells), out of 12 possible (white and colored). Cf. Figure 15, for possibilities for O^2. Figure 14 represents that in this stencil both the multiplication $c_0 X_i^t$ and the summation $(X_{i-1}^t + X_{i+1}^t)$ can use just one component of a fused multiply-add instruction, therefore missing half of its compute power. *As a consequence, even if the FPUs are kept completely busy (100% utilization), this stencil can achieve at most* $8/12 = 66.7\%$ *of the nominal peak floating point performance of the machine*.

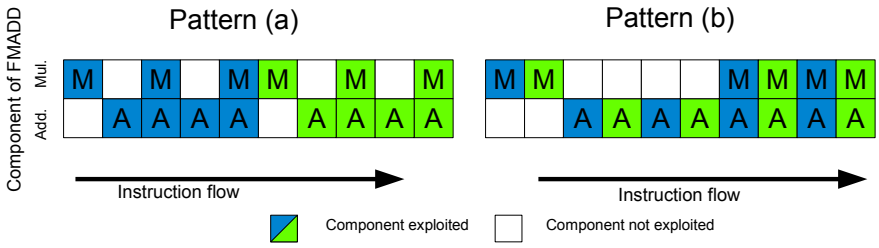
Definition 9 (Exploitation Factor (η))**.** *The exploitation factor* η *of a given computation pattern in a given machine is the fraction of total FMADD components (ADDs and MULs) used out of the amount that could have been used if every FP instruction used both components to perform a necessary FMADD. Alternatively, it is the fraction of the peak floating point operations that could be reached if the computation executed at 100% FPU utilization.*

Figure 14: Two instruction scheduling possibilities necessary to update two domain points.



Source: Author.

Figure 15: Two instruction scheduling possibilities to update two points when convolution degree 2 is applied to the 1D-3point stencil.



Source: Author.

Table 3: Breakdown of FMADD components used in simple and convolved forms of symmetric and asymmetric stencils of radius r.

	Original			Convolved		
	A	M	MA	A	M	MA
Symmetric	r	1	r	rRIS	1	rRIS
Asymmetric	0	1	2r	0	1	2rRIS

An implementation with kernel convolution of degree 2 of an originally 1D-3point stencil results in scheduling patterns equivalent to the ones depicted in Figure 15. It shows two instruction scheduling possibilities of the computation to update two points when a convolution degree of 2 is applied to the originally 1D-3point stencil depicted in Figure 14. For both patterns in this figure, 14 FLOPs are performed out of 20 possible, yielding an exploitation factor $\eta = 0.7$ (Cf. the simple form O^1 in Figure 14). In this figure one can appreciate that, also in the convolved kernel, even if each FP instruction completes every cycle (meaning 100% utilization), there would still be FMADD components not used (white blocks in Figure 15). For this kernel convolved pattern with RIS $= 2$, the exploitation factor is $\eta_2 = 7/10 > \eta_1 = 2/3$.

Figures 14 and 15 also make it clear how to calculate the exploitation factor η of a computation in terms of the amounts of additions (A), multiplies (M) and fused multiply-adds effectively used: $\eta = \frac{\text{A+M+2MA}}{2(\text{A+M+MA})}$. With the aid of the information summarized in Table 3, we see that Equation 5.5 expresses the exploitation factor η_{RIS} of different convolved kernels. In Equation 5.5, r is the radius of the original operator O^1, where an operator of radius r has $1+2r$ points. An interesting property is that η_{RIS} monotonically increases with RIS and with the original radius r of O^1. The limit is 75% for symmetric and 100% for asymmetric stencils.

$$\eta_{\text{RIS}} = \begin{cases} \dfrac{1+3r\text{RIS}}{2+4r\text{RIS}} & \text{if symmetric} \\ \dfrac{1+4r\text{RIS}}{2+4r\text{RIS}} & \text{if asymmetric} \end{cases} \tag{5.5}$$

5.7.3 Inherent Gain – ξ

With a better understanding and knowledge on how to calculate the maximum exploitation of simple and convolved forms, and having seen how to calculate the FLOPs Ratio between said forms, it is possible to appreciate that the maximum speed-up that a convolved form can get over the *best theoretical performance achievable* by an original stencil is given by the FLOPs Ratio times the exploitation of the convolved form, divided by the exploitation of the original stencil, as defined in Equation 5.6. This equation does not take memory-boundedness into account. The expansion of the terms in this definition yields Equation 5.7, which states that the kernel convolution of both symmetric and asymmetric 1D original stencil codes have the same *speed-up between maximums*, χ.

$$\chi_{\text{RIS}} \triangleq \phi_{\text{RIS}} \frac{\eta_{\text{RIS}}}{\eta_1} \tag{5.6}$$

$$\chi_{\text{RIS},sym} = \chi_{\text{RIS},asym} = \frac{1+2r}{\frac{1}{\text{RIS}}+2r} \tag{5.7}$$

As commented before, 1D simple stencils are far from approaching η_1 performance, which is its physically maximum achievable performance, so that the practical observation of Equation 5.6 is not possible. Notwithstanding, if the speed-up $\mathscr{S}_{O^1 \to O^{\text{RIS}}}$ observed with convolution is higher than χ_{RIS}, it means that the kernel convolution had the benefit of helping with the interactions with the memory subsystem and/or with register dependencies. If $1 < \mathscr{S} < \chi_{\text{RIS}}$ then in the convolved kernel the resulting effect from the interaction of the code with the memory subsystem and register dependencies is relatively worse than in the simple stencil, although the kernel transformation still resulted in a desirable speed-up.

Definition 10. *[Inherent Gain (ξ)] The Inherent Gain $\xi_{A \to B}$ between two kernel versions O^A and O^B with convolution A and B is given by $\xi_{A \to B} \triangleq \frac{\chi_B}{\chi_A}$.*

The concept of inherent gain is a convenient means to compare kernels with different degrees of convolution. Consider two kernel versions A and B with convolution RIS_A and RIS_B , respectively. In this scenario, a speed-up higher than $\xi_{A\rightarrow B}$ means that the convolution also brought relative gains in the interaction with the memory subsystem. One sees that the idea behind inherent gain is similar to that of *speed-up between maximums*. However, when engineering kernels one is not concerned just with the speed-up a convolved version could yield over the naive original stencil version, but also with what gains the newest version would offer over an intermediate or previous one. Therefore, having the distinct symbols χ and ξ is mandatory.

In practice, with increasing convolution we expect an observed relative gain to be smaller when going from $O^{\text{RIS}-1}$ to O^{RIS}. That is, we expect $\frac{\mathscr{S}_{(A-1)\rightarrow A}}{\xi_{(A-1)\rightarrow A}} > \frac{\mathscr{S}_{(B-1)\rightarrow B}}{\xi_{(B-1)\rightarrow B}}$ if $B > A$. This expectation reflects a belief that in grounds of smaller convolution it is more straightforward to diminish the negative effects in the memory interaction, which are overwhelming.

To conclude this section Table 4 summarizes the aspects ϕ, η, χ, and ξ of the 22 unidimensional kernels employed in the 1D experiments. A value RIS=1 identifies the base operator (O^1) of the category. This table helps to see that, fixed the stencil size (radius), higher RIS implies higher χ and ϕ, and, fixed RIS, larger sizes imply smaller χ and ϕ.

5.7.3.1 On the Viability of Hyper-Performance

It was commented in Section 5.2 that time-blocking allows a kernel to overcome the memory-bound roofline. However, even such kernels are limited by the exploitation factor η (Section 5.7.2). Therefore, without any kernel convolution, a symmetric 1D 3-point stencil can achieve at most $\eta_{O^1} M$ GFLOPS in a machine with fused multiply-add instruction set and peak floating point performance of M Giga-FLOPs per second. This can be accomplished only with 100% utilization of the computing units. But ideally

Table 4: Characteristics of some kernel convolved stencil codes.

Geometry:		Symmetric		Asymmetric		Both	
r	RIS	ϕ	η	ϕ	η	χ	$\xi_{O^{\text{RIS}-1}\rightarrow O^{\text{RIS}}}$
1	1	1.000	0.667	1.000	0.833	1.000	NA
	2	1.143	0.700	1.111	0.900	1.200	1.200
	3	1.200	0.714	1.154	0.929	1.286	1.071
	4	1.231	0.722	1.176	0.944	1.333	1.037
	5	1.250	0.727	1.190	0.955	1.364	1.023
	6	1.263	0.731	1.200	0.962	1.385	1.015
2	1	1.000	0.700	1.000	0.900	1.000	NA
	2	1.077	0.722	1.059	0.944	1.111	1.111
	3	1.105	0.731	1.080	0.962	1.154	1.038
3	1	1.000	0.714	1.000	0.929	1.000	NA
	2	1.053	0.731	1.040	0.962	1.077	1.077

a convolved form would still be able to achieve a speed-up of $\chi_{O^{\text{RIS}}}$ over such a peak. Let U be the observed utilization of the FP unit(s)[5] achieved by a convolved kernel. Thus, if $U > 1/\chi_{\text{RIS}}$, then the convolved kernel is performing better than the maximum performance the simple form could achieve with 100% utilization. Interestingly, if $U > 1/(\phi_{\text{RIS}}\eta_{\text{RIS}})$, then the convolved kernel is performing better than a simple kernel would achieve even if it performed at maximum peak floating point performance (M, in the example). This work purports to name these situations as *weak* and *strong hyper-performance*, respectively. Although weak hyper-performance is not achieved in this work, the author believes it is a feasible outcome, especially for 1D stencils of radius 1. Theoretically, in fused multiply-add architectures strong hyper-performance is achievable only by asymmetric stencils with convolution degree higher than 2.

5.7.3.2 Inherent Gain as a Benchmark

The notion of inherent gain ξ can be used to benchmark computational systems as a whole or in parts. Compilers, core architecture, main memory and cache memory, instruction scheduling, all play a whole in determining whether the seed-up that the next level of convolution will achieve is smaller or higher than the inherent gain

[5] A way to obtain U is through hardware counters for the amount of floating point instructions executed and total cycles spent.

between the current and the next level. Therefore, the transformation of kernel convolution and its inherent gain can be used to benchmark different systems or machines, showing how the whole systems or machines leverage their nominal capabilities, or it can be used to benchmark different components of a system or machine in a similar fashion.

5.8 Applicability and Scope

ASLI was initially designed to circumvent the increasing gap between memory and computing performance, with the former tending to lag behind. Take for example Figure 16, which characterizes four versions of an originally 2D 5-point stencil. The figure shows how the very same GPU (NVidia GTX 1070) has a wide variation in compute performance between double precision and single precision, whereas the bandwidth capacity does not vary between both cases. Therefore, in double precision, a convolution degree of 2 is already enough to turn the kernel compute-bound. It is important to note that here it is not possible to assess whether the resulting code is actually more memory-bound or compute-bound, because these limits are somewhat close (128.6 vs. 121.6 EGFLOPS). With convolution 3 or 4 it is clearer that probably the compute rate would be the limiting factor, so optimizing such kernels should start with optimizing instruction level parallelism, register reuse etc. Figure 16 also shows that here single performance is disproportionally higher than double performance (32 times higher, instead of the usual ratio of 2 times found in CPUs). This opens space for the effectiveness of higher degrees of kernel convolution in 2D and 3D, not to mention the 1D case where the total work is also reduced. With such a disparity in single precision, it becomes also easier to hide latency in back to back dependencies, index calculation, or even reuse in the PWC and CWR pipelines. Therefore, it is safe to say that at least in some GPUs the gains brought by ASLI can be relatively smaller in double precision or much higher in single precision.

Figure 16: Example of performance limits on a GPU.

GTX 1070					Double Precision		Single Precision	
					Perform.:	202 GFLOPS	Perform.:	6618.11 GFLOPS
					B.W.:	256.3 GB/s	B.W.:	256.3 GB/s
	Δ (RIS)	f_Δ	φ_Δ	η_Δ	Mem-Bound	Comp-Bound	Mem-Bound	Comp-Bound
2D	1	6	1	0.6	64.1	121.2	128.2	3970.9
	2	14	0.86	0.7	128.6	121.6	257.2	3984.1
	3	24	0.75	0.71	192.2	107.6	384.5	3524.1
	4	37	0.65	0.71	256.8	93.2	513.7	3054.3

Source: Author.

As with the GPU case, the same should happen with many-core architectures and processors with larger register files or faster computing units. Larger register files would be useful to store intermediate computation, which is more frequent in convolved kernels than in original operators. A similar phenomenon happens with memory architectures with larger or more numerous cache levels. Because often the two copies of the domain can be much larger than even the last level cache, let alone the first levels, optimization for cache reuse with techniques such as time-blocking is still closely dependent on the memory-bound limit, even if time-blocking allows to surpass said limit. This is so because with domain sizes being orders of magnitude larger than the cache files it is guaranteed that within a single domain sweep the contents of the cache will be recycled at least tens to thousands times. Larger cache-levels, however, are much valuable to the larger convolved kernels, as they also help with intermediate computation in case it does not fit into the register files. ASLI also increases the amount of equivalent computation performed with the cached values.

The present work focuses on stencils of constant coefficients. This author's personal implementations of kernel-convolved versions of an explicit way to compute Black-Scholes, based on (ODEGAARD, 2003), have however shown that kernel convolution also generates speed-ups for smaller original 1D stencils of variable coefficients. This is an interesting result, but as of today it is not clear whether speed-ups could be achieved even for 2D stencils of variable coefficients. It is also not clear to what degree of convolution 2D and 3D convolved kernels would cease to be beneficial,

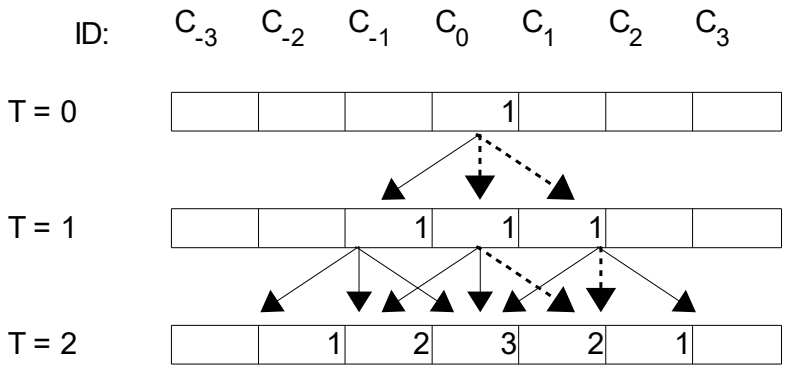
or if any degree of convolution is beneficial for 4D stencils. For disparities between memory and compute power such as the one in single precision found in Figure 16, higher degrees of convolution tend to pay off more naturally. To assess precisely which stencils, with what convolution, in which machine, would pay off is still a matter of practical experimentation. Thus, the importance of the discussion in Chapter 6, which is a step toward generating templates for multiple original stencils. The templates will be useful in the future when one conducts experiments with each geometry. Once one identifies that a template is beneficial to one's shape, then the specificities of one's problem can be implemented. This saves time in more domain-specific researches.

6 OTHER DEVELOPMENTS RELATED TO THE INFLUENCE TABLE, BUTANTAN NUMBERS

The Influence Table and the problem of deriving the convolved coefficients lead to interesting mathematical outcomes.

Consider a symmetric one dimensional stencil pattern, with three coefficients all of which are 1. Figure 17 shows Influence Tables for this stencil, up to $T = 2$. Note that the Influence Table when $T = 1$ coincides with the original stencil pattern, as is always the case for symmetric operators. The table helps to visualize how the present value of a point or cell will propagate and alter the value of itself and other cells in the future.

Figure 17: Influence Table of 1D 3-point stencil with every coefficient being 1.



Source: Author.

6.1 Definitions and Notation

Some concepts and notions useful for the reasoning of this chapter are defined below. The cells numbered from -3 to 3 in Figure 17 represent consecutive domain points.

1. Influence — is the measure of how the **unitary** value of a point or cell at an earlier time alters the value of a cell in a later time. In the figure, the value of C_0 at time $T = 0$ influences the value of C_1 at time $T = 2$ by a factor of 2, or $C_{1,T=2} = 2 \times C_{0,T=0}$.

2. Auto-influence — is the measure of how the unitary value of a point or cell alters its own value in a later time. Unless otherwise noted, auto-influence refers to immediately succeeding time steps. In the case of the operator in Figure 18, auto-influence always happens via a factor of a. In Figure 17 it happens via a factor of 1. The symbol $\circlearrowleft$ will represent it.

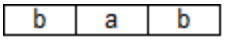
3. Lateral or side influence — is the measure of how the unitary value of a point or cell alters the value of an immediate neighbor. Unless otherwise noted, it refers to immediately succeeding time steps. In the case of the operator in Figure 18, lateral influence always happens via a factor of b. The directional symbols $\rightarrow, \leftarrow$ will represent it (also $\uparrow$ and $\downarrow$ in bi-dimensional cases).

4. Reciprocal Influence Principe (RIP) — is the notion that, if the operator has symmetric coefficients, the influence of the value of a current domain point into the future value of another domain point has the same (unitary) value of the current value of this other domain point into the future value of that point. It is intimately related to the concept of Influence Tables.

5. Influence Path — is a possible trajectory, understood in terms of succeeding auto-influences and side influences, of how the value of a point can alter the

value of another point. Represented with $\{\}$ encompassing an ordered sequence of $\rightarrow, \leftarrow, \circlearrowleft$ elements. The *weight* of an influence path is defined as the product of the factors of its constituent influences.

Figure 18: Example of symmetric 1D 3-point stencil.



Source: Author.

Figure 17 highlights with dotted lines the two *influence paths* from $C_{0,T=0}$ to $C_{1,T=2}$. The paths are an auto-influence of $C_{0,T=0}$ on $C_{0,T=1}$ followed by a side influence of $C_{0,T=1}$ on $C_{1,T=2}$, and a side influence of $C_{0,T=0}$ on $C_{1,T=1}$ followed by an auto-influence of $C_{1,T=1}$ on $C_{1,T=2}$. These influence paths can be respectively represented as $\{\circlearrowleft\rightarrow\}$ and $\{\rightarrow\circlearrowleft\}$, each having weight 1×1. In the case of the operator of Figure 18 both would weigh *ab*.

6.2 Trinomial Numbers

Now, it has been just commented that the cell $C_{1,T=2}$ in Figure 17 has value 2, which is the exact number of influence paths that end in $C_{1,T=2}$. Such equivalence happens when every coefficient of the original operator is 1, as discussed below. Therefore, the fact that $C_{0,T=2} = 3$ means that there are 3 influence paths from $C_{0,T=0}$ to $C_{0,T=2}$, to wit: $\{\circlearrowleft\circlearrowleft\}$, $\{\leftarrow\rightarrow\}$, and $\{\rightarrow\leftarrow\}$. This phenomenon holds true for 2D and 3D operators as well.

Additionally, when the original stencil operator is 1D, 3-point and every coefficient is 1, then the Influence Table at time t corresponds to the t-th row of the *trinomial triangle*, and the convolved coefficients correspond to the trinomial coefficients of said row. Whereas in the binomial triangle (the Pascal's or Tartaglia's triangle) each element of row t is obtained by adding two consecutive elements of row $t-1$, the trinomial triangle adds three consecutive elements of row $t-1$. The correspondence between the

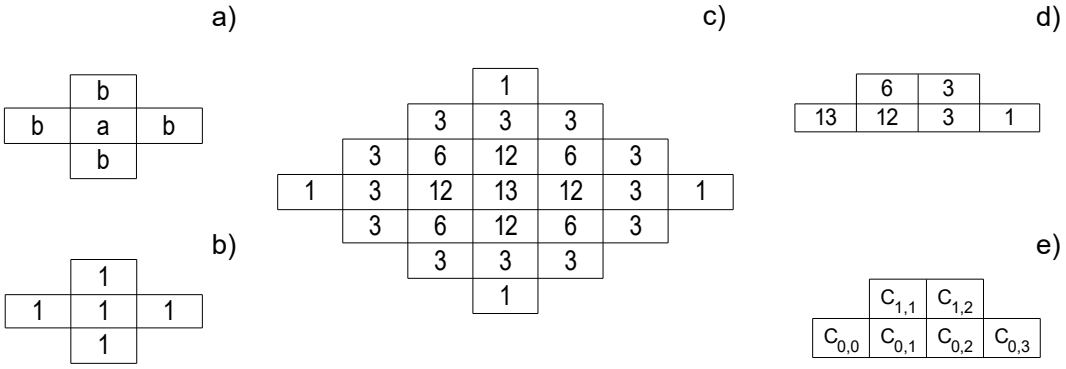
Influence Table and the trinomial triangle can be explored to validate the value calculated for the convolved coefficients, even when the original coefficients are not 1. Section 6.4 discusses this possibility, but first the next section practices the heretofore defined concepts.

6.3 Coefficients for 2D 5-point, RIS = 3

This section shows how to determine the convolved coefficients of the 2D 5-point stencil with literal coefficients in Figure 19a. But first consider the operator with unitary coefficients in Figure 19b and the corresponding Influence Table of RIS $= 3$ in Figure 19c. As pointed out in Section 6.2, this table highlights the unitary influence of a central initial element on its neighbors, and the value of each cell equals the amount of influence paths from the central cell to it, after 3 time steps.

Given the symmetry of the obtained influence table, it is possible to focus on the elements of an octant, or compact influence table, represented in Figure 19d. Figure 19e offers a nomenclature mapping.

Figure 19: Influence Table with unitary coefficients and nomenclature map.



Source: Author.

The value 1 of cell $C_{0,3}$ of Figure 19d indicates that, after 3 time steps, there is only one influence path from cell $C_{0,0}$ to impact $C_{0,3}$. The path is $\{\rightarrow\rightarrow\rightarrow\}$, with three lateral influences. In case the original operator had the form of Figure 19a, each lateral

influence would propagate with a factor b, rather than 1. In this case, the influence path would have developed as follows: first, with the influence of $C_{0,0}$ with factor b on cell $C_{0,1}$, followed by an influence with factor b on $C_{0,2}$, followed by the final influence, again through factor b, on $C_{0,3}$. Therefore, the influence of the value of $C_{0,0}$ at time $T = 0$ on $C_{0,3}$ at time $T = 3$ is proportional to b^3, implying that the convolved coefficient $e_{0,3}$ of O^3 equals b^3. Compare this result with the original coefficient d and the convolved d^3 in Figure 10, and -0.2 and -0.008 in Figure 11.

A similar reasoning allows to deduce that there are three influence paths to $C_{1,2}$, as $C_{1,2} = 3$. The possible influence paths are $\{\rightarrow\rightarrow\uparrow\}$, $\{\rightarrow\uparrow\rightarrow\}$, and $\{\uparrow\rightarrow\rightarrow\}$. Incidentally, it corresponds to the amount of ways to traverse a grid $h = 1$ cell upwards and $w = 2$ cells to the right, which equals $\binom{h+w}{h} = \binom{3}{2} = 3$. (The same applies whenever $h + w =$ RIS, e.g., $C_{0,3} = \binom{3}{0}$.) Each of the three influence paths contains three side influences, so that, in case the original operator was Figure 19a, then O^3 would have $e_{1,2} = 3b^3$. This represents a total of 3 influence paths, each weighing b^3.

Any influence path that accounts for the influence of $C_{0,0}$ at $T = 0$ on the value of $C_{0,2}$ at $T = 3$ should contain at least two side influences to the right. Additionally, such path should contain three influences, so that there is room for only one extra undefined influence. This cannot be a side influence, otherwise the influence path would not end in $C_{0,2}$. Therefore, this has to be an auto-influence. This auto-influence can be arranged in three ways: before both side influences, between the side influences, or after both of them. These combinations result in three paths ending in $C_{0,2}$. They are: $\{\circlearrowleft\rightarrow\rightarrow\}$, $\{\rightarrow\circlearrowleft\rightarrow\}$, and $\{\rightarrow\rightarrow\circlearrowleft\}$. In the case of the operator in Figure 19a, an auto-influence would propagate with factor a, so that O^3 would have $e_{0,2} = 3ab^2$.

Any influence path that accounts for the influence of $C_{0,0}$ at $T = 0$ on the value of $C_{1,1}$ at $T = 3$ should contain at least two lateral influences (one to the right and one upwards). The third influence should thus be an auto-influence. There are $3! = 6$ ways to order the three distinct influences $\rightarrow$, $\uparrow$, and $\circlearrowleft$. There are thus 6 influence paths

(as could have been deduced from the very fact that $C_{1,1} = 6$ in Figure 19c). Because $\rightarrow$ and $\uparrow$ propagate with factor b and $\circlearrowleft$ with factor a, in the case of Figure 19a the corresponding O^3 would have $e_{1,1} = 6ab^2$.

For $C_{0,1}$, one influence to the right is necessary. The other two influences make the influence paths take three forms:

- With extra two auto-influences, collaborating with $\binom{3}{2} = 3$ influence paths with factor a^2b each.
- With extra two opposite lateral influences,
  - being one upwards and one downwards. This contributes with $3! = 6$ influence paths with factor b^3 each. (There are six distinct ways to order $\rightarrow$, $\uparrow$, $\downarrow$); and
  - being one to the right and the other to the left. In this case the set of influences ($\rightarrow$, $\leftarrow$, $\rightarrow$) can be ordered in $\binom{3}{2} = 3$ distinct influence paths, each with factor b^3.

Therefore, in the case of Figure 19a, the corresponding O^3 would have $e_{0,1} = 3a^2b + 6b^3 + 3b^3 = 3a^2b + 9b^3$. The sum of the coefficients of this polynomial is $12 = C_{0,1}$.

Any influence path of $C_{0,0}$ at $T = 0$ on $C_{0,0}$ at $T = 3$ can happen in two ways. Either via 3 auto-influences, or via 1 necessary auto-influence and two opposite lateral influences. Opposite lateral influences can be ordered in four ways ($\{\leftarrow\rightarrow\}, \{\rightarrow\leftarrow\}, \{\uparrow\downarrow\}$, and $\{\downarrow\uparrow\}$). In all of these four scenarios the necessary auto-influence can occur in three ways: before both lateral influences, between them, or after both, so there are a total of $4 \times 3 = 12$ such movements. Therefore, $e_{0,0} = a^3 + 12ab^2$, and again the polynomial coefficients add up to $13 = C_{0,0}$. Finally, the six distinct convolved coefficients of the convolved operator with RIS $= 3$ of the original operator represented in Figure 19a are:

$$e_{1,1} = 6ab^2 \qquad e_{1,2} = 3b^3$$

$$e_{0,2} = 3ab^2 \qquad e_{0,3} = 1b^3$$

$$e_{0,0} = 1a^3 + 12ab^2 \qquad e_{0,1} = 3a^2b + 9b^3$$

6.4 Stating the Problem

The current and next few sections focus on coefficients of unidimensional stencils with varying degrees of convolution. For the sake of exposition, the notation $e_{M,N}$ will denote the convolution degree and the index of the unidimensional coefficient, rather than the coordinates of a bi-dimensional operator. Each coefficient $e_{\Delta,i}$ of the convolved stencil O^{Δ} can be obtained by the evaluation of a characteristic polynomial of degree Δ, the variables of which are the coefficients of the original stencil, and the polynomial coefficients of which are discussed in this section.

If the original stencil has $J+1$ coefficients $c_0, \ldots, c_J$, and P denotes a polynomial with K terms, then we can write $e_{\Delta,i} = P_{\Delta,i}(c_0, c_1, \ldots, c_J) = \sum_{k=1}^{K_{\Delta,i}} L_{\Delta,i,k} c_0^{g_{\Delta,i,k,0}} c_1^{g_{\Delta,i,k,1}} \ldots c_J^{g_{\Delta,i,k,J}}$, where L stands for a *polynomial coefficient*, and the g's are exponents of the original *stencil coefficients* in the expression of the convolved coefficients, under the restriction that $\sum_{j=0}^{J} g_{\Delta,i,k,j} = \Delta$, because $P_{\Delta,i}$ has degree Δ. The previous expression for each convolved coefficient can be simplified to Equation 6.1 if one keeps in mind that each has its own set of parameters K, L, and g.

$$e = \sum_{k=1}^{K} L_k \prod_{j=0}^{J} c_j^{g_{k,j}} \tag{6.1}$$

In case the original coefficients are all 1's, $P_{\Delta,i}$ is independent from the exponents $g_{\Delta,i,k,j}$, so that $e_{\Delta,i} = \sum_{k=1}^{K_{\Delta,i}} L_{\Delta,i,k}$. At the same time, as stated above, for a symmetric

3-point stencil with $c_0 = c_1 = 1$, then $e_{\Delta,i}$ corresponds to the i-th element of the Δ-th row of the trinomial triangle. Furthermore, each *polynomial coefficient L* corresponds to the number of influence paths of same weight from $X_{0,T=0}$ to $X_{i,T=\Delta}$.

The question that arises is whether it is possible to derive a closed formula for each $L_{\Delta,i,k}$, what would mean finding a closed formula for $e_{\Delta,i}$. When $i = 0$, the sequence $e_{1,0}, e_{2,0}, \ldots$ describes the central elements of the trinomial triangle, beginning at row 1. In this case, each sequence $L_{\Delta,0,0}, L_{\Delta,0,1}, \ldots, L_{\Delta,0,K_{\Delta,0}}$ represents a way to partition each central element of row Δ of the triangle. An integer sequence[1] is already known which describes such a way to partition the central elements of the trinomial triangle, and thus helps to describe $P_{\Delta,0}$ and $e_{\Delta,0}$. Nonetheless, it does not seem to exist a registered sequence, nor does a closed formula seem to be known that describes $P_{\Delta,i}$ when $i \neq 0$.

6.5 Deriving Expressions for Symmetric 1D 3-Point

Which are the terms and *polynomial coefficients* that describe the convolved coefficient $e_{\Delta,i}$ of the convolved operator O^{Δ}? It has been established that each polynomial coefficient corresponds to the number of influence paths of same weight that starts in C_0 and ends in C_i after Δ time steps, and that each polynomial term is the multiplication of said number by the weight of the influence path. Therefore, a way to answer the opening question of this section is by counting paths.

Given the stencil in question, influences can happen with factor c_0 if they are auto-influences ($\circlearrowleft$), or with factor c_1 if they are side influences ($\leftarrow$ or $\rightarrow$). Because $e_{\Delta,i}$ is associated with ending up in cell C_i after Δ movements, it is clear that any influence path necessarily has a minimum of i side moves and side influences. The remaining $\Delta - i$ influences must consist of a combination of auto-influences and pairs of $\leftarrow$ and

[1] Sequence registered as A105868 in OEIS, the On-Lline Encyclopedia of Integer Sequences (https://oeis.org/).

Table 5: The $K_{\Delta,i}$ terms and polynomial coefficients of $e_{\Delta,i}$. ($0 \leq r \leq K_{\Delta,i} - 1$, and $0 \leq 2r \leq \Delta - i$.)

R-influences	L-influences	Auto-influences	Combinations	Weight
i	0	$\Delta - i$	$\binom{\Delta}{i}\binom{\Delta-i}{0}$	$c_0^{\Delta-i}c_1^{i}$
$i+1$	1	$\Delta - i - 2$	$\binom{\Delta}{i+1}\binom{\Delta-i-1}{1}$	$c_0^{\Delta-i-2}c_1^{i+2}$
$i+2$	2	$\Delta - i - 4$	$\binom{\Delta}{i+2}\binom{\Delta-i-2}{2}$	$c_0^{\Delta-i-4}c_1^{i+4}$
$i+r$	r	$\Delta - i - 2r$	$\binom{\Delta}{i+r}\binom{\Delta-i-r}{r}$	$c_0^{\Delta-i-2r}c_1^{i+2r}$

$\rightarrow$ influences. Therefore each term can be written as a function of the number of right influences, as illustrated in Table 5.

Table 5 conveys that any influence path that departs from C_0 and after Δ movements ends in C_i and that contains $i+r$ right movements (or influences), necessarily has r counterbalancing left movements (or influences), all of which propagate with factor c_1. This leaves space for $\Delta - i - 2r$ auto-influences of factor c_0. Such paths with $i+r$ right movements thus weigh $c_0^{\Delta-i-2r}c_1^{i+2r}$. Determining the amount of such paths is a two-step process: (1) First distribute the $i+r$ right influences among Δ movements, for a total of $\binom{\Delta}{i+r}$ combinations. (2) Then, for each combination of step (1), distribute the r left influences among the remaining $\Delta - i - r$ movements, for a total of $\binom{\Delta-i-r}{r}$ combinations. The remaining $\Delta - i - 2r$ should necessarily be auto-influences. Finally, the total amount of such paths implies that $L_{\Delta,i,r} = \binom{\Delta}{i+r}\binom{\Delta-i-r}{r}$, with $0 \leq r \leq K_{\Delta,i} - 1$ and $0 \leq 2r \leq \Delta - i$. [2]

6.6 Visualization and Connection with Pascal's Triangle and Trinomial Numbers

Section 6.2 pointed out that the influence tables of the 3-point stencil with unitary coefficients describe the trinomial triangle, whereas Section 6.4 pointed out that even when the stencil coefficients are not 1's, still the sum of the polynomial coefficients of

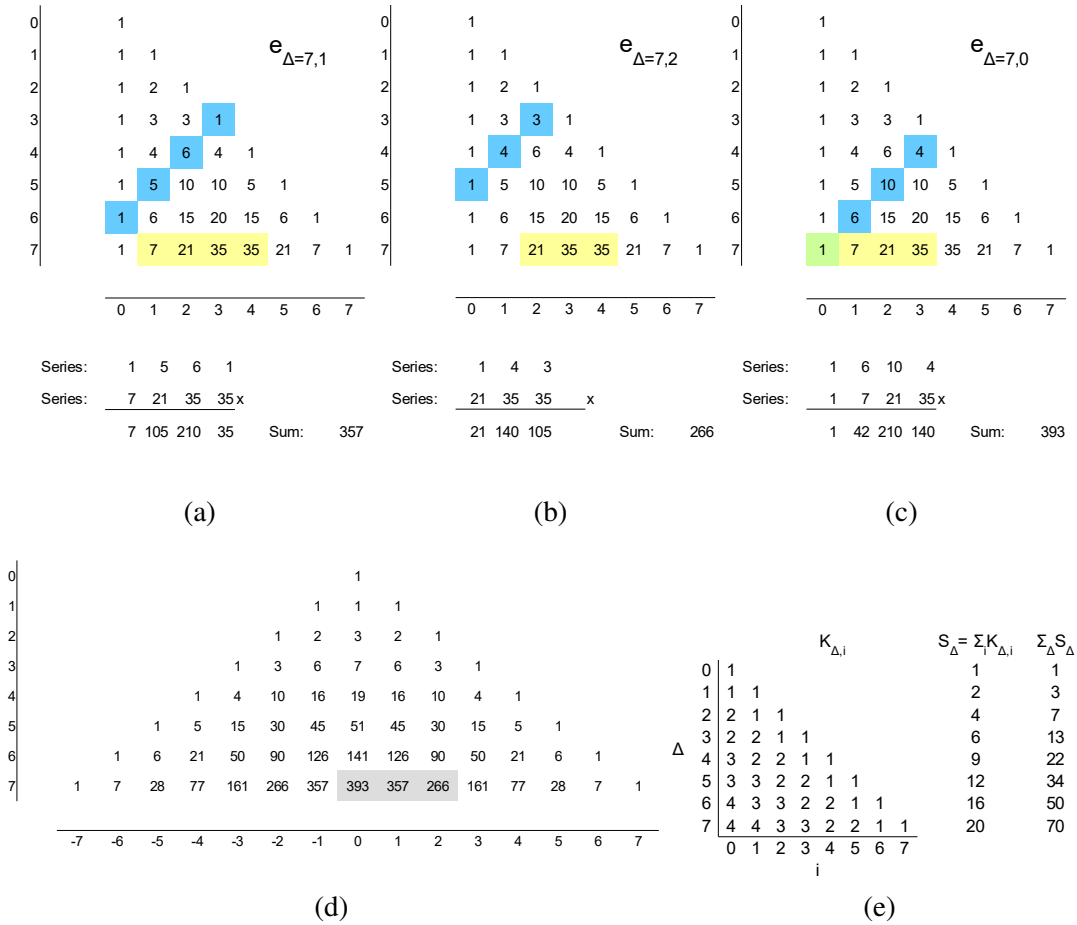
[2] By isolating the formula to work only with the central elements, where $i = 0$, its terms can be written as $T(\Delta, r) = \binom{\Delta}{r}\binom{\Delta-r}{r} = \binom{\Delta}{\Delta-r}\binom{\Delta-r}{r}$. By further manipulating the variables so that $N = \Delta$ and $K' = \Delta - r$, then $r = N - K'$ and $T = \binom{N}{K'}\binom{K'}{N-K'}$, which coincides with sequence A105868 in OEIS, applicable only to the central elements.

each expression of a convolved coefficient describes a number in the trinomial triangle. Closed formulas for such expressions were described in Section 6.5. Now a method to traverse the binomial triangle will be shown, which represents said formulas, and therefore represents a link between the binomial and trinomial triangles:

1. Consider a representation of the binomial numbers as a left-aligned Pascal's triangle, with $\binom{n+1}{0}$ under $\binom{n}{0}$.
2. Two sequences of size $K_{\Delta,i}$ will be selected from this arrangement.
3. The multiplication of the k-th elements of these sequences will yield $L_{\Delta,i,k}$.
4. The addition of the $K_{\Delta,i}$ elements of Step 3 produces the i-th element of the Δ-th row of the trinomial triangle.
5. The first sequence starts at $\binom{\Delta}{i}$ and runs to the right. The second element, if it exists, is $\binom{\Delta}{i+1}$. The last element is $\binom{\Delta}{i+K_{\Delta,i}-1}$.
6. Regarding the second sequence:
   6.1. From the first element of the first sequence (Item 5.), traverse the triangle diagonally up and to the left, until the element $\binom{\Delta-i}{0} = 1$ on the vertical leg is reached. This is the first element of the second series.
   6.2. This sequence runs diagonally up and to the right, until no other element in the triangle can be found by traversing along this direction. It therefore determines the value of $K_{\Delta,i} = 1 + \left\lfloor \frac{\Delta-i}{2} \right\rfloor$.

Figure 20 exemplifies the execution of the steps above to determine the polynomial coefficients of e_1, e_2, and e_0 of a convolved stencil of degree 7. Each subfigure (a), (b), and (c) highlights a way to partition one highlighted element of subfigure (d) into a sequence of size expressed in subfigure (d). The first and second series of the steps above are respectively represented by yellow and blue numbers in each subfigure, or green

Figure 20: Traversal of binomial numbers to generate convolved coefficients and trinomial numbers.



Source: Author.

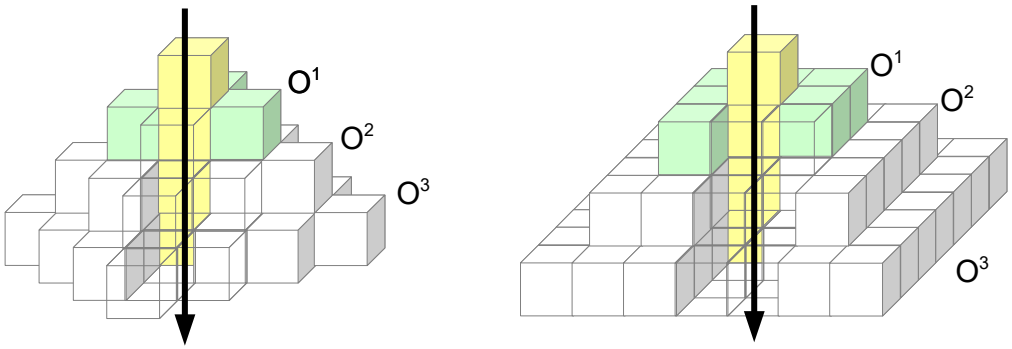
when the element takes part in both series. Multiplying the k-th terms of each sequence by one another yields a parameter $L_{\Delta,i,k}$ to evaluate one convolved stencil coefficient, and the summation of these parameters results in a number of the seventh row of the trinomial triangle (Figure 20d). For example, from Figure 20a, that exemplifies $\Delta = 7$ and $i = 1$, $L_{7,1,0} = 7, L_{7,1,1} = 105, L_{7,1,2} = 210$, and $L_{7,1,3} = 35$. These add up to 357, which is the i-th element of row Δ of the trinomial triangle. From Expression 6.1 and the weights in Section 6.5, it comes that $e_{\Delta=7,1} = 7c_0^6c_1^1 + 105c_0^4c_1^3 + 210c_0^2c_1^b + 35c_0^0c_1^7$.

6.7 Some Integer Sequences

An integer sequence with closed formula can be obtained by orderly arranging the $L_{\Delta,i,k}$'s, varying the indexes from the minimum up to the maximum value, starting with the least significant index k, then i, etc. The range for k can be either $0 \leq k \leq K_{\Delta,i} - 1$ or $1 \leq k \leq K_{\Delta,i}$, depending on the adopted indexing mode, with no difference to the sequence, where K is calculated as in Step 6.2. of Section 6.6. There are $\Delta + 1$ values for index i, in the range $0 \leq i \leq \Delta$. The index Δ represents a convolution degree and at first glance it should be higher than zero, but if the definitions $K_{0,0} = 1$ and $L_{0,0,0} = 1$ are applied, then $0 \leq \Delta < \inf$. This way, for example, the first eight terms of the sequence are named $L_{0,0,0}, L_{1,0,0}, L_{1,1,0}, L_{2,0,0}, L_{2,0,1}, L_{2,1,0}, L_{2,2,0}, \ldots$. The following is a list of the first seventy terms, where colon is used to group elements of same Δ, and semicolon, of same Δ and i: $1:1;1:1,2;2;1:1,6;3,3;3;1:1,12,6;4,12;6,4;4;1:1,20,30;5,30,10;10,20;10,5;5;1:1,30,90,20;6,60,60;15,60,15;20,30;15,6;6;1:1,42,210,140;7,105,210,35;21,140,105;35,105,21;35,42;21,7;7;1:\ldots$ These seventy elements describe every polynomial coefficient for every convolved stencil coefficient, up to convolution degree seven.

The present work proposes to call the elements of the above sequence of $L_{\Delta,i,k}$'s, due to an apparent lack of previous disclosure in the literature, as the *Butantan numbers*, in reference to the location of the main campus of the University of São Paulo.

Figure 21: Pyramids represented with the shape of influence tables of two bi-dimensional kernels.



(a) (b)

Source: Author.

Multiple other sequences related to the convolved coefficients, thus to kernel convolution as well, can be devised. The following is a non-exhaustive list of examples. Whenever an entry related to a sequence could be found in the OEIS, the associated OEIS identification is provided here:

Regarding 1D kernels — Whereas the Butantan numbers represent $L_{\Delta,i,k}$'s with the three indexes varying, it has been stated in Section 6.4 that $L_{\Delta,0,k}$ is associated to sequence A105868. These are the L's associated to the central trinomial numbers $1,1,3,7,19,51,\ldots$ (see Figure 20d), themselves associated to sequence A002426. The summation of the K's of each convolution degree Δ is the total number of terms in the polynomials of all convolved coefficients of kernel O^{Δ}. It can be expressed as $S_{\Delta} = \sum_i K_{\Delta,i}$, so that $S_0, S_1, S_2, S_3, \ldots$ describes sequence A002620 (see Figure 20e). A sequence related to the accumulation of S's can be determined if the Δ-th element is given by $\sum_{d=0}^{\Delta} S_{\Delta}$. This sequence is also shown in Figure 20e, up to the eighth element. It corresponds to A002623.

Regarding 2D kernels — Similar to how successive influence tables of 1D kernels describe a triangle, those of 2D kernels describe a pyramid. Figures 21a and 21b re-

present pyramids with the influence tables of O^1, O^2, and O^3 of the 2D 5-point and 2D 9-point stencils with symmetric coefficients, with one extra element on top, for the sake of mathematical extrapolations. The sequences of central elements of the pyramids are each indicated by an arrow. Similar to how in 1D scenarios the sum of the polynomial coefficients of $P_{\Delta,0}$, which is used to calculate $e_{\Delta,0}$, results in the central element of the Δ-th row of the trinomial triangle and thus describes a sequence, the sum of the polynomial coefficients to calculate the central elements $e_{\Delta,(0,0)}$ of Figure 21a describes the sequence $1,1,5,13,61,221,1001,4145,18733,\ldots$. It corresponds to A201805 in the OEIS. The second, third, and fourth elements can be appreciated in Figures 5 and 10, the fourth also in Figure 19. The fifth element appears in Figure 26. Recall from Section 6.4 that each sum in question corresponds to a convolved coefficient in case the original coefficients are made to be 1. With this in mind, in case the original stencil is the square 2D 9-points, the central elements in Figure 21b are $1,1,9,49,361,2601,19881,154449,\ldots$. These are the squares of the central trinomial numbers listed above, and describe A168597. The diagonal elements $e_{\Delta,(d,d)}$ of the pyramid in Figure 21a exist when $d = \Delta/2$, that is, when the convolution degree is pair. Their sequence is $1,2,6,20,70,252,924,\ldots$. It corresponds to A002623. They describe the central binomial coefficients, which appears as a special case of the grid traversal discussed in Section 6.3, when h equals w, in addition to $h+w$ equalling the convolution degree. In Figure 26, $e_{4,(2,2)} = 6 = \binom{4}{2}$. In Figures 5b and 10, if $k_1 = 1$ and $d = e = 1$, then both have $e_{2,(1,1)} = 2 = \binom{2}{1}$.

In the case of a star-shaped stencil with radius 2, that is, of 9 points, the central elements of successive influence tables would describe the sequence $1,1,9,37,265,1721,12861,96909,760649,\ldots$. Again, this assumes unitary coefficients. Without loss of generality, in case the coefficients are not 1 then this sequence does not reflect the actual influence tables, but describes the sums of the polynomial coefficients of the expressions to calculate the central elements. This sequence does

not find correspondence in the OEIS.

Regarding 3D kernels — If the star-shaped 3D 7-point stencil is taken into account, and the coefficients are made unitary as described above, then the succession of the central elements of the influence tables is $1, 1, 7, 19, 127, 511, 3301, 16297, 103279, \ldots$. It corresponds to A328713. If the cube 3D 27-point stencil is taken into account, the succession is $1, 1, 27, 343, 6859, 132651, \ldots$. This sequence does not find correspondence in the OEIS.

6.8 Practical Approaches to a Theoretical Quest

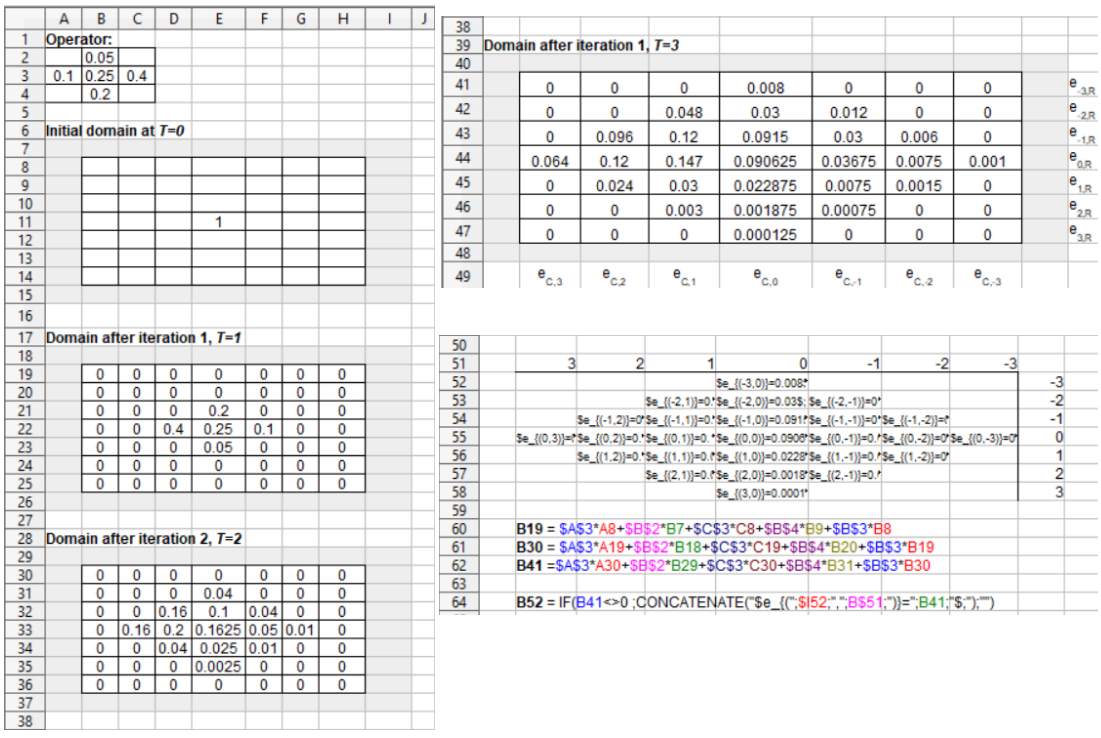
With the increase in the radius, asymmetry, dimension, and convolution degree, it becomes harder to infer general formulas for the convolved coefficients. This section discusses alternatives that analysts, researchers, and developers can employ to determine them when projecting, analysing, coding, automating, etc. high-performance stencil codes. The approaches can be *convergent* or *divergent*. A convergent approach (Section 6.8.4) focus on how all the points of the domain effect a given point, so that the result for that point represents the convolved operator. Divergent approaches focus on how one point effects the neighbors, and the result is the convolved operator flipped in every dimension and centered at that point.

6.8.1 Spreadsheet Processor

Figure 22 illustrates how to implement influence tables up to $T = 3$ of a 2D 5-point stencil. The rectangular range delimited by cells A1 and I48 entails the actual calculation process, whereas column J and the cells from row 49 downwards are merely expository. The first step is to define the size of the portion of domain points necessary to develop the calculation, including ghost zones, if necessary. Then fill up the static

values, namely, the operator, and the central point of the initial version of the domain. To implement influence tables, this has to be 1 (cell E11). If the first points of the domain versions are equidistant in the spreadsheet, then the only necessary formula to populate the dynamic cells is that of the first point of the domain at $T = 1$ (cell B19). Figure 22 shows this formula on cell B60. It merely multiples the neighboring cells of the first point (B8) of the initial domain version by the relevant coefficients of the operator. If this formula is expanded to act upon the whole domain at $T = 1$, then this becomes a flipped version of the operator. Because the domain versions are equidistant, now the whole domain at $T = 1$ (range determined by B19 and H25) can be copied and pasted onto the spaces previously determined to hold versions $T = 2$ and $T = 3$. Figure 22 displays on cells B61 and B42 how the formulas of the first points of the new domain versions should look like at this point.

Figure 22: A possible way to determine convolved coefficients from a spreadsheet software.

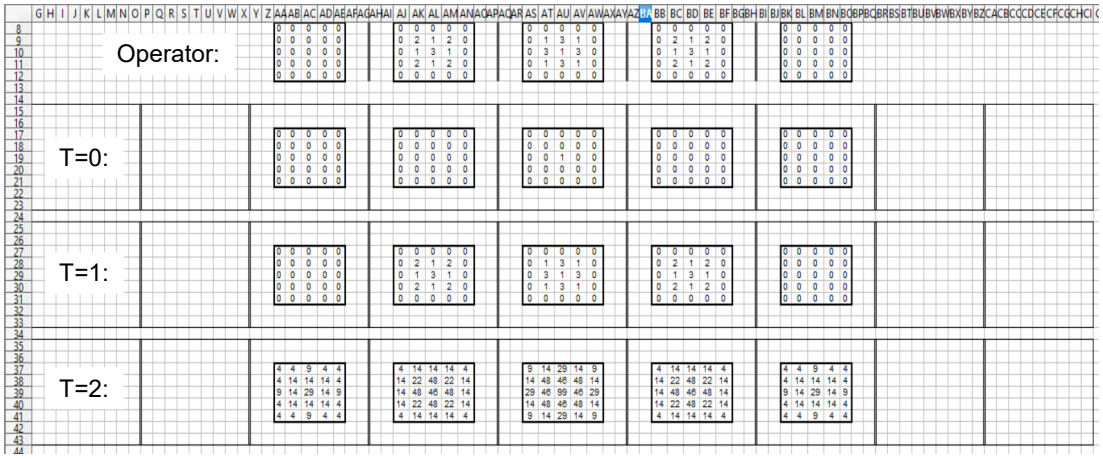


Source: Author.

Now the non-zero values in the range B41–H47 represent O^3 flipped in every di-

mension. Column J and row 49 call attention to this flipped nature. The range B51–I58 helps to export the coefficients. For example, if the formula of B52 is defined as shown on cell B62, then expanded to every cell in B52–H58, then this range holds the coefficients in a pattern suitable for LaTeX. The forty-nine cells in this rage can be at once copied and pasted between double quotation marks in a LaTeX processor to yield: "$e_{(-3,0)} = 0.008$; $e_{(-2,1)} = 0.048$; $e_{(-2,0)} = 0.03$; $e_{(-2,-1)} = 0.012$; $e_{(-1,2)} = 0.096$; $e_{(-1,1)} = 0.12$; $e_{(-1,0)} = 0.0915$; $e_{(-1,-1)} = 0.03$; $e_{(-1,-2)} = 0.006$; $e_{(0,3)} = 0.064$; $e_{(0,2)} = 0.12$; $e_{(0,1)} = 0.147$; $e_{(0,0)} = 0.090625$; $e_{(0,-1)} = 0.03675$; $e_{(0,-2)} = 0.0075$; $e_{(0,-3)} = 0.001$; $e_{(1,2)} = 0.024$; $e_{(1,1)} = 0.03$; $e_{(1,0)} = 0.022875$; $e_{(1,-1)} = 0.0075$; $e_{(1,-2)} = 0.0015$; $e_{(2,1)} = 0.003$; $e_{(2,0)} = 0.001875$; $e_{(2,-1)} = 0.00075$; $e_{(3,0)} = 0.000125$;".

Figure 23: Example of 3D influence table.



Source: Author.

Figure 23 exemplifies how to implement influence tables for 3D operators. In this case the operator is split into 2D layers or planes. Therefore the flipped version of O^2 is represented from column 37 to 41. There is enough room for the operator to be 13-point (star-shaped with radius 2) or even the 125-point, among others, but by leaving many of the operator's cells as 0 (in practice, they could have been left blank), Figure 23 exemplifies the 27-point operator. Listing 6.1 conveys a complete *Python* program that facilitates writing the formula of the cells that represent the domain points

at $T = 1$ and $T = 2$. The output of this program should be applied once to the central domain point of $T = 1$ (cell AU29). From this, it can be easily expanded to act upon the whole central slice of this domain. Then the holy slice can be copied and pasted onto the other slices of this domain. Finally the whole domain can be copied at once to fill up the version $T = 2$, if the operator and successive domains are equidistant.

Listing 6.1: Program to write formulas for a spreadsheet-facilitated way to compute convolved coefficients

```
operator_columns = ['AA','AB','AC','AD','AE',
'AJ','AK','AL','AM','AN',
'AS','AT','AU','AV','AW',
'BB','BC','BD','BE','BF',
'BK','BL','BM','BN','BO']

operator_lines = range(8,13)
doo = 9 #distance_from_operator_to_original_domain

terms = []

for col in operator_columns:
    for row in operator_lines:
        term = '$'+col+'$'+str(row)+"*"+col+str(row+doo)
        terms.append(term)
print ("="+'+'.join(terms))
```

6.8.2 Direct Code

Another *divergent* approach is to run a program that will produce successive versions of the domain. The code excerpt in Listing 6.2 keeps in memory every version of the domain, until the iteration relative to the reduction in iteration space (convolution degree) is reached. The code was also developed to facilitate various dimensions, but this version does not take into account radius higher than 1. To do so, the analyst would have to adjust the ghost region, to guarantee no memory access error.

Once the program finishes, the non-zero values of grid[T][mid][][] will contain the flipped version of the 2D operator stored in the matrix operator. Note that the intent is not to perform the stencil computation with this approach, only to determine the

Listing 6.2: Code to help determine convolved coefficients

```
#define CASE case2DsquareR1
#define SIZE 27
#define T 4
#define OP(di,dj,dk) operator[OPMID+di][OPMID+dj][OPMID+dk]
/* Initialization code suppressed here */
  const int mid = SIZE/2;
  grid[0][mid][mid][mid] = 1;
  for(t=0; t<=T; t++)
    for(i=1;i<SIZE-1;i++)
    for(j=1;j<SIZE-1;j++)
      for(k=1;k<SIZE-1;k++){
          if (CASE == case2DsquareR1){
grid[t+1][i][j][k]= grid[t][i][j][k]*OP(0,0,0) +
grid[t][i][j][k-1]*OP(0,0,-1)+grid[t][i][j][k+1]*OP(0,0,1)+
grid[t][i][j-1][k]*OP(0,-1,0)+grid[t][i][j+1][k]*OP(0,1,0)+
grid[t][i][j-1][k-1]*OP(0,-1,-1)+grid[t][i][j-1][k+1]*OP(0,-1,1)+
grid[t][i][j+1][k-1]*OP(0,1,-1)+grid[t][i][j+1][k+1]*OP(0,1,1);
          }
        }
```

convolved coefficients. Neither is the intent to store the coefficients in a matrix.

6.8.3 BFS Approach

The third non-analytical approach to determine the convolved coefficients is to genuinely regard an influence table as reflecting a wavefront propagation and confine the calculations to just the domain points reached by the desired iteration. Algorithm 6.1 performs this process for a generic stencil, whereas Figure 24 shows an implementation of this algorithm in *Python* next to the program's output. By appropriately setting the inputs to the algorithm, the figure calculates the influence table of O^3 of Figure 11.

Algorithm 6.1: Algorithmic approach to define convolved coefficients. Line 4 indicates a 2D domain.

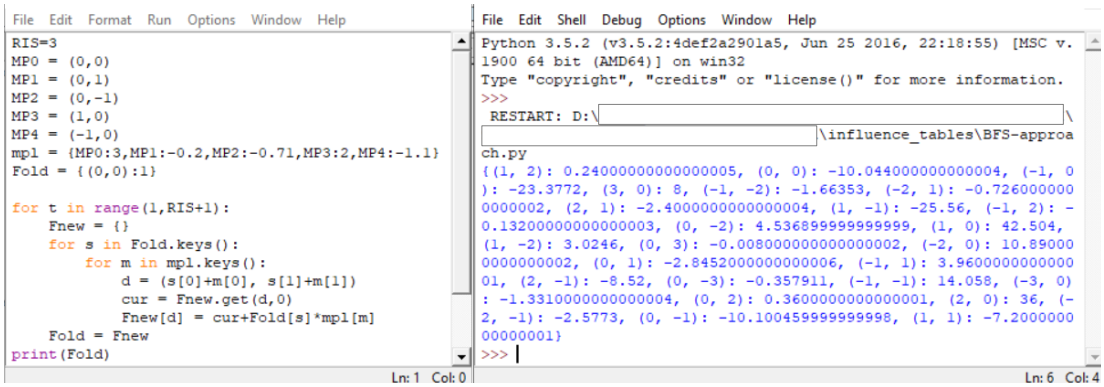
```
1  input: function M, int RIS
2  output: Flipped Influence Table
3  begin
4    F_old : {(0,0) ⟼ 1}
5    t ← 1
6    while t ≤ RIS
```

```
7        F_new ← ∅
8        foreach  s ∈ Dom(F_old)
9          foreach  m ∈ Dom(M)
10           d ← s + m
11           cur ← F_new(d), if d ∈ Dom(F_new), otherwise 0
12           Update F_new ← F_new \ {(d, f_new(d)} ∪ {d ⟼ cur + F_old(s) × M(m)}
13         end
14       end
15       F_old ← F_new
16     end
17     return  F_old
18   end
```

Algorithm 6.1 takes as input the desired reduction in iteration space RIS and a list M of movement patterns, each pattern associating a displacement vector with an influence factor or weight. The algorithm then executes a series of steps. Step 1 is to create a list associating the reached domain points to a value, and initialize the list with one point of value 1. This point represents the central point of the influence table and for convenience usually has coordinates zero. Mathematically, M and the list of reached domain points can be regarded as functions in $\mathbb{Z} \times \mathbb{Z} \longrightarrow \mathbb{R}$. Programmatically, they can be implemented with hash tables and the like, or as Python dictionaries in Figure 24. Let F_{old} denote said initial list of reached points, and F_{new} denote a list that is rebuilt during each time iteration. After the above F_{old} initialization, proceed as follows: (Step 2) For each time iteration $t \leq$ RIS, (S. 3) initialize F_{new} with the empty set. Then (S. 4) for each point p reached before current time iteration ($\forall p \in Dom(F_{old})$), and (S. 5) for each movement pattern m ($\forall m \in Dom(M)$), proceed as follows. Reach the destination point d as dictated from the movement pattern m and source point p, by (S. 6) adding it to the list F_{new} of points reached during the current time iteration, if necessary, and by (S. 7) influencing its value in F_{new} with the appropriate factor. Steps 6 and 7 correspond to lines 10, 11, and 12 in Algorithm 6.1. Next, Step 8 is to repeat Steps 4 through 7 for every m and s. Then (S. 9) make F_{old} become a copy of F_{new}, and repeat Steps 2 through 9 for every t. After completion, F_{old} contains an influence

table, corresponding to the flipped version of the convolved operator.

Figure 24: Example of Python 3.5.2 implementation of Algorithm 6.1 to calculate the influence table for O^3 of Figure 11.



Source: Author.

6.8.4 Literal Arithmetic

The fourth method to determine the coefficients relies on arithmetic of literal values. The literal approach does not require the coefficients to be numerically predetermined, whereas the three methods discussed above do. In essence, it could allow to understand how each original coefficient individually affects the cells of the influence table. Due to being literal, this approach also allows to trace back how each cell changes another. Thus this method does not need to focus on the initial propagation of only one cell of unitary value, what is a possibility explored in this section to develop a convergent approach.

The conundrum with this approach is the manipulation of literal values. Although libraries can be found which allow such manipulation, utilizing them for the task ahead usually comes with two major set backs: difficulty in automating the interaction with them, what would impose even a bigger problem in case the literal arithmetic approach were to be connected with an auto-tuner, compiler, etc., and difficulty in extracting the relevant excerpt of the source code and dependent files, if they even are available. It is probably the case that stencil researchers would have significant leeway if they

were in possession of a clean code to explore literal expressions that emerge with stencil and kernel convolution. Therefore the remaining of this section shows the most relevant features of an implementation in C that allows to find literal expressions for the convolved coefficients, whereas Appendix A brings a complete program for such.

The first requirement for the implementation is to allow the recovery of the original domain points that affect a given point after some time iterations, along with a multiplication factor. To recover an original point can be interpreted as recovering the points' name or identification. The second requirement is that each multiplication factor be traceable to the coefficients of the original operator.

$$A = \sum_{\mathbf{p} \in D} P \times e_{A,\mathbf{p}} \tag{6.2}$$

$$e_{A,\mathbf{p}} = \sum_{k=1}^{K_{A,\mathbf{p}}} L_{A,\mathbf{p},k} \prod_{j=0}^{J} c_j^{g_{A,\mathbf{p},k,j}} \forall \mathbf{p} \in D \tag{6.3}$$

Equations 6.2 and 6.3 express both requirements. Variable A is the value of one point after the desired iterations, boldface $\mathbf{p}$ represents a domain point in the domain D, P represents the initial value of $\mathbf{p}$, and $e_{A,\mathbf{p}}$ is a literal expression of how every $\mathbf{p}$ changes A. It is a requirement that P be literal. A possible implementation comes from regarding A as an aggregate of pairs of a point and a multiplicative literal expression. In this case, if $e_{A,\mathbf{p}} = 0$ for some P it is not necessary to represent the pair in the aggregate. This way, the number of points represented in A coincides with the number of original domain values that influence the current value of A. In Equation 6.3, J and the c_j's are the same across every $e_{A,\mathbf{p}}$, with J usually small and c_j representing a literal that should be traceable, namely, an original coefficient. A possible implementation is to codify $\prod_{j=0}^{J} c_j^{g_{A,\mathbf{p},k,j}}$ into an array of size $J+1$ with the value of the j-th element representing $g_{A,\mathbf{p},k,j}$. In this case, if a given original coefficient does not appear in the equation of $e_{A,\mathbf{p}}$, then the related array element can be set to 0. Also, each $e_{A,\mathbf{p}}$ should be associated

with $K_{A,\mathbf{p}}$ such arrays, each array associated to one non literal factor $L_{A,\mathbf{p},k}$.

Figure 25: Snippet of a possible C implementation that works with arithmetic of literals to determine expressions for convolved coefficients.

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

const char literals[] = "abcdefghijkl";
#define MAX_LITERALS sizeof(literals)
#define MAX_TERMS 100

typedef struct coeff{
    int mul;
    int exps[MAX_LITERALS];
} COEFFS;

typedef struct {
    char var_name[9];
    int combs_total;
    COEFFS coeff_list[MAX_TERMS];
} POINT;
typedef struct {
    char var_name[9];
    int points_total;
    POINT point_list[100];
} AGGREGATE;
```

Source: Author.

Figure 25 shows a code snippet of a possible implementation of the relationships described above. In this implementation, each A is represented by an *AGGREGATE* structure that allows for up to 100 pairs of P and $e_{A,\mathbf{p}}$. The $e_{A,\mathbf{p}}$'s are encoded into the *POINT* structure, under the constraint $K_{A,\mathbf{p}} \leq MAX\_TERMS$. The structure $COEFFS$ stores $L_{A,\mathbf{p},k}$ into the variable *mul* and $\prod_{j=0}^{J} c_j^{g_{A,\mathbf{p},k,j}}$ into the array *exps*, with $J+1 \leq$ *MAX_LITERALS*.

A third requirement is that the following operation should be allowed among aggregates: $A \leftarrow c_u U + c_v V$, where A, U, V are aggregates, and c_u, c_v are coefficients of the original operator. This implies the need of support for three main operations or functions: attribution of the value of one aggregate to another, the multiplication of an aggregate by a literal coefficient, and the addition of aggregates. Attribution can be decomposed into the initialization of the receiving aggregate followed by an addition. The multiplication by a coefficient can follow the rationale that $A \leftarrow c_a A \implies e_{A,\mathbf{p}} \leftarrow c_a e_{A,\mathbf{p}} \forall \mathbf{p} \in A$, that is, apply the coefficient to every point that takes part in the expression of A, implying $e_{A,\mathbf{p}} \leftarrow \sum_{k=1}^{K_{A,\mathbf{p}}} L_{A,\mathbf{p},k} (c_a \prod_{j=0}^{J} c_j^{g_{A,\mathbf{p},k,j}}) \implies e_{A,\mathbf{p}} \leftarrow \sum_{k=1}^{K_{A,\mathbf{p}}} L_{A,\mathbf{p},k} (\prod_{j=0}^{J} c_j^{g'_{A,\mathbf{p},k,j}})$, where $g'_{A,\mathbf{p},k,a} \leftarrow 1 + g_{A,\mathbf{p},k,a}, \forall k$, and $g'_{A,\mathbf{p},k,j} \leftarrow g_{A,\mathbf{p},k,j}, \forall k, j \neq a$.

The addition of aggregates, $A \leftarrow A + B$, can be decomposed into the addition of

points to an aggregate, $A \leftarrow A + \sum_{\mathbf{p} \in B} P \times e_{B,\mathbf{p}}$, and implemented as such. This in turn can be implemented as multiple calls to a function that adds one point of B at a time, $A \leftarrow A + P \times e_{B,\mathbf{p}}$. In this case, if $\mathbf{p} \notin A$, then a copy of the whole point can be appended to A. Otherwise, if $\mathbf{p} \in A$, the addition can be further decomposed into $K_{B,\mathbf{p}}$ calls to $e_{A,\mathbf{p}} \leftarrow e_{A,\mathbf{p}} + L_{B,\mathbf{p},k} \prod_{j=0}^{J} c_j^{g_{B,\mathbf{p},k,j}}$, where the products in the right-hand side of the expression are encoded into one structure *COEFFS* of Figure 25. Finally, if $\exists h$ such that $c_j^{g_{A,\mathbf{p},h,j}} = c_j^{g_{B,\mathbf{p},k,j}} \forall j$, then simply $L_{A,\mathbf{p},h} \leftarrow L_{A,\mathbf{p},h} + L_{B,\mathbf{p},k}$. Otherwise the whole structure that codifies $L_{B,\mathbf{p},k} \prod_{j=0}^{J} c_j^{g_{B,\mathbf{p},k,j}}$ can be copied and appended to $e_{A,\mathbf{p}}$.

Appendix A exemplifies a complete implementation of the above structures being used. Some features of the implementation should nonetheless be mentioned here. For the discussion, consider a 1D domain of d points, encoded into an array of d elements, with indexes varying from 0 to $d-1$, and a symmetric 1D 3-point operator with coefficients c_0 and c_1. For the sake of straightforward visualization in a computer terminal, let the coefficients be respectively renamed a and b in the implementation. A possible way to use the framework is to initially create d *POINT* structures with names (Line 14 in Figure 25) ranging from $I0, I1, I2$ to $d-1$, where the uppercase letter 'I' stands for 'input', and create d *AGGREGATE* structures with names (Line 19) $X0, X1, X2, \ldots$. Then add each point to the corresponding aggregate. By now each aggregate can have its value traced back to a single input point, represented with a literal value. Because by this time the influence of border conditions is not under scrutiny, and given that the value of $X0$ after one iteration depends on the left border of the domain and the value of the rightmost aggregate depends on the right border, then in order to calculate something similar to the influence table of RIS = 1, proceed to update the values of the points $X1$ through X_{d-2}. Analogously, to calculate something similar to the influence table of RIS = 2, proceed to update the points $X2$ through X_{d-3}. After a total of RIS =Δ such iterations, the aggregate structures X_i far enough from the border will hold inside themselves something similar to the influence table of O^{Δ} (in

the example, for the aggregates X_i with $\Delta \leq i \leq X_{d-1-\Delta}$). The information contained is said to be 'something similar' rather than just 'similar' to an influence table because this particular implementation resulted in a convergent approach, meaning that in the expression obtained for X_i the convolved coefficient applied to X_{i+2} corresponds to e_2, rather than e_{-2}, in other words, this convergent approach did not result in a flipped representation of O^2, as would happen with regular influence tables, but rather to a non-flipped version.

For example, had the stencil in the example above been asymmetric with update equation $X_{i,t+1} = bX_{i-1,t} + aX_{i,t} + cX_{i+1,t}$, then an excerpt of the output related to RIS=2 would be like Listing 6.4, indicating that $X_{8,t+2} = b^2X_{6,t} + 2abX_{7,t} + (a^2 + 2bc)X_{8,t} + 2acX_{9,t} + c^2X_{10,t}$, so that the convolved coefficient e_2 that should be applied to $X_{8+2,t}$ to obtain $X_{8,t+\Delta}$ is c^2, thus confirming that the output is not a flipped representation of the convolved operator O^{Δ}.

Listing 6.4: Part of output when using the approach of arithmetic with literals.

```
1 Aggregate: X8 = {I8*(a^2+2*b^1*c^1)
2 +I7*(2*a^1*b^1)
3 +I9*(2*a^1*c^1)
4 +I6*(b^2)
5 +I10*(c^2)
6 }
```

6.9 Convolved Coefficients of All Operators in the Experiments

The convolved coefficients for the symmetric 1D 3-point stencil are:

$$
\begin{array}{lll}
O^2 & O^3 & O^4 \\
e_0 = a^2 + 2b^2 & e_0 = a^3 + 6ab^2 & e_0 = a^4 + 12a^2b^2 + 6b^4 \\
e_1 = 2ab & e_1 = 3a^2b + 3b^3 & e_1 = 4a^3b + 12ab^3 \\
e_2 = b^2 & e_2 = 3ab^2 & e_2 = 6a^2b^2 + 4b^4 \\
 & e_3 = b^3 & e_3 = 4ab^3 \\
 & & e_4 = b^4
\end{array} \tag{6.4}
$$

$$
\begin{array}{lll}
O^5 & O^6 & O^7 \\
e_0 = a^5 + 20a^3b^2 + 30ab^4 & e_0 = a^6 + 30a^4b^2 + 90a^2b^4 + 20b^6 & e_0 = a^7 + 42a^5b^2 + 210a^3b^4 + 140ab^6 \\
e_1 = 5a^4b + 30a^2b^3 + 10b^5 & e_1 = 6a^5b + 60a^3b^3 + 60ab^5 & e_1 = 7a^6b + 105a^4b^3 + 210a^2b^5 + 35b^7 \\
e_2 = 10a^3b^2 + 20ab^4 & e_2 = 15a^4b^2 + 60a^2b^4 + 15b^6 & e_2 = 21a^5b^2 + 140a^3b^4 + 105ab^6 \\
e_3 = 10a^2b^3 + 5b^5 & e_3 = 20a^3b^3 + 30ab^5 & e_3 = 35a^4b^3 + 105a^2b^5 + 21b^7 \\
e_4 = 5ab^4 & e_4 = 15a^2b^4 + 6b^6 & e_4 = 35a^3b^4 + 42ab^6 \\
e_5 = b^5 & e_5 = 6ab^5 & e_5 = 21a^2b^5 + 7b^7 \\
 & e_6 = b^6 & e_6 = 7ab^6 \\
 & & e_7 = b^7
\end{array} \tag{6.5}
$$

The convolved coefficients for the symmetric 1D 5-point stencil, with $c_0 = a, c_1 = b, c_2 = c$:

$$
\begin{aligned}
e_0 &= a^2 + 2b^2 + 2c^2 & e_0 &= a^3 + 6ab^2 + 6ac^2 + 6b^2c \\
e_1 &= 2ab + 2bc & e_1 &= 3a^2b + 6abc + 3b^3 + 6bc^2 \\
e_2 &= 2ac + b^2 & e_2 &= 3a^2c + 3ab^2 + 6b^2c + 3c^3 \\
e_3 &= 2b^1c^1 & e_3 &= 6abc + 3bc^2 + b^3 \\
e_4 &= c^2 & e_4 &= 3ac^2 + 3b^2c \\
& & e_5 &= 3bc^2 \\
& & e_6 &= c^3
\end{aligned}
\tag{6.6}
$$

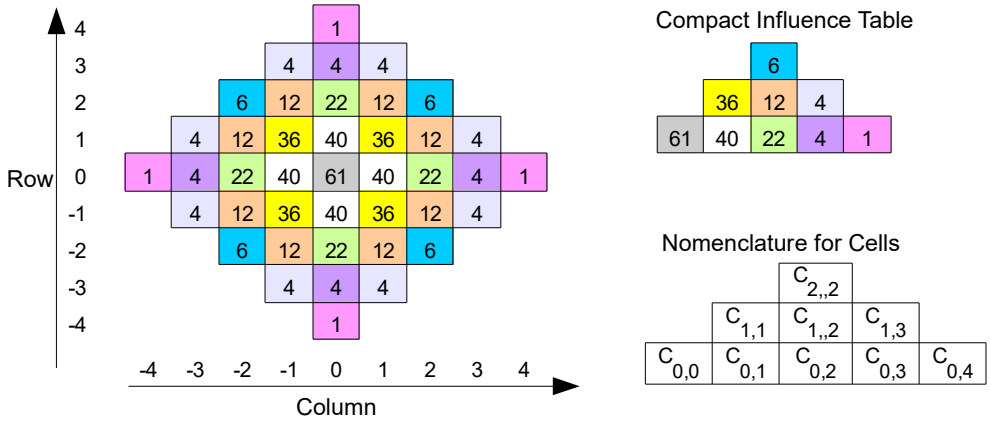
The convolved coefficients for the symmetric 1D 7-point stencil, with $c_0 = a, c_1 = b, c_2 = c, c_3 = d$:

$$
\begin{aligned}
e_0 &= a^2 + 2b^2 + 2c^2 + 2d^2 & e_4 &= 2bd + c^2 \\
e_1 &= 2ab + 2bc + 2cd & e_5 &= 2cd \\
e_2 &= 2ac + 2bd + b^2 & e_6 &= d^2 \\
e_3 &= 2ad + 2bc
\end{aligned}
\tag{6.7}
$$

6.9.1 Coefficients for 2D 5-point, RIS = 4

Consider the traditional 2D 5-point stencil operator with symmetric coefficients, such as the one in Figure 19a, where $c_{0,0} = a$, and $c_{0,1} = b$. To develop expressions for the convolved coefficients of the related operator with convolution degree 4, first consider the Influence Table of RIS $= 4$ of the operator O^1 in Figure 19b, whose every coefficient matches the unity. Such Influence Table is represented in Figure 26, which also represents O^4, due to the symmetry. The color code of this figure highlights the symmetry that O^4 retained from O^1. It was shown that when every coefficient of the influence table is 1 the value of the cells of the table represent the sum of the polynomial coefficients that describe the convolved stencil coefficient related to each cell. Therefore Figure 26 also works as a guide when deducing the coefficients.

Figure 26: Influence Table of a symmetric stencil.



Source: Author.

Because of the symmetry, it is possible to study the convolved coefficients by focusing solely on the first octant, represented as a Compact Influence Table in Figure 26. The figure also offers a name mapping for the nine distinct cell values of the influence table for O^4. The value of each cell conveys the amount of influence paths that start in $(0,0)$ and ends at the cell in exactly $\Delta = 4$ movements. To find expressions for the convolved coefficients $e_{R,C}$, where R and C stand for row and column according to the identification employed in the figure, is to find the amount and weight of the influence paths. The cells where $R+C = \Delta$ have influence paths made of solely side influences, thus with weight $b^{\Delta} = b^4$, and there are $\binom{R+C}{C}$ such paths. To assess the paths of the other cells $C_{R,C}$ one can regard them as made of r,l,u,d, and α, respectively, right, left, up, down, and auto-influences, with the five variables subject to the constraints $r-l = C$, $u-d = R$, and $r+l+u+d+\alpha = \Delta$.

Table 6 shows the possible combinations of influence paths for the cells when the row R and the column C do not add up to the convolution degree Δ. Because of the symmetry, $C_{R,C} = C_{-R,-C} \equiv e_{R,C}$, so that Equations 6.8 express the results from the table and from the more trivial cases when $R+C = \Delta$ in terms of convolved coefficients.

Take for example the third, fourth, and fifth rows of Table 6. They correspond to the three patterns of influence paths that reach $C_{0,1}$ in 4 movements. They indicate

Table 6: Breakdown of the influence paths of cells $C_{R,C}$ in Figure 26, for $R+C \neq \Delta$.

Cell	r, l	u, d, α	W.	Amount	Expr.
$C_{0,3}$	$r=3 \Longrightarrow l=0$	$u=0 \Longrightarrow d=0, \alpha=0$	ab^3	$\binom{4}{3}\binom{1}{1}$	$4ab^3$
$C_{1,2}$	$r=2 \Longrightarrow l=0$	$u=1 \Longrightarrow d=0, \alpha=1$	ab^3	$\binom{4}{2}\binom{2}{1}$	$12ab^3$
$C_{0,1}$	$r=1 \Longrightarrow l=0$	$u=0 \Longrightarrow d=0, \alpha=3$	a^3b	$\binom{4}{1}\binom{3}{3}$	$4a^3b$
		$u=1 \Longrightarrow d=1, \alpha=1$	ab^3	$\binom{4}{1}\binom{3}{1}\binom{2}{1}$	$24ab^3$
	$r=2 \Longrightarrow l=1$	$u=0 \Longrightarrow d=0, \alpha=1$	ab^3	$\binom{4}{2}\binom{2}{1}$	$12ab^3$
$C_{0,2}$	$r=2 \Longrightarrow l=0$	$u=0 \Longrightarrow d=0, \alpha=2$	a^2b^2	$\binom{4}{2}\binom{2}{2}$	$6a^2b^2$
		$u=1 \Longrightarrow d=1, \alpha=0$	b^4	$\binom{4}{2}\binom{2}{1}$	$12b^4$
	$r=3 \Longrightarrow l=1$	$u=0 \Longrightarrow d=0, \alpha=0$	b^4	$\binom{4}{3}$	$4b^4$
$C_{1,1}$	$r=1 \Longrightarrow l=0$	$u=1 \Longrightarrow d=0, \alpha=2$	a^2b^2	$\binom{4}{1}\binom{3}{2}$	$12a^2b^2$
		$u=2 \Longrightarrow d=1, \alpha=0$	b^4	$\binom{4}{1}\binom{3}{2}$	$12b^4$
	$r=2 \Longrightarrow l=1$	$u=1 \Longrightarrow d=0, \alpha=0$	b^4	$\binom{4}{2}\binom{2}{1}$	$12b^4$
$C_{0,0}$	$r=0 \Longrightarrow l=0$	$u=0 \Longrightarrow d=0, \alpha=4$	a^4	$\binom{4}{4}$	a^4
		$u=1 \Longrightarrow d=1, \alpha=2$	a^2b^2	$\binom{4}{1}\binom{3}{1}$	$12a^2b^2$
		$u=2 \Longrightarrow d=2, \alpha=0$	b^4	$\binom{4}{2}\binom{2}{2}$	$6b^4$
	$r=1 \Longrightarrow l=1$	$u=0 \Longrightarrow d=0, \alpha=2$	a^2b^2	$\binom{4}{1}\binom{3}{1}$	$12a^2b^2$
		$u=1 \Longrightarrow d=1, \alpha=0$	b^4	$\binom{4}{1}\binom{3}{1}\binom{2}{1}$	$24b^4$
	$r=2 \Longrightarrow l=2$	$u=0 \Longrightarrow d=0, \alpha=0$	b^4	$\binom{4}{2}\binom{2}{2}$	$6b^4$

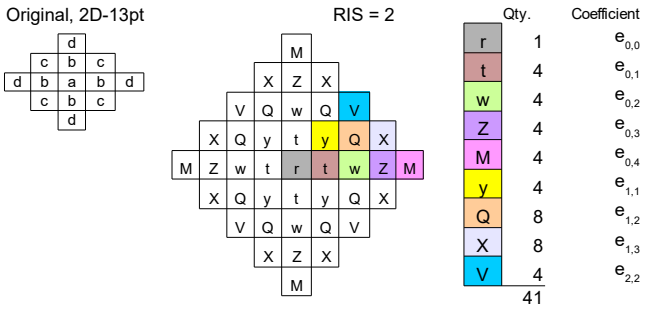
that if the influence path contains 1 right influence, then it necessarily contains no left influence. This leaves room for two patterns. If the path in question then contains 1 up influence, then it implies that the path necessarily contains 1 down influence and 1 auto-influence. This pattern has weight ab^3 and there are 24 distinct influence paths of this pattern. Equations 6.8 show the convolved coefficients of O^4:

$$
\begin{aligned}
e_{0,0} &= a^4 + 24a^2b^2 + 36b^4 & e_{1,1} &= 12a^2b^2 + 24b^4 \\
e_{0,1} &= 4a^3b + 36ab^3 & e_{1,2} &= 12ab^3 \\
e_{0,2} &= 6a^2b^2 + 16b^4 & e_{1,3} &= 4b^4 \\
e_{0,3} &= 4ab^3 & & \\
e_{0,4} &= 1b^4 & e_{2,2} &= 6b^4
\end{aligned}
\tag{6.8}
$$

6.9.2 Coefficients for higher order 2D 13-point and square 2D 9-point

Figure 27 displays a 2D original stencil of 13 points and symmetric, along with its convolved version. Letters were used to highlight the symmetry with original coefficients $c_{0,0} = a$, $c_{0,1} = b$, $c_{0,2} = d$, and $c_{1,1} = c$. The convolved operator with reduction in iteration space 2 has 41 points. The figure highlights with colour codes one exemplar of each of the 9 distinct coefficients. Letters have been used to represent the coefficients to leave the images of the operators less crowded. The figure also shows how in this case each name given by a letter translates to the usual system of coordinates used in this work for convolved coefficients. The set of Equations 6.9 express the convolved coefficients in terms of the originals.

Figure 27: Coefficients of O^2 for an original symmetric 13-point 2D stencil.



Source: Author.

$$
\begin{aligned}
e_{0,0} &= a^2 + 4b^2 + 4c^2 + 4d^2 & e_{1,1} &= 2ac + 2b^2 + 4cd \\
e_{0,1} &= 2ab + 2bd + 4bc & e_{1,2} &= 2bc + 2bd \\
e_{0,2} &= 2ad + b^2 + 2c^2 & e_{1,3} &= 2cd \\
e_{0,3} &= 2bd & & \\
e_{0,4} &= d^2 & e_{2,2} &= c^2 + 2d^2
\end{aligned} \tag{6.9}
$$

A typical pattern of a square-shaped 2D operator of 9 points has three coefficients

that can be expressed as $c_{0,0} = a$, $c_{0,1} = b$, and $c_{1,1} = c$. For an example of kernel convolution of a 2D operator of 9 points that is star-shaped rather than square, refer to Figure 9 and the related chapter. The original square operator can be obtained from a 13-point operator by nullifying the d coefficients of Figure 27. Therefore the convolved coefficients of RIS=2 can be obtained from the coefficients in Equations 6.9 by dropping the terms dependent on d, perhaps excluding convolved coefficients altogether, namely, X, Z, and M. The convolved coefficients of the resulting 2D 25-point operator can be represented as in Equations 6.10.

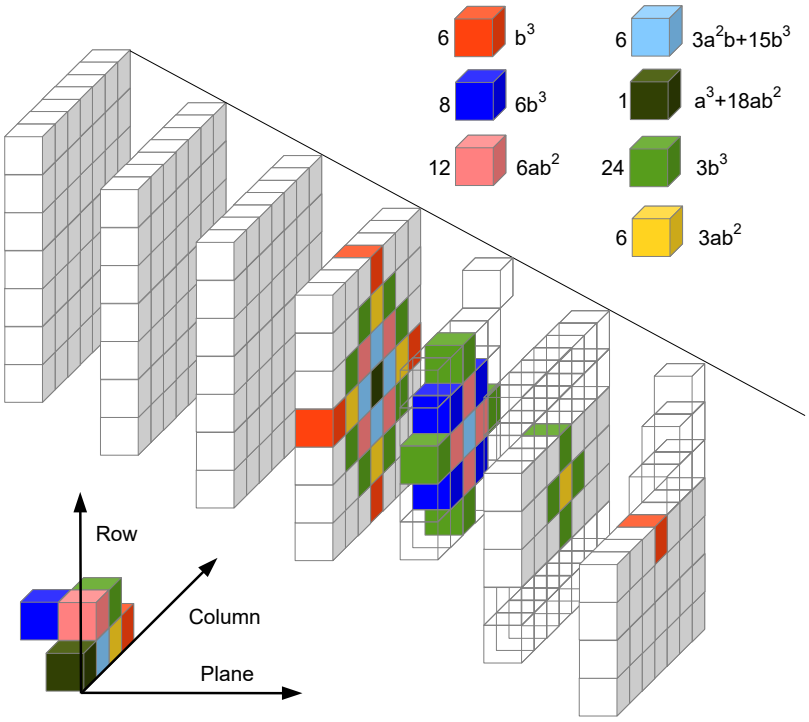
$$\begin{aligned} e_{0,0} &= a^2 + 4b^2 + 4c^2 & e_{1,1} &= 2ac + 2b^2 \\ e_{0,1} &= 2ab + 4bc & e_{1,2} &= 2bc \\ e_{0,2} &= b^2 + 2c^2 & e_{2,2} &= c^2 + 2d^2 \end{aligned} \tag{6.10}$$

6.9.3 Coefficients for 3D 7-point, RIS = 3

Figure 28 depicts the third convolution degree of the traditional 3D 7-point stencil. The resulting operator O^3 contains 63 points and is shown split into 7 slices or planes. For the sake of having a less crowded image, the coefficients in the three rear planes were left unmarked, whereas in the front three planes every coefficient is marked, but some neighboring positions were either left transparent or altogether removed.

The 63 coefficients in O^3 are of 7 types, as expressed in Equations 6.11. The indexes of the coefficients in the equations follow the pattern $e_{\text{plane, row, column}}$, with plane, row, and column numbers increasing as depicted in Figure 28. Note that the compass on the bottom left of the figure uses coefficient $e_{-1,1,1}$ (blue) for depiction purposes, but the equations actually show how to compute the equivalent $e_{1,1,1}$. Figure 28 also indicates how many of each coefficient there are in total. This operator is not further studied here, but a particular implementation could contain 24 ADDs, 3 MULs, and 11 FMADDs, resulting in an exploitation factor of 64.47%. For comparison, the original

Figure 28: Coefficients of O^3 for an original symmetric 5-point 3D stencil. O^3 contains 63 points with 7 distinct coefficients.



Source: Author.

operator O^1 has exploitation factor 60%.

$$
\begin{aligned}
e_{0,0,0} &= a^3 + 18ab^2 & e_{0,0,3} &= b^3 & e_{1,1,1} &= 6b^3 \\
e_{0,0,1} &= 3a^2b + 15b^3 & e_{0,1,1} &= 6ab^2 & & \\
e_{0,0,2} &= 3ab^2 & e_{0,1,2} &= 3b^3 & & \qquad (6.11)
\end{aligned}
$$

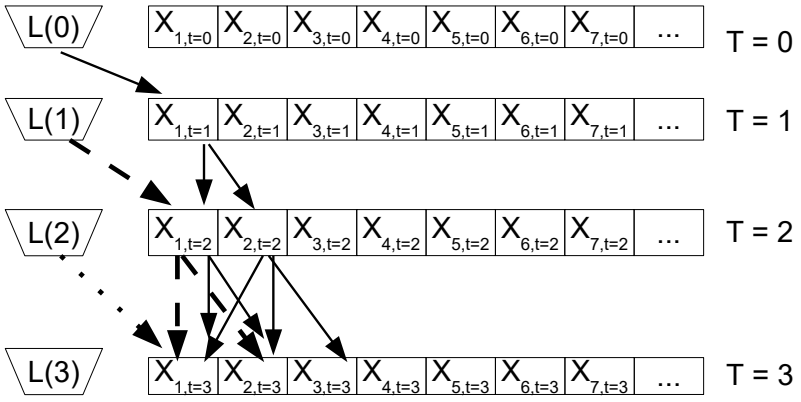
6.10 Dealing with Boundary Conditions

With the exception of this Section 6.10 the present work does not evaluate boundary conditions. Because of how the approach of kernel convolution acts, there are many further topics that could have been posed and discussed. Hence the impact of kernel convolution on boundary points is dealt with here only but enough to empower developers and researchers in the stencil community on how to tackle their specific

scenarios.

Figure 29 depicts the points of a one-dimensional domain evolving from the initial condition ($t = 0$) up to $t = 3$, with the leftmost point being X_1, and the base update equation $X_i \leftarrow aX_i + b(X_{i-1} + X_{i+1})$. Therefore the kernels O^1, O^2, and O^3 can be somewhat studied from the figure. The domain is subjected to a border condition on the left border given by the function $L(t)$. Oftentimes in studies of stencil performance L is assumed to be constant through time, sometimes even null. With different arrow types, Figure 29 shows up to what points the value of a past $L(t)$ influences the domain at time $t+3$.

Figure 29: Analyzing how the border conditions given by L(t) influence the domain points, and potentially O^2 and O^3.



Source: Author.

It is apparent that at time 3 the values $L(0)$ and $L(1)$ have influenced multiple domain points, whereas $L(2)$ has, just as expected from a state-of-the-art stencil implementation, influenced solely $X_{1,t=3}$. This indicates that the expressions for a few points should perhaps be slightly modified from the regular expressions in O^2 and O^3. Accordingly, in the base expression of O^3 there is no direct problem input from times 1 nor 2, only inputs from the initial condition. Neither relies the base expression previously derived for O^2 on inputs from time 1. Further evidences arise if the influence paths are taken into account. For example, at first glance the influence paths from $X_{1,t=0}$ to $X_{1,t=2}$, therefore the paths used to determine $e_{\Delta=2,0}$, could be $\{\leftarrow,\rightarrow\}$,$\{\rightarrow,\leftarrow\}$,

$\{\circlearrowleft, \circlearrowleft\}$. Alas, $\{\leftarrow, \rightarrow\}$ is not a valid influence path in this case, for when the related influence wavefront reaches the left border, it no longer propagates proportionally to the original wavefront source $X_{1,t=0}$, but rather proportionally to $L(1)$, as if the whole weight of the influence path had been reset. Similarly, the influence path $\{\leftarrow, \rightarrow, \rightarrow\}$ is not a valid path from $X_{1,t=0}$ to $X_{2,t=3}$ and should not take part in the expression of $e_{\Delta=3,1}$ when O^3 is applied to points close enough to the border.

A possible way to determine the convolved coefficients of points close to the border is with the aid of the automatized literal arithmetic approach discussed in Section 6.8.4. In this case, the leftmost point of the array can be used to represent the border values at different time steps. To implement this method, suffice it to change the name of the leftmost element of the array to "L(t)", with the appropriate number of the iteration in lieu of "t", instead of perpetually leaving the name as "X0". With this approach, the name of the leftmost point becomes "X1", instead of "X0", which now ceases to exist. The output for the first four points after two time iterations then becomes ...

X1={ I1 ∗(a^2+b^2)+L0 ∗(a∗b)+ I2 ∗(2∗ a∗b)+L1 ∗(b)+ I3 ∗(b^2) }

X2={ I2 ∗(a^2+2∗b^2)+ I1 ∗(2∗ a∗b)+ I3 ∗(2∗ a∗b)+L0 ∗(b^2)+ I4 ∗(b^2) }

X3={ I3 ∗(a^2+2∗b^2)+ I2 ∗(2∗ a∗b)+ I4 ∗(2∗ a∗b)+ I1 ∗(b^2)+ I5 ∗(b^2) }

X4={ I4 ∗(a^2+2∗b^2)+ I3 ∗(2∗ a∗b)+ I5 ∗(2∗ a∗b)+ I2 ∗(b^2)+ I6 ∗(b^2) }

...indicating that $X_{3,t+2}$ and $X_{4,t+2}$ have expressions of same pattern, and therefore their O^2 formula is not affected by the border condition. It also indicates that the expression of $X_{2,t+2}$ retained the base convolved coefficients of O^2, as defined in the set of Equations 6.4, and that it depends on the value of the left border at time 0. The most affected expression is that of $X_{1,t+2}$, depending on the value of the left border at both times 0 and 1, in addition to having all three convolved coefficients altered, to wit, e_0 becomes $e'_0 = a^2 + b^2$, e_1 unfolds into two coefficients e_1 and $e'_1 = ab$, and e_2 unfolds into e_2 and $e'_2 = b$. Thus Equation 6.12 reflects the complete representation of O^2, where the update equations of the points close to the borders are also represen-

ted. Equation 6.12 considers a domain of N points named from X_1 to X_N, under the boundary conditions L(t) and R(t) on the left and right borders.

$$O^2 : X_{i,t+2} \leftarrow \begin{cases} e'_0 X_{i,t} + e_1 X_{i+1,t} + e'_1 L(t) + e_2 X_{i+2,t} + e'_2 L(t+1) & \text{if } i = 1 \\ e_0 X_{i,t} + e_1 (X_{i-1,t} + X_{i+1,t}) + e_2 (L(t) + X_{i+2,t}) & \text{if } i = 2 \\ e_0 X_{i,t} + e_1 (X_{i-1,t} + X_{i+1,t}) + e_2 (X_{i-2,t} + X_{i+2,t}) & \text{if } 3 \leq i \leq N-3 \\ e_0 X_{i,t} + e_1 (X_{i-1,t} + X_{i+1,t}) + e_2 (R(t) + X_{i-2,t}) & \text{if } i = N-1 \\ e'_0 X_{i,t} + e_1 X_{i-1,t} + e'_1 R(t) + e_2 X_{i-2,t} + e'_2 R(t+1) & \text{if } i = N \end{cases} \tag{6.12}$$

The necessary FLOPs f and exploitation factor η consider that the same update equation is applied to every point. But Equation 6.12 shows that the expressions for the two points X_1 and X_N have different patterns than the other points, so that f and η as defined earlier are approximations. The equations for X_1 and X_N have 9 necessary FLOPs, whereas the equation of all the other points has 7 necessary FLOPs. But the equations for X_1 and X_N have exploitation factor 90%, whereas the others have 70%. While the ratio between the f's points out that the performance approximation might overestimate the speed of calculation of X_1 and X_N, the ratio between the exploitation factors points out that the approximation underestimates the speed. The key to solve this paradox is to realize that both patterns of expression require the same amount of 5 floating point instructions (in case the architecture contains fused multiply-add), so that, all other things being the same, both patterns execute at the same speed! This points out that the approximation is a good one for this type of stencil kernel. Additionally, this variation happens only for a few points (in the example, only to two points), whereas the domain is commonly larger than two by multiple orders of magnitude.

With increasing convolution degree, more points than just X_1 and X_N require their own set of adapted convolved coefficients. Degree Δ implies that the $\Delta - 1$ points from X_1 to $X_{\Delta-1}$ require adapted coefficients on the left extremity of the domain, with a similar reasoning for the right side. For the points in that range on the left border, the

i-th point requires $1+2(\Delta-i)$ adapted coefficients, totalling Δ^2-1 extra coefficients per boundary side. If the border is constant, then various of the extra coefficients can be merged, and X_i of O^Δ (again, for the points with $1 \leq i \leq \Delta-1$) requires $\Delta-i+1$ adapted coefficients, bringing the total of extra coefficients down to $\frac{\Delta(\Delta+1)}{2}-1$ per boundary side.

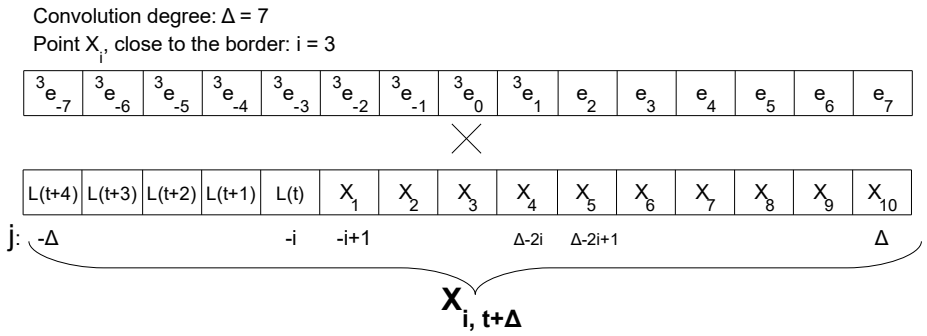
The facts that the altered coefficients of those points are not symmetric, and that they are point-dependent, varying from point to point, suggest a slight change in the nomenclature of the border coefficients. Let the $2\Delta+1$ coefficients of the update equation of of X_i of O^Δ be denoted by ${}^ie_j$, $j \leq \|\Delta\|, 1 \leq i \leq \Delta-1$. If ${}^ie_j = e_{\|j\|}$, the coefficient is not an altered one, and can simply be written as a regular e_j. This simplification is used in Equation 6.13. The equation shows how to update the domain points close to the left border by means of the border conditions and altered convolved coefficients, as well as by means of the common convolved coefficients and of the neighboring points.

$$X_{i,t+\Delta} \leftarrow \overbrace{\underbrace{\sum_{j=-\Delta}^{-i} {}^ie_j L(t-i+j)}_{1+\Delta-i} + \underbrace{\sum_{j=-i+1}^{\Delta-2i} {}^ie_j X_{i+j,t}}_{\Delta-i} + \underbrace{\sum_{j=\Delta-2i+1}^{\Delta} e_j X_{j,t}}_{2i}}^{2\Delta+1} \tag{6.13}$$

If $i=\Delta$ Equation 6.13 collapses to $X_{i,t+\Delta} \leftarrow {}^\Delta e_{-\Delta} L(t) + \sum_{j=-\Delta+1}^{\Delta} e_j X_{j,t}$. The would be adapted coefficient ${}^\Delta e_{-\Delta}$ corresponds in value to e_Δ. Thus, although the notation of the equation is helpful and accurate for this domain point, ${}^\Delta e_{-\Delta}$ actually is not a border adapted coefficient. ${}^\Delta e_{-\Delta} = e_\Delta$ is the only non-adapted convolved coefficient that participates with the border function ($L(t)$) in the equation of a point. Figure 30 represents Equation 6.13 being used to update the value of X_3 with the operator O^7. The limits of the sums of the equation are identified in the figure.

The sets of Equations 6.14 to 6.18 show the convolved coefficients that should be adapted to work with the points close to the left border of operators O^7, $O^6, \ldots, O^3$,

Figure 30: Combinations of adapted and non-adapted convolved coefficients interacting with values of the border function and neighboring points.



Source: Author.

whereas O^2 has been exposed at length with Equation 6.12. In each equation set, alignment was employed whenever possible to make evident that the aligned coefficients should be applied to the same input point or border value. For example, the set of Equations 6.15 shows the adapted convolved coefficients for O^6, and the alignment of ${}^3e_{-4}$ with ${}^4e_{-5}$ and ${}^5e_{-6}$ indicates that these adapted coefficients should be applied to the same input, namely, $L(t+1)$.

$$
\begin{aligned}
&O^7\\
&{}^1e_{-7} = b\\
&{}^1e_{-6} = ab\\
&{}^1e_{-5} = a^2b + b^3\\
&{}^1e_{-4} = a^3b + 3ab^3\\
&{}^1e_{-3} = a^4b + 6a^2b^3 + 2b^5\\
&{}^1e_{-2} = a^5b + 10a^3b^3 + 10ab^5\\
&{}^1e_{-1} = a^6b + 15a^4b^3 + 30a^2b^5 + 5b^7\\
&{}^1e_0 = a^7 + 21a^5b^2 + 70a^3b^4 + 35ab^6\\
&{}^1e_1 = 7a^6b + 70a^4b^3 + 105a^2b^5 + 14b^7\\
&{}^1e_2 = 21a^5b^2 + 105a^3b^4 + 63ab^6\\
&{}^1e_3 = 35a^4b^3 + 84a^2b^5 + 14b^7\\
&{}^1e_4 = 35a^3b^4 + 35ab^6\\
&{}^1e_5 = 21a^2b^5 + 6b^7
\end{aligned}
\qquad
\begin{aligned}
&{}^2e_{-7} = b^2\\
&{}^2e_{-6} = 2ab^2\\
&{}^2e_{-5} = 3a^2b^2 + 2b^4\\
&{}^2e_{-4} = 4a^3b^2 + 8ab^4\\
&{}^2e_{-3} = 5a^4b^2 + 20a^2b^4 + 5b^6\\
&{}^2e_{-2} = 6a^5b^2 + 40a^3b^4 + 30ab^6\\
&{}^2e_{-1} = 7a^6b + 70a^4b^3 + 105a^2b^5 + 14b^7\\
&{}^2e_0 = a^7 + 42a^5b^2 + 175a^3b^4 + 98ab^6\\
&{}^2e_1 = 7a^6b + 105a^4b^3 + 189a^2b^5 + 28b^7\\
&{}^2e_2 = 21a^5b^2 + 140a^3b^4 + 98ab^6\\
&{}^2e_3 = 35a^4b^3 + 105a^2b^5 + 20b^7
\end{aligned}
$$

$$
\begin{aligned}
&{}^3e_{-7} = b^3\\
&{}^3e_{-6} = 3ab^3\\
&{}^3e_{-5} = 6a^2b^3 + 3b^5\\
&{}^3e_{-4} = 10a^3b^3 + 15ab^5\\
&{}^3e_{-3} = 15a^4b^3 + 45a^2b^5 + 9b^7\\
&{}^3e_{-2} = 21a^5b^2 + 105a^3b^4 + 63ab^6\\
&{}^3e_{-1} = 7a^6b + 105a^4b^3 + 189a^2b^5 + 28b^7\\
&{}^3e_0 = a^7 + 42a^5b^2 + 210a^3b^4 + 133ab^6\\
&{}^3e_1 = 7a^6b + 105a^4b^3 + 210a^2b^5 + 34b^7
\end{aligned}
\qquad
\begin{aligned}
&{}^4e_{-7} = b^4\\
&{}^4e_{-6} = 4ab^4\\
&{}^4e_{-5} = 10a^2b^4 + 4b^6\\
&{}^4e_{-4} = 20a^3b^4 + 24ab^6\\
&{}^4e_{-3} = 35a^4b^3 + 84a^2b^5 + 14b^7\\
&{}^4e_{-2} = 21a^5b^2 + 140a^3b^4 + 98ab^6\\
&{}^4e_{-1} = 7a^6b + 105a^4b^3 + 210a^2b^5 + 34b^7
\end{aligned}
$$

$$
\begin{aligned}
&{}^5e_{-7} = b^5\\
&{}^5e_{-6} = 5ab^5\\
&{}^5e_{-5} = 15a^2b^5 + 5b^7\\
&{}^5e_{-4} = 35a^3b^4 + 35ab^6\\
&{}^5e_{-3} = 35a^4b^3 + 105a^2b^5 + 20b^7
\end{aligned}
\qquad
\begin{aligned}
&{}^6e_{-7} = b^6\\
&{}^6e_{-6} = 6ab^6\\
&{}^6e_{-5} = 21a^2b^5 + 6b^7
\end{aligned}
\tag{6.14}
$$

O^6

$${}^1e_{-6} = b$$

$${}^1e_{-5} = ab \qquad {}^2e_{-6} = b^2$$

$${}^1e_{-4} = a^2b + b^3 \qquad {}^2e_{-5} = 2ab^2$$

$${}^1e_{-3} = a^3b + 3ab^3 \qquad {}^2e_{-4} = 3a^2b^2 + 2b^4$$

$${}^1e_{-2} = a^4b + 6a^2b^3 + 2b^5 \qquad {}^2e_{-3} = 4a^3b^2 + 8ab^4$$

$${}^1e_{-1} = a^5b + 10a^3b^3 + 10ab^5 \qquad {}^2e_{-2} = 5a^4b^2 + 20a^2b^4 + 5b^6$$

$${}^1e_0 = a^6 + 15a^4b^2 + 30a^2b^4 + 5b^6 \qquad {}^2e_{-1} = 6a^5b + 40a^3b^3 + 30ab^5$$

$${}^1e_1 = 6a^5b + 40a^3b^3 + 30ab^5 \qquad {}^2e_0 = a^6 + 30a^4b^2 + 75a^2b^4 + 14b^6$$

$${}^1e_2 = 15a^4b^2 + 45a^2b^4 + 9b^6 \qquad {}^2e_1 = 6a^5b + 60a^3b^3 + 54ab^5$$

$${}^1e_3 = 20a^3b^3 + 24ab^5 \qquad {}^2e_2 = 15a^4b^2 + 60a^2b^4 + 14b^6$$

$${}^1e_4 = 15a^2b^4 + 5b^6$$

$${}^3e_{-6} = b^3$$

$${}^3e_{-5} = 3ab^3 \qquad {}^4e_{-6} = b^4$$

$${}^3e_{-4} = 6a^2b^3 + 3b^5 \qquad {}^4e_{-5} = 4ab^4 \qquad {}^5e_{-6} = b^5$$

$${}^3e_{-3} = 10a^3b^3 + 15ab^5 \qquad {}^4e_{-4} = 10a^2b^4 + 4b^6 \qquad {}^5e_{-5} = 5ab^5$$

$${}^3e_{-2} = 15a^4b^2 + 45a^2b^4 + 9b^6 \qquad {}^4e_{-3} = 20a^3b^3 + 24ab^5 \qquad {}^5e_{-4} = 15a^2b^4 + 5b^6$$

$${}^3e_{-1} = 6a^5b + 60a^3b^3 + 54ab^5 \qquad {}^4e_{-2} = 15a^4b^2 + 60a^2b^4 + 14b^6$$

$${}^3e_0 = a^6 + 30a^4b^2 + 90a^2b^4 + 19b^6 \tag{6.15}$$

$$
\begin{aligned}
&O^5\\
&{}^1e_{-5} = b\\
&{}^1e_{-4} = ab && {}^2e_{-5} = b^2\\
&{}^1e_{-3} = a^2b + b^3 && {}^2e_{-4} = 2ab^2\\
&{}^1e_{-2} = a^3b + 3ab^3 && {}^2e_{-3} = 3a^2b^2 + 2b^4\\
&{}^1e_{-1} = a^4b + 6a^2b^3 + 2b^5 && {}^2e_{-2} = 4a^3b^2 + 8ab^4\\
&{}^1e_0 = a^5 + 10a^3b^2 + 10ab^4 && {}^2e_{-1} = 5a^4b + 20a^2b^3 + 5b^5\\
&{}^1e_1 = 5a^4b + 20a^2b^3 + 5b^5 && {}^2e_0 = a^5 + 20a^3b^2 + 25ab^4\\
&{}^1e_2 = 10a^3b^2 + 15ab^4 && {}^2e_1 = 5a^4b + 30a^2b^3 + 9b^5\\
&{}^1e_3 = 10a^2b^3 + 4b^5\\
\\
&{}^3e_{-5} = b^3\\
&{}^3e_{-4} = 3ab^3 && {}^4e_{-5} = b^4\\
&{}^3e_{-3} = 6a^2b^3 + 3b^5 && {}^4e_{-4} = 4ab^4\\
&{}^3e_{-2} = 10a^3b^2 + 15ab^4 && {}^4e_{-3} = 10a^2b^3 + 4b^5\\
&{}^3e_{-1} = 5a^4b + 30a^2b^3 + 9b^5
\end{aligned}
\tag{6.16}
$$

$$
\begin{aligned}
&O^4\\
&{}^1e_{-4} = b\\
&{}^1e_{-3} = ab && {}^2e_{-4} = b^2\\
&{}^1e_{-2} = a^2b + b^3 && {}^2e_{-3} = 2ab^2 && {}^3e_{-4} = b^3\\
&{}^1e_{-1} = a^3b + 3ab^3 && {}^2e_{-2} = 3a^2b^2 + 2b^4 && {}^3e_{-3} = 3ab^3\\
&{}^1e_0 = a^4 + 6a^2b^2 + 2b^4 && {}^2e_{-1} = 4a^3b + 8ab^3 && {}^3e_{-2} = 6a^2b^2 + 3b^4\\
&{}^1e_1 = 4a^3b + 8ab^3 && {}^2e_0 = a^4 + 12a^2b^2 + 5b^4\\
&{}^1e_2 = 6a^2b^2 + 3b^4
\end{aligned}
\tag{6.17}
$$

$$
\begin{aligned}
&O^3\\
&{}^1e_{-3} = b\\
&{}^1e_{-2} = ab && {}^2e_{-3} = b^2\\
&{}^1e_{-1} = a^2b + b^3 && {}^2e_{-2} = 2ab^2\\
&{}^1e_0 = a^3 + 3ab^2 && {}^2e_{-1} = 3a^2b + 2b^3\\
&{}^1e_1 = 3a^2b + 2b^3
\end{aligned}
\tag{6.18}
$$

7 EXPERIMENTS AND RESULTS - 1D

Next, the concepts described above are exercised with multiple experiments varying the symmetry and size of 1D stencils. The metrics employed the most are EGFLOPS and EFLOCC (Definition 7), both expressed in Section 5.3. Some experiments have their running time also reported in Appendix B.

7.1 Platform

The experiments were run on a Power8 node of two chips of 12 cores each. Whereas each core can run with up to 8 hardware threads with IBM's SMT technology (192 in the whole node), the experiments run with 1 thread per core. The peak memory bandwidth per node is 300GB/s to read and 150GB/s to write. Each core has 2 VSX units and 64 vector registers. Each VSX unit has a width of 2 double precision points and is equipped with fused multiply-add instructions, for the capacity of 8 double precision FLOPs per core per cycle. Each core has private memory cache L1 and L2 of 64KB and 512KB. The L3 size per core is impressive 8MB. The node has a total of 4 NUMA domains and the measured bandwidth with STREAM SCALE is 286.2GB/s. The write-allocate cache policy causes the extra traffic of 8 bytes per double precision point written and read. Therefore, the maximum FLOPS in memory-bound scenarios is given by the following variation of Equation 5.1:

$$\mu = \frac{fB}{3s} = \frac{f \times 286.2}{24} (\text{GFLOPS}) \tag{7.1}$$

Table 7: Peak EGFLOPS achievable in memory-bound scenarios by a 1D-3pt stencil code. See Section 7.1 for platform description.

RIS	1	2	3	4	5	6
Peak EGFLOPS	47.7	95.4	143.1	190.8	238.5	286.18

In Equation 7.1, f is the amount of operations per application of the stencil's update equation. Table 7 shows the peak equivalent performance achievable for different kernel convolved forms of a symmetric 1D-3point stencil.

7.2 Codes

The loop iterations through the domain points are unrolled 8 times to help with instruction level parallelism and the compiler's ability to avoid unnecessary memory loads. Although Power8 has inexpensive unaligned vector load, the codes apply data layout transformation to make the code more portable to other dimensions and architectures. The codes ignore boundary conditions on the relatively insignificant two borders of the large domain. To show how Kernel Convolution leverages time-blocking approaches, the simplistic boxed time-blocking strategy described in Section 2.4 is implemented. The strategy has two main features: (1) the threads synchronize after every TB iterations, where TB is defined as the *time-blocking degree*, and (2) each thread divides its domain share into NB consecutive blocks, processing the TB time-iterations of each block before proceeding to the next block. The way the code deals with the borders of (2) actually incurs redundant computation in each block's borders. As each thread's NB block is very large, the redundant extra work is inexpressive. Table 8 shows how the boxed time-blocking strategy affects a symmetric 1D-3point operator. Note how the performance of the simple kernel without time-blocking (TB=1) is close to the memory-bound roofline of Table 7 (46.8 GFLOPS versus 47.7 GFLOPS), and how the performance could be tripled with time-blocking. For more details about the sizes, configuration, and runtimes of symmetric experiments please refer to Tables 19

Table 8: Effect of the boxed time-blocking strategy on a simple operator (in 24 cores), when ASLI is not applied.

Time-blocking degree (TB)	1	2	4	5	10
EGFLOPS	46.8	81.7	123	132	151
EFLOCC	0.651	1.13	1.7	1.84	2.1

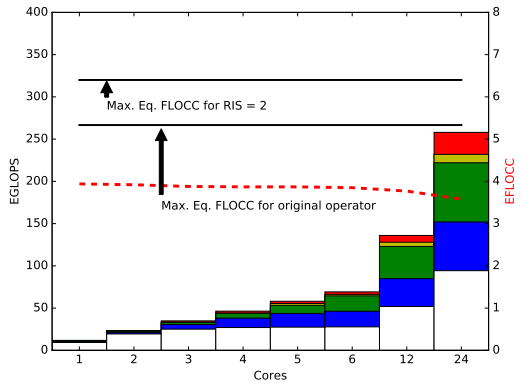
through 21 in Appendix B.

7.3 Convolution of 3-Point with RIS of 2, 3, 4, 5, 6

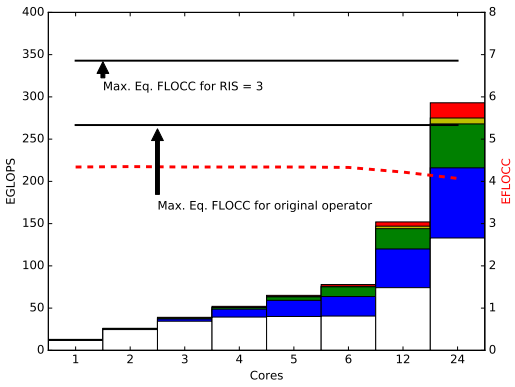
Figure 31 shows weak scaling results for kernel convolved stencils with convolution degrees (or RIS, Reduction in Iteration Space) of 2, 3, 4, 5, and 6 for the 1D-3point symmetric stencil. The problem size was 3072000 double precision points **per core**. Considering the two domain arrays of 24000KB each, the problem amounts to approx. $\times$750 L1, $\times$94 L2, and $\times$5.9 L3 of each core. The kernel convolved versions compute the work of 6000 iterations of a simple stencil. The time-blocking parameters are NB=50 blocks per core, and degree (TB) 1, 2, 4, 5, and 10 in each plot. The lower solid lines in each plot is fixed at 5.33, showing the maximum equivalent floating point operations per core per cycle (EFLOCC) attainable by a simple, state-of-the-art, stencil pattern. The upper solid lines show the maximum EFLOCC theoretically attainable by the convolved version, which is $5.33 \times \phi_{\text{RIS}} \times \eta_{\text{RIS}}$ (see Table 4 for values of ϕ and η). The dashed red lines show the experimental EFLOCC attained when the kernel convolution is applied along with the boxed time-blocking strategy with degree TB=10.

When the 24 cores in the 4 NUMA domains are used, the kernel convolved code with RIS=2 (Figure 31a) and TB=2 performs better than the best result achieved by the simple kernel with any tried degree of time-blocking (Table 8). The version with RIS=2 and TB=10 reaches 3.58 EFLOCC, or $3.58/5.33 = 67\%$ equivalent utilization. This means that a simple stencil would need to execute at 67% of the FPU utilization to have the same time to solution. The 3.58 EFLOCC achieved with a convolution of just

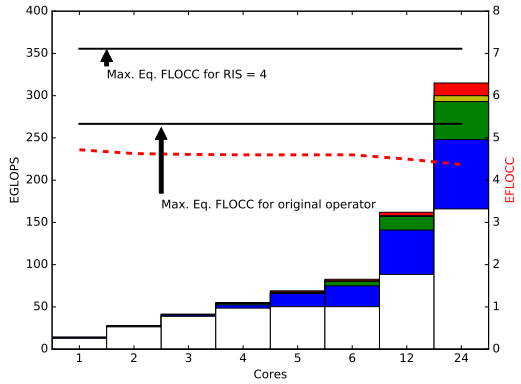
Figure 31: Weak scaling of a symmetric 1D-3point stencil implemented with varying degrees of kernel convolution (RIS).



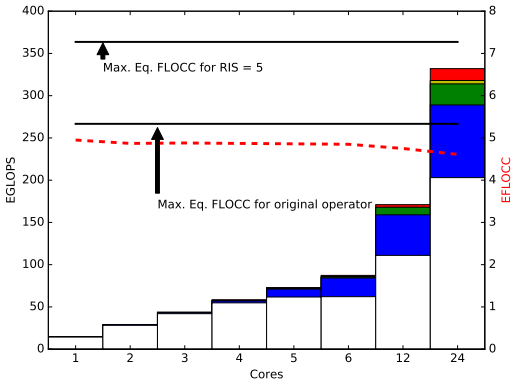
(a) RIS = 2



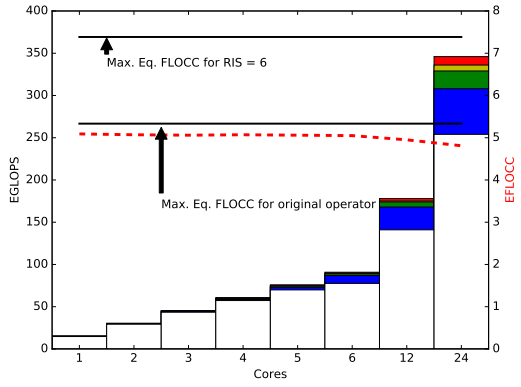
(b) RIS = 3



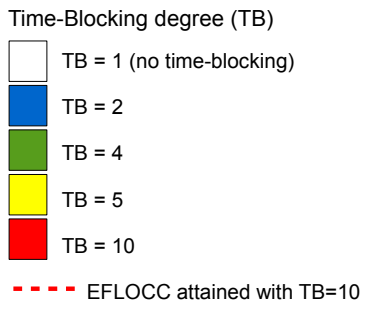
(c) RIS = 4



(d) RIS = 5



(e) RIS = 6



Source: Author.

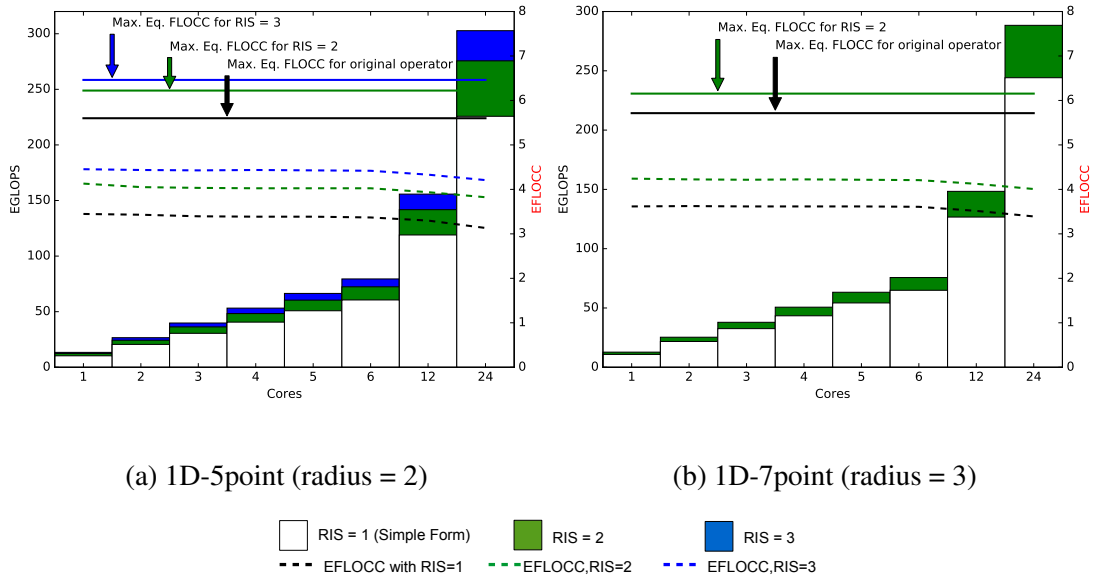
2 is already higher than the 3.09 EFLOCC reached by previous work (HENRETTY *et al.*, 2013). Notwithstanding the machines differing much (they use 4 cores of an Intel i7-2600K) and their domain per core being much smaller, their result is reported as a reference value for the reader. Another trend observable across all the graphs (Figures 31, 32, and 33) is a small decrease in the in-core performance when the algorithm scales from 6 to 12 and then 24 cores. The implementation of the 1D-3point stencil with convolution degree RIS=6 (Figure 31e) achieves 4.81 EFLOCC or 90% equivalent utilization when the 24 cores are used, for impressive *346 EGFLOPS in a single node*. The equivalent utilization is 94.7% if six cores (half of a chip) are used, for 90.8 EGLFOPS. When the two NUMA domains and 12 cores of a single chip are used, these figures go to 92.8% and 178 EGFLOPS.

As seen in the graphs, the convolved kernels perform around 2.5 EFLOCC below their compute-bound roofline (topmost black solid lines). The version with RIS=6 and no time-blocking at all reaches the same EGFLOPS as a version with RIS=2 and time-blocking of TB=10, which is $\approx$ 250 EGFLOPS. This is a better performance than the performance of TB=10 applied to the simple stencil operator, which has no kernel convolution. As a result, the time-blocking version of RIS=6 benefits from a much better baseline when interacting with other techniques. Another common feature among the graphs of Figure 31 is that the codes with no time-blocking (TB=1) perform close to their memory-bound roofline (cf. Table 7 with the results when the 24 cores are used). This way, one can see that *Kernel Convolution shifts the roofline and helps with the memory interaction*.

7.4 Convolution of 5-Point and 7-Point Stencils

The kernel convolution of originally larger stencils is analyzed while exemplifying the application of inherent gain (Definition 10). Figure 32 compares the 1D-5point and the 1D-7point stencils with their respective convolved forms. Every experiment

Figure 32: Weak scaling of two symmetric stencil operators and convolved forms.



(a) 1D-5point (radius = 2)

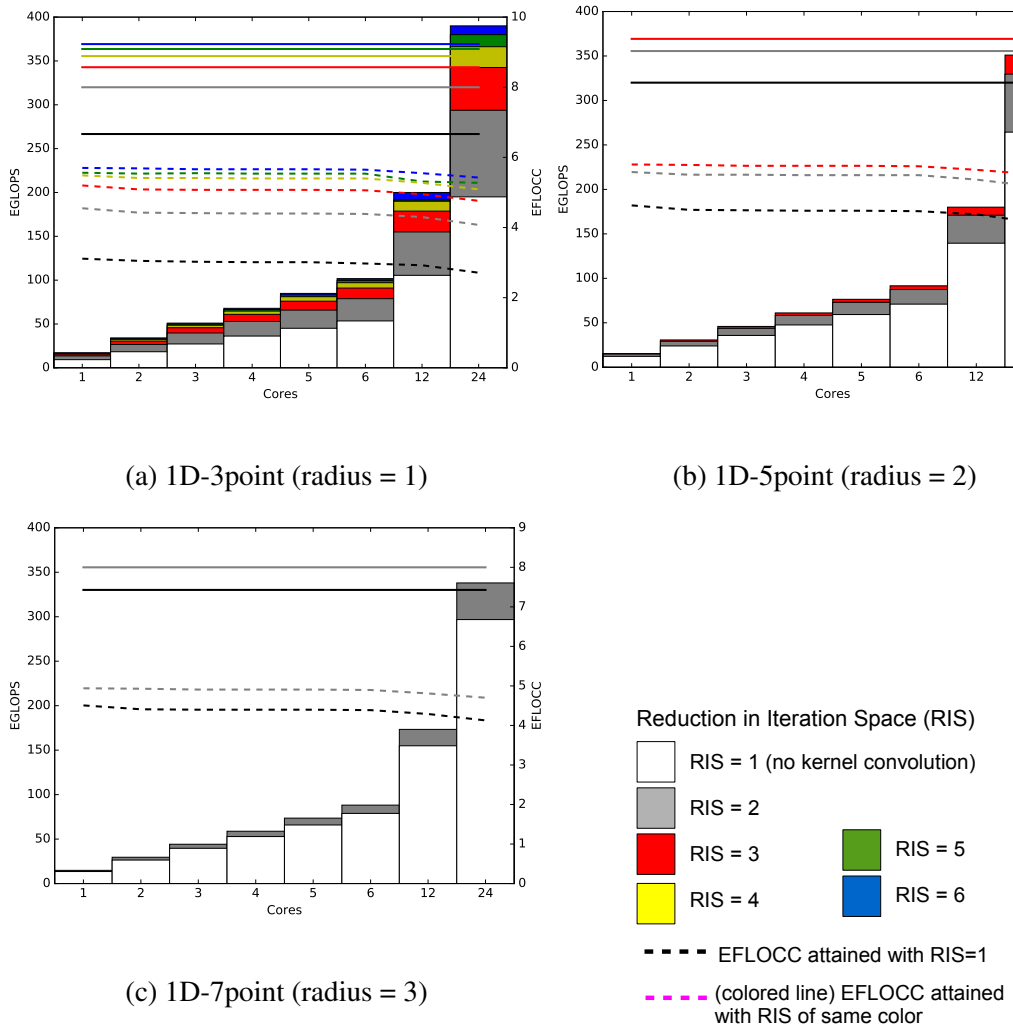
(b) 1D-7point (radius = 3)

Source: Author.

in the figure used a time-blocking degree of TB $= 10$. The 1D-5point stencil was implemented with kernel convolution of 1, 2, and 3, always solving the work of 3000 iterations of the simple stencil. The 1D-7point stencil was implemented with kernel convolution of 1, and 2, solving the work of 2000 iterations of the simple stencil. The other domain and time-blocking configurations are the same as in Figure 31.

When the 1D-5point stencil (radius 2) runs in the 24 cores, the kernels O^1, O^2, and O^3 achieve 3.13, 3.82, and 4.21 EFLOCC. Therefore, the speed-ups among successive kernel versions are $S_{1\to2} = 1.220$ and $S_{2\to3} = 1.102$. These numbers are higher than the respective *inherent gains* from Table 4 (1.111 and 1.038). This means that the kernel convolution resulted in a relative enhancement in the memory subsystem interaction and register dependencies. This is a remarkable result, showing that, even after applying time-blocking, the originally compute-bound O^1 and O^2 can still have the memory interaction enhanced, provided Kernel Convolution is applied. Finally, in accordance to the expectations of Section 5.7.3, it holds that $\frac{\mathscr{S}_{1\to2}}{\xi_{1\to2}} = 1.099 > \frac{\mathscr{S}_{2\to3}}{\xi_{2\to3}} = 1.062$. In other words, in grounds of smaller or no convolution there are more inefficiencies, which the kernel convolution also helps to allay.

Figure 33: Weak scaling of **asymmetric** operators and their convolved forms.



(a) 1D-3point (radius = 1)

(b) 1D-5point (radius = 2)

(c) 1D-7point (radius = 3)

Source: Author.

When the original stencil has radius 3 (1D-7point), time-blocking helps the code to achieve 3.39 EFLOCC (Figure 32). The kernel with convolution RIS $= 2$ reaches 4 EFLOCC, for a speed-up $S_{1\to2} = 1.18 > \xi_{1\to2} = 1.077$ (see Table 4 for ξ). Therefore, here also the convolution enhanced the interactions with the memory, even though the original code was already performing well because of the time-blocking. It also means that the convolved kernel gets closer to its theoretical limit than the simple kernel approaches its limit.

7.5 Asymmetric Operators

Figure 33 reports results for kernel convolved forms of originally asymmetric stencils of radius 1, 2, and 3. The solid colored lines mark the peak equivalent floating point per core per cycle (EFLOCC, in the right axis) of the convolved kernel versions. The reported experiments have a time-blocking degree TB $= 10$, thus being optimized regarding memory access. For the 1D-3point, kernel convolution degrees (RIS) of 1, 2, 3, 4, 5, and 6 were used, computing 6000 iterations of the simple form. For the 1D-5point stencil, RIS of 1, 2, and 3 were used, computing 3000 iterations of the simple form. The 1D-7point stencil used RIS of 1 and 2, computing 2000 iterations of the simple form. The other domain and time-blocking configurations are the same as in Figure 31. In IBM's Power8 architecture, the peak FLOCC is 8. Because of the multiply-add imbalance, note in the solid black lines of the three plots that the simple stencils cannot actually achieve 8 FLOCCs. However, as pointed out in Section 5.7.3.1, asymmetric 1D convolved kernels can theoretically reach *strong hyper-performance*. In other words, the kernels with RIS > 2 have theoretical maximum EFLOCC higher than 8. The convolved kernels reach high performance, e.g., O^3 reaches more than 5 EFLOCC and O^6 reaches 5.425 EFLOCC. Even though these kernels begin to approach the compute roofline of the simple form (6.67 EFLOCC) and perform better than comparable works in the pre-Kernel Convolution literature, they are still far from their respective compute roofline and more investigation is necessary to further enhance such kernels. The 5.425 EFLOCC of the asymmetric with RIS=6 is higher than the 4.81 EFLOCC of the symmetric of same RIS, as expected from the fact that asymmetric stencils have higher necessary FLOPs (Definition 5 in Section 5).

Table 9 shows the speed-ups experimentally obtained between kernels with consecutive degrees of convolution for the asymmetric 1D-3point stencil. The table also shows how the speed-ups diverge from the *inherent gain* (Section 5.7.3). As expected, it holds both that: the measured speed-up S between kernels with consecutive degrees

Table 9: Speed-ups and speed-up deviation from *inherent gain* for versions of asymmetric 1D-3pt stencil

RIS	2	3	4	5	6
$S_{(\text{RIS}-1)\rightarrow\text{RIS}}$	1.5	1.169	1.068	1.037	1.028
$\frac{S_{(\text{RIS}-1)\rightarrow\text{RIS}}}{\xi_{(\text{RIS}-1)\rightarrow\text{RIS}}}$	1.25	1.091	1.03	1.014	1.013

of convolution is higher than the related inherent gain ξ, and the relative difference between S and ξ decreases as the degree of convolution increases.

8 EXPERIMENTS AND RESULTS - 2D

This chapter describes the performance of 2D stencil codes optimized with kernel convolution. It will be shown that (a) kernel convolution allows a performance better than the one allowed by any space-blocking or time-blocking technique, and that (b) mixing these techniques with kernel convolution would bring no additional advantage for convolutions of high enough degree in the machines employed.

8.1 Platform and Codes

The codes used in the experiment do not implement any kind of space- or time-blocking. To exploit SIMDzation, data level transformation similar to (HENRETTY *et al.*, 2013) was applied. Boundary conditions for this experiments were left out, which in any case would not hurt the speed-ups achieved. To achieve good performance when $\Delta > 2$ or the original $R > 1$, it is paramount to add to the code both loop unrolling and a mechanism for more efficient index calculation. Thus, the codes benefit from the assembly instruction wrapper in Listing 8.1 to facilitate an indexed addressing with offset. For every value loaded when updating a given point, the function is invoked with the same *base* pointer.

Table 10 shows characterization properties of the 2D 5-point stencil used in the experiments, taking into account a machine with architectural support to fused multiply-add instructions. The FLOPs ratio and the exploitation factor are indicated with ϕ and η. The equivalent theoretical maximum compute rate achievable per core per cycle is

Listing 8.1: Wrapper to calculate indexes more efficiently.

```
inline vector double vec_ld_index
    (vector double* base, int offset){
        vector double vd;
        asm("lxvd2x %x0,%1,%2":"=v"(vd):
            "r" (base), "r"(offset) );
        return vd;}
```

Table 10: Characterization of a 5-point symmetric stencil operator and its convolutions of degrees 2 to 4.

Δ	Adds	Muls	Madds	Flops	η	ϕ	Max EFLOCC
1	3	1	1	6	0.60	1.00	4.80
2	5	1	4	14	0.70	0.86	4.80
3	8	2	7	24	0.71	0.75	4.24
4	11	4	11	37	0.71	0.65	3.69

also indicated in the table.

The platform used is the same of Section 7.1. Table 11 shows the peak equivalent compute rate achievable by the kernels in compute-bound and memory-bound scenarios, considering the write-allocate nature of the store instructions employed. The compute-bound limit of O^1 takes into account the machine's frequency of 3 GHz and $\eta_1 = 0.6$. Therefore, $345.6 = 2 \times 4 \times 3 \times 24 \times 0.6$ EGFLOPS. The compute-bound limit of O^Δ is then easily obtainable with Table 10 and χ_Δ of Equation 5.6. The memory-bound limit of the various convolved forms can be calculated by extending Equation 7.1 to embrace the *FLOPs ratio*:

$$\mu_\Delta = \phi_\Delta \times \frac{f_\Delta B}{3s}. \tag{8.1}$$

Then Table 11 can be completed by noticing that, in memory-bound grounds, the speed-up that O^Δ can get over O^1 is $\mu_\Delta/\mu_1 = \Delta$. This is an intuitive result, as one would expect the maximum speed-up in memory-bound cases to equal the convolution degree Δ, for this is the reduction factor of the domain sweeps.

Table 11: Peak EGFLOPS in memory-bound and compute-bound scenarios, for the 2D 5-point and convolved forms.

Convolution degree Δ:	1	2	3	4
Peak Memory-Bound:	71.5	143	214.5	286
Peak Compute-Bound (24 cores):	345.6	345.6	304.9	265.8

Table 12: Sizes (MB) for the (quadratic) weak scaling analysis. The problem size is two times the domain size, which is $(1024 \times \text{cores})^2$ elements.

Cores	1	2	3	4	5	6	12	24
Elements	1024^2	2048^2	3072^2	4096^2	5120^2	6144^2	12288^2	24576^2
Domain Size	8	32	72	128	200	288	1152	4608
Problem Size	16	64	144	256	400	576	2304	9216

8.2 The Symmetric 2D, 5-Point

The 2D, 5-point, as ubiquitous in literature as it is, was subjected to a more thorough analysis than larger 2D operators. The quadratic weak scaling experiment was of particular relevance due to its extensiveness in problem size.

8.2.1 Quadratic Weak Scaling

To begin the experimental analysis of kernel convolution in 2D stencil computations, Table 12 surmises the sizes employed for a quadratic weak scaling study. Having the domain size increase quadratically with the number of cores allows the analysis to span from a domain size of 8 MB, that is the amount of L3 per core, up to 4.5 GB. As the codes work with two copies of the domain, the total problem size reaches 9 GB on 24 cores, or 48$\times$ the total size of the L3 cache. IBM POWER8's stability helps with this wide span of sizes, as it causes the performance per core to remain nearly constant, provided enough balance between memory access and computation in the kernels.

Table 13 shows the quadratic weak scaling of the kernel convolution of degrees 1, 2, 3, and 4 of the canonical and symmetric 2D 5-point stencil operator. It is important

Table 13: Performance of the 2D 5-point stencil with various convolution degrees. No blocking strategy was used.

Size	Cores	EFLOCC				EGFLOPS			
		O^1	O^2	O^3	O^4	O^1	O^2	O^3	O^4
1024^2	1	1.14	1.67	1.92	1.97	3.42	5.01	5.75	5.92
2048^2	2	1.15	1.64	1.89	1.98	6.88	9.83	11.4	11.9
3072^2	3	1.04	1.66	1.92	2	9.4	15	17.3	18
4096^2	4	1.12	1.69	1.93	2	13.5	20.2	23.1	24
5120^2	5	1.1	1.66	1.94	2.02	16.5	24.9	29.1	30.2
6144^2	6	1.01	1.64	1.84	1.98	18.3	29.5	33.2	35.7
12288^2	12	1.02	1.61	1.87	1.95	36.7	57.9	67.3	70.2
24576^2	24	0.59	1.47	1.65	1.95	41.6	105	118	141

to note that no kind of blocking is applied to these kernels, in order to contrast the performance gains that come solely from different degrees of kernel convolution. The problem size was two times the domain size, which had $(1024 \times \text{cores})^2$ elements, and the codes solve 120 iterations of the original pattern. More details and the duration of each experiment can be found in Table 22 on Appendix B.

Interestingly, across the four NUMA domains and 24 cores of a single POWER8 node, the scalability achieved with **convolution degree 4** is $1.95/1.97 \approx 100\%$. If on the one hand the equivalent floating point operations per core per cycle (EFLOCC) across every experiment of **convolution 4** in Table 13 is around 2, or 25% of the machine's maximum capacity, on the other hand recall from Table 10 that the convolution of 4 sets the peak EFLOCC at 3.69, so that every experiment in this section actually approached 54% of the theoretical peak. The original kernel, of **convolution 1**, and the kernel with **convolution 2** reach 58.2% and 73.5% of the expected memory-bound roofline. Therefore, differently from O^4, they do not approach perfect scalability. These O^1 and O^2 are cases where *time-blocking* would help the related kernels to break the memory-bounded zone. Perhaps as interesting as convolution 4 was **convolution 3**, which with impressive 86% scalability reached 55% of the memory-bound peak.

8.2.2 Strong Scaling

Table 14 summarizes the performance of different kernel convolved versions of an originally 2D 5-point symmetric stencil. The figure presents the strong scaling of kernels with convolution degrees 1, 2, 3, and 4, when 1, 6, 12, and 24 cores are used to solve a domain of 24576×24576 points. In double precision, the two copies of the domain amount to 9 GB, or 48 times the L3 cache size. Therefore, and because time-blocking is not applied, the kernels are expected to be memory-bound. Accordingly, the performance of the original operator (convolution **degree 1**) and the operator with **convolution 2** reach 60% and 76% of their roofline predicted by Table 11. The kernel with **convolution Δ = 3** reaches 124 EGFLOPS, or 58% of its memory-bound roofline. This makes for an interesting speed-up of $2.97 \approx \Delta$ over the naive implementation (cf. discussion in Section 8.1). Recall from Section 8.2.1 that the **speed-of-light** of the naive implementation, even if implemented with advanced time-blocking and space-blocking approaches, is bound to 1.14 EFLOCC, or equivalent floating point operations per core per cycle. This is necessarily the case, as such performance was the best obtained for a domain compatible with the memory resources of single core (For example, in the quadratic weak scaling experiment the best performance of O^1 with 2 cores was 1.15 EFLOCC). Therefore, in the strong scaling experiment, *although not reaching its full potential, the 124 EGFLOPS or 1.72 EFLOCC reached by this particular implementation of* O^3 *is* $1.72/1.15 \approx 1.5\times$ *quicker than the speed-of-light, best case, of a 2D 5-point stencil implemented in accordance with the state-of-the-art*.

With **convolution 4**, the achieved 143 EGFLOPS is relatively far from the memory-bound roofline of 286 EGFLOPS. However, it was discussed in Section 8.2.1 that, for convolution degree 4, a compute rate of 141 EGFLOPS in 24 cores reflects a nearly perfect (quadratic) weak scaling departing from 1 core, where the total problem size was only two times the L3 cache of a single core. As this single core scenario reflects the **speed-of-light** of the implementation, the conclusion is that this type of

Table 14: Strong scaling: speed-up of convolved forms over the original operator and scalability between 1 and 24 cores.

Δ	Cores	EGFLOPS	EFLOCC	Scaling	Speed-up	Real FLOPS
1	1	1.86	0.62	N/A	N/A	1.86
1	6	10.9	0.61	0.98	N/A	10.9
1	12	21.6	0.60	0.97	N/A	21.6
1	24	42.2	0.59	0.95	N/A	42.2
2	1	4.84	1.61	N/A	2.60	5.63
2	6	28.3	1.57	0.97	2.60	32.91
2	12	55.9	1.55	0.96	2.59	65
2	24	109	1.51	0.94	2.58	126.74
3	1	5.33	1.78	N/A	2.87	7.11
3	6	32.1	1.78	1	2.94	42.8
3	12	63.2	1.76	0.99	2.93	84.27
3	24	124	1.72	0.97	2.94	165.33
4	1	6.11	2.04	N/A	3.28	9.4
4	6	36.5	2.03	1	3.35	56.15
4	12	72.6	2.02	0.99	3.36	111.69
4	24	143	1.99	0.98	3.39	220

kernel cannot actually achieve the memory-bound roofline, at least not with an implementation similar to the one employed. At the moment, it is not completely clear whether a machine code with better instruction interleaving, software pre-fetch, register allocation, etc. would be beneficial. This kind of concern comes into play just by means of the kernel convolution, and, again, the performance attained by the current implementation is already $1.99/1.15 = 1.73\times$ *quicker than the speed-of-light of state-of-the-art implementations that rely on time-blocking the original operator* O^1. *Also note that common sub-expression is not applicable to* O^1.

This experiment shows how the kernel convolution helps to alleviate the memory-boundedness of the original operator. Once again, the robustness of POWER8 helps one to focus on the new tasks of optimizing the transformation of kernel convolution, rendering space-blocking and time-blocking not necessary when $\Delta = 4$, and offers

good scalability with increasing cores.

In addition to the equivalent compute rate, measured in EGFLOPS, it is valuable to asses the actual compute rate executed by the machine. This rate is shown in Table 14 and can be calculated by dividing the EGFLOPS by the *FLOPs Ratio* shown in Table 10. Therefore, the real compute rate of O^4 reaches 38% of the machine's peak. Further, by taking into account the maximum compute rate allowed by the mix between adds, muls, and fused madds, we see that the algorithm reaches 54% of its theoretical peak, as pointed out in Section 8.2.1.

The experiments from Table 14 were obtained by running batches of three executions of each configuration. Tables 23 through 26 (Appendix B bring more details, including the execution times, of one such batch, showing little variation between the three executions of the same problem configuration. This also happened between the execution of different batches of experiments. Table 14 reports the best results from Tables 23—26.

8.3 High-Order Stencils

Table 15 shows performance results, in equivalent giga-flops per second (EGFLOPS), of the star-shaped 2D stencil of 9 points and symmetric coefficients. The problem size for the single core experiment takes only two times the L3, so that it serves as a speed-of-light of the analysis. The speed-up of the convolved form over the naive is 1.5× for both the 1 core and 24 experiments, and both kernels scale from the 16MB to the 9GB problem size with a scaling factor of 93%. Have in mind that the naive codes for both Tables 15 and 16 already present the CWR and PWC variations of naive operators, which is hand-optimized with common sub-expression elimination after loop unrolling.

Table 16 shows results for a 2D 13-point symmetric stencil and its convolved form

Table 15: EGFLOPS achieved in the (quadratic) weak scaling of the 2D 9-point and its O^2. See Table 12 for domain sizes.

Cores	1	24
2D 9-point (O^1)	3.94	88.7
O^2	5.95	133
Speed-up	1.5	1.5

Table 16: EGFLOPS achieved in the (quadratic) weak scaling of the 2D 13-point and its O^2. See Table 12 for the domain sizes.

Cores	1	24
2D 13-point (O^1)	5.88	126.74
O^2	7.99	190.67
Speed-up	1.36	1.5

with $\Delta = 2$. The shape of this originally 2D 13-point is itself the same of the kernel convolved form of the 2D 5-point, with degree 2. The scalability of the O^2 kernel was the nearly perfect $\frac{190.67}{24 \times 7.99} = 99.43\%$. *It means that time-blocking has no power to be beneficial for this kernel-convolved operator, for what time-blocking does is to help a code to have better scalability, and that code already presented a nearly perfect scalability with respect to the single core performance. Additionally, the single core performance of* O^2 *was already* $1.36\times$ *the best performance that a state-of-the-art implementation with no kernel convolution could possibly accomplish.*

8.4 Related Work

This section compares the performance of stencil codes optimized with both space-blocking and time-blocking techniques, found in the literature. Pochoir (TANG *et al.*, 2011) is a stencil compiler that applies time-blocking through a cache-oblivious approach. Bandishti et al. (BANDISHTI; PANANILATH; BONDHUGULA, 2012) develop an advanced time-blocking approach and compare their results with Pochoir, in the Intel Xeon E5645 and AMD Opteron 6136 architectures. Both sets of results are

reported in Table 17. In Table 17, the results of Pochoir also come from (BANDISHTI; PANANILATH; BONDHUGULA, 2012).

Before comparing the 2D results with the literature, a caveat is necessary: Bandishti et al. report the performance of a 2D 5-point stencil that applies 10 FLOPs per calculated stencil, as stated in Figure 11 of their work, instead of the more commonly found version of 6 FLOPs per stencil. They also compile their codes with the flag "-fp-model-precise", ensuring that those 10 FLOPs would not be reduced. For comparison purposes, the net result of that approach is that, in purely memory-bound scenarios, their performance would be $10/6 \approx 1.67$ that of the more common version, which is the one employed throughout this work and most common in literature. Although time-blocking helps a 2D code to overcome memory-boundedness, the instruction level parallelism and other characteristics of the computation with 10 FLOPs do cause the related code to achieve higher compute rate (FLOPS). For example, in the platform used for the 2D experiments of this thesis, the 2D 5-point with 10 FLOPs per stencil achieves a speed-of-light compute rate $1.47\times$ higher than the 2D 5-point with 6 FLOPs. Therefore, when comparing with results from (BANDISHTI; PANANILATH; BONDHUGULA, 2012), Table 17 also shows a corrected range of attained FLOPS, from $1/1.67$ to $1/1.45$.

The best performance achieved in Bandishti et al.'s experiments is 57.28 GFLOPS across the 12 cores of a full node. The corrected range is thus 34.3 – 39.5 GFLOPS. Compared with the single core performance, this means a scalability of 68%. In 12 cores, thus 2 NUMA domains, the kernel convolved of $\Delta = 4$ reached 72.6 EGFLOPS, showing nearly perfect scalability (Figure 14). With $\Delta = 2$, the convolved kernel reached 55.9 EGFLOPS. Additionally, from Table 14, the single-core performance of O^2, O^3, and O^4 was respectively 4.84 EGFLOPS, 5.33 EGFLOPS, and 6.11 EGFLOPS, whereas the best performance from Table 17, when the FLOPs correction is applied, lies at most somewhere in the range 4.2 – 4.9 GFLOPS, for a domain 2.36 times smal-

Table 17: Strong scaling of time-blocking approaches in the literature, between 1 and 12 cores. The domain is 16000×16000 points, amounting to 1.9 GB.

Approach	Architecture	Cores	GFLOPS	Scaling	Corrected Range
Pochoir	Intel	1	4.88		2.9 – 3.4
Pochoir	Intel	12	50.81	0.87	30.4 – 35.0
Bandishti	Intel	1	7.04		4.2 – 4.9
Bandishti	Intel	12	57.28	0.68	34.3 – 39.5
Pochoir	AMD	1	2.76		1.7 – 1.9
Pochoir	AMD	12	28.59	0.86	17.1 – 19.7
Bandishti	AMD	1	4.81		2.9 – 3.3
Bandishti	AMD	12	30.03	0.52	18.0 – 20.7

ler. That said, it would be interesting for the community to see authors of the state-of-the-art methods using their techniques in conjunction with convolved kernels, possibly generating better results than the ones in this thesis.

9 EXPERIMENTS AND RESULTS - 3D

To compare ASLI against other techniques in the state-of-the-art, start by noting that the convolution or ASLI version of a stencil operator O^1 results in yet another stencil operator O^{RIS}. Therefore, if a given published work already in the literature investigates two operators A and B and if B happens to be the implementation of an ASLI scheme of A, it is possible to build, on top of existing literature, reproducible and far-reaching empirical analyses of ASLI. Such analyses enabled by the nature of ASLI can at once: (a) prove the soundness of ASLI, (b) prove that it can be applied with existing techniques (c) and framework solutions, (d) and prove these for varying machines. After such analyses, a non-optimized implementation of ASLI is compared against implementations of naive stencil operators fine-tuned with other techniques, highlighting ASLI's relative simplicity.

The notion of Equivalent Stencils (ES) is used to compare the performance of kernel convolved and straightforward implementations. For example, if both kernels are given the same problem input (domain and amount of iterations), a convolved kernel issues fewer intermediate stencil outputs in total but both calculate the same ES, as they solve the same problem. See Sections 5.3 and 5.5.1.3 for further discussion about equivalent metrics and their relevance.

9.1 SIMD Properties of ASLI with Patus Framework, for a 3D stencil

The 3D, 7-point and the 3D, 25-point stencils are analyzed when implemented within the Patus framework in (CHRISTEN; SCHENK; CUI, 2012), among other stencils. Patus is an interesting framework that automates parts of the optimization of stencil computation, achieving impressive speed-ups. In the future, this kind of framework will be even more helpful to generate and optimize convolved kernels. By now, in addition to optimizing original stencil forms, that work can be reinterpreted to show that even highly optimized stencil implementations can be substantially sped-up by ASLI. Although a complete implementation of 3D ASLI with Patus should treat the borders in a different way than in the original stencil, the achieved speed-up is so high that the common assumption can be made that dealing with the border elements in a different way would not hinder a considerable speed-up.

One iteration of an ASLI scheme with *Reduction in Iteration Space* of 2 (*RIS=2*) performs the same computation of two iterations of the corresponding original stencil. By definition, the 3D, 25-point corresponds to the ASLI with $RIS = 2$ of a 3D, 7-point stencil. Therefore, one can exploit the data in (CHRISTEN; SCHENK; CUI, 2012) to assess how ASLI works with Patus and, more broadly, interacts with other existing techniques.

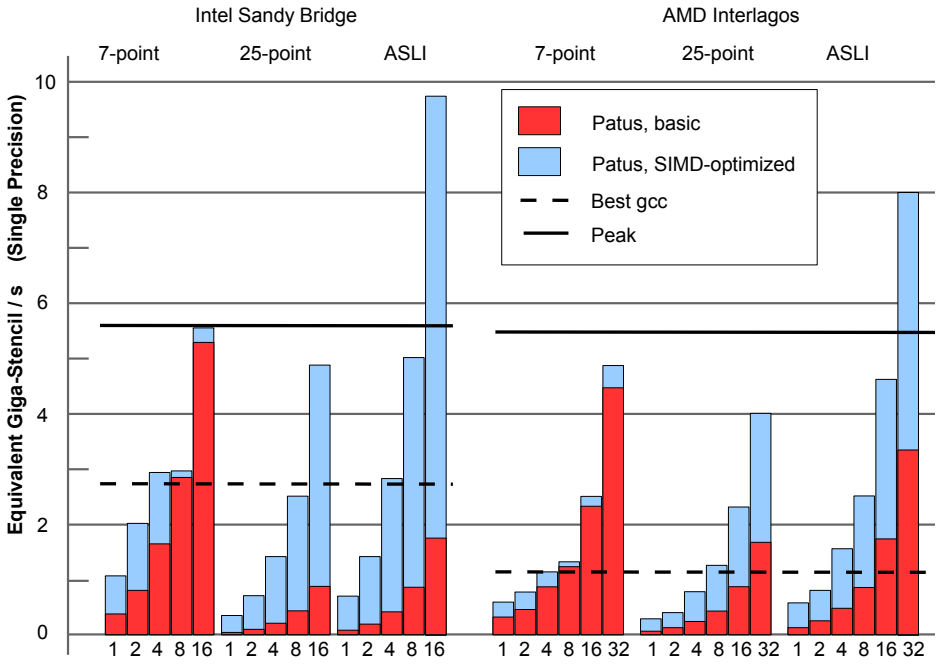
The platforms experimented in (CHRISTEN; SCHENK; CUI, 2012) were: (a) an Intel Sandy-Bridge (E5-2670) with 16 physical cores of 2.6 GHz, 32K L1 cache, 32K L2, 20M L3, and two NUMA domains, with a bandwidth of 62 GB/s observed with the STREAM Triad benchmark; and b) an AMD Opteron Interlagos node of a Cray XE6 system, with two sockets amounting to a bandwidth measured, with STREAM Triad, as 44 GB/s. This node offers a total of 32 hardware threads, each with its own 16KB L1 data cache, and every two threads share an engine that supports the AVX-256

Table 18: Speed-up that could have been attained with an ASLI traversal, based on (CHRISTEN; SCHENK; CUI, 2012)

	FLOPs	ES	GFlop/s (Intel)	EGS/s (Intel)	GFlop/s (AMD)	EGS/s (AMD)
3D, 7-point	8	1	44	5.5	40	5
3D, 25-point	29	1	144	4.97	117	4.03
ASLI,RIS=2	29	2	144	9.93	117	8.07
Speed-up obtained:				1.81		1.61

instruction set.

Figure 34: Speed-up attainable with ASLI, based on (CHRISTEN; SCHENK; CUI, 2012), for single precision 3D, 7-point, with domain of 256^3 points.



Source: Adapted from (CHRISTEN; SCHENK; CUI, 2012).

The yellow cells of Table 18 show performance results reported in (CHRISTEN; SCHENK; CUI, 2012) for the 3D, 7-point and 25-point, both in single precision. By taking into account the FLOPs required by each stencil output and that by definition for these two cases each stencil output corresponds to one single equivalent stencil (ES), it is possible to calculate the EGS/s (equivalent giga-stencil per second) that would have been achieved by Patus (red cells of the table). The blue row conveys that, by design, the 3D, 25-point has the same computation and memory access pattern of a possible

ASLI scheme with RIS=2 of the 3D, 7-point, but ASLI performs two times the EGS/s of the corresponding 25-point kernel. Thus, this rate must be compared to the EGS/s of the 3D, 7-point kernel, as this is the problem solved by the ASLI kernel in question. Such a comparison is carried on below the table and shows the speed-up that ASLI attains in this scenario. ASLI can get 1.81 speed-up over an optimized straightforward implementation in the Intel Sandy Bridge, and 1.61 in the AMD Opteron Interlagos. Central to the results of (CHRISTEN; SCHENK; CUI, 2012) was the exploitation of the SIMD unit (AVX) and inline assembly. The reasoning and conclusion above highlight the new vectorization possibilities created by ASLI. Also note that, with *CWR*, the 29 FLOPs of the ASLI kernel of Table 18 could be reduced to 24, as shown in Chapter 3. This reduction would also bring the desired effect of increasing the arithmetic intensity, and perhaps more speed-up.

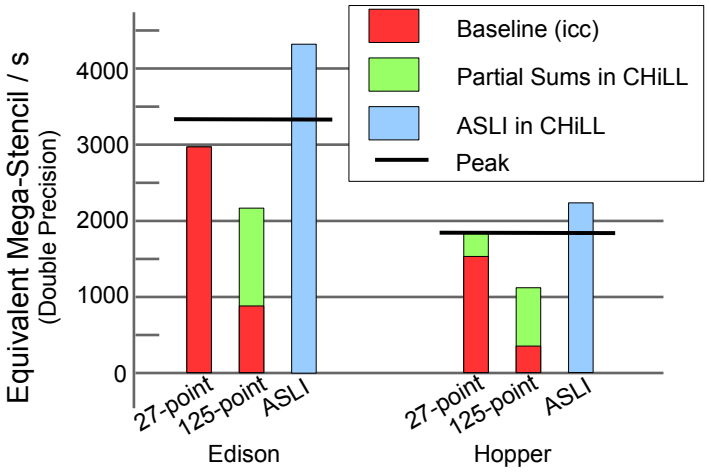
The reasoning carried on in Table 18 is replicated for an intra-node scalability study in Figure 34. The solid lines in the figure show the roofline of 3D 7-point in both machines, and how ASLI allows to circumvent this limitation. For this comparison, Patus allocated the threads to fill one NUMA-domain before using the other(s). The performance results identified as *Patus, basic* feature NUMA-awareness, cache-blocking, automatic vectorization, loop unrolling, and auto-tuning. The *Patus, SIMD-optimized* results have the extra optimization of explicit AVX vectorization with inline assembly, allowing more compact SIMD code and more efficient index computation. The figure shows that ASLI allows the performance to exceed roofline (just the roofline of full nodes are shown). Overall, ASLI generates opportunities for SIMD'zation to be more effective.

9.2 3D, 27-point cube-shaped stencil with CHiLL

The authors of (BASU *et al.*, 2015) extended previous optimizations of the CHiLL code generation framework (BASU *et al.*, 2013). They wrote a transformation recipe

for CHiLL to apply partial sums with loop transformations. Similarly to Patus (Section 9.1), CHiLL and such extension work to aid the programmer in the generation of optimized stencil code. The technique of partial sums is more effective for stencils that originally have many symmetric coefficients. It thus differs from the ancillary technique *CWR* presented here, which can also exploit stencils that are originally asymmetric or small, because through convolution ASLI increases the amount of computation suitable for reused. Nonetheless, this kind of framework can be very beneficial in aiding researchers and programmers to deal with higher levels of convolution and the convolution of more complex original operators.

Figure 35: Speed-up attainable with ASLI traversal, based on (BASU *et al.*, 2015), for double precision 3D, 27-point, with domain of 256^3 points.



Source: Adapted from (BASU *et al.*, 2015).

The platforms experimented in (BASU *et al.*, 2015) were located at NERSC: **Edison**, a Cray XC30, each node of which with two 12-core Xeon Ivy Bridge chips (E5-2695v2) containing four DDR3-1600 memory controllers and 30MB L3 cache. Each core features AVX SIMD and private 32KB L1 and 256B L2 cache. **Hopper**, a Cray XE6, each node of which with four 6-core Opteron chips containing two DDR3-1333 memory controllers and a 6MB L3 cache. Each core features SSE3 SIMD and private 64KB L1 and 512KB L2 caches. The BW in Edison and Hopper was measured with STREAM Copy to be 88GB/s and 48GB/s, respectively. The codes were compiled

with the flags -O3 -fno-alias -fno-fnalias and -xAVX or -msse3, with *icc*.

Figure 35 shows in the red and green bars the performance attained by Basu and others (BASU *et al.*, 2015) with the technique of partial sums, implemented in CHiLL, when applied to the 3D cube-shaped 27-point and 125-point stencils. The horizontal lines show the roofline of the 3D 27-point in both machines. The baseline results already include optimizations performed by CHiLL. This renders the approach of partial sums less effective for the 3D, 27-point, as its baseline performance nears the memory-bound roofline. Recall from Section 4.4 that the computation pattern of the 125-point stencil matches an ASLI scheme with RIS=2 of the 27-point stencil and that, if the original O^1 (in this case, the 27-point) is symmetric, than O^2 is symmetric as well, so that *PWC* and *CWR* can be further employed with ASLI. Now, in this symmetric and cube-shaped scenario, the *ASLI+CWR+PWC* version of 27-point matches the immediate application of the Partial Sums to the 125-point operator, except on the borders.

Therefore, with a reasoning similar to the one in Section 9.1, one can assess exactly how ASLI applied to the 3D, 27-point stencil performs when implemented within CHiLL. The blue bars of Figure 35 show that, again, ASLI's transforming the computation pattern allows a performance better than the expected roofline. For this case of cube-shaped 27-point O^1, the utilization of CWR and PWC was paramount to enhance the corresponding ASLI's performance — without them, the convolved implementation of the 27-point, as coded by that study, would meet just two thirds (Intel) and half (AMD) the time to solution of CHiLL's baseline implementation.

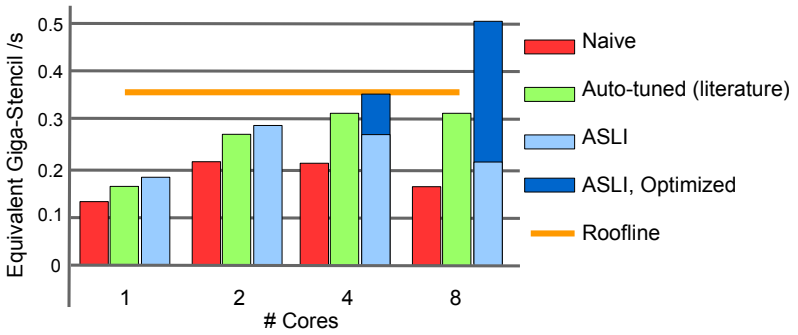
With CHiLL and Patus it has been shown that an optimized implementation of ASLI allows performance well beyond the memory-bound limit. In Section 9.1, CWR and PWC were not applied. On the other hand, these techniques were paramount for the ASLI version of the 3D, 27-point. Both sections show that ASLI can be implemented with existing frameworks and helps to enhance other optimization techniques.

9.3 Non-optimized ASLI vs. Fine-Tuned solutions, and In-Cache Behavior

Whereas the previous experiments used data already in the literature to show how beneficial ASLI would have been in already existing optimization frameworks, the following comparisons use 3D codes developed by the author without the aid of stencil-specific optimization tools.

9.3.1 Comparison with the Berkeley Auto-Tuner

Figure 36: Comparison with Berkeley auto-tuner. The roofline was determined by a speed-of-light approach, as described on Section 9.3.2.



Source: Author (ASLI results), and adapted from (DATTA *et al.*, 2008).

Figure 36 compares ASLI applied to a 3D, 7-point and symmetric stencil against the auto-tuner used on (DATTA *et al.*, 2008). Green and red bars show their reported results. The ASLI experiments and those in (DATTA *et al.*, 2008) were run on two different machines specified as Intel Clovertown E5355, 2.66GHz, 8 cores, 32K L1, 4x4M L2 (shared by two cores). Datta et al. found that core blocking and cache-bypass are the most important techniques in obtaining their results for domains of 256^3, which do not fit in the L2 cache. The results measured for ASLI were achieved with a relatively simple implementation of ASLI, with RIS=2, with an eight times unrolling of the innermost loop (List 3.1) and without time-blocking. ASLI codes were compiled with *gcc*, without vectorization flags. Even with the unrolling, more than the

necessary amount of register moves and dependencies were present in the generated code. Such imperfections point to the potential of ASLI and can be amortized with a better scheduling, but such study lies beyond the scope of this work. Clovertown's small bandwidth renders the exploitation of the 8 cores disadvantageous for the blind implementation of ASLI and ineffective for the auto-tuner. Whereas the previously analyzed optimized versions of ASLI surpassed the roofline of the corresponding naive implementation, the best result of the non-optimized ASLIs, which occurs when only 2 cores are used, reaches 92.2% the performance of the best auto-tuned configuration. This also is an encouraging outcome, given ASLI's simplicity in relation to developing and using an auto-tuner.

9.3.2 The In-Cache Results

In his thesis (DATTA; YELICK, 2009), Datta shows the in-cache (last cache level) performance of the Laplacian 3D, 7-point Stencil and argues that such performance would be the peak achievable by his optimizations when applied to larger domains, which means to say it was the speed-of-light (Section 5.1). He used a domain size of 128^3 elements for his experiments. In another work (DATTA *et al.*, 2008), Datta et al. observed that *"experiments on a smaller* 128^3 *calculation [...] saw little benefit from auto-tuning, as the entire working set easily fit within Clovertown's large 2MB per core L2 working set"*. In this scenario, Datta et al. achieved circa 0.35 GStencil/s (Figure 36), then using this value as the roofline targeted by their auto-tuner. *On a machine of same specification and for the same problem size, the ASLI approach achieves 0.47 GStencil/s, meaning an in-cache speedup, and therefore a roofline shift, of* $\times 1.34$. ASLI's shifting the in-cache results points that this technique, in this machine and for this 3D operator, might synergistically work with time-blocking, but this remains as future work. Corollary 9.3.1 argues that there are still many possibilities in the science of enhancing the performance of stencil codes and scientific kernels.

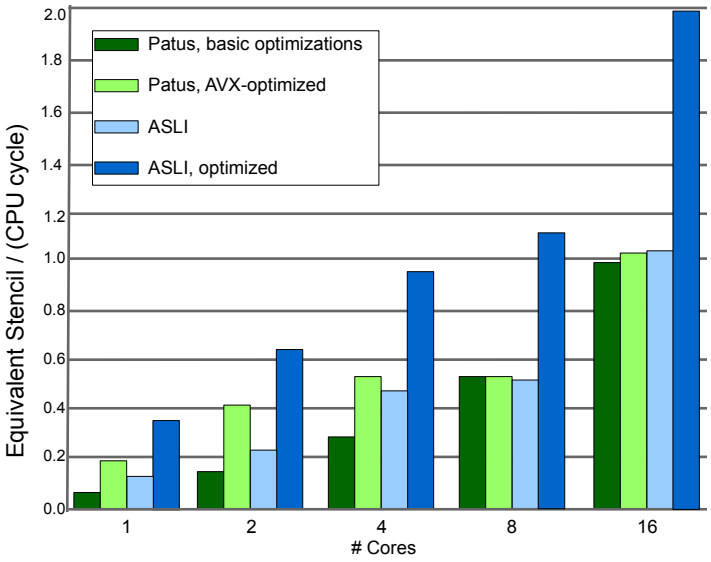
Corollary 9.3.1. *It follows that the **in-cache** 3D 7-point stencil is not compute bound in multi-level cache architectures. This conclusion stems from the facts that (i) for such a problem configuration the ASLI version performed better than a straightforward, highly optimized version and (ii) ASLI requires more FLOPs per equivalent stencil.*

9.3.3 Comparison with the Patus Framework

Figure 37 compares ASLI implementations with the stencil compiler Patus. Note that Section 9.1 has already proven that an ASLI implementation in Patus outperforms a non-ASLI implementation in Patus. In the current section experimental results of Patus reported in (CHRISTEN; SCHENK; CUI, 2012) are compared with ASLI implementations that do not use Patus. Christen and others (CHRISTEN; SCHENK; CUI, 2012) show results for single precision and state that the *"double precision variants perform at half of the numbers shown"*. The Patus results were collected in the 2.6 GHz Intel Sandy-Bridge (E5-2670) described in Section 9.1. The green bars of Figure 37 are the double precision equivalent of the red and blue bars in the leftmost cluster of Figure 34). The non-optimized ASLI results were collected on a 2.4 GHz Intel Sandy Bridge (E5-2665). Apart from the difference in the clock rate, both machines have same specification (cores-, caches-, NUMA domains-, BW-, and SIMD-wise).

To cope with the difference in the theoretical achievable peak floating point performance, the results are shown in terms of *Stencils per (CPU) cycle*, instead of the more usual *Giga-Stencil per second*. The performance that the original computation pattern achieves on the machine where ASLI was tested is also reported. Both the implementations of the straightforward pattern and of ASLI feature NUMA-awareness, but have no SIMD'zation flag enabled. The results tagged *Patus, basic* feature NUMA-awareness, cache-blocking, automatic vectorization, loop unrolling, and auto-tuning. The *Patus, SIMD-optimized* results have the extra optimization of explicit AVX vectorization with inline assembly.

Figure 37: Comparison with results from Patus compiler (CHRISTEN; SCHENK; CUI, 2012), for double precision 3D, 7-point stencil, with domain of 256^3 points.



Source: Author (ASLI results), and adapted from (CHRISTEN; SCHENK; CUI, 2012).

In the threading scheme experimented with ASLI, the threads are spread among the two NUMA domains. That accounts for the performance gap between this specific ASLI implementation and Patus when 8 cores are used. Once again, the non-optimized version of ASLI is on pair with a highly optimized implementation of the 3D, 7-points.

10 FINAL CONSIDERATION

Kernel convolution (KC) is a technique based on data reuse. It attempts to increase the amount of useful work performed on the data brought to the caches and register files. To do so, KC can even complicate or alter intermediate computations: the important directive is that the computation of the final output be mathematically equivalent to the non-convolved version. One possible application of KC to stencil is through Aggregate Stencil-Loop Iteration (ASLI). KC is by no means confined to the stencil domain. Suppose two computations A and B that must be successively applied to a domain. These computations can be convolved into $D = A \circ B$ or even $D = A \circ B \circ A \circ B$. In the cases analyzed here, a desired reduction in iteration space (RIS) is achieved when the computation uses, instead of the original stencil operator O^1, the convolved operator $O^{\text{RIS}} = O^1 \circ O^1$ ... where O^1 appears RIS times in the right-hand side of the previous definition. This new computation pattern cuts the sweeps through the domain points by a factor of RIS. Kernel characteristics of 1D, 2D, and 3D ASLI were discussed. Computation reuse with asymmetric and high-order was exemplified for 1D and 2D stencils. The high-order 3D, 27-point was analyzed empirically. In the 1D case, ASLI results in fewer FLOPs than a naive implementation. For the 2D and 3D cases, ASLI results in more FLOPs. The technique enhances the computation to load ratio, requiring fewer loads and stores, and does pay off in spite of additional FLOPs, as shown by the experiments with the 3D 7-point and 3D 27-point operators. Additionally, it was proven that ASLI allows to better exploit SIMD units, for example by increasing the ratio of computation to loaded bytes. ASLI also increased the in-cache

performance of the 3D star-shaped stencil, offering a better baseline performance or departure point for further optimizations.

KC and ASLI make the optimization more straightforward for scientists not directly involved in optimizing. In this regard, it was shown that ASLI can be stacked with existing dedicated frameworks or compilers, such as CHiLL and PATUS. The Influence Table was introduced to aid programmers and compiler developers. The concept of influence tables might be used to explore convolutions of kernels of other computational patterns. Additionally, two side techniques were introduced to help increase performance: the Computaion Wavefront Reuse (CWR) and the Pair-Wise Composition (PWC). In the road to exaflop performance, new algorithms and paradigms have a role to play. This work argues that ASLI, and kernel convolution in general, work in such a direction. In this scenario, 1D ASLI is the paragon, as its FLOPs ratio allows to achieve a time to solution faster than a hypothetical straightforward implementation could get with 100% peak floating-point performance. An integer sequence has been determined which describes the coefficients of convolutions of the 1D 3-point stencil and which relates traversals of the binomial numbers to the trinomial numbers.

10.1 Discussion

In the 2D and 3D cases, kernel convolution through ALSI increases the ratio of FLOPs to equivalent stencil. This should mean that no ASLI version would perform better than an approach that just iterates straightforwardly through the time domain provided all latency and register dependency are hidden and provided a perfect use of SIMD in the straightforward implementation. However, this work argues that ASLI does pay off. To prove that, the 3D case was particularly elucidating. Table 2 shows that the *FLOPs Ratio* in this case has the most challenging value of 0.67.

Because kernel convolution transforms between different stencil patterns, it might

be the case that some work in the literature, unbeknownst to the authors, evaluates some techniques or machines with two stencil patterns that correspond to a pair of original and convolved patterns. This way, it is possible to evaluate ASLI as coded by various programmers, in cooperation with different optimization techniques, in various platforms. This quirky aspect of kernel convolution helps to cope with the reproducibility issue in science, where different groups of authors are not able to reproduce results of another group, be it for differences in machines, in runtime environment, or any cryptic configuration knob.

The implicit conclusion reached in (CHRISTEN; SCHENK; CUI, 2012) and studied in Section 9.1 can be dubbed as the Patus Result. Although unrelated to ASLI, that work shows the possible benefits of ASLI when applied with complementary optimizations. The Patus Result is that an optimized ASLI implementation of the 3D, 7-point stencil is usually better than an optimized, straightforward implementation of the 3D 7-point stencil. A similar reasoning allowed to evaluate results from the CHiLL compiler, again showing the effectiveness of ASLI. In fact, results in literature that hint at the effectiveness of ASLI date as early as 1991, when the 2D 5-point and the 2D 13-point were evaluated in (BROMLEY *et al.*, 1991), among other patterns. Unknowingly, Bromly et al. showed that ASLI, even without reducing FLOPs with pair-wise composition and computation wavefront reuse, could obtain speed-ups of 1.1 and 1.16 for grids of size 128×256 and 256×256 per core, respectively. Thus, kernel convolution has been latent in the literature for nearly two decades, waiting to be unveiled.

Kernel convolution also tackles the issue brought up by Williams et al. (WILLIAMS; WATERMAN; PATTERSON, 2009) regarding productivity and programming difficulty, as in many cases it is relatively simple to apply. When it comes to more complex calculations of convolved coefficients, including of the ones on the borders, it suffices that the scientific community analyze the pattern once, and everyone else would be able to benefit from the pattern. It can be applied by the scientific com-

munity to a range of applications in various languages and environments. The more overhead the language and the environment add to the underlying computation, the higher the potential for speed-up. Other techniques and solutions are often discouraging, as they usually require specialized compilers or more complex infrastructure. Automated approaches and compiler techniques to apply kernel convolution could benefit from a notation that better allows the discovery of dependencies in workflows of computation. The work (KRUEGER *et al.*, 2011) notes that "the free-lunch of getting performance improvements from ever-increasing clock frequencies is over and more radical approaches to improving the energy efficiency of computer architectures are going to be required to avert a power crisis or catastrophic stall in computing performance". Aggregate Stencil-Loop Iteration is one such more radical approach, and comes to add new fuel to the science of fast stencil and similar computation. ASLI hides latencies and register dependency, besides being communication-avoiding.

10.2 Differentiation from other techniques

"ASLI not just reorders the computation, but merges two or more iterations of the outer-loop and allows CWR and PWC, which in symmetric non-star shaped operators already large enough is the reuse of sub-expressions." Parts of the description above might sound similar to other techniques, but the similarity ends there. Precision in language is necessary when contrasting the approaches, in order to avoid equivocal misunderstanding. For example, "time-blocking performs two or more iterations with one memory pass, thus merging iterations" in reality accompanies the implicit fact that "memory" means "main memory" and the implicit statement that "it still updates every point for the pre-determined amount of time, so it also updates the memory cache in the usual way". The entirety of the previous statements thus differs by a great deal from the statement "ASLI merges two or more iterations, updating each domain point fewer times". Perhaps this kind of difficulty had hindered the disclosure of kernel

convolution and accounts for the lack of the insight that the original symmetry would be kept in convolved forms, allowing for reuse in convolved operators. This section attempts to clear some behavioral aspects of ASLI, while explaining how it differs from other techniques in the literature, such as time-blocking (TB), common sub-expression elimination (CSE), and associative reorder (AR) (STOCK *et al.*, 2014):

- ASLI changes the amount of *necessary* FLOPs, reducing for 1D and increasing for 2D and 3D, meaning that even if CSE is applied to the original stencil and to the respective ASLI version, both would require different amounts of FLOPs.
- ASLI always increases the *arithmetic and operational intensities*, where *arithmetic intensity* denotes the FLOPs per data loaded from caches into register, and increases the amount of independent sub-expressions, allowing better instruction scheduling.
- ASLI never explicitly computes the value of any point for the skipped iterations, thus not writing said skipped values into caches nor main memory.
- ASLI can reuse the computation of lines and surface, even when the original stencil is asymmetric.

Regarding other techniques:

- TB, AR do not change the amount of FLOPs.
- TB, AR do not imply changing the arithmetic intensity.
- TB, AR, CSE compute every value of every iteration.
- CSE is not applicable to stencils originally asymmetric, let alone to the also asymmetric convolved kernels, which CSE does not generate.
- CSE, AR would never generate a code that requires more computation, as ASLI does for 2D and 3D.

As ASLI works by still applying a stencil, it can be implemented in conjunction with other techniques otherwise applicable solely to the original stencil operators. In this regard, ASLI adds a new dimension to auto-tuners and code-generators, adding an extra fuel to the science of high-performance stencil computation. It would be really interesting to investigate how Associative Reorder and CSE, among so many other techniques used in Patus, CHiLL, Pochoir, etc., could be applied to the convolved kernels.

10.3 Contributions

The main contributions of this work are:

- A technique (ASLI) to speed up stencil computation.
- The concept of Kernel Convolution.
- The Influence Table.
- Schemes to apply computation reuse with ASLI for 1D, 2D, and 3D stencils.
- Schemes to apply reuse with higher-order and/or asymmetric stencils, a reuse which emerges solely as consequence of the kernel convolution.
- It has been shown that ASLI has been a latent optimization in the literature, here unveiled.
- Closed formulas for the convolved coefficients of 1D 3-point with any degree of convolution.
- The Butantan numbers and a way to obtain any trinomial number from the sum of products of two describable sequences of binomial numbers.
- Practical approaches to derive convolved coefficients.

- Practical approach to determine equations for the elements close to the border of the domain.
- Experimental results:
  - Results combined with other techniques and frameworks, proving ASLI's viability;
  - Results for 1D stencils of multiple sizes and degrees of convolution;
  - Results for 2D stencils, for multiple operators and degrees of convolution;
  - Results for 3D stencils and the exegesis of previous literature, thus proving the effectiveness of ASLI for multiple machines, including for the cube 27-point, in synergy with multiple other optimization techniques.
- Practical example of how ASLI increases the in-cache performance and the speed-of-light of stencil codes, in addition to enabling better scalability.
- Practical example of how some degrees of convolution can achieve by themselves better performance than any time-blocked approach that just uses the naive operator.
- Metrics to analyze the optimization achieved with ASLI, at least one of which can be used as a benchmark to evaluate computational platforms. And how to assess whether further convolution could yield more gains.

10.4 Future Work

From this point, maybe an avenue of research topics both theoretical and more practical can be devised. Below are some ideas.

1. Compiler Related Aspects:
   (a) Auto-Tuning.

(b) Stencil notation, try to generalize to other applications.

(c) Array-Notation and other compiling techniques, not just for stencil.

(d) Automation of the Influence Table.

(e) Detection and conditions necessary to apply the technique.

2. Speed Aspects:

(a) Over 100% EFLOPs in 1D, and hyper-performance of the strong type.

(b) Unprecedented performance in 2D.

(c) Unprecedented performance in 3D.

(d) Many Many Cores, GPU, Xeon Phi.

3. Performance Evaluation Aspects:

(a) Memory Behavior.

(b) Algorithm for Characterization or Prediction.

(c) Kernel convolution as benchmark for compilers and machines, with the concept of inherent gain discussed here.

4. General Aspects:

(a) The pursuit of closed formula for other configurations, with higher radius in 1D, and 2D and 3D in general.

(b) Analyze the reuse in asymmetric and higher-order 3D.

(c) Other applications than stencil.

REFERENCES

BANDISHTI, V.; PANANILATH, I.; BONDHUGULA, U. Tiling stencil computations to maximize parallelism. In: IEEE COMPUTER SOCIETY PRESS. ***Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis***. [S.l.], 2012. p. 40.

BASU, P. *et al.* Compiler-directed transformation for higher-order stencils. In: ***Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International***. [S.l.: s.n.], 2015. p. 313–323. ISSN 1530-2075.

____. Compiler generation and autotuning of communication-avoiding operators for geometric multigrid. In: ***High Performance Computing (HiPC), 2013 20th International Conference on***. [S.l.: s.n.], 2013. p. 452–461.

BROMLEY, M. *et al.* Fortran at ten gigaflops: The connection machine convolution compiler. In: ***Proceedings of the ACM SIGPLAN 1991 Conference on Programming Language Design and Implementation***. New York, NY, USA: ACM, 1991. (PLDI '91), p. 145–156. ISBN 0-89791-428-7. Disponível em: <http://doi.acm.org/10.1145/113445.113458>.

CHRISTEN, M.; SCHENK, O.; CUI, Y. Patus for convenient high-performance stencils: evaluation in earthquake simulations. In: IEEE. ***High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for***. [S.l.], 2012. p. 1–10.

CRUZ, R. de la; ARAYA-POLO, M. Algorithm 942: Semi-stencil. ***ACM Trans. Math. Softw.***, ACM, New York, NY, USA, v. 40, n. 3, p. 23:1–23:39, abr. 2014. ISSN 0098-3500. Disponível em: <http://doi.acm.org/10.1145/2591006>.

DATTA, K. *et al.* Optimization and performance modeling of stencil computations on modern microprocessors. ***SIAM Rev.***, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, v. 51, n. 1, p. 129–159, fev. 2009. ISSN 0036-1445. Disponível em: <http://dx.doi.org/10.1137/070693199>.

____. Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. In: IEEE PRESS. ***Proceedings of the 2008 ACM/IEEE conference on Supercomputing***. [S.l.], 2008. p. 4.

DATTA, K.; YELICK, K. A. *Auto-tuning stencil codes for cache-based multicore platforms*. Tese (Doutorado) — **University of California, Berkeley/**, 2009.

DURSUN, H. *et al.* A multilevel parallelization framework for high-order stencil computations. In: ***Euro-Par 2009 Parallel Processing***. [S.l.]: Springer, 2009. p. 642–653.

FRIGO, M.; STRUMPEN, V. Cache oblivious stencil computations. In: ***Proceedings of the 19th Annual International Conference on Supercomputing***. New York, NY, USA: ACM, 2005. (ICS '05), p. 361–366. ISBN 1-59593-167-8. Disponível em: <http://doi.acm.org/10.1145/1088149.1088197>.

GYSI, T.; GROSSER, T.; HOEFLER, T. MODESTO: Data-centric Analytic Optimization of Complex Stencil Programs on Heterogeneous Architectures. In: ***Proceedings of the 29th International Conference on Supercomputing (ICS'15)***. [S.l.]: ACM, 2015. p. 177–186. ISBN 978-1-4503-3559-1.

HENRETTY, T. *et al.* A stencil compiler for short-vector simd architectures. In: ***Proceedings of the 27th International ACM Conference on International Conference on Supercomputing***. New York, NY, USA: ACM, 2013. (ICS '13), p. 13–24. ISBN 978-1-4503-2130-3. Disponível em: <http://doi.acm.org/10.1145/2464996.2467268>.

HOEFLER, T.; SCHNEIDER, T. Optimization principles for collective neighborhood communications. In: IEEE. ***High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for***. [S.l.], 2012. p. 1–10.

Guilherme Januario, Yoonho Park e Bryan Rosenburg. *Kernel convolution for stencil computation optimization*. **US Patent** 9 916 678, Mar. 13, 2018.

Guilherme Januario, Yoonho Park e Bryan Rosenburg. *Kernel convolution for stencil computation optimization*. **US Patent App.** 14/986,195, Jul. 6, 2017.

JANUARIO, G. *et al.* Speeding up stencil computations with kernel convolution. In: ***Computer Architecture and High Performance Computing (SBAC-PAD), 2016 IEEE 28th International Symposium on***. [S.l.: s.n.], 2016. p. 1–8.

KAMIL, S. *et al.* An auto-tuning framework for parallel multicore stencil computations. In: ***2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)***. [S.l.: s.n.], 2010. p. 1–12.

KING, J.; KIRBY, R. M. A scalable, efficient scheme for evaluation of stencil computations over unstructured meshes. In: ACM. ***Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis***. [S.l.], 2013. p. 79.

KORAEI, M.; FATEMI, O.; JAHRE, M. Dcmi: A scalable strategy for accelerating iterative stencil loops on fpgas. ***ACM Trans. Archit. Code Optim.***, Association for Computing Machinery, New York, NY, USA, v. 16, n. 4, out. 2019. ISSN 1544-3566. Disponível em: <https://doi.org/10.1145/3352813>.

KRUEGER, J. *et al.* Hardware/software co-design for energy-efficient seismic modeling. In: ACM. ***Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis***. [S.l.], 2011. p. 73.

MACHADO, R.; NASCIUTTI, T.; PANETTA, J. Avaliando o impacto de mudanÃ§as na arquitetura de memoria entre geracoes de gpgpus no desempenho das otimizacoes de computacoes de estenceis. In: ***Anais da IX Escola Regional de Alto Desempenho***

de SÃ£o Paulo. Porto Alegre, RS, Brasil: SBC, 2018. p. 41–44. Disponível em: <https://sol.sbc.org.br/index.php/eradsp/article/view/13598>.

MALAS, T. *et al.* Multicore-optimized wavefront diamond blocking for optimizing stencil updates. ***SIAM Journal on Scientific Computing***, SIAM, v. 37, n. 4, p. C439–C464, 2015.

MARUYAMA, N. *et al.* Physis: an implicitly parallel programming model for stencil computations on large-scale gpu-accelerated supercomputers. In: IEEE. ***High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for***. [S.l.], 2011. p. 1–12.

MATTSON, T.; WIJNGAART, R. V. D.; FRUMKIN, M. Programming the intel 80-core network-on-a-chip terascale processor. In: ***High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for***. [S.l.: s.n.], 2008. p. 1–11.

MYCZKOWSKI, J.; STEELE, G. Seismic modeling at 14 gigaflops on the connection machine. In: ***Supercomputing, 1991. Supercomputing '91. Proceedings of the 1991 ACM/IEEE Conference on***. [S.l.: s.n.], 1991. p. 316–326.

NASCIUTTI, T.; PANETTA, J. Impacto da arquitetura de memoria de gpgpus na velocidade de computacao de estenceis. ***WSCAD-SSC***, Simposio de Sistemas Computacionais, Aracaju, SE, p. 1–8, 2016.

ODEGAARD, B. A. *Financial Numerical Recipes in C++*. [S.l.], 2003. 152 p.

RIVERA, G.; TSENG, C.-W. Tiling optimizations for 3d scientific computations. In: ***Proceedings of the 2000 ACM/IEEE Conference on Supercomputing***. Washington, DC, USA: IEEE Computer Society, 2000. (SC '00). ISBN 0-7803-9802-5. Disponível em: <http://dl.acm.org/citation.cfm?id=370049.370403>.

ROTH, G. *et al.* Compiling stencils in high performance fortran. In: ***Proceedings of the 1997 ACM/IEEE Conference on Supercomputing***. New York, NY, USA: ACM, 1997. (SC '97), p. 1–20. ISBN 0-89791-985-8. Disponível em: <http://doi.acm.org/10.1145/509593.509605>.

SAROOD, O.; MENESES, E.; KALE, L. V. A 'cool' way of improving the reliability of hpc machines. In: IEEE. ***High Performance Computing, Networking, Storage and Analysis (SC), 2013 International Conference for***. [S.l.], 2013. p. 1–12.

SHIMOKAWABE, T.; AOKI, T.; ONODERA, N. High-productivity framework on gpu-rich supercomputers for operational weather prediction code asuca. In: IEEE PRESS. ***Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis***. [S.l.], 2014. p. 251–261.

STOCK, K. *et al.* A framework for enhancing data reuse via associative reordering. In: ***Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation***. New York, NY, USA: ACM, 2014. (PLDI '14), p. 65–76. ISBN 978-1-4503-2784-8. Disponível em: <http://doi.acm.org/10.1145/2594291.2594342>.

TANG, Y. *et al.* The pochoir stencil compiler. In: ***Proceedings of the Twenty-third Annual ACM Symposium on Parallelism in Algorithms and Architectures***. New York, NY, USA: ACM, 2011. (SPAA '11), p. 117–128. ISBN 978-1-4503-0743-7. Disponível em: <http://doi.acm.org/10.1145/1989493.1989508>.

WILLIAMS, S.; WATERMAN, A.; PATTERSON, D. Roofline: an insightful visual performance model for multicore architectures. ***Communications of the ACM***, ACM, v. 52, n. 4, p. 65–76, 2009.

APÊNDICE A – PROGRAM TO FIND LITERAL EXPRESSIONS FOR CONVOLVED COEFFICIENTS

This appendix shows the two files of a C implementation of the approach that uses literal arithmetics to determine the convolved coefficients, discussed in Section 6.8.4. The code also shows how to use the approach to derive coefficients for the border points, as suggested in Section 6.10.

Code of the file coeff_generator.h:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAX_LITERALS 11
const char literals[] = "abcdefghijklmnopqrs";
#define MAX_TERMS 100

// The coefficiens are supposed to be 'abcdefghijklmn..', with 'a' having index 0, and the last liter
typedef struct coeff{
        int mul;
        int exps[MAX_LITERALS];
} COEFFS;
typedef struct {
        char var_name[9];
        int combs_total;
        COEFFS coeff_list[MAX_TERMS];
} POINT;
```

```
typedef struct {
        char var_name[9];
        int points_total;
        POINT point_list[100];
} AGGREGATE;

#define LOOP(lim) int i;for(i=0;i<lim;i++)
void POINT_ini(POINT *p){
        strcpy(p->var_name, "UNK_P");
        memset(&(p->coeff_list),0, sizeof(COEFFS) * MAX_TERMS);
        p->coeff_list[0].mul = 1; // initial value: X = 1*a^0*b^0*c^0...
        p->combs_total=1;
}
void AGGREGATE_ini(AGGREGATE *agr){
        strcpy(agr->var_name, "UNK_AGR");
        agr->points_total=0;
}

void POINT_add_one_coeffs(POINT *dest, COEFFS *cosrc){
        LOOP(dest->combs_total){
                if(memcmp(&dest->coeff_list[i].exps, cosrc->exps, sizeof(cosrc->exps)) == 0 ){
                                dest->coeff_list[i].mul += cosrc->mul;
                                return;
                        }
        }
        memcpy(&dest->coeff_list[i], cosrc, sizeof(COEFFS));
        dest->combs_total +=1 ;
}
void POINT_add_coeffs(POINT *dest, POINT *src){
        LOOP(src->combs_total)  POINT_add_one_coeffs(dest, &src->coeff_list[i]);
}
void AGGREGATE_add_point(AGGREGATE *agr, POINT *p){
        LOOP(agr->points_total){ // the point is already represented in the aggregate
                if(strcmp(agr->point_list[i].var_name, p->var_name ) == 0){
                        POINT_add_coeffs(&agr->point_list[i], p);
                        return;
                }
        } // POINT not in the AGGREGATE --- adds it
        memcpy(&agr->point_list[agr->points_total], p, sizeof(POINT));
        agr->points_total++;
}
void AGGREGATE_add_agr(AGGREGATE *dest, AGGREGATE *other){
```

```
        LOOP(other->points_total) AGGREGATE_add_point(dest, &other->point_list[i]);
}
void AGGREGATE_attrib(AGGREGATE *dest, AGGREGATE *other){
        dest->points_total=0;
        strcpy(&dest->var_name, other->var_name);
        AGGREGATE_add_agr(dest, other);
}
void POINT_times_COE(POINT *dest, POINT *src, char coe){
        strcpy(dest->var_name, src->var_name);
        dest->combs_total=src->combs_total;
        LOOP(src->combs_total){
                memcpy(&dest->coeff_list[i].exps, &src->coeff_list[i].exps, sizeof(int) * MAX_LITERAL
                dest->coeff_list[i].exps[coe - 'a'] = src->coeff_list[i].exps[coe - 'a']+1;
                dest->coeff_list[i].mul = src->coeff_list[i].mul;
        }
}
void AGGREGATE_times_COE(AGGREGATE *dest, AGGREGATE *src, char coe){
        strcpy(dest->var_name, src->var_name);
        dest->points_total=src->points_total;
        LOOP(src->points_total){
                POINT_times_COE(&dest->point_list[i], &src->point_list[i], coe);
        }
}
void COEFF_print(COEFFS *co){
        if(co->mul==0)  printf("Attention, no multiplier\n");
        if(co->mul > 1) printf("%d*", co->mul);

        int found_literal=0;
        LOOP(MAX_LITERALS){
                if(co->exps[i] != 0){
                        if(found_literal>0)printf("*");
                        printf("%c^%d",literals[i], co->exps[i]);
                        found_literal=1;
                }
        }
        if(found_literal == 0)printf("1");
}
void POINT_print(POINT *p){
        printf("%s*(", p->var_name);
        LOOP(p->combs_total){
                COEFF_print(&p->coeff_list[i]);
                if(i!=p->combs_total-1)printf("+");
```

```
        }
        printf(")\n");
}
void AGGREGATE_print(AGGREGATE *agr){
        printf("------------\nAggregate: %s = {", agr->var_name);
        int i;
        for(i=0;i<agr->points_total;i++){
                POINT_print(&agr->point_list[i]);
                if(i!=agr->points_total-1)
                        printf("+");
        }
        printf("}\n------------\n");
}
```

Code of the file coeff_generator.c:

```
#include "coeff_generator.h"

int main(){
        COEFFS co_ini;

        const int DOMAIN_SIZE = 17;
        POINT xt0[DOMAIN_SIZE + 1];

        AGGREGATE *agA = malloc((DOMAIN_SIZE+1)*sizeof(AGGREGATE));
        AGGREGATE *agB = malloc((DOMAIN_SIZE+1)*sizeof(AGGREGATE));

        int i;
        for(i=0;i<=DOMAIN_SIZE;i++){
                POINT_ini(&xt0[i]); strcpy(&xt0[i].var_name[0],"I");
                itoa(i, &xt0[i].var_name[1],10);

                AGGREGATE_ini(&agA[i]); strcpy(&agA[i].var_name[0],"X");
                itoa(i, &agA[i].var_name[1],10);
                AGGREGATE_add_point(&agA[i], &xt0[i]);
        }

        AGGREGATE *temp0 = (AGGREGATE*)malloc(sizeof(AGGREGATE));
        AGGREGATE *temp1 = (AGGREGATE*)malloc(sizeof(AGGREGATE));
        AGGREGATE *temp2 = (AGGREGATE*)malloc(sizeof(AGGREGATE));

        // border condition
```

```
strcpy(&agA[0].point_list[0].var_name[0],"F0");
for(i=1;i<DOMAIN_SIZE-1;i++){
        AGGREGATE_times_COE(temp0, &agA[i-1], 'b');
        AGGREGATE_times_COE(temp1, &agA[i], 'a');
        AGGREGATE_times_COE(temp2, &agA[i+1], 'b');

        AGGREGATE_add_agr(temp1,temp0);
        AGGREGATE_add_agr(temp1,temp2);
        AGGREGATE_print(temp1);
        AGGREGATE_attrib(&agB[i], temp1);
        printf("END␣OF␣ITERATION␣%d\n", i);
}

printf("############################\n");
// border condition
AGGREGATE_ini(&agB[0]);strcpy(&agB[0].var_name[0],"f1");
AGGREGATE_add_point(&agB[0], &xt0[0]);
strcpy(&agB[0].point_list[0].var_name[0],"F1");

for(i=1;i<DOMAIN_SIZE-2;i++){
        AGGREGATE_times_COE(temp0, &agB[i-1], 'b');
        AGGREGATE_times_COE(temp1, &agB[i], 'a');
        AGGREGATE_times_COE(temp2, &agB[i+1], 'b');


        AGGREGATE_add_agr(temp1,temp0);
        AGGREGATE_add_agr(temp1,temp2);
        //AGGREGATE_print(temp1);
        AGGREGATE_attrib(&agA[i], temp1);
        AGGREGATE_print(&agA[i]);
}

printf("############################\n");
// border condition
AGGREGATE_ini(&agA[0]);strcpy(&agA[0].var_name[0],"f2");
AGGREGATE_add_point(&agA[0], &xt0[0]);
strcpy(&agA[0].point_list[0].var_name[0],"F2");

for(i=1;i<DOMAIN_SIZE-3;i++){
        AGGREGATE_times_COE(temp0, &agA[i-1], 'b');
        AGGREGATE_times_COE(temp1, &agA[i], 'a');
        AGGREGATE_times_COE(temp2, &agA[i+1], 'b');
```

```
                AGGREGATE_add_agr(temp1,temp0);
                AGGREGATE_add_agr(temp1,temp2);
                //AGGREGATE_print(temp1);
                AGGREGATE_attrib(&agB[i], temp1);
                AGGREGATE_print(&agB[i]);
        }
        return 0;
}
```

APÊNDICE B – DETAILS OF SELECTED EXPERIMENTS

Table 19: Details of 1D sym. experiments with NB 50, 6000 iters. of the original operator, RIS 1 and 2.

Side	RIS	NT	TB	time	EGF	EFLOCC
73728000	1	24	1	35.225	46.8	0.65
73728000	1	24	2	20.187	81.7	1.13
73728000	1	24	4	13.392	123	1.70
73728000	1	24	5	12.478	132	1.83
73728000	1	24	10	10.898	151	2.10
3072000	2	1	1	7.369	9.32	3.11
3072000	2	1	2	6.714	10.2	3.41
3072000	2	1	4	6.201	11.1	3.69
3072000	2	1	5	6.043	11.4	3.79
3072000	2	1	10	5.807	11.8	3.94
6144000	2	2	1	7.080	19.4	3.23
6144000	2	2	2	6.509	21.1	3.52
6144000	2	2	4	6.069	22.6	3.77
6144000	2	2	5	6.005	22.9	3.81
6144000	2	2	10	5.846	23.5	3.92
9216000	2	3	1	8.214	25.1	2.79
9216000	2	3	2	6.761	30.5	3.39
9216000	2	3	4	6.227	33.1	3.68
9216000	2	3	5	6.140	33.5	3.73
9216000	2	3	10	5.895	34.9	3.88
12288000	2	4	1	10.168	27	2.25
12288000	2	4	2	7.208	38.1	3.18
12288000	2	4	4	6.259	43.9	3.66
12288000	2	4	5	6.224	44.1	3.68
12288000	2	4	10	5.917	46.4	3.87
15360000	2	5	1	12.496	27.5	1.83
15360000	2	5	2	7.853	43.7	2.91
15360000	2	5	4	6.443	53.3	3.55
15360000	2	5	5	6.182	55.5	3.70
15360000	2	5	10	5.907	58.1	3.87
18432000	2	6	1	14.819	27.8	1.54
18432000	2	6	2	8.904	46.3	2.57
18432000	2	6	4	6.391	64.5	3.58
18432000	2	6	5	6.212	66.3	3.68
18432000	2	6	10	5.951	69.2	3.85
36864000	2	12	1	15.889	51.9	1.44
36864000	2	12	2	9.704	84.9	2.36
36864000	2	12	4	6.705	123	3.41
36864000	2	12	5	6.444	128	3.55
36864000	2	12	10	6.070	136	3.77
73728000	2	24	1	17.467	94.3	1.31
73728000	2	24	2	10.855	152	2.11
73728000	2	24	4	7.410	222	3.09
73728000	2	24	5	7.089	232	3.23
73728000	2	24	10	6.398	258	3.58

Table 20: Details of 1D sym. experiments with NB 50, 6000 iters. of the original operator, RIS 3 and 4.

Side	NT	TB	RIS	time	EGF	EFLOCC	RIS	time	EGF	EFLOCC
3072000	1	1	3	5.851	11.7	3.91	4	5.367	12.8	4.26
3072000	1	2	3	5.648	12.2	4.05	4	5.066	13.6	4.52
3072000	1	4	3	5.411	12.7	4.23	4	4.923	13.9	4.65
3072000	1	5	3	5.381	12.8	4.25	4	4.876	14.1	4.69
3072000	1	10	3	5.279	13	4.34	4	4.850	14.2	4.72
6144000	2	1	3	5.592	24.6	4.09	4	5.209	26.4	4.39
6144000	2	2	3	5.478	25.1	4.18	4	5.102	26.9	4.49
6144000	2	4	3	5.361	25.6	4.27	4	5.044	27.2	4.54
6144000	2	5	3	5.337	25.7	4.29	4	4.989	27.5	4.59
6144000	2	10	3	5.267	26.1	4.35	4	4.944	27.8	4.63
9216000	3	1	3	5.974	34.5	3.83	4	5.297	38.9	4.32
9216000	3	2	3	5.553	37.1	4.12	4	5.113	40.3	4.48
9216000	3	4	3	5.401	38.1	4.24	4	5.065	40.7	4.52
9216000	3	5	3	5.362	38.4	4.27	4	5.040	40.9	4.54
9216000	3	10	3	5.272	39.1	4.34	4	4.965	41.5	4.61
12288000	4	1	3	6.968	39.4	3.28	4	5.657	48.6	4.05
12288000	4	2	3	5.634	48.8	4.06	4	5.243	52.4	4.37
12288000	4	4	3	5.418	50.7	4.22	4	5.081	54.1	4.50
12288000	4	5	3	5.376	51.1	4.26	4	5.054	54.3	4.53
12288000	4	10	3	5.271	52.1	4.34	4	4.972	55.2	4.60
15360000	5	1	3	8.558	40.1	2.67	4	6.822	50.3	3.36
15360000	5	2	3	5.798	59.2	3.95	4	5.208	65.9	4.39
15360000	5	4	3	5.428	63.3	4.22	4	5.106	67.2	4.48
15360000	5	5	3	5.387	63.7	4.25	4	5.071	67.7	4.51
15360000	5	10	3	5.274	65.1	4.34	4	4.975	69	4.60
18432000	6	1	3	10.152	40.6	2.25	4	8.194	50.3	2.79
18432000	6	2	3	6.466	63.7	3.54	4	5.500	74.9	4.16
18432000	6	4	3	5.479	75.2	4.18	4	5.134	80.2	4.46
18432000	6	5	3	5.433	75.8	4.21	4	5.093	80.9	4.49
18432000	6	10	3	5.288	77.9	4.33	4	4.980	82.7	4.60
36864000	12	1	3	11.101	74.2	2.06	4	9.311	88.5	2.46
36864000	12	2	3	6.853	120	3.34	4	5.846	141	3.92
36864000	12	4	3	5.709	144	4.01	4	5.264	157	4.35
36864000	12	5	3	5.605	147	4.08	4	5.222	158	4.38
36864000	12	10	3	5.417	152	4.22	4	5.088	162	4.50
73728000	24	1	3	12.351	133	1.85	4	9.924	166	2.31
73728000	24	2	3	7.616	216	3.01	4	6.632	248	3.45
73728000	24	4	3	6.144	268	3.73	4	5.616	293	4.08
73728000	24	5	3	5.992	275	3.82	4	5.491	300	4.17
73728000	24	10	3	5.630	293	4.07	4	5.238	315	4.37

Table 21: Details of 1D sym. experiments with NB 50, 6000 iters. of the original operator, RIS 5 and 6.

Side	NT	TB	RIS	time	EGF	EFLOCC	RIS	time	EGF	EFLOCC
3072000	1	1	5	4.812	14.3	4.76	6	4.671	14.7	4.90
3072000	1	2	5	4.791	14.3	4.78	6	4.597	14.9	4.98
3072000	1	4	5	4.694	14.6	4.88	6	4.539	15.1	5.04
3072000	1	5	5	4.662	14.7	4.91	6	4.529	15.2	5.05
3072000	1	10	5	4.622	14.9	4.95	6	4.499	15.3	5.09
6144000	2	1	5	4.873	28.2	4.70	6	4.691	29.3	4.88
6144000	2	2	5	4.760	28.9	4.81	6	4.651	29.5	4.92
6144000	2	4	5	4.764	28.8	4.80	6	4.588	29.9	4.99
6144000	2	5	5	4.747	28.9	4.82	6	4.547	30.2	5.03
6144000	2	10	5	4.699	29.2	4.87	6	4.512	30.4	5.07
9216000	3	1	5	4.907	42	4.66	6	4.741	43.5	4.83
9216000	3	2	5	4.800	42.9	4.77	6	4.661	44.2	4.91
9216000	3	4	5	4.787	43	4.78	6	4.609	44.7	4.97
9216000	3	5	5	4.759	43.3	4.81	6	4.574	45	5.00
9216000	3	10	5	4.694	43.9	4.88	6	4.523	45.5	5.06
12288000	4	1	5	5.010	54.8	4.57	6	4.790	57.3	4.78
12288000	4	2	5	4.818	57	4.75	6	4.684	58.6	4.89
12288000	4	4	5	4.798	57.2	4.77	6	4.607	59.6	4.97
12288000	4	5	5	4.766	57.6	4.80	6	4.585	59.9	4.99
12288000	4	10	5	4.698	58.5	4.87	6	4.517	60.8	5.07
15360000	5	1	5	5.562	61.7	4.12	6	4.909	69.9	4.66
15360000	5	2	5	4.827	71.1	4.74	6	4.698	73.1	4.87
15360000	5	4	5	4.798	71.6	4.77	6	4.614	74.4	4.96
15360000	5	5	5	4.770	72	4.80	6	4.592	74.8	4.98
15360000	5	10	5	4.707	72.9	4.86	6	4.525	75.9	5.06
18432000	6	1	5	6.631	62.1	3.45	6	5.312	77.6	4.31
18432000	6	2	5	4.894	84.2	4.68	6	4.735	87	4.83
18432000	6	4	5	4.830	85.3	4.74	6	4.625	89.1	4.95
18432000	6	5	5	4.791	86	4.78	6	4.593	89.7	4.98
18432000	6	10	5	4.718	87.3	4.85	6	4.537	90.8	5.05
36864000	12	1	5	7.454	111	3.07	6	5.844	141	3.92
36864000	12	2	5	5.185	159	4.41	6	4.892	168	4.68
36864000	12	4	5	4.919	168	4.65	6	4.735	174	4.83
36864000	12	5	5	4.939	167	4.63	6	4.709	175	4.86
36864000	12	10	5	4.814	171	4.75	6	4.628	178	4.95
73728000	24	1	5	8.116	203	2.82	6	6.482	254	3.53
73728000	24	2	5	5.700	289	4.02	6	5.349	308	4.28
73728000	24	4	5	5.250	314	4.36	6	5.015	329	4.56
73728000	24	5	5	5.176	318	4.42	6	4.899	336	4.67
73728000	24	10	5	4.960	332	4.61	6	4.759	346	4.81

Table 22: Details of weak-scaling experiments for 2D, 5-point sym. without time-blocking, executing 120 iters. of the original operator, RIS 1 to 4.

size	Elements	Mbytes	TB	NB	iters	RIS	NT	time	EGF	EFLOCC
1024x1024	1048576	8	1	1	120	1	1	0.20552	3.42	1.14
2048x2048	4194304	32	1	1	120	1	2	0.40876	6.88	1.15
3072x3072	9437184	72	1	1	120	1	3	0.67345	9.4	1.04
4096x4096	16777216	128	1	1	120	1	4	0.83558	13.5	1.12
5120x5120	26214400	200	1	1	120	1	5	1.0676	16.5	1.1
6144x6144	37748736	288	1	1	120	1	6	1.3865	18.3	1.01
12288x12288	150994944	1152	1	1	120	1	12	2.7576	36.7	1.02
24576x24576	603979776	4608	1	1	120	1	24	9.7374	41.6	0.578
1024x1024	1048576	8	1	1	120	2	1	0.1403	5.01	1.67
2048x2048	4194304	32	1	1	120	2	2	0.28613	9.83	1.64
3072x3072	9437184	72	1	1	120	2	3	0.42283	15	1.66
4096x4096	16777216	128	1	1	120	2	4	0.55605	20.2	1.69
5120x5120	26214400	200	1	1	120	2	5	0.70662	24.9	1.66
6144x6144	37748736	288	1	1	120	2	6	0.8578	29.5	1.64
12288x12288	150994944	1152	1	1	120	2	12	1.7482	57.9	1.61
24576x24576	603979776	4608	1	1	120	2	24	3.8394	105	1.47
1024x1024	1048576	8	1	1	120	3	1	0.12226	5.75	1.92
2048x2048	4194304	32	1	1	120	3	2	0.24778	11.4	1.89
3072x3072	9437184	72	1	1	120	3	3	0.36526	17.3	1.92
4096x4096	16777216	128	1	1	120	3	4	0.48603	23.1	1.93
5120x5120	26214400	200	1	1	120	3	5	0.60505	29.1	1.94
6144x6144	37748736	288	1	1	120	3	6	0.76286	33.2	1.84
12288x12288	150994944	1152	1	1	120	3	12	1.5043	67.3	1.87
24576x24576	603979776	4608	1	1	120	3	24	3.418	118	1.65
1024x1024	1048576	8	1	1	120	4	1	0.11868	5.92	1.97
2048x2048	4194304	32	1	1	120	4	2	0.2373	11.9	1.98
3072x3072	9437184	72	1	1	120	4	3	0.35076	18	2
4096x4096	16777216	128	1	1	120	4	4	0.46958	24	2
5120x5120	26214400	200	1	1	120	4	5	0.5812	30.2	2.02
6144x6144	37748736	288	1	1	120	4	6	0.70992	35.7	1.98
12288x12288	150994944	1152	1	1	120	4	12	1.4419	70.2	1.95
24576x24576	603979776	4608	1	1	120	4	24	2.8777	141	1.95

Table 23: Details of strong-scaling experiments for 2D, 5-point sym. without time-blocking, RIS 1 (original operator).

size (elements)	NB	TB	iters	RIS	NT	time	EGF	EFLOCC
24576x24576	1	1	120	1	1	217.94	1.86	0.619
24576x24576	1	1	120	1	1	217.83	1.86	0.62
24576x24576	1	1	120	1	1	217.87	1.86	0.62
24576x24576	1	1	120	1	2	109.69	3.69	0.615
24576x24576	1	1	120	1	2	109.67	3.69	0.615
24576x24576	1	1	120	1	2	109.8	3.69	0.615
24576x24576	1	1	120	1	3	73.267	5.53	0.614
24576x24576	1	1	120	1	3	73.305	5.52	0.614
24576x24576	1	1	120	1	3	73.365	5.52	0.613
24576x24576	1	1	120	1	4	55.146	7.34	0.612
24576x24576	1	1	120	1	4	55.132	7.35	0.612
24576x24576	1	1	120	1	4	55.141	7.34	0.612
24576x24576	1	1	120	1	6	37.143	10.9	0.606
24576x24576	1	1	120	1	6	37.159	10.9	0.606
24576x24576	1	1	120	1	6	37.18	10.9	0.605
24576x24576	1	1	120	1	12	18.767	21.6	0.599
24576x24576	1	1	120	1	12	18.804	21.5	0.598
24576x24576	1	1	120	1	12	18.795	21.5	0.599
24576x24576	1	1	120	1	24	9.5913	42.2	0.586
24576x24576	1	1	120	1	24	9.6071	42.2	0.586
24576x24576	1	1	120	1	24	9.6163	42.1	0.585

Table 24: Details of strong-scaling experiments for 2D, 5-point sym. without time-blocking, RIS 2.

size (elements)	NB	TB	iters	RIS	NT	time	EGF	EFLOCC
24576x24576	1	1	120	2	1	83.74	4.84	1.61
24576x24576	1	1	120	2	1	83.824	4.83	1.61
24576x24576	1	1	120	2	1	83.843	4.83	1.61
24576x24576	1	1	120	2	2	41.728	9.71	1.62
24576x24576	1	1	120	2	2	41.751	9.7	1.62
24576x24576	1	1	120	2	2	41.71	9.71	1.62
24576x24576	1	1	120	2	3	27.989	14.5	1.61
24576x24576	1	1	120	2	3	27.997	14.5	1.61
24576x24576	1	1	120	2	3	27.959	14.5	1.61
24576x24576	1	1	120	2	4	21.143	19.2	1.6
24576x24576	1	1	120	2	4	21.138	19.2	1.6
24576x24576	1	1	120	2	4	21.122	19.2	1.6
24576x24576	1	1	120	2	6	14.34	28.2	1.57
24576x24576	1	1	120	2	6	14.319	28.3	1.57
24576x24576	1	1	120	2	6	14.307	28.3	1.57
24576x24576	1	1	120	2	12	7.2818	55.6	1.54
24576x24576	1	1	120	2	12	7.2421	55.9	1.55
24576x24576	1	1	120	2	12	7.2709	55.7	1.55
24576x24576	1	1	120	2	24	3.7555	108	1.5
24576x24576	1	1	120	2	24	3.7293	109	1.51
24576x24576	1	1	120	2	24	3.7973	107	1.48

Table 25: Details of strong-scaling experiments for 2D, 5-point sym. without time-blocking, RIS 3.

size (elements)	NB	TB	iters	RIS	NT	time	EGF	EFLOCC
24576x24576	1	1	120	3	1	76.029	5.33	1.78
24576x24576	1	1	120	3	1	76.276	5.31	1.77
24576x24576	1	1	120	3	1	76.247	5.31	1.77
24576x24576	1	1	120	3	2	37.7	10.7	1.79
24576x24576	1	1	120	3	2	37.7	10.7	1.79
24576x24576	1	1	120	3	2	37.86	10.7	1.78
24576x24576	1	1	120	3	3	25.204	16.1	1.79
24576x24576	1	1	120	3	3	25.165	16.1	1.79
24576x24576	1	1	120	3	3	25.316	16	1.78
24576x24576	1	1	120	3	4	18.94	21.4	1.78
24576x24576	1	1	120	3	4	18.892	21.4	1.79
24576x24576	1	1	120	3	4	19.011	21.3	1.78
24576x24576	1	1	120	3	6	12.64	32	1.78
24576x24576	1	1	120	3	6	12.609	32.1	1.78
24576x24576	1	1	120	3	6	12.703	31.9	1.77
24576x24576	1	1	120	3	12	6.409	63.2	1.76
24576x24576	1	1	120	3	12	6.4083	63.2	1.76
24576x24576	1	1	120	3	12	6.4387	62.9	1.75
24576x24576	1	1	120	3	24	3.2702	124	1.72
24576x24576	1	1	120	3	24	3.3287	122	1.69
24576x24576	1	1	120	3	24	3.2752	124	1.72

Table 26: Details of strong-scaling experiments for 2D, 5-point sym. without time-blocking, RIS 4.

size (elements)	NB	TB	iters	RIS	NT	time	EGF	EFLOCC
24576x24576	1	1	120	4	1	66.314	6.11	2.04
24576x24576	1	1	120	4	1	66.367	6.1	2.03
24576x24576	1	1	120	4	1	66.353	6.1	2.03
24576x24576	1	1	120	4	2	33.294	12.2	2.03
24576x24576	1	1	120	4	2	33.294	12.2	2.03
24576x24576	1	1	120	4	2	33.298	12.2	2.03
24576x24576	1	1	120	4	3	22.192	18.2	2.03
24576x24576	1	1	120	4	3	22.202	18.2	2.03
24576x24576	1	1	120	4	3	22.227	18.2	2.02
24576x24576	1	1	120	4	4	16.643	24.3	2.03
24576x24576	1	1	120	4	4	16.655	24.3	2.03
24576x24576	1	1	120	4	4	16.649	24.3	2.03
24576x24576	1	1	120	4	6	11.098	36.5	2.03
24576x24576	1	1	120	4	6	11.103	36.5	2.03
24576x24576	1	1	120	4	6	11.104	36.5	2.03
24576x24576	1	1	120	4	12	5.5768	72.6	2.02
24576x24576	1	1	120	4	12	5.6053	72.3	2.01
24576x24576	1	1	120	4	12	5.6018	72.3	2.01
24576x24576	1	1	120	4	24	2.892	140	1.94
24576x24576	1	1	120	4	24	2.8303	143	1.99
24576x24576	1	1	120	4	24	2.8928	140	1.94

APÊNDICE C – FURTHER COMMENTS ON THE RELATION OF ASLI AND KERNEL CONVOLUTION TO THE LITERATURE

As commented in Chapter 2.5, a given work is dismissive of its similarity to ASLI, while at the same time misstating some key concepts. Some key passages of (KORAEI; FATEMI; JAHRE, 2019) are discussed here. These passages deepen the understanding of how ASLI relates to the literature, and studying them might prove beneficial.

1. "One example [of a stencil strategy for CPU or GPU] is ASLI, which is similar to DCMI as it creates a new stencil operator that covers multiple time-steps by convolving the operator with itself. Although this approach enables data reuse within a cone, it suffers from the same redundant computation issue as CA [cone-based architecture], because it does not enable reuse between cones. The DCMI strategy [...] avoids redundant computation [...]". (KORAEI; FATEMI; JAHRE, 2019).

    (a) ASLI does not prescribe a cone-like structure as stated by Koraei et al. They are probably referring to Figure 2 above, which also had appeared in (JANUARIO *et al.*, 2016), but this figure exemplifies a state-of-the-art approach to multi-threading, instead of ASLI. ASLI can be used in conjunction with other state-of-the-art approaches, but such approaches are not a distinctive characteristic of ASLI, therefore they cannot be a distinction between DCMI and ASLI.

(b) It had been briefly shown on (JANUARIO *et al.*, 2016) how to benefit from reuses that would emerge only with ASLI or Kernel Convolution in 3D cases, whereas (JANUARIO; PARK; ROSENBURG, **US Patent** 9 916 678, Mar. 13, 2018; JANUARIO; PARK; ROSENBURG, **US Patent App.** 14/986,195, Jul. 6, 2017) had exemplified such reuses for 2D cases. DCMI does not benefit from such sources of reuse, incurring multiple redundancies that could have been avoided. This contradicts the statement “DCMI [...] avoids redundant computation".

(c) Section 4.1 exemplifies that an ASLI kernel of convolution degree 3 that employs these previously published reuses can save 18 FLOPs in the calculation of an asymmetric 2D. This accounts for $18/49 = 37\%$ reuse that DCMI misses. ASLI makes it less necessary to care with multi-threading schemes for 2D, so much so that the 2D codes in this work (Chapter 8) do not employ any technique similar to Figure 2, again showing that it is false that “[ASLI] suffers from the same redundant computation issue as CA".

(d) It is a category error to compare Figure 2 above with Figure 4 of (KORAEI; FATEMI; JAHRE, 2019). Figure 2 depicts **tb applications** of the computational kernel (convolved or not) that is supposed to be multi-thread’ized, whereas Figure 4 of Koraei et al. depicts the computation skipped within **one application** of the kernel.

(e) Figure 2 thus relates to the kind of concern Koraei et al. will have when using multiple instances of their FPGA in parallel, perhaps turning the FPGA approach more on-pair with ASLI for CPU’s. Boxed time-blocking was used in Chapter 7 to reduce the overhead of thread synchronization, thus saving time and power consumption.

(f) For comparison, the 1D experiments (Chapter 7) use 3,072,000 points per core, one thread per core. If we were to use a 2D domain of similar sto-

rage size, this would have side 1752. If we were to apply the boxed time-blocking approach to an ASLI kernel of convolution degree 3, with time-blocked parameter $tb = 2$, the redundancy would be extra $12 \times \text{nb} + 24$ redundant output elements. With one thread per core, this would mean an overhead of only 0.46% (cf. the 37% reuse that DCMI misses, item 1.c).

2. “Our algorithm first identifies the number of result values affected by a single input, which is 2RD + 1 with stencil pattern radius R and iteration depth D. Second, it creates an input array of this length with a single one as the middle element and zeros elsewhere. Then, we apply the stencil pattern S to this input array over D iterations. The outcome of this procedure is the aggregate coefficient for each affected result value." (KORAEI; FATEMI; JAHRE, 2019).

   (a) The process described above is much similar to the Influence Table, which first appeared in (JANUARIO; PARK; ROSENBURG, **US Patent** 9 916 678, Mar. 13, 2018; JANUARIO; PARK; ROSENBURG, **US Patent App.** 14/986,195, Jul. 6, 2017). Influence Tables are here discussed at length in Chapters 3, 4, and 6.

   (b) Similarly, they employ the term “aggregate" (as in the A of ASLI), whereas the “influence" of Influence Table finds counterparts in “affect(ed)" or “effect" in items 2, 2.b.i, and 2.b.ii:

      i. “After D iterations, the impulse array contains the coefficients that will compute the result component for each affected output when multiplied with the input element."(KORAEI; FATEMI; JAHRE, 2019).

      ii. “At design time, we determine the effect a single input element will have on all its reachable output elements (i.e., compute the effective coefficients)". (ibid.)

   (c) However, the process described is not accurate. As commented on (JANUARIO; PARK; ROSENBURG, **US Patent** 9 916 678, Mar. 13, 2018; JANUARIO; PARK; ROSENBURG, **US Patent App.** 14/986,195, Jul. 6, 2017) and as explained in detail here, this process generates the **flipped** representation of the new coefficients. Missing the flipped nature of Influence Tables has no practical implications when the original operator is symmetric, becoming a problem only when asymmetric operators are evaluated,

as is the case in the work of Koraei et al.

(d) The flipped nature of the influence table is clearly missed on line “Coeff[i] = T[i]" of Algorithm 2 of (KORAEI; FATEMI; JAHRE, 2019).

3. “Other boundary conditions can be implemented by changing the OCM and DCE controllers appropriately." (KORAEI; FATEMI; JAHRE, 2019).

   (a) It is not that simple to compute the elements close to the borders with convolved coefficients. Multiple elements close to the borders require multiple distinct sets of distinct modified coefficients. It is also necessary to take into account the function of the border condition (see Chapter 6.10). Therefore, a possible approach for Koraei et al. would be to add some complexity to design time to calculate all the border coefficients, or to update the border elements without using the convolved kernel.

4. “We propose a novel ISL [iterative stencil loop] acceleration scheme called Direct Computation of Multiple Iterations (DCMI) that improves upon prior work by pre-computing the effective stencil coefficients after a number of iterations at design time" (KORAEI; FATEMI; JAHRE, 2019).

   (a) The above statement indicates the authors deem that their main novelty coincides with the kernel convolution introduced by ASLI (JANUARIO *et al.*, 2016), which they implement on FPGA. Indeed, they describe ASLI as doing precisely that in items 4.a.i and 4.a.ii below. In item 4.a.iii, they again asseverate that the **key idea** is the kernel convolution of ASLI:

      i. “The ASLI approach can be used to achieve the same result by mathematically convolving the stencil operator with itself D times and then computing the effective coefficients." (KORAEI; FATEMI; JAHRE, 2019).

      ii. “ASLI is an application-level technique that creates a new stencil operator that covers multiple time-steps by convolving the original stencil operator with itself two or more times." (ibid.)

      iii. “We provide a fundamentally new approach for simultaneously exploiting the temporal and spatial parallelism of ISLs in FPGAs. The key idea is to compute the effective coefficients after D time-steps at design time, and use these coefficients to generate all possible partial results for a pre-defined number of input elements at runtime." (KORAEI; FATEMI; JAHRE, 2019).

5. Other similarities can be found on the names: “. . . over **Multiple** Iterations (DCMI)" and “Aggregate . . . Loop Iteration (ASLI)"; and on metrics proposed: “(G)EPS, (giga-) element iterations per second" in (KORAEI; FATEMI; JAHRE, 2019) and "EGS, Equivalent (Giga)-Stencil per second" in (JANUARIO *et al.*, 2016). Interestingly, the high-level performance analysis can be done with the metric “total time to solution", being an equivalent metric not a requirement. Finally, applications of stencil most often have coefficients that sum up to 1. Therefore, it is unusual to see any coefficient greater than 1 in the literature. However, integer coefficients greater than 1 had been used in (JANUARIO; PARK; ROSENBURG, **US Patent** 9 916 678, Mar. 13, 2018; JANUARIO; PARK; ROSENBURG, **US Patent App.** 14/986,195, Jul. 6, 2017) to make it easier to showcase the Influence Table and a process to derive convolved coefficients, before making their way into (KORAEI; FATEMI; JAHRE, 2019), where they were used to showcase how to calculate the new coefficients.