UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA DE SÃO CARLOS

*“ESTUDO DA DISTRIBUIÇÃO DE OCORRÊNCIAS DE
INSTRUÇÕES EM GRAFOS DE DEPENDÊNCIA DE DADOS DA
ARQUITETURA WOLF”*

ok

Wilson Bittencourt Vicentini

*Dissertação apresentada ao Instituto
de Física de São Carlos, Universidade
de São Paulo, para obtenção do Título
de Mestre em Ciências Física Aplica-
da - Subárea Física Computacional*

ORIENTADOR: Prof. Dr. Álvaro Garcia Neto

DEPARTAMENTO DE FÍSICA E INFORMÁTICA
SÃO CARLOS

1996

IFSC-USP SERVIÇO DE BIBLIOTECA E
INFORMAÇÃO

Vicentini, Wilson Bittencourt

Estudo da distribuição de ocorrências de instruções em grafos de dependência de dados da arquitetura Wolf / Wilson Bittencourt Vicentini. - São Carlos, 1996. 132 p.

Dissertação (Mestrado) - Instituto de Física de São Carlos, 1996.

Orientador: Prof. Dr. Álvaro Garcia Neto.

1.Arquiteturas paralelas 2. Arquiteturas a fluxo de dados.

**UNIVERSIDADE
DE SÃO PAULO**

Instituto de Física de São Carlos

Av. Dr. Carlos Botelho, 1465
CEP 13560-250 - São Carlos - SP
Brasil

Fone (016) 272-6222

Fax (016) 272-2218

**MEMBROS DA COMISSÃO JULGADORA DA DISSERTAÇÃO DE MESTRADO DE WILSON
BITTENCOURT VICENTINI APRESENTADA AO INSTITUTO DE FÍSICA DE SÃO CARLOS,
DA UNIVERSIDADE DE SÃO PAULO, EM 16 DE OUTUBRO DE 1996.**

COMISSÃO JULGADORA:

Prof. Dr. Alvaro Garcia Neto/IFSC-USP

Prof. Dr. Carlos Antonio Ruggiero/IFSC-USP

Dr. Fredy João Valente/ICMSC-USP

Dedicatória

Para Ele...

que estabeleceu os Céus e a Terra;
que criou o homem com seus atributos;
que equilibra os astros com precisão e harmonia;
que pulsa no coração, nos concede o livre arbítrio,
o discernimento e a inteligência...

Ao Deus vivo, autor da minha fé,
minha exclusiva dedicação.

Agradecimentos

Aos meus pais, Wilson e Moêma, por terem me concedido a oportunidade de chegar até aqui e por acreditarem neste trabalho.

À Priscila, minha esposa, pelo amor, carinho e compreensão; também como profissional, pelo suporte técnico oferecido.

À Beatriz, minha irmã, pelo apoio dado.

Aos meus demais familiares, aos meus sogros, Bento e Lourdes, aos familiares da minha esposa, que também souberam incentivar-me nos momentos difíceis.

Ao Prof. Dr. Álvaro Garcia Neto, pela confiança e orientação, valiosas para a realização desta pesquisa e aperfeiçoamento da minha formação acadêmica.

Ao CNPq, pelo recurso financeiro concedido.

À Wladerez A. G. Caiado, da Seção de Pós-Graduação, pela atenção e disposição.

Aos funcionários do Laboratório de Instrumentação Eletrônica (LIE), Lírio e Marquinhos, pelo auxílio e consideração.

Aos amigos Odemir e Luís Eduardo, pela amizade e companheirismo de todas as horas.

À colega Patrícia Magna, pelo apoio técnico oferecido.

Às funcionárias da biblioteca do IFSC, pela colaboração prestada.

À Profª. Drª. Lília Maria Soares Marmorato, por atender-me tão prontamente.

À Iza D. Ferreira, pela atenção oferecida.

A todos que, direta ou indiretamente, contribuíram para o êxito desse trabalho.

Conteúdo

CAPÍTULO I.....	1
ARQUITETURAS PARALELAS BASEADAS EM FLUXO DE DADOS	1
1.1 <i>Introdução</i>	1
1.2 <i>Esquemas de Classificação de Arquiteturas de Computadores</i>	1
1.3 <i>Classificação de Flynn</i>	2
1.4 <i>Classificação de Duncan</i>	5
1.5 <i>Conceitos Básicos Sobre Arquiteturas Baseadas em Fluxo de Dados</i>	14
1.5.1 <i>O Modelo Estático</i>	18
1.5.2 <i>Reentrância em Arquiteturas a Fluxo de Dados Estática</i>	19
1.5.3 <i>O Modelo Dinâmico</i>	20
1.5.4 <i>Reentrância em Arquiteturas a Fluxo de Dados Dinâmica</i>	21
1.5.5 <i>Organização Dinâmica X Organização Estática</i>	21
CAPÍTULO II.....	23
IMPLEMENTAÇÕES DO MODELO A FLUXO DE DADOS	23
2.1 <i>Introdução</i>	23
2.2 <i>MIT Tagged Dataflow Architecture - MTTDA</i>	23
2.3 <i>Monsoon</i>	26
2.4 <i>SIGMA-1</i>	27
2.5 <i>EM-4 Dataflow Multiprocessor</i>	29
2.6 <i>Epsilon-2 Multiprocessor</i>	32
2.7 <i>P-RISC</i>	34
2.8 <i>TAM (Treaded Abstract Machine)</i>	35
2.9 <i>StarT-NG</i>	35
2.10 <i>Manchester Multi-Ring Dataflow Machine</i>	38
2.11 <i>Considerações Finais sobre Arquiteturas a Fluxo de Dados</i>	39
CAPÍTULO III	41
DESCRIÇÃO FUNCIONAL DA ARQUITETURA WOLF E DA LINGUAGEM SISAL.....	41
3.1 <i>Introdução</i>	41
3.2 <i>Descrição Funcional das Unidades do Wolf</i>	43
3.3 <i>Linguagem de Alto Nível para Fluxo de Dados</i>	45
3.3.1 <i>A Linguagem SISAL</i>	47
3.3.2 <i>Descrição da Linguagem SISAL</i>	47
3.3.3 <i>Estruturas de Dados em SISAL</i>	48
3.3.4 <i>Estruturas de controle de fluxo</i>	52
3.3.5 <i>O bloco let</i>	53
3.3.6 <i>Funções</i>	54
3.3.7 <i>Processo de Compilação</i>	54
CAPÍTULO IV	56
DESCRIÇÃO DO PROBLEMA ESTUDADO.....	56
4.1 <i>Discussão dos Problemas Identificados</i>	56
4.2 <i>Análises dos Códigos</i>	58
4.3 <i>Descrição da Ferramenta SOC</i>	60
CAPÍTULO V.....	64
ANÁLISE DOS RESULTADOS DA FERRAMENTA SOC.....	64
5.1 <i>Introdução</i>	64
5.2 <i>Função de Ackerman</i>	65
5.3 <i>Integração Binária</i>	67
5.4 <i>Fatorial</i>	70

<i>5.5 Multiplicação de Matrizes</i>	72
<i>5.6 N_queens</i>	75
<i>5.7 Dinâmica Molecular</i>	77
<i>5.8 Simple</i>	79
<i>5.9 Utilização de Instruções Durante a Análise Estática</i>	81
<i>5.10 Utilização de Seqüências de Instruções Durante a Análise Seqüencial</i>	87
<i>5.11 Utilização de Estruturas Durante a Análise Estrutural</i>	90
<i>5.12 Conclusões</i>	97
APÊNDICE A	100
APÊNDICE B	102
APÊNDICE C	115
APÊNDICE D	117
APÊNDICE E	119
BIBLIOGRAFIA	126

Lista de Figuras

FIGURA 1: SISD	2
FIGURA 2: MISD	3
FIGURA 3: SIMD	4
FIGURA 4: MIMD.....	5
FIGURA 5: CLASSIFICAÇÃO DE DUNCAN	6
FIGURA 6: OPERAÇÃO DE UMA ARQUITETURA VETORIAL DE REGISTRADOR PARA REGISTRADOR	7
FIGURA 7: ORGANIZAÇÃO DE PROCESSAMENTO DE UMA MEMÓRIA ASSOCIATIVA	9
FIGURA 8: OPERAÇÃO DE COMPARAÇÃO DE UMA MEMÓRIA ASSOCIATIVA.....	9
FIGURA 9: MECANISMO DE FUNCIONAMENTO DA ARQUITETURA HÍBRIDA MIMD/SIMD	13
FIGURA 10: GRAFO DE FLUXO DE DADOS	15
FIGURA 11: UM GRAFO DE BAIXO NÍVEL A FLUXO DE DADOS	17
FIGURA 12: ORGANIZAÇÃO BÁSICA DE UM MODELO A FLUXO DE DADOS ESTÁTICO.....	18
FIGURA 13: ARQUITETURA BASEADA NO MODELO A FLUXO DE DADOS DINÂMICO	20
FIGURA 14: ARQUITETURA MTTDA	25
FIGURA 15: ARQUITETURA MONSOON	27
FIGURA 16: ARQUITETURA SIGMA-1	29
FIGURA 17: BLOCO FORTEMENTE CONECTADO	31
FIGURA 18: ARQUITETURA EM-4	31
FIGURA 19: ARQUITETURA EPSILON-2	33
FIGURA 20: ARQUITETURA P-RISC	34
FIGURA 21: ARQUITETURA START-NG.....	37
FIGURA 22: ARQUITETURA MMDM	38
FIGURA 23: ARQUITETURA WOLF	42
FIGURA 24: PROCESSO DE COMPILAÇÃO EM SISAL	55
FIGURA 25: UTILIZAÇÃO DA NANOMEMÓRIA	57
FIGURA 26: EXEMPLO DE FORMATO DE ESTRUTURAS	60
FIGURA 27: MÓDULO PARA ANÁLISE ESTATICA.....	61
FIGURA 28: MÓDULO PARA ANÁLISE SEQUENCIAL	62
FIGURA 29: MÓDULO PARA ANÁLISE ESTRUTURAL.....	63
FIGURA 30 : EXEMPLO DE UM PROGRAMA EM FORMATO COD PARA A FUNÇÃO DE ACKERMAN.....	63
FIGURA 31: GRÁFICO PARA ANÁLISE ESTATICA DE ACKERMAN.COD	65
FIGURA 32: GRÁFICO PARA ANÁLISE SEQUENCIAL DE ACKERMAN.COD	66
FIGURA 33: GRÁFICO PARA ANÁLISE ESTRUTURAL DE ACKERMAN.COD.....	67
FIGURA 34: GRÁFICO PARA ANÁLISE ESTATICA DE BIN.COD	68
FIGURA 35: GRÁFICO PARA ANÁLISE SEQUENCIAL DE BIN.COD	69
FIGURA 36: GRÁFICO PARA ANÁLISE ESTRUTURAL DE BIN.COD	69
FIGURA 37: GRÁFICO PARA ANÁLISE ESTATICA DE FAT.COD	70
FIGURA 38: GRÁFICO PARA ANÁLISE SEQUENCIAL DE FAT.COD.....	71
FIGURA 39: GRÁFICO PARA ANÁLISE ESTRUTURAL DE FAT.COD	72
FIGURA 40: GRÁFICO PARA ANÁLISE ESTATICA DE MATRIX.COD.....	73
FIGURA 41: GRÁFICO PARA ANÁLISE SEQUENCIAL DE MATRIX.COD	74
FIGURA 42: GRÁFICO PARA ANÁLISE ESTRUTURAL DE MATRIX.COD.....	74
FIGURA 43: GRÁFICO PARA ANÁLISE ESTATICA DE N_QUEENS.COD.....	75
FIGURA 44: GRÁFICO PARA ANÁLISE SEQUENCIAL DE N_QUEENS.COD	76
FIGURA 45: GRÁFICO PARA ANÁLISE ESTRUTURAL DE N_QUEENS.COD.....	76
FIGURA 46: GRÁFICO PARA ANÁLISE ESTATICA DE DINÂMICA.COD	77
FIGURA 47: GRÁFICO PARA ANÁLISE SEQUENCIAL DE DINÂMICA.COD	78
FIGURA 48: GRÁFICO PARA ANÁLISE ESTRUTURAL DE DINÂMICA.COD	78
FIGURA 49: GRÁFICO PARA ANÁLISE ESTATICA DE SIMPLE.COD	79
FIGURA 50: GRÁFICO PARA ANÁLISE SEQUENCIAL DE SIMPLE.COD	80
FIGURA 51: GRÁFICO PARA ANÁLISE ESTRUTURAL DE SIMPLE.COD	80
FIGURA 52: GRAFO A FLUXO DE DADOS	81
FIGURA 53: COMPARAÇÃO SOBRE A UTILIZAÇÃO DA INSTRUÇÃO DUP	82

FIGURA 54: FRAGMENTO DE CÓDIGO COM A UTILIZAÇÃO DA INSTRUÇÃO SYN	83
FIGURA 55: COMPARAÇÃO SOBRE A UTILIZAÇÃO DA INSTRUÇÃO SYN	83
FIGURA 56: ESTRUTURA DE UMA CHAMADA DE FUNÇÃO	84
FIGURA 57: COMPARAÇÃO SOBRE A UTILIZAÇÃO DA INSTRUÇÃO SAN	85
FIGURA 58: UTILIZAÇÃO DA INSTRUÇÃO BRR	85
FIGURA 59: COMPARAÇÃO SOBRE A UTILIZAÇÃO DA INSTRUÇÃO BRR	86
FIGURA 60: UTILIZAÇÃO DA INSTRUÇÃO PRL	86
FIGURA 61: COMPARAÇÃO SOBRE A UTILIZAÇÃO DA INSTRUÇÃO PRL	87
FIGURA 62: COMPARAÇÃO SOBRE A UTILIZAÇÃO DA SEQUÊNCIA DUP-DUP(N)	88
FIGURA 63: COMPARAÇÃO SOBRE A UTILIZAÇÃO DA SEQUÊNCIA DUP-SAN(L)	88
FIGURA 64: COMPARAÇÃO SOBRE A UTILIZAÇÃO DA SEQUÊNCIA DUP-SYN(L)	89
FIGURA 65: COMPARAÇÃO SOBRE A UTILIZAÇÃO DA SEQUÊNCIA BRR-DUP(N)	89
FIGURA 66: COMPARAÇÃO SOBRE A UTILIZAÇÃO DA SEQUÊNCIA DUP-PRL(N)	90
FIGURA 67: COMPARAÇÃO SOBRE A UTILIZAÇÃO DA ESTRUTURA L -> L-	91
FIGURA 68: COMPARAÇÃO SOBRE A UTILIZAÇÃO DA ESTRUTURA N -> L-	91
FIGURA 69: COMPARAÇÃO SOBRE A UTILIZAÇÃO DA ESTRUTURA N -> L-L	92
FIGURA 70: GRAFICO PARA ANÁLISE ESTÁTICA X EXECUÇÃO DINÂMICA DE QUICKSOR.SIS	94
FIGURA 71: GRÁFICO PARA ANÁLISE ESTÁTICA X EXECUÇÃO DINÂMICA PARA GAUSS.SIS	95
FIGURA 72: GRÁFICO PARA ANÁLISE ESTÁTICA X EXECUÇÃO DINÂMICA PARA MMS.SIS	96

Resumo

As arquiteturas *Dataflow* apresentam características baseadas em um modelo cujo controle de execução é feito pela disponibilidade dos dados, possibilitando a exploração de paralelismo implícito em um programa. Para contextualizar o trabalho, são apresentadas as principais implementações em fluxo de dados.

Este trabalho estuda a distribuição da ocorrência de instruções em Grafos de Dependência de Dados da arquitetura Wolf desenvolvida no IFSC. O estudo realizado concluiu que existe uma grande concentração na utilização das instruções por parte dessa arquitetura: das cento e uma instruções da máquina, cinco apresentam, em média, de trinta por cento a cinquenta por cento do total. Essa popularidade de instruções ocorre em todas as formas de análise e para todos os *benckmarks* investigados. Os resultados desse trabalho fornecem uma base para a implementação mais eficiente dessas arquiteturas, e indicam que qualquer mecanismo que explore essa popularidade, produzirá ganhos.

Abstract

The Dataflow architecture presents characteristics based on a model in which execution control is done by the data availability, making the exploitation of the implicit paralelism possible in a program. The main implementation in data flux are showed, having in view to put it inside the work's context.

This work studies the distribution of the occurrence of the instructions in Graphs of Data Dependence of the Wolf architecture developed at IFSC. The present study concluded that there is a large concentration on the utilizations of the instructions in this architecture: from the one hundred and one instructions of the machine, five present, in average, from thirty to fifty per cent of the total. This popularity of instructions occurs in all forms of analysis and for all investigated benckmarks. The results of this work furnish a base for a more efficient implementation of these architectures and they indicate that any mechanism which explores this popularity, will produce gains.

Capítulo I

Arquiteturas Paralelas Baseadas em Fluxo de Dados

1.1 Introdução

Este capítulo discorre sobre conceitos fundamentais de classificação de arquiteturas a fluxo de dados e algumas de suas implementações.

1.2 Esquemas de Classificação de Arquiteturas de Computadores

A classificação das arquiteturas de computadores se torna importante a medida que novas tecnologias vão se proliferando. Com os esquemas de classificação, é possível situar um tipo de máquina, segundo suas características, no contexto de uma classe de computadores de arquitetura paralela. A partir dessa contextualização é possível comparar o comportamento da máquina proposta, com outras máquinas do mesmo tipo, evitando um dos maiores problemas na pesquisa em arquiteturas paralelas, que é a comparação entre sistemas de características diferentes. Dentre os principais esquemas de classificação de computadores, pode-se citar:

- Flynn (baseada em conceitos de fluxo de dados e fluxo de instruções) [Fly66]

- Feng (classificação orientada a performance) [Hwa+84]
- Händler (paralelismo versus “*pipelining*”) [Hwa+84]
- Gajski (descrição sobre o ponto de vista do usuário) [Amo+80]
- Duncan (taxonomia de alto nível que mantém elementos da taxonomia de Flynn) [Dun90]

Esse estudo irá considerar as classificações de Flynn e Duncan porque a primeira é mais utilizada e a segunda, mais abrangente.

1.3 Classificação de Flynn

A classificação de M. J. Flynn [Fly66], apresentada em 1966, é bastante difundida. Essa taxonomia sugere uma classificação baseada em conceitos de fluxos de dados e fluxos de instruções.

A classificação de Flynn, divide as arquiteturas de computadores em 4 categorias:

- **SISD (*Single Instruction Single Data*)**: Nessa categoria está incluída a arquitetura convencional de Von Neumann, a qual se refere ao tratamento de um único dado sendo executado por uma única instrução.

Figura 1: SISD

- **MISD (*Multiple Instruction Single Data*)**: Nessa categoria vários processadores aplicam diferentes instruções sobre um único dado. Alguns autores consideram o *pipeline* [Alm+94] como exemplo dessa arquitetura.

MC: Mem. Compartilhada
 FD: Fluxo de Dados
 FI: Fluxo de Instruções
 MM: Módulo de Memória
 UP: Unidade de Processamento
 UC: Unidade de Controle

Figura 2: MISD

- **SIMD (*Single Instruction Multiple Data*)**: Usa múltiplos processadores executando a mesma instrução sobre dados simultâneos e distintos. Para isso há uma unidade de controle que distribui a mesma instrução para os processadores, que as executam sobre esses dados distintos. Existe também uma rede de interconexão, responsável pela comunicação entre os processadores ou entre os processadores e as memórias. Podemos citar como exemplos dessa classificação de arquitetura, o UI ILLIAC-IV [Hwa+84, Amo+88], o ASPRO [DiG+81], a série *Cray* [Amo+88], o CDC *Cyber 205* [Hwa+84, Amo+88, Alm+94], o ES/9000 *Vector Facility* [Alm+94], o *MasPar* MP-1 [Gow+93], o *Goodyear* MPP [Alm+94], o ICL DAP [Hoc+88, Alm+94], o CMU/GE WARP [Alm+94].

Figura 3: SIMD

- **MIMD (*Multiple Instruction Multiple Data*)**: Usa múltiplos processadores supervisionados por suas correspondentes unidades de controle executando ao mesmo tempo várias instruções independentes sobre vários dados distintos. A maioria dos sistemas multiprocessadores e multicomputadores podem ser classificados nessa categoria. São exemplos desse tipo de arquitetura o *Columbia DADO Production-System Machine* [Sto+82], o KSR1 da *Kendell Square Research* [Gow+93], o *Intel iPSC* [Amo+88, Alm+94], o NCUBE/ten e o NCUBE2 [Alm+94], o *Intel Paragon*, o HSCP PAX [Alm+94], o U. *Utah Rediflow* [Alm+94] e as arquiteturas a fluxo de dados, como a *Manchester Dataflow Machine*.

MC: Mem. Compartilhada
FD: Fluxo de Dados
FI: Fluxo de Instruções
MM: Módulo de Memória
UP: Unidade de Processamento
UC: Unidade de Controle

Figura 4: MIMD

1.4 Classificação de Duncan

De acordo com Duncan [Dun90], a proliferação de novas tecnologias para processamento paralelo, juntamente com o desenvolvimento de novos tipos de hardware e arquitetura (sistólica e hipercubo), tecnologias de interconexão (topologias de chaveamento multiestágio), tem sido um obstáculo para que se compreenda os tipos de arquiteturas paralelas existentes e a relação de um tipo com outro para definição de um esquema de classificação. Diversas definições tem sido propostas para arquiteturas paralelas. Segundo Duncan, o problema central para a especificação e a definição de uma taxonomia, principalmente no contexto de arquiteturas paralelas modernas, decorre do seguinte conjunto de imperativas:

- Exclusão de arquiteturas que incorporem somente mecanismos de paralelismo de baixo nível.

- Manter elementos da taxonomia de Flynn baseada em fluxo de instruções e fluxo de dados.

- Incluir processadores vetoriais *pipelined* e outras arquiteturas que possuam uma dificuldade de classificação dentro da taxonomia de Flynn.

A taxonomia de Duncan é baseada na tentativa de solução dessas imperativas e para isso utiliza categorias de alto nível para delinear as aproximações entre as principais classificações de arquiteturas paralelas.

Figura 5: Classificação de Duncan

- **Processadores vetoriais *pipelined***

As arquiteturas de processamento vetorial surgiram no fim dos anos 60 e início dos anos 70 para apoiar diretamente uma grande quantidade de cálculos vetoriais e matriciais. Processadores vetoriais são caracterizados por múltiplas unidades funcionais *pipelined*. Qualquer operação aritmética e *booleana* para dois vetores podem ser operadas concorrentemente. Desta maneira, a arquitetura providencia operações vetoriais paralelas através da sequencialidade de processamento de elementos vetoriais através de unidades funcionais *pipelined* com a saída dos resultados de uma unidade desse *pipelined*. Podem ser citados como exemplos dessa arquitetura o *Cray-1* [Amo+88], o *Fujitsu VP-200* [Amo+88], *Control Data Cyber 205* [Hwa+84, Amo+88, Alm+94], *Cray X-MP/4* [Amo+88].

Figura 6: Operação de uma arquitetura vetorial de registrador para registrador

- **Arquiteturas SIMD**

As arquiteturas SIMD empregam uma unidade de controle central, múltiplos processadores e uma rede de interconexão para comunicação processador a processador e/ou processador com a memória. A unidade de controle envia uma única instrução para todos os processadores, que executam a instrução sobre dados distintos. A rede de interconexão permite que o resultado calculado por um processador possa ser comunicado a outro processador para uso como operando em uma instrução subsequente.

- **Arquiteturas de Processadores matriciais**

Processadores matriciais para processamento em arquiteturas SIMD podem freqüentemente ser empregados para uma grande quantidade de cálculos científicos, como processamento de imagens e alguns tipos específicos de simulações. Vários esquemas podem ser usados para providenciar a comunicação entre processadores e memória, sendo o “*crossbar*” o mais popular. São exemplos dessa arquitetura o *Illiac-IV* [Hwa+84, Amo+88], o *ICL’s Distributed Array Processor* [Alm+94] e o *Connection Machine da Thinking Machines* [Alm+94].

- **Arquiteturas de Memórias Associativas**

Computadores construídos em torno de memórias associativas constituem um tipo distinto de arquitetura SIMD que utiliza uma comparação lógica especial para controlar dados armazenados em seus volumes. Nesse tipo de memória, um controlador de programa lê e executa instruções, invocando uma matriz controladora especial que associa instruções de memória encontradas. Registradores especiais habilitam o controlador de programa e a memória associativa para dados compartilhados.

Figura 7: Organização de processamento de uma memória associativa

A figura 8 descreve uma operação de comparação de uma coluna de uma arquitetura genérica. Uma parte do registrador de comparação contém o valor que deve ser encontrado. Todos os elementos de processamento associativos começam pela coluna de memória especificada e comparam o conteúdo dos 4 bits consecutivos nessa coluna em relação ao conteúdo do registrador de comparação, colocando um bit no registrador A para indicar se aquela coluna contém ou não o conteúdo exato. É exemplo dessa arquitetura o *Loral's Associative Processor* ou ASPRO [DiG+81].

Figura 8: Operação de comparação de uma memória associativa

- **Arquiteturas sistólicas**

No início dos anos 80, H.T. Kung da *Carnegie Mellon University* propôs as arquiteturas sistólicas para solucionar problemas específicos. Arquiteturas sistólicas são multiprocessadores *pipelined* nos quais os dados fluem em um ritmo padrão da memória para uma rede de processadores, ligados por uma rede de interconexão local, e retornam de volta para a memória, de forma síncrona.

A sincronização do *pipeline* é feita através de um relógio global. Durante cada intervalo de tempo, os processadores executam uma seqüência de instruções pequena e invariável. A partir da entrada de um dado no *pipeline*, esse é passado de processador para processador sem o uso da memória. Dessa forma, diminui-se o problema da sobrecarga de memória. Arquiteturas sistólicas apresentam bom comportamento para aplicações específicas pelo êxito significativo do processamento paralelo, através do fluxo de dados em *pipelining* com múltiplos processadores, além de minimizar o sobrecarregamento da memória, já mencionado. São exemplos dessa arquitetura o Warp da *Carnegie Mellon* [Alm+94] e o *Configurable High Parallel* (CHIP) [Alm+94].

- **Arquiteturas MIMD**

Arquiteturas MIMD empregam múltiplos processadores que executam instruções independentes sobre dados distintos. O processo de execução de um *software* numa arquitetura MIMD é sincronizado pela passagem de mensagens através de uma rede de interconexão ou pelo acesso a dados em memórias compartilhadas. Arquiteturas MIMD são compostas por computadores assíncronos, caracterizados pela descentralização de um *hardware* de controle central.

- **Arquiteturas de Memória Distribuída**

Arquiteturas de memória distribuída conectam nós de processamento de um processador autônomo e sua memória local com uma rede de interconexão processador a processador. Os nós compartilham os dados através da passagem de

mensagens explícitas através da rede de interconexão. Várias topologias de redes de interconexão tem sido propostas para suportar a expansão da arquitetura e providenciar uma performance eficiente para programas paralelos com diferentes modelos de comunicação entre os processadores. Os sistemas de memória distribuída são classificados como fracamente acoplados ou multicomputadores. São exemplos dessas topologias: a topologia em anel, malha, árvore, hipercúbica, árvore mapeada e malha reconfigurável. Podemos citar como exemplo, o DADO2 da *Columbia University* [Dun90] e o TMC *Connection Machine* CM-5 [Alm+94] com topologias de interconexão em árvore [Dun90], o *Ametek Series 2010* e o Ncube/10 de topologia hipercúbica [Dun90], o protótipo *Configurable Highly Parallel Computer*, ou CHIP da *Lawrence Snyder* [Dun90] e o *Transputer* [Tan92], ambos de topologia de interconexão em malha e o IBM 9076 SP-1 [Alm+94].

- **Arquiteturas de Memória Compartilhada**

Arquiteturas de memória compartilhada executam a coordenação entre os processadores, providenciando uma memória compartilhada global, com um espaço de endereçamento único para cada processador. Arquiteturas de memória compartilhada não têm os mesmos problemas encontrados em arquiteturas por passagem de mensagem. Entretanto outros problemas precisam ser solucionados, como a sincronização de acesso aos dados e a coerência de cache. A coordenação de processadores com variáveis compartilhadas requer mecanismos de sincronização atômica para prevenir que um processo disponha dos dados antes que outro processo termine de atualizá-los. Além disso, cada processador também possui uma memória local usada como cache. Múltiplas cópias dos mesmos dados da memória compartilhada podem existir na memória cache de vários processadores em um determinado momento. Não violar a semântica de memória compartilhada é o problema referente a coerência de cache, que deve providenciar novas versões dos dados de cada processador envolvido, para que um processo que esteja sendo executado não obtenha dados antigos (*stale data*). Podemos citar, como exemplos, algumas alternativas para conectar processadores com uma memória compartilhada que são as interconexões em barra, as interconexões “*crossbar*” e as redes de

interconexão multiestágio. São exemplos dessa arquitetura o *Alliant FX/8* que utiliza uma interconexão crossbar [Dun90], e o BBN (Bolt, Beranek e Newman) *Butterfly* que se utiliza de uma rede de interconexão multiestágio [Dun90].

- **Arquiteturas Não Convencionais**

Arquiteturas híbridas MIMD/SIMD, máquinas de redução, matrizes de frente de onda e arquiteturas a fluxo de dados são difíceis de enquadrar em uma taxonomia na área de arquiteturas paralelas. Cada uma dessas arquiteturas é derivada de princípios MIMD de operação assíncrona e manipulação concorrente de múltiplas instruções e múltiplos dados. Entretanto, cada uma dessas arquiteturas é baseada em princípios de organização distintos para serem englobados em características MIMD globais.

- **Arquiteturas MIMD/SIMD**

Uma variedade de arquiteturas híbridas experimentais foram construídas durante a década de 80 permitindo que uma parte de uma arquitetura MIMD pudesse ser controlada por uma arquitetura SIMD padrão. Usando uma árvore estruturada para ilustrar o conceito, é possível verificar a passagem de mensagens coma a base da arquitetura MIMD/SIMD. A relação *master/slave* de controle de uma arquitetura SIMD e os processadores podem ser mapeados para uma relação de nós descendentes de uma sub-árvore. Quando o nó do processador raiz de uma sub-árvore opera como um controlador SIMD, ele transmite instruções aos nós descendentes para que executem as instruções em dados de memória locais. São exemplos dessa arquitetura o DADO [Alm+94, Dun90], o Non-Von [Alm+94, Dun90].

Figura 9: Mecanismo de funcionamento da arquitetura híbrida MIMD/SIMD

- **Arquiteturas a Fluxo de Dados**

Será discutida em detalhes na seção 1.5 deste capítulo e no capítulo II.

- **Arquiteturas de Redução**

Arquiteturas de redução ou dirigidas a demanda implementam um paradigma de execução na qual a instrução é habilitada para execução quando seus resultados são requeridos como operandos para outra instrução, já habilitada para execução. A maior parte das pesquisas em arquiteturas de redução começaram na década de 70 para explorar um novo paradigma de execução paralela que providenciasse suporte para aplicações em linguagens de programação funcionais. São exemplos dessa

arquitetura a *Newcastle Reduction Machine* [Dun90], *North Carolina Cellular Tree Machine* [Dun90] e a *Utah Aplicative Multiprocessing System* [Dun90].

- **Arquiteturas de Matrizes de Frente de Onda**

Processadores de matrizes de frente de onda combinam o *pipelining* de dados sistólicos com a paradigma de execução a fluxo de dados assíncrona. Arquiteturas de frente de onda são máquinas sistólicas caracterizadas por processadores modulares e por uma rede de interconexão local. Entretanto matrizes de frente de onda substituem o relógio global e o atraso de tempo explícito usada para sincronização de dados em *pipelining* por um ‘aperto’ assíncrono no tempo, como um mecanismo para coordenação do movimento dos dados entre os processadores. As máquinas de frente de onda desfrutam de certas vantagens em relação as máquinas sistólicas como a grande escalabilidade e a programação mais simples. Como exemplos, podemos citar as máquinas construídas pela *Jonhs Hopkins University* e pela *Standard Telecommunications Company* [Dun90].

1.5 Conceitos Básicos Sobre Arquiteturas Baseadas em Fluxo de Dados

De acordo com a taxonomia de Flynn, a arquitetura baseada no paradigma de Fluxo de Dados se enquadra na categoria MIMD, e de acordo com a taxonomia de Duncan, ela se enquadra numa especificação própria de arquiteturas a fluxo de dados. A principal característica dessa arquitetura é a relação de dependência intrínseca entre a seqüência de execução das instruções com a disponibilidade dos dados, o que se reflete num paralelismo existente nos algoritmos, sem a interferência do programador. Isto quer dizer que uma instrução somente pode ser executada assim que todos os seus operandos estiverem disponíveis, sem um controle centralizado da execução do programa. Um grafo representa o programa e todas as dependências de dados em relação as instruções. Esse mesmo grafo é formado por nós que representam uma

instrução e por arcos que conectam esses mesmos nós, formando assim os caminhos pelos quais os dados necessitam percorrer até a instrução subsequente.

Figura 10: Grafo de fluxo de dados

Dessa maneira, na arquitetura a fluxo de dados, o paralelismo é explorado automaticamente. A essência do modelo computacional *dataflow* é a de que ele define meios diretos para execução concorrente através dos grafos. Portadores de valores de dados, conhecidos como fichas, transversam os grafos. Cada instrução determinará individualmente se irá ou não ser executada de acordo com a disponibilidade dos dados de entrada requeridos. Como cada decisão é feita localmente sobre cada instrução, o padrão global de execução é totalmente assíncrono. A figura 11 ilustra duas importantes características de grafos cíclicos que cobrem o modelo estático e dinâmico, que serão vistos detalhadamente a seguir. A primeira característica é a natureza acíclica de execução do programa *dataflow* que pode causar condições de disputa em gráficos cíclicos. Por exemplo, nessa mesma figura, o ciclo que define o próximo valor do contador de *loops* *i* pode disputar a dianteira desses ciclos calculando *x* e a integral, deste modo causando confusão entre fichas pertencentes a sucessivos ciclos desse *loop* na entrada dos dados da esquerda da instrução *BRANCH*. De modo idêntico esta situação deve ser evitada (no modelo estático, isto é realizado impedindo que uma instrução seja executada quando qualquer dos seus arcos de

saída estejam ocupados com fichas), ou deverá haver um meio de distinguir duas fichas (como no modelo dinâmico, mostrado a seguir).

A segunda característica que pode-se observar é a de que duas chamadas de funções *f* podem ocorrer concorrentemente, além do que, sucessivos ciclos do *loop* podem introduzir uma concorrência adicional. Se algumas funções são implementadas no gráfico equivalente como sub-rotinas, com o corpo da função podendo ser compartilhado por todas as chamadas de funções, então é provável que várias ativações do corpo da função possam ocorrer simultaneamente. Nas arquiteturas *dataflow*, ambos os problemas podem ser solucionados através de um esquema denominado rotulação de fichas. Cada ficha possui um campo extra denominado *tag*, que distingue entre todas as possíveis ativações de uma determinada parte do código. O esquema de *tagging* ou rotulação é geral e pode garantir a implementação de todo tipo de funções recursivas paralelas. Um esquema alternativo é a cópia do código, representando o corpo da função, toda vez que esta é chamada, expandindo o programa em tempo de execução.

Figura 11: Um grafo de baixo nível a fluxo de dados

A implementação dessa arquitetura pode ser baseada em dois modelos: o modelo estático e o modelo dinâmico.

1.5.1 O Modelo Estático

O modelo estático foi proposto por Jack Dennis e seu grupo de pesquisa no MIT [Den80]. A organização geral da Máquina a Fluxo de Dados Estática é a seguir:

Figura 12: Organização básica de um modelo a fluxo de dados estático

A Memória de Programas contém gabaritos de instruções que representam nós em uma representação a fluxo de dados. Cada gabarito de instrução contém um código de operando, um campo para esses operandos, e os endereços de destinos. Para determinar a disponibilidade dos operandos, esses campos devem conter bits extras (*Presence Bits*). Quando esta condição é verificada, a Unidade de Atualização envia o endereço da instrução disponível à Unidade de Busca. A Unidade de Busca o

encontra e envia o resultado dessa operação em uma ficha contendo o correspondente código de operação, os dados, e a lista de destino para a Unidade de Processamento e também limpa os bits extras contidos no campo de operandos. A Unidade de Processamento realiza a operação indicada na ficha sobre os dados também contidos na ficha, cria uma ficha de resultados, e envia para a Unidade de Atualização. A Unidade de Atualização armazena cada resultado no campo do operando apropriado e checa a presença de bits extras para determinar qual atividade está ou não disponível. Com isso, vale dizer, em outras palavras, que no modelo estático todos os nós da representação são lidos na memória durante a inicialização do programa, com a execução de somente uma instância do nó por vez.

1.5.2 Reentrância em Arquiteturas a Fluxo de Dados Estática

Em *dataflow* estático, a reentrância é manuseada pelo método do reconhecimento e pelo método *lock* [Alm+94]. O método *lock* obriga que cada arco tente, em um dado instante, operar uma única ficha, permitindo um pequeno paralelismo através de um *pipeline*. Por exemplo, se cada iteração de um nó é um gráfico acíclico e conseqüentemente não pode causar execução reentrante, então é necessário somente garantir que não haverá iterações antes que o predecessor a complete.

Dataflow estático é definido previamente como o armazenamento de não mais do que uma ficha por arco. Também pode ser definido através de uma adição à regra de disponibilidade, onde a instrução de um nó somente pode ser disparada se todos os seus operandos estão disponíveis e se todos os arcos de saída estão livres de fichas. O método do reconhecimento é uma alternativa para obrigar essa regra de disparo restrita. O método do reconhecimento requer que um sinal seja enviado de volta para o nó quando uma ficha de um dos arcos de saída é consumida pelo próximo nó desse arco e se o nó espera por esses sinais antes de produzir mais fichas.

1.5.3 O Modelo Dinâmico

O modelo dinâmico foi proposto por Arvind no MIT e por Gurd e Watson da Universidade de Manchester [Arv+86].

Figura 13: Arquitetura baseada no modelo a fluxo de dados dinâmico

Nesse modelo, um conjunto de fichas é recebido pela Unidade de Emparelhamento, que é uma memória com um conjunto de fichas que estão à espera para emparelhar-se com outras. A operação básica da Unidade de Emparelhamento é causar o encontro de fichas com um mesmo rótulo ou *tag*. Se o encontro ocorre, a ficha correspondente é extraída da Unidade de Emparelhamento e as fichas emparelhadas serão passadas para a Unidade de Busca. Se nenhum emparelhamento ocorreu, as fichas são armazenadas na Unidade de emparelhamento a espera de uma ficha “parceira”. Na Unidade de Busca, os rótulos da ficha par única identificam uma instrução para ser buscada na Memória de Programa.

1.5.4 Reentrância em Arquiteturas a Fluxo de Dados Dinâmica

Em uma arquitetura com rotulação de fichas, o rótulo é o mecanismo usado para manejar o reuso de código, como explica Arvind: “Cada ficha em uma máquina *dataflow* estática é obrigada a alcançar o endereço da instrução pela qual é destinada. Isto já é um rótulo. Supondo que, em adição a especificação de um nó destino, que o rótulo também especifique uma particular regra de disparo do nó, então, duas fichas participam do mesmo disparo de um nó somente se seus rótulos forem os mesmos. Outro caminho de modelar os rótulos, é simplesmente manter uma estrutura lógica FIFO para cada arco, indiferente da ordem de chegada física das fichas” [Alm+94]. Em vez dos rótulos, um método alternativo para *dataflow* dinâmico é a cópia do código, proposta pela família Form IV, desenvolvida por Dennis e seus colaboradores no MIT.

1.5.5 Organização Dinâmica X Organização Estática

Arquiteturas a Fluxo de Dados podem ser classificadas em sistemas centralizados ou distribuídos, baseados na organização de suas instruções na memória [Lee+93]. Numa organização centralizada, o custo de transferência da comunicação de uma ficha com outra é independente da alocação atual da instrução na Memória de Programa. Em uma organização distribuída, as instruções são distribuídas ao longo dos elementos de processamento (PEs), fazendo com que os custos de comunicação entre esses elementos sejam altos.

O Modelo a Fluxo de Dados Estático tem um mecanismo simples para detecção de nós disponíveis. No modelo estático apenas uma ficha por vez pode ocupar determinado arco. Bits (ou um contador) são utilizados para determinar quando todos os operandos requeridos estão disponíveis. O modelo estático tem uma performance prejudicada em procedimentos com construções iterativas e reentrância. Isto pode ser notado em casos de grafos de reentrância, gerando uma regra de disparo, através de fichas de controle, responsável pela habilitação das instruções.

Este controle é realizado através de sinais de reconhecimento. Este sinal de reconhecimento assegura que nenhum arco conterá mais que uma ficha. O método de reconhecimento poderá transformar o código de reentrância em um grafo equivalente que permitirá a execução de iterações consecutivas em uma forma de *pipeline*. Entretanto, esta transformação causará um aumento expressivo no número de arcos e fichas.

Neste caso, a maior vantagem do modelo dinâmico sobre o modelo estático está na alta performance computacional que pode ser obtida através da rotulação de fichas para que múltiplas fichas existam simultaneamente em qualquer arco, possibilitando com isso a exploração de um paralelismo maior. No modelo dinâmico, várias instâncias do mesmo nó, criadas em tempo de execução, são executadas simultaneamente. Por exemplo, um ciclo é dinamicamente formado em tempo de execução pela criação de múltiplas instâncias do corpo do ciclo e através da execução dessas instâncias concorrentemente, através da diferenciação das mesmas por mecanismo de coloração ou rotulagem.

Capítulo II

Implementações do Modelo a Fluxo de Dados

2.1 Introdução

Esse capítulo estuda conceitos de sistemas computacionais que empregam a filosofia de fluxo de dados apresentada no capítulo anterior. As arquiteturas descritas são: MIT Tagged Dataflow Architecture - MTTDA, Monsoon, EM-4 Dataflow Multiprocessor, Epsilon-2 Multiprocessor, P-RISC, StarT-NG, MFDM (Máquina a Fluxo de Dados de Manchester) e a arquitetura Wolf.

2.2 MIT Tagged Dataflow Architecture - MTTDA

O MTTDA (*MIT Tagged-Token Dataflow Architecture*) é uma arquitetura proposta por Arvind e seu grupo de pesquisa do MIT. Trata-se de uma máquina com uma arquitetura a Fluxo de Dados dinâmico e assíncrono, com 64 processadores idênticos, interligados por uma rede de conexão n-cúbica, conectando também unidades de armazenamento denominadas Estruturas-I ou *I-structure* [Arv+90] . Cada elemento processador é um processador voltado a Fluxo de Dados e cada unidade *I-structure* possui um controlador responsável pela coordenação de leituras e

escritas na mesma. A arquitetura explora o conceito de coloração de fichas para reutilização de grafos.

As operações de escrita e leitura seguem uma disciplina. Existe uma área para armazenamento de dados, onde cada posição contém espaços para bits extras, responsáveis pela identificação de três estados possíveis: *present*, *absent* e *waiting*.

No processo de leitura, a ficha de leitura deve trazer o endereço da posição a ser lida, e o rótulo para a instrução que está a espera do valor. Se, nesse dado instante, o estado da posição for *absent* ou *waiting*, a leitura é indeferida e o rótulo é armazenado na posição encontrada, senão, caso o estado da posição seja *present*, o dado está capacitado a ser lido. No processo de escrita, se o estado da posição descrito na ficha de escrita for *absent*, o valor é escrito e o dado alterado para *present*. Se, no entanto, o estado encontrado for *waiting*, o valor da mesma forma é escrito na posição endereçada e o estado alterado para *present*, com o valor sendo emitido para todos os endereços armazenados naquela posição. Por fim, se o estado for *present*, significa que já existe algum valor naquela posição.

Para a programação dessa máquina foi utilizada uma linguagem funcional com paralelismo implícito desenvolvida no MIT, denominada Id [Ari90, Alm+94].

Figura 14: Arquitetura MTTDA

2.3 Monsoon

A arquitetura de Monsoon é baseada em multiprocessadores voltados a arquitetura a Fluxo de Dados dinâmica desenvolvidos no MIT em conjunto com a Motorola [Pap+90, Ang+95]. A formulação de Moonson começou como uma evolução do *MIT Tagged-Token Dataflow Architecture* (MTTDA). Diferentemente da MTTDA, o esquema de emparelhamento é implementado mais eficientemente, baseado no modelo de Armazenamento de Fichas Explícito (ETS), que é uma aproximação simplificada para execução a Fluxo de Dados. A idéia básica do modelo ETS é a eliminação de um grande e complexo processo de pesquisa associativa usada em arquiteturas a Fluxo de Dados Dinâmica para emparelhamento de fichas parceiras. Uma ficha em um modelo ETS consiste em um ponteiro e um número que indicam o contexto e os locais dos processadores. Desta forma, nomes de ativação são restritos ao processador para o qual foram alocados, e essa instanciação é executada integralmente no processador. Com isso, a regra de disparo é implementada através de uma busca direta e indexada da ficha parceira, restrita as fichas encontradas em um mesmo processador. Os multiprocessadores Monsoon consistem em uma coleção de processadores, baseados no modelo ETS conectados por uma rede de chaveamento de pacotes multiestágio (*Multistage Packet-Switching Network*) e diversos módulos de memória de *I-structure* interligados. Cada processador consiste em oito estágios que realizam algumas formas de operações dispostas em *pipeline*.

Os processadores dessa máquina foram desenvolvidos para processar 6 milhões de fichas por segundo. A partir de 1993, o grupo de pesquisa do MIT começou a trabalhar em conjunto com a Motorola para desenvolver protótipos de multiprocessadores Monsoon, com a expectativa de serem capazes de processar cerca de 10 milhões de fichas por segundo, com uma grande memória para armazenamento.

Figura 15: Arquitetura Monsoon

2.4 SIGMA-1

O projeto SIGMA-1 teve início em 1982, no Japão, no *Electrotechnical Laboratory (ETL/MITI)*, através do grupo de Toshitsugu Yuba [Shi+83, Yub+87]. O principal objetivo era pesquisar e estabelecer uma base tecnológica para construção de

supercomputadores dataflow altamente paralelos. Em janeiro de 1988, tornou-se operacional um computador com mais de 128 elementos processadores síncronos com memórias de estruturas divididos em 32 grupos conectados por uma rede hierárquica de dois níveis. Cada grupo é composto por quatro elementos processadores, quatro elementos de estrutura, e uma rede de comunicação local. Todos os grupos são conectados por meio da rede local de comunicação.

Os principais aspectos do projeto SIGMA-1 são as seguintes:

- O elemento processador trabalha na forma de um curto “*pipeline*” de dois estágios, reduzindo o “*gap*” entre a performance média e a performance máxima.
- O emparelhamento dos operandos ocorre ao mesmo tempo da busca de instruções, oferecendo eficiência e alta velocidade de sincronização.
- Uma rede global com uma função distribuída de leituras dinâmicas reduz degradações causadas por desequilíbrios na leitura.
- Uma arquitetura paralela com o propósito especial para manutenções, testes e depurações, produz altas velocidades de E/S (para os padrões de 1980) e realiza medidas de performance precisas.

Há algumas observações a serem feitas no design dessa arquitetura, em particular no sistema de memória do SIGMA-1:

- Em diferentes funções da memória, como naquelas necessárias para o emparelhamento, as instruções são alocadas em módulos de memórias distintos, contribuindo para otimização do *design*, mas causando uma ineficiente utilização dos módulos de memória.
- Não há memória hierárquica, havendo a necessidade de uma grande capacidade de memória.
- O gerenciamento da memória é feito principalmente por *hardware*.

A máquina SIGMA-1 usa a linguagem de programação DFC-2 (Dataflow-C) [Yub+87].

Figura 16: Arquitetura Sigma-1

2. 5 EM-4 Dataflow Multiprocessor

O EM-4 começou a ser desenvolvido em 1989 pelo grupo de pesquisa do Electrotechnical Laboratories do Japão [Lee+93]. O EM-4 é uma máquina com multiprocessadores baseados em Fluxos de Dados com um alto nível de paralelismo e com uma estrutura que objetiva mais ou menos 1.000 processadores baseados no projeto SIGMA-1 [Sak+90]. Suas metas são:

- Simplificar a arquitetura pelo desenho de uma pastilha de circuito integrado única baseado numa arquitetura RISC e um esquema de emparelhamento direto.

- Introduzir e utilizar um modelo de arcos fortemente conectados a um avançado modelo a Fluxo de Dados.

O EM-4 utiliza um modelo de arcos fortemente conectados para otimizar a execução a Fluxo de Dados. Neste modelo, arcos em um grafo a Fluxo de Dados são divididos em arcos normais e arcos fortemente conectados. O subgrafo a Fluxo de Dados cujos nós são unidos pelos arcos fortemente conectados é chamado de bloco fortemente conectado.

A estratégia de execução para blocos fortemente conectados é aquela de que esses blocos fortemente conectados atuam como um “macro-nó”, ao qual inclui várias instruções que são executadas como se fossem um nó básico único.

Há várias vantagens no modelo de arcos fortemente conectados. Primeiro, um fluxo de controle mais simples disposto em *pipeline* pode ser utilizado. Segundo, o ciclo de instrução pode ser reduzido pela utilização de um arquivo contendo registros para armazenagem de fichas em um bloco fortemente conectado. Isto é possível desde que dados no arquivo não sejam violados por outros dados. Terceiro, a sobrecarga de comunicação é reduzida porque não há transferência de fichas em um bloco fortemente conectado. Quarto, o emparelhamento pode ser muito mais facilmente realizado dentro de um bloco que está sendo executado, pelo motivo de que a cada momento somente uma ativação de um bloco fortemente conectado é executada. Os processadores do protótipo do EM-4 consistem em um módulo de memória e um único processador denominado EMC-R integrado a uma única pastilha de circuito integrado de arquitetura RISC. Os processadores são organizados em 16 grupos de elementos processadores interconectados por uma rede multiestágio ômega formada por unidades de chaveamento com controle descentralizado.

Figura 17: Bloco fortemente conectado

Figura 18: Arquitetura EM-4

2.6 Epsilon-2 Multiprocessor

O multiprocessador a Fluxo de Dados Epsilon-2 foi proposto por Grafe e Hoch na Sandia National Laboratories em 1990 [Gra+89, Lee+93]. O projeto do Epsilon-2 é baseado no modelo a Fluxo de Dados Dinâmico, com suporte a recursividade e *loopings*, emparelhamento direto, processamento *pipeline* e características RISC, sendo baseado no projeto anterior denominado Epsilon-1. Essa máquina implementa um modelo computacional híbrido que combina características seqüenciais de von Neumann com paralelismo de granularidade fina.

O sistema é uma coleção de módulos de processamento interconectados através de uma rede de interconexão global. Cada módulo consiste de um processador com características RISC, uma estrutura de memória, e uma porta de E/S, interconectados através de uma chave crossbar 4x4.

O Epsilon-2 executa código compilado de linguagens Id, SISAL e Fortran.

Figura 19: Arquitetura Epsilon-2

2.7 P-RISC

P-RISC (*Parallel-Reduced Instruction Set Computer*) foi proposto por Nikhil e Arvind no MIT em 1989 [Nik+89]. A idéia básica por trás do P-RISC é a forte influência da arquitetura híbrida Von Neumann/ Iannucci's *dataflow*. O P-RISC coloca um conjunto de instruções como RISC e generaliza para um “*multithreaded*” para obter a capacidade de realizar uma granulação fina a Fluxo de Dados. Desde que essa arquitetura seja uma extensão da arquitetura de Von Neumann, ela pode agir tanto como uma arquitetura convencional, como uma arquitetura a Fluxo de Dados. Ela pode ser considerada como uma máquina a Fluxo de Dados que providencia uma capacidade completa de softwares, como uma máquina convencional [Lee+93].

O multiprocessador P-RISC consiste de processadores e Memória de Estrutura de Elementos conectados por uma rede de interconexão.

Figura 20: Arquitetura P-RISC

2.8 TAM (Treaded Abstract Machine)

O projeto TAM foi proposto por Culler em 1991 na Universidade da Califórnia, Berkeley [Cul+91]. A idéia principal que está por trás do projeto TAM, está em colocar toda sincronização e requisitos de gerenciamento de memória para execução paralela explícita, na forma de granularidade fina e debaixo do controle do compilador. Isso alivia a demanda ocupada pelo *hardware* da máquina, dando ao compilador a responsabilidade para relacionar módulos de programas de baixo nível e otimizar o uso de recursos críticos de processamento [Lee+93].

No TAM, o recurso de armazenagem básico são os blocos de códigos e os *frames* ou porções de memória. Os blocos de código consistem em uma coleção de *threads*, que representam seqüências de instruções. Sempre que um bloco de código é invocado, um *frame* é alocado. Cada *frame* contém uma área para armazenamento para a colocação de valores de argumentos associados com o código de bloco invocado. Instruções podem referenciar áreas de armazenamento sem o *frame* corrente ou registradores. É responsabilidade do compilador determinar estaticamente o tamanho do *frame*, para que cada bloco de código possa ser usado corretamente pelas áreas de armazenamento ou registradores. Um protótipo de um grupo de instruções para o TAM denominado TL0 (*Treaded Language Version 0*), tem sido desenvolvido pela Universidade da Califórnia. Programas em Id foram primeiramente compilados para produzir TL0, e em seguida para um código de máquina nativo, usando C.

2.9 StarT-NG

O StarT-NG é projeto de uma junção do MIT com a Motorola para construir uma máquina com passagem de mensagens de alta performance para sistemas comerciais [Chi+95]. Cada *site* da máquina consiste de um PowerPC 620 baseado em Multiprocessadores Simétricos Motorola (SMPs) rodando o sistema operacional AIX 4.1. Cada processador é conectado para uma baixa latência, através de uma rede com uma larga banda que é diretamente conectada através de um código à nível de

usuário. O StarT-NG pode possuir até 4 placas com processadores que podem ser conectados com a memória principal através de uma interconexão *crossbar*. O StarT-NG pode substituir uma até quatro placas por uma interface NES (*network-endpoint-subsystem*), cada uma contendo um processador 620, 4 Mbytes de cache L2 e uma unidade de interface para rede ou NIU (*network interface unit*) que permite que o 620 se comunique com um chip roteador de rede desenvolvido no MIT. Cada uma dessas placas NES em cada site podem endereçar dispositivos de captura (ACD) que permitem que processadores designados em cada *site* monitorem e respondam a todo tipo de transações.

Em adição a capacidade de rápidas passagens de mensagem, a máquina possui um suporte experimental para que exista a coerência de cache com o compartilhamento da memória através dos *sites*. Este suporte objetiva explorar o sistema operacional e o gerenciamento de memória virtual em adição a um sistema de memória compartilhada com coerência de cache, além da organização necessária de *hardware* para prevenir *deadlocks* e um modelo de memória agradável para programação. Em relação a este fato, quando a máquina requer memória, para estar globalmente coerente, um processador em cada *site* é dedicado ao suporte a memória compartilhada, senão ele pode ser usado normalmente para tarefas computacionais normais. O StarT-NG é uma arquitetura de passagem de mensagens fortemente influenciada pelas arquiteturas *dataflow* através da ênfase no envio de pequenas mensagens com um baixo *overhead*, para tentar se obter a execução paralela com uma granularidade fina. As máquinas que influenciaram o projeto *T foram a máquina Monsoon, o EM-4, o J-Machine [Dal+92] e o M-Machine [Hei+94].

usuário. O StarT-NG pode possuir até 4 placas com processadores que podem ser conectados com a memória principal através de uma interconexão *crossbar*. O StarT-NG pode substituir uma até quatro placas por uma interface NES (*network-endpoint-subsystem*), cada uma contendo um processador 620, 4 Mbytes de cache L2 e uma unidade de interface para rede ou NIU (*network interface unit*) que permite que o 620 se comunique com um chip roteador de rede desenvolvido no MIT. Cada uma dessas placas NES em cada site podem endereçar dispositivos de captura (ACD) que permitem que processadores designados em cada *site* monitorem e respondam a todo tipo de transações.

Em adição a capacidade de rápidas passagens de mensagem, a máquina possui um suporte experimental para que exista a coerência de cache com o compartilhamento da memória através dos *sites*. Este suporte objetiva explorar o sistema operacional e o gerenciamento de memória virtual em adição a um sistema de memória compartilhada com coerência de cache, além da organização necessária de *hardware* para prevenir *deadlocks* e um modelo de memória agradável para programação. Em relação a este fato, quando a máquina requer memória, para estar globalmente coerente, um processador em cada *site* é dedicado ao suporte a memória compartilhada, senão ele pode ser usado normalmente para tarefas computacionais normais. O StarT-NG é uma arquitetura de passagem de mensagens fortemente influenciada pelas arquiteturas *dataflow* através da ênfase no envio de pequenas mensagens com um baixo *overhead*, para tentar se obter a execução paralela com uma granularidade fina. As máquinas que influenciaram o projeto *T foram a máquina Monsoon, o EM-4, o J-Machine [Dal+92] e o M-Machine [Hei+94].

Figura 21: Arquitetura StarT-NG

A possibilidade de poder contar com uma coleção de SMPs padrão e uma máquina com rápida passagem de mensagens com uma memória compartilhada com coerência de cache tornam o StarT-NG uma boa opção para incrementar a expansão das máquinas paralelas. Estão sendo utilizados para esta máquina, uma linguagem funcional com paralelismo implícito como o Id, e o pH, e linguagens *multithreaded* como Cid e Cilk.

2.10 Manchester Multi-Ring Dataflow Machine

Figura 22: Arquitetura MMDM

A MDFM (*Manchester Dataflow Machine*) é um projeto que teve início no ano de 1975 pelo grupo de John Gurd e Ian Watson da Universidade de Manchester [Gur+85]. Essa arquitetura é baseada no paradigma de fluxo de dados dinâmico através da rotulação de fichas as quais contêm os dados e que circulam através de uma estrutura em anel, interligadas a uma chave de entrada e saída (*I/O Switch*); responsáveis pela entrada e saída das fichas desse mesmo anel; e por quatro unidades (*Token Queue*, *Matching Unit*, *Node Store* e *Funcional Unit*), compondo um *pipeline*. Os dados circulam através desse anel contendo esses módulos e obtêm as instruções a que se destinam. A unidade denominada *Token Queue*, se utiliza de uma estrutura de dados baseado no mecanismo FIFO (*First In First Out*) e é responsável pelo controle do tráfego pelo anel. A *Matching Unit* ou unidade de emparelhamento, é o módulo mais crítico do sistema, necessitando de um grande espaço de armazenamento. Por isso, a *Matching Unit* foi projetada através da utilização de uma

técnica de “*hashing*”, para simular uma memória puramente associativa, cujo custos são altamente proibitivos. Nessa unidade, as fichas com mesmo destino e mesmo rótulo são emparelhadas, efetuando-se com isso a regra de disparo. A *Node Store*, ou unidade de instrução, é a responsável pelo armazenamento do código do programa em execução. Nesta unidade, as fichas emparelhadas e as fichas destinadas a instrução de uma só entrada buscam as instruções à qual estão endereçadas. Com isso as fichas contendo as instruções e os dados são mandadas para a *Funcional Unit*, onde são executadas.

2.11 Considerações Finais sobre Arquiteturas a Fluxo de Dados

A realização de projetos de computadores com o paradigma da arquitetura a Fluxo de Dados identificaram um conjunto de problemas técnicos que tem sido estudados nos últimos 15 anos, para que soluções definitivas possam ser alcançadas. Dentre os principais problemas existentes, podemos citar alguns [Hwa+84]:

- A necessidade do desenvolvimento de linguagens voltadas para o paradigma de fluxo de dados que possam ser de fácil uso e possam ser interpretadas pelo hardware da máquina.
- A decomposição de programas e a tarefa de designar os módulos desses programas para os processadores voltados a fluxo de dados.
- O controle e o suporte a comunicação de um grande número de processadores através de uma rede de interconexão de baixo custo. Para melhor explorar o paralelismo, com a execução concorrente de tarefas cada vez menores, as redes de interconexão se tornam em importantes canais de comunicação para a coordenação dessas tarefas.
- O manuseio eficiente de estruturas de dados complexas, como matrizes e vetores, em um ambiente voltados a fluxo de dados, que são responsáveis por um paralelismo excessivo.
- O desenvolvimento de um esquema de alocação de memória para suporte a computação voltada a fluxo de dados, visto que existe uma sobrecarga na utilização

de memória, devido a grande quantidade de dados, já que cada um desses dados é tratado individualmente. Isso ocorre também por causa da relação entre a parte útil de uma ficha e a parte destinada ao controle.

- A grande necessidade de conhecimento de linguagens funcionais voltadas a fluxo de dados, além de suporte do software, compilação voltada ao paradigma a fluxo de dados e novas metodologias de programação.

Capítulo III

Descrição Funcional da Arquitetura Wolf e da Linguagem SISAL

3.1 Introdução

A arquitetura a Fluxo de Dados Wolf foi proposta pelo Grupo de Instrumentação e Informática da USP e pelo Grupo de Fluxo de Dados da UNICAMP em 1992. Ela é uma arquitetura baseada no paradigma de fluxo de dados dinâmico com granularidade variável e deriva da Máquina a Fluxo de Dados de Manchester, incorporando soluções para diversos dos problemas identificados. A figura 23 mostra essa arquitetura.

Figura 23: Arquitetura Wolf

A arquitetura Wolf é composta de dez unidades interligadas, que trocam dados entre si através de fichas.

3.2 Descrição Funcional das Unidades do Wolf

A função de cada uma das dez unidades que compõem a arquitetura Wolf é:

- **Mecanismo de Entrada e Saída:** É o responsável pelo interfaceamento com o mundo exterior, enviando para os dispositivos de E/S as fichas de saída produzidas durante o processamento e enviando para as unidades internas as fichas contendo programas e dados.
- **Chave de Coleta :** É a responsável pelo recebimento das fichas provenientes das diferentes unidades conectadas em suas entradas (Unidade de Entrada e Saída, Unidade Fila de Fichas, Unidade de Processador Vetorial e Memória Vetorial, Unidade de Controle de Paralelismo e os anéis, compostos pelas Memórias de Instruções e suas respectivas Unidades Funcionais) e pelo posterior envio dessas fichas à Unidade de Emparelhamento (UE), através de duas saídas distintas. Uma delas transmite as fichas para a Memória de Dados (MD), se a mesma estiver disponível. A outra envia os endereços para a Unidade de Emparelhamento, se esta também estiver disponível. Caso haja indisponibilidade dessas unidades, as fichas são enviadas à Fila de Fichas, para uma posterior distribuição.
- **Fila de Fichas:** A fila de fichas é uma fila do tipo FIFO (*First In, First Out*), responsável pelo armazenamento das fichas que não puderam ser enviadas a Unidade de Emparelhamento, para evitar um possível sobrecarregamento dos anéis. Também evita *deadlocks*.
- **Unidade de Emparelhamento:** É a responsável pelo emparelhamento dos operandos de duas fichas de mesma coloração, dirigidas a um mesmo nó, ou por uma ficha do tipo *BYPASS* em conformidade com uma ficha do tipo *null*, formando dessa forma uma nova ficha do tipo par. Para realizar esse emparelhamento, é necessário o endereço recebido da Chave de Coleta, para busca da ficha parceira. Caso a ficha seja localizada, a ficha é colocada à disposição pela Memória de Dados, senão fica armazenada pela mesma, à espera do par correspondente.

- **Memória de Dados:** É a responsável pelo armazenamento dos dados, quando uma ficha parceira não é localizada pela Unidade de Emparelhamento, onde fica à espera da ficha correspondente. É também responsável pela disponibilidade dos dados para o emparelhamento, quando da chegada da ficha parceira, possibilitando a formação da ficha par (PA).
- **Chave de Distribuição:** Tem por função distribuir as fichas recebidas da Memória de Dados, ou então as fichas provenientes da Memória de Instruções, para distribuir para os anéis, compostos pela Memória de Instruções e sua Unidade Funcional correspondente, ou para Unidade de Processador Vetorial e Memória Vetorial (PVMV), ou ainda para a Unidade de Controle de Paralelismo, conforme o tipo da ficha.
- **Memória de Instruções:** Contém todas as informações necessárias sobre o nó para o qual a ficha par se destina, como por exemplo, à operação a ser efetuada sobre os dados da ficha par, bem como o destino do(s) resultado(s). De posse dessas informações, a Memória de Instruções constrói a ficha executável (EX) que contém as informações necessárias para que a instrução possa ser executada, enviando-a posteriormente à Unidade Funcional correspondente.
- **Unidade Funcional:** É a responsável pela execução da ficha recebida da Memória de Instruções. A ficha gerada pode ser de dois tipos: livre ou encadeada. As fichas livres são passadas para a Chave de Coleta para recircular no *pipeline*, enquanto as fichas encadeadas continuam nos registradores internos da Unidade Funcional que a gerou para uso posterior como operandos de instruções subsequentes. A Memória de Instruções contém informações que podem identificar nós como partes de uma cadeia. A Memória de Instruções inicia um ciclo de busca usando o endereço de destino de uma ficha executável produzida e pertencente a uma cadeia, para utilizar como endereço de busca com o intuito de produzir a próxima ficha executável dessa cadeia. Com isso, a ficha executável anterior que foi processada pela Unidade Funcional servirá como operando de entrada de uma nova ficha executável produzida pela Memória de Instruções.

- **Memória de Rascunho:** Contém os resultados produzidos pela Unidade Funcional, ou resultados intermediários.
- **Unidade Processador Vetorial e Memória Vetorial:** É uma unidade dedicada à tarefa de processar os dados estruturados de forma mais eficiente, com a diminuição da necessidade de índice de fichas nas próprias fichas, reduzindo o uso de memória. Pelo fato dessa unidade tratar do armazenamento e processamento de estrutura de dados, não é necessário que estes dados estejam circulando pelo *pipeline*, evitando o tráfego desnecessário. As unidades de Memória Vetorial podem armazenar pedidos de leitura de dados ainda não produzidos, tornando possíveis as execuções de operações de leitura e escrita em ordem inversa.
- **Unidade Controle de Paralelismo:** É responsável pelo controle do paralelismo excessivo, através de mecanismos que restrinjam o número de instanciações de uma mesma instrução.

3.3 Linguagem de Alto Nível para Fluxo de Dados

Não existe um consenso sobre qual seria a linguagem de programação mais adequada para este tipo de arquitetura. Pode-se dizer, entanto, que a característica mais importante é a de que essa linguagem favoreça o paralelismo implícito. A computação voltada a fluxo de dados é compatível com o uso de grafos de dependência na análise de um programa em um computador convencional. Uma linguagem eficiente de programação para compilação voltada a fluxo de dados deve ser capaz de expressar o paralelismo de um programa de forma natural sem necessidade de explicitação pelo programador, facilitando a interação entre a construção de algoritmos e a estrutura da máquina. Algumas propriedades numa linguagem a fluxo de dados são desejáveis, como localidade de efeitos, escalonamento vinculado e atribuição única, enquanto outras não, como é o caso dos efeitos colaterais [Ack82, Hwa+84]:

- **Localidade de Efeito:** Localidade de efeito é uma propriedade que consegue que uma instrução não tenha uma dependência de dados muito longa, fazendo com que as variáveis temporárias usadas para cálculos possam ser reutilizadas posteriormente.

- **Escalonamento Vinculado:** Essa é a característica que garante que uma instrução só será executada quando todos os seus operandos estiverem disponíveis. Desse modo, todas as informações necessárias para executar um programa estão contidas no grafo de fluxo de dados do programa. Desde que o grafo de fluxo de dados esteja em conformidade com o código de máquina para um computador a fluxo de dados, esta é a propriedade que torna uma linguagem adequada para utilização em um computador.

- **Atribuição Única:** Esta característica oferece o método necessário para promover o paralelismo em um programa. Esta regra é usada para impedir que um mesmo nome de variável possa aparecer no lado esquerdo de uma atribuição mais de uma vez na área de programa em que ela está ativa. Em outras palavras, um novo nome é escolhido para qualquer variável redefinida, e todas referências subsequentes são mudadas para o novo nome. Este conceito é exemplificado a seguir:

$X := P - Q$		$X := P - Q$
$X := X \times Y$	$\rightarrow$	$X1 := X \times Y$
$W := X - Y$		$W := X1 - Y$

As declarações da coluna da direita são feitas para satisfazer a propriedade da atribuição única. Ela facilita a detecção de paralelismo no programa. A propriedade da atribuição única oferece clareza e facilidade de verificação, ao qual geralmente superam a conveniência do reuso do mesmo nome.

- **Efeitos Colaterais:** Esta característica deve estar ausente das linguagens de programação a fluxo de dados, para garantir a consistência dos dados durante a execução do programa. Efeitos colaterais podem ocorrer de várias maneiras, como em uma *procedure* que modifica variáveis numa chamada de programa. Outro problema comum é a modificação de variáveis do programa principal através da passagem de parâmetros para procedimento por referência e não por valor. Para

evitar esses problemas a linguagem não deve suportar variáveis globais e o compilador deve ser capaz de analisar variáveis envolvidas, tendo em vista a restrição de seu alcance.

3.3.1 A Linguagem SISAL

SISAL (*Streams and Iteration in a Single Assignment Language*) é a principal linguagem utilizada pela máquina de Manchester e tem a mesma utilização dentro da arquitetura Wolf. Ela foi desenvolvida a partir da atuação conjunta da Digital Corporation, Lawrence Livermore National Laboratory, Universidade de Manchester e Universidade de Colorado. SISAL é uma linguagem funcional que não possui efeitos colaterais, não possui variáveis globais, não permite a reatribuição de valores à variáveis e possui uma forma de execução explícita.

3.3.2 Descrição da Linguagem SISAL

A linguagem SISAL é uma linguagem fracamente tipada e orientada a expressões, onde valores constantes ou variáveis podem ser associadas com o resultado de uma determinada expressão [Gur+87]. Nenhuma variável ou valor pode ser redefinido, desta maneira todo o uso dado à uma variável em outras expressões será sempre referenciado com o mesmo valor. A ordem de execução de um programa em SISAL é determinada exclusivamente pela disponibilidade dos valores para uso em expressões, e não afetam os resultados computados. Uma expressão pode ter mais que um resultado. Chamadas de funções podem ocorrer em qualquer parte da expressão. A definição da função simplesmente encapsula uma expressão dentro de um mecanismo de passagem de parâmetros.

A linguagem SISAL possui tipos escalares comuns aos de outras linguagens de programação como: `integer`, `real`, `boolean`, `double_real` e `character`. As operações booleanas E (&), OU (I) e não (~) são fornecidas, juntas com testes para igualdade (=) e desigualdade ($\neq$). Todos os usuais `integer`,

`real`, e `double_real` aritméticos são fornecidos, juntamente com um número de funções predefinidas como as operações de módulo (`mod`), de valor absoluto (`abs`), de máximo (`max`) e o mínimo (`min`), além da exponenciação (`exp`). Tipos de conversão explícitos são fornecidos. Em cada caso o nome do operador é o nome do tipo requerido.

3.3.3 Estruturas de Dados em SISAL

As estruturas de dados permitidas em SISAL são quatro: `array`, `stream`, `record` e `union`.

- ***Arrays***

Operações de criação e preenchimento de *arrays* podem ser obtidas da seguinte forma:

Criação de um array vazio

```
A:= array aa_real []
```

Criação de um array listando todos os seus elementos explicitamente

```
B:= array a_real [0: 1.0 , 10.0 , 100.0]
```

Preenchimento de um array

```
C:= array_fill (1 , 10 , 0.0)
```

A primeira expressão cria um *array* vazio, enquanto a segunda expressão cria um *array* listando todos os seus elementos explicitamente. A terceira expressão preenche um *array* de um tamanho especificado com um certo valor. Outras operações relacionadas ao conhecimento do limite superior de um *array*, do limite inferior e do tamanho de um *array* são observados respectivamente:

Tamanho de um *array*

```
I := array_size (A)
```

Limite Superior de um *array*

```
J := array_limh (A)
```

Limite inferior de um *array*

```
K := array_liml (A)
```

É possível também a concatenação de *arrays*:

Concatenação

```
S := array [1: 2, 5, 7] % 2 5 7
```

```
V := S [2: 4 ; 3 : 8]; % 2 4 8
```

A concatenação junta dois *arrays* de mesmo tipo.

```
X := array [1: 1]
```

```
Y := X II V; % 1 2 4 8
```

A operação de alteração de valores de um *array*, de remoção de elementos, de inserção de elementos extras e de ajuste são observadas respectivamente:

Alteração de Valor

```
A[J : V]
```

Onde o J-ésimo elemento do vetor A é extraído e o novo valor V é inserido em seu lugar.

Remoção de elementos

A remoção de elementos de um *array* pode somente ser feita na primeira ou última posição. As operações `array_reml (A)` e `array_remh (A)` retornam um valor idêntico ao vetor A, com a diferença de que na primeira o índice superior é somado de 1 ($LO = LO + 1$) e na segunda o limite superior é decrementado de 1 ($HI = HI - 1$).

Inserção de elementos extras

Como na remoção, um elemento só pode ser inserido em um *array* na primeira ou última posição através das operações `array_addl (A,V)` e `array_addh (A,V)`, onde na primeira, é retornado um *array* igual a A com o índice mais baixo decrementado de 1 e o valor V inserido como primeiro elemento e na segunda, com o índice mais alto sendo incrementado de 1 e V inserido como último elemento.

Ajuste

Essa operação retorna um *array* no intervalo de LO até HI contendo os mesmos dados de A, através da operação `array_ajust(A , LO , HI)`.

• Streams

Stream é um tipo de dado que permite que produtores e consumidores de informação executem simultaneamente uma estrutura compartilhada, gerando paralelismo. *Stream* são agregados homogêneos, como *arrays*, exceto pelo fato dos componentes de uma *stream* não poderem ter acesso randômico, mas somente em ordem, usando operações como `first` e `rest`.

Operações de criação de *streams* são obtidas da seguinte forma:

```
type s_bool = stream [boolean]
type s_int = stream [integer]
S := stream s_bool [];
T := stream s_init [2 , 3 , 5 , 7 , 11]
```

Onde a primeira operação cria uma *stream* vazia e a segunda operação cria uma *stream* listando todos os seus elementos.

Operações de acesso a valores são obtidas a partir das seguintes operações:

```
next := stream_first (T)
how_many := stream_size (T)
all_used := stream_empty
```

Onde a primeira operação seleciona o primeiro elemento de uma *stream*, a segunda operação retorna o número de valores da *stream* e a terceira operação retorna um valor booleano indicando se a *stream* está vazia.

Operações de transformação criam uma nova *stream* a partir dos valores de uma *stream* existente:

```
U := stream_rest (T)
W := stream_append (S, false)
X := stream_append (W, true)
Y := W II X
```

A primeira operação descarta o primeiro valor de uma *stream*, a segunda e a terceira operação adicionam um valor no final da *stream*, e a quarta operação concatena duas *streams* do mesmo tipo.

• Registros

Um registro consiste de um ou mais campos de tipos arbitrários, onde cada campo é identificado por um nome. Há três operações concernentes a registros que são a criação de registros, a seleção de um campo do registro e a substituição de um campo em um registro por um novo valor. Mais de um campo pode ser modificado por uma simples operação de substituição, por exemplo:

```
type dest =record[Segment,Node:integer;
                Input:character;MatchFn:array[character]]
D := record dest[Segment:2;Node:2064;           % criação
                Input:'L';MatchFn:'EW' ];
in_point:=D.Input;                             % seleção
F := D replace[Segment:1;node:1045];           % substituição
```

• Uniões

Uma união consiste de um conjunto de tipos, onde cada tipo está associado a um nome de rótulo. Um valor do tipo união consiste de um valor de rótulo, e um valor constituído, onde o tipo corresponde ao valor do rótulo:

```

type arith_op = union [ int_op  : integer;
                        real_op  : real;
                        dble_op  : double_real ]

```

O valor do tipo `arith_op` terá um valor de rótulo de `int_op`, `real_op` ou `dble_op` e o correspondente valor constituído do tipo `integer`, `real` ou `double_real`.

3.3.4 Estruturas de controle de fluxo

A linguagem SISAL também permite estruturas de execução condicional como IF-THEN- ELSE e a construção FOR possui duas formas: a paralela e a iterativa. A iterativa, da forma como conhecemos, se refere a dependência de um ciclo em relação aos resultados de ciclos anteriores, e é realizada através da construção `for init`. A forma paralela é usada quando da necessidade dos ciclos serem executados concorrentemente, e é onde se utiliza a construção `for all`.

• Forma iterativa de construção for

Neste exemplo é observado a construção de um algoritmo usando `for initial` para o cálculo da raiz quadrada usando o método de Newton-Raphson:

```

sqrt_x :=
  for initial est := 0.5 * x
repeat
  est := 0.5 * (old est + x /old est) ;
  dif := abs (est * est - x)
until diff < tolerancia
returns value of est
end for

```


- **Forma paralela de construção for**

Neste exemplo, é assumido `Vec` como um tipo `array[real]`, usando a construção `for all` para calcular um novo `array new_vec` no qual cada elemento de `vec` é correspondido:

```
New_Vec := for elem in Vec
  recp := 1.0 / elem
returns array of recp
end for
```

É importante ressaltar que a construção `value of` e `array of`, vistos nas construções `for`, formam operações de redução que determinam valores a serem retornados. Como exemplo, podemos citar as seguintes operações de redução: `sum` (somatório do resultado de cada ciclo), `product` (multiplicação dos resultados de cada ciclo), `greatest` (maior resultado de todos os ciclos), `catenate` (retorna `array` ou `stream` concatenados) e `array of stream` (retorna os resultados em forma de `array` ou `stream`).

3.3.5 O bloco `let`

O propósito do bloco `let` é o de definir um ou mais nomes de valores, os quais podem ser usados durante a evolução de uma expressão. O resultado dessa expressão é o resultado do bloco. Nomes de valores são definidos no bloco, mas não estão disponíveis para uso fora do bloco. Esse exemplo calcula duas raízes de uma equação quadrática:

```
x_1, x_2 := let disc := b * b - 4.0 * a * c ;
              root :=sqrt (disc)
              two_a := 2.0 * a
            in
              (-b + root) / two_a ,
              (-b - root) / two_a
            end let
```

3.3.6 Funções

Os programas em SISAL são construídos através de funções que recebem parâmetros de entrada e retornam valores de saída. Dentro dessa visão, SISAL pode apresentar funções recursivas, com o intuito de gerar paralelismo, como no caso da função ser duplamente recursiva, como é o caso do programa para cálculo do fatorial.

```
function fat(x,y:integer returns integer)
    if x<y then y
        else fat(x, (x+y)/2+1)*fat((x+y)/2,y)
    end if
end function
function main(returns integer)
    fat(7,1)
end function
```

3.3.7 Processo de Compilação

A compilação de um programa escrito em SISAL, passa por n etapas. Na primeira, o programa é convertido em um formato intermediário e independente da máquina, denominado IF1 [Ske+85]. Esse formato intermediário IF1, é uma representação gráfica independente da máquina. Um grafo de fluxo de dados IF1 é acíclico e nele, nós representam operações e linhas definem o caminho dos dados entre os nós. Cada linha tem especificado o tipo de dado que é transmitido por ela.

Através do formato IF1, que contém um conjunto de nós primitivos e macro-nós, é gerado o formato IDC (*Intermediate Dataflow Code*), responsável pela compilação das “macros” nos programas, quebrando-os em nós primitivos. Para isso é utilizado outro compilador, que transforma o formato IDC em um formato onde existem apenas nós primitivos. Esse formato de baixo nível é chamado de IGF (*Intermediate Graphical Format*). Um otimizador produz um código eficiente e executa algumas otimizações como: Troca de chamada de função com o código da função “*inline*”, o que reduz o “*overhead*” do tempo de execução da chamada de

função, a eliminação de sub-expressões comuns e a detecção e remoção de expressões que são invariantes dentro de blocos repetitivos. Com isso, o linker gera o formato final do código, onde três arquivos são produzidos. Um contém o programa em si (.COD), outro contém a entrada de dados (.DAT) e por fim, uma tabela de segmentos usada na carga de arquivos de programa (.SEG). O programa Exemp.sis mostra as etapas processo de compilação.

Todas as informações apresentadas a respeito da linguagem funcional SISAL são de grande relevância, pois todos os *benchmarks* obtidos nesse estudo foram codificados nessa linguagem.

Figura 24: Processo de Compilação em SISAL

Capítulo IV

Descrição do Problema Estudado

4.1 Discussão dos Problemas Identificados

Através do desenvolvimento de simuladores (SIM, SAW e MR) [Cav92], foi possível a observação de informações quantitativas com relação ao desempenho de hardware de fluxo de dados executando programas realistas.

Uma das questões observadas refere-se a realização do agrupamento de fichas parceiras de forma associativa na unidade de emparelhamento. Como o custo de implementação de uma memória puramente associativa é proibitivo, foi utilizado um mecanismo de *hashing* por hardware para simular uma memória puramente associativa. Esse mecanismo é composto por uma tabela de *hash* paralela em conjunto com bancos paralelos de memória, um gerador de chave de *hash* e uma unidade de controle. Porém, através de estudos anteriores, foi possível observar que esse tipo de mecanismo de utilização de memória sofre grande degradação quando ocorre a ocupação da memória em mais de 25%, o que pode limitar o paralelismo exposto do modelo e gerando problemas de desempenho.

Dessa forma, é possível observar que por sua complexidade, a Unidade de Emparelhamento torna-se um gargalo dentro da arquitetura de Manchester [Mag92]. Esta unidade acaba impondo um limite máximo relativo a performance da máquina devido ao tempo gasto para o agrupamento das fichas parceiras. Com o aumento

provável das fichas na memória devido ao paralelismo proporcionado, a tendência de degradação torna-se mais preocupante. Na arquitetura Wolf, ocorre a presença da Unidade Controle de Paralelismo, que possui um mecanismo de controle que restringe o número de instâncias distintas de uma mesma instrução. O método de *Trottle* foi desenvolvido para a máquina de Manchester por Ruggiero [Rug87] e faz o gerenciamento de paralelismo através do controle da ativação de funções, de acordo com o nível de atividade da máquina.

Uma outra maneira eficiente que forneceria elementos para minimizar a saturação da máquina, em colaboração com a Unidade de Controle de Paralelismo e diminuindo o prejuízo do aproveitamento de todo o potencial de paralelismo exposto, seria o estudo do comportamento das instruções e conjunto de instruções ou ainda ‘macros’. É relevante saber quais instruções, grupos de instruções ou mesmo estruturas formadas por essas instruções são mais utilizadas pelos diversos tipos de programas. De posse dessas informações, ocorre a possibilidade da arquitetura Wolf fazer utilização de uma nanomemória. A figura 25 mostra um microprograma de n microinstruções, cada um com w bits de largura. Um total de nw bits de memória é necessário para armazenar o microprograma. Após o estudo do comportamento das instruções, revelou que somente que apenas 2^m possibilidades são realmente usadas, com $m \ll n$ [Tan92].

(a) Um microprograma convencional (b) o nanoprograma correspondente se apenas m microinstruções diferentes ocorrem no microprograma.

Figura 25: Utilização da nanomemória

Uma nanomemória com m palavras de w bits poderia ser usada para armazenar cada uma das microinstruções. Cada microinstrução do programa original poderia então ser substituída pelo endereço da palavra da nanomemória que contém aquela microinstrução. Como existem m palavras na nanomemória, a memória de controle (usada para guardar o microprograma) ou principal precisaria ter uma largura de apenas $\log_2 m$ bits. No caso da arquitetura Wolf isso traria um aumento do desempenho, além de proporcionar um número menor dos ciclos de busca das fichas parceiras.

O objetivo central desse trabalho é analisar, a nível de linguagem montadora, esse compromisso de granularidade de instrução. Portanto, foi desenvolvido um programa de análise de código descrito a seguir.

4.2 Análises dos Códigos

A análise do código é importante e se faz necessária para poder-se monitorar o comportamento da máquina sob condições variáveis, podendo-se com isso encontrar situações referentes ao seu desempenho. Com isso, podemos ter leituras impressas que podem ser analisadas e comparadas com outras leituras. Além disso, através dos *benchmarks* (programas de comportamento), podemos obter características do sistema, conseguindo tirar conclusões e obter um melhor entendimento das mesmas.

Os simuladores (SIM, MR, SAW), realizam uma execução dinâmica das instruções através da utilização de múltiplos *loops*, facilitando o estudo de detalhes do funcionamento do sistema, podendo-se fazer simplificações do modelo e evitando erros conceituais na construção do protótipo. Foi criado dessa forma, a ferramenta SOC, para que realizasse a análise estática, para fornecer informação referentes ao comportamento das instruções, geradas pelo compilador SISAL.

O aplicativo SOC fornece três tipos distintos de análise para obtenção de resultados concernentes a comportamento e desempenho com relação a granularidade das instruções, que são respectivamente a análise estática, a análise sequencial e análise estrutural.

A análise estática conta o número de repetições de instruções simples ao longo do programa em formato COD, como por exemplo, o número de repetições da instrução GAN (*Generate Activation Name*). Com isso, é possível analisar a popularidade das instruções geradas pelo compilador.

A análise sequencial conta pares de instruções sequenciais existentes ao nível do programa, como por exemplo, a análise do número de instruções DUP (*duplicate*) precedidas da instrução SYN (*synchronize*) dentro do programa. Nesse tipo de operação é tomado o formato SYN - DUP(N) para descrever essa análise. O comportamento diferenciado dos pares de instruções, é uma característica das máquinas *dataflow*, e o estudo do emparelhamento de instruções, demonstra a geração por templates do compilador. Esta característica, juntamente com o estudo da popularidade das instruções, já mencionadas anteriormente, revelam os mesmos princípios demonstrados no projeto de desenvolvimento de arquiteturas RISC.

O terceiro tipo de análise feita, denominado estrutural, conta o número de repetições de blocos de instruções que formam determinados tipos de estruturas, como por exemplo, um estrutura vista através da forma de representação N -> L - L que demonstra que uma ficha de prefixo N (*normal*), produziu duas fichas de saída de prefixo L (*left*) contendo os resultados obtidos da primeira ficha. O estudo da ocorrência repetida de estruturas é importante, a medida que pode indicar casos a explorar.

O formato de instruções nos arquivos de extensão COD é o seguinte:

<instrução> ::= <prefixo><operador> <destino>[<destino>|<literal>|<>]

Operadores podem ser classificados em duas classes, de acordo com os resultados gerados. No primeiro caso, os operadores podem gerar um resultado através de um único resultado, ainda que uma ou duas fichas contendo este resultado possam ser produzidas. Esses operadores são dados através do prefixo N (*normal*) e são chamados de operadores de resultado único. No segundo caso, são gerados dois dados distintos. Esses operadores são chamados de operadores de resultado duplo. Existem três prefixos para este segundo caso que são respectivamente L (*left*), R(*right*) e D (*different*). Um operador com o prefixo L pode produzir uma ou duas fichas de saída, mas ambas conterão exatamente o resultado da esquerda (*left*) encontrado. Um operador com prefixo R realiza as operações de maneira semelhante

ao prefixo L, somente com alteração de que as fichas de saída conterão exatamente o resultado da direita (*right*) encontrado. A única maneira de produzir duas fichas com resultados diferentes é através do prefixo D (*different*). A figura 26 mostra exemplos de estruturas. Com essa explanação, podemos identificar algumas das estruturas possíveis de serem formadas.

Figura 26: Exemplo de formato de estruturas

4.3 Descrição da Ferramenta SOC

A ferramenta SOC (Sistema para Leitura e Análise para Otimização de Arquivos de Extensão COD), desenvolvida pelo autor, é uma ferramenta destinada a realizar a leitura e compilação de arquivos com extensão COD para análises posteriores. O resultado dessa análise é mostrado tanto de forma gráfica, como textual. Essas análises são respectivamente feitas pelos módulos que formam o

programa, que são: leitura do programa em formato COD, análises estáticas, análises seqüenciais e análises estruturais desse programa.

Figura 27: Módulo para análise estática

O programa foi desenvolvido através de um banco de dados montado em linguagem CLIPPER 5.01, com uma extensão de 1.440 linhas de código fonte. Foi utilizado essa linguagem, pela sua característica de criação de sistemas práticos e flexíveis para a implementação de banco de dados.

Os programas em formato COD foram lidos linha a linha e o resultado do processamento foi alocado em campos específicos, de acordo com o tipo de análise (estática, seqüencial ou estrutural). O aplicativo montou um arquivo de banco de dados para cada tipo de análise. O tempo de execução do programa variou de acordo com os *benchmarks* analisados, indo de 2 minutos para a análise completa do código do programa Ackerman.cod a mais de 7 horas de processamento para análise completa do código de Simple.cod.

GERAR ARQUIVO AN. ESTATICA AN. SEQUENCIAL AN. ESTRUTURAL SAI			
FONTE SEQUENC	INSTRU	SEQUEM	REPETE
	ADI	SCD (R)	1
	ERR	DUP (N)	2
	RRR	ADI (N)	1
	RRR	SVN (L)	1
	RRR	SBI (N)	2
	RRR	RRR (D)	1
	CEI	DUP (N)	2
	DUP	CEI (N)	1
	DUP	RRR (L)	2
	DUP	SBI (N)	1
<ENTER> CONSULTA TEXTUAL <F2> CONSULTA GRAFICA <ESC> SAI			

Figura 28: Módulo para análise seqüencial

O aplicativo SOC consiste de 4 módulos. O primeiro módulo é responsável pela leitura dos arquivos em formato COD interpretados segundo a sintaxe definida por Kirkham [Gur+87], como mostrado na figura 30, além da geração dos arquivos de banco de dados em formato DBF. O segundo módulo é responsável pela análise estática, que é realizada a partir do arquivo gerado no primeiro módulo. A figura 27 mostra o resultado da análise estática para os arquivos ESTAT1 e ESTAT2. A diferença entre os dois arquivos gerados durante a análise estática, está no fato do primeiro apresentar os resultados sem levar em conta o prefixo utilizado para indicar operadores que geram um ou dois resultados (figura 27). O arquivo ESTAT2, dessa forma, mostraria esses mesmos resultados com os prefixos. O terceiro módulo é responsável por um processo de compilação do arquivo COD e por sua análise seqüencial, e da mesma maneira que o segundo módulo, necessita do arquivo gerado no primeiro módulo, para executar essa função. A figura 28 mostra o resultado de uma análise seqüencial na forma textual. É possível visualizar, além do arquivo SEQUENC, que apresenta o resultado da análise, o arquivo FONTE, que é o arquivo gerado no primeiro módulo. O campo INSTRU em conjunto com o campo SEQUEM, formam o par de instrução, como está demonstrado em destaque através do par ADI-SCD(R). O campo REPETE contém o número de repetições dos pares de instruções. A quarta análise, denominada análise estrutural, é também realizada a partir do

arquivo FONTE gerado no primeiro módulo. A figura 29 mostra o resultado da análise estrutural de forma gráfica. Cada um dos módulos responsáveis por um tipo de análise podem ser visualizados de forma textual ou gráfica, como é observado na figura 28 e na figura 29 respectivamente, e ainda gerar um relatório dos resultados de cada um desses módulos.

Figura 29: Módulo para análise estrutural

0 ONKILO 0LBY	1 23NGAN1 25REW1 24LBY
1 ONKILO 0LBY	1 24NDUP1 26REW1 27LBY
0 IRSCD0 0LBY	1 25LSAN1 6LBY1 8LEW
1 1NGAN1 3REW1 2LBY	1 26LSAN1 9LEW
1 2NDUP1 4REW1 5LBY	1 27LPRP0 1REWD0 2LEW
1 3LSAN1 6LBY1 8LEW	1 28NGAN1 30REW1 29LBY
1 4LSAN1 9LEW	1 29NDUP0 2REW1 31LBY
1 5LPRP0 1REWD0 3LBY	1 30LSAN1 6LBY1 8LEW
1 6NCEI1 7LBY10	0 2LSAN1 9LEW
1 7NDUP1 8REW1 9REW	1 31LPRP0 1REWD0 1LEW
1 8LBRR1 16LEW	1 32NSBI1 33LBY11
1 9DBRR1 10LBY1 40LBY	1 33NDUP1 35LBY1 37LEW
1 10NDUP1 11LBY1 15LEW	1 34LSYN1 38LEW11
1 11NCEI1 12LBY10	1 35NGAN1 37REW1 36LBY
1 12NDUP1 13LBY1 14LBY	1 36NDUP1 38REW1 39LBY
1 13NDUP1 15REW1 16REW	1 37LSAN1 6LBY1 8LEW
1 14NDUP1 19LEW1 19REW	1 38LSAN1 9LEW
1 15LBRR1 22LBY	1 39LPRP0 1REWD0 1LEW
1 16DBRR1 17LBY1 32LBY	1 40NADI0 1LEW11
1 17NDUP1 20LBY1 18LBY	1 41NDUP1 42LBY
1 18NDUP1 23LBY1 25LEW	1 42NDUP1 1LBY1 3LEW
1 19RBRR1 34RBY	1 43NDUP1 4LEW
1 20NSBI1 21LBY11	0 3ROPT0 0LBYG0
1 21NDUP1 28LBY1 30LEW	
1 22NSBI1 26LEW11	

Figura 30 : Exemplo de um programa em formato COD para a função de Ackerman

Capítulo V

Análise dos Resultados da Ferramenta SOC

5.1 Introdução

A ferramenta SOC foi empregada para analisar alguns programas utilizados como *benchmarks* no estudo de arquiteturas paralelas. Os *benchmarks* são programas de comportamento que procuram explorar os recursos computacionais da máquina, levando-a a trabalhar em determinadas situações de limite. Os programas utilizados são todos codificados em linguagem SISAL, e analisados no formato .COD e incluem Função de Ackerman, Integração Binária, Cálculo do Fatorial, Multiplicação de Matrizes, Algoritmo N_queens, Programa para Cálculos na Área de Dinâmica Molecular e Programa para Cálculos na Área de Hidrodinâmica. É importante ressaltar, que para efeito de visualização nos gráficos, não foram exibidos valores que atingiram abaixo de 2% para os critérios de avaliação estática e sequencial. A totalidade dos valores obtidos nestas avaliações estão disponíveis no apêndice A.

5.2 Função de Ackerman

O programa Ackerman.cod é utilizado para calcular a Função de Ackerman. É um algoritmo muito usado para explorar características de recursividade e eficiência de chamadas de funções. O número de chamadas do algoritmo para efetuar o cálculo de Ackerman (3,n) é dado por: $(128 * 4^n - 120 * 2^n + 9 * n + 37) / 3$. O resultado é dado por: $Ackerman(3,n) - 1 = 2^{n+3} - 4$. O código do programa em SISAL para o cálculo da função de Ackerman é apresentado a seguir:

```
Export main
Function Ack(m,n:integer returns integer)
  If m=0 then
    n+1
  elseif n=0 then
    Ack(m-1,1)
  else
    Ack(m-1,Ack(m,(n-1)))
  end if
end function
```

A figura 31 mostra os resultados obtidos para a análise estática de Ackerman.cod.

Figura 31: Gráfico para análise estática de Ackerman.cod

A figura 32 mostra os resultados obtidos para a análise seqüencial de Ackerman.cod.

Figura 32: Gráfico para análise seqüencial de Ackerman.cod

A figura 33 mostra os resultados obtidos para a análise seqüencial de Ackerman.cod.

Figura 33: Gráfico para análise estrutural de Ackerman.cod

5.3 Integração Binária

O programa Bin.cod é utilizado, da mesma forma que o programa Ackerman.cod, por ser um *benchmark* que apresenta um alto índice de recursividade. Esse programa faz o cálculo da área de uma função por integração binária utilizando a regra do trapézio, utilizando a função $y = x^2 - 6x - 10$. O código do programa em SISAL para o cálculo da função de integração binária é apresentado a seguir:

```
export main
  function F(x:integer returns integer)
    X*X-6*X-10
  end function
  function area(L,R,depth:integer returns integer)
    let
      mid:=(L+R)/2
    in
      if depth<=1 then
        (R-L)*F(mid)
      else
        area(L,mid,depth-1)+area(mid,R,depth-1)
      end if
    end let
  end function
```

```

end function
function main(A,B,depth:integer returns integer)
    area(A,B,depth)
end function

```

A figura 34 mostra os resultados obtidos para a análise estática de Bin.cod.

Figura 34: Gráfico para análise estática de Bin.cod

A figura 35 mostra os resultados obtidos para a análise sequencial de Bin.cod.

Figura 35: Gráfico para análise sequencial de Bin.cod

A figura 36 mostra os resultados obtidos para a análise estrutural de Bin.cod.

Figura 36: Gráfico para análise estrutural de Bin.cod

5.4 Fatorial

O programa `Fat.cod` é utilizado como *benchmark* por ser um algoritmo duplamente recursivo. Ele efetua o cálculo do fatorial de um número, e é um algoritmo interessante utilizado em SISAL para gerar paralelismo, através da implementação dessas funções duplamente recursivas. O código do programa em SISAL para o cálculo da função de Fatorial é apresentado a seguir:

```
function fat(x,y:integer returns integer)
  if x<y then y
    else fat(x, (x+y)/2+1)*fat((x+y)/2,y)
  end if
end function
function main(returns integer)
  fat(7,1)
end function
```

A figura 37 mostra os resultados obtidos para a análise estática de `Fat.cod`.

Figura 37: Gráfico para análise estática de `fat.cod`

A figura 38 mostra os resultados obtidos para a análise sequencial de Fat.cod.

Figura 38: Gráfico para análise sequencial de Fat.cod

A figura 39 mostra os resultados obtidos para a análise estrutural de Fat.cod.

Figura 39: Grafico para análise estrutural de Fat.cod

5.5 Multiplicação de Matrizes

O programa Matrix.cod é utilizado por ser um *benchmark* que explora as características da Unidade de Memória de Estruturas. É também um *benchmark* bastante popular para testar várias arquiteturas paralelas, pois muitos problemas da vida real reduzem-se a multiplicação de matrizes. O código do programa em SISAL para o cálculo da multiplicação de matrizes é apresentado a seguir:

```
export Main
type Vector = array[real];
type Matrix = array[Vector]
function Fill(N1,N2:integer;R:real returns Matrix)
let
Row:=array_fill(1,N2,R)
in array_fill(1,N1,Row)
end let
end function
function matmult(A,B:Matrix;N1,N2,M:integer returns
Matrix)
for I in 1,N1
  Row:=
  for J in 1,N2
```

```

        E1:=
            for K in 1,M
                R:=A[I,K]*B[K,J]
                returns value of sum R
            end for
        returns array of E1
    end for
returns array of Row
end for
end function
function Main(returns real)
let
M1:=Fill(12,12,0.5);
M2:=Fill(12,12,1.2);
M3:=Matmult(M1,M2,12,12,12)
in M3[12,12]
end let
end function

```

A figura 40 mostra os resultados obtidos para a análise estática de Matrix.cod.

Figura 40: Gráfico para análise estática de Matrix.cod

A figura 41 mostra os resultados obtidos para a análise seqüencial de Matrix.cod.

Figura 41: Gráfico para análise seqüencial de Matrix.cod

A figura 42 mostra os resultados obtidos para a análise estrutural de Matrix.cod.

Figura 42: Gráfico para análise estrutural de Matrix.cod

5.6 N_queens

Da mesma forma que o programa *Fat.cod*, o programa *N_queens* também é utilizado como *benchmark* por ser um algoritmo responsável pela geração de um alto índice de paralelismo, através da implementação de funções duplamente recursivas. É também um exemplo de *benchmark* não numérico, da classe de ‘dividir para conquistar’.

A figura 43 mostra os resultados obtidos para a análise estática de *N_queens.cod*.

Figura 43: Gráfico para análise estática de *N_queens.cod*

A figura 44 mostra os resultados obtidos para a análise seqüencial de N_queens.cod.

Figura 44: Gráfico para análise seqüencial de N_queens.cod

A figura 45 mostra os resultados obtidos para a análise estrutural de N_queens.cod.

Figura 45: Gráfico para análise estrutural de N_queens.cod

5.7 Dinâmica Molecular

Como o próprio nome diz, este programa está relacionado com cálculos na área de dinâmica molecular, onde ocorre a possibilidade de um alto nível de paralelismo através da sincronização de seus elementos. Este é um programa de grande porte, cuja análise excedeu 3 horas de processamento em um computador PC 486. O programa possui 2.419 linhas de código.

A figura 46 mostra os resultados obtidos para a análise estática de Dinâmica.cod.

Figura 46: Gráfico para análise estática de Dinâmica.cod

A figura 47 mostra os resultados obtidos para a análise seqüencial de Dinâmica.cod.

Figura 47: Gráfico para análise seqüencial de Dinâmica.cod

A figura 48 mostra os resultados obtidos para a análise estrutural de Dinâmica.cod.

Figura 48: Gráfico para análise estrutural de Dinâmica.cod

5.8 Simple

O programa Simple está relacionado com cálculos na área de hidrodinâmica. Trata-se de um código extenso com 10.816 linhas, que pode ser estudado tanto analiticamente, como por experimentação. Escondido em seu perfil está uma substancial paralelismo produtor/consumidor, possível somente com a sincronização de elemento por elemento. Este programa excedeu 7 horas de processamento em um computador PC 486.

A figura 49 mostra os resultados obtidos para a análise estática de Simple.cod.

Figura 49: Gráfico para análise estática de Simple.cod

A figura 50 mostra os resultados obtidos para a análise sequencial de Simple.cod.

Figura 50: Gráfico para análise sequencial de Simple.cod

A figura 51 mostra os resultados obtidos para a análise estática de Simple.cod.

Figura 51: Gráfico para análise estrutural de Simple.cod

5.9 Utilização de Instruções Durante a Análise Estática

A análise estática mostra claramente que algumas instruções são muito mais frequentes que outras. Em particular, a instrução **DUP** (*Duplicate*) corresponde a mais de 20% das instruções de todos os programas, chegando a 33% na porcentagem de utilização em relação as outras instruções, no caso de Ackerman, como mostram respectivamente a figura 53 e os dados para análise estática no apêndice A. Essa concentração é facilmente explicável em face ao mecanismo de sincronização do *Dataflow*: é necessário gerar dados apenas para “disparar” trechos de código, e esses dados são gerados pela instrução **DUP**.

Além disso, para explorar paralelismo, também é necessário gerar cópia dos dados como no exemplo.

Figura 52: Grafo a fluxo de dados.

Nesse caso, parece que um DUP com maior fan-out produzirá código mais compacto. Entretanto, essa modificação tem profundas implicações estruturais. Uma possível modificação será a otimização, para compilação, de DUP para TUP (*Tuplicate*). As vantagens dessa otimização dizem respeito a diminuição do número de instruções, além da diminuição das necessidades de sincronização, com um tempo menor da latência de comunicação.

Figura 53: Comparação sobre a utilização da instrução DUP

Outra instrução freqüente usada para sincronização, é a SYN (*Sincronize*) que aparece mais freqüentemente em programas maiores e/ou mais complexos. Essa instrução é necessária quando ocorre a necessidade de uma operação sincronizada entre elementos, garantindo que nenhum elemento executará alguma instrução posterior sem a presença de todos os elementos necessários para a execução da instrução atual. Podemos observar isso através do seguinte fragmento de código.

Figura 54: Fragmento de código com a utilização da instrução SYN

Figura 55: Comparação sobre a utilização da instrução SYN

A instrução SAN aparece muito em programas recursivos. Esse fato decorre por causa da implementação de chamadas de funções. Quando é realizada a chamada de função, um nome de ativação tem que ser gerado e as fichas dos parâmetros que devem ser repassados para a função são rotuladas com esse novo nome de ativação. Esse processo de mudança do nome de ativação dos parâmetros de entrada, é

realizado pela instrução SAN (*Set Activation Name*), sendo esta a razão da sua grande utilização em programas recursivos. A partir daí, as fichas dos parâmetros passam para o “corpo da função”.

Figura 56: Estrutura de uma Chamada de Função

Na figura 56 observa-se além da instrução SAN, explicada anteriormente, a instrução GAN (*Generate Activation Name*), é responsável pelo retorno de um único nome de ativação que será utilizado pelas fichas da função numa determinada instância. A instrução PRP (*Prepare Access*) gera uma ficha com o campo de dado sendo formado pelo contexto da função que fez a chamada e pelo endereço da instrução destino que espera o resultado da função [Mag92]. A instrução SCD (*Set Color and Destination*), é responsável pelo envio do valor do resultado para o destino traçado pela instrução PRP, com o nome de ativação da função responsável pela chamada.

Figura 57: Comparação sobre a utilização da instrução SAN

A instrução BRR é um operador de controle de fluxo, que recebe como entrada um tipo de dado qualquer, como operando da esquerda, e um tipo lógico (true ou false), como operando da direita, que serve como condição para o operando da esquerda. O resultado de saída é a combinação dos operandos de entrada, em relação ao prefixo da instrução, como é observado na figura 58.

Figura 58: Utilização da instrução BRR

Figura 59: Comparação sobre a utilização da instrução BRR

A instrução PRL (Proliferate) é utilizada para manipular dados estruturados. Na figura 60, é possível observar o emparelhamento de um ponteiro com os vários elementos de uma estrutura, gerando dessa forma N fichas tendo como valor de dado o ponteiro e índices variando de 0 até N-1.

Figura 60: Utilização da instrução PRL

Figura 61: Comparação sobre a utilização da instrução PRL

Na análise estática, as 5 instruções mais populares são usadas para gerenciamento e exposição do paralelismo, e não para “cálculo útil”.

5.10 Utilização de Sequências de Instruções Durante a Análise Sequencial

Com mais de um quinto das instruções sendo DUP e tendo uma porcentagem de utilização bastante equilibrada entre os programas além de 30% a 50% das instruções serem pertencentes ao conjunto {DUP, SAN, SYN, BRR, PRL}, é estatisticamente provável que duas instruções DUP sejam justapostas, além da formação de pares de instruções entre DUP, SAN, SYN, BRR, PRL. A análise sequencial foi idealizada para investigar essa possibilidade, e as figuras 62, 63, 64, 65 e 66 mostram que isso efetivamente ocorre.

Figura 62: Comparação sobre a utilização da sequência DUP-DUP(N)

Figura 63: Comparação sobre a utilização da sequência DUP-SAN(L)

Figura 64: Comparação sobre a utilização da sequência DUP-SYN(L)

Figura 65: Comparação sobre a utilização da sequência BRR-DUP(N)

Figura 66: Comparação sobre a utilização da sequência DUP-PRL(N)

Verifica-se que, em muitos (mas não todos) os programas, há grande concentração de pares de instruções.

5.11 Utilização de Estruturas Durante a Análise Estrutural

Além da regularidade de ocorrências de instruções, há regularidade na ocorrência de estruturas. Como já foi dito anteriormente, o estudo da ocorrência repetida de estruturas é importante, a medida que pode indicar casos a explorar.

Figura 67: Comparação sobre a utilização da estrutura L -> L-

Figura 68: Comparação sobre a utilização da estrutura N -> L-

Figura 69: Comparação sobre a utilização da estrutura N -> L-L

Confrontando os dados da execução dinâmica com os dados da análise estática, é possível desenvolver uma relação entre os programas que estão sendo executados e o comportamento de suas instruções, com as instruções que são mais comuns na análise estática. Foi possível observar dessa forma, dois tipos diferenciados de comportamento. Na maior parte dos programas, as instruções que são mais comuns na análise estática, continuam a ser mais comuns na análise dinâmica, como é apresentado na figura 70. Em programas com a característica específica de apresentarem grandes quantidades de dados estruturados e de operações sobre esses dados, houveram algumas variações entre os dados da análise estática e da execução dinâmica. Na maioria dos programas com a característica citada acima, as instruções mais populares na análise estática continuaram sendo as mais populares durante a execução dinâmica, com o conjunto de instruções DUP, SAN, SYN, BRR e PRL sendo utilizados em 30% a 50% dos programas, mas com a diferença de que na execução dinâmica, as instruções responsáveis pelo tratamento e operações sobre dados estruturados tiveram um aproveitamento maior do que na análise estática. Essa observação é possível através da figura 71, com relação as instruções RSS (*Read from Structure Store*), MIV (*Modify Indexing Value*), DRC (*Decrement Reference Count*), WSS (*Write to Structure Store*), SOX (*Split on Index*), MID (*Modify Indexing Value and Double*) e DSR (*Double Structure Store Read*). Em uma pequena

parte dos casos, as instruções mais populares na análise estática continuaram com uma alta popularidade na execução dinâmica, mas foram ultrapassadas por instruções referentes ao tratamento e operações sobre dados estruturados, como está apresentado na figura 72. Para efeito de visualização dos gráficos, não foram colocados valores com percentual menor que 1%, tanto em relação ao gráfico da análise estática, como em relação ao gráfico de execução dinâmica.

Figura 70: Grafico para Análise Estática X Execução Dinâmica de Quicksor.sis

Figura 71: Gráfico para Análise Estática X Execução Dinâmica para Gauss.sis

Figura 72: Gráfico para Análise Estática X Execução Dinâmica para MMS.sis

5.12 Conclusões

Os resultados obtidos pelos gráficos relacionados a análise estática das instruções dos programas Ack.cod, Bin.cod, Fat.cod, Matrix.cod, N_queens.cod, Dinamica.cod e Sample.cod revelam que a instrução DUP (*duplicate*) foi a que obteve o maior percentual em todos eles. Ela obteve 33%, 24%, 24%, 31%, 25% e 24% respectivamente. Esse foram percentuais bem acima do percentual obtido pela segunda instrução na ordem de classificação. As outras instruções melhores classificadas de acordo com cada programa foram SYN (*Synchronize*) com 13% em Matrix.cod, 9% em N_queens.cod, 13% em Dinâmica.cod e 10% em Simple.cod. A instrução SAN (*Set Activation Name*) foi a segunda melhor qualificada com 17% em Akerman.cod, 18% em Bin.cod e 17% em Fat.cod. Neste aspecto, a instrução SAN é mais utilizada nos programas que apresentaram um alto índice de recursividade, enquanto a instrução SYN foi mais utilizada em programas com uma quantidade de código maior e mais complexos. A instrução PRL (*Proper Proliferate*) também foi mais utilizada nestes programas mais complexos, enquanto a instrução GAN (*Generate Activation Name*) apareceu com maior frequência em programas que exploram a recursividade, além da instrução BRR (*Branch on Boolean and Repeat*). Com relação a análise sequencial, os pares de instruções mais utilizados ficaram pouco a frente dos segundos mais utilizados, diferenciando da análise estática. Os pares mais utilizados foram DUP-SAN(L) com 11% em Ackerman.cod, DUP-SAN(L) com 18% em Bin.cod, DUP-SAN(L) com 10% em Fat.cod, DUP-SYN(L) com 6% em Matrix.cod, DUP-DUP(N) com 7% em N_queens, DUP-DUP(N) com 5% em Dinâmica.cod e DUP-DUP(N) com 7% em Simple.cod. Também é possível observar, que nos programas altamente recursivos, o par de instruções mais utilizado foi DUP-SAN(L), enquanto nos programas de maior complexidade foi utilizado o par de instruções DUP-DUP(N), com exceção de Matrix.cod que é um programa utilizado como *benchmark* para a exploração da Memória de Estruturas, cujo resultado apontou o par de instruções mais utilizado DUP-SYN(L). Os pares de instruções classificados em segundo foram PRP-SCD(R) com 8% para

Ackerman.cod, GAN-DUP(N) com 7% para Bin.cod, SAN-CLI(N) com 8% para Fat.cod, DUP-DUP(N) com 5% para Matrix.cod, DUP-SAN(L) com 5% para N_queens, DUP-SYN(L) com 4% para Dinâmica.cod e DUP-SYN(N) com 3% para Simple.cod. É importante também ressaltar a alta utilização do par DUP-PRL(L) nos programas com um grande número de linhas de código. Com relação a análise estrutural, as estruturas mais utilizadas, como explicadas e exemplificadas no capítulo IV, foram as seguintes: R->L- com 21% para Ackerman.cod, L->L- com 27% para Bin.cod, N->R-R com 22% para Fat.cod, L->L- com 28% para Matrix.cod, N->L-L com 21% para N_queens.cod, L->L com 23% para Dinâmica.cod e N->L- com 19% para Simple.cod. Foi possível observar, através de uma análise geral, as estruturas que mais foram utilizados num contexto mais amplo, foram L->L - e N->L-.

Através do confronto entre os dados da execução dinâmica e os dados da análise estática, foi possível desenvolver uma relação entre os programas executados e suas instruções, com as instruções que são mais comuns na análise estática. Foi possível observar dessa forma, que na maioria dos casos onde os programas não apresentaram características específicas de utilização de grandes quantidades de dados estruturados e operações sobre esses dados, a análise estática concorda plenamente com a execução dinâmica, havendo apenas uma pequena variação na utilização da instrução BRR. Esse fator ocorre devido ao fato da instrução BRR (*Branch on Boolean and Repeat*) ser um operador de controle de fluxo, ocorrendo, com isso, durante a execução dinâmica, a chamada de trechos de programas que invocam várias instâncias de blocos repetitivos, que serão executados várias vezes através de múltiplos loops. Enquanto isso, na análise estática essa instrução aparece uma vez em cada trecho do programa, mesmo com a execução de várias instâncias da mesma.

Nos programas que apresentaram a característica específica de terem grandes quantidades de dados estruturados e de operações sobre esses dados, houveram algumas variações entre as instruções mais populares na análise estática e as instruções responsáveis pelo tratamento e operações sobre dados estruturados da execução dinâmica. Na maioria dos casos, as instruções mais populares na análise estática (DUP, SAN, SYN, BRR e PRL) continuaram mantendo a média de utilização de 30% a 50% dos programas, mas com a diferença de que na execução dinâmica, as instruções responsáveis pelo tratamento e operações sobre dados estruturados tiveram

um aproveitamento maior do que na análise estática. Somente em uma pequena parcela dos casos, as instruções mais populares na análise estática perderam essa popularidade para instruções referentes ao tratamento e operações sobre dados estruturados. É importante ressaltar, que na arquitetura Wolf, os dados estruturados são armazenados e processados na Unidade Processador Vetorial e Memória Vetorial, reduzindo o uso de memória e evitando que os dados circulem pelo *pipeline*.

Na análise sequencial foi possível observar que a maioria dos pares de instruções geradas pelo compilador foram formados dentro do conjunto de instruções mais populares (DUP, SAN, SYN, PRL, BRR), havendo um aproveitamento pequeno dos pares de instruções formados pelo conjunto das outras instruções. Isso revela uma alta concentração de popularidade em torno de somente algumas instruções.

Durante a análise estrutural, foi observada a maior utilização de estruturas formadas por instruções que geram um resultado de saída nos programas de maior complexidade e tamanho, chegando na maioria das vezes, de 50% a 80% dos totais das estruturas desses programas.

Todos esses resultados indicam um alto grau de regularidade em diversos níveis da análise do código em .COD. Essa regularidade pode, e deve, ser explorada para aumentar a eficiência da arquitetura.

A principal conclusão desse trabalho é que as arquiteturas a fluxo de dados gastam excessivos recursos para exposição e gerenciamento de paralelismo. Qualquer arquitetura a fluxo de dados que pretende ser bem sucedida deve tratar desse problema.

Apêndice A

Dados para Análise Estática - Ackerman.cod

Instruções	Repetições	Porcentagem
DUP	16	33,33%
SAN	8	16,67%
BRR	5	10,42%
GAN	4	8,33%
PRP	4	8,33%
SBI	3	6,25%
CEI	2	4,17%
KIL	2	4,17%
ADI	1	2,08%
OPT	1	2,08%
SCD	1	2,08%
SYN	1	2,08%

Dados para Análise Estática - Bin.cod

Instruções	Repetições	Porcentagem
DUP	15	29,41%
SAN	9	17,65%
SBI	5	9,80%
BRR	4	7,84%
GAN	3	5,88%
MLI	3	5,88%
PRP	3	5,88%
ADI	2	3,92%
KIL	2	3,92%
CLI	1	1,96%
DMD	1	1,96%
OPT	1	1,96%
SCD	1	1,96%
TUP	1	1,96%

Dados para Análise Estática - Fat.cod

Instruções	Repetições	Porcentagem
DUP	9	25,00%
SAN	6	16,67%
ADI	3	8,33%
GAN	3	8,33%
PRP	3	8,33%
BRR	2	5,56%
DMD	2	5,56%
SYN	2	5,56%
CLI	1	2,78%
KIL	1	2,78%
MLI	1	2,78%
OPT	1	2,78%
SCD	1	2,78%
SWA	1	2,78%

Dados para Análise Estática - Matrix.cod

Instruções	Repetições	Porcentagem
DUP	69	21,10%
SYN	41	12,54%
PRL	37	11,31%
SIX	19	5,81%
SAZ	16	4,89%
TUP	12	3,67%
ADX	9	2,75%
SBI	8	2,45%
ADI	7	2,14%
BRR	7	2,14%
CEI	7	2,14%
BRT	6	1,83%
CRJ	6	1,83%
GSP	6	1,83%
MLX	6	1,83%
OST	6	1,83%

SOX	6	1,83%
SRC	6	1,83%
WSS	6	1,83%
DRC	5	1,53%
DSR	3	0,92%
MID	3	0,92%
MIV	3	0,92%
MLI	3	0,92%
POF	3	0,92%
RSS	3	0,92%
SLB	3	0,92%
SOI	3	0,92%
ERC	2	0,61%
GAN	2	0,61%
MRC	2	0,61%
SFZ	2	0,61%
WDS	2	0,61%
ADR	1	0,31%
BIX	1	0,31%
BRV	1	0,31%
KIL	1	0,31%
MLR	1	0,31%
OPT	1	0,31%
SWA	1	0,31%
SXI	1	0,31%

Dados para Análise Estática - N_queens.cod

Instruções	Repetições	Porcentagem
DUP	161	30,96%
SYN	48	9,23%
SAN	26	5,00%
BRR	23	4,42%
PRL	23	4,42%
SIX	22	4,23%
ADI	21	4,04%
SBI	19	3,65%
ADX	13	2,50%
CEI	13	2,50%
TUP	12	2,31%
PRP	10	1,92%
DRC	8	1,54%
BRT	7	1,35%
DMD	7	1,35%
ESP	7	1,35%
FSS	7	1,35%
GAN	7	1,35%
OST	7	1,35%
SOX	7	1,35%
BIX	6	1,15%
GSP	6	1,15%
SAZ	6	1,15%
SRC	6	1,15%
WSS	6	1,15%
MLI	5	0,96%
SXI	5	0,96%
SCD	4	0,77%
AND	3	0,58%
NOT	3	0,58%
ABI	2	0,38%
CRJ	2	0,38%
OPT	2	0,38%
SLB	2	0,38%
SPJ	2	0,38%
SWA	2	0,38%
BRV	1	0,19%
CGI	1	0,19%
IRC	1	0,19%
KIL	1	0,19%

MIV	1	0,19%
POF	1	0,19%
RSS	1	0,19%
SFZ	1	0,19%
SOI	1	0,19%
SXN	1	0,19%

Dados para Análise Estática - Dinâmica.cod

Instruções	Repetições	Porcentagem
DUP	616	25,47%
SYN	315	13,02%
PRL	124	5,13%
SIX	110	4,55%
TUP	90	3,72%
BRR	78	3,22%
SAZ	78	3,22%
ADX	73	3,02%
ADI	46	1,90%
CEI	46	1,90%
SLB	43	1,78%
CRJ	39	1,61%
SAN	38	1,57%
BRT	36	1,49%
GSP	36	1,49%
OST	36	1,49%
SOX	36	1,49%
SRC	36	1,49%
WSS	36	1,49%
DRC	34	1,41%
MIV	34	1,41%
SBI	34	1,41%
RSS	33	1,36%
MLI	31	1,28%
BIX	27	1,12%
GAN	27	1,12%
SPJ	26	1,07%
SXI	24	0,99%
PRP	22	0,91%
DMD	20	0,83%
IRC	17	0,70%
SFZ	17	0,70%
MLX	16	0,66%
ESP	12	0,50%
POF	12	0,50%
SCD	12	0,50%
FSS	10	0,41%
DVR	9	0,37%
FLT	9	0,37%
SOI	9	0,37%
MLR	8	0,33%
SXN	8	0,33%
ADR	5	0,21%
BRV	4	0,17%
CGI	4	0,17%
KIL	4	0,17%
SWA	4	0,17%
SPL	3	0,12%
SXO	3	0,12%
YAN	3	0,12%
DRM	2	0,08%
DSR	2	0,08%
MID	2	0,08%
MSP	2	0,08%
OPT	2	0,08%
WDS	2	0,08%
YXI	2	0,08%
ABI	1	0,04%
CLI	1	0,04%
COS	1	0,04%
ERC	1	0,04%
ESS	1	0,04%
EXR	1	0,04%

LGN	1	0,04%
MAP	1	0,04%
MRC	1	0,04%
NGR	1	0,04%
SBR	1	0,04%
SQR	1	0,04%

Dados para Análise Estática - Simple.cod

Instruções	Repetições	Porcentagem
DUP	2564	23,71%
SYN	1119	10,35%
PRL	585	5,41%
SIX	495	4,58%
DSR	451	4,17%
MID	408	3,77%
BRR	384	3,55%
TUP	347	3,21%
ADX	329	3,04%
ADI	292	2,70%
SBI	244	2,26%
SAZ	213	1,97%
RSS	207	1,91%
CEI	196	1,81%
SXI	161	1,49%
BRT	159	1,47%
OST	159	1,47%
SOX	159	1,47%
MLR	145	1,34%
WSS	144	1,33%
GSP	143	1,32%
SRC	142	1,31%
SAN	138	1,28%
CRJ	136	1,26%
MIV	136	1,26%
ADR	127	1,17%
DRC	118	1,09%
SLB	111	1,03%
BIX	100	0,92%
MLX	88	0,81%
SBR	80	0,74%
ESP	70	0,65%
FSS	70	0,65%
SPJ	70	0,65%
IRC	68	0,63%
PRP	64	0,59%
GAN	47	0,43%
MLI	44	0,41%
DVR	36	0,33%
OPT	30	0,28%
POF	27	0,25%
SOI	27	0,25%
SCD	22	0,20%
SFZ	21	0,19%
CGR	11	0,10%
ERC	10	0,09%
MRC	10	0,09%
WDS	10	0,09%
BRV	9	0,08%
DMD	9	0,08%
NOT	9	0,08%
SOR	9	0,08%
SPL	8	0,07%
SXN	8	0,07%
SWA	5	0,05%
CGI	5	0,05%
NGR	5	0,05%
KIL	4	0,04%
ABR	4	0,04%
ORB	4	0,04%
YXI	4	0,04%
CEV	3	0,03%
SQR	3	0,03%

FLT	3	0,03%
YAN	2	0,02%
AND	1	0,01%
EXR	1	0,01%
SXO	1	0,01%
COS	1	0,01%
SIN	1	0,01%

DMD	BRR (D)	1	1,22%
MLI	SCD (R)	1	1,22%
MLI	BRR (D)	1	1,22%
MLI	SBI (N)	1	1,22%
OPT	KIL (N)	1	1,22%
PRP	OPT (R)	1	1,22%
SBI	SBI (N)	1	1,22%
SCD	KIL (N)	1	1,22%
TUP	BRR (L)	1	1,22%

Apêndice B

Dados para Análise Seqüencial - Ackerman.cod

Pares de Instruções	Repetições	Porcentagem
DUP SAN (L)	8	10,53%
PRP SCD (R)	6	7,89%
DUP DUP (N)	5	6,58%
DUP GAN (N)	5	6,58%
DUP PRP (L)	4	5,26%
GAN SAN (L)	4	5,26%
GAN DUP (N)	4	5,26%
SAN BRR (D)	4	5,26%
SAN BRR (L)	4	5,26%
SAN CEI (N)	4	5,26%
BRR DUP (N)	2	2,63%
BRR SBI (N)	2	2,63%
CEI DUP (N)	2	2,63%
DUP BRR (L)	2	2,63%
DUP BRR (R)	2	2,63%
DUP BRR (D)	2	2,63%
KIL KIL (N)	2	2,63%
SBI DUP (N)	2	2,63%
ADI SCD (R)	1	1,32%
BRR ADI (N)	1	1,32%
BRR SYN (L)	1	1,32%
BRR BRR (D)	1	1,32%
DUP CEI (N)	1	1,32%
DUP SBI (N)	1	1,32%
OPT KIL (N)	1	1,32%
PRP OPT (R)	1	1,32%
PRP SAN (L)	1	1,32%
SBI SAN (L)	1	1,32%
SCD KIL (N)	1	1,32%
SYN SAN (L)	1	1,32%

Dados para Análise Seqüencial - Bin.cod

Pares de Instruções	Repetições	Porcentagem
DUP SAN (L)	15	18,29%
GAN DUP (N)	6	7,32%
SAN BRR (D)	6	7,32%
SAN ADI (N)	5	6,10%
BRR SBI (N)	4	4,88%
SAN BRR (L)	4	4,88%
BRR DUP (N)	3	3,66%
DUP MLI (N)	3	3,66%
DUP DUP (N)	3	3,66%
DUP PRP (L)	3	3,66%
DUP GAN (N)	3	3,66%
PRP SCD (R)	3	3,66%
SAN CLI (N)	3	3,66%
KIL KIL (N)	2	2,44%
PRP ADI (N)	2	2,44%
SBI SAN (L)	2	2,44%
SBI MLI (N)	2	2,44%
ADI DMD (L)	1	1,22%
ADI SCD (R)	1	1,22%
BRR SAN (L)	1	1,22%
CLI TUP (N)	1	1,22%

Dados para Análise Seqüencial - Fat.cod

Pares de Instruções	Repetições	Porcentagem
DUP SAN (L)	6	9,84%
SAN CLI (N)	5	8,20%
DUP ADI (N)	4	6,56%
BRR DUP (N)	3	4,92%
DUP GAN (N)	3	4,92%
DUP PRP (L)	3	4,92%
GAN DUP (N)	3	4,92%
PRP SCD (R)	3	4,92%
SAN BRR (L)	3	4,92%
SAN BRR (D)	3	4,92%
ADI DMD (L)	2	3,28%
DUP DUP (N)	2	3,28%
GAN SAN (L)	2	3,28%
PRP MLI (N)	2	3,28%
SWA SYN (L)	2	3,28%
ADI SAN (L)	1	1,64%
BRR SCD (R)	1	1,64%
CLI BRR (D)	1	1,64%
CLI BRR (L)	1	1,64%
DMD DUP (N)	1	1,64%
DMD ADI (N)	1	1,64%
GAN SYN (L)	1	1,64%
KIL KIL (N)	1	1,64%
MLI SCD (R)	1	1,64%
OPT KIL (N)	1	1,64%
PRP OPT (R)	1	1,64%
SAN SCD (R)	1	1,64%
SCD KIL (N)	1	1,64%
SYN SAN (L)	1	1,64%
SYN DUP (N)	1	1,64%

Dados para Análise Seqüencial - Matrix.cod

Pares de Instruções	Repetições	Porcentagem
DUP SYN (L)	26	6,18%
DUP DUP (N)	20	4,75%
DUP PRL (L)	17	4,04%
SYN DUP (N)	13	3,09%
SAZ PRL (L)	12	2,85%
SIX SOX (R)	11	2,61%
PRL SAZ (L)	10	2,38%
SYN PRL (L)	9	2,14%
PRL WSS (N)	7	1,66%
SIX SYN (L)	7	1,66%
BRR SIX (L)	6	1,43%
BRR DUP (N)	6	1,43%
BRT DUP (N)	6	1,43%
DUP CRJ (N)	6	1,43%
DUP SIX (L)	6	1,43%
DUP OST (N)	6	1,43%
DUP SYN (R)	6	1,43%
DUP SRC (N)	6	1,43%
DUP SBI (N)	6	1,43%
GSP DUP (N)	6	1,43%
OST SIX (L)	6	1,43%
SOX BRT (L)	6	1,43%
SRC KIL (N)	6	1,43%
SYN ADX (L)	6	1,43%

WSS	KIL (N)	6	1,43%
ADX	SYN (L)	5	1,19%
CEI	BRR (D)	5	1,19%
DRC	KIL (N)	5	1,19%
DUP	BRR (D)	5	1,19%
DUP	CEI (N)	5	1,19%
PRL	SIX (L)	5	1,19%
SYN	SYN (L)	5	1,19%
ADI	DUP (N)	4	0,95%
CRJ	PRL (L)	4	0,95%
DUP	SAZ (L)	4	0,95%
DUP	GSP (N)	4	0,95%
PRL	MLX (L)	4	0,95%
PRL	SYN (L)	4	0,95%
SBI	ADI (N)	4	0,95%
DSR	RSS (N)	3	0,71%
DUP	WSS (N)	3	0,71%
DUP	TUP (N)	3	0,71%
MID	DSR (D)	3	0,71%
MIV	RSS (N)	3	0,71%
MLI	DUP (N)	3	0,71%
MLX	ADX (L)	3	0,71%
MLX	DUP (N)	3	0,71%
PRL	MID (N)	3	0,71%
SBI	SOI (R)	3	0,71%
SFZ	SLB (L)	3	0,71%
SOI	TUP (N)	3	0,71%
SYN	DRC (N)	3	0,71%
TUP	TUP (N)	3	0,71%
ADI	TUP (N)	2	0,48%
ADX	DUP (N)	2	0,48%
DSR	MIV (N)	2	0,48%
DUP	MRC (N)	2	0,48%
DUP	WDS (N)	2	0,48%
DUP	MLI (N)	2	0,48%
DUP	POF (L)	2	0,48%
DUP	SFZ (L)	2	0,48%
GAN	TUP (N)	2	0,48%
GAN	DUP (N)	2	0,48%
MRC	SYN (D)	2	0,48%
POF	GAN (N)	2	0,48%
POF	SAZ (L)	2	0,48%
PRL	GSP (N)	2	0,48%
RSS	MLR (N)	2	0,48%
SAZ	ADI (N)	2	0,48%
SAZ	DUP (N)	2	0,48%
SLB	MLX (L)	2	0,48%
SYN	SBI (N)	2	0,48%
SYN	WDS (N)	2	0,48%
SYN	ERC (N)	2	0,48%
TUP	DUP (N)	2	0,48%
TUP	SAZ (L)	2	0,48%
TUP	SYN (L)	2	0,48%
TUP	POF (L)	2	0,48%
WDS	KIL (N)	2	0,48%
ADI	CEI (N)	1	0,24%
ADR	BIX (D)	1	0,24%
ADX	TUP (N)	1	0,24%
ADX	SOX (R)	1	0,24%
BIX	SIX (L)	1	0,24%
BIX	GSP (N)	1	0,24%
BRR	SLB (L)	1	0,24%
BRR	ADR (N)	1	0,24%
BRV	SBI (N)	1	0,24%
CEI	TUP (N)	1	0,24%
CEI	SAZ (L)	1	0,24%
CRJ	MID (N)	1	0,24%
CRJ	SLB (L)	1	0,24%
DSR	GSP (N)	1	0,24%
DUP	SXI (L)	1	0,24%
DUP	DRC (N)	1	0,24%
ERC	MRC (N)	1	0,24%
ERC	DUP (N)	1	0,24%

KIL	KIL (N)	1	0,24%
MLR	ADR (N)	1	0,24%
OPT	KIL (N)	1	0,24%
POF	MIV (N)	1	0,24%
POF	MID (N)	1	0,24%
PRL	MIV (N)	1	0,24%
PRL	DSR (D)	1	0,24%
PRL	BIX (D)	1	0,24%
PRL	DUP (N)	1	0,24%
RSS	OPT (R)	1	0,24%
RSS	TUP (N)	1	0,24%
SBI	DUP (N)	1	0,24%
SIX	SLB (L)	1	0,24%
SLB	WSS (N)	1	0,24%
SLB	SIX (L)	1	0,24%
SWA	TUP (N)	1	0,24%
SXI	BIX (D)	1	0,24%
SYN	MIV (N)	1	0,24%
SYN	SAZ (L)	1	0,24%
SYN	DSR (D)	1	0,24%
SYN	SLB (L)	1	0,24%
SYN	BRR (D)	1	0,24%
SYN	ADI (N)	1	0,24%
TUP	DRC (N)	1	0,24%

Dados para Análise Sequencial - N_queens.cod

Pares de Instruções	Repetições	Porcentagem
DUP	DUP (N)	55 7,19%
DUP	SAN (L)	38 4,97%
DUP	SYN (L)	31 4,05%
DUP	PRL (L)	23 3,01%
SIX	SOX (R)	14 1,83%
DUP	BRR (D)	13 1,70%
BRR	DUP (N)	12 1,57%
GAN	DUP (N)	12 1,57%
SAN	DUP (N)	12 1,57%
DUP	GAN (N)	11 1,44%
DUP	SIX (L)	11 1,44%
SYN	DUP (N)	11 1,44%
ADI	DUP (N)	10 1,31%
DUP	CEI (N)	10 1,31%
DUP	PRP (L)	10 1,31%
DUP	SBI (N)	10 1,31%
PRP	SCD (R)	10 1,31%
DUP	DRC (N)	9 1,18%
PRP	DUP (N)	9 1,18%
CEI	BRR (D)	8 1,05%
DRC	KIL (N)	8 1,05%
DUP	BRR (R)	8 1,05%
DUP	BIX (R)	8 1,05%
DUP	ADX (L)	8 1,05%
SAN	PRL (L)	8 1,05%
SYN	PRL (L)	8 1,05%
SYN	SAN (L)	8 1,05%
DUP	OST (N)	7 0,92%
DUP	SYN (R)	7 0,92%
ESP	SBI (N)	7 0,92%
FSS	DUP (N)	7 0,92%
OST	SIX (L)	7 0,92%
PRL	SIX (L)	7 0,92%
SOX	BRT (L)	7 0,92%
SYN	ADX (L)	7 0,92%
SYN	SYN (L)	7 0,92%
BRR	SIX (L)	6 0,78%
BRT	DUP (N)	6 0,78%
DUP	GSP (N)	6 0,78%
DUP	ADI (N)	6 0,78%
DUP	SAZ (L)	6 0,78%
DUP	SRC (N)	6 0,78%
GSP	DUP (N)	6 0,78%
PRL	WSS (N)	6 0,78%

SAN	SAN (L)	6	0,78%
SIX	SYN (L)	6	0,78%
SRC	KIL (N)	6	0,78%
WSS	KIL (N)	6	0,78%
ADI	SAN (L)	5	0,65%
BRR	SXI (L)	5	0,65%
DMD	ADI (N)	5	0,65%
DUP	BRR (L)	5	0,65%
SBI	DUP (N)	5	0,65%
SBI	ADI (N)	5	0,65%
ADX	FSS (N)	4	0,52%
ADX	SOX (R)	4	0,52%
BRR	SYN (L)	4	0,52%
DUP	FSS (N)	4	0,52%
DUP	ESP (L)	4	0,52%
MLI	ADI (N)	4	0,52%
PRL	SAZ (L)	4	0,52%
SAN	ADI (N)	4	0,52%
SAN	MLI (N)	4	0,52%
SAZ	BRR (R)	4	0,52%
SCD	KIL (N)	4	0,52%
SYN	SLB (L)	4	0,52%
BIX	BIX (R)	3	0,39%
BRR	ADI (N)	3	0,39%
CEI	NOT (N)	3	0,39%
DUP	TUP (N)	3	0,39%
DUP	SWA (L)	3	0,39%
DUP	OPT (R)	3	0,39%
NOT	AND (N)	3	0,39%
SBI	DMD (R)	3	0,39%
ABI	CEI (N)	2	0,26%
ADI	ADI (N)	2	0,26%
ADI	SCD (R)	2	0,26%
AND	AND (N)	2	0,26%
BIX	DUP (N)	2	0,26%
BRR	TUP (N)	2	0,26%
CEI	DUP (N)	2	0,26%
CGI	DUP (N)	2	0,26%
DUP	DMD (R)	2	0,26%
DUP	WSS (N)	2	0,26%
DUP	POF (L)	2	0,26%
DUP	CGI (N)	2	0,26%
GAN	TUP (N)	2	0,26%
OPT	KIL (N)	2	0,26%
PRL	DUP (N)	2	0,26%
PRL	SBI (N)	2	0,26%
SAN	TUP (N)	2	0,26%
SAN	SBI (N)	2	0,26%
SAZ	PRL (L)	2	0,26%
SBI	CEI (N)	2	0,26%
SBI	MLI (N)	2	0,26%
SBI	DMD (L)	2	0,26%
SBI	ABI (N)	2	0,26%
SIX	TUP (N)	2	0,26%
SXI	FSS (N)	2	0,26%
SXI	DUP (N)	2	0,26%
SYN	BRR (D)	2	0,26%
SYN	DRC (N)	2	0,26%
SYN	SCD (R)	2	0,26%
TUP	SYN (L)	2	0,26%
TUP	TUP (N)	2	0,26%
ADI	CEI (N)	1	0,13%
ADI	BRR (L)	1	0,13%
ADI	BIX (R)	1	0,13%
ADI	ADX (L)	1	0,13%
ADI	SBI (N)	1	0,13%
ADI	MLI (N)	1	0,13%
ADX	SAN (L)	1	0,13%
ADX	WSS (N)	1	0,13%
ADX	SIX (L)	1	0,13%
ADX	SXN (D)	1	0,13%
ADX	ADI (N)	1	0,13%
ADX	TUP (N)	1	0,13%

AND	DUP (N)	1	0,13%
BIX	TUP (N)	1	0,13%
BIX	SIX (L)	1	0,13%
BIX	MLI (N)	1	0,13%
BIX	RSS (N)	1	0,13%
BIX	ADI (N)	1	0,13%
BRR	SBI (N)	1	0,13%
BRR	BIX (R)	1	0,13%
BRR	CEI (N)	1	0,13%
BRR	BRR (D)	1	0,13%
BRR	SPJ (D)	1	0,13%
BRR	ESP (L)	1	0,13%
BRR	ADX (L)	1	0,13%
BRR	BRR (L)	1	0,13%
BRR	FSS (N)	1	0,13%
BRR	AND (N)	1	0,13%
BRT	PRP (L)	1	0,13%
BRV	SBI (N)	1	0,13%
CEI	TUP (N)	1	0,13%
CRJ	BRR (D)	1	0,13%
CRJ	DUP (N)	1	0,13%
DMD	MLI (N)	1	0,13%
DMD	TUP (N)	1	0,13%
DUP	SFZ (L)	1	0,13%
DUP	BRV (L)	1	0,13%
DUP	DMD (L)	1	0,13%
DUP	SLB (L)	1	0,13%
DUP	SXI (L)	1	0,13%
DUP	MIV (N)	1	0,13%
IRC	SYN (R)	1	0,13%
KIL	KIL (N)	1	0,13%
MIV	RSS (N)	1	0,13%
MLI	DMD (L)	1	0,13%
POF	GAN (N)	1	0,13%
POF	SAZ (L)	1	0,13%
PRL	BIX (D)	1	0,13%
PRL	DRC (N)	1	0,13%
PRL	ADX (L)	1	0,13%
PRL	SXN (D)	1	0,13%
PRP	ADI (N)	1	0,13%
RSS	SBI (N)	1	0,13%
SAN	DMD (L)	1	0,13%
SAN	BRR (D)	1	0,13%
SAZ	DUP (N)	1	0,13%
SAZ	SYN (L)	1	0,13%
SAZ	SPJ (R)	1	0,13%
SAZ	ESP (L)	1	0,13%
SBI	PRL (L)	1	0,13%
SBI	SYN (L)	1	0,13%
SBI	SOI (R)	1	0,13%
SFZ	SAN (L)	1	0,13%
SFZ	SLB (L)	1	0,13%
SLB	ESP (L)	1	0,13%
SLB	SBI (N)	1	0,13%
SLB	BRR (L)	1	0,13%
SLB	DUP (N)	1	0,13%
SOI	TUP (N)	1	0,13%
SPJ	FSS (N)	1	0,13%
SPJ	SBI (N)	1	0,13%
SPJ	DUP (N)	1	0,13%
SPJ	CRJ (N)	1	0,13%
SWA	SYN (L)	1	0,13%
SWA	TUP (N)	1	0,13%
SXI	BIX (D)	1	0,13%
SXN	DUP (N)	1	0,13%
SXN	SXI (L)	1	0,13%
SYN	CRJ (N)	1	0,13%
SYN	IRC (N)	1	0,13%
SYN	TUP (N)	1	0,13%
SYN	SXN (D)	1	0,13%
SYN	SYN (R)	1	0,13%
TUP	DUP (N)	1	0,13%
TUP	BRR (L)	1	0,13%

TUP	SAN (L)	1	0,13%
TUP	BIX (R)	1	0,13%
TUP	BRR (R)	1	0,13%
TUP	SWA (L)	1	0,13%
TUP	POF (L)	1	0,13%
TUP	DRC (N)	1	0,13%

Dados para Análise Sequencial - Dinâmica.cod

Pares de Instruções	Repetições	Porcentagem
DUP DUP (N)	173	5,20%
DUP SYN (L)	148	4,45%
DUP PRL (L)	97	2,91%
SYN SYN (L)	73	2,19%
SYN DUP (N)	73	2,19%
SIX SOX (R)	67	2,01%
DUP SAZ (L)	59	1,77%
DUP RSS (N)	56	1,68%
DUP MIV (N)	56	1,68%
SYN ADX (L)	51	1,53%
DUP SIX (L)	49	1,47%
BRR DUP (N)	46	1,38%
DUP SYN (R)	46	1,38%
SYN PRL (L)	46	1,38%
PRL WSS (N)	40	1,20%
DUP SAN (L)	38	1,14%
CEI BRR (D)	37	1,11%
SIX SYN (L)	37	1,11%
BRT DUP (N)	36	1,08%
DUP SRC (N)	36	1,08%
GAN DUP (N)	36	1,08%
GSP DUP (N)	36	1,08%
SOX BRT (L)	36	1,08%
SRC KIL (N)	36	1,08%
WSS KIL (N)	36	1,08%
BRR SIX (L)	35	1,05%
DUP GSP (N)	35	1,05%
DUP TUP (N)	35	1,05%
OST SIX (L)	35	1,05%
DRC KIL (N)	34	1,02%
DUP CEI (N)	34	1,02%
DUP OST (N)	34	1,02%
DUP DRC (N)	32	0,96%
ADX DUP (N)	30	0,90%
PRL SAZ (L)	30	0,90%
SYN CRJ (N)	28	0,84%
MIV RSS (N)	27	0,81%
DUP BRR (D)	26	0,78%
PRL SIX (L)	26	0,78%
DUP WSS (N)	25	0,75%
SFZ SLB (L)	24	0,72%
DUP SBI (N)	22	0,66%
SYN MIV (N)	22	0,66%
DUP BIX (R)	21	0,63%
PRP SCD (R)	21	0,63%
BIX SAZ (L)	20	0,60%
DUP PRP (L)	20	0,60%
DUP BRR (L)	18	0,54%
SAN DUP (N)	18	0,54%
TUP DUP (N)	18	0,54%
TUP SYN (L)	18	0,54%
IRC SYN (R)	17	0,51%
RSS MLI (N)	17	0,51%
SYN SLB (L)	17	0,51%
BRR SXI (L)	16	0,48%
TUP TUP (N)	15	0,45%
DUP ADI (N)	14	0,42%
DUP SLB (L)	14	0,42%
SAN TUP (N)	14	0,42%
SLB MLX (L)	14	0,42%
SLB BIX (R)	14	0,42%
DUP MLI (N)	13	0,39%

SYN DRC (N)	13	0,39%
ADX SOX (R)	12	0,36%
DMD DUP (N)	12	0,36%
SCD KIL (N)	12	0,36%
DUP POF (L)	11	0,33%
ESP SBI (N)	11	0,33%
POF GAN (N)	11	0,33%
SAZ SYN (L)	11	0,33%
SBI ADI (N)	11	0,33%
ADI DUP (N)	10	0,30%
ADX SYN (L)	10	0,30%
DUP SFZ (L)	10	0,30%
MLI DUP (N)	10	0,30%
SPJ CRJ (N)	10	0,30%
DUP CGI (N)	9	0,27%
DUP IRC (N)	9	0,27%
MLI ADI (N)	9	0,27%
SAN BRR (D)	9	0,27%
SAZ SAN (L)	9	0,27%
SBI SOI (R)	9	0,27%
TUP POF (L)	9	0,27%
BRR TUP (N)	8	0,24%
DUP FSS (N)	8	0,24%
GAN TUP (N)	8	0,24%
MLX ADX (L)	8	0,24%
MLX DUP (N)	8	0,24%
POF SAZ (L)	8	0,24%
PRL SYN (L)	8	0,24%
PRP DUP (N)	8	0,24%
SAZ PRL (L)	8	0,24%
SAZ ADI (N)	8	0,24%
SAZ DUP (N)	8	0,24%
ADI TUP (N)	7	0,21%
BRR DRC (N)	7	0,21%
CRJ SPJ (R)	7	0,21%
DUP ADX (L)	7	0,21%
FLT DVR (N)	7	0,21%
FSS DUP (N)	7	0,21%
SAZ MIV (N)	7	0,21%
SOI TUP (N)	7	0,21%
SPJ DUP (N)	7	0,21%
SXI FSS (N)	7	0,21%
CEI TUP (N)	6	0,18%
CRJ SLB (L)	6	0,18%
CRJ DUP (N)	6	0,18%
DUP CRJ (N)	6	0,18%
DUP BRR (R)	6	0,18%
DUP GAN (N)	6	0,18%
PRL PRL (L)	6	0,18%
SAZ RSS (N)	6	0,18%
SAZ TUP (N)	6	0,18%
SLB TUP (N)	6	0,18%
SPL SAZ (L)	6	0,18%
ADI ADI (N)	5	0,15%
BIX BIX (R)	5	0,15%
CRJ SYN (L)	5	0,15%
DUP DMD (L)	5	0,15%
PRL DRC (N)	5	0,15%
PRP SLB (L)	5	0,15%
SLB DUP (N)	5	0,15%
SLB BRR (L)	5	0,15%
SPJ SBI (N)	5	0,15%
ADI SBI (N)	4	0,12%
ADI SXI (L)	4	0,12%
ADI ADX (L)	4	0,12%
ADX SXN (D)	4	0,12%
BRR PRL (L)	4	0,12%
BRR SYN (L)	4	0,12%
BRV SYN (L)	4	0,12%
CRJ SXI (L)	4	0,12%
CRJ PRL (L)	4	0,12%
DUP SWA (L)	4	0,12%
DUP ESP (L)	4	0,12%

DUP	SXI (L)	4	0,12%	DMD	BIX (R)	2	0,06%
GAN	KIL (N)	4	0,12%	DMD	SYN (L)	2	0,06%
GAN	SFZ (L)	4	0,12%	DRM	MIV (N)	2	0,06%
KIL	KIL (N)	4	0,12%	DUP	MLX (L)	2	0,06%
MIV	PRL (L)	4	0,12%	DUP	BIX (L)	2	0,06%
MLI	SBI (N)	4	0,12%	DUP	SCD (R)	2	0,06%
PRP	MLR (N)	4	0,12%	DUP	MSP (L)	2	0,06%
RSS	FLT (N)	4	0,12%	DUP	DVR (N)	2	0,06%
RSS	SYN (L)	4	0,12%	DUP	FLT (N)	2	0,06%
SAZ	BIX (R)	4	0,12%	DUP	SPJ (R)	2	0,06%
SBI	CEI (N)	4	0,12%	DVR	EXR (N)	2	0,06%
SBI	DUP (N)	4	0,12%	EXR	DUP (N)	2	0,06%
SLB	SPJ (R)	4	0,12%	FLT	MLR (N)	2	0,06%
SLB	ESP (L)	4	0,12%	MIV	ADI (N)	2	0,06%
SLB	SXN (L)	4	0,12%	MLI	SYN (L)	2	0,06%
SXI	DUP (N)	4	0,12%	MLI	MLI (N)	2	0,06%
SXI	SYN (L)	4	0,12%	MLR	TUP (N)	2	0,06%
SXI	PRL (L)	4	0,12%	MLR	ADR (N)	2	0,06%
SXN	SPJ (L)	4	0,12%	MLR	GAN (N)	2	0,06%
SXN	SXI (L)	4	0,12%	MLR	MLR (N)	2	0,06%
SYN	SIX (L)	4	0,12%	MSP	SAN (L)	2	0,06%
SYN	SXN (D)	4	0,12%	OPT	KIL (N)	2	0,06%
SYN	IRC (N)	4	0,12%	PRL	CEI (N)	2	0,06%
SYN	WSS (N)	4	0,12%	PRL	MLX (L)	2	0,06%
SYN	BRR (L)	4	0,12%	PRL	SLB (L)	2	0,06%
TUP	SAZ (L)	4	0,12%	SAN	SQR (N)	2	0,06%
TUP	BRR (L)	4	0,12%	SAN	BRR (L)	2	0,06%
ADI	DMD (R)	3	0,09%	SAZ	ADX (L)	2	0,06%
ADI	SYN (L)	3	0,09%	SAZ	FSS (N)	2	0,06%
ADI	PRL (L)	3	0,09%	SAZ	GAN (N)	2	0,06%
ADX	PRL (L)	3	0,09%	SAZ	BRR (D)	2	0,06%
ADX	TUP (N)	3	0,09%	SAZ	SLB (L)	2	0,06%
BIX	GAN (N)	3	0,09%	SBI	DRM (L)	2	0,06%
BRR	ADI (N)	3	0,09%	SBI	FLT (N)	2	0,06%
BRR	SAN (L)	3	0,09%	SFZ	TUP (N)	2	0,06%
DMD	ADI (N)	3	0,09%	SFZ	DUP (N)	2	0,06%
DVR	ADR (N)	3	0,09%	SIX	DUP (N)	2	0,06%
FSS	SPL (D)	3	0,09%	SLB	SYN (L)	2	0,06%
MLI	DMD (L)	3	0,09%	SPJ	SYN (R)	2	0,06%
PRL	TUP (N)	3	0,09%	SPJ	CEI (N)	2	0,06%
PRP	PRL (L)	3	0,09%	SWA	TUP (N)	2	0,06%
SAN	CEI (N)	3	0,09%	SXI	TUP (N)	2	0,06%
SAZ	MLR (N)	3	0,09%	SXI	WSS (N)	2	0,06%
SAZ	SPJ (D)	3	0,09%	SXI	IRC (N)	2	0,06%
SAZ	SBI (N)	3	0,09%	SXI	SLB (L)	2	0,06%
SAZ	ESP (L)	3	0,09%	SXO	DUP (N)	2	0,06%
SBI	MLI (N)	3	0,09%	SYN	ADI (N)	2	0,06%
SBI	DMD (L)	3	0,09%	SYN	TUP (N)	2	0,06%
SLB	SXO (R)	3	0,09%	SYN	BRR (D)	2	0,06%
SLB	DRC (N)	3	0,09%	SYN	WDS (N)	2	0,06%
SPJ	POF (L)	3	0,09%	TUP	MIV (N)	2	0,06%
SPJ	BRR (L)	3	0,09%	TUP	DRC (N)	2	0,06%
SXI	SIX (L)	3	0,09%	TUP	PRL (L)	2	0,06%
SXN	DUP (N)	3	0,09%	TUP	BIX (R)	2	0,06%
SYN	MLI (N)	3	0,09%	TUP	MLI (N)	2	0,06%
SYN	SXO (R)	3	0,09%	WDS	KIL (N)	2	0,06%
SYN	SYN (R)	3	0,09%	YXI	SIX (L)	2	0,06%
TUP	RSS (N)	3	0,09%	ABI	SIX (L)	1	0,03%
TUP	SAN (L)	3	0,09%	ABI	DUP (N)	1	0,03%
YAN	SFZ (L)	3	0,09%	ADI	SAZ (L)	1	0,03%
ADI	DMD (L)	2	0,06%	ADI	MAP (L)	1	0,03%
ADR	ADR (N)	2	0,06%	ADI	YAN (N)	1	0,03%
ADX	FSS (N)	2	0,06%	ADI	GSP (N)	1	0,03%
ADX	BRR (D)	2	0,06%	ADR	SBI (N)	1	0,03%
ADX	GSP (N)	2	0,06%	ADR	DUP (N)	1	0,03%
ADX	DRC (N)	2	0,06%	ADR	SYN (L)	1	0,03%
BRR	SLB (L)	2	0,06%	ADR	SIX (L)	1	0,03%
BRR	BIX (R)	2	0,06%	ADR	PRL (L)	1	0,03%
CEI	DUP (N)	2	0,06%	ADX	ADI (N)	1	0,03%
CGI	DUP (N)	2	0,06%	ADX	SIX (L)	1	0,03%
CRJ	SPJ (D)	2	0,06%	ADX	SAZ (L)	1	0,03%
CRJ	SCD (R)	2	0,06%	ADX	WSS (N)	1	0,03%
DMD	CEI (N)	2	0,06%	BIX	DUP (N)	1	0,03%

BIX	DMD (L)	1	0,03%
BIX	DMD (R)	1	0,03%
BIX	YXI (N)	1	0,03%
BIX	SYN (L)	1	0,03%
BIX	ADI (N)	1	0,03%
BRR	SBI (N)	1	0,03%
BRR	MLI (N)	1	0,03%
BRR	DMD (R)	1	0,03%
BRR	BRR (R)	1	0,03%
BRR	BRR (D)	1	0,03%
BRR	YAN (N)	1	0,03%
BRR	ABI (N)	1	0,03%
CEI	SLB (L)	1	0,03%
CGI	TUP (N)	1	0,03%
CGI	SLB (L)	1	0,03%
CLI	SLB (L)	1	0,03%
COS	SCD (R)	1	0,03%
CRJ	BIX (R)	1	0,03%
CRJ	BRV (R)	1	0,03%
CRJ	SAN (L)	1	0,03%
DMD	TUP (N)	1	0,03%
DMD	SIX (L)	1	0,03%
DSR	DUP (N)	1	0,03%
DSR	RSS (N)	1	0,03%
DSR	SAN (L)	1	0,03%
DSR	MIV (N)	1	0,03%
DUP	SPJ (L)	1	0,03%
DUP	OPT (R)	1	0,03%
DUP	MAP (L)	1	0,03%
DUP	BRV (R)	1	0,03%
DUP	SYN (D)	1	0,03%
DUP	ESS (N)	1	0,03%
DUP	CLI (N)	1	0,03%
DUP	SPJ (D)	1	0,03%
DUP	DMD (R)	1	0,03%
DUP	WDS (N)	1	0,03%
DVR	DUP (N)	1	0,03%
DVR	SBR (N)	1	0,03%
DVR	PRL (L)	1	0,03%
DVR	SYN (R)	1	0,03%
DVR	SBI (N)	1	0,03%
ERC	MRC (N)	1	0,03%
ESP	SYN (L)	1	0,03%
ESS	SYN (R)	1	0,03%
GAN	ADI (N)	1	0,03%
GAN	DRC (N)	1	0,03%
LGN	MLR (N)	1	0,03%
MAP	WDS (N)	1	0,03%
MID	DSR (D)	1	0,03%
MID	PRL (L)	1	0,03%
MIV	ESS (N)	1	0,03%
MIV	FLT (N)	1	0,03%
MLI	DMD (R)	1	0,03%
MLI	SAN (L)	1	0,03%
MLI	BRR (L)	1	0,03%
MLI	GAN (N)	1	0,03%
MLR	SLB (L)	1	0,03%
MLR	PRL (L)	1	0,03%
MLR	DRC (N)	1	0,03%
MRC	SYN (D)	1	0,03%
NGR	MLR (N)	1	0,03%
OST	PRL (L)	1	0,03%
POF	MIV (N)	1	0,03%
PRL	DVR (N)	1	0,03%
PRL	DSR (D)	1	0,03%
PRL	DUP (N)	1	0,03%
PRL	MID (N)	1	0,03%
PRP	SYN (L)	1	0,03%
PRP	SAN (L)	1	0,03%
PRP	MLI (N)	1	0,03%
RSS	ADR (N)	1	0,03%
RSS	SBR (N)	1	0,03%
RSS	MLR (N)	1	0,03%

RSS	PRL (L)	1	0,03%
RSS	MIV (N)	1	0,03%
RSS	DMD (R)	1	0,03%
RSS	DMD (L)	1	0,03%
RSS	CEI (N)	1	0,03%
SAN	COS (N)	1	0,03%
SAN	FLT (N)	1	0,03%
SAN	LGN (N)	1	0,03%
SAN	SYN (L)	1	0,03%
SAZ	MLI (N)	1	0,03%
SAZ	MID (N)	1	0,03%
SAZ	DSR (D)	1	0,03%
SAZ	SIX (L)	1	0,03%
SAZ	ADR (N)	1	0,03%
SAZ	DVR (N)	1	0,03%
SAZ	SPJ (L)	1	0,03%
SAZ	IRC (N)	1	0,03%
SAZ	SAZ (L)	1	0,03%
SBI	PRL (L)	1	0,03%
SBI	MID (N)	1	0,03%
SBI	CRJ (N)	1	0,03%
SBR	SIX (L)	1	0,03%
SBR	PRL (L)	1	0,03%
SFZ	SAZ (L)	1	0,03%
SIX	MLI (N)	1	0,03%
SIX	CEI (N)	1	0,03%
SIX	PRL (L)	1	0,03%
SIX	DRC (N)	1	0,03%
SLB	SIX (L)	1	0,03%
SLB	BRR (D)	1	0,03%
SLB	MLI (N)	1	0,03%
SOI	DUP (N)	1	0,03%
SOI	SYN (L)	1	0,03%
SPJ	TUP (N)	1	0,03%
SPJ	DRC (N)	1	0,03%
SPJ	SYN (L)	1	0,03%
SPJ	BRV (R)	1	0,03%
SQR	TUP (N)	1	0,03%
SWA	ADI (N)	1	0,03%
SWA	DUP (N)	1	0,03%
SWA	YAN (N)	1	0,03%
SWA	SYN (R)	1	0,03%
SXI	SYN (R)	1	0,03%
SXI	BIX (R)	1	0,03%
SXN	DRC (N)	1	0,03%
SXO	BRR (R)	1	0,03%
SYN	ERC (N)	1	0,03%
SYN	BRV (R)	1	0,03%
SYN	ESP (L)	1	0,03%
SYN	CEI (N)	1	0,03%
SYN	FLT (N)	1	0,03%
SYN	GSP (N)	1	0,03%
SYN	NGR (N)	1	0,03%
SYN	GAN (N)	1	0,03%
SYN	SCD (R)	1	0,03%
SYN	SXI (L)	1	0,03%
SYN	OPT (R)	1	0,03%
SYN	BIX (R)	1	0,03%
SYN	MLX (L)	1	0,03%
TUP	BIX (D)	1	0,03%
TUP	SWA (L)	1	0,03%
TUP	SBI (N)	1	0,03%
TUP	SLB (L)	1	0,03%
TUP	DMD (L)	1	0,03%
TUP	BRR (D)	1	0,03%

Dados para Análise Seqüencial - Simple.cod

Pares de Instruções	Repetições	Porcentagem
DUP DUP (N)	1099	7,31 %
DUP SYN (L)	486	3,23 %
SYN DUP (N)	420	2,79 %

DUP	PRL (L)	373	2,48%
DUP	SIX (L)	286	1,90%
DUP	SYN (R)	281	1,87%
BRR	DUP (N)	215	1,43%
DUP	MID (N)	181	1,20%
DUP	ADX (L)	179	1,19%
MID	DSR (D)	157	1,04%
DUP	ADI (N)	156	1,04%
SYN	SYN (L)	148	0,96%
SWA	KIL (N)	143	0,95%
YAN	KIL (N)	143	0,95%
DUP	WSS (N)	141	0,94%
SIX	SOX (R)	139	0,92%
DSR	DUP (N)	138	0,92%
SYN	ADX (L)	133	0,88%
DUP	OST (N)	132	0,88%
DUP	BRR (D)	131	0,87%
DUP	CEI (N)	131	0,87%
ADX	DUP (N)	125	0,83%
SYN	PRL (L)	121	0,80%
PRL	DUP (N)	119	0,79%
DRC	KIL (N)	118	0,78%
DUP	SRC (N)	116	0,77%
ADI	DUP (N)	112	0,74%
DUP	SXI (L)	111	0,74%
DUP	GSP (N)	104	0,69%
BRR	SIX (L)	100	0,66%
BRT	DUP (N)	99	0,66%
GSP	DUP (N)	96	0,64%
SIX	SYN (L)	96	0,64%
DUP	SAZ (L)	94	0,62%
SIX	DUP (N)	89	0,59%
DUP	SBI (N)	87	0,58%
DUP	TUP (N)	87	0,58%
CEI	BRR (D)	84	0,56%
DUP	DSR (D)	84	0,56%
PRP	SCD (R)	84	0,56%
DUP	IRC (N)	82	0,55%
PRL	WSS (N)	80	0,53%
SAN	DUP (N)	77	0,51%
OST	SIX (L)	76	0,51%
SOX	BRT (L)	74	0,49%
DUP	CRJ (N)	71	0,47%
SAZ	PRL (L)	71	0,47%
DSR	MID (N)	68	0,45%
SXI	SIX (L)	68	0,45%
SXI	WSS (N)	68	0,45%
SBI	DUP (N)	67	0,45%
PRL	SAZ (L)	66	0,44%
RSS	DUP (N)	64	0,43%
MID	DUP (N)	61	0,41%
PRL	SIX (L)	58	0,39%
PRL	TUP (N)	57	0,38%
DUP	SLB (L)	56	0,37%
TUP	TUP (N)	56	0,37%
BRR	SYN (L)	51	0,34%
DSR	RSS (N)	51	0,34%
SYN	SIX (L)	50	0,33%
ADX	SOX (R)	49	0,33%
DUP	SOX (R)	48	0,32%
SXI	DUP (N)	46	0,31%
DSR	MIV (N)	45	0,30%
MIV	RSS (N)	45	0,30%
SBI	MID (N)	44	0,29%
SPJ	DUP (N)	44	0,29%
DUP	MLX (L)	43	0,29%
DUP	BRT (L)	41	0,27%
PRL	SYN (L)	41	0,27%
SAZ	DUP (N)	41	0,27%
SBI	ADI (N)	41	0,27%
TUP	DUP (N)	41	0,27%
KIL	SYN (R)	40	0,27%
ADI	SXI (L)	39	0,26%

DSR	SYN (L)	39	0,26%
FSS	DUP (N)	39	0,26%
DSR	DSR (L)	38	0,25%
MIV	DUP (N)	38	0,25%
SYN	SXI (L)	37	0,25%
SYN	DRC (N)	37	0,25%
TUP	SYN (L)	37	0,25%
CEI	DUP (N)	36	0,24%
DSR	SYN (R)	36	0,24%
TUP	SYN (R)	36	0,24%
CRJ	DUP (N)	35	0,23%
MLX	DUP (N)	35	0,23%
SLB	MLX (L)	34	0,23%
SAN	SYN (L)	33	0,22%
ADX	FSS (N)	32	0,21%
BRR	ADX (L)	32	0,21%
DSR	SAN (L)	32	0,21%
MID	DSR (L)	32	0,21%
ADR	DUP (N)	31	0,21%
BIX	SAZ (L)	31	0,21%
DSR	SBR (N)	31	0,21%
DUP	DRC (N)	31	0,21%
MLI	DUP (N)	31	0,21%
MLR	DUP (N)	31	0,21%
OST	DUP (N)	31	0,21%
SYN	TUP (N)	31	0,21%
ORB	KIL (N)	30	0,20%
SOX	DUP (N)	30	0,20%
ADR	MLR (N)	29	0,19%
DUP	ESP (L)	29	0,19%
SAN	SXI (L)	29	0,19%
ADI	MID (N)	28	0,19%
SYN	IRC (N)	28	0,19%
SYN	BRR (D)	28	0,19%
DSR	DSR (R)	27	0,18%
WDS	SYN (R)	27	0,18%
ADI	SBI (N)	26	0,17%
CRJ	SYN (L)	26	0,17%
DSR	TUP (N)	26	0,17%
ESP	SBI (N)	26	0,17%
RSS	ADR (N)	26	0,17%
DUP	BIX (R)	25	0,17%
GAN	TUP (N)	25	0,17%
DSR	DSR (D)	24	0,16%
DSR	ADR (N)	24	0,16%
SYN	CRJ (N)	24	0,16%
BRR	PRL (L)	23	0,15%
DUP	SPJ (R)	23	0,15%
DUP	ADR (N)	23	0,15%
MLX	ADX (L)	23	0,15%
SYN	WSS (N)	23	0,15%
TUP	IRC (N)	23	0,15%
BIX	DUP (N)	22	0,15%
DUP	RSS (N)	22	0,15%
DUP	MLR (N)	22	0,15%
MID	SYN (L)	22	0,15%
PRL	MID (N)	22	0,15%
SFZ	KIL (N)	22	0,15%
SXI	SYN (L)	22	0,15%
DUP	BIX (D)	21	0,14%
SBR	DUP (N)	21	0,14%
SYN	SRC (N)	21	0,14%
SYN	ESP (L)	21	0,14%
SYN	SBI (N)	21	0,14%
TUP	SXI (L)	21	0,14%
BRR	SYN (R)	20	0,13%
DUP	PRP (L)	20	0,13%
DUP	POF (L)	20	0,13%
SBR	MLR (N)	20	0,13%
TUP	MID (N)	20	0,13%
DUP	MLI (N)	19	0,13%
DUP	FSS (N)	19	0,13%
DUP	GAN (N)	19	0,13%

GAN	DUP (N)	19	0,13%	SYN	SPJ (R)	12	0,08%
MID	DSR (R)	19	0,13%	ADX	SIX (L)	11	0,07%
SYN	MLX (L)	19	0,13%	BRR	MLX (L)	11	0,07%
ADI	MIV (N)	18	0,12%	CRJ	SLB (L)	11	0,07%
DSR	ADX (L)	18	0,12%	CRJ	PRL (L)	11	0,07%
DSR	SIX (L)	18	0,12%	CRJ	TUP (N)	11	0,07%
PRL	PRL (L)	18	0,12%	DSR	BRT (L)	11	0,07%
PRP	DUP (N)	18	0,12%	DUP	SFZ (L)	11	0,07%
SAN	SAZ (L)	18	0,12%	MLR	SYN (L)	11	0,07%
SAZ	TUP (N)	18	0,12%	PRL	ADX (L)	11	0,07%
SOI	TUP (N)	18	0,12%	PRP	SYN (R)	11	0,07%
SYN	ADI (N)	18	0,12%	PRL	IRC (N)	11	0,07%
SYN	DSR (D)	18	0,12%	SAZ	DSR (D)	11	0,07%
SYN	MID (N)	18	0,12%	SFZ	DUP (N)	11	0,07%
BIX	SIX (L)	17	0,11%	SIX	CEI (N)	11	0,07%
PRL	DSR (D)	17	0,11%	SXI	ADX (L)	11	0,07%
PRL	ADI (N)	17	0,11%	BIX	SYN (R)	10	0,07%
SAZ	SYN (R)	17	0,11%	BRR	ESP (L)	10	0,07%
SAZ	MID (N)	17	0,11%	BRR	WSS (N)	10	0,07%
SAZ	SYN (L)	17	0,11%	BRR	ADR (N)	10	0,07%
SIX	PRL (L)	17	0,11%	DUP	BRR (R)	10	0,07%
SLB	TUP (N)	17	0,11%	IRC	DUP (N)	10	0,07%
ADR	ADR (N)	16	0,11%	MID	SIX (L)	10	0,07%
ADX	ADX (L)	16	0,11%	MID	IRC (N)	10	0,07%
DSR	SRC (N)	16	0,11%	POF	GAN (N)	10	0,07%
ESP	DUP (N)	16	0,11%	PRL	SXI (L)	10	0,07%
MID	BIX (R)	16	0,11%	PRP	PRL (L)	10	0,07%
MLR	ADR (N)	16	0,11%	SBI	SYN (R)	10	0,07%
RSS	SYN (L)	16	0,11%	SIX	DSR (D)	10	0,07%
SIX	SIX (L)	16	0,11%	SYN	OST (N)	10	0,07%
SIX	ADX (L)	16	0,11%	SYN	BRT (L)	10	0,07%
SLB	BIX (R)	16	0,11%	WSS	KIL (N)	10	0,07%
SPJ	ADX (L)	16	0,11%	ADI	PRL (L)	9	0,06%
SYN	CEI (N)	16	0,11%	ADX	SYN (R)	9	0,06%
BRR	CRJ (N)	15	0,10%	BRR	MIV (N)	9	0,06%
BRR	BRR (D)	15	0,10%	BRR	SBI (N)	9	0,06%
CEI	SYN (L)	15	0,10%	BRR	SRC (N)	9	0,06%
DUP	MIV (N)	15	0,10%	DSR	SOX (R)	9	0,06%
DUP	SAN (L)	15	0,10%	DSR	MLI (N)	9	0,06%
DUP	DVR (N)	15	0,10%	DUP	BRR (L)	9	0,06%
SBI	SOI (R)	15	0,10%	DUP	DSR (L)	9	0,06%
SLB	DUP (N)	15	0,10%	DUP	SOR (L)	9	0,06%
SPJ	SBI (N)	15	0,10%	MIV	SYN (R)	9	0,06%
SYN	SLB (L)	15	0,10%	MID	PRL (L)	9	0,06%
TUP	SAZ (L)	15	0,10%	MIV	SIX (L)	9	0,06%
BRR	ADI (N)	14	0,09%	MLR	PRL (L)	9	0,06%
DSR	IRC (N)	14	0,09%	SAN	SYN (R)	9	0,06%
DSR	PRL (L)	14	0,09%	RSS	PRL (L)	9	0,06%
DVR	DUP (N)	14	0,09%	SAN	CEI (N)	9	0,06%
MLR	SBR (N)	14	0,09%	SAZ	CEI (N)	9	0,06%
PRL	MLX (L)	14	0,09%	SPJ	CEI (N)	9	0,06%
RSS	TUP (N)	14	0,09%	SYN	FSS (N)	9	0,06%
ADI	SYN (L)	13	0,09%	SYN	GSP (N)	9	0,06%
ADI	ADI (N)	13	0,09%	TUP	SBI (N)	9	0,06%
ADX	SYN (L)	13	0,09%	TUP	SAN (L)	9	0,06%
BRR	BRT (L)	13	0,09%	ADI	SIX (L)	8	0,05%
BRR	MID (N)	13	0,09%	ADI	ADX (L)	8	0,05%
DSR	CEI (N)	13	0,09%	ADR	SIX (L)	8	0,05%
DSR	BIX (R)	13	0,09%	BIX	SYN (L)	8	0,05%
SFZ	SLB (L)	13	0,09%	BRR	CEI (N)	8	0,05%
TUP	SLB (L)	13	0,09%	BRR	DSR (D)	8	0,05%
TUP	PRL (L)	13	0,09%	DSR	CRJ (N)	8	0,05%
TUP	POF (L)	13	0,09%	DUP	DSR (R)	8	0,05%
ADI	TUP (N)	12	0,08%	DUP	SPJ (D)	8	0,05%
ADX	PRL (L)	12	0,08%	DUP	SQR (N)	8	0,05%
CRJ	SXI (L)	12	0,08%	DUP	BIX (L)	8	0,05%
DSR	BRR (D)	12	0,08%	MLX	SYN (R)	8	0,05%
DSR	ADI (N)	12	0,08%	MLR	SIX (L)	8	0,05%
DUP	SCD (R)	12	0,08%	OST	SYN (L)	8	0,05%
RSS	SIX (L)	12	0,08%	POF	DUP (N)	8	0,05%
SAZ	ADI (N)	12	0,08%	POF	SAZ (L)	8	0,05%
SBI	ADX (L)	12	0,08%	SAN	MLI (N)	8	0,05%
SXI	PRL (L)	12	0,08%	SAZ	BRT (L)	8	0,05%

SBI	SIX (L)	8	0,05%
SBI	BIX (R)	8	0,05%
SBR	SYN (R)	8	0,05%
SBI	SBI (N)	8	0,05%
SBR	SYN (L)	8	0,05%
SIX	WSS (N)	8	0,05%
SIX	SYN (R)	8	0,05%
SPJ	CRJ (N)	8	0,05%
SXN	SYN (R)	8	0,05%
ADR	PRL (L)	7	0,05%
ADX	GSP (N)	7	0,05%
BIX	ADX (L)	7	0,05%
BRR	SLB (L)	7	0,05%
BRR	SOX (R)	7	0,05%
BRR	TUP (N)	7	0,05%
BRV	SBI (N)	7	0,05%
CEI	SLB (L)	7	0,05%
DSR	WSS (N)	7	0,05%
DSR	GSP (N)	7	0,05%
GAN	SFZ (L)	7	0,05%
MID	ADX (L)	7	0,05%
OST	CEI (N)	7	0,05%
PRL	SRC (N)	7	0,05%
SAN	SOX (R)	7	0,05%
SBI	PRL (L)	7	0,05%
SIX	OST (N)	7	0,05%
SOX	ADX (L)	7	0,05%
SPJ	SYN (L)	7	0,05%
SPJ	PRL (L)	7	0,05%
SXI	TUP (N)	7	0,05%
SXI	BRT (L)	7	0,05%
SYN	SOI (R)	7	0,05%
SYN	SPJ (D)	7	0,05%
TUP	CRJ (N)	7	0,05%
ADI	SYN (R)	6	0,04%
ADR	ADX (L)	6	0,04%
BIX	TUP (N)	6	0,04%
BIX	PRL (L)	6	0,04%
BIX	MID (N)	6	0,04%
BRR	SOR (L)	6	0,04%
BRR	FSS (N)	6	0,04%
BRT	CEI (N)	6	0,04%
BRT	SYN (L)	6	0,04%
BRT	SIX (L)	6	0,04%
CRJ	SOX (R)	6	0,04%
DSR	SBI (N)	6	0,04%
DUP	SOI (R)	6	0,04%
DUP	SWA (L)	6	0,04%
ERC	SCD (R)	6	0,04%
FSS	ADX (L)	6	0,04%
GSP	SYN (L)	6	0,04%
MLR	DSR (D)	6	0,04%
POF	TUP (N)	6	0,04%
PRL	BIX (D)	6	0,04%
PRL	SOX (R)	6	0,04%
RSS	SXI (L)	6	0,04%
RSS	MLR (N)	6	0,04%
RSS	CEI (N)	6	0,04%
SAN	ADX (L)	6	0,04%
SAN	TUP (N)	6	0,04%
SAN	PRL (L)	6	0,04%
SAZ	IRC (N)	6	0,04%
SAZ	SIX (L)	6	0,04%
SAZ	SAN (L)	6	0,04%
SBI	CEI (N)	6	0,04%
SFZ	TUP (N)	6	0,04%
SIX	GSP (N)	6	0,04%
SIX	CRJ (N)	6	0,04%
SIX	MID (N)	6	0,04%
SIX	SOI (R)	6	0,04%
SLB	PRL (L)	6	0,04%
SLB	SIX (L)	6	0,04%
SOX	SYN (L)	6	0,04%

SPJ	SLB (L)	6	0,04%
SXI	OST (N)	6	0,04%
SYN	ADR (N)	6	0,04%
SYN	BIX (R)	6	0,04%
TUP	DSR (D)	6	0,04%
TUP	BIX (R)	6	0,04%
TUP	MLI (N)	6	0,04%
ADR	BIX (D)	5	0,03%
ADR	SLB (L)	5	0,03%
ADR	SYN (L)	5	0,03%
ADX	DSR (D)	5	0,03%
ADX	MID (N)	5	0,03%
BIX	GSP (N)	5	0,03%
BRR	RSS (N)	5	0,03%
BRR	OST (N)	5	0,03%
CEI	WSS (N)	5	0,03%
CEI	MID (N)	5	0,03%
CEI	SYN (R)	5	0,03%
CEI	PRL (L)	5	0,03%
CEI	ADX (L)	5	0,03%
CGR	DUP (N)	5	0,03%
DUP	WDS (N)	5	0,03%
DUP	SYN (D)	5	0,03%
ESP	MRC (N)	5	0,03%
GAN	SXI (L)	5	0,03%
GAN	MLI (N)	5	0,03%
MID	SLB (L)	5	0,03%
MID	GSP (N)	5	0,03%
MID	SOX (R)	5	0,03%
MLR	SOX (R)	5	0,03%
MLR	SRC (N)	5	0,03%
MLR	SLB (L)	5	0,03%
MLR	ADX (L)	5	0,03%
MRC	SYN (D)	5	0,03%
PRL	SBI (N)	5	0,03%
PRL	ADR (N)	5	0,03%
PRL	GSP (N)	5	0,03%
PRL	BRT (L)	5	0,03%
PRL	CRJ (N)	5	0,03%
PRP	SLB (L)	5	0,03%
PRP	SYN (L)	5	0,03%
RSS	BRR (D)	5	0,03%
SAN	SRC (N)	5	0,03%
SAZ	SAZ (L)	5	0,03%
SAZ	CRJ (N)	5	0,03%
SAZ	BIX (R)	5	0,03%
SAZ	GAN (N)	5	0,03%
SBI	SRC (N)	5	0,03%
SBI	SYN (L)	5	0,03%
SIX	MLX (L)	5	0,03%
SIX	ADR (N)	5	0,03%
SIX	SOR (L)	5	0,03%
SLB	SYN (L)	5	0,03%
SLB	IRC (N)	5	0,03%
SOX	DUP (N)	5	0,03%
SOX	PRL (L)	5	0,03%
SOX	SIX (L)	5	0,03%
SYN	ERC (N)	5	0,03%
SYN	SOX (R)	5	0,03%
SYN	SCD (R)	5	0,03%
TUP	CEI (N)	5	0,03%
ADI	WSS (N)	4	0,03%
ADI	SRC (N)	4	0,03%
ADR	FSS (N)	4	0,03%
ADR	SYN (R)	4	0,03%
ADR	SRC (N)	4	0,03%
ADX	MLX (L)	4	0,03%
ADX	CEI (N)	4	0,03%
ADX	WSS (N)	4	0,03%
ADX	OST (N)	4	0,03%
ADX	SXI (L)	4	0,03%
BIX	BIX (R)	4	0,03%
BRT	ADI (N)	4	0,03%

BRT	GSP (N)	4	0,03%	ADI	GSP (N)	3	0,02%
BRT	PRL (L)	4	0,03%	ADI	MLX (L)	3	0,02%
BRT	OST (N)	4	0,03%	ADR	BRR (D)	3	0,02%
CRJ	SPJ (R)	4	0,03%	ADX	SAZ (L)	3	0,02%
DSR	OST (N)	4	0,03%	ADX	SOI (R)	3	0,02%
DSR	SPJ (R)	4	0,03%	BIX	ADR (N)	3	0,02%
DVR	PRL (L)	4	0,03%	BIX	FSS (N)	3	0,02%
DVR	SYN (L)	4	0,03%	BIX	SPJ (R)	3	0,02%
ESP	ADX (L)	4	0,03%	BIX	CEI (N)	3	0,02%
FSS	SIX (L)	4	0,03%	BIX	WSS (N)	3	0,02%
GAN	CRJ (N)	4	0,03%	BRR	SBR (N)	3	0,02%
GAN	CEI (N)	4	0,03%	BRR	BIX (D)	3	0,02%
GAN	SIX (L)	4	0,03%	BRR	MLI (N)	3	0,02%
GSP	SIX (L)	4	0,03%	BRR	SXI (L)	3	0,02%
GSP	RSS (N)	4	0,03%	BRT	ADX (L)	3	0,02%
GSP	ADX (L)	4	0,03%	BRT	SYN (R)	3	0,02%
IRC	SYN (L)	4	0,03%	BRT	WSS (N)	3	0,02%
MID	KIL (N)	4	0,03%	BRT	SBI (N)	3	0,02%
MID	BRT (L)	4	0,03%	CEI	BRT (L)	3	0,02%
MID	BRR (D)	4	0,03%	CEI	DSR (D)	3	0,02%
MID	WSS (N)	4	0,03%	CEI	MLX (L)	3	0,02%
MIV	BRR (D)	4	0,03%	CEI	SIX (L)	3	0,02%
MIV	SRC (N)	4	0,03%	CEI	ADI (N)	3	0,02%
MIV	PRL (L)	4	0,03%	CEI	CEI (N)	3	0,02%
MLI	SIX (L)	4	0,03%	CRJ	BIX (R)	3	0,02%
MLR	MLR (N)	4	0,03%	DMD	DUP (N)	3	0,02%
MLX	PRL (L)	4	0,03%	DSR	BRV (L)	3	0,02%
NOT	MLR (N)	4	0,03%	DSR	SXI (L)	3	0,02%
NOT	PRL (L)	4	0,03%	DUP	SBR (N)	3	0,02%
OST	PRL (L)	4	0,03%	DVR	IRC (N)	3	0,02%
OST	ADX (L)	4	0,03%	DVR	SLB (L)	3	0,02%
POF	SYN (L)	4	0,03%	DVR	SIX (L)	3	0,02%
PRL	MIV (N)	4	0,03%	ESP	PRL (L)	3	0,02%
PRL	MLR (N)	4	0,03%	ESP	DSR (D)	3	0,02%
PRL	BRR (D)	4	0,03%	GAN	SYN (R)	3	0,02%
PRL	FSS (N)	4	0,03%	FSS	SYN (L)	3	0,02%
PRL	CEI (N)	4	0,03%	GAN	SYN (L)	3	0,02%
RSS	MID (N)	4	0,03%	GAN	GSP (N)	3	0,02%
RSS	MLI (N)	4	0,03%	GSP	SYN (R)	3	0,02%
RSS	FSS (N)	4	0,03%	GSP	SXI (L)	3	0,02%
RSS	DSR (D)	4	0,03%	GSP	ADI (N)	3	0,02%
RSS	NGR (N)	4	0,03%	IRC	WSS (N)	3	0,02%
RSS	SRC (N)	4	0,03%	MID	OST (N)	3	0,02%
SAN	MLR (N)	4	0,03%	MID	ADI (N)	3	0,02%
SAN	GSP (N)	4	0,03%	MID	CEI (N)	3	0,02%
SAZ	SBI (N)	4	0,03%	MID	SRC (N)	3	0,02%
SBI	SOX (R)	4	0,03%	MID	SOR (L)	3	0,02%
SBI	CRJ (N)	4	0,03%	MID	CRJ (N)	3	0,02%
SBI	DSR (D)	4	0,03%	MIV	DSR (D)	3	0,02%
SBR	SIX (L)	4	0,03%	MIV	OST (N)	3	0,02%
SBR	SRC (N)	4	0,03%	MIV	BRT (L)	3	0,02%
SIX	BRT (L)	4	0,03%	MIV	ADX (L)	3	0,02%
SIX	FSS (N)	4	0,03%	MLR	DVR (N)	3	0,02%
SIX	TUP (N)	4	0,03%	MLR	MLX (L)	3	0,02%
SIX	ADI (N)	4	0,03%	MLR	ADI (N)	3	0,02%
SLB	ADR (N)	4	0,03%	MLR	SCD (R)	3	0,02%
SLB	WSS (N)	4	0,03%	MLR	BRR (D)	3	0,02%
SOX	CEI (N)	4	0,03%	MLX	DSR (D)	3	0,02%
SPJ	BRT (L)	4	0,03%	MLX	SYN (L)	3	0,02%
SPJ	DSR (D)	4	0,03%	MLX	SIX (L)	3	0,02%
SPL	DUP (N)	4	0,03%	NGR	DUP (N)	3	0,02%
SXI	TUP (N)	4	0,03%	NOT	SXI (L)	3	0,02%
SXI	BIX (D)	4	0,03%	POF	SYN (R)	3	0,02%
SXI	SCD (R)	4	0,03%	OST	SOI (R)	3	0,02%
SXN	DUP (N)	4	0,03%	OST	OST (N)	3	0,02%
SYN	BIX (D)	4	0,03%	OST	WSS (N)	3	0,02%
SYN	MIV (N)	4	0,03%	OST	GSP (N)	3	0,02%
SYN	WDS (N)	4	0,03%	PRL	MLI (N)	3	0,02%
TUP	ADI (N)	4	0,03%	PRL	SOI (R)	3	0,02%
ADI	DSR (D)	3	0,02%	PRL	OST (N)	3	0,02%
ADI	BRT (L)	3	0,02%	RSS	ADI (N)	3	0,02%
ADI	BRR (D)	3	0,02%	RSS	SBI (N)	3	0,02%
ADI	CEI (N)	3	0,02%	RSS	CRJ (N)	3	0,02%

RSS	ESP (L)	3	0,02%	DSR	GAN (N)	2	0,01%
SAN	IRC (N)	3	0,02%	DUP	CGI (N)	2	0,01%
SAZ	BRR (D)	3	0,02%	DUP	MRC (N)	2	0,01%
SAZ	MLR (N)	3	0,02%	DUP	BRV (L)	2	0,01%
SBI	FLT (N)	3	0,02%	DVR	OST (N)	2	0,01%
SBI	WSS (N)	3	0,02%	DVR	ADR (N)	2	0,01%
SBI	BRT (L)	3	0,02%	ERC	DUP (N)	2	0,01%
SBI	GSP (N)	3	0,02%	EXR	SOX (R)	2	0,01%
SBR	DSR (D)	3	0,02%	ESP	GSP (N)	2	0,01%
SCD	SYN (R)	3	0,02%	ESP	BRT (L)	2	0,01%
SBR	ADX (L)	3	0,02%	ESP	ADI (N)	2	0,01%
SFZ	SIX (L)	3	0,02%	ESP	IRC (N)	2	0,01%
SIX	SRC (N)	3	0,02%	FSS	MLR (N)	2	0,01%
SLB	MID (N)	3	0,02%	FSS	BRT (L)	2	0,01%
SLB	SRC (N)	3	0,02%	FSS	BRR (D)	2	0,01%
SOI	SYN (R)	3	0,02%	GAN	MLX (L)	2	0,01%
SLB	ADI (N)	3	0,02%	GAN	SOX (R)	2	0,01%
SOX	OST (N)	3	0,02%	GAN	SBI (N)	2	0,01%
SOX	SOX (R)	3	0,02%	GAN	MID (N)	2	0,01%
SOX	BRR (D)	3	0,02%	GSP	MID (N)	2	0,01%
SPJ	SYN (R)	3	0,02%	GSP	GSP (N)	2	0,01%
SPJ	SIX (L)	3	0,02%	GSP	ESP (L)	2	0,01%
SPJ	MID (N)	3	0,02%	GSP	WSS (N)	2	0,01%
SPL	SIX (L)	3	0,02%	GSP	PRL (L)	2	0,01%
SWA	DUP (N)	3	0,02%	GSP	OST (N)	2	0,01%
SXI	FSS (N)	3	0,02%	GSP	SRC (N)	2	0,01%
SYN	SOR (L)	3	0,02%	IRC	GSP (N)	2	0,01%
SYN	DSR (R)	3	0,02%	IRC	SIX (L)	2	0,01%
TUP	GSP (N)	3	0,02%	IRC	PRL (L)	2	0,01%
ABR	CGR (N)	2	0,01%	MID	RSS (N)	2	0,01%
ADI	SOR (L)	2	0,01%	MID	SPJ (D)	2	0,01%
ADI	SOX (R)	2	0,01%	MID	SQR (N)	2	0,01%
ADR	DSR (D)	2	0,01%	MID	SBI (N)	2	0,01%
ADR	GSP (N)	2	0,01%	MLI	SYN (R)	2	0,01%
ADR	SXI (L)	2	0,01%	MIV	GSP (N)	2	0,01%
ADR	MLX (L)	2	0,01%	MIV	CEI (N)	2	0,01%
ADX	TUP (N)	2	0,01%	MIV	SYN (L)	2	0,01%
ADX	BRR (D)	2	0,01%	MIV	SOX (R)	2	0,01%
ADX	BRT (L)	2	0,01%	MLI	PRL (L)	2	0,01%
ADX	SRC (N)	2	0,01%	MLR	TUP (N)	2	0,01%
ADX	CRJ (N)	2	0,01%	MLR	SFZ (L)	2	0,01%
ADX	IRC (N)	2	0,01%	MLX	MID (N)	2	0,01%
ADX	ADI (N)	2	0,01%	MLX	CRJ (N)	2	0,01%
BIX	ADI (N)	2	0,01%	MLX	MIV (N)	2	0,01%
BIX	IRC (N)	2	0,01%	MLX	SRC (N)	2	0,01%
BIX	BRV (L)	2	0,01%	MLX	SBI (N)	2	0,01%
BIX	MIV (N)	2	0,01%	MLX	SOX (R)	2	0,01%
BIX	RSS (N)	2	0,01%	MRC	SYN (R)	2	0,01%
BIX	SBI (N)	2	0,01%	NOT	DUP (N)	2	0,01%
BRR	IRC (N)	2	0,01%	NOT	SAZ (L)	2	0,01%
BRR	SCD (R)	2	0,01%	NOT	GSP (N)	2	0,01%
BRR	GSP (N)	2	0,01%	ORB	DSR (D)	2	0,01%
BRR	DVR (N)	2	0,01%	OST	IRC (N)	2	0,01%
BRR	SOR (R)	2	0,01%	OST	SPJ (R)	2	0,01%
BRR	MLR (N)	2	0,01%	OST	MID (N)	2	0,01%
BRT	DSR (D)	2	0,01%	OST	SOX (R)	2	0,01%
BRT	BRT (L)	2	0,01%	POF	SBI (N)	2	0,01%
BRT	BRR (D)	2	0,01%	POF	SIX (L)	2	0,01%
BRT	ADR (N)	2	0,01%	PRL	SOX (R)	2	0,01%
BRT	MID (N)	2	0,01%	PRL	RSS (N)	2	0,01%
CEI	CRJ (N)	2	0,01%	PRL	GAN (N)	2	0,01%
CEI	OST (N)	2	0,01%	PRL	SLB (L)	2	0,01%
CEI	SXI (L)	2	0,01%	RSS	SYN (R)	2	0,01%
CEI	ESP (L)	2	0,01%	PRP	SAN (L)	2	0,01%
CEV	IRC (N)	2	0,01%	PRP	MLR (N)	2	0,01%
CGI	DUP (N)	2	0,01%	RSS	GSP (N)	2	0,01%
CGR	FSS (N)	2	0,01%	RSS	BRT (L)	2	0,01%
CGR	BRR (L)	2	0,01%	RSS	SOX (R)	2	0,01%
CRJ	DSR (D)	2	0,01%	RSS	SAN (L)	2	0,01%
CRJ	CEI (N)	2	0,01%	RSS	ADX (L)	2	0,01%
CRJ	SIX (L)	2	0,01%	SAN	SCD (R)	2	0,01%
CRJ	BRR (D)	2	0,01%	SAN	ESP (L)	2	0,01%
DSR	SPJ (D)	2	0,01%	SAZ	GSP (N)	2	0,01%

SAZ	ADR (N)	2	0,01%	ADX	BIX (D)	1	0,01%
SAZ	MIV (N)	2	0,01%	ADX	SBI (N)	1	0,01%
SAZ	FSS (N)	2	0,01%	ADX	DSR (L)	1	0,01%
SBI	MIV (N)	2	0,01%	ADX	MIV (N)	1	0,01%
SBI	MLX (L)	2	0,01%	ADX	SPJ (R)	1	0,01%
SBI	SXI (L)	2	0,01%	ADX	MLR (N)	1	0,01%
SBR	WSS (N)	2	0,01%	AND	SLB (L)	1	0,01%
SBR	SOX (R)	2	0,01%	BIX	SOR (R)	1	0,01%
SBR	ADR (N)	2	0,01%	BIX	MLX (L)	1	0,01%
SBR	BRR (L)	2	0,01%	BIX	BRT (L)	1	0,01%
SBR	ABR (N)	2	0,01%	BIX	SOR (L)	1	0,01%
SBR	DVR (N)	2	0,01%	BIX	OST (N)	1	0,01%
SBR	GSP (N)	2	0,01%	BIX	SRC (N)	1	0,01%
SFZ	ADX (L)	2	0,01%	BIX	BRR (D)	1	0,01%
SIX	SLB (L)	2	0,01%	BIX	SXI (L)	1	0,01%
SIX	SXI (L)	2	0,01%	BIX	YXI (N)	1	0,01%
SIX	BIX (D)	2	0,01%	BIX	MLR (N)	1	0,01%
SIX	MLR (N)	2	0,01%	BRR	SPJ (R)	1	0,01%
SIX	SPJ (R)	2	0,01%	BRR	BRV (L)	1	0,01%
SIX	BRR (D)	2	0,01%	BRR	GAN (N)	1	0,01%
SIX	IRC (N)	2	0,01%	BRR	DSR (R)	1	0,01%
SIX	SBI (N)	2	0,01%	BRR	SFZ (L)	1	0,01%
SLB	BRR (D)	2	0,01%	BRR	SOI (R)	1	0,01%
SLB	MIV (N)	2	0,01%	BRT	FSS (N)	1	0,01%
SLB	CRJ (N)	2	0,01%	BRT	MLX (L)	1	0,01%
SLB	SOR (R)	2	0,01%	BRT	CRJ (N)	1	0,01%
SOI	IRC (N)	2	0,01%	BRT	SRC (N)	1	0,01%
SOI	SXI (L)	2	0,01%	BRV	CEI (N)	1	0,01%
SOR	SLB (L)	2	0,01%	BRV	ESP (L)	1	0,01%
SOR	SYN (L)	2	0,01%	CEI	SOX (R)	1	0,01%
SOX	WSS (N)	2	0,01%	CEI	BRV (L)	1	0,01%
SOX	MID (N)	2	0,01%	CEI	ADR (N)	1	0,01%
SOX	GSP (N)	2	0,01%	CEI	MLR (N)	1	0,01%
SOX	ADR (N)	2	0,01%	CEI	SOI (R)	1	0,01%
SOX	ADI (N)	2	0,01%	CEI	SRC (N)	1	0,01%
SPJ	WSS (N)	2	0,01%	CEV	TUP (N)	1	0,01%
SPJ	GSP (N)	2	0,01%	CGI	NOT (N)	1	0,01%
SPJ	BRR (D)	2	0,01%	CGI	ADX (L)	1	0,01%
SPL	BRT (L)	2	0,01%	CGI	PRL (L)	1	0,01%
SPL	PRL (L)	2	0,01%	CGR	PRL (L)	1	0,01%
SXI	GSP (N)	2	0,01%	CGR	SXI (L)	1	0,01%
SXI	MID (N)	2	0,01%	CGR	GSP (N)	1	0,01%
SXI	DSR (D)	2	0,01%	CGR	SYN (R)	1	0,01%
SXI	BRR (D)	2	0,01%	CGR	SYN (L)	1	0,01%
SXN	PRL (L)	2	0,01%	COS	MLX (L)	1	0,01%
SXN	ADI (N)	2	0,01%	CRJ	SBI (N)	1	0,01%
SXO	SYN (R)	2	0,01%	CRJ	SPJ (D)	1	0,01%
SYN	GAN (N)	2	0,01%	CRJ	SOI (R)	1	0,01%
TUP	MIV (N)	2	0,01%	CRJ	SYN (R)	1	0,01%
TUP	BIX (L)	2	0,01%	CRJ	SCD (R)	1	0,01%
TUP	BRV (L)	2	0,01%	CRJ	GSP (N)	1	0,01%
YXI	SIX (L)	2	0,01%	CRJ	ADX (L)	1	0,01%
ABR	SLB (L)	1	0,01%	CRJ	MLX (L)	1	0,01%
ABR	FSS (N)	1	0,01%	DMD	SIX (L)	1	0,01%
ADI	DSR (L)	1	0,01%	DMD	ESP (L)	1	0,01%
ADI	CGI (N)	1	0,01%	DMD	SXI (L)	1	0,01%
ADI	CRJ (N)	1	0,01%	DMD	IRC (N)	1	0,01%
ADI	FSS (N)	1	0,01%	DMD	SYN (R)	1	0,01%
ADI	SFZ (L)	1	0,01%	DMD	ADX (L)	1	0,01%
ADI	SLB (L)	1	0,01%	DSR	MLX (L)	1	0,01%
ADI	BIX (L)	1	0,01%	DSR	SOI (R)	1	0,01%
ADI	SPJ (D)	1	0,01%	DSR	SQR (N)	1	0,01%
ADR	SBI (N)	1	0,01%	DSR	FSS (N)	1	0,01%
ADR	SOX (R)	1	0,01%	DSR	ESP (L)	1	0,01%
ADR	SPJ (D)	1	0,01%	DSR	SAZ (L)	1	0,01%
ADR	OST (N)	1	0,01%	DUP	CEV (N)	1	0,01%
ADR	CEI (N)	1	0,01%	DUP	YAN (N)	1	0,01%
ADR	SQR (N)	1	0,01%	DVR	ABR (N)	1	0,01%
ADR	IRC (N)	1	0,01%	DVR	MLR (N)	1	0,01%
ADR	SFZ (L)	1	0,01%	DVR	SOR (L)	1	0,01%
ADR	SBR (N)	1	0,01%	DVR	GAN (N)	1	0,01%
ADR	DVR (N)	1	0,01%	DVR	CEI (N)	1	0,01%
ADR	SCD (R)	1	0,01%	DVR	BRR (D)	1	0,01%

DVR	FSS (N)	1	0,01%	MLX	WSS (N)	1	0,01%
DVR	WSS (N)	1	0,01%	MLX	SOR (R)	1	0,01%
ERC	PRL (L)	1	0,01%	MRC	ADX (L)	1	0,01%
ERC	BRT (L)	1	0,01%	MRC	PRL (L)	1	0,01%
ERC	ADX (L)	1	0,01%	NGR	ADR (N)	1	0,01%
ESP	CRJ (N)	1	0,01%	NOT	SIX (L)	1	0,01%
ESP	ESP (L)	1	0,01%	OPT	TUP (N)	1	0,01%
ESP	SIX (L)	1	0,01%	NOT	SYN (L)	1	0,01%
ESP	SPJ (D)	1	0,01%	ORB	FSS (N)	1	0,01%
ESP	SRC (N)	1	0,01%	ORB	PRL (L)	1	0,01%
ESP	CEI (N)	1	0,01%	OST	MLR (N)	1	0,01%
ESP	GAN (N)	1	0,01%	OST	ADR (N)	1	0,01%
ESP	SYN (L)	1	0,01%	OST	BRT (L)	1	0,01%
FLT	SOR (L)	1	0,01%	OST	SRC (N)	1	0,01%
FLT	PRL (L)	1	0,01%	OST	SXI (L)	1	0,01%
FLT	DVR (N)	1	0,01%	OST	ADI (N)	1	0,01%
FSS	OST (N)	1	0,01%	OST	BRR (D)	1	0,01%
FSS	PRL (L)	1	0,01%	OST	SBI (N)	1	0,01%
FSS	ADI (N)	1	0,01%	POF	ADX (L)	1	0,01%
FSS	MID (N)	1	0,01%	POF	CRJ (N)	1	0,01%
FSS	SOI (R)	1	0,01%	POF	DSR (D)	1	0,01%
FSS	IRC (N)	1	0,01%	POF	MLX (L)	1	0,01%
FSS	GSP (N)	1	0,01%	POF	MID (N)	1	0,01%
FSS	WSS (N)	1	0,01%	POF	IRC (N)	1	0,01%
FSS	CRJ (N)	1	0,01%	POF	SXI (L)	1	0,01%
FSS	FSS (N)	1	0,01%	POF	CEI (N)	1	0,01%
FSS	SPJ (R)	1	0,01%	POF	BRR (D)	1	0,01%
GAN	WSS (N)	1	0,01%	PRL	BRR (R)	1	0,01%
GAN	ADX (L)	1	0,01%	PRL	ESP (L)	1	0,01%
GAN	IRC (N)	1	0,01%	PRL	BRV (L)	1	0,01%
GAN	BRR (D)	1	0,01%	PRL	CEV (N)	1	0,01%
GSP	SBI (N)	1	0,01%	PRL	DVR (N)	1	0,01%
GSP	CRJ (N)	1	0,01%	PRL	CGR (N)	1	0,01%
GSP	CEI (N)	1	0,01%	PRL	SQR (N)	1	0,01%
GSP	FSS (N)	1	0,01%	PRL	SPJ (R)	1	0,01%
GSP	DSR (D)	1	0,01%	RSS	SQR (N)	1	0,01%
GSP	BRR (D)	1	0,01%	RSS	WSS (N)	1	0,01%
GSP	BRT (L)	1	0,01%	RSS	SPJ (R)	1	0,01%
GSP	MLX (L)	1	0,01%	RSS	MLX (L)	1	0,01%
IRC	SOX (R)	1	0,01%	RSS	SAZ (L)	1	0,01%
IRC	BRR (D)	1	0,01%	RSS	RSS (N)	1	0,01%
IRC	BRT (L)	1	0,01%	RSS	SOR (R)	1	0,01%
IRC	MID (N)	1	0,01%	RSS	GAN (N)	1	0,01%
IRC	ADX (L)	1	0,01%	RSS	SBR (N)	1	0,01%
IRC	ADI (N)	1	0,01%	RSS	DVR (N)	1	0,01%
MID	MLI (N)	1	0,01%	RSS	OST (N)	1	0,01%
MID	SPJ (R)	1	0,01%	RSS	SCD (R)	1	0,01%
MID	ESP (L)	1	0,01%	SAN	ADI (N)	1	0,01%
MIV	WSS (N)	1	0,01%	SAN	SBI (N)	1	0,01%
MIV	FSS (N)	1	0,01%	SAN	MLX (L)	1	0,01%
MIV	SQR (N)	1	0,01%	SAZ	SOX (R)	1	0,01%
MIV	SBI (N)	1	0,01%	SAZ	ADX (L)	1	0,01%
MIV	MLX (L)	1	0,01%	SAZ	MLI (N)	1	0,01%
MIV	SPJ (D)	1	0,01%	SAZ	SXI (L)	1	0,01%
MIV	DSR (L)	1	0,01%	SAZ	SLB (L)	1	0,01%
MIV	ESP (L)	1	0,01%	SAZ	SCD (R)	1	0,01%
MIV	MID (N)	1	0,01%	SAZ	OST (N)	1	0,01%
MIV	CRJ (N)	1	0,01%	SAZ	DMD (R)	1	0,01%
MLI	MLX (L)	1	0,01%	SAZ	SOR (L)	1	0,01%
MLI	DSR (D)	1	0,01%	SAZ	CGI (N)	1	0,01%
MLI	SBI (N)	1	0,01%	SAZ	SOR (R)	1	0,01%
MLI	ADI (N)	1	0,01%	SBI	BRR (D)	1	0,01%
MLR	SYN (R)	1	0,01%	SBI	ESP (L)	1	0,01%
MLI	SXI (L)	1	0,01%	SBI	SQR (N)	1	0,01%
MLI	SYN (L)	1	0,01%	SBI	SPJ (D)	1	0,01%
MLR	COS (N)	1	0,01%	SBI	ADR (N)	1	0,01%
MLR	SIN (N)	1	0,01%	SBI	IRC (N)	1	0,01%
MLR	MID (N)	1	0,01%	SBR	MLX (L)	1	0,01%
MLR	GSP (N)	1	0,01%	SBR	PRL (L)	1	0,01%
MLR	SPJ (D)	1	0,01%	SBR	BRR (D)	1	0,01%
MLR	SBI (N)	1	0,01%	SBR	SBI (N)	1	0,01%
MLR	SOR (L)	1	0,01%	SBR	OST (N)	1	0,01%
MLX	SXI (L)	1	0,01%	SBR	CRJ (N)	1	0,01%

SBR	BIX (D)	1	0,01%
SBR	SQR (N)	1	0,01%
SBR	SBR (N)	1	0,01%
SIN	SCD (R)	1	0,01%
SFZ	SBI (N)	1	0,01%
SIX	MLR (N)	1	0,01%
SIX	SAZ (L)	1	0,01%
SIX	DVR (N)	1	0,01%
SIX	BRR (L)	1	0,01%
SIX	SQR (N)	1	0,01%
SLB	SOR (L)	1	0,01%
SLB	DSR (D)	1	0,01%
SLB	ADX (L)	1	0,01%
SLB	BRR (L)	1	0,01%
SLB	SBI (N)	1	0,01%
SOR	SYN (R)	1	0,01%
SOI	SYN (L)	1	0,01%
SOI	SAZ (L)	1	0,01%
SOI	POF (L)	1	0,01%
SOI	DUP (N)	1	0,01%
SOR	BIX (D)	1	0,01%
SOR	SIX (L)	1	0,01%
SOR	SOR (L)	1	0,01%
SOX	MLX (L)	1	0,01%
SOX	SRC (N)	1	0,01%
SOX	SOI (R)	1	0,01%
SOX	DSR (D)	1	0,01%
SOX	SBI (N)	1	0,01%
SOX	MLR (N)	1	0,01%
SPJ	GAN (N)	1	0,01%
SPJ	SOX (R)	1	0,01%
SPJ	SRC (N)	1	0,01%
SPJ	IRC (N)	1	0,01%
SPL	SYN (R)	1	0,01%
SPJ	SPJ (R)	1	0,01%
SPJ	FSS (N)	1	0,01%
SPJ	ADI (N)	1	0,01%
SQR	SYN (R)	1	0,01%
SPL	ADX (L)	1	0,01%
SPL	IRC (N)	1	0,01%
SPL	SOX (R)	1	0,01%
SPL	SYN (L)	1	0,01%
SQR	BRR (D)	1	0,01%
SRC	SYN (R)	1	0,01%
SQR	DUP (N)	1	0,01%
SQR	SOR (L)	1	0,01%
SWA	BRT (L)	1	0,01%
SXI	CRJ (N)	1	0,01%
SXI	POF (L)	1	0,01%
SXI	ESP (L)	1	0,01%
SXI	SOX (R)	1	0,01%
SXI	SFZ (L)	1	0,01%
SXI	CEI (N)	1	0,01%
SXI	BIX (R)	1	0,01%
SXN	OST (N)	1	0,01%
SXN	BRT (L)	1	0,01%
SXN	SYN (L)	1	0,01%
SXN	SOX (R)	1	0,01%
SXN	SRC (N)	1	0,01%
SXN	ADX (L)	1	0,01%
SYN	TUP (N)	1	0,01%
SYN	SBR (N)	1	0,01%
SYN	SAN (L)	1	0,01%
SYN	NGR (N)	1	0,01%
SYN	DVR (N)	1	0,01%
SYN	MLR (N)	1	0,01%
SYN	DSR (L)	1	0,01%
SYN	RSS (N)	1	0,01%
SYN	POF (L)	1	0,01%
TUP	PRP (L)	1	0,01%
TUP	BRR (D)	1	0,01%
TUP	SIX (L)	1	0,01%
TUP	RSS (N)	1	0,01%

TUP	BRR (L)	1	0,01%
TUP	DRC (N)	1	0,01%
TUP	SWA (L)	1	0,01%
YAN	DUP (N)	1	0,01%
YXI	IRC (N)	1	0,01%

Apêndice C

Dados para Análise estrutural - Ackerman.cod

Estrutura	Repetições	Porcentagem
N -> L-	10	20,8%
N -> R-L	8	16,7%
N -> L-L	7	14,6%
L -> L-	7	14,6%
L -> L-L	4	8,3%
L -> R-L	4	8,3%
R -> L-	2	4,2%
N -> R-R	2	4,2%
D -> L-L	2	4,2%
N -> L-R	1	2,1%
R -> R-	1	2,1%

Dados para Análise Estrutural - Bin.cod

Estrutura	Repetições	Porcentagem
N -> L-	14	27,5%
N -> L-L	8	15,7%
L -> L-L	7	13,7%
L -> R-L	5	9,8%
N -> R-R	4	7,8%
N -> R-	3	5,9%
N -> R-L	3	5,9%
D -> L-L	2	3,9%
R -> L-	2	3,9%
L -> L-	1	2,0%
L -> R-R	1	2,0%
D -> L-R	1	2,0%

Dados para Análise estrutural - Fat.cod

Estrutura	Repetições	Porcentagem
N -> R-L	8	22,2%
N -> L-	5	13,9%
L -> R-L	5	13,9%
N -> L-L	4	11,1%
L -> L-	4	11,1%
L -> L-L	4	11,1%
R -> L-	2	5,6%
L -> R-R	2	5,6%
N -> R-R	1	2,8%
D -> L-L	1	2,8%

Dados para Análise Estrutural - Matrix.cod

Estrutura	Repetições	Porcentagem
L -> L-	92	28,1%
N -> L-	64	19,6%
N -> L-L	41	12,5%
L -> R-	30	9,2%
N -> R-	28	8,6%
R -> L-	16	4,9%
N -> L-R	12	3,7%
N -> R-L	10	3,1%
L -> L-L	8	2,4%
D -> L-L	8	2,4%
N -> R-R	6	1,8%

D -> L-R	5	1,5%
L -> L-R	3	0,9%
L -> -E	2	0,6%
L -> R-R	1	0,3%
L -> -	1	0,3%

Dados para Análise estrutural - N_queens.cod

Estrutura	Repetições	Porcentagem
N -> L-L	108	20,8%
L -> L-	91	17,5%
N -> L-	86	16,5%
L -> R-	39	7,5%
N -> R-	31	6,0%
N -> R-R	28	5,4%
R -> L-	28	5,4%
L -> L-L	27	5,2%
N -> R-L	25	4,8%
N -> L-R	15	2,9%
D -> L-L	15	2,9%
L -> R-L	11	2,1%
R -> L-L	5	1,0%
L -> L-R	3	0,6%
L -> R-R	3	0,6%
R -> L-R	2	0,4%
D -> R-R	1	0,2%
R -> R-R	1	0,2%

Dados para Análise estrutural - Dinâmica.cod

Estrutura	Repetições	Porcentagem
L -> L-	548	22,8%
N -> L-	436	18,1%
N -> L-L	381	15,9%
L -> R-	194	8,1%
N -> R-	167	6,9%
R -> L-	132	5,5%
N -> R-L	93	3,9%
N -> L-R	92	3,8%
L -> L-L	89	3,7%
N -> R-R	82	3,4%
D -> L-L	66	2,7%
L -> R-L	26	1,1%
L -> R-R	24	1,0%
L -> L-R	24	1,0%
R -> R-	17	0,7%
R -> R-L	15	0,6%
R -> L-L	10	0,4%
R -> L-R	4	0,2%
D -> L-R	2	0,1%
R -> R-R	1	0,0%

Dados para Análise estrutural - Simple.cod

Estrutura	Repetições	Porcentagem
N -> L-	2044	18,9%
L -> L-	1893	17,5%
N -> L-L	1618	15,0%
N -> R-	1202	11,1%
L -> R-	852	7,9%
R -> L-	493	4,6%
L -> L-L	428	4,0%
N -> L-R	383	3,5%
N -> R-R	355	3,3%
N -> R-L	342	3,2%
D -> L-L	292	2,7%

D -> L-R	288	2,7%
L -> L-R	153	1,4%
R -> R-	100	0,9%
R -> R-L	96	0,9%
L -> R-L	88	0,8%
R -> L-L	73	0,7%
L -> R-R	71	0,7%
R -> L-R	27	0,2%
R -> R-R	12	0,1%
D -> R-R	2	0,0%

Apêndice D

Tipo de Fichas

- **Tipo Executável (EX)**

Ficha: Contém o tipo de ficha (EX)

Dado0: Contém o operando esquerdo de uma operação.

Tipo0: Contém o tipo de dado que está no campo Dado0.

Dado1: Contém o operando direito de uma operação.

Tipo1: Contém o tipo de dado que está no campo Dado1.

Cód.oper.: Contém o código de operação. Corresponde a uma instrução do programa.

Nó_dest0: Contém o endereço do destino do primeiro resultado gerado pela execução da ficha executável.

F_emp0: Contém a função de emparelhamento do primeiro resultado gerado pela execução da ficha executável.

Nó_dest1: Contém o endereço de destino do segundo resultado gerado pela execução da ficha executável.

F_emp1: Contém a função de emparelhamento do segundo resultado gerado pela execução da ficha executável.

Na: Contém a identificação da instanciação a que a ficha pertence.

Ix: Identifica o índice a que a ficha pertence.

- **Tipo Par (PA)**

Ficha: Contém o tipo de ficha (PA)

Dado0: Contém o operando esquerdo de uma operação.

Tipo0: Contém o tipo de dado que está no campo Dado0.

Dado1: Contém o operando direito de uma operação.

Tipo1: Contém o tipo de dado que está no campo Dado1.

Nó: Contém o endereço para o qual os dados da ficha se destinam.

Na: Contém a identificação da instanciação a que a ficha pertence.

Ix: Identifica o índice a que a ficha pertence.

- **Tipo Programa (PR)**

Ficha: Contém o tipo de ficha (PR)

Cód.oper.: Contém o código de operação. Corresponde a uma instrução do programa.

Nó: Contém o endereço onde a ficha PR será armazenada.

Nó_dest0: Contém o endereço do destino do primeiro resultado gerado pela execução da ficha executável.

F_emp0: Contém a função de emparelhamento do primeiro resultado gerado pela execução da ficha executável.

Nó_dest1: Contém o endereço de destino do segundo resultado gerado pela execução da ficha executável.

F_emp1: Contém a função de emparelhamento do segundo resultado gerado pela execução da ficha executável.

Na: Contém a identificação da instanciação a que a ficha pertence.

Ix: Identifica o índice a que a ficha pertence.

- **Tipo Saída (SA)**

Ficha: Contém o tipo de ficha (SA)

Dado0: Contém um dado que será enviado à saída como resultado de um programa.

Tipo0: Contém o tipo de dado que está no campo Dado0.

Dado: Contém o valor de stream. Esse valor identifica o dado de saída. Se o programa possuir mais de uma saída, então o valor de stream identificará cada saída correspondente a quais dados obtidos

Na: Contém a identificação da instanciação a que a ficha pertence.

Ix: Identifica o índice a que a ficha pertence.

- **Outras:** As fichas dos tipos Cor-Destino (CD), Destino (DE), Inteiro (IN), Real (RE), Booleana (BO), Nome de Ativação (NA), Ponteiro (QQ) e Fim de stream (ZZ) são semelhantes, apresentando diferenças apenas nos campos de Ficha de Dado.

Apêndice E

Execução dinâmica de Gauss.sis:

MONITORING OF DATA FLOW PROGRAM EXECUTION

START OF EXECUTION
END OF EXECUTION

BREAKDOWN OF STRUCTURE STORE STATISTICS BY BLOCKS:
(SIM assumes that the SS consists of 4 interleaved blocks)

PRESENCE ACTIONS:

	READ	EXTRACT	INC	DEC	DEC/ACK	SPECIAL	ERROR
BLOCK 0 :1132	293	25	50	0	0	0	0
BLOCK 1 :1174	292	18	44	0	0	0	0
BLOCK 2 :1268	293	59	88	0	0	0	0
BLOCK 3 :1158	293	32	59	0	0	0	0
TOTAL : 4732	1171	134	241	0	0	0	0

ABSENCE ACTIONS:

	WRITE	DEFER	ABORT	GENERATE
BLOCK 0 : 293	0	0	0	0
BLOCK 1 : 292	0	0	0	0
BLOCK 2 : 293	0	0	0	0
BLOCK 3 : 293	0	0	0	0
TOTAL : 1171	0	0	0	0

OTHER STATISTICS:

	TOT. ACCESSES	MAX. DAQ. LENGTH
BLOCK 0 :	1793	0
BLOCK 1 :	1820	0
BLOCK 2 :	2001	0
BLOCK 3 :	1835	0
TOTAL :	7449	0

HIGHEST REF. COUNT PRODUCED BY INCREMENT WAS 3
HIGHEST INITIAL REF. COUNT WAS 1

ALLOCATION PROFILE:

SIZE	FREQUENCY
3	1
5	1
7	1
9	81
11	1
13	1
15	1
17	19
65	1

TOTAL SPACE ALLOCATED WAS 1171

MAX. B-STORE OCCUPANCY WAS 954 VALUES
WITH B.TOP AT LOCATION 954
B-STORE UTILISATION AT PEAK OCCUPANCY WAS 100.00%
B-ALLOCATOR CALLED 107 TIMES

INSTRUCTION BREAKDOWN:

ADI	488	1.697%	(1 IN 58.9)
AND	8	0.028%	(1 IN 3595.5)
CEI	325	1.130%	(1 IN 88.5)
CGI	260	0.904%	(1 IN 110.6)
CLI	64	0.223%	(1 IN 449.4)
DUP	4515	15.697%	(1 IN 6.4)

TUP	704	2.448%	(1 IN 40.9)
GAN	160	0.556%	(1 IN 179.8)
NOT	24	0.083%	(1 IN 1198.5)
ORB	8	0.028%	(1 IN 3595.5)
SBI	452	1.571%	(1 IN 63.6)
YAN	2	0.007%	(1 IN 14382.0)
CRJ	108	0.375%	(1 IN 266.3)
MIV	816	2.837%	(1 IN 35.3)
MID	1532	5.326%	(1 IN 18.8)
ABR	64	0.223%	(1 IN 449.4)
CGR	36	0.125%	(1 IN 799.0)
MLI	18	0.063%	(1 IN 1598.0)
OST	108	0.375%	(1 IN 266.3)
ADR	232	0.807%	(1 IN 124.0)
DVR	128	0.445%	(1 IN 224.7)
MLR	616	2.142%	(1 IN 46.7)
SBR	456	1.585%	(1 IN 63.1)
SRC	108	0.375%	(1 IN 266.3)
GSP	108	0.375%	(1 IN 266.3)
RSS	1460	5.076%	(1 IN 19.7)
WSS	1064	3.699%	(1 IN 27.0)
FSS	18	0.063%	(1 IN 1598.0)
IRC	170	0.591%	(1 IN 169.2)
DRC	62	0.216%	(1 IN 463.9)
ADX	276	0.960%	(1 IN 104.2)
BRR	3806	13.232%	(1 IN 7.6)
BIX	656	2.281%	(1 IN 43.8)
SPJ	19	0.066%	(1 IN 1513.9)
SAZ	1432	4.978%	(1 IN 20.1)
SIX	1401	4.871%	(1 IN 20.5)
SXI	112	0.389%	(1 IN 256.8)
SLB	458	1.592%	(1 IN 62.8)
SOI	146	0.508%	(1 IN 197.0)
SWA	1	0.003%	(1 IN 28764.0)
SYN	816	2.837%	(1 IN 35.3)
PRL	1118	3.887%	(1 IN 25.7)
POF	138	0.480%	(1 IN 208.4)
MLX	288	1.001%	(1 IN 99.9)
BRT	216	0.751%	(1 IN 133.2)
BRV	57	0.198%	(1 IN 504.6)
DSR	1532	5.326%	(1 IN 18.8)
DMD	1	0.003%	(1 IN 28764.0)
OPT	66	0.229%	(1 IN 435.8)
SOX	1236	4.297%	(1 IN 23.3)
SXO	18	0.063%	(1 IN 1598.0)
SXN	17	0.059%	(1 IN 1692.0)
SPL	16	0.056%	(1 IN 1797.8)
SFZ	162	0.563%	(1 IN 177.6)
ESP	18	0.063%	(1 IN 1598.0)

BREAKDOWN OF INSTRUCTION TYPES:

(Classification is by type of the left input, as specified in the BPM)

Real	1532	5.326%	(1 IN 18.8)
Boolean	40	0.139%	(1 IN 719.1)
Integer	5208	18.106%	(1 IN 5.5)
J	19	0.066%	(1 IN 1513.9)
Act. name	163	0.567%	(1 IN 176.5)
Pointer	4200	14.602%	(1 IN 6.8)
Any	17602	61.195%	(1 IN 1.6)

Breakdown of instructions executed by segment:

Segment	Count
1	28764

BREAKDOWN OF MS ACTIONS:

BYPASS	EXTRACT	PRESER	INC	DEC	WAIT	DEFER	ABORT	GENER
--------	---------	--------	-----	-----	------	-------	-------	-------

9108	18528	0	0	1128	18528	0	0	0
------	-------	---	---	------	-------	---	---	---

OTHER ASSORTED STATISTICS:

FINAL A.N. COUNT WAS 160

HIGHEST COLOUR IN CONTEXTS WAS AN=0, IL=0

THERE WERE 1532 FLOATING POINT ARITHMETIC OPERATIONS (18.78 MIPS/MEGAFLOP)
THERE WERE 6780 "USEFUL" ARITHMETIC OPERATIONS (1 IN 4.2).
(This may include some integer operations used for housekeeping)

OVERALL SUMMARY STATISTICS:

Total instructions executed (S1)	28764
Critical path length (Sinf)	1018
Average parallelism (Pi)	28.3
Peak parallelism	481
Number of result tokens	66
Mips / Megaflops ratio	18.78
Proportion of bypass actions (Pby)	0.317
Total token queue traffic	47292
Maximum token queue length	720
Maximum matching store occupancy	5095
 Total structure store accesses	 7449
Maximum structure store occupancy	954
Total number of allocations	107

There were 1265 uses of proliferate orders, generating 10137 tokens
There were 704 uses of TUPlicate, generating 4939 tokens

Execução dinâmica de Quicksor.sis:

MONITORING OF DATA FLOW PROGRAM EXECUTION

START OF EXECUTION
END OF EXECUTION

BREAKDOWN OF STRUCTURE STORE STATISTICS BY BLOCKS:
(SIM assumes that the SS consists of 4 interleaved blocks)

PRESENCE ACTIONS:

	READ	EXTRACT	INC	DEC	DEC/ACK	SPECIAL	ERROR
BLOCK 0 : 228	160	22	53	0	0	0	0
BLOCK 1 : 195	161	55	123	0	0	0	0
BLOCK 2 : 254	158	19	49	0	0	0	0
BLOCK 3 : 189	155	50	112	0	0	0	0
TOTAL : 866	634	146	337	0	0	0	0

ABSENCE ACTIONS:

	WRITE	DEFER	ABORT	GENERATE
BLOCK 0 : 160	0	0	0	0
BLOCK 1 : 161	0	0	0	0
BLOCK 2 : 158	0	0	0	0
BLOCK 3 : 155	0	0	0	0
TOTAL : 634	0	0	0	0

OTHER STATISTICS:

	TOT. ACCESSES	MAX. DAQ. LENGTH
BLOCK 0 :	623	0
BLOCK 1 :	695	0
BLOCK 2 :	638	0
BLOCK 3 :	661	0
TOTAL :	2617	0

HIGHEST REF. COUNT PRODUCED BY INCREMENT WAS 2
HIGHEST INITIAL REF. COUNT WAS 1

ALLOCATION PROFILE:

SIZE	FREQUENCY
2	161
3	6
4	2
5	2

6	2
7	2
8	2
11	2
13	2
16	2
17	2
19	2
20	2
21	2

TOTAL SPACE ALLOCATED WAS 634

MAX. B-STORE OCCUPANCY WAS 198 VALUES
 WITH B.TOP AT LOCATION 202
 B-STORE UTILISATION AT PEAK OCCUPANCY WAS 98.02%
 B-ALLOCATOR CALLED 191 TIMES

INSTRUCTION BREAKDOWN:

ADI	498	1.847%	(1 IN 54.1)
CEI	1049	3.891%	(1 IN 25.7)
CGI	313	1.161%	(1 IN 86.1)
DUP	5969	22.143%	(1 IN 4.5)
TUP	1048	3.888%	(1 IN 25.7)
GAN	172	0.638%	(1 IN 156.7)
SBI	515	1.910%	(1 IN 52.3)
CRJ	484	1.795%	(1 IN 55.7)
MIV	423	1.569%	(1 IN 63.7)
OST	192	0.712%	(1 IN 140.4)
SRC	484	1.795%	(1 IN 55.7)
GSP	484	1.795%	(1 IN 55.7)
RSS	423	1.569%	(1 IN 63.7)
WSS	443	1.643%	(1 IN 60.9)
FSS	202	0.749%	(1 IN 133.5)
IRC	157	0.582%	(1 IN 171.7)
DRC	641	2.378%	(1 IN 42.1)
ADX	694	2.574%	(1 IN 38.8)
BRR	2693	9.990%	(1 IN 10.0)
SPJ	560	2.077%	(1 IN 48.1)
SAN	62	0.230%	(1 IN 434.8)
SAZ	564	2.092%	(1 IN 47.8)
SIX	1140	4.229%	(1 IN 23.6)
SXI	327	1.213%	(1 IN 82.4)
SLB	846	3.138%	(1 IN 31.9)
SCD	62	0.230%	(1 IN 434.8)
SWA	1	0.004%	(1 IN 26957.0)
SYN	2955	10.962%	(1 IN 9.1)
PRL	796	2.953%	(1 IN 33.9)
POF	15	0.056%	(1 IN 1797.1)
PRP	62	0.230%	(1 IN 434.8)
BRT	384	1.424%	(1 IN 70.2)
BRV	45	0.167%	(1 IN 599.0)
DRM	31	0.115%	(1 IN 869.6)
OPT	21	0.078%	(1 IN 1283.7)
SOX	655	2.430%	(1 IN 41.2)
SXN	891	3.305%	(1 IN 30.3)
SFZ	141	0.523%	(1 IN 191.2)
ESP	515	1.910%	(1 IN 52.3)

BREAKDOWN OF INSTRUCTION TYPES:
 (Classification is by type of the left input, as specified in the BPM)

Integer	4004	14.853%	(1 IN 6.7)
J	560	2.077%	(1 IN 48.1)
Act. name	204	0.757%	(1 IN 132.1)
Pointer	2067	7.668%	(1 IN 13.0)
Any	20122	74.645%	(1 IN 1.3)

Breakdown of instructions executed by segment:

Segment	Count
0	62
1	26895

BREAKDOWN OF MS ACTIONS:

BYPASS EXTRACT PRESER INC DEC WAIT DEFER ABORT GENER

15707	10787	0	0	463	10787	0	0	0
-------	-------	---	---	-----	-------	---	---	---

OTHER ASSORTED STATISTICS:

FINAL A.N. COUNT WAS 172

HIGHEST COLOUR IN CONTEXTS WAS AN=168, IL=0

THERE WERE 0 FLOATING POINT ARITHMETIC OPERATIONS.
THERE WERE 4004 "USEFUL" ARITHMETIC OPERATIONS (1 IN 6.7).
(This may include some integer operations used for housekeeping)

OVERALL SUMMARY STATISTICS:

Total instructions executed (S1)	26957
Critical path length (Sinf)	1590
Average parallelism (Pl)	17.0
Peak parallelism	220
Number of result tokens	21
Proportion of bypass actions (Phy)	0.583
Total token queue traffic	37744
Maximum token queue length	342
Maximum matching store occupancy	374
Total structure store accesses	2617
Maximum structure store occupancy	198
Total number of allocations	191

There were 720 uses of proliferate orders, generating 2663 tokens
There were 1048 uses of TUPlicate, generating 4003 tokens

Execução dinâmica de mms.sis:

MONITORING OF DATA FLOW PROGRAM EXECUTION

START OF EXECUTION

END OF EXECUTION

BREAKDOWN OF STRUCTURE STORE STATISTICS BY BLOCKS:
(SIM assumes that the SS consists of 4 interleaved blocks)

PRESENCE ACTIONS:

	READ	EXTRACT	INC	DEC	DEC/ACK	SPECIAL	ERROR
BLOCK 0 : 2594	66	0	17	0	0	0	0
BLOCK 1 : 2593	65	0	16	0	0	0	0
BLOCK 2 : 2593	64	0	4	0	0	0	0
BLOCK 3 : 2592	64	0	4	0	0	0	0
TOTAL : 0372	259	0	41	0	0	0	0

ABSENCE ACTIONS:

	WRITE	DEFER	ABORT	GENERATE
BLOCK 0 : 66	0	0	0	0
BLOCK 1 : 65	0	0	0	0
BLOCK 2 : 64	0	0	0	0
BLOCK 3 : 64	0	0	0	0
TOTAL : 259	0	0	0	0

OTHER STATISTICS:

	TOT. ACCESSES	MAX. DAQ. LENGTH
BLOCK 0 :	2743	0
BLOCK 1 :	2739	0
BLOCK 2 :	2725	0
BLOCK 3 :	2724	0

TOTAL : 10931 0

HIGHEST REF. COUNT PRODUCED BY INCREMENT WAS 0
HIGHEST INITIAL REF. COUNT WAS 1

ALLOCATION PROFILE:

SIZE FREQUENCY

13 14

25 3

TOTAL SPACE ALLOCATED WAS 257

MAX. B-STORE OCCUPANCY WAS 257 VALUES
WITH B.TOP AT LOCATION 257

B-STORE UTILISATION AT PEAK OCCUPANCY WAS 100.00%

B-ALLOCATOR CALLED 17 TIMES

INSTRUCTION BREAKDOWN:

ADI	161	0.618% (1 IN 161.8)
ADR	1728	6.632% (1 IN 15.1)
ADX	53	0.203% (1 IN 491.6)
BIX	1728	6.632% (1 IN 15.1)
BRR	161	0.618% (1 IN 161.8)
BRT	34	0.130% (1 IN 766.3)
BRV	144	0.553% (1 IN 180.9)
CEI	161	0.618% (1 IN 161.8)
CRJ	17	0.065% (1 IN 1532.6)
DRC	5	0.019% (1 IN 5210.8)
DSR	3457	13.269% (1 IN 7.5)
DUP	795	3.051% (1 IN 32.8)
ERC	2	0.008% (1 IN 13027.0)
GAN	156	0.599% (1 IN 167.0)
GSP	17	0.065% (1 IN 1532.6)
MID	3457	13.269% (1 IN 7.5)
MIV	3457	13.269% (1 IN 7.5)
MLI	3	0.012% (1 IN 8684.7)
MLR	1728	6.632% (1 IN 15.1)
MLX	72	0.276% (1 IN 361.9)
MRC	2	0.008% (1 IN 13027.0)
OPT	1	0.004% (1 IN 26054.0)
OST	17	0.065% (1 IN 1532.6)
POF	157	0.603% (1 IN 165.9)
PRL	1126	4.322% (1 IN 23.1)
RSS	3457	13.269% (1 IN 7.5)
SAZ	1248	4.790% (1 IN 20.9)
SBI	305	1.171% (1 IN 85.4)
SFZ	156	0.599% (1 IN 167.0)
SIX	418	1.604% (1 IN 62.3)
SLB	168	0.645% (1 IN 155.1)
SOI	157	0.603% (1 IN 165.9)
SOX	257	0.986% (1 IN 101.4)
SRC	17	0.065% (1 IN 1532.6)
SWA	1	0.004% (1 IN 26054.0)
SXI	144	0.553% (1 IN 180.9)
SYN	371	1.424% (1 IN 70.2)
TUP	474	1.819% (1 IN 55.0)
WDS	2	0.008% (1 IN 13027.0)
WSS	240	0.921% (1 IN 108.6)

BREAKDOWN OF INSTRUCTION TYPES:

(Classification is by type of the left input, as specified in the BPM)

Real	3456	13.265% (1 IN 7.5)
Integer	7909	30.356% (1 IN 3.3)
Act. name	157	0.603% (1 IN 165.9)
Pointer	7175	27.539% (1 IN 3.6)
Ref. count	2	0.008% (1 IN 13027.0)
Any	7355	28.230% (1 IN 3.5)

Breakdown of instructions executed by segment:

Segment Count

1 26054

BREAKDOWN OF MS ACTIONS:

BYPASS EXTRACT PRESER INC DEC WAIT DEFER ABORT GENER

3571	22243	0	0	240	22243	0	0	0
------	-------	---	---	-----	-------	---	---	---

OTHER ASSORTED STATISTICS:

FINAL A.N. COUNT WAS 156

HIGHEST COLOUR IN CONTEXTS WAS AN=0, IL=0

THERE WERE 3456 FLOATING POINT ARITHMETIC OPERATIONS (7.54 MIPS/MEGAFLOP)

THERE WERE 11365 "USEFUL" ARITHMETIC OPERATIONS (1 IN 2.3).

(This may include some integer operations used for housekeeping)

OVERALL SUMMARY STATISTICS:

Total instructions executed (S1)	26054
Critical path length (Sinf)	92
Average parallelism (Pi)	283.2
Peak parallelism	3604
Number of result tokens	1
Mips / Megaflops ratio	7.54
Proportion of bypass actions (Pby)	0.137
Total token queue traffic	48297
Maximum token queue length	10661
Maximum matching store occupancy	9908
Total structure store accesses	10931
Maximum structure store occupancy	257
Total number of allocations	17

There were 1283 uses of proliferate orders, generating 17232 tokens

There were 1283 uses of proliferate orders, generating 17232 tokens

There were 474 uses of TUPlicate, generating 2714 tokens

Bibliografia

- [Ack82] ACKERMAN, W.B. Dataflow languages. IEEE Computer, v.15, n.2, p.15-25, feb.1982.
- [Age82] AGERWALA, T. Dataflow systems. IEEE Computer, p.10-13, feb. 1982.
- [Alm+94] ALMASI, G.S.; GOTTLIEB, A. Highly parallel computing. 2.ed. California, The Benjamin / Cummings Publishing Company, Inc., 1994.
- [Amo+88] AMORIM, C.L.; BARBOSA, V.C.; FERNANDES, E.S.T. Uma introdução à computação paralela e distribuída. VI Escola de Computação, Campinas, UNICAMP, 1988.
- [Ang+94] ANG, B.S.; CHIOU, D. StarT the next generation: integrating global caches and dataflow architecture. Massachusetts, Laboratory for Computer Science, Institute of Technology, aug. 1994.
- [Ari90] ARIOLA, Z.M. Compilation of Id⁻: a subset of Id. Massachusetts, Laboratory for Computer Science, Institute of Technology, 1990.
- [Arv+86] ARVIND; CULLER, D.E. Dataflow architectures. Annual review computer science, v. 1, p. 225-253, 1986.
- [Arv+90] ARVIND; NIKHIL, R.S. Executing a program on the MIT Tagged-Token dataflow architecture. IEEE Transaction on Computers, v.39, n.3, p.300-318, 1990.
- [Bec+90] BECKERLE, M.J.; TRAUB, K.R. Monsoon project software guidelines. Cambridge, Motorola Cambridge Research Center, march 1990.
- [Bic+89] BIC, L.; HARTMANN, R.L. AGM: a dataflow database machine. ACM Transactions on Database Systems, v.14, n.1, p.114-146, mar.1989.
- [Cal+s.d.] CALUWAERTS, L.J.; DEBACKER, J.; PEPERSTRAETÉ, J.A. Implementing code reentrancy in functional programs on a dataflow computer system with a paged memory. Belgium, Katholieke Universiteit Leuven, Dept. Elektrotechniek - ESAT, s.d.

- [Cau+95] CAUGHEY, G. et al. Usando CA - visual objects. Rio de Janeiro, Campus, 1995.
- [Cav92] CAVENAGHI, M.A. . Implementação de um simulador para a arquitetura Fluxo de dados WOLF. São Carlos, 1992. 80 p. Dissertação (Mestrado) - Instituto de Física e Química de São Carlos, Universidade de São Paulo.
- [Chi+95] CHIOU, D.; ANG, S. B.; ARVIND; BECKERLE, M.J.; BOUGHTON, A. GREINER, R.; HICKS, J.E.; HOE, J.C. StarT-ng: delivering seamless parallel computing. Massachusetts, Laboratory for Computer Science, Institute of Technology, july 1995.
- [Con89] CONCEPCION, A.I. A hierarchical computer architecture for distributed simulation. IEEE Transactions on Computers, v.38, n.2, p.311-319, feb. 1989.
- [Cul+88] CULLER, D.E.; MAA, G.K. Assessing the benefits of Fine - grained parallelism in dataflow programs, Proceedings of Supercomputing 88. Massachusetts, Laboratory for computer Science, Institute of Technology, 1988. In: Journal of Supercomputing Applications, nov.1988.
- [Cul+91] CULLER, D.E.; SAH, A.; SCHAUER, K. E.; VON EICKEN, T.; WAWRZYNEK, J. Fine-Grain parallelism with minimal hardware support: a compiler-controlled treaded abstracted machine. In 4th. International Conference on Architectural Support for Programming Languages and Operation Systems. Santa Clara, California, 1991.
- [Dal+92] DALLY, W. J. et al. Architecture of a message-driven processor. IEEE Micro, v.12, n.2, p.23-39, 1992.
- [Den80] DENNIS, J. B. Data-flow supercomputers. IEEE Computer, p.48-56, nov.1980.
- [Dig+81] DIGIACINTO, T. Airborne Associative Processor (ASPRO). Akron/ OH, Goodyear Aerospace Corporation, 1981.
- [Dun90] DUNCAN, R. A survey of parallel computer architectures. IEEE Computer, v.23, n.2, p.05-16, 1990.

- [Fen81] FENG, T.Y. A survey of interconnection networks. Computer, v.14, p.12-27, dec. 1981.
- [Fly72] FLYNN, M.J. Some computer organization and their effectiveness. IEEE Transactions on Computers, v.21C, p.948-960, 1972.
- [Fly66] FLYNN, M.J. Very High Speed Computing. Proc.IEEE, v. 54, 1966.
- [Gaj+82] GAJSKI, D.D.; PADUA, D.A.; KUCK, D.J. A second opinion on dataflow machines and languages. Computer, v. 15, p.58-69, feb. 1982.
- [Gho90] GHOSAL, D.; BHUYAN, L.N. Performance evaluation of a dataflow architecture. IEEE Transactions on Computers, v.39, n.5, p.615-627, 1990.
- [Gos+79] GOSTELOW, K.P.; THOMAS, R.E. Performance of a dataflow computer. Irvine, University of California, Dept. of Information and Computer Science, oct.1979.
- [Gra+89] GRAFE, V.G.; DAVIDSON, G.S.; HOCH, J.E.; HOLMES, V.P. The epsilon dataflow processor. New Mexico, Albuquerque, U.S. Departament of Energy at Sandia National Laboratories. In 16 th. International Symposium on Computer Architecture, 1989.
- [GRU92] GRUPO DE INSTRUMENTAÇÃO E INFORMÁTICA. The Wolf Dataflow architecture programming manual. Universidade de São Paulo, Instituto de Física e Química de São Carlos, Departamento de Física e Ciências dos Materiais, may 1992.
- [Gur+87] GURD, J.R.; KIRKHAM, C.C.; BÖHM, W. The manchester dataflow computing system. In: DONGARRA, J.J. Experimental parallel computing systems. Nort-Holland, Elsevier Science Publishing, 1987. p.177-219.
- [Gur+85] GURD, J.R.; KIRKHAM, C.C.; WATSON, I. The manchester prototype dataflow computer. Communications of the ACM, v.28, n.1, p.34-52, 1985.
- [Has+84] HASEGAWA, R.; MIKAMI, H.; AMAMIYA, M. An architecture for list - processing - oriented dataflow machine. Review of the Electrical Communication Laboratories, v.32, n.5, p.771-782, 1984.

- [Hay88] HAYES, J.P. Computer architecture and organization. 2. ed. New York, McGraw - Hill, 1988.
- [Hei+94] HEINRICH, M. et.al. The performance impact of flexibility in the stanford FLASH Multiprocessor. In Proc. of the Sixth International Conference on Architecture Support for Programming Languages and Operating Systems. San Jose, CA, p. 274-285, Oct. , 1994.
- [Her88] HERATH, J.; YAMAGUCHI, Y.; SAITO, N.; YUBA, T. Dataflow computing models, languages, and machines for intelligence computations. IEEE Transactions on Software Engineering, v.14, n.12, p.1805-1828, dec.1988.
- [Her87] HERATH, J.; YUBA, T.; SAITO, N. Dataflow computing. Lecture Notes in Computer Science, n.269, p.25-36, may 1987.
- [Hil+93] HILLIS, D.W.; TUCKER, L.W. The CM-5 connection machine: a scalable supercomputer. Communications of the ACM, v.36, n.11, p.31-40, nov. 1993.
- [Hir+s.d.] HIRAKI, K.; SEKIGUCHI, S.; SHIMADA, T. System architecture of a dataflow supercomputer. Japan, Electrotechnical Laboratory, s.d.
- [Hoc+88] HOCKNEY, R.W.; JESSHOPE, C.R. Parallel computers 2. 2. ed. IOP, 1988.
- [Hwa+84] HWANG, K.; BRIGGS, F.A. Computer architecture and parallel processing. New York, McGraw - Hill Book, 1984. p.732- 811.
- [Ina+s.d.] INAGAMI, Y.; FOLEY, J.F. The specification of a new manchester dataflow machine. England, Central Research Laboratory, University Road, s.d. p.371-380.
- [Kal+88] KALLSTROM, M.C.; BÖHM, A.P.W.; GARCIA NETO, A. ParSiFlow: a dataflow simulator on a transputer network. In Proc. Alvey Technical Conference. Manchester, University of Manchester, july 1988.
- [Kaw+86] KAWAKAMI, K.; GURD, J. A scalable dataflow structure store proceedings. In: 13 th. Annual International Symposium on Computer Architecture. New York, jun.1986.

- [Kod+95] KODAMA, Y.; SAKANE, H.; SATO, M.; YAMANA, H.; SAKAI, S.; YAMAGUCHI, Y. The EM-X parallel computer: architecture and basic performance. Computer Architecture News, p.14-23, may 1995.
- [LAB88] LABORATORY FOR COMPUTER SCIENCE. Future scientific programming on parallel machines. Massachusetts, Laboratory for Computer Science, Institute of Technology, 1988.
- [Lee+93] LEE, B.; HURSON, A.R. Issues in dataflow computing. Advances in Computers, v.37, p.285-333, 1993.
- [Mac85] MACGRAW, J.R. SISAL: streams and iteration in single assignment language: language reference manual, version 1.2. Lawrence Livermore National Laboratory, march 1985.
- [Mag92] MAGNA, P. Redução dos bits de emparelhamento da máquina de fluxo de dados de manchester. São Carlos, 1992. 79 p. Dissertação (Mestrado) - Instituto de Física e Química de São Carlos, Universidade de São Paulo.
- [Mar94] MARTINI, J.A. Estudo de desempenho de redes de interconexão para a arquitetura a fluxo de dados Proto - WOLF. São Carlos, 1994. 110 p. Dissertação (Mestrado) - Instituto de Física de São Carlos, Universidade de São Paulo.
- [Nik+89] NIKIL, R.S.; ARVIND. Can dataflow subsume von newmann computing? In Proc. 16th. Annual Int'l Symposium on Computer Architecture. New York, 1989.
- [Pap+90] PAPADOULOS, G.M.; CULLER, D.E. Moonson: an explicit token store Architecture. In Proc. 17th. Annual Int'l. Symposium on Computer Architecture. New York, 1990.
- [Ram+93] RAMANATHAN, G.; OREN, J. Survey of commercial parallel machines. Computer Architecture News, v.21, n.3, p.13-33, jun. 1993.
- [Rug87] RUGGIERO, C. A. Throttle Mechanisms for Manchester Dataflow Machine. Manchester, 1987. Ph.D. Thesis, University of Manchester, Department of Computer Science.

- [Sak+90] SAKAI, S.; YAMAGUCHI, Y.; HIRAKI, K.; KADAMA, Y.; YUBA, T. An architecture of a dataflow single chip processor. In Proc. 16th. Annual Int'l. Symposium on Computer Architecture. New York, 1990.
- [Sar85] SARGEANT, J. Simulator users guide. Manchester, University of manchester , Dept. Computer Science, jan.1985.
- [Shi+83] SHIMADA, T.; HIRAKI, K.; NISHIDA, K. Architecture of a dataflow computer sigma-1 for scientific computation. Japan, Institute of Electronic and Communications Engineers of Japan, july 1983.
- [Ske+85] SKEDZIELEWSKI, S.K.; GLAUERTI, J.R.W. IF1: an intermediate form for applicative language reference manual. Lawrence Livermore National Laboratory, july 1985.
- [Sto+82] STOLFO, S.J.; DAVID, E.S. DADO: a tree-structured machine architerture for production systems. DCS, Columbia University, 1982.
- [Tan92] TANENBAUM, A.S. Organização estruturada de computadores. Trad. de Hélio Marques Sobrinho e Luiz Fernando Costa. 3. ed. Rio de Janeiro, Prentice - Hall, 1992.
- [Toc+93] TOCCI, R.J.; LASKOWSKI, L.P. Microprocessadores e microcomputadores. Trad. de Richard Percival Geyer. 2. ed. Rio de Janeiro, Prentice - Hall, 1993.
- [Vid91/2] VIDAL, A.G.R. Clipper 5.0: funções, compilação, linkedição, execução. Rio de Janeiro, LTC, 1991. v.2.
- [Vid91/3] VIDAL, A .G.R. Clipper 5.0: rede local, utilitários, aplicações, erros. Rio de Janeiro, LTC, 1991. v.3.
- [Wat+82] WATSON, I.; GURD, J. A practical dataflow computer. IEEE Computer, v.15, n.2, p.51-57, feb.1982.
- [Yub+87] YUBA, T.; HIRACHI, K.; SHIMADA, T.; SEKIGUCHI, S.; NISHIDA, K. The SIGMA-1 Dataflow computer. Japan, Electrotechnical Laboratory, 1987.

[Yub+90] YUBA, T.; SHIMADA, T.; YAMAGUCHI, Y.; HIRAKI, K.; SAKAI, S.
Dataflow computer development in Japan. Communications of the
ACM, v. 33, p.140-147, 1990.