Universidade de São Paulo

Instituto de Física e Química de São Carlos

Estudo de Processamento Paralelo

para

Dinâmica Molecular

Gonzalo Travieso

Dissertação apresentada ao

Instituto de Física e Química de São Carlos,

para a obtenção do título de

Mestre em Física Aplicada

Orientador: Prof. Dr. *Jan Frans Willem Slaets*

Departamento de Física e Ciência dos Materiais

São Carlos - 1989

MEMBROS DA COMISSÃO JULGADORA DA DISSERTAÇÃO DE MESTRADO DE Gonzalo Travieso APRESENTADA AO INSTITUTO DE FÍSICA E QUÍMICA DE SÃO CARLOS, DA UNIVERSIDADE DE SÃO PAULO, EM 10 DE março DE 1989.

COMISSÃO JULGADORA:

Dr. Jan F.W.Slaets Orientador

Dr. Carlos Antonio Ruggiero

Dr. Marcos Nereu Arenales

Agradecimentos

Aos meus pais, pelo carinho e pelo apoio constantes.

Ao prof. Jan, cujas orientações neste e em outros trabalhos muito influenciaram minha formação profissional.

Aos profs. Álvaro, Caetano e Luciano e ao Lírio, pelas discussões e comentários ricos e amizade profunda.

Aos colegas Ailton, Marquinhos, Ivanilda, Lia, Benê, prof. Ruggiero, prof. Valentin e profa. Stela, pela atmosfera de amizade.

À Mônica, pela inestimável ajuda no desenvolvimento deste trabalho.

Índice

Resumo 11

Abstract 13

Introdução 15

1 Dinâmica Molecular 19

1.1 Introdução 19

1.2 Algoritmo básico 23

1.3 Algoritmo básico modificado 31

1.4 Algoritmo de tabela de vizinhos 33

1.5 Algoritmo de granulação grosseira 34

1.6 Conclusão 40

2 Eficiência dos algoritmos seqüenciais 41

2.1 Introdução 41

2.2 Tempos de execução 41

2.2.1 Tempo de execução do algoritmo básico 44

2.2.2 Tempo de execução do algoritmo básico modificado 48

2.2.3 Tempo de execução do algoritmo de tabela de vizinhos com atualização infreqüente 50

2.2.4 Tempo de execução do algoritmo de granulação grosseira 54

2.3 Comparaçao dos algoritmos 60

3 Paralelismo **65**

3.1 Introdução 65

3.2 Paralelismos inerentes ao problema 66

3.3 Paralelismo no algoritmo básico 73

3.4 Paralelismo no algoritmo básico modificado 74

3.5 Paralelismo no algoritmo de tabela de vizinhos 77

3.6 Paralelismo no algoritmo de granulação grosseira 80

3.7 Conclusão 86

4 Definição da proposta **87**

4.1 O *transputer* T800 87

4.1.1 Estrutura da unidade de processamento inteiro 88

4.1.2 Introdução às instruções de máquina 89

4.2 A linguagem OCCAM 90

4.2.1 Processos básicos 91

4.2.2 Estruturas de controle de fluxo 91

4.2.3 Estruturas de dados e de canais 95

4.2.4 Outras características importantes 99

4.3 Esquema da implementação 101

4.3.1 Divisão geométrica 102

4.3.2 Divisão algoritmica 108

4.4 Escolha dos parâmetros 118

5 Avaliação da proposta **125**

5.1 Introdução 125

5.2 Descrição da metodologia 126

5.3 Variação de N, fixos ϱ e p 130

5.4 Variação de p, fixos N e ϱ 133

5.5 Tempos no Cray X-MP 140

5.6 Conclusão 142

6 Conclusões **145**

6.1 Considerações gerais 145

6.2 Futuros trabalhos 146

A Esquematização do Paralelismo **149**

A.1 Introdução 149

A.2 Elementos básicos 149

A.2.1 Representação de processos 150

A.2.2 Representação de comunicações entre processos 152

A.2.3 Representação de elementos de armazenamento 152

A.3 Algumas convenções auxiliares 155

A.4 Exemplos de representações 162

B Descrição em OCCAM dos processos avaliados **175**

Lista de Figuras

1 Partículas distribuídas no espaço de simulação 20

2 Periodicidade do espaço de simulação 20

3 Pseudo-código do algoritmo básico 24

4 Algoritmo para a renormalização das velocidades 30

5 Pseudo-código do algoritmo básico modificado 32

6 Raio de vizinhança em relação ao raio de corte 34

7 Pseudo-código do algoritmo de tabela de vizinhos 35

8 Divisão do espaço em grãos. 36

9 Raio de corte de uma partícula nos grãos. 37

10 Pseudo-código do algoritmo de granulação grosseira 39

11 Evolução do tempo de execução com o número de partículas para o algoritmo básico 48

12 Evolução do tempo de execução do algoritmo básico modificado com o número de partículas 52

13 Evolução do tempo de execução com o número de partículas para o algoritmo de tabela de vizinhos 56

14 Evolução do tempo de execução com o número de partículas para o algoritmo de granulação grosseira 60

15 Comparação dos tempos de execução dos algoritmos básico e básico modificado 61

16 Comparação dos tempos de execução dos algoritmos básico modificado e tabela de vizinhos com atualização infreqüente 62

17 Comparaçao dos tempos de execução dos algoritmos de tabela de vizinhos com atualização infreqüente e de granulação grosseira 62

18 Esquema do paralelismo da dinâmica molecular 72

19 Diagrama do algoritmo básico modificado 76

20 Estrutura de dados para tabela de vizinhos com ponteiros 78

21 Diagrama do algoritmo de tabela de vizinhos 81

22 Diagrama do algoritmo de granulação grosseira 82

23 Estrutura do *transputer* T800 88

24 Divisão do espaço de simulação em faixas 103

25 Estrutura de interligação das unidades de processamento 104

26 Expansão de um anel com $p = 4$ para $p = 5$ 105

27 Divisão do espaço de simulação em duas direções 106

28 Anel bidimensional de processadores 106

29 Expansão do anel bidimensional 107

30 Duplicação dos dados de grãos dos limites 108

31 Estrutura para o armazenamento de dados dos grãos 110

32 Estrutura do programa incluindo processos para comunicação entre as unidades de processamento 115

33 Distribuição inicial das partículas em rede 120

34 Gráfico de $T \times N$ para $\varrho = 0,6$ e $p = 8$ 131

35 Gráfico de $T \times N$ para $\varrho = 0,82$ e $p = 8$ 132

36 Variação de T com p para $N = 2016$ e $\varrho = 0,6$ 137

37 Variação de T com p para $N = 2016$ e $\varrho = 0,82$ 137

38 Variação de T com p para $N = 8064$ e $\varrho = 0,6$ 138

39 Variação de T com p para $N = 8064$ e $varrho = 0,82$ 138

40 Variação de T com p para $N = 32256$ e $varrho = 0,6$ 139

41 Variação de T com p para $N = 32256$ e $\varrho = 0,82$ 139

42 Representação de processos 150

43 Processo de soma $a = b + c$ 151

44 Multiplicação elemento a elemento de duas matrizes 152

45 Diagrama para o produto de um vetor de 20 componentes por um escalar . 153

46 Representação do armazenamento de uma matriz duplamente indexada . . 153

47 Representação de índices alternativos 154

48 Representação de acessos independentes ao mesmo elemento de armazenamento 155

49 Entradas e saídas múltiplas 156

50 Exemplo de expansão de multiplicidade com n múltiplo de m 157

51 Exemplo de redução de multiplicidade com m múltiplo de n 157

52 Exemplo de expansão de multiplicidade com m e n primos entre sí 158

53 Exemplo de redução de multiplicidade com m e n primos entre sí 158

54 Comunicação entre diferentes elementos do mesmo símbolo 163

55 Exemplo de multiplicidade variante com ampliação e redução de multiplicidade 164

56 Processo para realização da operação $a = (b + c)(d + e)$ 165

57 Processo para produto escalar de dois vetores 166

58 Produto escalar com memória única 167

59 Processo para $A \leftarrow A + B$ 168

60 Diagrama para $a \leftarrow a + 1$ seguido de $b \leftarrow 2a$ 169

61 Exemplo de sincronização com valor de dados: cálculo do fatorial 169

62 Sincronização por valor de dados: Soma de elementos positivos de um vetor 170

63 Exemplo de utilização de decisão sobre dados 171

64 Exemplo da utilização dos sinais D e L 172

65 Produto de matriz por vetor 173

66 Produto de matriz por vetor com a multiplicidade explicitada 173

67 Combinação de 10 elementos dois a dois e soma dos 10 elementos 174

Lista de Tabelas

I Tempo de execução do algoritmo básico 47

II Tempo de execução do algoritmo básico modificado 51

III Tempo de execução do algoritmo de tabela de vizinhos com atualização infreqüente 55

IV Tempo de execução do algoritmo de granulação grosseira 59

V Confrontamento dos tempos de execução dos diversos algoritmos 61

VI Tempos de execução dos processos 117

VII Ocupação de memória pelos vários processos 119

VIII Variações em N com $\varrho = 0,6$ e $p = 8$ 131

IX Variação de N com $\varrho = 0,82$ e $p = 8$ 132

X Efeitos da variação de p com $N = 2016$ e $\varrho = 0,6$ 134

XI Efeitos da variação de p com $N = 2016$ e $\varrho = 0,82$ 134

XII Efeitos da variação de p com $N = 8064$ e $\varrho = 0,6$ 135

XIII Efeitos da variação de p com $N = 8064$ e $\varrho = 0,82$ 135

XIV Efeitos da variação de p com $N = 32256$ e $\varrho = 0,6$ 136

XV Efeitos da variação de p com $N = 32256$ e $\varrho = 0,82$ 136

XVI Tempos de simulação no Cray X-MP 140

Resumo

Apresentamos um problema de dinâmica molecular e quatro algoritmos seqüenciais para a implementação do mesmo. Em seguida esses algoritmos são estudados quanto ao tempo de execução e possibilidades de paralelização. É escolhido então dentre os quatro o algoritmo que apresenta melhores características para a paralelização. Introduzimos a seguir uma proposta de implementação do mesmo em um rede de *transputers*, com a definição das interligações entre os processadores e da programação dos mesmos. A seguir é realizado um estudo da eficiência da estrutura proposta quanto a tempo de execução e características de expansibilidade do número de processadores. Os resultados mostram que conseguem-se velocidades de execução próximas às de supercomputadores para redes com baixos números de elementos.

Abstract

In the present work, we describe four sequential algorithms for simulating molecular dynamics. The parallelism and execution times of these algorithms are assessed. Using the best suited algorithm for parallelism exploitation a *transputer* based architecture is suggested including needed link and software. The evaluation of the eficiency regarding execution time and number of processors is analyzed. The results show that speeds close to those of supercomputers can be achieved with a low number of processors.

Introdução

Ao lado do grande crescimento de velocidade de execução alcançado pelos computadores nas últimas décadas, crescimento este que permitiu a ampliação da faixa de situações em que o computador se mostra útil, temos um crescimento ainda maior da demanda de recursos computacionais cada vez mais eficientes por parte dos usuários.

Uma das faixas de aplicação em que isto se faz mais sentir é a área denominada processamento numérico, ou *number crunching*. Mesmo com a introdução relativamente recente de computadores vetoriais com capacidade de pico da ordem de 200 MFlops, muitos problemas não podem ser executados a contento em computadores tradicionais devido ao enorme tempo de cálculo exigido, mesmo em computadores com processamento vetorial. Exemplos típicos de problemas que se encaixam nesta categoria são os de meteorologia, simulação de aeronaves e problemas de física teórica, como a cromodinâmica quântica e a dinâmica molecular.

Particularmente em dinâmica molecular, diversas simulações têm sido executadas em supercomputadores, explorando sua capacidade de processamento vetorial através de reorganização dos algoritmos puramente seqüenciais. Veja-se por exemplo [1]. Ainda assim as simulações têm se restrito a no máximo poucos milhares de partículas.

Além da utilização de computadores extremamente rápidos, podemos também reduzir os tempos de execução pelo emprego de processamento altamente paralelo. Muitas máquinas de propósito geral com utilização de processamento paralelo têm sido propostas e implementadas recentemente, como por exemplo o *hipercubo* [2] e a estrutura *dataflow* [3].

Com raras exceções (como o *dataflow*) essas arquiteturas exigem o desenvolvimento de programas específicos para a utilização nas mesmas, sendo necessário um grande conhecimento da máquina por parte do programador para se conseguir uma execução eficiente (veja por exemplo [4]).

Como alternativa à utilização de supercomputadores, que para estes problemas implicariam um custo muito elevado, surgiu a técnica de desenvolvimento de computadores chamados *máquinas dedicadas* ou *computadores de propósito específico*, cuja principal característica é terem a sua estrutura interna especificamente adaptada à resolução eficiente de um único problema. Na verdade esta técnica consiste em uma especialização máxima, aprofundando-se a técnica de projeto de estruturas de computadores voltadas a algoritmos (isto é, de aplicação a uma faixa limitada de problemas). Exemplos de projetos voltados a algoritmos podem ser encontrados em [5] e nos *systolic arrays* [6]. Para mais detalhes sugerimos o trabalho de Uhr [7].

A vantagem desta abordagem é que, possuindo-se uma máquina dedicada, mesmo que sem toda a velocidade de um supercomputador, podem-se executar cálculos de problemas muito mais amplos e realistas, pois a máquina está disponível todo o tempo para o problema, e não precisa ter seu tempo compartilhado com outros usuários.

A idéia deste trabalho é propor uma tal máquina para um problema específico de dinâmica molecular, sendo a mesma utilizável em outros problemas semelhantes. Bakker [8] já apresentou uma arquitetura dedicada à dinâmica molecular, e outra está em desenvolvimento pela IBM [9]. As razões para estarmos aprentando uma outra proposta aqui, podem ser resumidas nos seguintes objetivos:

- a máquina deve apresentar uma estrutura simples, que permita sua fácil duplicação
- as expansões para a inclusão de novos processadores devem ser fáceis
- o *hardware* da máquina deve permitir sua utilização para problemas diferentes do

originalmente proposto sem muitas alterações

No capítulo 1 apresentamos rapidamente o problema de dinâmica molecular a ser utilizado, seguido de quatro algoritmos seqüenciais para o mesmo.

No capítulo 2 realizamos uma análise de eficiência seqüencial dos quatro algoritmos em termos de tempo de execução. Em seguida, o capítulo 3 apresenta um estudo das possibilidades de implementação paralela dos mesmos algoritmos. Ao fim desse capítulo, realizamos a escolha do algoritmo a utilizar.

No capítulo 4 apresentamos inicialmente, de uma forma breve, o *transputer* T800 e a linguagem OCCAM, sugeridos para a implementação da máquina. Apresentamos também as razões dessa escolha. Em seguida apresentamos a proposta de programação e interligação de processadores que julgamos adequada ao problema em questão.

No capítulo 5 apresentamos uma avaliação da proposta, através do levantamento de tempos de execução e eficiência de utilização de processadores para diversos números de partículas e processadores, bem como uma comparação com um tempo de execução obtido no Cray X-MP.

Capítulo 1

Dinâmica Molecular

1.1 Introdução

Chamamos de *dinâmica molecular* a uma técnica que simule a evolução temporal de um dado agregado de partículas governadas por suas interações mútuas.

O experimento é realizado através da simulação do movimento de cada uma das partículas de um agregado, determinado pela interação entre as mesmas, e por um conjunto de condições iniciais e de condições de contorno do espaço onde as partículas estão distribuídas. Chamaremos de um "quadro" à distribuição das partículas pelo espaço em um dado instante da simulação (conforme fig. 1, para o caso bidimensional).

Devido à pequena quantidade de partículas que podem ser utilizadas em um experimento por computador, em relação às quantidades presentes em sistemas reais relevantes, devem ser impostas algumas condições de contorno para o problema, de forma a que a influência desse fator possa ser minimizada. O procedimento comumente adotado consiste em considerar condições de contorno periódicas, fazendo com que o espaço no qual as partículas estão distribuídas seja repetido periodicamente em todas as direções. Desta forma, uma partícula que sai por uma das faces que limitam esse espaço, entra pela face oposta (veja fig. 2).

Fornecidas essas condições iniciais e de contorno, os quadros sucessivos do sis-

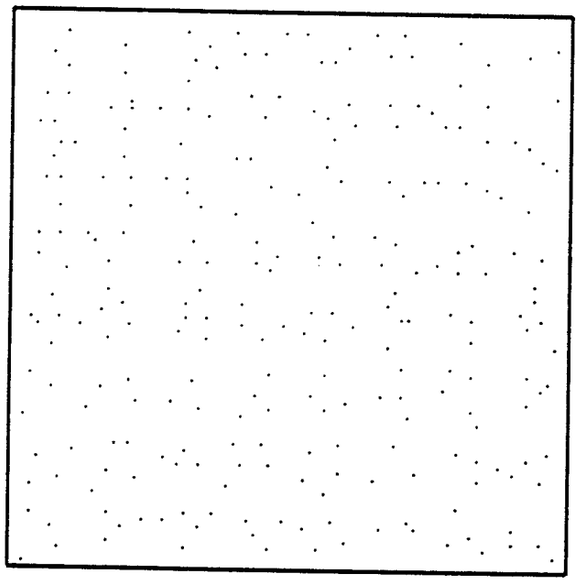


Figura 1: Partículas distribuídas no espaço de simulação

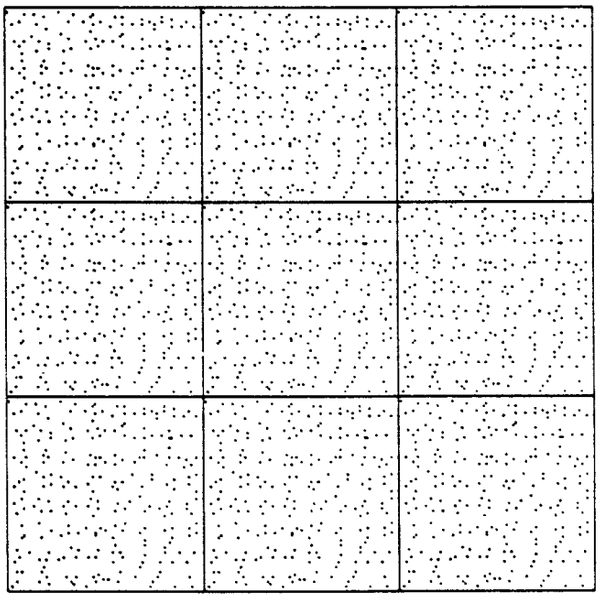


Figura 2: Periodicidade do espaço de simulação

tema de partículas podem ser encontrados através do cálculo da força de interação entre as diversas partículas (realizado levando-se em consideração a função de potencial especificada pelo usuário) e as equações de movimento de Newton (mais sobre esse processo de cálculo será apresentado ainda neste capítulo).

Para levar em consideração condições termodinâmicas específicas (como por exemplo: sistema isotérmico ou isobárico) inclue-se uma fase de renormalização a ser realizada em apenas alguns dos "quadros". Esta fase corresponde a uma mudança a ser introduzida nas coordenadas ou velocidades das partículas de modo a forçar as variáveis termodinâmicas ao valor desejado. Como exemplo, para conseguirmos um sistema isotérmico devemos introduzir uma renormalização das velocidades.

Todo o processo é acompanhado pelo usuário apenas através de um pequeno número de varíaveis referentes ao agregado como um todo. Essas varíaveis são escolhidas de acordo com as características termodinâmicas que o usuário pretende monitorar no experimento em andamento. Durante o cálculo dos quadros sucessivos, o computador encontra os valores dessas variáveis referentes ao agregado. Elas representam a "saída" do sistema, e serão utilizadas para o estudo físico a ser realizado.

Resumindo, o processo de simulação pode ser indicado como:

1. formar os "quadros" (ou instantes de simulação) sucessivos, sendo cada qual calculado a partir de dois anteriores (esta é uma das formas de especificação das condições iniciais);

2. para o cálculo da posição de uma partícula, encontrar todas as forças de interação dessa partícula com as demais do agregado, e através da força total encontrada e das condições iniciais, calcular a nova posição;

3. em alguns dos quadros, realizar a renormalização especificada;

4. durante o processo de cálculo dos "quadros" sucessivos, calcular também os valores

de algumas quantidades referentes ao agregado, e que serão utilizadas para o acompanhamento da evolução do processo pelo usúario.

O processo de simulação pode se desenrolar para os casos bi e tridimensional, sendo todo o processamento basicamente o mesmo. Como o problema proposto originalmente para este trabalho trata do caso bidimensional [10], todas as considerações a serem realizadas a diante se referem a este caso. Entretanto, as alterações para o caso tridimensional são triviais, pela simples inclusão da nova coordenada nos cálculos.

A simulação de dinâmica molecular pode ser realizada com um único ou vários tipos diferentes de partículas. No trabalho apresentado aqui, consideramos apenas o caso de um único tipo de partícula. Para trabalhar com mais de um tipo de partícula, as alterações a serem consideradas se restringem ao cálculo do potencial interpartículas e aos pontos onde é importante a massa da mesma, desde que as partículas não sejam sujeitas à reações químicas entre sí (isto é, a estrutura das partículas permanece inalterada por toda a simulação).

Na proposição deste trabalho, incluia-se a renormalização das velocidades de forma a se ter um sistema isotérmico. A descrição dessa renormalização, que será a única considerada daqui por diante, pode ser encontrada à pag. 30.

A seguir descreveremos quatro algoritmos que podem ser utilizados para se executar a dinâmica molecular. O primeiro algoritmo, que chamamos de *algoritmo básico*, apresenta baixa eficiência (isto é, grande quantidade de cálculos para um dado número do partículas). A partir do mesmo serão desenvolvidos os outros três algoritmos, chamados aqui de *algoritmo básico modificado*, *algoritmo de tabela de vizinhos* e *algoritmo de granulação grosseira*, os quais consistem em métodos de otimização dos cálculos. Esses algoritmos estão baseados numa execução puramente seqüencial; os estudos de paralelismo serão realizados no capítulo 3.

1.2 Algoritmo básico

O algorítmo básico pode ser descrito como mostrado no pseudo-código da fig. 3. Para a construção desse algoritmo, partimos do princípio de que as quantidades agregadas a serem calculadas são a energia potencial e a energia cinética. Este pode não ser o caso em muitos dos problemas, mas foi escolhido a título de ilustração e por consistir nas quantidades propostas originalmente para este trabalho. No pseudo-código da fig. 3 utilizamos a seguinte notação:

N é o número total de partículas do agregado

N_q é o número de quadros da simulação

q_r é o número de quadros entre renormalizações (veja pag. 28 para maiores detalhes)

além de outros fatores definidos a seguir.

Os vários cálculos envolvidos são operados da seguinte forma:

Interação entre partículas i e j: Este cálculo é realizado a partir de uma função potencial que é fornecida pelo usuário. Uma função potencial comumente utilizada é a de Lennard-Jones [9]:

$$u_{ij} = 4\varepsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right] \tag{1}$$

onde:

u_{ij} indica o potencial entre as partículas i e j,

r_{ij} indica a distância entre as mesmas,

σ indica a distância para a qual o potencial interpartículas se anula e

ε indica a profundidade do mínimo do potencial de Lennard-Jones.

para $q = 1$ até N_q

{para $i = 0$ até $N - 1$

{para $j = 0$ até $N - 1$, $j \neq i$

{calcule distância entre i e j (r_{ij}^2)

calcule potencial entre i e j

calcule força entre i e j

acumule potencial em potencial total

acumule força em força sobre i }

calcule nova posição de i

calcule velocidade de i

calcule energia cinética de i

acumule em energia cinética total }

atualize valores de posições das partículas

se q é múltiplo de q_r, faça renormalização }

Figura 3: Pseudo-código do algoritmo básico

Para o cálculo do potencial é necessário, portanto, conhecer-se a distância entre as duas partículas. Essa distância pode ser calculada conforme fórmula abaixo:

$$r_{ij}^2 = (x_i - xj)^2 + (y_i - y_j)^2 \tag{2}$$

onde:

(x_i, y_i) indicam as coordenadas da partícula i.

(x_j, y_j) indicam as coordenadas da partícula j.

Veja que a equação 2 exprime o cálculo da distância quadrática. Não é necessário o cálculo da sua raiz quadrada.

Sendo o potencial variável com a distância entre as partículas, podemos também descrever a força interpartículas a partir da distância entre as mesmas. Isto pode ser feito levando-se em consideração a relação entre o potencial e a força de interação, como mostrado pela equação abaixo:

$$\begin{aligned} \vec{F}_{ij} &= -\frac{\partial u_{ij}}{\partial x_i}\hat{x} - \frac{\partial u_{ij}}{\partial y_i}\hat{y} \\ &= -\frac{\partial u_{ij}}{\partial r_{ij}}\left(\frac{\partial r_{ij}}{\partial x_i}\hat{x} + \frac{\partial r_{ij}}{\partial y_i}\hat{y}\right) \end{aligned} \tag{3}$$

onde:

$\vec{F}_{ij}$ indica a força exercida na partícula i pela partícula j,

$\hat{x}$ indica a direção do eixo de coordenadas x

$\hat{y}$ indica a direção do eixo de coordenadas y

Levando em consideração (2) encontramos:

$$\begin{aligned} \frac{\partial r_{ij}}{\partial x_i} &= \frac{x_i - x_j}{r_{ij}} \\ \frac{\partial r_{ij}}{\partial y_i} &= \frac{y_i - y_j}{r_{ij}} \end{aligned}$$

e, portanto:

$$\vec{F}_{ij} = -\frac{1}{r_{ij}}\frac{\partial u_{ij}}{\partial r_{ij}}\left[(x_i - x_j)\hat{x} + (y_i - y_j)\hat{y}\right] \tag{4}$$

Para o caso do potencial de Lennard-Jones temos:

$$\frac{\partial u_{ij}}{\partial r_{ij}} = \frac{4\varepsilon}{r_{ij}}\left[-12\left(\frac{\sigma}{r_{ij}}\right)^{12} + 6\left(\frac{\sigma}{r_{ij}}\right)^{6}\right]$$

e podemos descrever a força como função da distância da seguinte forma:

$$\vec{F}_{ij} = \frac{4\varepsilon}{r_{ij}^2}\left[12\left(\frac{\sigma}{r_{ij}}\right)^{12} - 6\left(\frac{\sigma}{r_{ij}}\right)^{6}\right]\left[(x_i - x_j)\hat{x} + (y_i - y_j)\hat{y}\right] \tag{5}$$

Em termos do processo discreto de simulação, e levando em consideração a possibilidade de tabelamento das relações de força e potencial com a distância, as relações acima ficam:

$$u_{ij} = H(r_{ij}) \tag{6}$$

$$\vec{F}_{ij} = G(r_{ij})\left[(x_i - x_j)\hat{x} + (y_i - y_j)\hat{y}\right] \tag{7}$$

onde:

$H(r_{ij})$ é uma função que nos dá o potencial interpartículas, para uma dada distância

e

$G(r_{ij})$ representa o fator comum no cálculo da força interpartículas.

Acumulação de Energia Potencial: O potencial calculado conforme indicado no ítem anterior é acumulado num totalizador de energia potencial.

$$U = \frac{1}{2}\sum_{i=0}^{N-1}\sum_{\substack{j=0 \\ j\neq i}}^{N-1} u_{ij} \tag{8}$$

onde:

U é o totalizador do potencial

Acumulação da Força sobre a partícula i: A força de interação exercida por todas as partículas j sobre a partícula i é acumulada para produzir a força total agindo sobre a partícula i (em cada direção).

$$\vec{F}_i = \sum_{\substack{j=0 \\ j\neq i}}^{N-1} \vec{F}_{ij} \tag{9}$$

onde:

$\vec{F}_i$ é a força sobre a partícula i

Cálculo da nova posição da partícula i: A nova posição de uma partícula pode ser calculada a partir da força total que age sobre a mesma e das equações de movimento. Utilizando um algoritmo de diferença central sugerido por Verlet em 1967 [11], podemos escrever a nova posição como:

$$x_i(t+\Delta t) = 2x_i(t) - x_i(t-\Delta t) + \frac{\Delta t^2}{m} F_{x_i}(t) \quad (10)$$

$$y_i(t+\Delta t) = 2y_i(t) - y_i(t-\Delta t) + \frac{\Delta t^2}{m} F_{y_i}(t) \quad (11)$$

onde:

Δt indica o intervalo de tempo entre dois "quadros" sucessivos

m é a massa das partículas

F_{x_i} indica a componente x da força sobre a partícula i

F_{y_i} indica a componente y da força sobre a partícula i

Acumulação da Energia Cinética: A energia cinética da partícula é calculada a partir da velocidade da mesma, sendo que esta pode ser encontrada pelas fórmulas seguintes (diferença central):

$$v_{x_i} = \frac{x_i(t+\Delta t) - x_i(t-\Delta t)}{2\Delta t} \quad (12)$$

$$v_{y_i} = \frac{y_i(t+\Delta t) - y_i(t-\Delta t)}{2\Delta t} \quad (13)$$

onde:

v_{x_i} é a velocidade da partícula i na direção x

v_{y_i} é a velocidade da partícula i na direção y

Calculamos então a energia cinética:

$$E_{c_i} = \frac{m}{2}\left(v_{x_i}^2 + v_{y_i}^2\right) \quad (14)$$

A energia cinética deve ser acumulada num totalizador da energia cinética do agregado:

$$E_c = \sum_{i=0}^{N-1} E_{c_i} \tag{15}$$

Renormalização: Para o nosso caso específico, a renormalização consiste em reescalonar todas as velocidades de forma a que sua média aritmética seja zero (pode deixar de ser zero devido à precisão limitada do processo numérico envolvido), e a média quadrática corresponda à média requerida para que o sistema se mantenha à temperatura especificada. Como a velocidade não entra no processo de cálculos dos quadros, a mesma deve ser encontrada, ter seu valor renormalizado, e então ser utilizada para formar dois quadros sucessivos, que serão as condições iniciais para o cálculo do próximo. Todo esse processo pode ser visto no pseudo-código da fig. 4. Os passos apresentados nesse pseudo-código podem ser definidos como a seguir:

Acumulação das velocidades: Corresponde ao processo de somar as velocidades de todas as partículas em cada uma das direções:

$$V_x = \sum_{i=0}^{N-1} v_{x_i} \tag{16}$$

$$V_y = \sum_{i=0}^{N-1} v_{y_i} \tag{17}$$

onde:

V_x é o total das velocidades na direção x

V_y é o total das velocidades na direção y

Cálculo da média aritmética das velocidades: Simplesmente uma divisão dos totais em cada direção pelo número de partículas:

$$\overline{v_x} = \frac{V_x}{N} \tag{18}$$

$$\overline{v_y} = \frac{V_y}{N} \tag{19}$$

$\overline{v_x}$ é a velocidade média na direção x

$\overline{v_y}$ é a velocidade média na direção y

Fator de renormalização das velocidades: É encontrado a partir da temperatura requerida e da média das temperaturas de todos os quadros desde a última renormalização:

$$v_f = \sqrt{\frac{T_r}{T}} \tag{20}$$

$$T = \frac{\sum_{k=q-q_r+1}^{q} T_k}{q_r} \tag{21}$$

onde:

v_f é o fator para renormalização das velocidades

T_r é a temperatura requerida

T é a temperatura média dos últimos q_r quadros

q é o número do quadro atual

T_k é a temperatura do k-ésimo quadro

Nova velocidade: Devemos encontrar a nova velocidade das partículas considerando-se v_f e as velocidades médias:

$$v_{x_i} = (v_{x_i} - \overline{v_x})v_f \tag{22}$$

$$v_{y_i} = (v_{y_i} - \overline{v_y})v_f \tag{23}$$

Nova posição: Calculamos qual seria a nova posição de uma partícula considerando sua nova velocidade e a aceleração do último quadro calculado:

$$x_i(t - \Delta t) \leftarrow x_i(t) \tag{24}$$

$$x_i(t) \leftarrow x_i(t - \Delta t) + v_{x_i}\Delta t + \frac{1}{2}a_{x_i}(t)\Delta t^2 \tag{25}$$

$$y_i(t - \Delta t) \leftarrow y_i(t) \tag{26}$$

$$y_i(t) \leftarrow y_i(t - \Delta t) + v_{y_i}\Delta t + \frac{1}{2}a_{y_i}(t)\Delta t^2 \tag{27}$$

onde:

para $i = 0$ até $N - 1$

{acumule velocidade de i em total de velocidade }

calcule a média aritmética das velocidades

encontre fator para renormalização das velocidades

para $i = 0$ até $N - 1$

{encontre nova velocidade de i

encontre nova posição a partir dessa velocidade }

Figura 4: Algoritmo para a renormalização das velocidades

$x_i(t - \Delta t)$ é a coordenada anterior da partícula i em x

$x_i(t)$ é a coordenada atual da partícula i em x

a_{x_i} é a aceleração da partícula i na direção x

$y_i(t - \Delta t)$ é a coordenada anterior da partícula i em y

$y_i(t)$ é a coordenada atual da partícula i em y

a_{y_i} é a aceleração da partícula i na direção y

Vê-se que, com o processo numérico utilizado (de diferença central), a nova posição de uma partícula é calculada conhecendo-se duas posições anteriores (ao invés da posição e velocidade anteriores), desta forma, para o seguimento do processo de cálculo, devemos sempre acumular a posição de todas as partículas em dois quadros consecutivos, o que possibilita encontrar o quadro seguinte. Devido a esse fato, ao fim do cálculo de um novo quadro e antes de iniciar o cálculo do próximo, devemos realizar a substituição de forma a que, para cada partícula, as duas posições mais recentes sejam utilizadas como base para o cálculo da próxima posição.

1.3 Algoritmo básico modificado

Na forma em que foi implementado, o algoritmo anterior apresenta diversas ineficiências que podem ser eliminadas sem uma alteração significativa na estrutura do mesmo.

Uma primeira ineficiência que pode ser eliminada com facilidade se refere ao fato de que as forças envolvidas são simétricas, isto é, a força que age sobre a partícula i devido à partícula j é igual em módulo, porém de sinal oposto, à força que age sobre a partícula j devido à partícula i. Se aproveitamos essa simetria, podemos realizar a metade dos cálculos de interação entre partículas.

Outra alteração que pode resultar em um grande ganho em termos de tempo de execução é o aproveitamento do fato de que, para grande parte dos potenciais inter-partículas de interesse (por exemplo o de Lennard-Jones), o número de partículas que apresentam uma força de interação significativa com uma partícula dada é extremamente baixo com relação ao total de partículas do agregado. Desta forma, podemos estabelecer um valor de distância entre as partículas (que chamaremos de *raio de corte* e designaremos por r_c), a partir da qual podemos considerar a força de interação como zero.

As alterações correspondentes aos dois fatos mencionados acima foram incluídas no algoritmo que denominamos de *algoritmo básico modificado* e estão representadas no pseudo-código da fig. 5. A existência de um raio de corte é aproveitada através de um teste sobre o valor da distância quadrática entre as partículas, e a simetria de forças é aproveitada criando uma tabela de forças sobre cada partícula, acumulando a interação entre duas partículas nos totalizadores de força das duas, e tomando cuidado de calcular cada par de partículas apenas uma vez.

As mesmas fórmulas anteriores são válidas, acrescentando-se apenas que:

$$F_{ij} = -F_{ji} \tag{28}$$

Note que desta vez, cada potencial de interação entre duas partículas é calculado

para $q = 1$ até N_q

{para $i = 0$ até $N - 2$

{para $j = i$ até $N - 1$

{calcule distância entre i e j (r_{ij}^2)

se $r_{ij}^2 < r_c^2$

{calcule potencial entre i e j

calcule força entre i e j

acumule potencial em potencial total

acumule força em força sobre i e sobre j } }

calcule nova posição de i e atualize

calcule velocidade de i

calcule energia cinética de i

acumule em energia cinética total }

se q é múltiplo de q_r, faça renormalização }

Figura 5: Pseudo-código do algoritmo básico modificado

apenas uma vez, o que dispensa a necessidade de se dividir o total acumulado por dois. Isto pode ser resumidc na seguinte fórmula:

$$\frac{1}{2}\sum_{i=0}^{N-1}\sum_{\substack{j=0\\ j\neq i}}^{N-1} u_{ij} = \sum_{i=0}^{N-2}\sum_{j=i+1}^{N-1} u_{ij}$$

Com isto completamos a descrição do algoritmo básico de dinâmica molecular. Mais adiante, no capítulo 2, realizaremos estudos das vantagens em termos de tempo de execução oferecidas pelas alterações realizadas acima.

1.4 Algoritmo de tabela de vizinhos

Este algoritmo, utilizado por Verlet em 1967 no seu experimento em computador de dinâmica molecular [11], e também chamado de algoritmo de tabela de vizinhos com atualização infreqüente (por razões que ficarão claras a seguir), é baseado fundamentalmente nos dois fatos seguintes:

1. o número de partículas que efetivamente interagem com uma dada partícula é pequeno perto do total de partículas;

2. o movimento de cada partícula entre dois quadros sucessivos é suficientemente pequeno para que o conjunto de partículas interagentes com uma dada partícula não varie significativamente por um certo número de "quadros".

Tendo em vista esses dados, Verlet implementou um algoritmo que se utiliza do seguinte método: para cada partícula é construída uma *tabela de vizinhos*, que relaciona todas as partículas que se encontram a uma distância menor que um dado *raio de vizinhança*, ligeiramente superior ao raio de corte (veja fig. 6), escolhido de forma que se garanta que nenhuma partícula que não esteja nesse raio de vizinhança vá entrar no círculo do raio de corte dentro de um número dado de interações. Isto permite que a atualização da tabela de vizinhos seja realizada infreqüentemente, espaçada do número de interações necessário a se garantir a condição de validade da relação entre o raio de vizinhança e o raio de corte.

As vantagens desse algoritmo com relação ao anterior, que se tornarão mais claras quando da realização do estudo de tempos de execução (no capítulo 2), provêm do fato de que a busca de partículas interagentes com uma partícula dada deve ser realizada verificando-se a distância entre todos os pares de partículas do sistema, o que toma muito tempo quando o número de partículas é grande. Este algoritmo, fazendo com que este cálculo seja realizado com menos freqüência, permite grande ganho de tempo de execução

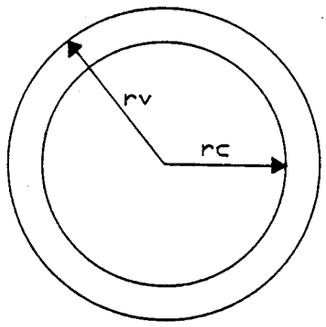


Figura 6: Raio de vizinhança em relação ao raio de corte

para praticamente qualquer quantidade de partículas.

O algoritmo completo pode ser descrito como no pseudo-código da fig. 7, onde introduzimos os seguintes fatores:

q_v é o número de quadros entre atualizações da tabela de vizinhos

r_v é o raio de vizinhança

$n_v[i]$ indica o número de vizinhos da partícula i

$v[i,k]$ indica qual a k-ésima vizinha da partícula i

Uma comparação do tempo de execução desse algoritmo com relação ao algoritmo básico será realizada no capítulo 2, junto com a apresentação de valores possíveis para a relação entre o raio de vizinhança e o raio de corte e para o número de passos sem atualização da tabela de vizinhos.

1.5 Algoritmo de granulação grosseira

O último algoritmo de implementação da dinâmica molecular em computador que estudaremos aqui, o qual denominamos *granulação grosseira* (do inglês: "coarse-grain") tem como base para sua utilização o fato de que as forças de interação entre as partículas são

para $q = 1$ até N_q

{se q é múltiplo de q_v

{para $i = 0$ até $N - 2$

para $j = i$ até $N - 1$

{calcule distância entre i e j (r_{ij}^2)

se $r_{ij}^2 < r_v^2$

{coloque j na tabela de vizinhos de i

atualize $n_v[i]$ } }

para $i = 0$ até $N - 1$

para $k = 0$ até $n_v[i]$

{calcule potencial entre i e $v[i,k]$

calcule força entre i e $v[i,k]$

acumule potencial em potencial total

acumule força em força sobre i e sobre $v[i,k]$ } }

calcule nova posição de i e atualize

calcule velocidade de i

calcule energia cinética de i

acumule em energia cinética total }

se q é multiplo de q_r, faça renormalização }

Figura 7: Pseudo-código do algoritmo de tabela de vizinhos

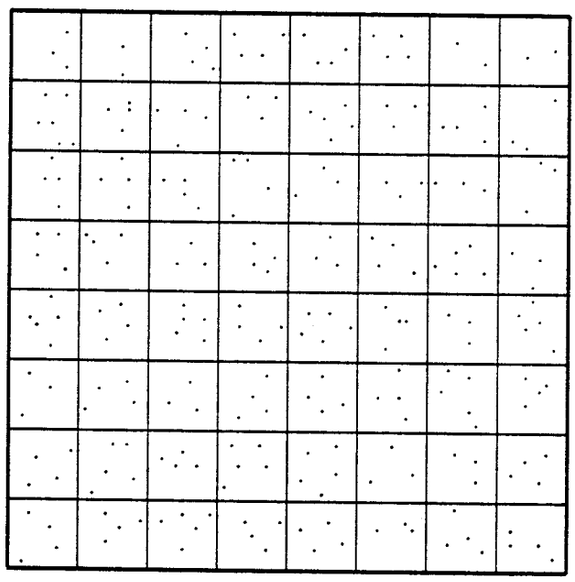


Figura 8: Divisão do espaço em grãos.

localizadas, isto é, que partículas que interagem entre sí estão próximas no espaço. Como veremos no capítulo 3, este fato é também extremamente importante para a implementação paralela do experimento.

O processo utilizado nesse algoritmo para se conseguir uma melhora no tempo de execução é o seguinte: o espaço de simulação é dividido em trechos iguais, chamados *grãos*, conforme fig. 8 (para o caso bidimensional). Escolhendo-se convenientemente o tamanho do grão, é possível garantir-se que todas as partículas que interagem com uma dada partícula de um grão estão presentes no mesmo grão ou nos grãos vizinhos ao que contém a partícula. Particularmente, se o tamanho do grão é maior ou igual ao raio de corte (r_c), todas as partículas que interagem com as partículas de um dado grão estão contidas nesse mesmo grão ou em um de seus vizinhos imediatos. Na fig. 9 mostra-se este fato para o caso de um espaço bidimensional.

A diminuição do tempo de execução pela utilização desse algoritmo é devida ao

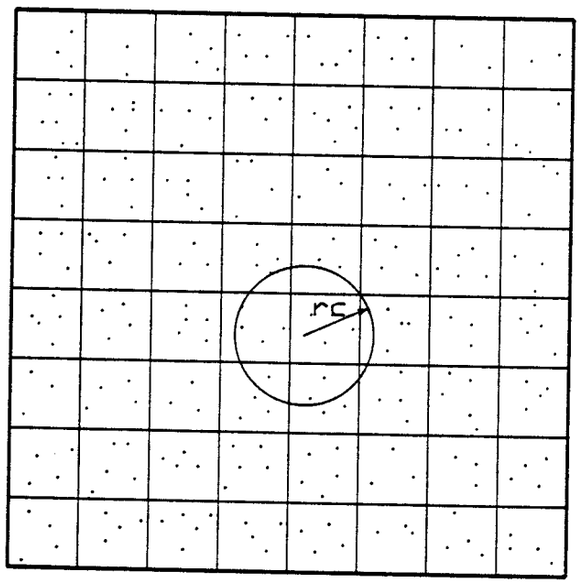


Figura 9: Raio de corte de uma partícula nos grãos.

fato de que, se no algoritmo básico precisávamos varrer todas as partículas do agregado para encontrar quais as que interagiam com uma partícula dada, no caso atual devem ser varridas apenas as partículas de um pequeno número de grãos. Como veremos no capítulo 2, isto representa um ganho extremamente elevado quando se trata com grandes quantidades de partículas.

O algoritmo envolvido pode ser descrito como no pseudo-código apresentado na fig. 10. Partiu-se da hipótese de que o tamanho do grão é maior ou igual ao do raio de corte (as razões que levam à escolha do valor preciso do tamanho do grão serão apresentadas futuramente neste trabalho, pag. 121).

Algumas explicações são necessárias sobre esse algoritmo. Em primeiro lugar, deve-se dizer que o método utilizado para se implementar o armazenamento das partículas nos grãos correspondentes não é o único possível, nem necessáriamente o mais eficiente. O método utilizado foi escolhido por permitir uma construção simples do pseudo-código

explicativo.

A cada partícula do agregado se associou um índice, pelo qual ela passa a ser conhecida durante todo o resto do experimento. O espaço foi dividido em N_g grãos, com a estrutura de vizinhança entre os mesmos armazenada em uma matriz $g_v[g,k]$, indicando todos os vizinhos do grão g (k sendo o índice de vizinhança, variando de 0 ao número total de grãos vizinhos menos um, $n_{gv}-1$). Para encontrar o índice de uma partícula, utilizamos a matriz $p[g,l]$, onde o primeiro índice indica qual o grão a que a mesma pertence e o segundo indica o número da partícula dentro do grão (a informação deste segundo índice é irrelevante para o processamento, servindo apenas para organização do armazenamento). Cada grão possui um número de partículas indicado por $n_p[g]$.

Deve-se notar um fato importante: da forma como está apresentado, esse algoritmo não aproveita a simetria de forças de ação e reação. Desta forma, temos uma duplicação de toda a parte do algoritmo que trata do cálculo de distâncias, potenciais e forças. O aproveitamento da simetria não é, entretanto, impossível com o presente algoritmo, não tendo sido apresentado apenas com o intuito de concentrar a atenção sobre os aspectos particulares desse algoritmo, que se tornaria mais complexo com a indicação da forma mais otimizada do mesmo.

Uma outra otimização do algoritmo é possível se realizarmos a atualização da divisão das partículas pelos grãos de forma infreqüente, do mesmo modo como foi feito com a tabela de vizinhos. No entanto, esta alteração não representa um ganho tão significativo quanto o que representou para o caso da tabela de vizinhos, pois o grão ao qual uma partícula pertence pode ser encontrado unicamente levando em consideração as coordenadas da própria partícula, o que indica que o crescimento do tempo de execução desse processo é linear com o número de partículas, enquanto que o cálculo da tabela de vizinhos, por levar em consideração a distância entre *pares* de partículas, cresce com o quadrado do número de partículas.

para $q = 1$ até N_q

{para $g = 0$ até $N_g - 1$

{para $l = 0$ até $n_p[g]$

{i=p[g,l]

para $k = 0$ até n_{g_v}

para $m = 0$ até $n_p[g_v[g,k]]$

$\{j = p[g_v[g,k], m]$

calcule distância entre i e j (r_{ij}^2)

se $r_{ij}^2 < r_c^2$

{calcule potencial entre i e j

calcule força entre i e j

acumule potencial em potencial total

acumule força em força sobre i } }

calcule nova posição de i e atualize

calcule velocidade de i

calcule energia cinética de i

acumule em energia cinética total }

para $i = 0$ até $N - 1$

{encontre novo grão de i

coloque i nesse grão

atualize contador de partículas no grão }

se q é múltiplo de q_r, faça renormalização }

Figura 10: Pseudo-código do algoritmo de granulação grosseira

1.6 Conclusão

Tratando-se de um problema intensivo em tempo de computação, a *dinâmica molecular* requer um estudo cuidadoso para sua implementação, de forma a que se possam utilizar os recursos computacionais disponíveis de uma forma efetiva. Os quatro algoritmos apresentados acima serão utilizados no estudo a ser realizado no capítulo seguinte. Deve-se notar que, na forma em que foram apresentados, esses algoritmos estão baseados numa execução puramente seqüencial. Isto não significa que os mesmos não possam ser utilizados para uma implementação paralela. Estudos relativos a isto serão realizados no capítulo 3.

Capítulo 2

Eficiência dos algoritmos seqüenciais

2.1 Introdução

Este capítulo trata de um aspecto extremamente importante no desenvolvimento de um sistema computacional para um dado problema, que é o estudo da complexidade do mesmo, em relação a tempo de execução. Este estudo será realizado para os quatro algoritmos apresentados no capítulo 1, de forma a que seja possível uma comparação entre os mesmos neste aspecto particular.

Deve-se enfatizar que nos restringiremos, neste capítulo à consideração da execução seqüencial dos algoritmos apresentados, sendo que as possibilidades de paralelismo dos mesmos serão estudadas no capítulo 3.

2.2 Tempos de execução

Os cálculos de tempo de execução a serem realizados têm o único objetivo de apresentar uma comparação da eficiência dos diversos algoritmos, e não apresentar o tempo preciso de execução dos mesmos em uma dada máquina. Tendo isto em vista, procedemos a uma série de simplificações em fatores que não alterarão significativamente as relações entre os tempos de execução de nenhum algoritmo. Citaremos agora as simplificações a serem

realizadas:

1. desprezaremos os tempos de administração dos *loops* de repetição. Isto inclui tempo de inicialização da variável de *loop*, tempo de atualização da mesma e tempo de teste da condição de parada;
2. desprezaremos tempos de teste de variáveis inteiras;
3. desprezaremos todos os tempos de inicialização;
4. desprezaremos os tempos gastos no controle do fluxo do programa (saltos e chamadas de subrotinas, bem como inicialização e finalização de processos);

Para o cálculo dos tempos de execução, basearemo-nos em valores dados para o *transputer* T800 da INMOS, conforme apresentados em [12] e [13]. Note-se que incluiremos nos cálculos apenas os tempos gastos nas operações aritméticas envolvidas.

Os tempos do *transputer* são calculados como se tanto o programa como os dados estivessem totalmente contidos na memória interna do mesmo (para maiores explicações sobre o T800, veja capítulo 4).

Antes de calcular os tempos de execução dos algoritmos, vamos determinar o tempo despendido nas renormalizações. A renormalização, conforme definida no pseudo-código da fig. 4, ocupa o seguinte tempo:

$$t_r = 2Nt_{S_v} + 2t_{\bar{v}} + t_f + N(2t_{v_r} + 2t_{y_r}) \tag{29}$$

onde:

t_r é o tempo da renormalização

t_{S_v} é o tempo de soma de uma velocidade no totalizador de velocidades

$t_{\bar{v}}$ é o tempo para o cálculo de média da velocidade

t_f é o tempo do cálculo do fator de renormalização

t_{v_r} é o tempo de cálculo da nova velocidade renormalizada

t_{p_r} é o tempo de cálculo da nova posição a partir da velocidade renormalizada

Podemos avaliar esses tempos da forma seguinte (ao lado do valor de tempo apresentamos o trecho de programa OCCAM correspondente ao mesmo):

t_{S_v} corresponde ao tempo de acesso em um vetor de velocidades e de acumulação no totalizador:

```
vt := vt + v[i]
```

$t_{S_v} = 1,4\mu s$

$t_{\overline{v}}$ cooresponde ao tempo de uma divisão do totalizador pelo número de partículas:

```
vm := vt/N
```

$t_{\overline{v}} = 1,9\mu s$

t_f corresponde a uma multiplicação, uma divisão e uma raiz quadrada:

```
vf := sqrt(Tr*qr/Scin)
```

$t_f = 6,0\mu s$

t_{v_r} uma leitura em vetor, uma subtração, uma multiplicação e uma escrita em vetor:

```
v[i] := vf*(v[i]-vm)
```

$t_{v_r} = 2,7\mu s$

t_{p_r} três acessos em vetor, duas multiplicações e duas somas (note que o fator $\frac{1}{2}\Delta t^2$ pode ter seu valor resultante utilizado, pois é constante durante toda a simulação):

```
x[i]:=x[i]+vx[i]*dt+ax[i]*mdt2
```

$t_{p_r} = 4,9\mu s$

Com esses fatores, podemos calcular t_r como segue:

$$\begin{aligned} t_r &= N2 \cdot 1,4 + 2 \cdot 1,9 + 6,0 + N(2 \cdot 2,7 + 2 \cdot 4,9) \\ t_r &= 18,0N + 9,8 \end{aligned} \tag{30}$$

Vejamos agora os tempos de execução dos diversos algorítmos.

2.2.1 Tempo de execução do algoritmo básico

Este algoritmo consiste basicamente de três ciclos iterativos "aninhados". O mais externo desse ciclos representa a repetição dos cálculos pelos diversos quadros da simulação. O ciclo intermediário representa a repetição dos cálculos para as diversas partículas do agregado. O ciclo mais interno é o que faz com que seja realizado o cálculo da interação da partícula com todas as outras do agregado.

Considerando o pseudo-código da fig. 3 (cap. 1), podemos escrever:

$$t_B = N_q(N(t_{b_B} + (N-1)(t_{b_B} + t_d + t_a + t_c + t_{S_a}) + t_p + t_{E_c} + 2t_m) + \frac{t_r}{q_r}) \qquad (31)$$

onde:

t_B é o tempo total da simulação do algoritmo básico

N_q é o número total de quadros da simulação

N é o número total de partículas do agregado

t_{b_B} é o tempo necessário para se acessar as coordenadas de uma partícula

t_d é o tempo necessário para o cálculo da distância entre as duas partículas

t_a é o tempo necessário para o cálculo do potencial interpartículas, conhecida a distância entre as mesmas

t_c é o tempo necessário ao cálculo dos vários componentes da aceleração em uma partícula devida a outra

t_{S_a} é o tempo necessário à acumulação dos componentes da aceleração sobre o totalizador de aceleração sobre partícula

t_p é o tempo utilizado para o cálculo da nova posição da partícula (isto é, a que ela ocupará no próximo quadro)

t_{E_c} é o tempo necessário para o cálculo da energia cinética da partícula em questão e sua acumulação num totalizador de energia cinética do agregado

t_m é o tempo de movimentação dos dados (para a atualização de coordenadas temdo em vista o novo quadro a ser calculado)

q_r é o número de quadros entre renormalizações

Realizaremos a seguir uma avaliação dos diversos tempos envolvidos de forma a possibilitar uma avaliação aproximada do tempo total de simulação desse algoritmo, para comparação do mesmo com os outros apresentados.

N_q suporemos a necessidade de realização de 1000 quadros de simulação. Note que o tempo total de simulação é linear com o número de quadros;

N deixaremos como variável o número total de partículas, para analisar a influência do mesmo no tempo de simulação;

t_{ba} suporemos que as coordenadas das partículas serão armazenadas em dois vetores associados, sendo o índice do vetor um número identificador da partícula (isto é, $x[i]$ e $y[i]$ são as coordenadas x e y da partícula i). Desta forma, t_{ap} corresponde ao tempo desses dois acessos. Portanto:

xi := x[i]

yi := y[i]

ou

$t_{ba} = 1,8\mu s$

xj := x[j]

yj := y[j]

t_d o cálculo da distância entre os mesmos precisa ser feito apenas quadraticamente, de acordo com (2). Temos então:

$t_d = 4,0\mu s$

```
xij := xi - xj
yij := yi - yj
rij2 := xij*xij + yij*yij
```

t_u este cálculo é determinado por (1), somente simplificado pelo fato de que nas simulação trabalhamos com unidades de distância normalizadas em relação a σ e unidades de energia normalizadas em relação a ε. Portanto, o tempo necessário é:

$t_u = 6,4\mu s$

```
inrij2 := 1/rij2
inrij6 := inrij2*inrij2*inrij2
uij := inrij6*(inrij6-1.)
U := U + uij
```

t_a o cálculo da aceleração, conforme indicado em (5), pode ser executado em:

$t_a = 6,2\mu s$

```
a := 48.0*inrij6*(inrij6-0.5)*inrij2
ax := a*xij
ay := a*yij
```

t_{S_a} corresponde a uma soma simples:

$t_{S_a} = 3,8\mu s$

```
rax[i] := rax[i]+ax
ray[i] := ray[i]+ay
```

t_p Este cálculo corresponde às fórmulas (10) e (11), ocupando:

$t_p = 11,4\mu s$

```
xn[i] := 2*x[i]-xa[i]+rax[i]*dt2
yn[i] := 2*y[i]-ya[i]+ray[i]*dt2
```

t_{E_c} corresponde ao cálculo das componentes da velocidade e à soma de seus quadrados:

$t_{E_c} = 3,7\mu s$

```
Ec := Ec+vx[i]*vx[i]+vy[i]*vy[i]
```

t_m corresponde a duas leituras em vetor seguidas de duas escritas em vetor. Portanto temos:

$t_m = 2,7\mu s$

```
x[i] := xn[i]
y[i] := yn[i]
```

Tabela I: Tempo de execução do algoritmo básico

No. Partículas	t_B (em horas)
100	0,06
200	0,25
500	1,54
1000	6,17
2000	24,67
3000	55,50
4000	98,67
5000	154,17
10000	616,68
20000	2466,69
50000	15416,72
100000	61666,77

q_r conforme utilizado em [14], faremos:

$$q_r = 5$$

Com estes valores, podemos encontrar a relação entre o número de partículas e o tempo total de simulação (em microssegundos):

$$\begin{aligned} t_B &= 1000(N(1,8+(N-1)(1,8+4,0+6,4+6,2+3,8)+11,4+ \\ &\quad 3,7+2\cdot 2,7)+\frac{18,0N+9,8}{5}) \\ t_B &= 22200N^2+3700N+1960 \end{aligned} \tag{32}$$

Os tempos de simulação para alguns valores de número de partículas N são apresentados na tabela I.

Para uma melhor visualização do crescimento do tempo de simulação com o

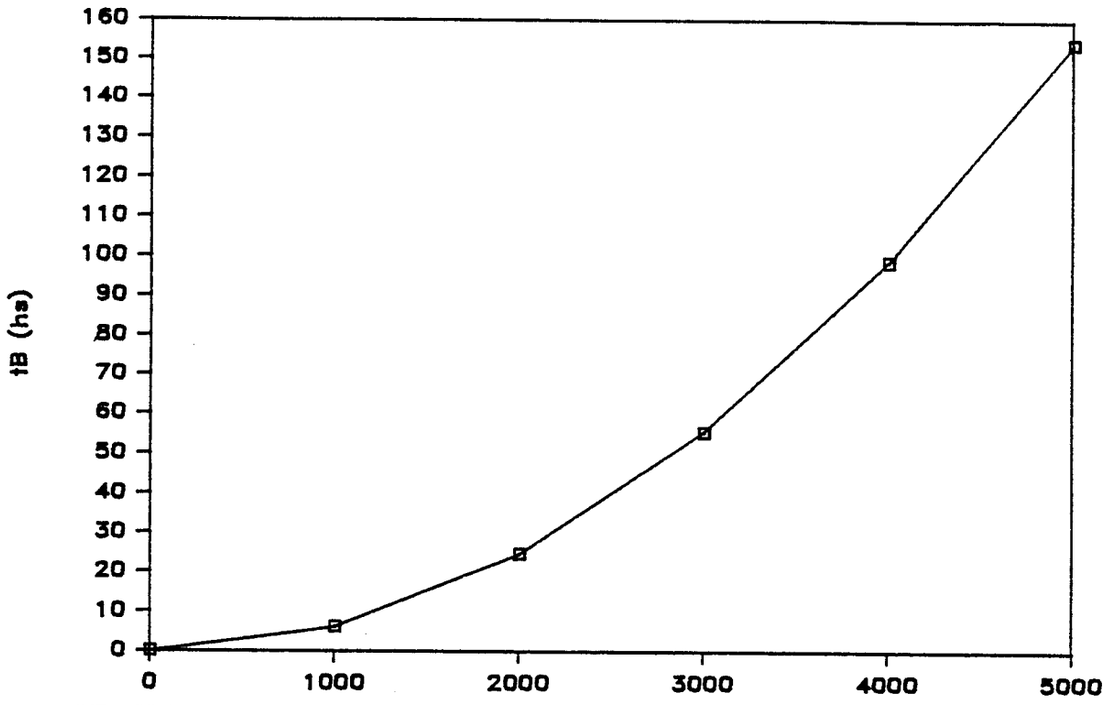


Figura 11: Evolução do tempo de execução com o número de partículas para o algoritmo básico

número de parículas apresentamos o mesmo em forma gráfica, na fig. 11.

2.2.2 Tempo de execução do algoritmo básico modificado

Para o cálculo desse tempo nos basearemos no pseudo-código da fig. 5 e realizaremos as mesmas considerações de simplificação do algoritmo anterior. Veja que o algoritmo consiste de três ciclos "aninhados", correspondendo aos mesmos três ciclos do algoritmo anterior. No entanto, neste caso, os dois ciclos mais internos são alterados, da seguinte forma:

1. o ciclo que executa a repetição dos cálculos para todas as partículas é realizado $N-1$ vezes ao invés de N vezes;

2. o ciclo que varre os pares de partículas percorre apenas os pares que não foram considerados para cada partícula (isto é, procura novos pares apenas entre as partículas seguintes, pois todos os pares de uma dada partícula com suas anteriores já foram considerados no cálculo das anteriores).

Outra diferença é o fato de que o cálculo da interação entre as duas partículas é executado apenas se as mesmas se encontram a uma distância inferior ao raio de corte.

Para considerar essas modificações na expressão do tempo total, devemos incluir alguns novos parâmetros, em relação aos já apresentados anteriormente.

t_t é o tempo de teste do raio, corresponde ao tempo necessário para se verificar se a distância entre as duas partículas é menor que o raio de influência;

$\overline{n_i}$ é o número médio de partículas interagentes com uma dada partícula.

Com esses novos parâmetros podemos escrever o tempo total de simulação deste algoritmo como:

$$t_M = N_s(N(t_{b_x} + \frac{N-1}{2}(t_{b_x} + t_d + t_t) + \frac{\overline{n_i}}{2}(t_s + t_a + 2t_{S_a}) + t_p + t_{E_c} + 2t_m) + \frac{t_r}{q_r}) \quad (33)$$

Para o levantamento dessa fórmula foram considerados os seguintes fatos:

1. o cálculo de distância entre as partículas e o teste da mesma deve ser realizado para todos os pares de partículas que, para N partículas, totalizam $N(N-1)/2$;

2. o cálculo de interação entre as partículas deve ser realizado apenas para os pares de partículas cuja distância seja menor que o raio de corte. Como para cada partícula, o número médio de partículas que com ela interage é $\overline{n_i}$, então o total de pares interagentes será (em média) $N\overline{n_i}/2$.

Avaliaremos agora os novos parâmetros incluídos neste algoritmo, de modo a podermos realizar o cálculo do tempo de simulação do mesmo.

t_t consiste de um teste:

rij2 < rc2

$t_t = 0,7\mu s$

$\overline{n_i}$ Este valor pode ser avaliado se consideramos uma distribuição uniforme de partículas com a densidade reduzida especificada ϱ, o raio de corte (em coordenadas reduzidas) r_c, da seguinte forma:

$$\overline{n_i} = \varrho \pi r_c^2$$

o que, com $\varrho = 0,6$ e $r_c = 2,5$[1] nos dá:

$$\overline{n_i} = 11,8 \text{ partículas}$$

Podemos então encontrar a relação entre o tempo total de simulação desse algoritmo e o número de partículas:

$$\begin{aligned} t_M &= 1000(N(1,8 + \frac{N-1}{2}(1,8+4,0+0,7) + \frac{11,8}{2}(6,4+6,2+2\cdot 3,8) + \\ &\quad 11,4+3,7+2\cdot 2,7) + \frac{18,0N+9.8}{5}) \\ t_M &= 3250N^2 + 141830N + 1960 \end{aligned} \tag{34}$$

Na tabela II apresentamos o tempo de simulação do algoritmo básico modificado para alguns valores de número de partículas. Na fig. 12 apresentamos a mesma relação em forma de gráfico.

Uma comparação entre os tempos de simulação dos dois algoritmos apresentados será dada adiante, após apresentados os tempos de simulação para os dois outros algoritmos restantes.

2.2.3 Tempo de execução do algoritmo de tabela de vizinhos com atualização infreqüente

Para o cálculo do tempo de execução do algoritmo de tabela de vizinhos, considerando-se as simplificações já utilizadas nos cálculos relativos aos algorítmos anteriores, devemos

[1] Valores utilizados numa simulação realizada por Kalia[14]

Tabela II: Tempo de execução do algoritmo básico modificado

No. Partículas	t_M (em horas)
100	0,013
200	0,044
500	0,245
1000	0,942
2000	3,690
3000	8,243
4000	14,602
5000	22,766
10000	90,672
20000	361,899
50000	2258,914
100000	9031,718

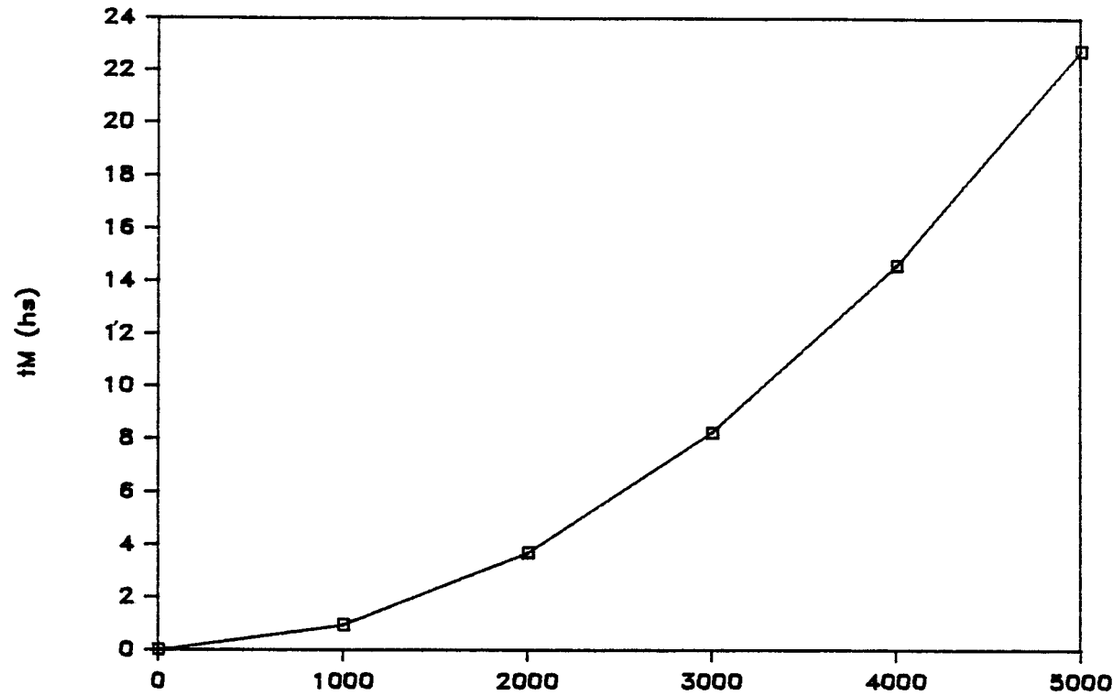


Figura 12: Evolução do tempo de execução do algoritmo básico modificado com o número de partículas

introduzir alguns fatores próprios deste algorítmo, que podem ser inferidos por uma análise do pseudo-código da fig. 7

Note que este algoritmo consiste de um ciclo mais externo (que corresponde à repetição dos cálculos para os quadros sucessivos) subdividido em dois ciclos. O primeiro desses ciclos se encarrega da formação de uma tabela de vizinhos para cada partícula. O segundo ciclo realiza o cálculo da nova posição de cada partícula a partir da interação da mesma com todas as partículas presentes em sua tabela de vizinhos.

Note também que o ciclo de atualização das posições das partículas deve ser realizado em todos os quadros, enquanto que o ciclo de formação de tabelas de vizinhos é executado, devido à utilização da técnica de atualização infreqüente, apenas em uma certa porcentagem dos mesmos.

Introduziremos então os seguintes fatores:

t_j é o tempo de acesso (busca ou escrita) na tabela de vizinhos da partícula i de uma partícula j sua vizinha

t_n é o tempo de incremento do número de vizinhos de uma partícula

q_v é o número de quadros entre atualizações da tabela de vizinhos

n_v é o número médio de partículas j presentes na tabela de vizinhos de uma partícula i

Avaliamos estes fatores como segue:

t_j Suporemos que as tabelas de vizinhos das partículas estão armazenadas em um vetor de vetores, sendo que cada um dos vetores menores armazena a tabela de vizinhos de uma dada partícula. Neste caso temos:

```
j:=v[i][k]
```

$t_j = 2,7\mu s$

t_n Corresponde a um acesso de leitura de vetor, uma soma e uma escrita em vetor:

```
nv[i] := nv[i]+1
```

$t_n = 1,3\mu s$

q_v Utilizando a mesma razão de atualização da simulação realizada por Kalia[14], que consiste na atualização das tabelas de vizinho a cada 15 quadros, temos:

$$q_v = 15$$

$\overline{n_v}$ Podemos encontrar o número médio de partículas na tabela de vizinhos de uma dada partícula a partir do conhecimento do raio de vizinhança (r_v) e da densidade reduzida de partículas (ϱ):

$$\overline{n_v} = \frac{\varrho \pi r_v^2}{2}$$

obs: o valor é dividido por dois pois consideramos apenas uma vez cada par de partículas.

Considerando $\varrho = 0,6$ e $r_v = 3,2$ temos

$$\overline{n_v} = 9,7 \text{ partículas}$$

Com a inclusão desses fatores, e baseando-nos no pseudo-código da fig. 7, podemos calcular o tempo de simulação empregado pelo algoritmo de tabela de vizinhos como:

$$t_V = N_t(N(\frac{1}{q_v}(\frac{N-1}{2}(t_d+t_t)+t_{b_s}+\overline{n_v}(t_j+t_a))+\overline{n_v}(t_j+t_{b_s}+t_d+t_t)+ \frac{\overline{n_i}}{2}(t_a+t_a+2t_{S_a})+t_p+t_{E_a}+2t_m)+\frac{t_r}{q_r}) \quad (35)$$

Substituindo os valores dos parâmetros na equação (33), podemos encontrar a relação entre o tempo de simulação do algoritmo de tabela de vizinhos e o número de partículas:

$$\begin{aligned} t_V &= 1000(N(\frac{1}{15}(\frac{N-1}{2}(4,0+0,7)+1,8+9,7(2,7+1,3))+ \\ &\quad 9,7(2,7+1,8+4,0+0,7)+\frac{11,8}{2}(6,4+6,2+23,8)+ \\ &\quad 11,4+3,7+22,7)+\frac{18,0N+9,8}{5}) \\ t_V &= 160N^2+235060N+1960 \end{aligned} \quad (36)$$

A apresentação da relação entre número de partículas e tempo de simulação é dada em forma tabular na tabela III e em forma gráfica na fig. 13.

2.2.4 Tempo de execução do algoritmo de granulação grosseira

O cálculo do tempo de execução do algoritmo de granulação grosseira, conforme definido no pseudo-código da fig. 10, envolve a definição de um novo conjunto de parâmetros, que citaremos a seguir:

t_g é o tempo necessário para se encontrar o grão ao qual a partícula pertence

t_{b_g} é o tempo de busca ou armazenamento de uma partícula em um dado grão

t_{n_p} é o tempo necessário para se encontrar o número de partículas de um dado grão

t_{g_v} é o tempo para encontrar grão vizinho a um dado grão

$\overline{n_g}$ é o número médio de partículas dentro de um grão

Tabela III: Tempo de execução do algoritmo de tabela de vizinhos com atualização infreqüente

No. Partículas	t_V (em horas)
100	0,0070
200	0,0148
500	0,0438
1000	0,1097
2000	0,3084
3000	0,5959
4000	0,9723
5000	1,4376
10000	5,0974
20000	19,0837
50000	114,3758
100000	450,9739

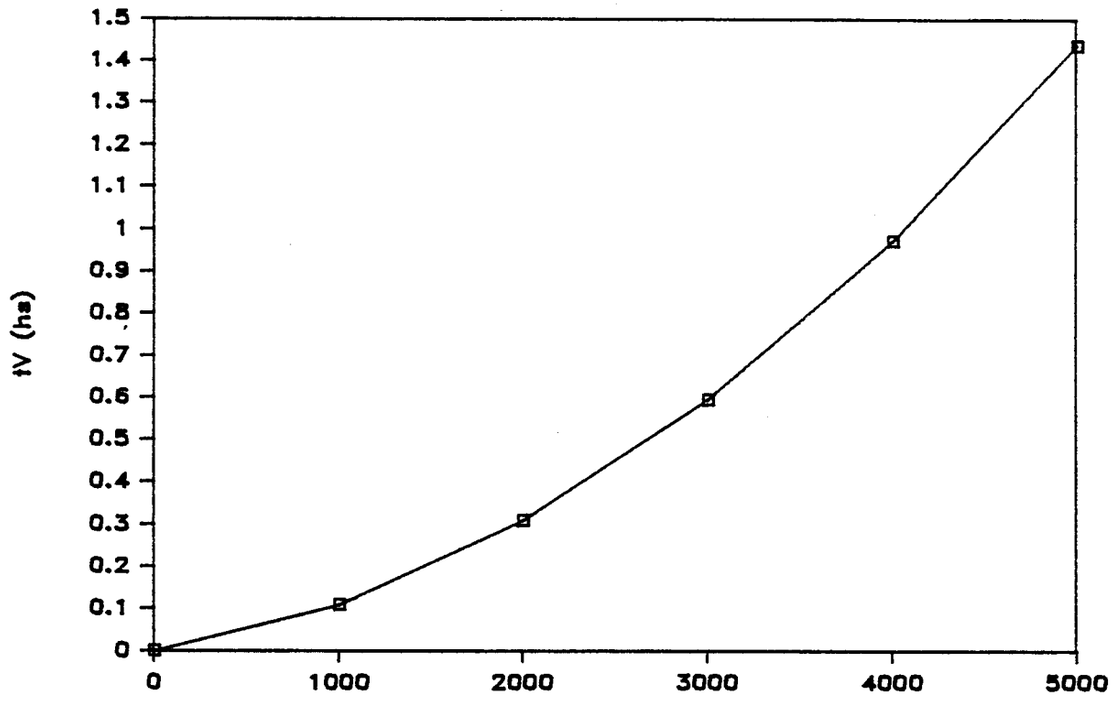


Figura 13: Evolução do tempo de execução com o número de partículas para o algoritmo de tabela de vizinhos

t_{i_n} é o tempo necessário para incrementar o número de partículas de um dado grão

n_{g_v} é o número de grãos vizinhos a um dado grão

Os valores desses parâmetros podem ser avaliados como segue:

t_g : Para se encontrar o grão ao qual uma dada partícula pertence, podemos proceder a cálculos simples sobre as coordenadas da mesma (como será mostrado em maiores detalhes na seqüência deste trabalho), que compreendem uma divisão inteira das coordenadas da partícula pelo número total de grãos. Além disso devemos contar o tempo de cálculo da indexação do vetor e armazenamento da partícula no grão. Portanto podemos estimar:

$$t_g = 9,0\mu s$$

```
gx := x[i]/ngx
gy := y[i]/ngy
g := gy*ngx+gx
```

t_{b_g} : O tempo de acesso de grão corresponde ao tempo necessário para se encontrar uma dada partícula em um dado grão. Isto pode ser implementado com uma matriz de

duas dimensões, o que fornece:

$$\text{i := p[g][k]}$$
$$t_{b_g} = 2,7\mu s$$

t_{n_p} : O número de partículas em um dado grão é determinado por um acesso em vetor:

$$\text{acesso a np[g]}$$
$$t_{n_p} = 0,6\mu s$$

t_{g_v} : O acesso de grão vizinho, que corresponde a encontrar um vizinho de um determinado grão, pode ser realizado através de uma matriz de dimensão 2. Portanto:

$$\text{k := gv[g][l]}$$
$$t_{g_v} = 2,7\mu s$$

$\overline{n_g}$: O número médio de partículas por grão pode ser estimado considerando-se a densidade média de partículas, bem como o tamanho por grão, além das seguintes hipóteses:

- distribuição espacial homogênea das partículas
- tamanho do grão exatamente igual ao raio de corte

Desta forma encontramos:

$$\overline{n_g} = \varrho r_c^2$$

ou:

$$\overline{n_g} = 3,8 \text{ partículas}$$

t_{i_n} : Corresponde a um acesso de leitura e um de escrita em um vetor e mais uma soma com constante:

$$\text{np[g] := np[g]+1}$$
$$t_{i_n} = 1,3\mu s$$

n_{g_v} : Para o caso bidimensional temos

$$n_{g_v} = 9 \text{ grãos}$$

considerando um grão como vizinho dele mesmo

Juntando-se esses novos parâmetros aos já anteriormente definidos podemos, seguindo as mesmas simplificações utilizadas para os algoritmos anteriores, definir, para o pseudo-código do algoritmo de *granulação grosseira*, a relação entre o número de partículas e o tempo de simulação do mesmo.

Note que, de forma semelhante a todos os algoritmos anteriores, este também apresenta um ciclo mais externo que corresponde à repetição do processo pelos diversos quadros. Este ciclo é por sua vez dividido em dois outros, sendo o primeiro um ciclo simples em que se realiza a distribuição das partículas pelos respectivos grãos. O segundo ciclo corresponde à repetição dos cálculos pelas diversas partículas, só que realizando a varredura das partículas por uma forma diferente das anteriores, que passaremos a descrever.

As partículas são varridas de acordo com a sua distribuição em grãos, isto é, são varridas as partículas de um grão, seguidas pelas de outro grão, e assim sucessivamente, numa ordem pré-estabelecida para os grãos.

Para se encontrar as partículas interagentes com as de um determinado grão são varridos apenas os grãos vizinhos deste.

O ciclo de varredura das partículas corresponde portanto a (acompanhe pelo pseudo-código da fig. 10):

1. um ciclo externo que pega sucessivamente os diversos grãos
2. um ciclo mais interno que varre todas as partículas do grão escolhido
3. um terceiro ciclo que varre todos os grao vizinhos ao escolhido
4. um último ciclo que varre todas as partículas do determinado grão vizinho

Disto podemos extrair a seguinte relação entre o tempo de simulação deste algoritmo e o número de partículas:

$$t_G = 1000(N_g(\overline{n_g}(t_{b_C} + t_{b_B} + n_{g_v}\overline{n_g}(t_{b_C} + t_{b_B} + t_r + t_t) + \frac{\overline{n_i}}{2}(t_a + t_a + t_{S_a}) + t_p + t_{E_c})) + N(t_g + t_{b_C} + t_{i_n}) + \frac{t_r}{q_r}) \quad (37)$$

Tabela IV: Tempo de execução do algoritmo de granulação grosseira

No. Partículas	t_G (em horas)
100	0,01243
200	0,02487
500	0,06217
1000	0,12434
2000	0,24867
3000	0,37300
4000	0,49733
5000	0,62167
10000	1,24333
20000	2,48667
50000	6,21667
100000	12,43333

Considerando-se (37), os valores de parâmetros encontrados acima, e o fato de que quando multiplicamos o número de grãos N_g pelo número médio de partículas por grão $\overline{n_g}$ encontramos o número total de partículas N, podemos escrever:

$$\begin{aligned} t_G &= 1000\Big(\frac{N}{3,8}(3,8(2,7+1,8+9\cdot 3,8(2,7+1,8+4,0+0,7)+ \\ &\quad \frac{11,8}{2}(6,4+6,2+3,8)+11,4+3,7))+N(9,0+2,7+1,3)+ \\ &\quad \frac{18,0N+9,8}{5}\Big) \\ t_G &= 447600N+1960 \end{aligned} \tag{38}$$

A partir de (38) construímos a tabela IV e o gráfico da fig. 14 abaixo, que apresentam a relação entre o tempo de simulação do algoritmo de *granulação grosseira* e o número de partículas.

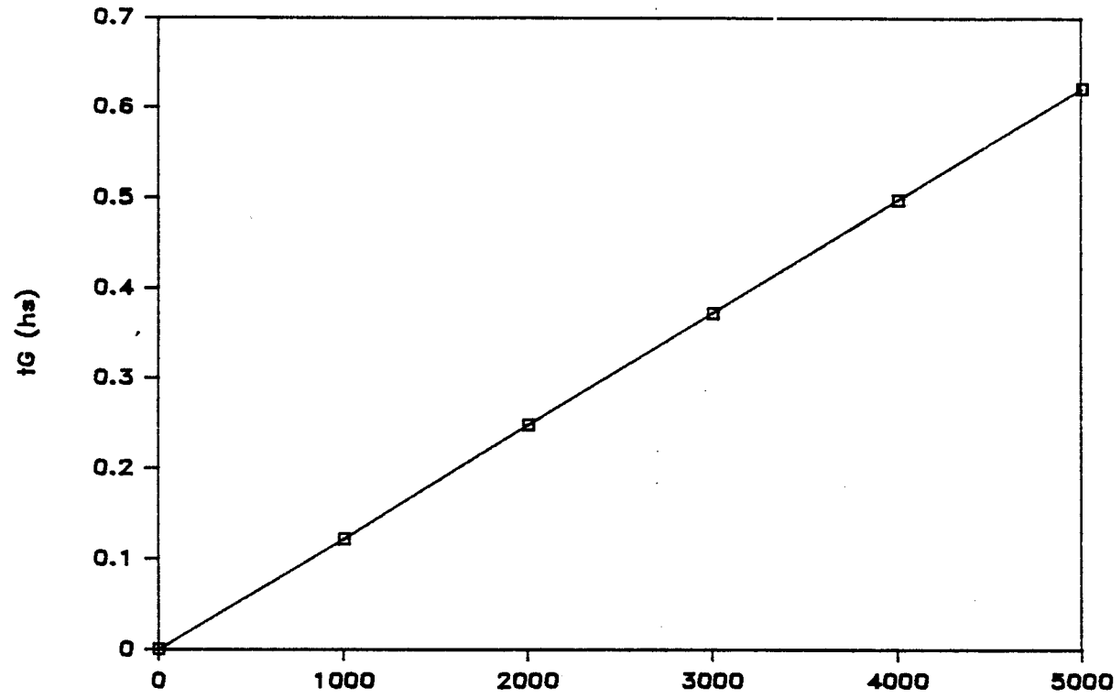


Figura 14: Evolução do tempo de execução com o número de partículas para o algoritmo de granulação grosseira

2.3 Comparação dos algoritmos

Os dados de tempo de execução apresentados no ítem anterior permitem a realização de uma comparação dos algoritmos acima em termos de execução seqüencial.

Para facilitar o processo de comparação, foi organizada uma tabela com os tempos de execuçao de todos os algoritmos aqui discutidos (tabela V), bem como três gráficos comparativos da relação entre tempo de execução e número de partículas, sendo o primeiro gráfico (fig. 15) comparativo dos algorítmos básico e básico modificado, o segundo gráfico (fig. 16) comparativo dos algorítmos basico modificado e tabela de vizinhos com atualização infreqüente e o terceiro gráfico (fig. 17) comparativo dos algoritmos de tabela de vizinhos com atualização infreqüente e de granulação grosseira.

Podemos verificar através do gráfico da fig. 15, que a diminuição do tempo de execução com a inclusão do teste da distância com o raio crítico representa um grande incremento na eficiência na execução do algorítmo, para todos os casos práticos de número

Tabela V: Confrontamento dos tempos de execução dos diversos algoritmos

No. Partículas	t_B	t_M	t_V	t_G
100	0,06	0,013	0,0070	0,01243
200	0,25	0,044	0,0148	0,02487
500	1,54	0,245	0,0438	0,06217
1000	6,17	0,942	0,1097	0,12434
2000	24,67	3,690	0,3084	0,24867
3000	55,50	8,243	0,5959	0,37300
4000	98,67	14,602	0,9723	0,49733
5000	154,17	22,766	1,4376	0,62167
10000	616,68	90,672	5,0974	1,24333
20000	2466,69	361,899	19,0837	2,48667
50000	15416,72	2258,914	114,3758	6,21667
100000	61666,77	9031,718	450,9739	12,43333

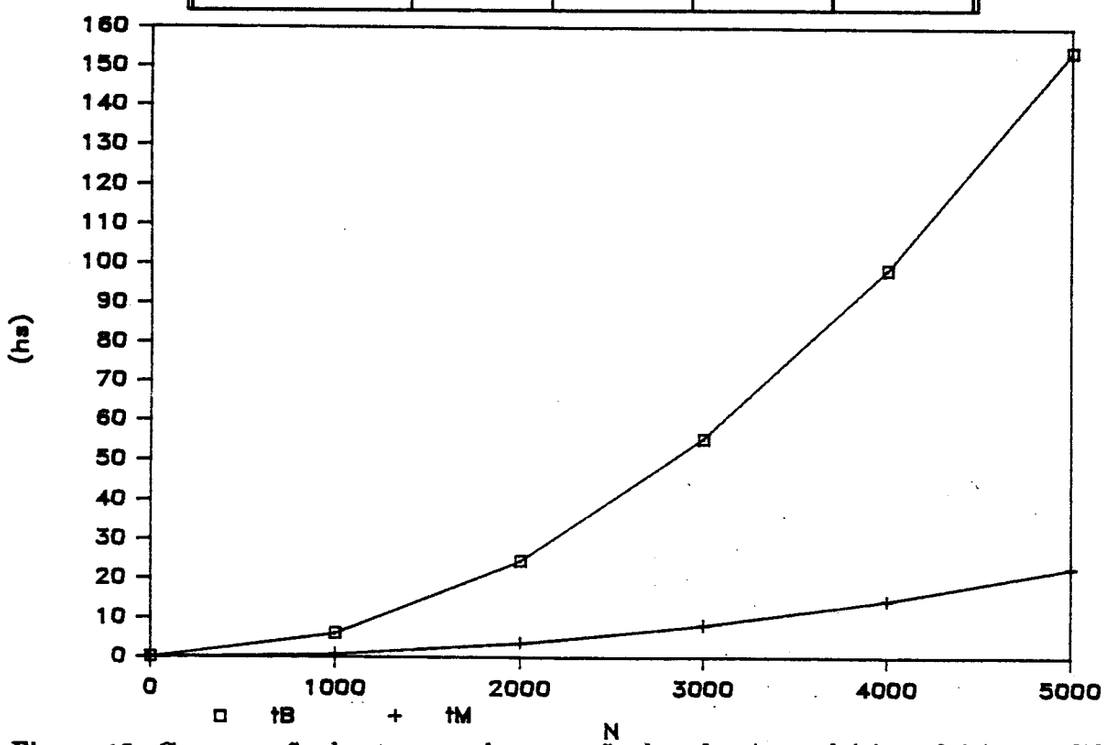


Figura 15: Comparação dos tempos de execução dos algoritmos básico e básico modificado

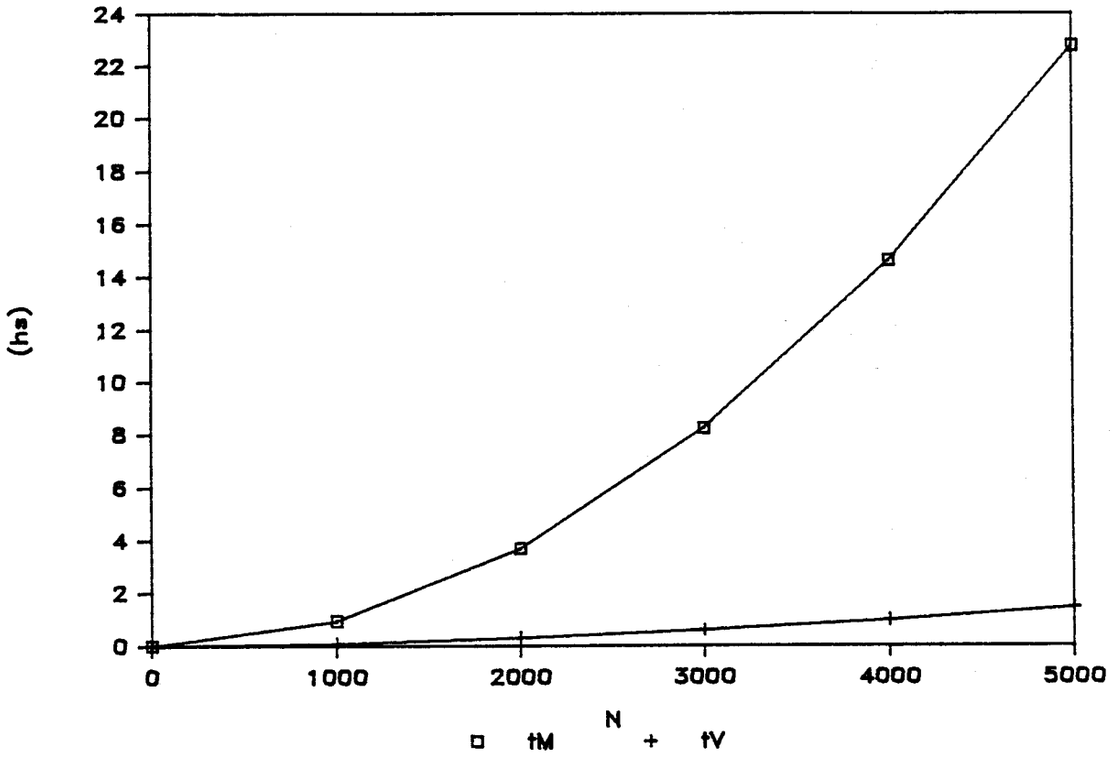


Figura 16: Comparação dos tempos de execução dos algoritmos básico modificado e tabela de vizinhos com atualização infreqüente

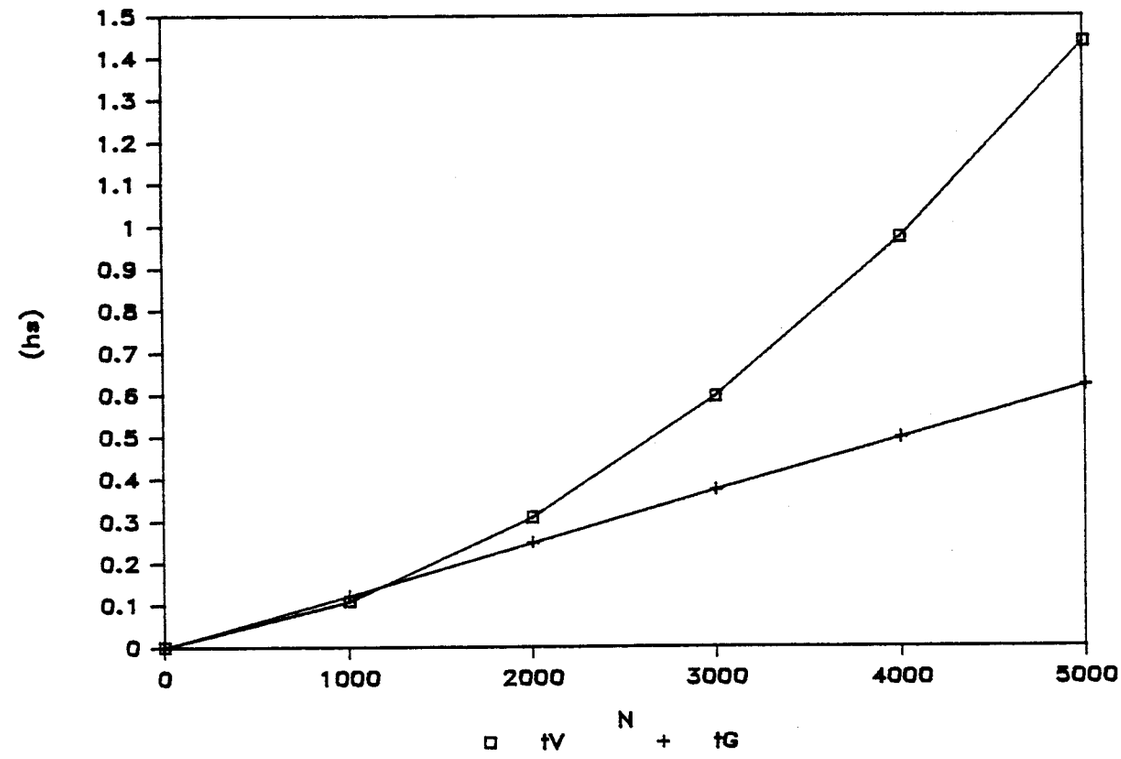


Figura 17: Comparaçao dos tempos de execução dos algoritmos de tabela de vizinhos com atualização infreqüente e de granulação grosseira

de partículas. O número exato de partículas a partir do qual é vantajosa a inclusão desse teste pode ser calculado exatamente encontrando-se o valor de N para o qual:

$$t_B > t_M$$

Utilizando-se as expressões (32) e (34) podemos encontrar:

$$N_1 = 8 \text{ partículas}$$

Este valor demonstra o que foi afirmado acima a partir da análise da fig. 15 e da tabela V, pois qualquer simulação que pretenda um mínimo de realismo dever ser feita com uma quantidade significativamente superior a 8 partículas.

A vantagem do algorítmo de tabela de vizinhos com atualização infreqüente pode ser apreciada através do gráfico da fig. 16, onde fica claro que este algorítmo é bastante superior ao básico modificado. Esta diferença é cada vez mais acentuada ao aumentarmos a quantidade de partículas do sistema (isto é, ao ganharmos realismo na simulação), mas ela se apresenta mesmo para quantidades pequenas de partículas. O número de partículas exato a partir do qual o algoritmo de tabela de vizinhos é superior ao básico modificado pode ser encontrado através de:

$$t_M > t_V$$

Utilizando as expressões (34) e (36) encontramos:

$$N_2 = 31 \text{ partículas}$$

Este número de partículas também representa uma simulação extremamente pobre, o que indica que o algorítmo de tabela de vizinhos com atualização infreqüente é superior aos outros dois anteriores para qualquer efeito prático.

Uma situação mais interessante surge na comparação dos algorítmos de tabela de vizinhos com atualização infreqüente e o de granulação grosseira. Podemos ver, na fig. 17, que o primeiro apresenta um melhor tempo de execuçao para sistemas com até

por volta de 2000 partículas. O valor exato de número de partículas a partir do qual o algorítmo de granulação grosseira é superior pode ser calculado por:

$$t_V > t_G$$

Substituindo as expressões (36) e (38) encontramos:

$$N_s = 1329 \text{ partículas}$$

Este número de partículas já representa um sistema bastante apropriado para a simulação, o que nos coloca frente ao problema de decidir qual dos dois algorítmos utilizar para uma dada simulação. Este problema fica ainda mais claro quando pensamos na implementação de processamento paralelo para o problema, pois neste caso podemos dividir um problema com grande número de partículas em diversos subproblemas com quantidades menores de partículas, portanto a escolha entre um dos dois algorítmos para implementação dada deve ser bem pensada.

Conforme fica claro nos dados apresentados acima, a escolha do algoritmo para o problema de dinâmica molecular tem profunda importância, com alterações de eficiência altamente significativas, principalmente quando consideramos problemas com quantidades elevadas de partículas.

A análise de eficiência seqüencial dos algoritmos apresentada neste capítulo é, entretanto, apenas um dos aspectos de eficiência, quando tratamos da implementaçao da simulação em um sistema paralelo. O próximo capítulo tratará dos aspectos de paralelismo.

Capítulo 3

Paralelismo

3.1 Introdução

Nos capítulos anteriores, apresentamos o problema de simulação da dinâmica molecular em termos de quatro algoritmos, sendo as análises realizadas considerando-se a execução puramente seqüencial.

Neste capítulo, iniciaremos a apresentação dos aspectos da simulação de dinâmica molecular relevantes à implementação da mesma em um sistema de computação paralela.

A implementação paralela de um algoritmo é fortemente dependente da forma de paralelismo existente na máquina a ser utilizada, conforme pode ser visto, por exemplo, em [15]. No entanto, neste capítulo não nos ocuparemos com nenhuma arquitetura paralela específica, focalizando apenas os paralelismos dos diversos algoritmos que poderão ser explorados conforme as possibilidades de cada implementação específica.

Começaremos com a análise dos paralelismos inerentes ao próprio problema, seguindo com a de cada um dos algoritmos apresentados.

Visando uma maior clareza na exposição, introduzimos, no apêndice A, uma simbologia que nos permitirá apresentar de forma diagramática as relações entre os diversos processos distintos que formam um algoritmo. Passamos então à análise dos paralelismos com a utilização desta simbologia.

3.2 Paralelismos inerentes ao problema

Antes do estudo dos paralelismos, apresentaremos a descrição dos processos fundamentais que serão utilizados para a representação dos cálculos executados em dinâmica molecular. Estes processos são os seguintes (utilizaremos apenas a coordenada x para representar todas as coordenadas do sistema):

1. Cálculo da distância entre duas partículas

   Nome do processo: r_{ij}^2

   Dados de entrada: x_i e x_j (coordenadas das partículas i e j)

   Dados de saída: r_{ij}^2 (quadrado da distância entre as partículas) e $(x_i - x_j)$

   Entradas de sincronização: indicação de que um novo quadro pode começar (através do aviso da disponibilidade de dados para um novo cálculo)

   Saídas de sincronização: nenhuma

   Descrição do processamento: calcula os dados de saída

   Multiplicidade: $(N-1)$ para cada partícula

2. Cálculo do potencial interpartículas

   Nome do processo: u_{ij}

   Dados de entrada: r_{ij}^2

   Dados de saída: u_{ij} (potencial entre as partículas)

   Entradas de sincronização: nenhuma

   Saídas de sincronização: nenhuma

   Descrição do processamento: calcula o dado de saída

   Multiplicidade: $(N-1)$ para cada partícula

3. Acumulacão do potencial na energia potencial total;

   Nome do processo: U

   Dados de entrada: u_{ij}

   Dados de saída: U (energia potencial total)

   Entradas de sincronização: nenhuma

   Saídas de sincronização: nenhuma

   Descrição do processamento: realiza a somatória das entradas

   Multiplicidade: Um

4. Cálculo da força entre partículas

   Nome do processo: F_{ij}

   Dados de entrada: r_{ij}^2 e $(x_i - x_j)$

   Dados de saída: F_{ij} (força interpartículas)

   Entradas de sincronização: nenhuma

   Saídas de sincronização: nenhuma

   Descrição do processamento: calcula o dado de saída

   Multiplicidade: $(N-1)$ para cada partícula

5. Acumulação das forças sobre cada uma das partículas;

   Nome do processo: F_i

   Dados de entrada: F_{ij}

   Dados de saída: F_i (força sobre a partícula i)

   Entradas de sincronização: nenhuma

   Saídas de sincronização: nenhuma

Descrição do processamento: realiza a somatória das forças sobre cada partícula

Multiplicidade: Um para cada partícula

6. Cálculo da nova posição de cada partícula

Nome do processo: x_{n_i}

Dados de entrada: F_i, x_i e x_{a_i} (posição no quadro anterior da partícula i)

Dados de saída: x_{n_i} (nova posição da partícula i)

Entradas de sincronização: nenhuma

Saídas de sincronização: nenhuma

Descrição do processamento: calcula o dado de saída

Multiplicidade: Um para cada partícula

7. Cálculo da velocidade da partícula;

Nome do processo: v_i

Dados de entrada: x_{n_i} e x_{a_i}

Dados de saída: v_i (velocidade da partícula i)

Entradas de sincronização: nenhuma

Saídas de sincronização: aviso da liberação de x_{a_i}

Descrição do processamento: calcula o dado de saída

Multiplicidade: Um para cada partícula

8. Cálculo da energia cinética

Nome do processo: E_c

Dados de entrada: v_i e q (número do quadro atual)

Dados de saída: E_c (energia cinética) e SE_c (soma da energia cinética dos quadros desde a última renormalização)

Entradas de sincronização: nenhuma

Saídas de sincronização: nenhuma

Descrição do processamento: somatório das energias cinéticas de cada partícula e acumulação do total de cada quadro para utilização na renormalização

Multiplicidade: Um

9. Atualização das variáveis para preparar para um novo quadro da simulação;

Nome do processo: Atz

Dados de entrada: x_i e x_{n_i}

Dados de saída: x_{a_i} e x_i

Entradas de sincronização: aviso da liberação de x_{a_i}

Saídas de sincronização: nenhuma

Descrição do processamento: coloca os valores de x_i em x_{a_i} e os de x_{n_i} em x_i

Multiplicidade: Um para cada partícula

10. Contagem do número de quadros;

Nome do processo: Qds

Dados de entrada: q_m (número total de quadros requerido)

Dados de saída: q (número do quadro atual)

Entradas de sincronização: aviso de que os x_i já foram atualizados

Saídas de sincronização: $q < q_m$, usado para sinalizar o início de um novo quadro

Descrição do processamento: Recebe o valor de q_m e dispara os outros processos. Após terminado cada quadro, decrementa o valor de q, verifica se o mesmo chegou ao valor máximo, e inicia um novo quadro caso contrário.

Multiplicidade: Um

11. Renormalização

Nome do processo: Ren

Dados de entrada: x_{a_i}, v_i, F_i, q e SE_a

Dados de saída: x_{a_i}

Entradas de sincronização: nenhuma

Saídas de sincronização: nenhuma

Descrição do processamento: verifica se o quadro atual requer renormalização. Caso sim, recalcula x_{a_i} levando em consideração o valor de energia cinética média requerida. Caso não, deixa x_{a_i} como está.

Multiplicidade: Um

Já os elementos de armazenamento requeridos são os explanados abaixo:

1. Armazenamento das coordenadas atuais

Nome: x_i

Dados de entrada: x_i

Dados de saída: x_i e x_j (com $j \neq i$)

Entradas de sincronização: nenhuma

Saídas de sincronização: aviso de que as coordenadas já foram atualizadas e estão prontas para o cálculo de um novo quadro

Multiplicidade: Um para cada partícula

2. Armazenamento das coordenadas anteriores

Nome: x_{a_i}

Dados de entrada: x_{a_i}

Dados de saída: x_{a_i}

Entradas de sincronização: nenhuma

Saídas de sincronização: nenhuma

Multiplicidade: Um para cada partícula

O problema em sí de dinâmica molecular apresenta diversos graus de paralelismo que podem ser explorados de forma a se melhorar os tempos de execução. Passaremos agora a descrever estes paralelismos.

- O paralelismo mais evidente no problema corresponde à existência de diversas partículas, sendo que para todas elas os cálculos para a determinação de sua nova posição são semelhantes, podendo ser realizados em paralelo;
- Os cálculos de distância entre quaisquer pares de partículas, bem como os cálculos, decorrentes unicamente desse dado, de potencial interpartículas e força de atração entre as mesmas, podem ser realizados independentemente para todos os pares de partículas existentes;
- Os cálculos, citados no ítem acima, de potencial interpartículas e força de atração, são independentes entre sí;
- Em diversas partes da simulação, as diferentes dimensões têm cálculos semelhantes e independentes.

Podemos então representar o problema de dinâmica molecular como apresentado na fig. 18, onde os cálculos são simbolizados por processos e os valores de variáveis do problema simbolizados por elementos de memória. Note-se que o acesso às memórias é considerado completamente paralelo, com o intuito de simbolizar a independência das partículas do agregado.

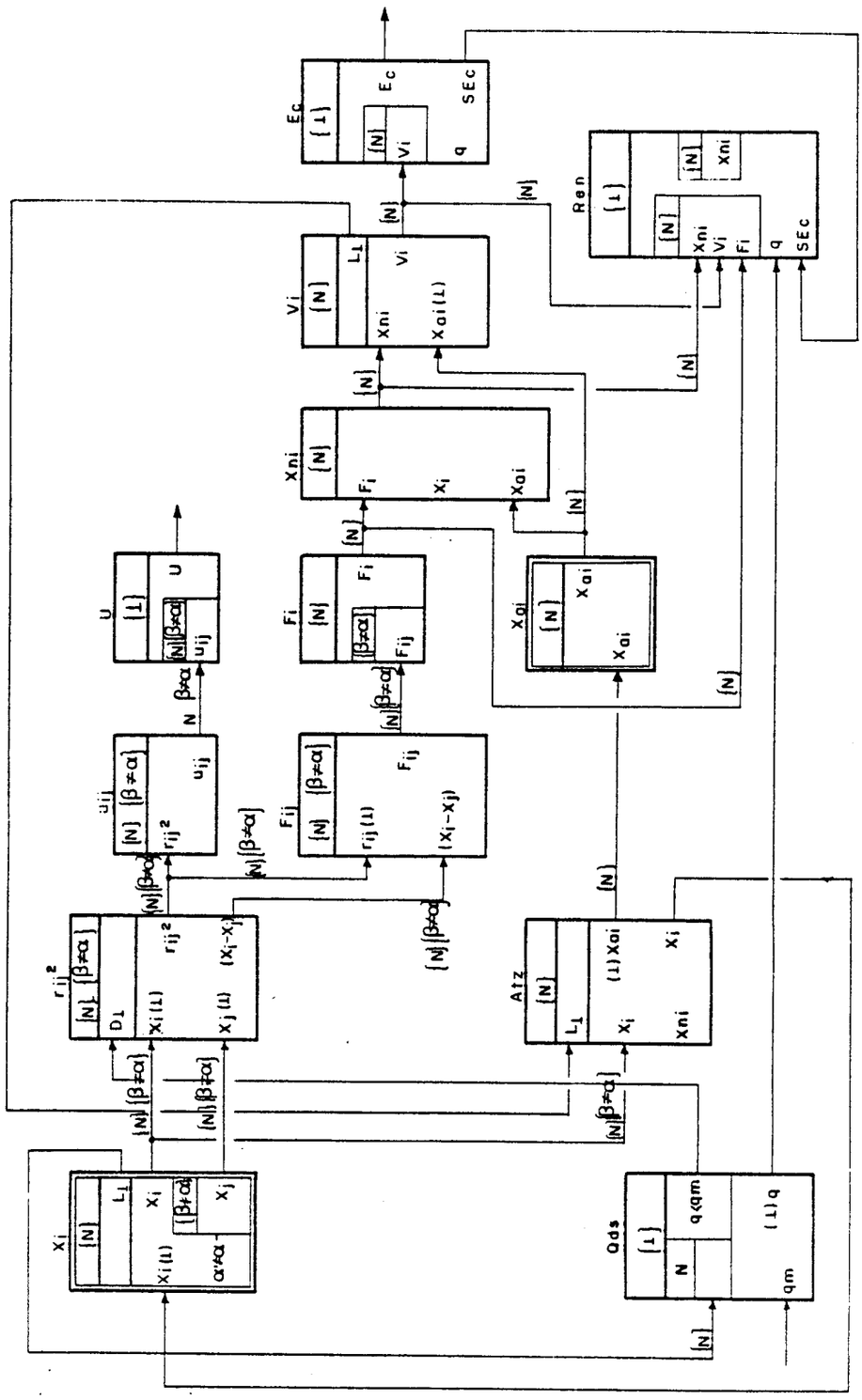


Figura 18: Esquema do paralelismo da dinâmica molecular

3.3 Paralelismo no algoritmo básico

O esquema de máximo paralelismo para o algoritmo básico (veja definição do algoritmo na seção 1.2) é o mesmo apresentado na fig. 18. No entanto, devemos fazer algumas considerações sobre a implementação prática do mesmo, visto que neste caso devemos tratar com elementos de memória e processamento reais e portanto em quantidade limitada, e não, como analisado anteriormente, com simbolizações de entidades físicas:

1. A implementação de todo o paralelismo descrito no diagrama implica a necessidade de uma quantidade extremamente alta de elementos de memória distintos e independentes. Por exemplo, suponhamos que as informações sobre coordenadas de cada partícula sejam armazenadas em elementos distintos de memória, com um elemento para cada partícula. Neste caso, em que temos necessidade de uma grande quantidade de elementos de memória, mesmo para números de partículas relativamente pequenos (como 1000 partículas), nos defrontaríamos com as seguintes limitações:

   - O cálculo de cada uma das partículas exige acesso a todas as outras partículas, o que implicaria numa forte concorrência pelo acesso dos dados em cada um dos elementos de memória
   - Independentemente do número de processadores alocados para o cálculo da distância entre os pares de partículas, somente tantos pares quantos são o número de elementos de memória (neste caso, tantos quantas são as partículas) podem ser calculados simultaneamente

   Este problema de concorrência fica ainda mais grave quando consideramos a necessidade de limitar a quantidade de elementos independentes de memória a um número pequeno em comparação com o número de partículas do sistema.

2. Existe também a necessidade de limitação do número de elementos de processamento em relação aos apresentados no diagrama. Isto implica na necessidade de serialização

de muitos dos cálculos que são apresentados como paralelos, com o conseqüente aumento do tempo de execução.

3. As comunicações entre os processos que executam os diversos cálculos também apresentarão limitações em relação ao diagrama apresentado, pois a mesma deve ser realizada através de algum meio físico, o que limita o número de comunicações que podem ser realizadas em paralelo.

As limitações apresentadas, principalmente a relativa à concorrência de acessos nas memórias, impedem que este algoritmo seja eficiente em uma implementação paralela. Como veremos, as limitações podem ser muito menores quando mudamos de algoritmo.

3.4 Paralelismo no algoritmo básico modificado

As diferenças entre este algoritmo e o anterior são a inclusão de um teste na distância, e o cálculo de cada um dos pares de partículas apenas uma vez (veja seção 1.3). Isto provoca mudanças nos seguintes ítens:

- multiplicidade dos processos, comunicações e entradas e saídas referentes aos pares de partículas
- inclusão de um processo de teste da distância entre as partículas com o raio crítico
- alteração no processo F_i para que considere cada entrada F_{ij} com representando uma força sobre i e sobre j (simetria das forças)

A descrição do processo novo e do alterado fica então:

1. Acumulação das forças sobre cada uma das partículas

   Nome do processo: F_i

   Dados de entrada: F_{ij} e $-F_{ji}$

Dados de saída: F_i (força sobre a partícula i)

Entradas de sincronização: sinais de seleção, para escolher as entradas em que a partícula correspondente ao processo entra como j em F_{ij}

Saídas de sincronização: nenhuma

Descrição do processamento: realiza a somatória das forças sobre cada partícula, considerando que cada par de partículas e computado apenas uma vez

Multiplicidade: Um para cada partícula

2. Teste da distância com o raio de corte

Nome do processo: Tst

Dados de entrada: r_{ij}^2 e $(x_i - x_j)$

Dados de saída: r_{ij}^2, $(x_i - x_j)$ e j

Entradas de sincronização: nenhuma

Saídas de sincronização: nenhuma

Descrição do processamento: seleciona entre os pares de entrada aqueles que apresentam distância menor que o raio de corte. Fornece também, para cada par de partículas selecionado, o número da partícula j (o valor de i pode ser abstraído a partir da própria organização dos processos). Note que o diagrama especifica um certo número de interagentes por partícula; entretanto, nem todas as partículas têm tantos pares quanto especificado. Desta forma, deve-se preencher as saídas restantes com algum sinal que indique que a mesma não está sendo utilizada.

Multiplicidade: Um para cada partícula

Considerando essas alterações, o diagrama do algoritmo básico modificado fica como na fig. 19.

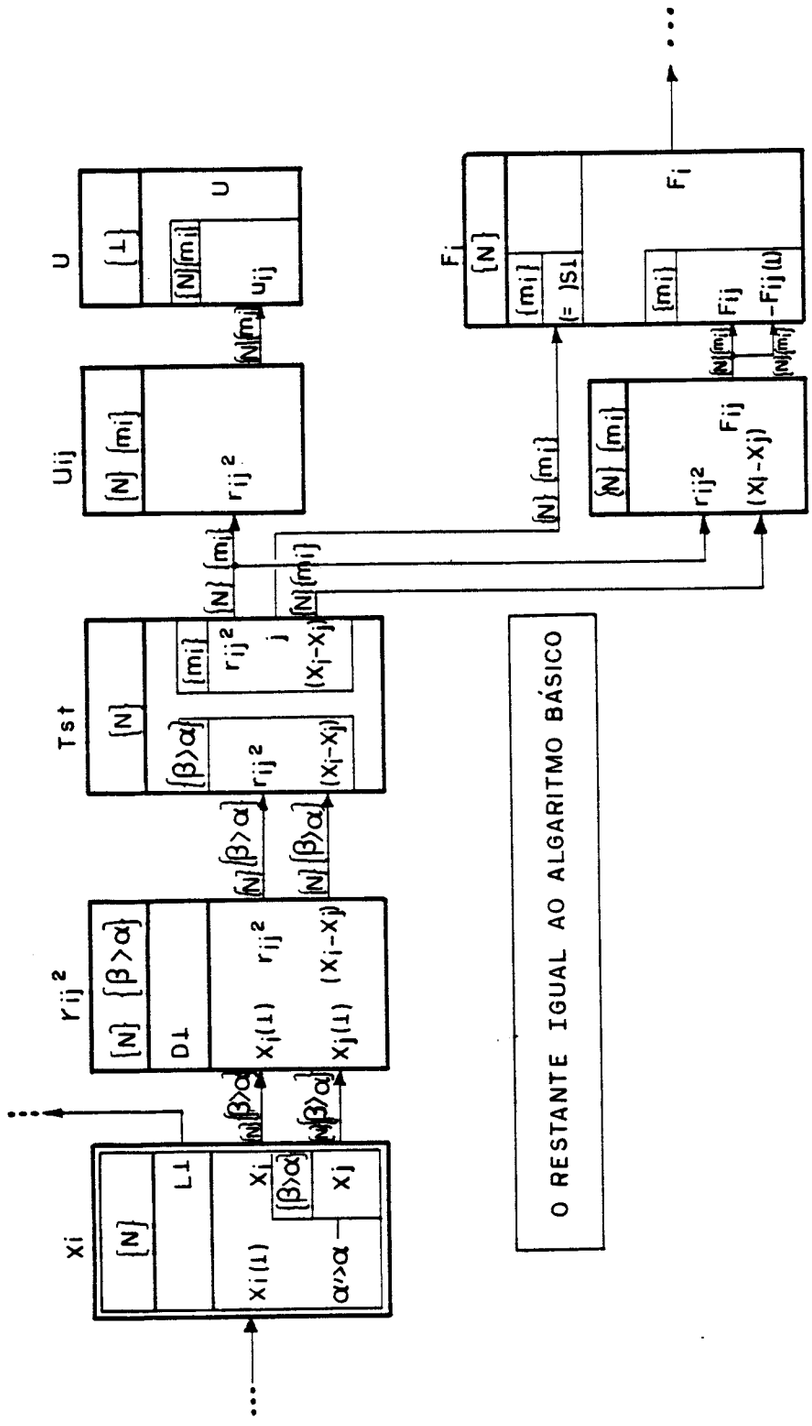


Figura 19: Diagrama do algoritmo básico modificado

Os mesmos problemas de implementação prática citados para o algoritmo anterior valem para este, sendo que é acrescentado um novo: Após o processo que realiza o teste da distância com o raio de corte, existem processos onde não é possível saber *a priori* a quantidade de cálculos que serão efetuados. Isto propicia o surgimento de desbalanceamentos entre os diferentes processos, o que concorre para a dificuldade na distribuição dos processos por diferentes processadores de uma forma eficiente.

3.5 Paralelismo no algoritmo de tabela de vizinhos

Notemos inicialmente que este algoritmo (veja seção 1.4) consiste de duas fases distintas quais sejam: a dos cálculos propriamente ditos e a de atualização da tabela de vizinhos. Como estas duas fases não podem ser executadas simultaneamente, nos deparamos com as seguintes opções:

1. *Destinação de processadores separados para cada uma das fases*: isto implica em que ao estar qualquer uma das fases ativa, os processadores da outra fase estarão parados, o que provoca desperdício de recursos computacionais e, portanto, baixa eficiência

2. *Utilização dos mesmos processadores para as duas fases*: neste caso, a estrutura de interligação entre os processadores deve ser otimizada para uma das duas fases, o que implica em baixa eficiência na execução da outra fase

O armazenamento aqui consiste em tabelas de vizinhos que contém números identificadores das partículas, que serão utilizados para se acessar os dados das mesmas em uma tabela de coordenadas (veja fig. 20).

O diagrama para a implementação deste algoritmo está apresentado na fig. 21, onde incluimos representações para o armazenamento da tabela de vizinhos e da tabela de coordenadas. Incluímos também um novo processo (AtzTV) para cuidar da atualização periódica da tabela de vizinhos. Os processos e elementos de memória novos e os alterados

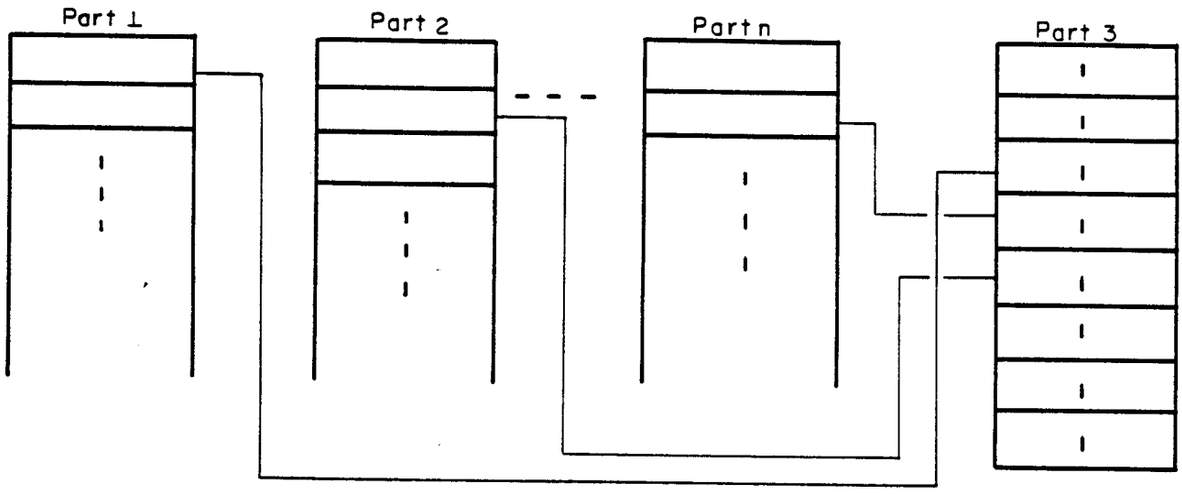


Figura 20: Estrutura de dados para tabela de vizinhos com ponteiros

ficam então da seguinte forma:

1. Armazenamento das coordenadas atuais

   Nome: x_i

   Dados de entrada: x_i

   Dados de saída: x_i e x_j (com j pertencente à tabela de vizinhos de i)

   Entradas de sincronização: Sinais para seleção de quais as partículas que pertencem à tabela de vizinhos da partícula armazenada neste elemento (Estes sinais selecionam com quais elementos de armazenamento deve ser estabelecida a comunicação para termos todos os dados necessários ao cálculo da nova posição da partícula em questão)

   Saídas de sincronização: aviso de que as coordenadas já foram atualizadas e estão prontas para o cálculo de um novo quadro

   Multiplicidade: Um para cada partícula

2. Armazenamento da tabela de vizinhos

   Nome: TV

Dados de entrada: j (lista das partículas que pertencem à tabela de vizinhos da partícula especificada)

Dados de saída: j

Entradas de sincronização: sinal para início de um novo quadro (liberação dos dados de saída)

Saídas de sincronização: aviso de que as tabelas já foram atualizadas e estão prontas para o cálculo de um novo quadro

Multiplicidade: Um para cada partícula

3. Teste da distância com o raio de corte

Nome do processo: Tst

Dados de entrada: r_{ij}^2, $(x_i - x_j)$ e j (número das partículas que constam na tabela de vizinhos de i)

Dados de saída: r_{ij}^2, $(x_i - x_j)$ e j

Entradas de sincronização: nenhuma

Saídas de sincronização: nenhuma

Descrição do processamento: seleciona entre os pares de entrada aqueles que apresentam distância menor que o raio de corte. Fornece também, para cada par de partículas selecionado, o número da partícula j (o valor de i pode ser abstraído a partir da própria organização dos processos)

Multiplicidade: Um para cada partícula

4. Atualização da tabela de vizinhos

Nome do processo: AtzTV

Dados de entrada: x_i e q

Dados de saída: j (partículas pertencentes às tabelas de vizinhos de cada uma das partículas)

Entradas de sincronização: sinal de que é necessário o cálculo de um novo quadro

Saídas de sincronização: sinal de que o cálculo do próximo quadro pode prosseguir

Descrição do processamento: Quando o quadro não necessita recálculo da tabela de vizinhos, simplesmente dispara novo quadro. Caso o quadro exija a atualização, realiza a atualização e só então dispara novo quadro

Multiplicidade: Um

Para a implementação paralela do mesmo nos deparamos com os seguintes fatos:

1. As tabelas de vizinhos das diversas partículas podem ser distribuídas entre diferentes processadores. Entretanto, como cada partícula pode aparecer em várias tabelas de vizinhos, haverá concorrência pelo acesso dos dados da mesma presentes na tabela de coordenadas

2. Ao exposto no ítem anterior agregamos os fato de que, na tabela de coordenadas, não existirá um elemento de memória independente para cada partícula, o que ampliará ainda mais a citada concorrência

3. Se quisermos aumentar o número de partículas, aumentaremos a carga dos processadores ou, se também aumentamos o número de processadores, crescerá a concorrência no acesso da tabela de coordenadas

3.6 Paralelismo no algoritmo de granulação grosseira

O algoritmo de granulação grosseira (seção 1.5) apresenta, como veremos, melhores características de paralelização, por possuir em sua estrutura informações sobre a distribuição física das partículas pelo espaço. O diagrama do mesmo é apresentado na fig. 22.

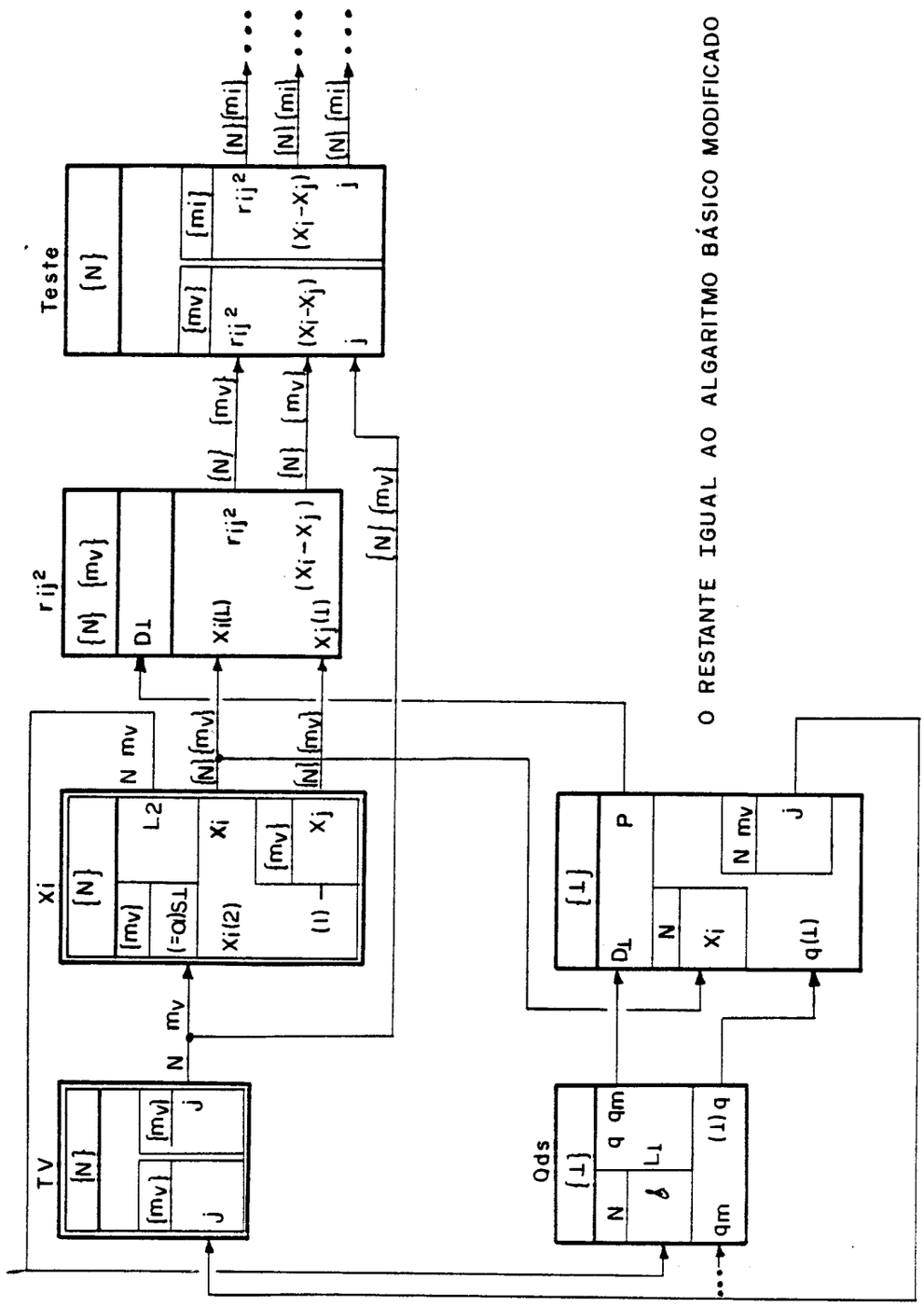
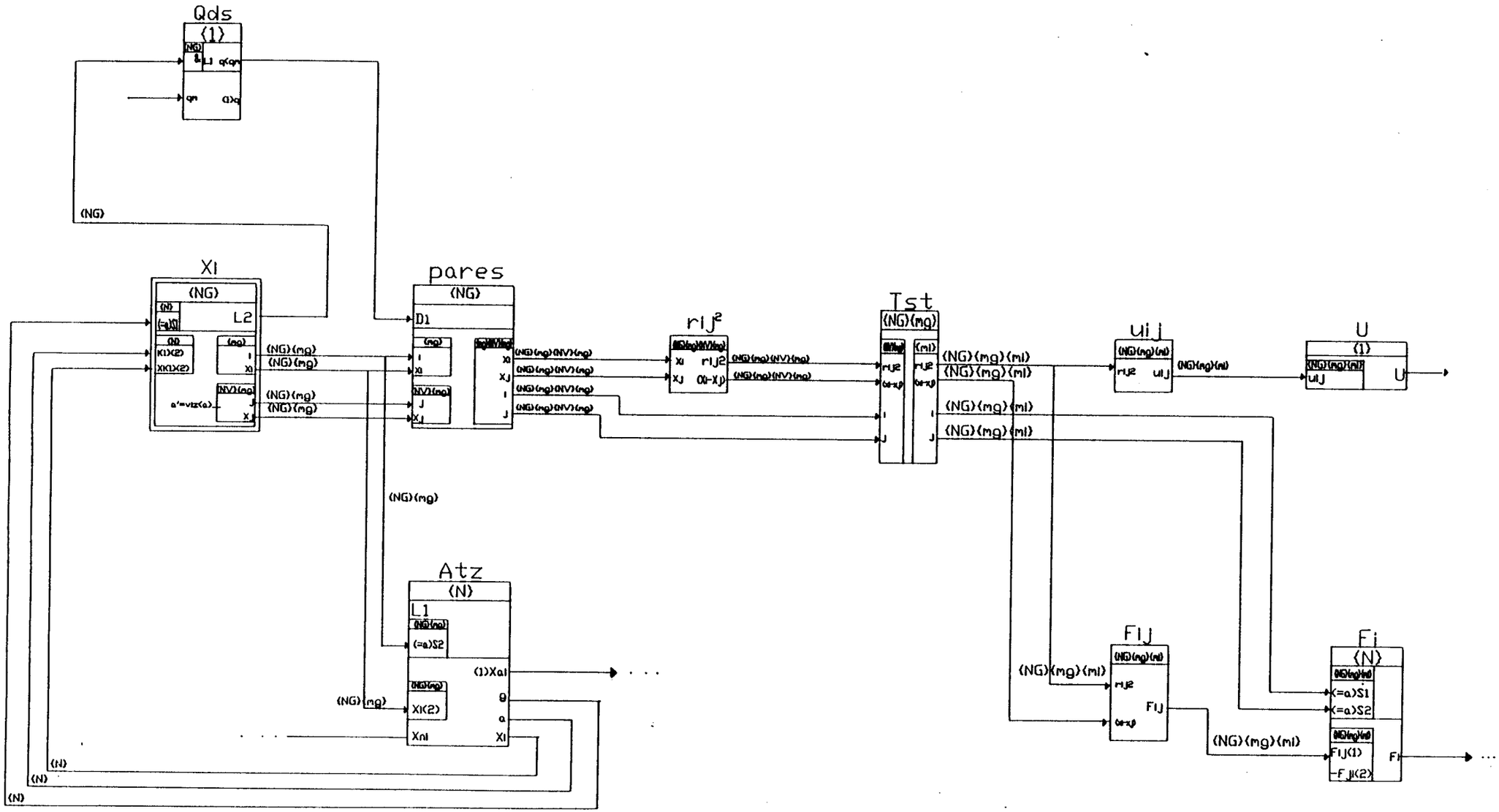


Figura 21: Diagrama do algoritmo de tabela de vizinhos

Figura 22: Diagrama do algoritmo de granulação grosseira


Qds
⟨1⟩
(NG)
L1
qn
(NG)
Xl
(NG)
L2
(N)
(NG)(mg)
(NG)(mg)
(NG)(mg)
(NG)(mg)
pares
(NG)
D1
(NG)(mg)(NV)(mg)
(NG)(mg)(NV)(mg)
(NG)(mg)(NV)(mg)
(NG)(mg)(NV)(mg)
rij²
(NG)(mg)(NV)(mg)
(NG)(mg)(NV)(mg)
Tst
(NG)(mg)
(NG)(mg)(ml)
(NG)(mg)(ml)
(NG)(mg)(ml)
(NG)(mg)(ml)
uij
(NG)(mg)(ml)
(NG)(mg)(ml)
U
⟨1⟩
(NG)(mg)(ml)
(NG)(mg)
Atz
⟨N⟩
L1
(NG)(mg)
(=a)S2
(1)Xal
(NG)(mg)
Xl(2)
Xnl
Xl
(N)
(N)
(N)
Fij
(NG)(mg)(ml)
(NG)(mg)(ml)
Fij
(NG)(mg)(ml)
Fi
⟨N⟩
(NG)(mg)(ml)
(=a)S1
(=a)S2
(NG)(mg)(ml)
Fij(1)
Fij(2)
Fi

Os processos e elementos de armazenamento novos e alterados para este algoritmo são apresentados abaixo:

1. Armazenamento das coordenadas atuais

   Nome: Grão

   Dados de entrada: x_i e i (escrever dados de uma partícula corresponde a adicionar uma partícula ao grão com coordenadas e número indicados)

   Dados de saída: x_i, x_j, i e j (com j pertencente a grãos vizinhos do grão a que i pertence)

   Entradas de sincronização: Sinais para seleção do grão ao qual uma dada partícula deve ser acrescentada

   Saídas de sincronização: aviso de que as coordenadas já foram atualizadas e estão prontas para o cálculo de um novo quadro

   Multiplicidade: Um para cada grão

2. Formação de pares de partículas

   Nome do processo: Pares

   Dados de entrada: i, j, x_i e x_j

   Dados de saída: os mesmos da entrada

   Entradas de sincronização: aviso de que pode começar um novo quadro

   Saídas de sincronização: nenhuma

   Descrição do processamento: com os dados de cada partícula apresentados na entrada, monta todos os pares de partículas necessários

   Multiplicidade: Um por grão

3. Teste com o raio de corte

Nome do processo: Tst

Dados de entrada: r_{ij}^2, $(x_i - x_j)$, i e j

Dados de saída: os mesmos da entrada

Entradas de sincronização: nenhuma

Saídas de sincronização: nenhuma

Descrição do processamento: Seleciona entre todos os pares da entrada apenas os que possuem distância menor que o raio de corte

Multiplicidade: Um por cada partícula de cada grão

4. Cálculo da força sobre cada partícula

Nome do processo: F_i

Dados de entrada: F_{ij} e $-F_{ji}$

Dados de saída: F_i

Entradas de sincronização: uma entrada para a seleção dos F_{ij} que agem sobre a partícula do processo e outra para a seleção dos $-F_{ji}$

Saídas de sincronização: nenhuma

Descrição do processamento: Seleciona as forças que agem sobre cada partícula e soma todas

Multiplicidade: Um para cada partícula

5. Atualização

Nome do processo: Atz

Dados de entrada: x_i e x_{a_i}

Dados de saída: x_{a_i}, x_i e g (número do grão ao qual cada partícula pertence)

Entradas de sincronização: sinal de seleção da partícula correta (isto é, a que corresponde ao processo, entre todas as apresentadas por cada um dos grãos) e sinal que indica que x_{a_i} pode ser atualizada

Saídas de sincronização: nenhuma

Descrição do processamento: Entre todas as partículas apresentadas por cada grão, seleciona aquela que corresponde a cada um dos processos e escreve-a como coordenada anterior quando permitido. Lê valor de x_{a_i}, calcula o novo grão da partícula (g) e envia para armazenamento

Multiplicidade: Um para cada partícula

Para uma implementação paralela deste algoritmo nos deparamos com os seguintes fatos:

1. Os grãos podem ser distribuídos por diferentes processadores

2. Para o cálculo de um grão, necessitamos apenas o conhecimento dos dados de seus grãos vizinhos, o que limita sensivelmente e de uma forma bem definida as necessidades de acesso por um processador aos dados de outro

3. Num caso real, em que dispomos de menos processadores do que os grãos necessários, a alocação de grãos próximos no mesmo processador diminui ainda mais a necessidade de acesso a dados de outros processadores

4. O sistema pode crescer uniformemente, acrescentanto processadores quando quisermos aumentar o número de partículas processadas, de forma a que o número de grãos por processador permaneca constante, o que garante que o tempo de processamento fique praticamente inalterado

3.7 Conclusão

Das análises apresentadas acima, bem como das realizadas no capítulo anterior, podemos verificar que o algoritmo de granulação grosseira é, dentre os estudados neste trabalho, o que melhor se adapta para a implementação em um sistema dedicado de computação paralela. Resumindo as características que o tornam o mais indicado, dizemos:

- Crescimento linear do tempo de execução com o número de partículas (para sistema com número fixo de processadores)
- Acesso a dados de outros processadores é necessário apenas entre vizinhos imediatos
- O número de partículas pode aumentar sem que exista um aumento no tempo total de execução, desde que a quantidade de processadores do sistema cresça proporcionalmente ao número de novos grãos

Capítulo 4

Definição da proposta

Escolhido o algoritmo que formará a base do sistema dedicado à dinâmica molecular a ser proposto, passaremos agora a apresentar os detalhes de uma implementação para a mesma. Começaremos apresentando as características básicas do *transputer* T800 [16], seguida de uma apresentação resumida da linguagem OCCAM [17], adaptada especialmente à família dos *transputers*. Após isso, passaremos à descrição da estrutura de processos e da rede de *transputers* escolhida.

4.1 O *transputer* T800

O T800 é um integrado que reune, numa mesma pastilha, os seguintes elementos (veja fig. 23):

- Um processador inteiro de 32 bits, com amplas facilidades para a multiprogramação
- Um processador de ponto flutuante que trabalha com 32 ou 64 bits
- Uma unidade de memória com 4 kilobytes de capacidade
- Quatro elementos de comunicação, cada um com duas ligações (*links*) com taxas de até 20 Mbits por segundo
- Interface para memória externa de até 4 Gbytes

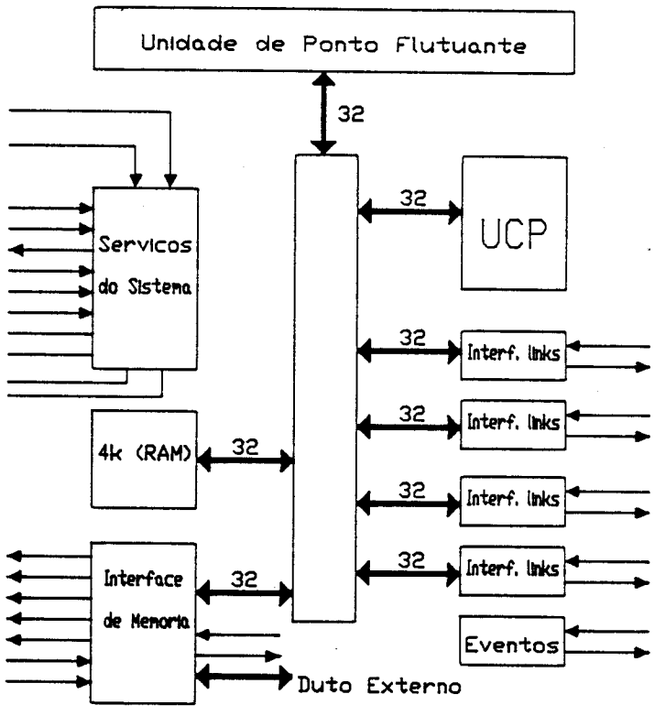


Figura 23: Estrutura do *transputer* T800

Essa estrutura permite a implementação de sistemas multiprocessador com muita facilidade, bastando a interligação dos diferentes processadores através dos *links* dos mesmos. Isto garante também facilidade para a expansão pois, dado um sistema, se queremos aumentar o número de processadores do mesmo basta incluir novos *transputers* e realizar a interconexão adequada dos *links*, sem a introdução de nenhum problema de carga, pois cada *link* estabelece a comunicação entre apenas dois *transputers*.

4.1.1 Estrutura da unidade de processamento inteiro

A unidade de processamento inteiro opera com base nos seguintes registradores, todos de 32 bits:

`Iptr` que aponta a próxima instrução a ser executada

`Wptr` que aponta a região de trabalho do processo corrente

Areg topo da pilha de avaliação

Breg pilha de avaliação

Creg pilha de avaliação

Oreg registrador de operando

O registrador **Iptr** corresponde ao contador de programa, presente em todos os computadores de arquitetura de Von-Neuman. O **Wprt** representa um ponteiro de referência para todas as variáveis locais do processo corrente. Os registradores **Areg**, **Breg** e **Creg** formam a pilha de avaliação. Quando um dado é introduzido na pilha o mesmo entra pelo registrador **Areg**, fazendo com que o conteúdo anterior de **Areg** se desloque para **Breg** e o de **Breg** para **Creg**. Quando um dado é retirado de **Areg**, ocorre o movimento inverso, sendo o dado anterior de **Breg** deslocado para **Areg** e o dado anterior de **Creg** deslocado para **Breg**; o novo conteúdo de **Creg** é indefinido. Desta forma, estes três registradores operam como uma pilha, sendo que todas as operações do *transputer* são executadas sobre essa pilha, devendo as operações sobre os dados ser desenvolvidas em notação polonesa reversa.

Ao lado desta estrutura, o processador conta com filas para o controle de processos concorrentes e com dois temporizadores, um para cada uma das prioridades existentes.

4.1.2 Introdução às instruções de máquina

O T800 opera basicamente com um elenco de dezesseis operações, sendo que a partir destas podem-se montar operações mais complexas. As dezesseis operações básicas são as seguintes:

pfix	prefixo
nfix	prefixo negativo
opr	opere (interpreta **Oreg** como código de operação)
ldc	carregue de constante

`ldl`	carregue de variável local
`stl`	armazene em variável local
`ldlp`	carregue ponteiro local
`adc`	some com constante
`eqc`	teste igualdade com constante
`j`	desvie incondicionalmente
`cj`	desvie condicionalmente
`ldnl`	carregue de variável não local
`stnl`	armazene em variável não local
`ldnlp`	carregue ponteiro não local
`call`	chame subrotina
`ajw`	ajuste região de trabalho (para subrotina)

Todas essas instruções ocupam um dos *nibbles* de um *byte*, sendo que o *nibble* restante será ocupado por um valor inteiro de 4 bits. As instruções de prefixo e prefixo negativo servem para extender o valor utilizado, quando 4 bits não são suficientes. A instrução "opere" serve para interpretar o conteúdo de `Oreg` como um código de operação, que se referirá a uma das outras operações, não apresentadas acima.

Para uma descrição mais completa dessas instruções e para conhecimento das não citadas sugerimos uma consulta a [18].

4.2 A linguagem OCCAM

A linguagem OCCAM foi desenvolvida pela INMOS em conjunto com o desenvolvimento dos *transputers*, com o intuito de oferecer uma linguagem de alto nível especialmente adaptada à estrutura dos mesmos. A linguagem realiza a implementação de paralelismo baseado no conceito de *processos seqüenciais comunicantes*, introduzido por Hoare [19].

Neste conceito, os programas consistem em **processos** seqüenciais rodando em paralelo e que se comunicam entre sí unicamente através de **canais** unidirecionais. Não existe nenhuma outra forma de comunicação entre os processos.

Passaremos agora a uma descrição resumida da linguagem, através da apresentação de seus elementos e estruturas básicas, bem como das características que a fazem bem adaptada à programação eficiente de redes de *transputers*.

4.2.1 Processos básicos

São três os processos básicos da linguagem OCCAM: atribuições, entradas de dados por canais e saídas de dados por canais.

A atribuição se representa como:

```
a := (expressao)
```

e corresponde à alteração do valor da variável a, que passa a ter o valor resultante da expressão da direita.

A entrada de dados por canal se faz da seguinte forma:

```
c ? x
```

que significa que a variável x terá seu valor alterado de acordo com o que for recebido através do canal c.

A saída de dados por canal apresenta o seguinte formato:

```
c ! (expressao)
```

e indica que o resultado da expressão deve ser transmitido pelo canal c.

4.2.2 Estruturas de controle de fluxo

A primeira estrutura a se notar corresponde à que indica a necessidade de execução seqüencial de um certo número de processos. A indicação se faz da seguinte forma:

```
SEQ
  proc1
  proc2
  proc3
  ...
```

indicando que os processos `proc1`, `proc2`, `proc3`, etc. devem ser executados seqüencialmente. Notemos que além dos processos básicos, citados acima, qualquer estrutura também é considerada um processo. Assim, as estruturas podem ser aninhadas umas dentro de outras. Um fato importante aqui é o denteamento dos diversos processos em relação ao `SEQ`. Um processo, para ser considerado como fazendo parte de uma estrutura, tanto o `SEQ` como as que serão apresentadas a seguir, deve ser denteado de dois espaços em relação ao identificador da estrutura.

Para indicar a execução paralela de um conjunto de processos, procedemos assim:

```
PAR
  proc1
  proc2
  proc3
  ...
```

que indica que os processos `proc1`, `proc2`, `proc3`, etc. devem executar em paralelo. Quando cada um dos processo que aparecem em um mesmo `PAR` é alocado a um processador diferente (veremos a forma de alocação adiante) então temos verdadeiro paralelismo. Quando alguns (ou todos) dos processos paralelos são alocados no mesmo processador, então os mesmos são executados concorrentemente através de técnica de *time-sharing*.

Outra estrutura é a que permite tomada de decisões conforme ao valor de dados. A sua representação é a seguinte:

```
IF
  cond1
    proc1
  cond2
    proc2
  cond3
    proc3
  ...
```

Neste caso, o processador testa as condições cond1, cond2, cond3, etc. na ordem especificada e executa o processo correspondente à primeira condição verdadeira.

Outra construção existente permite a escolha de um entre vários processos, de acordo com qual canal, entre vários, recebe primeiro um dado. A representação é como segue:

```
ALT
  guarda1
    proc1
  guarda2
    proc2
  guarda3
    proc3
  ...
```

indicando que será ativado o processo correspondente ao primeiro guarda ativado, sendo cada guarda corresponde à entrada de dados por um canal. Para esclarecer melhor a operação, vejamos os seguinte exemplo:

```
ALT
  ab ? b
   a := a - b
  fg ? d
   SEQ
    a := a + d
    c := 1
  (a > 0) & xy ? e
   a := 0
```

O significado deste código é o seguinte: se o primeiro canal (entre os canais ab, fg e xy) a receber dados for o canal ab, então o dado que veio por este canal deve ser armazenado na variável b, sendo em seguida o valor de b subtraído do valor atual de a. Caso o primeiro dado chegue por fg, então o mesmo deve ser armazenado em d, sendo em seguida o valor de d somado ao de a e, após isto, o valor de c é alterado para 1. No caso do canal xy temos uma condição adicional, a de que $a > 0$. Isto significa que se, no instante em que o ALT for executado o valor de a for menor ou igual a zero, então o canal xy não será testado, isto é, a chegada de dados pelo mesmo será ignorada.

Para a execução de ciclos, dispomos da seguinte estrutura:

```
WHILE condição
  processo
```

representando que o processo indicado deve ser repetido enquanto a condição apresentada for verdadeira.

Outra forma de realizar repetições é através da replicação de processos. O modo de indicar isto é como o apresentado abaixo:

```
SEQ var := ini FOR cont
```

```
proc
```

significando a existência de `cont` processos semelhantes a `proc`, sendo que os mesmos diferem pelo valor de `var`, e o primeiro processo apresenta `var` igual a `ini`. Temos ainda, neste caso, que os diversos processos devem ser executados seqüencialmente. A replicação pode ser utilizada também para outras estruturas, como a de processos paralelos (`PAR`) e de decisões (`IF`).

4.2.3 Estruturas de dados e de canais

A linguagem OCCAM apresenta os seguintes tipos de dados primitivos:

`CHAN`	canal de comunicação entre processos
`TIMER`	temporizador
`BOOL`	variável verdadeira ou falsa
`BYTE`	corresponde a 8 bits
`INT`	inteiro com sinal com implementação mais eficiente no *transputer* em questão (para o T800, 32 bits)
`INT16`	inteiro com sinal de 16 bits
`INT32`	inteiro com sinal de 32 bits
`INT64`	inteiro com sinal de 64 bits
`REAL32`	ponto flutuante 32 bits, padrão IEEE P754 draft 10.0
`REAL64`	ponto flutuante 64 bits, padrão IEEE P754 draft 10.0

Para declarar que uma variável é de um dado tipo, escrevemos:

```
tp var :
```

que significa que a variável `var` é do tipo `tp`, e que a declaração é válida para o processo a seguir. Uma declaração semelhante pode ser utilizada para os canais, simplesmente incluindo o tipo entre parêntesis e precedido de `CHAN`:

```
CHAN (tp) c :
```

indicando que o canal c transmite dados do tipo tp.

A estrutura de dados básica em OCCAM é o vetor. Sua representação é através de colchetes, semelhantemente à utilizada na linguagem C, onde:

```
vet[ind1][ind2]...[indn]
```

indica que o vetor vet possui n índices, sendo o primeiro ind1, o segundo ind2 e assim sucessivamente.

Se quisermos nos referir apenas a trechos de um vetor, então podemos utilizar a seguinte notação:

```
[vet FROM ini FOR cont]
```

que indica que estamos nos referindo ao trecho do vetor vet[ind] que começa com ind valendo ini e compreende cont elementos. Note-se que vet pode ser tanto um vetor simples (com apenas um índice) quanto um vetor multiplamente indexado. Neste caso o índice que será variado dentro da faixa especificada será o último. Por exemplo, para um vetor triplamente indexado, vet[i][j][k], podemos nos referir a uma faixa de variação de k que vai desde 4 até 6 (inclusive os extremos) da seguinte forma:

```
[vet[i][j] FROM 4 FOR 3]
```

considerando-se os valores de i e j constantes.

Assim como as variáveis, os canais também apresentam tipos que no caso são chamados *protocolos*, e que incluem, além dos tipos que se aplicam a variáveis, também os seguintes:

Protocolos seqüenciais: equivalem a uma seqüência especificada de transmissões de diversos dados, possivelmente com diferentes tipos

```
CHAN (INT; REAL32) c1 :
INT a :
REAL32 b :
 PAR
  ...
  c1 !  3; 2.1
  ...
  c1 ?  a; b
  ...
```

Protocolos de vetor: corresponde à transmissão de uma cadeia de dados, sendo em primeiro lugar enviado o tamanho da cadeia, e então enviados os dados do tipo especificado

```
CHAN ([INT]INT) c2 :
[4]INT v1 :
[10]INT v2 :
INT a :
 PAR
  ...
  c2 !  4; v1
  ...
  c2 ?  a; [v2 FROM 0 FOR a]
  ...
```

Protocolos variantes: permitem o envio de dados de diferentes tipos através de um mesmo canal. Isto é feito através da transmissão de um *tag* (identificador do tipo), seguido pelo dado

```
CHAN (1; INT OR 2; REAL32) c3 :
INT a, tag :
REAL32 b :
 PAR
  ...
  c3 !  1; 3
  ...
  c3 !  2 ; 2.1
  ...
  c3 ?  tag; a
  ...
  c3 ?  tag; b
  ...
```

Entrada conferida: confere o valor de uma entrada no momento em que a mesma chega, parando a execução do processo se o valor recebido não for o esperado

```
CHAN (1; INT OR 2; REAL32) c3 :
INT a, tag :
REAL32 b :
 PAR
  ...
  c3 !  1; 3
  ...
  c3 !  2 ; 2.1
  ...
  c3 ?= 1; a
```

```
...
c3 ?= 2; b
...
```

Protocolo de *tag* apenas: corresponde a um protocolo variante onde não existem dados, mas apenas o *tag*.

```
CHAN (1 OR 2 OR 3) c4 :
INT t1 :
 SEQ
  ...
  c4 !  2
  ...
  c4 ?  t1
  ...
```

4.2.4 Outras características importantes

Existem, em OCCAM, formas de ser realizar a alocação de certas estruturas lógicas a elementos físicos. Passaremos a citá-las:

Alocação de processadores: a alocação de determinado processo a determinado processador se faz através da estrutura PLACED PAR, com a seguinte forma:

```
PLACED PAR
 PROCESSOR 0
  proc0
 PROCESSOR 1
  proc1
```

...

indicando que o processo `proc0` deve ser executado no processador 0, o processo `proc1` no processador 1, etc.

Alocação de *links*: se faz com a utilização da construção:

```
PLACE nm.cnl AT num :
```

representando que o canal lógico `nm.cnl` deve ser associado ao *link* físico representado pelo número `num`.

Alocação de memória para dados: faz-se da mesma forma que para a alocação de *links*, só que citando o endereço de memória que queremos utilizar:

```
PLACE var AT loc :
```

indicando que a variável `var` deve ser colocada no endereço de memória `loc`.

Alocação de memória para programa: se faz através da alocação de um vetor com o tamanho necessário para conter as instruções de máquina da rotina. Utilizamos então as instruções:

```
PLACE v AT e :
PLACE r IN v :
```

que resulta na alocação do vetor `v` no endereço `e` e na alocação da rotina `r` dentro do espaço reservado ao vetor `v`.

Para simplificar a leitura e escrita de programas, são utilizadas *constantes* e *abreviaturas*. Elas apresentam as mesmas regras de escopo das declarações de variáveis. Uma constante é declarada da seguinte forma:

```
VAL cte IS vlr :
```

que declara que a constante chamada `cte` terá um valor de `vlr`. Esta declaração também é utilizada para vetores constantes. Uma abreviatura se representa como em:

```
nme IS elm
```

que indica que o nome `nme` será uma abreviatura para o elemento `elm`, onde `elm` é qualquer elemento de OCCAM, por exemplo, trecho de um vetor.

As rotinas em OCCAM são definidas da seguinte forma:

```
PROC nme ( lst.prm )
  proc.rot
```

onde `nme` é o nome pelo qual a rotina será invocada, `lst.prm` é a lista dos parâmetros da rotina e `proc.rot` é o processo que executa as funções da rotina.

Esta apresentação da linguagem OCCAM foi resumida tendo em vista os objetivos deste trabalho. Para uma descrição mais completa sugerimos uma consulta a [17].

4.3 Esquema da implementação

Descreveremos agora a estrutura proposta para a implementação de dinâmica molecular em uma rede de *transputers*. Antes, porém, de prosseguir, e com o intuito de evitar confusões com os termos, introduziremos a seguinte notação:

Processador: corresponde a um *transputer*, encarregado de uma tarefa especial no processamento total

Unidade de processamento: corresponde a um conjunto de transputers que se encarrega do processamento total dos cálculos de dinâmica molecular para uma parte das partículas

Isto se refere, em termos já consagrados na comunidade de processamento paralelo, à existência de uma *divisão geométrica* do processamento total em *unidades de processamento*, e de uma *divisão algoritmica* de cada unidade de processamento em *processadores*. Para maior clareza, definimos os termos utilizados:

Divisão Geométrica: Utiliza-se quando um problema, que apresenta processamentos semelhantes para grande quantidade de dados diferentes tem os dados divididos entre várias unidades de processamento, sendo todas elas encarregadas de realizar o mesmo processamento.

Divisão Algoritmica: Quando o algoritmo para o processamento de certos dados apresenta diversas fases, sendo que cada uma pode ser realizada em paralelo, podemos implementar a divisão algoritmica alocando processadores diferentes para as diversas fases de processamento

Definidos os termos, passemos agora à apresentação da estrutura proposta.

4.3.1 Divisão geométrica

Devido à grande quantidade de dados semelhantes que devem ser processados, conforme vimos no capítulo 1 e no capítulo 3, temos a sugestão de realização de uma divisão geométrica do problema. Isto implica a divisão do espaço de simulação (aonde as partículas se distribuem) de uma dada forma entre um certo número de unidades de processamento. No entanto esta distribuição deve obedecer algumas diretrizes básicas do nosso trabalho, quais sejam:

1. Facilidade de expansão, de acordo com a disponibilidade de recursos computacionais do usuário, com baixo comprometimento do desempenho global do sistema

2. Implementação a mais simples possível com uma rede de *transputers*

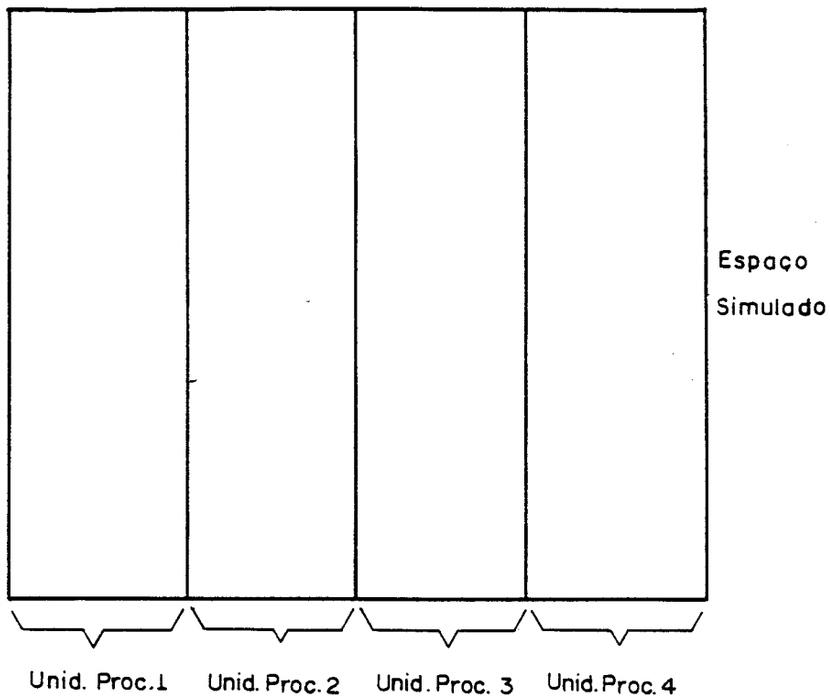


Figura 24: Divisão do espaço de simulação em faixas

3. Capacidade de crescimento do sistema de processamento proporcionalmente ao crescimento do dimensionamento do problema

Tendo em vista esses objetivos, estudaremos agora alguns tipos de interligação possíveis entre as unidades de processamento.

Podemos realizar a divisão do espaço de simulação entre um certo número p de unidades de processamento simplesmente dividindo um trecho do espaço para cada unidade. Assim, o esquema da divisão seria como apresentado na fig. 24, onde exemplificamos para $p = 4$. Note que, devido ao fato das partículas estarem em movimento, as mesmas podem trocar de unidade de processamento, o que significa que:

- Deve haver comunicação entre unidades de processamento
- O número de partículas por unidade de processamento é variável

Devido ao fato de que o intervalo de tempo deve ser suficientemente pequeno para garantir a correção da simulação, teremos também deslocamentos de partículas pequenos (veja avaliação no cap. 5), de forma que a comunicação deve ser apenas entre divisões de pro-

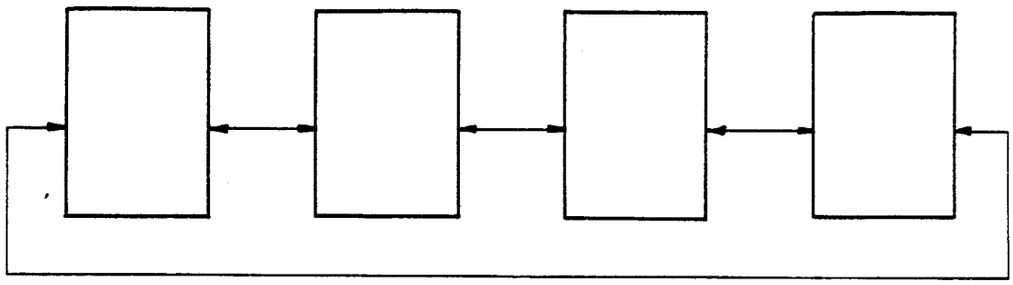


Figura 25: Estrutura de interligação das unidades de processamento

cessamento vizinhas. Ainda quanto às interligações entre unidades de processamento, o fato de que o espaço de simulação é repetido periodicamente nos indica que a unidade de processamento mais à esquerda deve ser interligada à unidade de processamento mais à direita. Estes fatos mostram que a estrutura de interligação deve ser em anel, conforme especificado na fig. 25.

Vejamos agora qual o comportamento desse anel com relação à expansibilidade do sistema. Podemos notar que a inclusão de uma nova unidade de processamento implica apenas na reformulação das ligações entre duas outras: necessitamos apenas interromper uma dessa ligações e introduzir nesse intervalo a nova unidade de processamento. Vemos também que a expansão pode ser feita de acordo com a disponibilidade de novas unidades de processamento, sem restrições, bastando redividir os dados do espaço de simulação entre todas as unidades de processamento. A fig. 26 mostra a expansão de $p = 4$ para $p = 5$. Existe um limite para a possibilidade de expansão mas, como veremos no cap. 5, é suficientemente grande.

Quanto à implementação com rede de *transputers*, basta utilizarmos para a comunicação entre unidades de processamento os *links*, o que garante flexibilidade e simplicidade para alterações (como expansão).

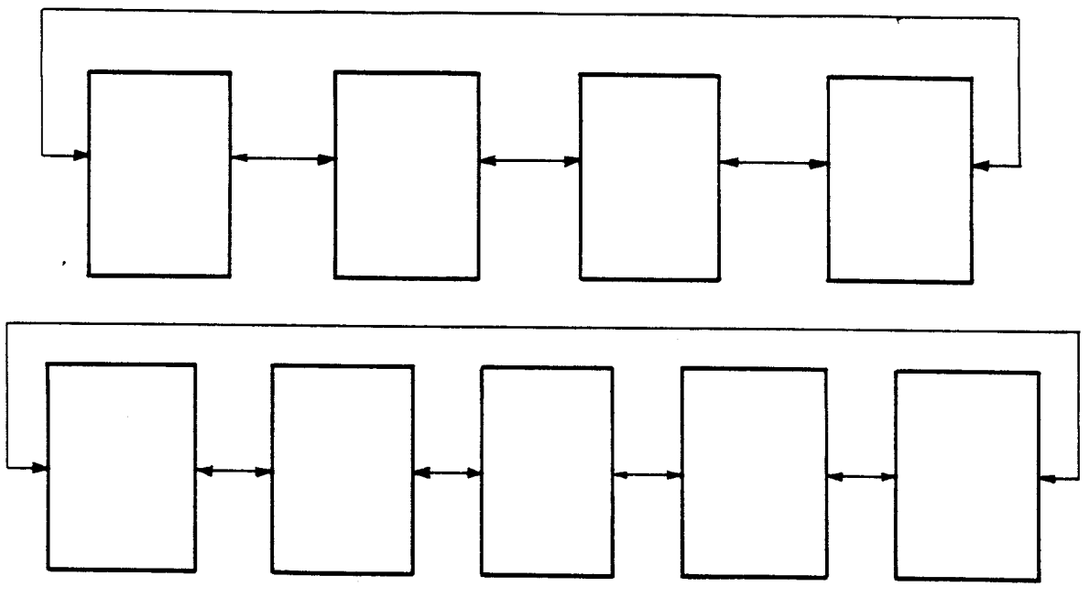


Figura 26: Expansão de um anel com $p = 4$ para $p = 5$

Se a quantidade de partículas que o usuário deseja processar aumenta, basta aumentar proporcionalmente o número de unidades de processamento para garantir que o tempo de processamento permanece aproximadamente constante.

Mais detalhes sobre a avaliação desta estrutura serão apresentados no cap. 5.

Outra possibilidade de divisão corresponderia à associação da divisão do espaço apresentada acima com uma outra divisão transversal, como na fig. 27. As considerações quanto à necessidade de comunicação permanecem as mesmas que para o caso acima, acrescentando-se apenas que deve haver comunicação também na outra direção, formando um anel bidimensional, como na fig. 28.

Esta estrutura apresenta algumas dificuldades em relação à anterior, que passaremos a citar:

1. Devido à necessidade de quatro comunicações por processador temos um maior comprometimento de *links* dos *transputers* que formam a unidade de processamento. Note que se todo o processo de comunicação fosse alocado ao mesmo processadores teríamos um comprometimento de todos os seus *links* com a comunicação com as

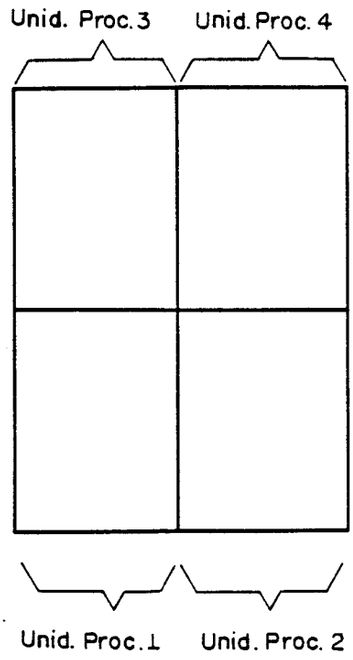


Figura 27: Divisão do espaço de simulação em duas direções

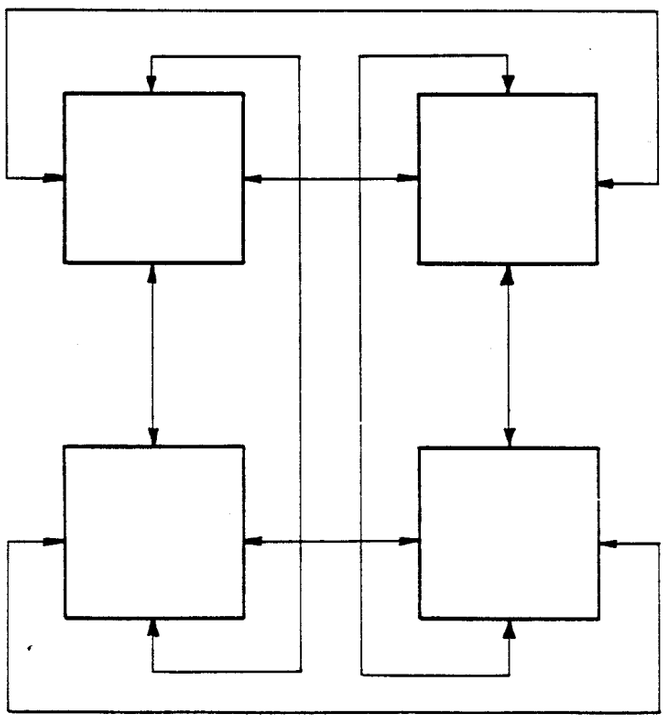


Figura 28: Anel bidimensional de processadores

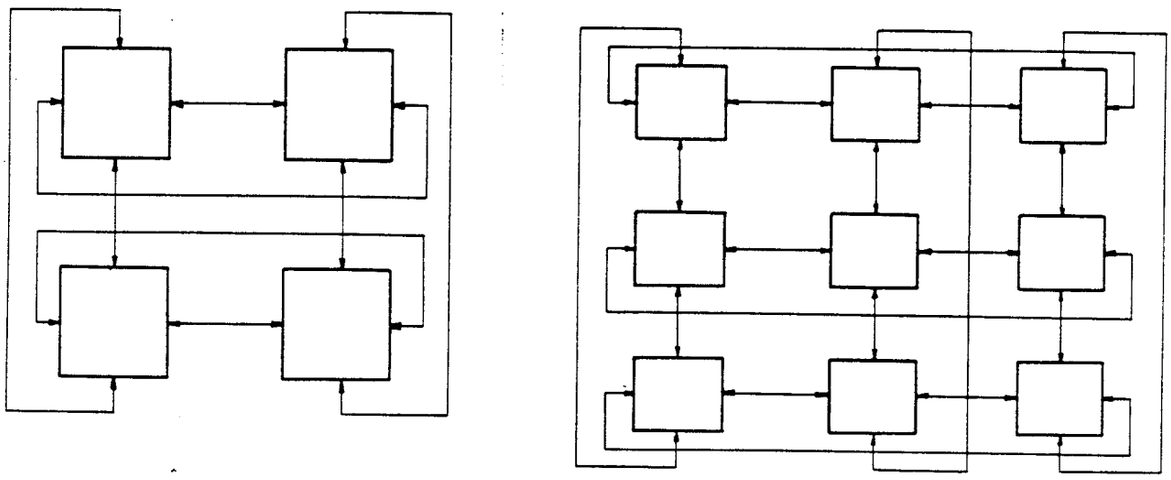


Figura 29: Expansão do anel bidimensional

unidades vizinhas, forçando a criar mecanismos externos para a comunicação com outros processadores da mesma unidade de processamento ou a utilização de apenas um processador por unidade

2. Ao desejarmos uma expansão necessitamos manter a estrutura do anel bidimensional, o que faz com que o crescimento do número de unidades de processamento somente possa ser realizado em quadrados perfeitos. A fig. 29 mostra a expansão de $p = 4$ para $p = 9$.

Tendo em vista as vantagens e a simplicidade da estrutura em anel unidimensional apresentada, terminaremos aqui a apresentação de estruturas de interligação de unidades de processamento, com a escolha do anel para a nossa proposta.

Como, para o cálculo da nova posição das partículas de um grão necessitamos das posições das partículas de todos os grãos vizinhos então, quando o grão a calcular for de um dos limites da unidade de processamento serão necessários dados de partículas da unidade de processamento vizinha. Isto implicaria na necessidade de parar o processamento e aguardar que a unidade vizinha transmita os dados necessários. Para evitar essa quebra no processamento, utilizaremos o recurso de duplicar os dados sobre grão que estão no

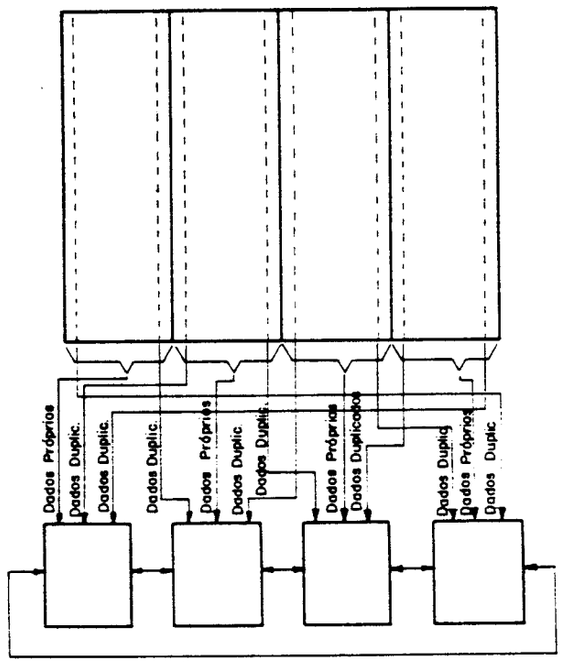


Figura 30: Duplicação dos dados de grãos dos limites

limite de uma unidade na unidade vizinha. O esquema desta duplicação pode ser visto na fig. 30. O efeito dessa alteração é possibilitar a transmissão em instantes mais apropriados pois, durante o processamento dos grãos as comunicações não são necessárias, podendo as mesmas ser realizadas em qualquer instante disponível, desde que antes de iniciado o cálculo de um novo quadro.

4.3.2 Divisão algoritmica

Dentro de uma unidade de processamento, procuraremos repartir as tarefas entre um certo número de processadores, de forma a alcançar um bom balanço de carga entre os diferentes processadores. Quando falamos em balanço de carga, nos referimos à distribuição da carga computacional pelos diferentes processadores de forma a evitar que algum processador permaneça longos períodos desocupado enquanto existem outros ocupados.

Devido à inexistência de uma rede de *transputers* ou de um simulador para ela em nosso laboratório, realizaremos o estudo de cargas baseado nos dados de desempenho apresentados em [12] e em [18].

Antes de passar à análise da utilização de recursos pelos diversos processos, devemos apresentar a estrutura de dados utilizada para o armazenamento das coordenadas das partículas. Se alocássemos um vetor para cada grão, deveríamos prever, dentro do espaço reservado a cada grão, uma capacidade de armazenamento que pudesse conter o máximo de partículas que um grão pode ter durante uma simulação. Isto significaria que deveríamos superestimar o espaço para cada grão, e, por conseguinte utilizar uma grande quantidade de memória com espaço reservado apenas à margem de segurança. Para evitar isto, realizamos o armazenamento dos dados dos grãos em listas, onde para cada grão temos um ponteiro para uma lista ligada que apresenta os números de suas partículas. Para cada partícula temos, então, um espaço reservado para suas coordenadas, seguido de um ponteiro para a próxima partícula do mesmo grão. A estrutura pode ser representada como indicado na fig. 31.

Devido ao fato de que o número de partículas em cada grão é flutuante, devemos estabelecer uma estratégia para lidar com as constantes entradas e saídas de partículas. O método escolhido foi o de manter uma lista de números de partículas disponíveis, isto é, que não estão associados a nenhuma partícula dentro desta unidade de processamento. Quando uma partícula entra na unidade de processamento a mesma recebe um número que provém desta lista. Ao sair da unidade de processamento, a partícula envia o número que ela estava utilizando para esta lista de números disponíveis. Isto permite a entrada e saída de qualquer número de partículas, desde que o total de partículas dentro da unidade de processamento não exceda a capacidade máxima estipulada para a mesma (que deve ser escolhida tendo em vista a quantidade de memória disponível).

Para que possamos sugerir uma distribuição de processamento, devemos inicialmente estabelecer quais os processos básicos, seus tempos médios de execução e sua ocupação de memória. A ocupação de memória neste caso é importante pois se conseguirmos distribuir toda a programação de cada processador dentro da memória interna

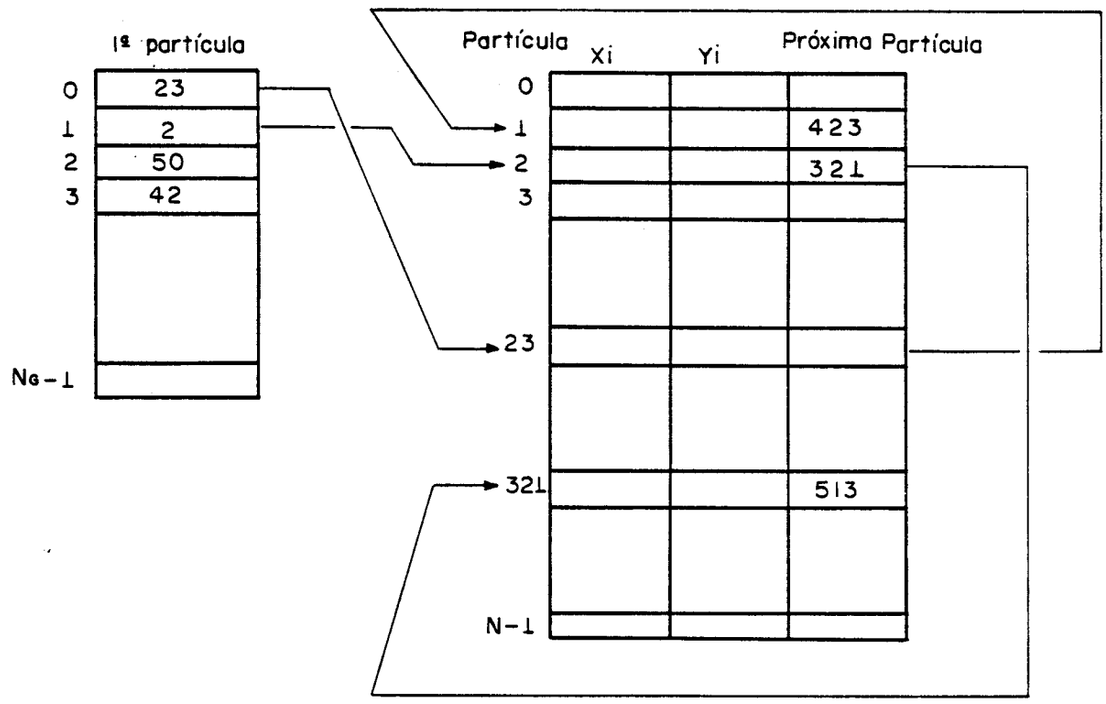


Figura 31: Estrutura para o armazenamento de dados dos grãos

do mesmo teremos uma grande vantagem em termos de tempo de execução, devido à diferença de tempos de acesso entre a memória interna e a externa em *transputers*.

Além dos processos já descritos no cap. 3 para o algoritmo de granulação grosseira, devemos incluir mais alguns processos para garantir a comunicação entre as diversas unidades de processamento, além de alterações nos processos já apresentados de forma a adaptá-los às características das redes de *transputers*.

Uma outra alteração que introduziremos se refere ao cálculo de renormalização. Como apresentado nos capítulos anteriores, a renormalização se realizava por um recálculo do valor da nova posição de cada partícula em certos quadros. Isto era feito para garantir que a energia cinética das partículas se mantinha próxima de um valor especificado. O problema com isto é que, quando um quadro necessita renormalização, precisaríamos cálcular as novas posições e as velocidades de todas as partículas para, com estes dados, encontrar o fator a ser utilizado, e só então é que poderíamos iniciar a renormalização, de forma que teríamos os seguintes problemas:

- Necessidade de término dos cálculos de todas as unidades de processamento para iniciar a renormalização

- Se a renormalização é associada com um processador, todos os demais processadores da unidade de processamento devem permanecer parados enquanto a mesma se processa

Podemos contornar essas dificuldades através do seguinte método: manter o valor das novas coordenadas como está e alterar o valor da coordenada anterior para se conformar à energia cinética requerida. Isto faz com que o cálculo do novo quadro possa começar imediatamente após o término do anterior, pois os dados renormalizados somente serão necessários quando for calculada a nova posição de uma partícula, e portanto somente neste momento precisa ser requisitado o cálculo da renormalização da posição anterior.

Para tornar mais claro os dois processos de renormalização, representaremos os mesmos utilizando a seguinte notação:

x_{a_i}: posição anterior da partícula i ao término do quadro q

x_i: posição atual da partícula i ao término do quadro q

v_i: velocidade calculada para a partícula i no quadro q

F_i: força sobre a partícula i calculada no quadro q

x'_i: posição da partícula após a renormalização

x'_{a_i}: posição anterior da partícula após a renormalização

Utilizando esta notação, podemos dizer que a renormalização utilizada nos capítulos anteriores corresponde à inclusão do cálculo:

$$x'_{a_i} = x_i \tag{39}$$

$$v'_i = (v_i - \overline{v})v_f \tag{40}$$

$$x'_i = x_i + v'_i\Delta t + \frac{F_i}{2m}\Delta t^2 \tag{41}$$

no quadro q, enquanto que o novo método corresponde à inclusão de:

$$v'_i = (v_i - \overline{v})v_f \tag{42}$$

$$x'_{a_i} = x_i - v'_i\Delta t - \frac{F_i}{2m}\Delta t^2 \tag{43}$$

$$x'_i = x_i \tag{44}$$

no quadro $q+1$, onde:

q: é o número de qualquer quadro que necessite de renormalização.

v_i: é a velocidade da partícula i antes da renormalização

v'_i: é a velocidade renormalizada da partícula i

$\overline{v}$: é a velocidade média das partículas antes da renormalização

v_f: é o fator de renormalização das velocidades

Δt: é o intervalo de tempo entre dois quadros sucessivos

m: é a massa das partículas

O novo elenco de processos fica então o seguinte:

cont.list.arm: responsável pelo armazenamento das coordenadas atuais das partículas e pelo controle das listas de partículas de cada grão

cont.aux: o mesmo que o anterior, só que para os grãos duplicados, isto é, os grãos das unidades de processamento vizinhas que aparecem repetidos dentro desta unidade de processamento

calc.grao: calcula o grão a que uma partícula (já selecionada como sendo desta unidade de processamento) pertence

calc.g.aux: calcula a qual grão duplicado uma partícula pertence

quadros: controla a execução de todos os quadros requisitados; para cada quadro, fornece a seqüência de varredura dos grãos

vizinhos: dado um grão, encontra todos os vizinhos do mesmo que devem ser varridos

pares: fornecidas as partículas de um grão e de seus vizinhos, este processo gera todos os pares necessários ao cálculo das partículas do grão central. Inclui também o teste com o raio de corte (veja no próximo ítem)

teste.dis: é apenas uma rotina auxiliar ao processo **pares**. Testa se a distância entre as partículas de um par é menor que o raio de corte e envia o par para processamento caso positivo

acel: calcula o fator comum entre as acelerações em cada direção para cada par de partículas

poten: calcula e acumula a energia potencial de um par de partículas

ac.coord: calcula a aceleração em cada uma das direções

calc.coord: calcula as novas coordenadas de uma partícula

veloc: calcula a velocidade de uma partícula

cinet: calcula e acumula energia cinética das partículas

renorm: realiza a renormalização quando necessário; armazena as coordenadas anteriores das partículas

energias: junta os valores de energia das diversas unidades de processamento e os envia

envia: verifica se a partícula pertence a esta unidade de processamento; envia os dados da mesma para uma das unidades vizinhas, caso necessário

envio.dados: quando os dados de uma partícula devem ser comunicados a uma unidade de processamento vizinha, este processo se encarrega da tarefa

cont.fila: realiza o controle da lista de números de partícula disponíveis

recep: recebe os diferentes dados vindos pelos *links* que ligam esta unidade de processamento a suas vizinhas e os envia para os processos que os requerem

junta.ext: junta as partículas vindas das unidades de processamento à esquerda e à direita

A interligação entre os processos pode ser apresentada como na fig. 32, utilizando a mesma notação do cap. 3.

Para uma avaliação, apresentamos a codificação em OCCAM dos processos acima e realizamos uma estimativa de seus tempos de execução. O código desses processos pode ser encontrado no apêndice B.

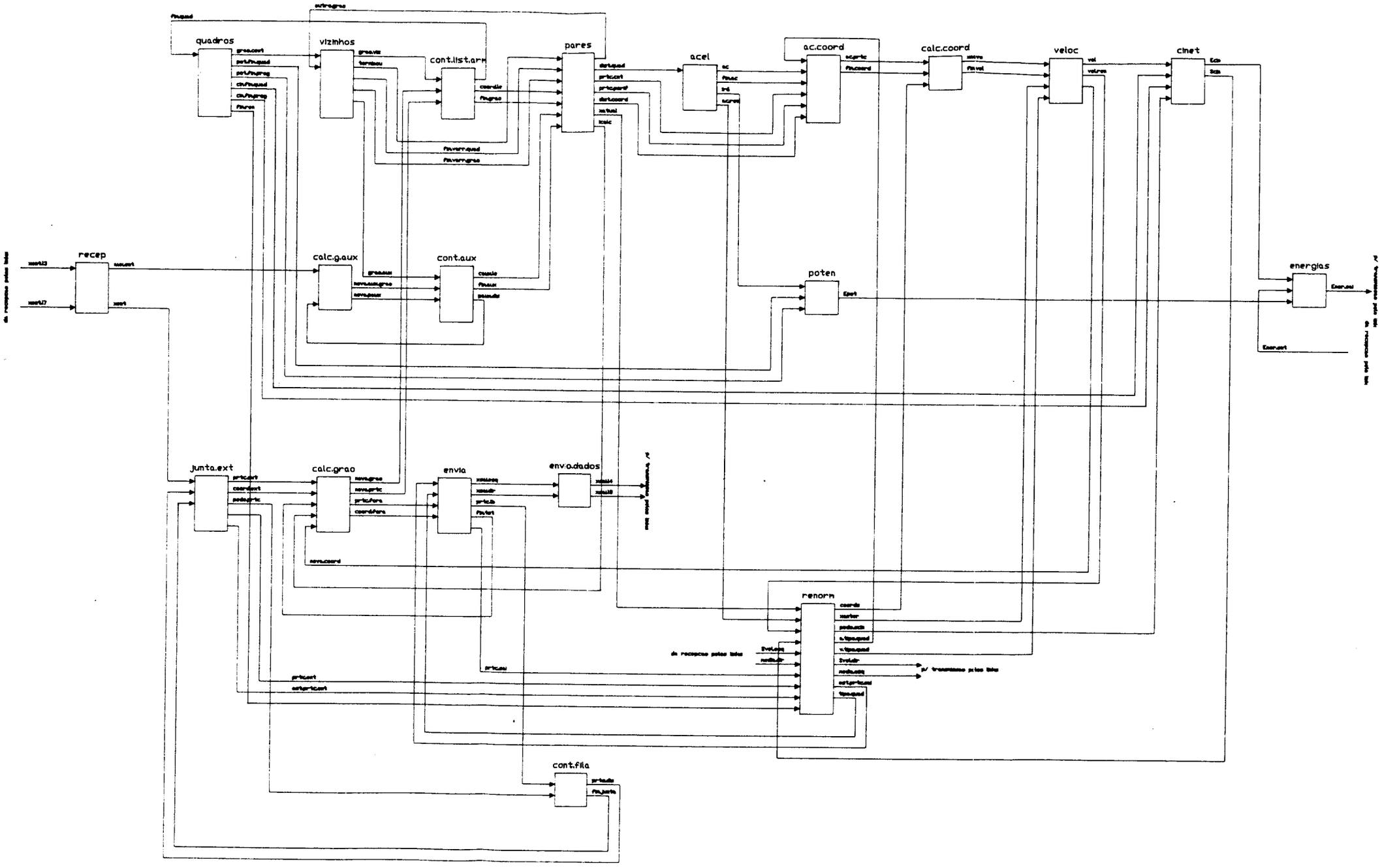

quadros
vizinhos
cont.list.arr
pares
acel
ac.coord
calc.coord
veloc
cinet
recep
calc.g.aux
cont.aux
poten
energias
junta.ext
calc.grao
envia
envio.dados
renorm
cont.fila

Figura 32: Estrutura do programa incluindo processos para comunicação entre as unidades de processamento

Um fato adicional que deve ser levado em consideração antes de decidirmos a distribuição dos processos é o da grande quantidade de dados que circula entre os processos de formação de pares e o de teste da distância com o raio de corte. Para reduzir o tempo de execução, reunimos esses dois processos em um só, de forma que eliminamos a necessidade de transmissão desses dados por canais.

Os tempos de execução dos diversos processos é variável fundamentalmente com os seguintes parâmetros:

1. Número total de partículas no sistema
2. Densidade de distribuição das partículas
3. Número de unidades de processamento

Como se tornaria inconveniente realizar uma distribuição de processos para cada variação em alguns desses parâmetros, realizaremos os cálculos apenas para um caso típico onde a densidade reduzida de partículas é $\varrho = 0,6$ (reduzida pois está expressa em unidades de comprimento onde $\sigma = 1$, veja cap. 1), com distribuição em uma rede triangular, e 8064 partículas, considerando a existência de apenas uma unidade de processamento. Consideraremos ainda que a renormalização é necessária a cada cinco quadros, e calcularemos os tempos para uma simulação englobando 1000 quadros. Partindo destes valores, podemos encontrar os tempos de execução apresentados na tabela VI, onde são apresentados, além dos tempos em segundos também a porcentagem de cada processo no tempo total. Uma descrição mais completa do método utilizado para encontrar estes valores de tempo será apresentada na seção 5.2.

Com relação à ocupação de memória, uma avaliação nos leva à tabela VII, onde expressamos o número de bytes ocupados por cada processo na memória interna. É claro que o número de bytes efetivamente ocupado só pode ser um valor inteiro, porém, como estamos utilizando os valores médios apresentados pelo fabricante, estes apresentam

Tabela VI: Tempos de execução dos processos

Processo	Tempo (hs)	Porcentagem do tempo
cont.list.arm	0,1185471	7,128
cont.aux	0,0034130	0,205
calc.grao	0,0769328	4,626
calc.g.aux	0,0021036	0,126
quadros	0,0013305	0,080
vizinhos	0,0486294	2,924
pares	0,3297801	19,828
teste.dis	0,3178957	19,119
acel	0,1477145	8,881
poten	0,0936841	5,633
ac.coord	0,3644220	21,911
calc.coord	0,0346864	2,085
veloc	0,0380336	2,287
cinet	0,0234329	1,409
renorm	0,0585747	3,522
energias	0,0000035	0,000
envia	0,0009875	0,059
envio.dados	0,0015989	0,096
cont.fila	0,0000321	0,002
recep	0,0012644	0,076
junta.ext	0,0000451	0,003

a precisão de uma casa decimal, como mostrado na tabela. O espaço ocupado em memória externa não foi especificado.

Devido à grande porcentagem de tempo necessária ao processo que gera os pares e testa a distância, percebemos que a melhor divisão é a realizada em dois processadores apenas. Dividiremos então os processos em dois processadores da seguinte forma:

Processador 1: cont.list.arm, cont.aux, calc.g.aux, quadros, vizinhos, pares, teste.dis, envio.dados

Processador 2: calc.grão, acel, poten, ac.coord, calc.coord, veloc, cinet, renorm, energias, envia, cont.fila, recep, junta.ext

O que nos dá a seguinte divisão de tempo de processamento:

Processador 1: 0,840 horas (50,5 % da carga)

Processador 2: 0,823 horas (49,5 % da carga)

e de ocupação de memória interna:

Processador 1: 3373,5 bytes

Processador 2: 3740,1 bytes

Apesar dos dados apresentados acima tratarem de um caso específico, veremos, no capítulo 5 que a distribuição escolhida apresenta bons resultados mesmo para outros valores.

4.4 Escolha dos parâmetros

Desenvolveremos agora a forma de encontrar, dado um problema de dinâmica molecular e o número de divisões de processamento disponíveis, os parâmetros para a realização da divisão do processamento entre os processadores.

Os parâmetros a serem definidos são os seguintes:

Tabela VII: Ocupação de memória pelos vários processos

Processo	Memória Interna (bytes)
cont.list.arm	229,7
cont.aux	267,7
calc.grao	422,5
calc.g.aux	163,5
quadros	108,8
vizinhos	232,2
pares	1421,8
teste.dis	134,8
acel	107,9
poten	114,1
ac.coord	335,4
calc.coord	132,4
veloc	175,8
cinet	197,5
renorm	1005,5
energias	106,3
envia	343,1
envio.dados	815,0
cont.fila	137,9
recep	456,6
junta.ext	205,1

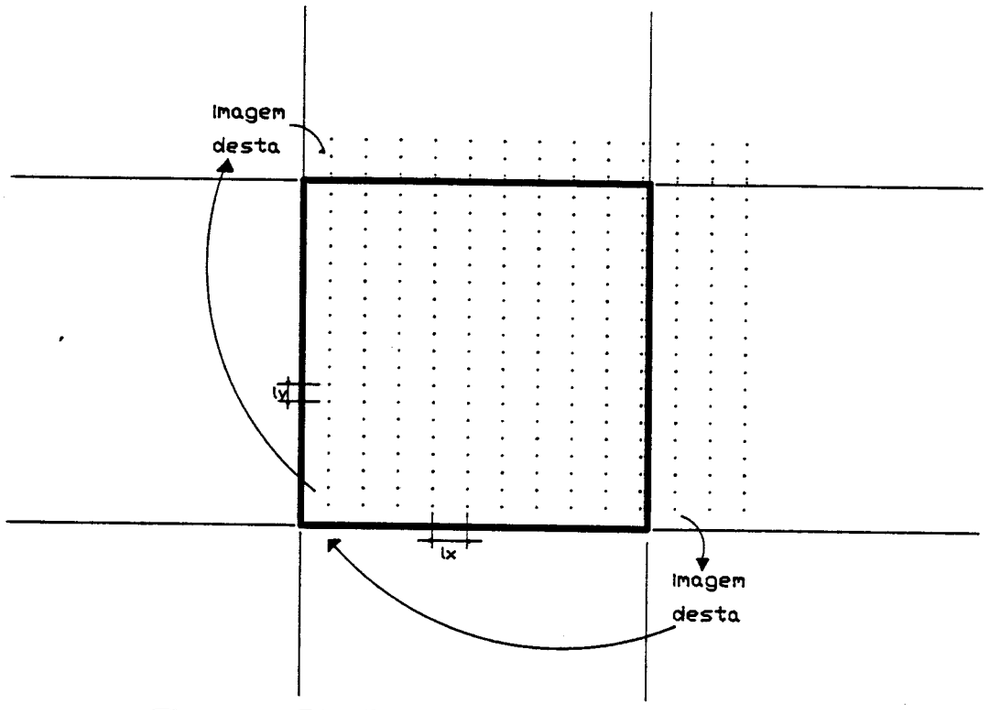


Figura 33: Distribuição inicial das partículas em rede

1. Dimensões do grão

2. Número de grãos por unidade de processamento

Definidos esses parâmetros, podemos determinar quais partículas pertencem a quais grãos e quais grãos a quais unidades de processamento.

Um outro fator de que precisamos para determinar os parâmetros acima é o tipo de distribuição inicial das partículas. Normalmente isto é feito através de uma estrutura regular tipo rede. Precisamos conhecer então o comprimento do período de repetição deste padrão em cada uma das direções, que chamaremos de l_x e l_y. Notemos que o comprimento total do espaço de simulação em cada uma das direções deve ser um múltiplo do correspondente período, para garantir a exatidão da periodicidade do espaço. Veja um exemplo disso na fig. 33, onde utilizamos uma rede retangular.

Um outro fator a ser fornecido para definirmos os parâmetros necessários é o número de repetições do padrão básico da rede em cada uma das direções (chamamos de padrão básico da rede o elemento que se apresenta repetidamente na mesma). Isto também

pode ser visto na fig. 33. Temos então:

$$k_x = \frac{L_x}{l_x} \tag{45}$$

$$k_y = \frac{L_y}{l_y} \tag{46}$$

$$N = k_x k_y n \tag{47}$$

onde:

N é o número total de partículas

k_x é o número de repetições do padrão da rede na direção x

k_y é o número de repetições do padrão da rede na direção y

L_x é o comprimento do espaço na direção x

L_y é o comprimento do espaço na direção y

n é o número de partículas em cada repetição do padrão da rede

Suporemos que o usuário apresenta um problema onde já está definida a configuração inicial de partículas, isto é, são dados os valores de l_x, l_y, k_x, k_y e n (sendo estes escolhidos compatível com o tipo de rede utilizada). Com estes dados, podemos encontrar o número de grãos em cada direção, que chamaremos de n_x e n_y, da seguinte forma:

$$u_x = \lceil \frac{L_x}{r_c p} \rceil \tag{48}$$

$$n_x = u_x p \tag{49}$$

$$n_y = \lceil \frac{L_y}{r_c} \rceil \tag{50}$$

onde:

r_c é o valor do raio de corte

u_x é o número de grão da direção x em cada unidade de processamento

p é o número de unidades de processamento

Na eq. 48, dividimos também por p para repartir, inicialmente, o espaço de simulação entre as unidades de processamento, de forma a que cada uma receba em média uma carga igual. Veja que a eq. 49 temos a garantia de que n_x é múltiplo de p, de forma que os grão podem ser divididos igualmente entre as unidades de processamento.

Com estes valores calculados, podemos encontrar as dimensões dos grãos em cada direção:

$$g_x = \frac{L_x}{n_x} \tag{51}$$

$$g_y = \frac{L_y}{n_y} \tag{52}$$

onde:

g_x: é o tamanho do grãos na direção x

g_y: é o tamanho do grãos na direção y

Como exemplo, calculemos estes fatores para o seguinte caso:

- Rede triangular, com $\varrho = 0,6$, $n = 1$ e:

$$l_x = \sqrt{\frac{2}{\sqrt{3}\varrho}} \tag{53}$$

$$= 1,387 \tag{54}$$

$$l_y = \sqrt{\frac{\sqrt{3}}{2\varrho}} \tag{55}$$

$$= 1,201 \tag{56}$$

- Repetições de $k_x = 84$ e $k_y = 96$
- Três unidades de processamento ($p = 3$)

Com estes valores encontramos:

$$u_x = \left\lceil \frac{84 \cdot 1,387}{2,5 \cdot 3} \right\rceil$$

$$= 15 \tag{57}$$

$$n_x = 15 \cdot 3$$

$$= 45 \tag{58}$$

$$n_y = \left\lceil \frac{96 \cdot 1,201}{2,5} \right\rceil$$

$$= 46 \tag{59}$$

Portanto, a simulação consistiria em um arranjo de 45 × 46 grãos, sendo que a cada unidade de processamento corresponde um arranjo de 15 × 46 grãos.

Capítulo 5

Avaliação da proposta

5.1 Introdução

Apresentaremos neste capítulo uma avaliação da arquitetura proposta, sendo mantido o método adotado no cap. 4.

Realizaremos a avaliação através do estudo do comportamento do sistema proposto com relação à variação de três de seus parâmetros: número de partículas (N), densidade reduzida de partículas (ϱ) e número de unidades de processamento (p). Utilizaremos, em todos os cálculos, uma rede triangular de partículas. Os seguintes fatores foram mantidos constantes em todos os cálculos:

- 1000 quadros de simulação
- renormalização a cada 5 quadros

Com vistas a facilitar a interpretação dos dados, introduziremos alguns índices:

Tempo de execução representado por T e definido por:

$$T = \max(T_1, T_2)$$

onde:

T_1 é o tempo de processamento do processador 1

T_2 é o tempo de processamento do processador 2

Distribuição de carga representada por d e definida pela expressão:

$$d = \frac{\min(T_1, T_2)}{T}$$

Veja que o valor ideal para este índice é 1

Aceleração indicando o ganho de velocidade de execução quando acrescentamos mais unidades de processamento, e representado por a_k:

$$a_k = \frac{T_{p=1}}{T_{p=k}}$$

onde:

a_k é a aceleração com k unidades de processamento

$T_{p=1}$ é o valor de T quando $p = 1$

$T_{p=k}$ é o valor de T quando $p = k$

O valor ideal para este índice é k.

Eficiência designada e e definida por:

$$e = \frac{a_p}{p}$$

que representa a taxa de utilização dos recursos extras (em relação a uma única unidade de processamento)

5.2 Descrição da metodologia

Para o levantamento dos valores de tempo de execução utilizados neste capítulo e no capítulo 4, realizaremos uma estimativa baseada em dados de número de ciclos de execução médios da programação em OCCAM apresentados pelo fabricante dos *transputers* na referência [12], complementados por dados específicos do T800 apresentados em [18]. A

estimativa foi realizada avaliando, para cada uma das linhas de código OCCAM de cada processo o tempo de execução das mesmas, em número de ciclos de máquina, bem como o número médio de execuções que ocorrerão durante uma simulação, em função de alguns parâmetros da simulação como número de partículas, densidade e número de unidades de processamento.

Realizando um levantamento minucioso encontramos os seguintes resultados para os tempos dos diversos processos:

cont.list.arm: chamando de t_1 ao seu tempo de execução, temos:

$$t_1 = 218936,7 + (798127,6 + 849000,0\overline{n_g})u_x n_y \tag{60}$$

cont.aux: chamando de t_2 ao seu tempo de execução, temos:

$$t_2 = 150146,1 + (810278,2 + 1188294,6\overline{n_g})n_y \tag{61}$$

calc.grao: chamando de t_3 ao seu tempo de execução, temos:

$$t_3 = 379527,6 + ((684200,0u_x + 102600,0)\overline{n_g} + 988400,0\overline{n_c})n_y \tag{62}$$

calc.g.aux: chamando de t_4 ao seu tempo de execução, temos:

$$t_4 = 30537,0 + 863800,0n_y\overline{n_g} \tag{63}$$

quadros: chamando de t_5 ao seu tempo de execução, temos:

$$t_5 = 153029,5 + 45200,0u_x n_y \tag{64}$$

vizinhos: chamando de t_6 ao seu tempo de execução, temos:

$$t_6 = 96147,0 + (1666300,0n_y - 32400,0)u_x - 503900,0n_y \tag{65}$$

pares: chamando de t_7 ao seu tempo de execução, temos:

$$\begin{aligned} t_7 \;=\; & 255508,0 + (687600,0 + 903800,0\overline{n_g} + 480800,0\overline{n_g}^2)u_x n_y + \\ & + (363000,0 + 244300,0\overline{n_g} + 248400,0\overline{n_g}^2)n_y \end{aligned} \tag{66}$$

teste.dis: chamando de t_8 ao seu tempo de execução, temos:

$$t_8 = (133700,0\overline{n_v} + 87100,0\overline{n_i})u_x n_y \overline{n_g} \tag{67}$$

acel: chamando de t_9 ao seu tempo de execução, temos:

$$t_9 = 58,8 + 223900,0 u_x n_y \overline{n_g} \overline{n_i} \tag{68}$$

poten: chamando de t_{10} ao seu tempo de execução, temos:

$$t_{10} = 131803,7 + 142000,0 u_x n_y \overline{n_g} \overline{n_i} \tag{69}$$

ac.coord: chamando de t_{11} ao seu tempo de execução, temos:

$$t_{11} = 86761028,0 + (234260,0 + 481700,0\overline{n_i})u_x n_y \overline{n_g} \tag{70}$$

calc.coord: chamando de t_{12} ao seu tempo de execução, temos:

$$t_{12} = 127,7 + 309700,0 u_x n_y \overline{n_g} \tag{71}$$

veloc: chamando de t_{13} ao seu tempo de execução, temos:

$$t_{13} = 47219,8 + 339580,0 u_x n_y \tag{72}$$

cinet: chamando de t_{14} ao seu tempo de execução, temos:

$$t_{14} = 181084,9 + 209200,0 u_x n_y \overline{n_g} \tag{73}$$

renorm: chamando de t_{15} ao seu tempo de execução, temos:

$$t_{15} = 18903489,0 + 883720,0 n_y \overline{n_c} + 485980,0 u_x n_y \overline{n_g} \tag{74}$$

energias: chamando de t_{16} ao seu tempo de execução, temos:

$$t_{16} = 253929,7 \tag{75}$$

envia: chamando de t_{17} ao seu tempo de execução, temos:

$$t_{17} = 166360,4 + (388600,0\overline{n_g} + 813880,0\overline{n_c})n_y \quad (76)$$

envio.dados: chamando de t_{18} ao seu tempo de execução, temos:

$$t_{18} = 631428,9 + (818120,0\overline{n_c} + 637000,0\overline{n_g})n_y \quad (77)$$

cont.fila: chamando de t_{19} ao seu tempo de execução, temos:

$$t_{19} = 30569,7 + 528200,0n_y\overline{n_c} \quad (78)$$

recep: chamando de t_{20} ao seu tempo de execução, temos:

$$t_{20} = 441199,6 + (503800,0\overline{n_g} + 660080,0\overline{n_c})n_y \quad (79)$$

junta.ext: chamando de t_{21} ao seu tempo de execução, temos:

$$t_{21} = 271010,9 + 878000,0\overline{n_c}n_y \quad (80)$$

onde:

$\overline{n_g}$: número médio de partículas por grão, avaliado em termos da densidade reduzida de partículas:

$$\overline{n_g} = g_x g_y \varrho \quad (81)$$

u_x: já definido no cap. 4 (veja eq. 48)

n_y: definido juntamente com u_x

$\overline{n_c}$: é o valor de quantas partículas em média serão testadas para verificar se interagem com cada partícula. Pode ser avaliado considerando-se o número médio de partículas por grão ($\overline{n_g}$), o número médio de grãos vizinhos de cada grão $(4,5)$ e o fato de que os grãos das bordas devem realizar interações com os grão duplicados (isto é, aqueles copiados de uma unidade de processamento vizinha). A expressão fica:

$$\overline{n_c} = (4,5 + \frac{3}{u_x})\overline{n_g} \quad (82)$$

$\overline{n_v}$: é um valor avaliado para o número médio de partículas, por grão, que cruzarao deste grão para um vizinho à esquerda (ou à direita). Utilizado para avaliar a quantidade de comunicações entre unidades de processamento. Este parâmetro é dependente da densidade

$\overline{n_i}$: número médio de partículas cuja interação com uma dada partícula precisa ser calculada. Avaliamos este parâmetro pela seguinte equação:

$$\overline{n_i} = \frac{\pi r_c^2}{2} \varrho \tag{83}$$

Consideramos também os seguintes parâmetros fixos:

- 1000 quadros de simulação
- 16384 partículas no máximo por unidade de processamento
- 1024 partículas no máximo nos grãos repetidos
- renormalização a cada cinco quadros

Os valores máximos de números de partículas são importantes devido aos processos de inicialização de áreas de memória.

Seguimos agora com a apresentação dos resultados.

5.3 Variação de N, fixos ϱ e p

Como este estudo já foi realizado para a operação seqüencial, restringiremos a apresentação aqui a um caso altamente paralelo, com $p = 8$. A tabela VIII apresenta o efeito da variação no número de partículas assumindo-se os valores de $\varrho = 0,6$ e $p = 8$. São apresentados os valores de T_1, T_2, T (todos em horas) e d. Desta tabela extraímos o gráfico da fig. 34, onde apresentamos os valores de T.

Na tabela IX e no gráfico da fig. 35 temos o mesmo estudo com $\varrho = 0,82$.

Tabela VIII: Variações em N com $\varrho = 0,6$ e $p = 8$

N	T_1	T_2	T	d
2016	0,052	0,042	0,052	0,81
4032	0,078	0,073	0,078	0,94
6048	0,104	0,101	0,104	0,97
8064	0,129	0,125	0,129	0,97
10080	0,155	0,150	0,155	0,97

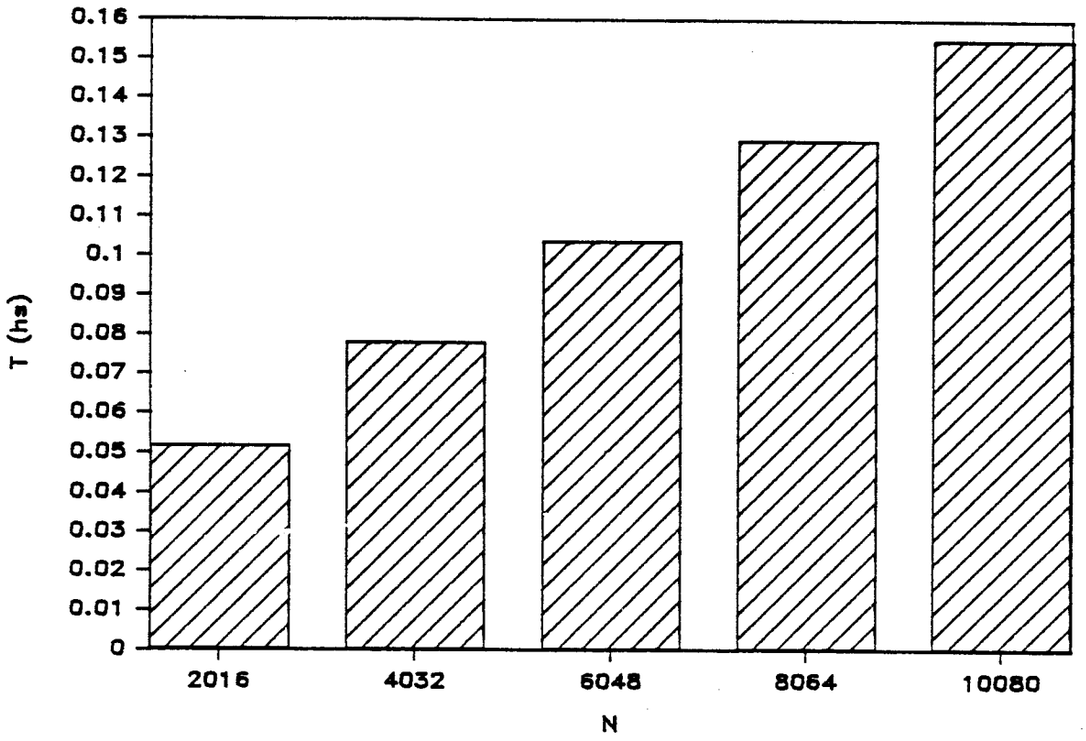


Figura 34: Gráfico de $T \times N$ para $\varrho = 0,6$ e $p = 8$

Tabela IX: Variação de N com $\varrho = 0,82$ e $p = 8$

N	T_1	T_2	T	d
2016	0,058	0,048	0,058	0,83
4032	0,091	0,081	0,091	0,89
6048	0,123	0,113	0,123	0,92
8064	0,155	0,165	0,165	0,94
10080	0,187	0,188	0,188	0,99

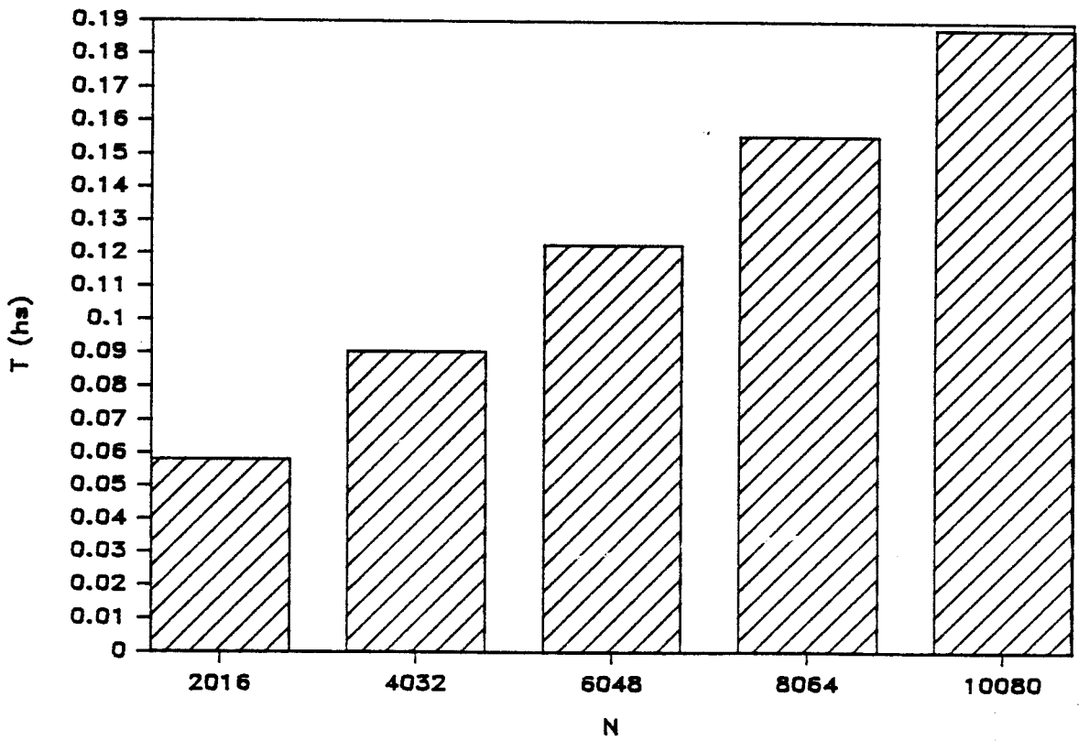


Figura 35: Gráfico de $T \times N$ para $\varrho = 0,82$ e $p = 8$

Como podemos notar pelos dados acima, o crescimento de T com N para o caso de $p = 8$ é menor que linear. Por exemplo, para $\varrho = 0,6$, quando $N = 2016$ temos $T = 0,052$ e multiplicando N por quatro (para $N = 8064$), temos $T = 0,129$, isto é, um crescimento de $2,48$. A explicação para este fato é a relativa diminuição na duplicação de cálculos e dados pelas diversas unidades de processamento. De fato, no caso de $p = 8$, a duplicação de dados para $N = 2016$ é de 100%, pois encontraremos o valor $k_x = 2$, o que implica que todos os grão apareceram tanto na sua unidade de processamento própria quanto na da esquerda ou da direita (pois todos os grãos estão nos limites, não existem grãos centrais). É claro que, com o crescimento de N teremos um crescimento de k_x, o que implica em uma menor duplicação de dados.

5.4 Variação de p, fixos N e ϱ

Estudaremos agora o efeito da variação do número de unidades de processamento, dados o número de partículas e a densidade da distribuição das mesmas. Veremos então o efeito da inclusão de mais processadores para um problema fixo.

Na tabela X apresentamos os resultados para T_1, T_2, T, d e a_k quando $N = 2016$ e $\varrho = 0,6$. Na tabela XI apresentamos o caso de $\varrho = 0,82$.

Os mesmos cálculos realizados para 2016 partículas foram realizados também para 8064 (tabelas XII e XIII) e 32256 (tabelas XIV e XV) partículas. No caso das tabelas com 32256 partículas, não foram realizados os cálculos para $p = 1$ pois isto implicaria em uma quantidade muito grande de partículas em uma única unidade de processamento. Para o cálculo dos valores a_k e e, onde necessitamos o valor de $T_{p=1}$, supusemos $T_{p=1} = 2T_{p=2}$.

Para facilitar a análise, apresentamos também gráficos dos valores de T com relação à variação de p. Nas figuras 36 e 37 apresentamos o caso de $N = 2016$, nas figuras 38 e 39 temos o de $N = 8064$ e nas figuras 40 e 41 o de $N = 32256$.

Tabela X: Efeitos da variação de p com $N = 2016$ e $\varrho = 0,6$

p	T_1	T_2	T	d	a_k	e
1	0,229	0,208	0,229	0,91	1,00	1,00
2	0,127	0,109	0,127	0,86	1,80	0,90
3	0,094	0,077	0,094	0,82	2,44	0,81
4	0,077	0,061	0,077	0,79	2,97	0,74
8	0,052	0,039	0,052	0,75	4,40	0,55
16	0,039	0,024	0,039	0,62	5,87	0,37

Tabela XI: Efeitos da variação de p com $N = 2016$ e $\varrho = 0,82$

p	T_1	T_2	T	d	a_k	e
1	0,280	0,261	0,280	0,93	1,00	1,00
2	0,153	0,139	0,153	0,91	1,83	0,92
3	0,111	0,095	0,111	0,86	2,52	0,84
4	0,090	0,080	0,090	0,89	3,11	0,78
8	0,058	0,044	0,058	0,76	4,83	0,61
16	0,042	0,027	0,042	0,64	6,67	0,42

Tabela XII: Efeitos da variação de p com $N = 8064$ e $\varrho = 0,6$

p	T_1	T_2	T	d	a_k	e
1	0,840	0,823	0,840	0,98	1,00	1,00
2	0,433	0,419	0,433	0,96	1,94	0,97
3	0,298	0,287	0,298	0,96	2,82	0,94
4	0,230	0,221	0,230	0,96	3,65	0,91
8	0,129	0,125	0,129	0,97	6,51	0,81
16	0,079	0,085	0,085	0,93	9,88	0,62

Tabela XIII: Efeitos da variação de p com $N = 8064$ e $\varrho = 0,82$

p	T_1	T_2	T	d	a_k	e
1	1,045	1,003	1,045	0,96	1,00	1,00
2	0,537	0,519	0,537	0,97	1,95	0,98
3	0,367	0,347	0,367	0,95	2,85	0,95
4	0,282	0,280	0,282	0,99	3,71	0,93
8	0,155	0,165	0,165	0,94	6,33	0,79
16	0,092	0,095	0,095	0,97	11,00	0,69

Tabela XIV: Efeitos da variação de p com $N = 32256$ e $\varrho = 0,6$

p	T_1	T_2	T	d	a_k	e
1	n.r.	n.r.				
2	1,655	1,647	1,655	1,00	2,00	1,00
3	1,113	1,101	1,113	0,99	2,97	0,99
4	0,842	0,837	0,842	0,99	3,93	0,98
8	0,436	0,442	0,442	0,99	7,49	0,94
16	0,234	0,250	0,250	0,94	13,24	0,83

Tabela XV: Efeitos da variação de p com $N = 32256$ e $\varrho = 0,82$

p	T_1	T_2	T	d	a_k	e
1	n.r.	n.r.	n.r.			
2	2,066	2,008	2,066	0,97	2,00	1,00
3	1,388	1,351	1,388	0,97	2,98	0,99
4	1,049	1,039	1,049	0,99	3,94	0,99
8	0,540	0,559	0,559	0,97	7,39	0,92
16	0,286	0,329	0,329	0,87	12,56	0,79

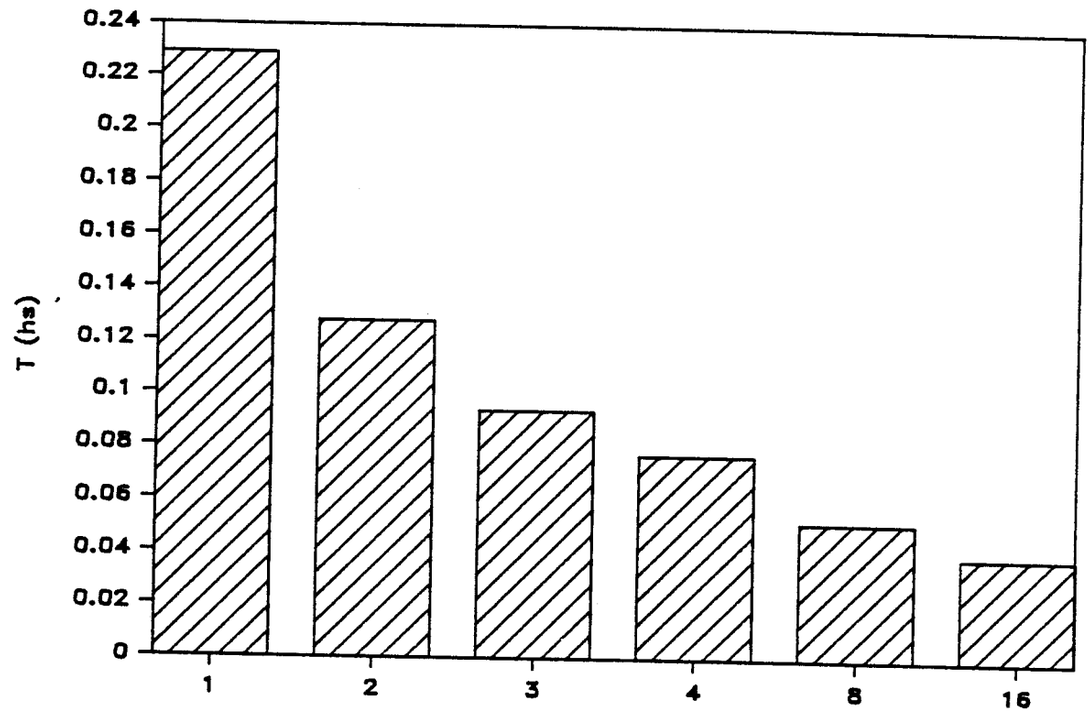


Figura 36: Variação de T com p para $N = 2016$ e $\varrho = 0,6$

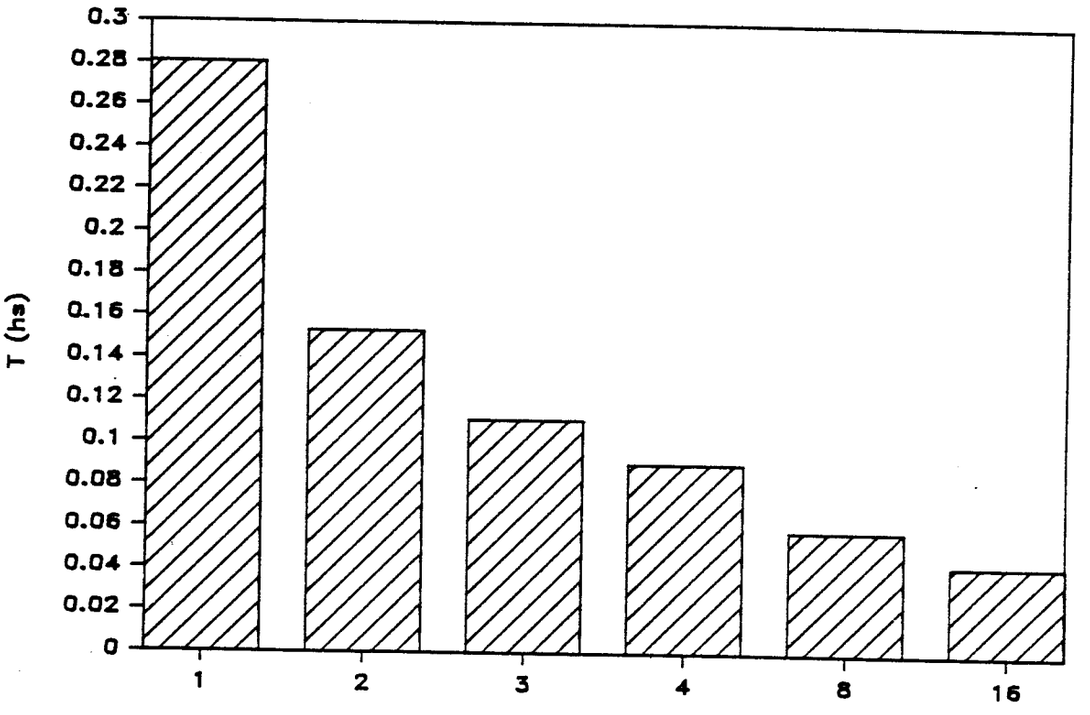


Figura 37: Variação de T com p para $N = 2016$ e $\varrho = 0,82$

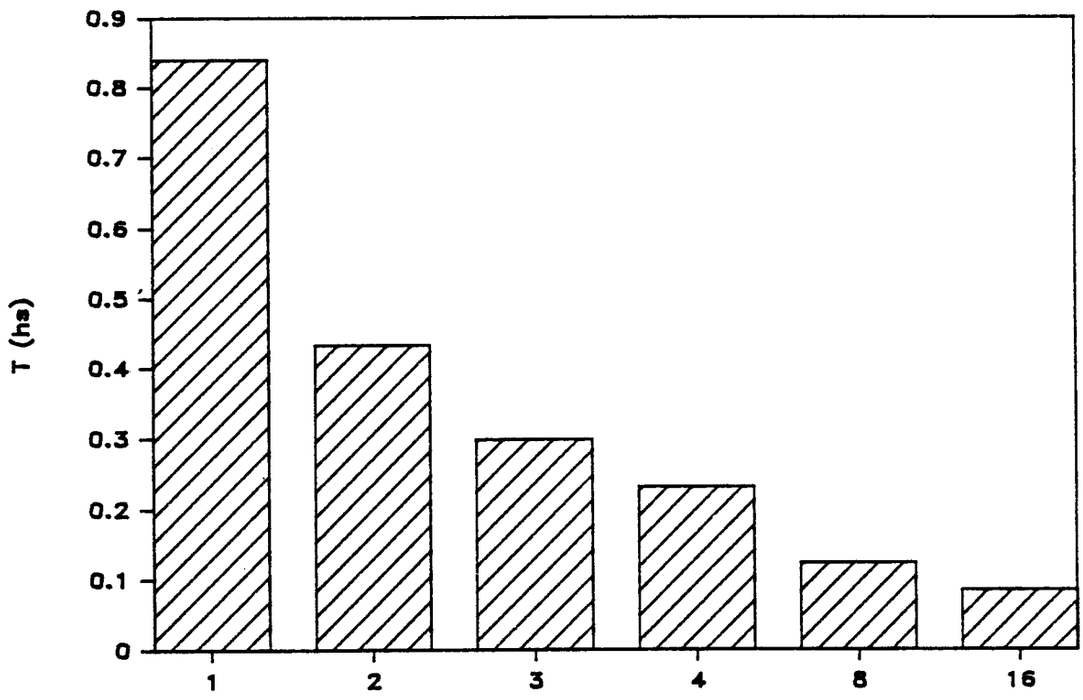


Figura 38: Variação de T com p para $N = 8064$ e $\varrho = 0,6$

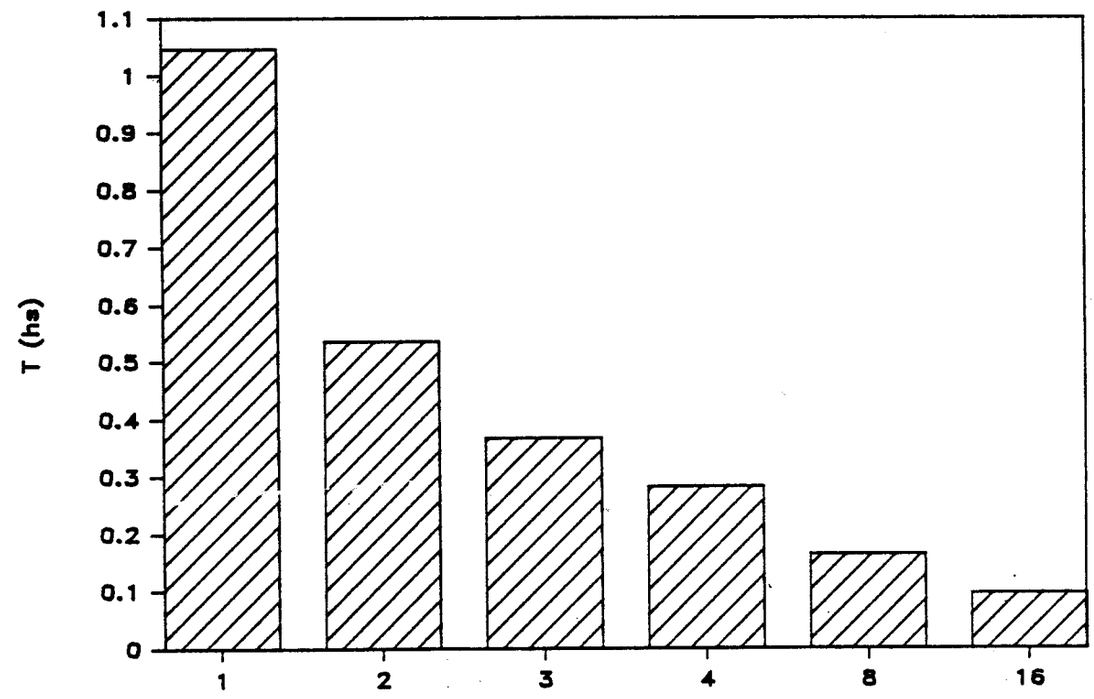


Figura 39: Variação de T com p para $N = 8064$ e $varrho = 0,82$

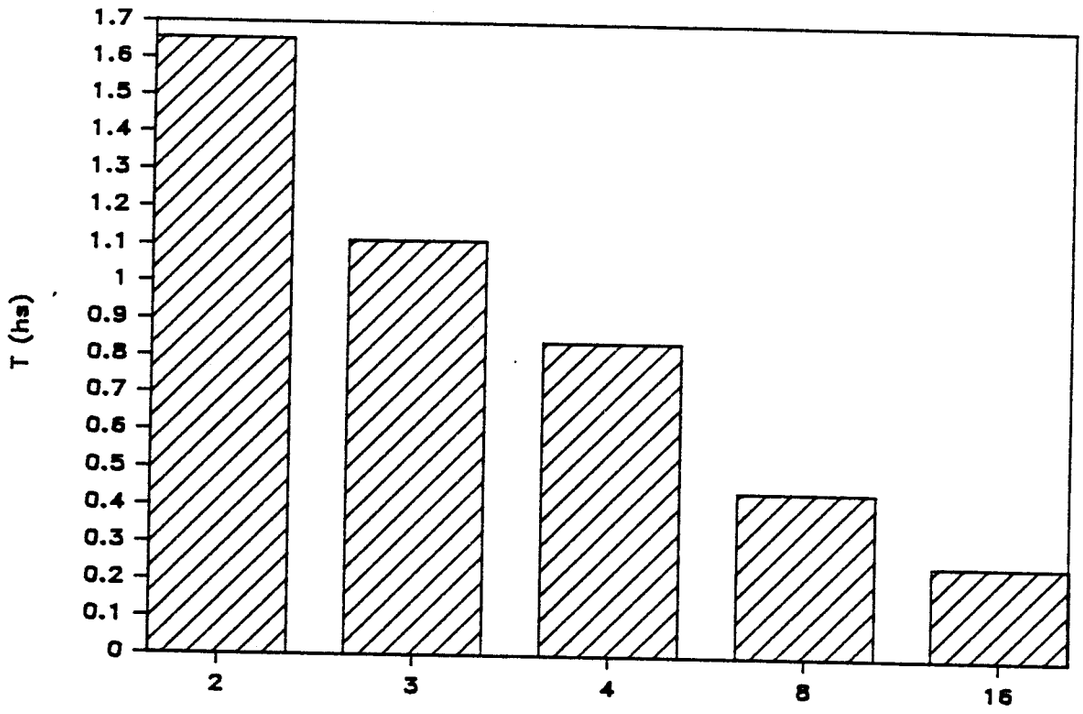


Figura 40: Variação de T com p para $N = 32256$ e $varrho = 0,6$

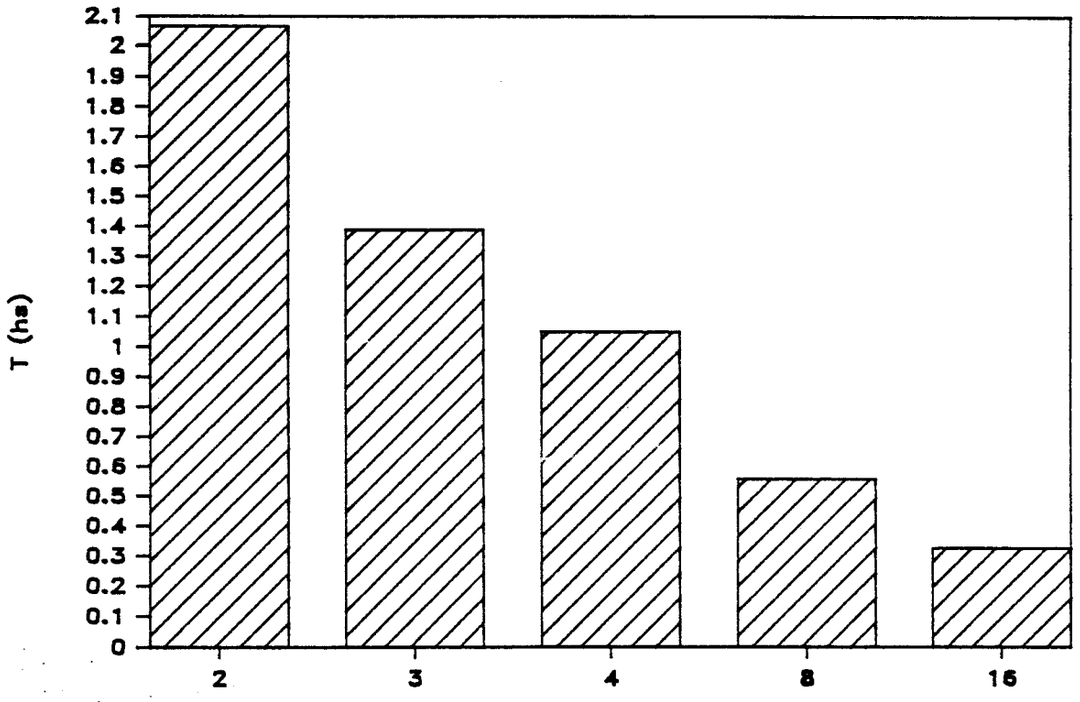


Figura 41: Variação de T com p para $N = 32256$ e $\varrho = 0,82$

Tabela XVI: Tempos de simulação no Cray X-MP

Fase do programa	Tempo para $N = 2016$ (seg)	Tempo para $N = 8064$ (seg)
At. da Tab. Viz.	0,95	15,00
Tempo total	1,00	15,32

Estes resultados nos permitem chegar às seguintes conclusões:

1. *Descréscimo aproximadamente linear de T com p*, isto desde que se esteja trabalhando com uma simulação suficientemente grande;

2. *Alta eficiência de utilização de processadores* assegurada desde que o número de partículas por unidade de processamento seja maior que algo em torno de 1000;

3. *Pequena influência de ϱ sobre a distribuição do processamento*, como podemos ver pelo fato de que d variou muito pouco com a mudança de $0,6$ para $0,82$ em ϱ.

5.5 Tempos no Cray X-MP

Forneceremos aqui alguns tempos de simulação que nos foram fornecidos por Kalia [14], para o caso de $\varrho = 0,6$ e demais parâmetros como utilizados em nossos cálculos.

Suas simulações englobaram os casos de $N = 2016$ e $N = 8064$, utilizando um algoritmo de tabela de vizinhos com atualização a cada 15 quadros. Os resultados obtidos aparecem na tabela XVI, sendo que os mesmos se referem a apenas *um* quadro, que contém atualização da tabela de vizinhos.

Como a atualização da tabela de vizinhos e realizada apenas a cada 15 quadros, devemos alterar os valores de tempo total se quisermos que o mesmo represente tempo médio gasto em cada quadro. Assim, o tempo médio gasto com a atualização da tabela de vizinhos por quadro corresponde a 1/15 do tempo de uma atualização. Assim, se

somamos os tempos médios de atualização da tabela de vizinhos com o tempo do restante do algoritmo, encontramos o tempo médio total para um quadro:

$$t_{N=2016} = \frac{0,95}{15} + 0,05$$
$$= 0,11s \quad (84)$$
$$t_{N=8064} = \frac{15,00}{15} + 0,32$$
$$= 1,32s \quad (85)$$

o que, para 1000 quadros corresponde a:

$$T_{N=2016} = 0,031h \quad (86)$$
$$T_{N=8064} = 0,367h \quad (87)$$

Comparando estes valores com os apresentados acima, vemos que para o caso de $N = 2016$ precisamos de $p > 16$ para conseguir o mesmo tempo de simulação do Cray X-MP. Já para o caso de $N = 8064$, um sistema com $p = 3$ será mais rápido que o Cray. Note que isto corresponde a 6 *transputers*.

Podemos realizar uma projeção do tempo de execução do Cray X-MP para $N = 32256$ procedendo da seguinte forma: ajustamos uma curva do tipo:

$$T_N = aN^2 + bN$$

(as razões para isto podem ser encontradas referindo-nos à forma de crescimento do tempo de execução com N para o algoritmo de tabela de vizinhos, como apresentado no cap. 2 e lembrando que os altos valores de N aqui utilizados permitem-nos desprezar o fator constante)

Encontramos os seguintes valores:

$$a = 5,0 \cdot 10^{-9} \quad (88)$$
$$b = 5,3 \cdot 10^{-6} \quad (89)$$

para a avaliação do tempo de execução em horas:

$$T_N = 5,0 \cdot 10^{-9} N^2 + 5,3 \cdot 10^{-6} N$$

Substituindo $N = 32256$ encontramos:

$$T_{N=32256} = 5,373h$$

e vemos que neste caso, duas unidades de processamento já são suficientes para alcançar um tempo bem inferior ao do Cray.

Devemos ressaltar aqui que já foi mostrado no capítulo 2 que, para grandes quantidades de partículas o algoritmo de granulação grosseira apresenta eficiência bem maior que o de tabela de vizinhos. É portanto de se esperar que, desenvolvido um programa de dinâmica molecular com o algoritmo de granulação grosseira para o Cray X-MP, o mesmo fosse capaz de rivalizar com quantidades maiores de unidades de processamento.

5.6 Conclusão

Do estudo realizado, podemos ver que o sistema proposto comporta-se muito bem com relação à capacidade de expansão, eficiência de utilização de recursos computacionais e crescimento do tempo de execução com o número de partículas.

Devemos lembrar que os cálculos foram realizados tendo em conta unicamente valores médios apresentados pelo fabricante do *transputer*, sendo que para uma simulação real teremos uma degradação dos tempos apresentados devido aos problemas de variações estatísticas nas quantidades de dados a processar e de existência de latência de estabelecimento e terminação de um conjunto de processos encadeados, isto é, quando o processo B utiliza dados fornecidos pelo processo A e por sua vez fornece dados para o processo C, então, o processo B necessita esperar o primeiro ciclo de processamento de A antes de possuir dados. Da mesma forma, o processo C aguarda os ciclos iniciais de A e B. Vemos então que isto implica em ciclos mortos e, portanto, desperdício de tempo de computação.

No entanto, os efeitos desses fatores não podem ser corretamente avaliados sem possuir sistemas de simulação ou execução dos processos.

Capítulo 6

Conclusões

6.1 Considerações gerais

Como pudemos perceber pelas avaliações realizadas no capítulo 5, a estrutura proposta apresenta boas características de expansibilidade, eficiência de utilização de recursos computacionais e tempo de execução.

No entanto, a estrutura como apresentada neste trabalho não representa a totalidade da proposta de dinâmica molecular. Restam ainda os seguintes fatores não definidos:

1. Comunicação com um computador hospedeiro, que deve realizar a interface com o usuário
2. Inicialização dos dados sobre as partículas em cada nó antes do início dos cálculos
3. Estrutura física dos nós (processadores e suas interligações com memórias e demais dispositivos externos)

Estes fatores não foram definidos no presente trabalho pelas seguintes razões:

1. O método de comunicação com o hospedeiro é fortemente dependente tanto do hospedeiro disponível como da forma de implementação dos nós de processamento
2. O processo de inicialização é realizado apenas uma vez para cada simulação, o que significa que o mesmo não representará grande esforço computacional frente ao restante

da simulação. Desta forma, esta parte do processamento não requer cuidados especiais. Além disto, esta inicialização também é dependente dos recursos disponíveis no sistema em que a máquina for implementada

3. A estrutura dos nós é fortemente dependente da forma de implementação escolhida. Pode-se realizar a implementação tanto pelo projeto específico dos nós para o nosso problema como através da aquisição de nós prontos e da interligação dos mesmos

Sugerimos então que essas fases restantes sejam cuidadas unicamente quando for definida a forma de implementação da proposta.

6.2 Futuros trabalhos

Para a continuação deste trabalho, propomos as seguintes linhas de pesquisa:

1. Implementação da proposta em um sistema com quatro *transputers*, interligado a um computador pessoal, e teste final das suas características. Utilização do sistema assim formado para cálculos reais de dinâmica molecular

2. Devido aos grandes tempos de execução envolvidos, existe alto risco de ocorrência de falhas (como por exemplo queda da força) durante uma simulação. Pode-se evitar as perdas de muitos dos cálculos realizados através do seguinte artifício: a cada certo intervalo de tempo, os processadores param os cálculos que estão executando e enviam ao hospedeiro todas as informações sobre as partículas naquele instante, para armazenamento em unidades de disco, de forma que, ao ocorrer uma falha, o cálculo só precisa ser retomado desde o último instante gravado

3. Realização da análise do desempenho para algumas extensões simples do problema considerado como, por exemplo, o caso tridimensional

4. Desenvolvimento da programação para permitir a realização de cálculos envolvendo diversos tipos de partículas

Também sugerimos um trabalho não diretamente relacionado com o trabalho aqui realizado, mas indicada pelo mesmo e que consiste no desenvolvimento de ferramental automático para o auxílio no desenvolvimento de máquinas semelhantes para outros problemas. Poderiam ser incluídas ferramentas como:

1. Simulador de redes de processadores, para permitir um estudo acurado de uma dada proposta de implementação

2. Sistema para estudar a alocação de processos em processadores de uma forma eficiente, isto é, um programa que, dado um conjunto de processos, apresenta ou analisa alternativas para a implementação eficiente dos mesmos em uma dada rede de processadores. Isto pode ser realizado através da análise dos tempos de execução dos processos e dos comunicações entre os mesmos.

3. Desenvolvimento de uma linguagem de especificação de paralelismo, baseada na simbologia adotada neste trabalho, que permita o estudo de utilização dos paralelismos do problema por uma dada proposta de implementação

4. Desenvolvimento de uma linguagem de especificação de problemas que permita o cálculo da complexidade do mesmo sem a necessidade de se chegar ao nível de especificação de algoritmos

Apêndice A

Esquematização do Paralelismo

A.1 Introdução

Tendo em vista uma representação sintética e de fácil visualização de processos paralelos, desenvolveremos uma notação que indica as dependências entre os diversos processos envolvidos de uma forma gráfica. Foi nosso intuito o de que esta notação possibilitasse a mostra de muitos dos paralelismos próprios do algoritmo envolvido, sendo que as restrições impostas pela implementação específica do mesmo em um sistema computacional seriam indicadas independentemente. Enfatizemos também que a redução do paralelismo apresentado, por restrições do sistema real a ser utilizado, podem em grande parte ser deduzidas à partir do diagrama apresentado utilizando-se técnicas de compartilhamento de tempo e de realização em série.

A.2 Elementos básicos

Passaremos agora à descrição da forma de representação que utilizaremos para os elementos básicos de um processamento paralelo, quais sejam: os processos, as comunicações entre os mesmos e os elementos de armazenamento (memória).

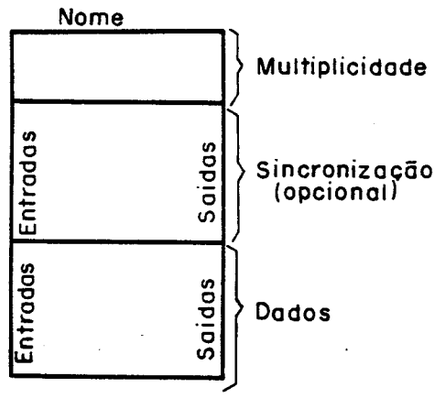


Figura 42: Representação de processos

A.2.1 Representação de processos

Os processos serão indicados por retângulos, divididos por linhas horizontais em 3 partes (sendo uma opcional). A parte inferior indica os dados de entrada e dados de saída do processo. A parte central, que pode ser omitida quando julgado conveniente, representa sinais para sincronização deste processo com os demais e com os elementos de memória. Na parte superior do retângulo encontramos indicações referentes à multiplicidade deste processo, isto é, quantos processos idênticos, porém operando em dados distintos, existem em paralelo. Veja a figura 42 onde é apresentada graficamente esta notação.

Tanto sinais de sicronização como dados que se apresentam ligados ao retângulo do processo pelo lado esquerdo são considerados *entradas* do mesmo, enquanto que os que aparecem à direita são considerados *saídas*.

Os nomes dos sinais de sincronização e dos dados são colocados na parte interna do retângulo. O nome do processo representado é colocado externamente ao retângulo, em sua face superior.

Como exemplo inicial, na fig. 43 encontramos a representação de um processo que executa a soma $a = b + c$, sendo o mesmo um processo único (multiplicidade 1). Note que o número que indica a multiplicidade é indicado entre chaves. A representação dos

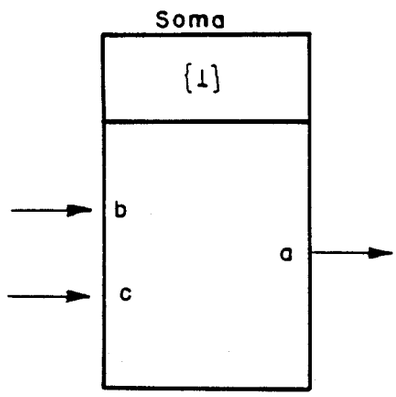


Figura 43: Processo de soma $a = b + c$

sinais de sincronização foi considerada desnecessária neste caso, de forma que o retângulo está dividido apenas em duas partes.

Suponha que dispomos de duas matrizes 4×5 que devem ser multiplicadas elemento a elemento. Alocando-se um processo para cada multiplicação, podemos indicar todos por um único retângulo (pois todos são idênticos), sendo que a multiplicidade seria 20. No entanto, é mais instrutivo indicarmos que dispomos de um arranjo 4×5 de processos. Isto é indicado em nossa notação através de uma seqüência de elementos entre chaves, neste caso $\{4\}\{5\}$, que indica que dispomos de quatro conjuntos, cada qual com cinco processos identicos ao indicado. Esta notação pode ser extendida para um número maior de elementos.

Muitas vezes é conveniente indicar-se qual a alocação dos processos pelos processadores, isto é, para cada processo, qual processador em que o mesmo será executado. Isto é feito através de um número ou símbolo entre colchetes na parte do retângulo dedicada à multiplicidade. Na fig. 44 vemos a representação de um conjunto de processos destinados a realizar a multiplicação elemento a elemento de duas matrizes 2×3, sendo que todos os processos estão alocados ao processador número 7.

Mais adiante veremos extensões da notação apresentada até aqui.

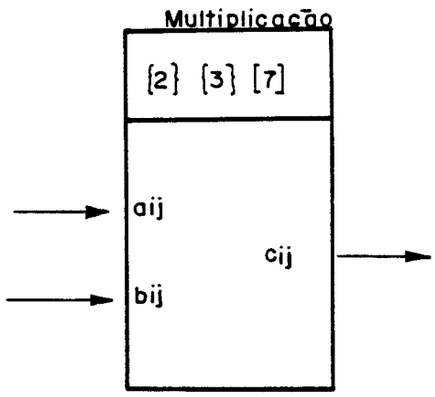


Figura 44: Multiplicação elemento a elemento de duas matrizes

A.2.2 Representação de comunicações entre processos

As comunicações de dados entre os processos serão representadas por linhas ligando os mesmos. No caso das comunicações serem múltiplas, a multiplicidade é anotada, seguindo a mesma convenção que para os processos, acima da linha. Quando não existe anotação de multiplicidade, o valor assumido é *1*.

Na fig. 45 temos a representação das comunicações que fornecem os dados para um processo de multiplicação de um vetor de 20 elementos por um escalar. Note que tanto o processo quanto as comunicações apresentam multiplicidade 20. O local de onde provém o vetor não foi especificado, bem como o local de destino.

A.2.3 Representação de elementos de armazenamento

Os elementos de armazenamento são representados por uma notação semelhante à dos processos. As diferenças são as seguintes:

1. As bordas do retângulo são desenhadas com linhas duplas, para haver uma diferenciação visual imediata

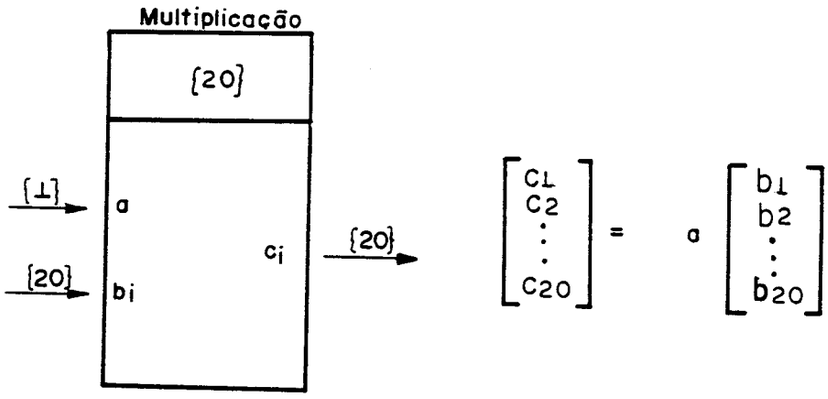


Figura 45: Diagrama para o produto de um vetor de 20 componentes por um escalar

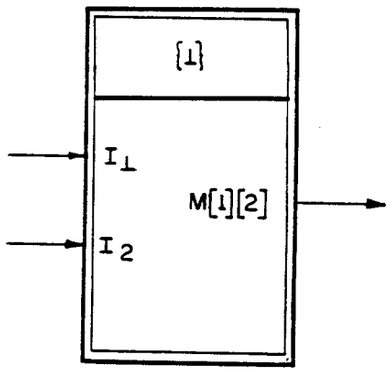


Figura 46: Representação do armazenamento de uma matriz duplamente indexada

2. Nas memórias existe, além da entrada de dados, um outro tipo de entrada que corresponde ao endereço de armazenamento (ou índice de um vetor). Para diferenciar os dois tipos de entrada, as que formam endereços para acessos são marcadas por um I (abreviação de *índice*) seguido de um número. Este número aparecerá entre colchetes na entrada ou saída de dados à qual se refere o índice. Caso uma entrada ou saída possua dois ou mais índices, estes aparecerão em pares de colchetes justapostos. Na fig. 46 mostramos o diagrama de um elemento que armazena uma matriz duplamente indexada. Existem casos em que algum dado pode ser indexado alternativamente por

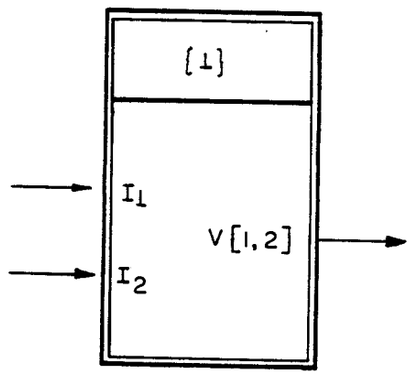


Figura 47: Representação de índices alternativos

uma ou outra entrada. Para indicar indexações alternativas, colocamos os números dos dois índices dentro de um mesmo par de colchetes, separados por vírgulas. Na fig. 47 temos um exemplo de um vetor indexado alternativamente por um de dois índices. Vale como índice aquele que estiver ativo no momento em que o dado for acessado.

3. Para indicar que dois ou mais acessos ao mesmo elemento de armazenamento são independentes (isto é, sem concorrência), englobamos cada um deles em retângulos distintos, sendo que os que apresentam concorrência entre sí aparecem dentro do mesmo retângulo. Na fig. 48 vemos a representação de um elemento que armazena as variáveis A, B e C, sendo que, na leitura os acessos a A e B apresentam concorrência entre sí, enquanto que os acessos a C são independentes aos de A e B. Da mesma forma na escrita temos concorrência nos acessos de B e C, enquanto que o acesso a A é independente dos mesmos. Note que foram utilizados números nos cantos dos retângulos. Este números servem para relacionar retângulos da leitura com outros da escrita, indicando que são os mesmos, isto é, se um retângulo de leitura e um de escrita apresentam o mesmo número isto indica que os correspondentes acesso de escrita e de leitura são concorrentes. Desta forma, na figura indicada temos

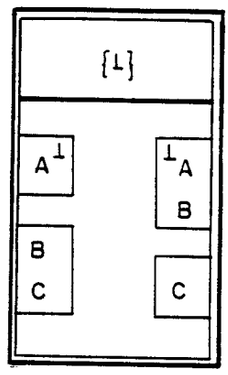


Figura 48: Representação de acessos independentes ao mesmo elemento de armazenamento

concorrência nos acessos de escrita de A e de leitura de A e B.

A.3 Algumas convenções auxiliares

Descreveremos a seguir algumas convenções importantes, ainda não descritas em nenhum dos ítens acima:

1. Em alguns casos ocorre de termos uma certa quantidade de dados de entrada ou de saída que executam exatamente a mesma função em um elemento de armazenamento ou em um processo. Para representar esse tipo de dados, introduzimos o conceito de entradas e saídas múltiplas, que são representadas através de retângulos como os que indicam dados de acesso independente, discutidos acima, com a inclusão de um nível determinador da múltiplicidade, como na fig. 49, onde representamos um processo que realiza a somatória dos 30 valores apresentados em sua entrada. Esta notação se extende não apenas a dados mas também a sinais de sincronização. Para simplificar a notação, diversas saídas ou diversas entradas de um *processo* que possuam a mesma multiplicidade podem ser ligadas dentro de um mesmo retângulo, sem que isso represente concorrência na comunicação desses dados.

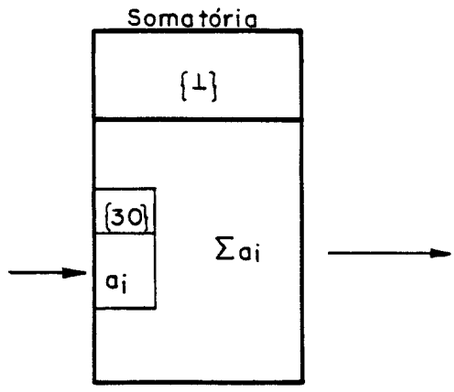


Figura 49: Entradas e saídas múltiplas

2. Mostraremos agora algumas regras de ampliação e redução de multiplicidade, isto é, qual as interligações a serem realizadas quando um elemento de dada multiplicidade é ligado a um de multiplicidade maior ou a um de multiplicidade menor. Designando por m a multiplicidade do elemento que fornece os dados e por n a do que os capta, estudaremos os seguintes casos:

   $m < n$, n múltiplo de m : neste caso repetimos cada um dos dados n/m vezes, sendo que cada dado é apresentado todas as vezes antes que o próximo dado seja apresentado. Veja fig. 50 para um exemplo com $n = 6$ e $m = 2$.

   $m > n$, m múltiplo de n : aqui tomamos apenas alguns dos dados fornecidos, a uma razão de m/n, isto é, tomamos um dado, ignoramos $m/n - 1$, tomamos outro dado, ignoramos outros $m/n - 1$ dados e assim sucessivamente. Veja fig. 51

   $m < n$, m e n primos entre si: seguimos o mesmo procedimento que para o caso em que n é múliplo de m, só que utilizando para o número de repetições o resultado da divisão inteira de n por m. Os restantes componentes de captação são ligados ao primeiro dos dados. Veja fig. 52

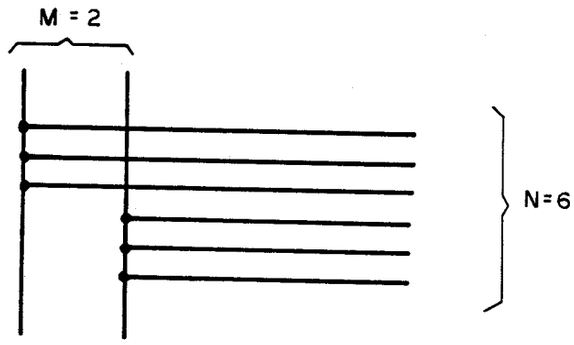


Figura 50: Exemplo de expansão de multiplicidade com n múltiplo de m

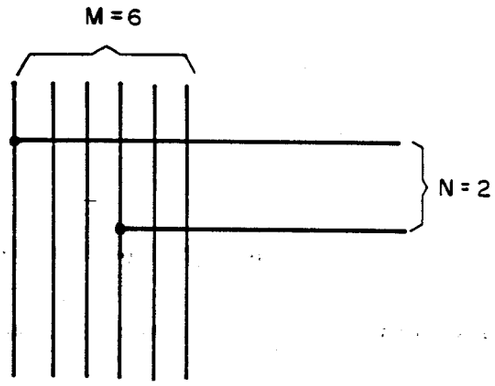


Figura 51: Exemplo de redução de multiplicidade com m múltiplo de n

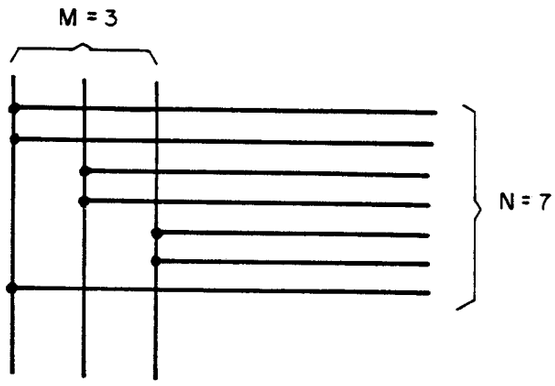


Figura 52: Exemplo de expansão de multiplicidade com m e n primos entre sí

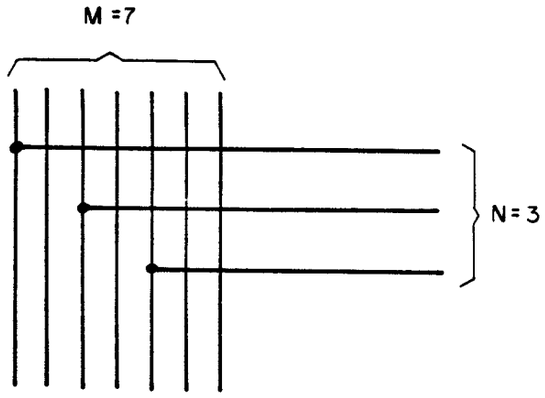


Figura 53: Exemplo de redução de multiplicidade com m e n primos entre sí

$m > n$, m e n primos entre sí: o procedimento é igual ao do caso em que m é múliplo de n, utilizando-se para a separação entre os dados considerados o resultado da divisão inteira entre m e n. Veja fig. 53.

m do tipo $\{m_1\}\{m_2\}$ e n do tipo $\{m_1\}$: Neste caso, associamos a cada um dos elementos captadores a linha inicial de cada um dos m_1 blocos indicados pela multiplicidade dos elementos fornecedores. Para o caso em que m_2 é constante esta regra não passa de um caso de $m > n$ com m múltiplo de n, no entanto a regra apresentada neste ítem continua válida mesmo para os casos de multipli-

cidades variantes a serem apresentadas a seguir.

m do tipo $\{m_1\}$ e n do tipo $\{m_1\}\{m_2\}$: Quando isto ocorre, realizamos m_1 blocos, cada um consistindo de m_2 elementos iguais. Vale para esta regra a mesma observação feita para a anterior.

Nota: as regras dos dois ítens anteriores aplicam-se também quando $\{m_1\}$ é uma multiplicidade do tipo $\{m_3\}\{m_4\}...\{m_k\}$.

3. Consideremos agora a utilização de índices de multiplicidade para multiplicidade variante. Como vimos, a multiplicidade pode ser expressa por um conjunto de números, cada um deles entre chaves, representando grupos sucessivos de repetições. A notação apresentada até o momento permite apenas a representação de conjuntos formados por repetições constantes, por exemplo, uma multiplicidade de $\{2\}\{3\}$ representa duas repetições de um conjunto de três elementos. Para permitir a representação de multiplicidades variantes, introduziremos a representação seguinte:

   - A letra α representa o índice associado ao primeiro conjunto de chaves à esquerda
   - A letra β representa o índice associado ao segundo conjunto de chaves da esquerda para a direita
   - Assim sucessivamente, na ordem do alfabeto grego, com um índice associado a cada conjunto de chaves da multiplicidade.

   Assumimos que os índices iniciam de 1, crescendo até o número máximo. Com esta convenção, um conjunto de linhas de comunicação com multiplicidade $\{2\}\{3\}$ corresponde a seis linhas com índices associados da seguinte forma:

   primeira linha : $\alpha = 1, \beta = 1$

   segunda linha : $\alpha = 1, \beta = 2$

terceira linha : $\alpha = 1, \beta = 3$

quarta linha : $\alpha = 2, \beta = 1$

quinta linha : $\alpha = 2, \beta = 2$

sexta linha : $\alpha = 2, \beta = 3$

Podemos então utilizar estes índices para indicar multiplicidades. Por exemplo, uma multiplicidade $\{3\}\{\alpha\}$ corresponde a elementos representados como a seguir:

primeiro elemento : $\alpha = 1, \beta = 1$

segundo elemento : $\alpha = 2, \beta = 1$

terceiro elemento : $\alpha = 2, \beta = 2$

quarto elemento : $\alpha = 3, \beta = 1$

quinto elemento : $\alpha = 3, \beta = 2$

sexto elemento : $\alpha = 3, \beta = 3$

Isto pois o α dentro do segundo par de chaves indica que para cada um dos três conjuntos de elementos indicados pelo primeiro par de chaves temos tantos elementos quanto for o índice α. Como último exemplo, descrevemos abaixo o significado da multiplicidade representada por $\{3\}\{3-\alpha\}$:

primeiro elemento : $\alpha = 1, \beta = 1$

segundo elemento : $\alpha = 1, \beta = 2$

terceiro elemento : $\alpha = 2, \beta = 1$

Como vemos, são apenas três elementos, pois para $\alpha = 3$ não teremos nenhum elemento.

Outra utilização dos índices é com a introdução de critérios de seleção no indicador de multiplicidade. Assim, uma multiplicidade indicada por $\{3\}\{\beta > \alpha\}$ corresponderia ao seguinte:

primeiro elemento : $\alpha = 1, \beta = 2$

segundo elemento : $\alpha = 1, \beta = 3$

terceiro elemento : $\alpha = 2, \beta = 3$

Que são as únicas condições onde $\beta > \alpha$. Da mesma forma podem ser utilizados quaisquer outros critérios de seleção.

4. Às vezes, um sinal de sincronização deve ser gerado por uma combinação lógica de diversos sinais externos. Nos casos em que isto ocorre, utilizaremos uma notação igual à empregada na notação de dependência do padrão IEEE Std 91/ANSI Y32.14 (para lógica digital). Desta forma, por exemplo, um **OU** lógico será indicado por um retângulo marcado com ≥ 1.

5. Quando queremos que um sinal de sincronização seja gerado de acordo com o valor de um dado de entrada, ligamos este dado a uma entrada de sincronização marcada por uma condição que o dado deve satisfazer, indicada entre parêntesis, seguida do indicador de tipo de sinal de sincronização. Por exemplo, uma entrada de sincronização marcada com $(= \alpha)$**S1** significa que o dado de entrada marcado com (1) será selecionado apenas se o valor de dado de entrada ligado ao sinal de sincronização apresentar um valor igual ao do índice de multiplicidade do processo correspondente.

6. Quando existem comunicações entre diversos elementos (de processamento ou armazenamento) representados por um único elemento múltiplo, indicamos esta comunicação da seguinte forma:

   - Colocamos, incrustado no elemento principal, um representante de um outro elemento através de um retângulo (com ou sem indicador de multiplicidade, conforme o caso)

- Marcamos este retângulo com um traço curto horizontal, que representará a comunicação entre os elementos
- Indicamos próximo ao traço horizontal uma condição de seleção do elemento com o qual se estabelecerá a comunicação
- Para representar o índice do outro elemento utilizamos um apóstrofe após o índice de multiplicidade. Assim, α' representa um *outro* elemento para o qual α satisfaz a condição apresentada para α'.

Esta simbolização pode ser vista no exemplo da fig. 54, onde representamos um processo de multiplicidade 10, em que cada elemento se comunica com o seu anterior e o posterior.

Para esclarecer melhor a utilização de índices de multiplicidade associado com reduções e ampliações da mesma, apresentamos na fig.55 um exemplo que associa os dois fatos. Neste caso dois processos com multiplicidade 3 são ligados através de um conjunto de 6 elementos de comunicação distribuídos numa multiplicidade $\{3\}\{4-\alpha\}$. São apresentados o diagrama resumido e a expansão do mesmo de acordo com as convenções apresentadas. Note como as mesmas propiciam uma ligação correta de elemento a elemento.

Além de na própria indicação de multiplicidade, os símbolos α e β podem ser utilizados em qualquer outra parte dentro dos processos, sempre indicando os números indicadores daquele processo dentre todos os processos múltiplos semelhantes.

A.4 Exemplos de representações

Vejamos agora como esses elementos se interrelacionam para formar a representação de um processo paralelo. Conforme forem necessárias serão apresentadas algumas extensões à notação apresentada até aqui.

Um exemplo simples é o de um processo para realizar a operação $a = (b+c)(d+$

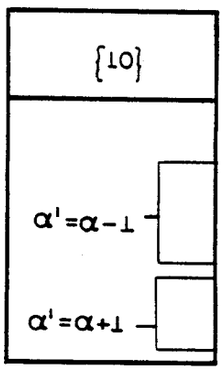


Figura 54: Comunicação entre diferentes elementos do mesmo símbolo

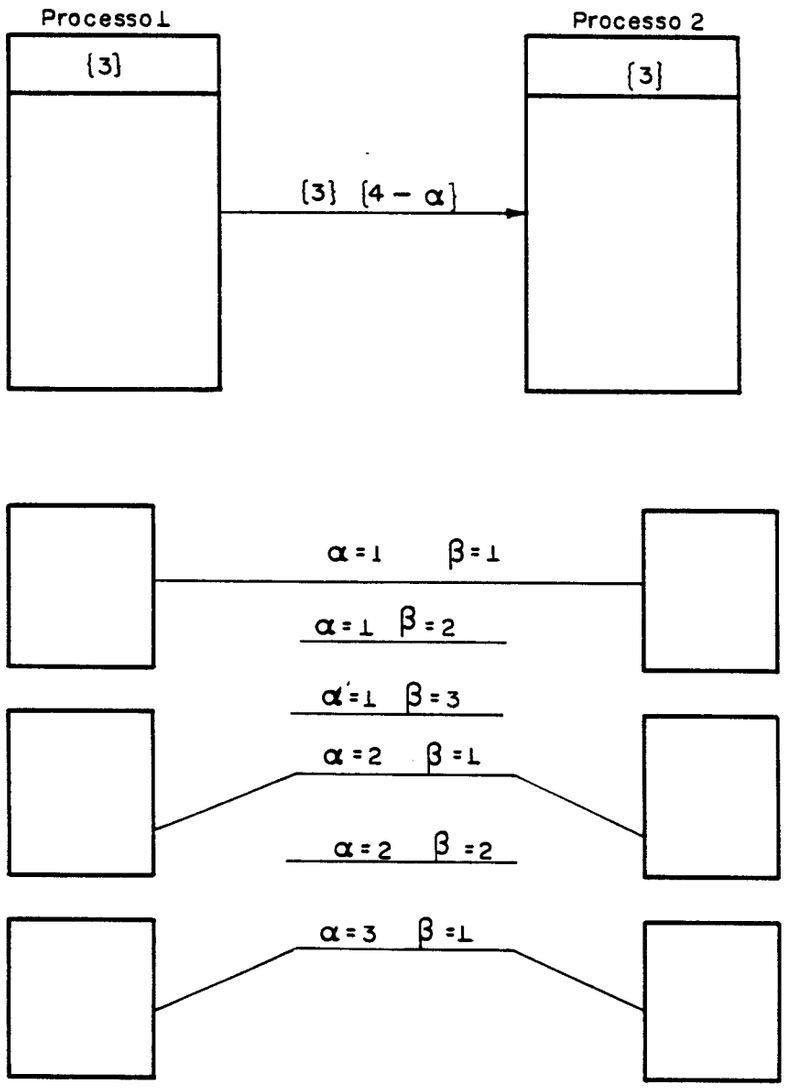


Figura 55: Exemplo de multiplicidade variante com ampliação e redução de multiplicidade

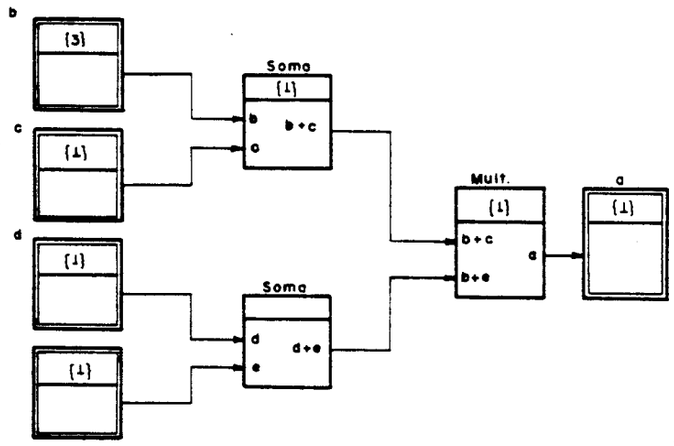


Figura 56: Processo para realização da operação $a = (b + c)(d + e)$

e), constituído de dois processo de soma e um de multiplicação, além de 5 elementos de memória, um para cada variável. Este processo é apresentado na fig. 56, onde partimos do princípio de que os processos possuem um sistema de sincronização da transmissão dos dados em que quando um processo necessita de dados de um outro ele fica parado aguardando a chegada dos mesmos, e prossegue seu processamento quando dispõe de todos os dados necessários. Este tipo de sincronização será assumido sempre que não houver sincronizações explicitamente indicadas.

Na fig. 57 apresentamos um processo mais complexo, onde se apresenta a utilização de multiplicidade. Nessa figura temos um processo para realizar o produto escalar de dois vetores A e B de ordem 5, sendo que o resultado é armazenado na variável e. Os pontos importantes a considerar nesta figura são:

- Assumimos que ambos os vetores estão armazenados de forma a que tanto o acesso aos dois vetores como o acesso a cada elemento dos mesmos é independente
- Assumimos que o acesso à variável de saída e é independente do acesso aos vetores
- Assumimos que todos os processos de multiplicação de elementos são realizados em paralelo

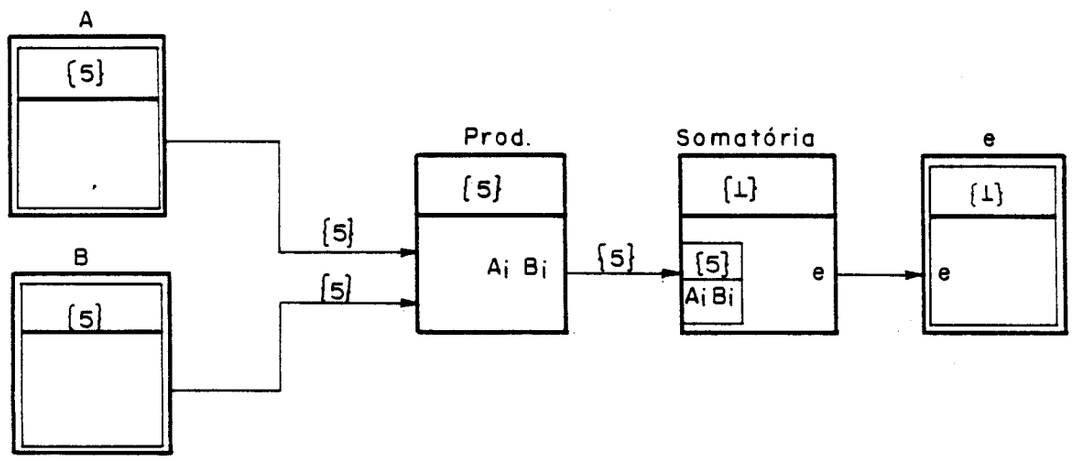


Figura 57: Processo para produto escalar de dois vetores

- Note como os resultados das diversas multiplicações são levados a um único processo de soma

Na fig. 58 temos também a realização do mesmo produto escalar da figura anterior, só que partindo do princípio de que existe um único elemento físico de armazenamento, de forma que os componentes dos vetores devem ser acessados um por vez, assim como os dois vetores diferentes. Nessa figura consideramos que:

- Existe um novo processo encarregado de ler os componentes corretos dos vetores na memória e enviar os dados aos processos que realizam o produto dos mesmos
- Os componentes dos vetores A e B são diferenciados a partir de índices. Vemos portanto no elemento de armazenamento um índice, que dirige o acesso ao vetor A e ao vetor B
- O produto escalar e é armazenado no mesmo elemento físico de memória que os vetores (note a indicação no campo de multiplicidade)
- O processo que realiza o produto é suficientemente lento em relação ao acesso na

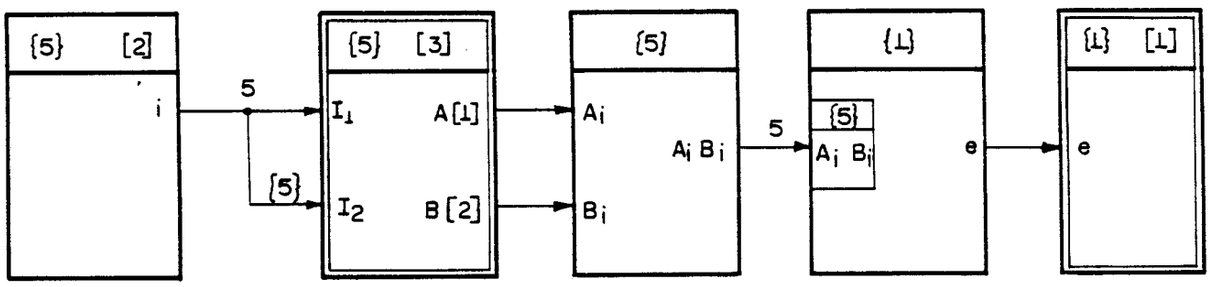


Figura 58: Produto escalar com memória única

memória para justificar a existência de vários do mesmo enquanto existe apenas um elemento físico de memória

Na fig. 59 temos um processo que realiza a operação $A \leftarrow A+B$, onde A e B são matrizes 3×7. Supomos que todos os elementos de ambas matrizes podem ser acessados independentemente. Note que os elementos da matriz A devem ser lidos e escritos. No entanto não existe confusão quanto a isso pois antes de se escrever um elemento de A o mesmo já deve ter tido seu valor lido, pois é necessário para a produção do valor que será escrito. Não existe portanto neste caso a possibilidade de desvirtuamento dos dados, o que faz com que não seja necessária a indicação das sincronizações.

Em certos momentos se deseja que dois processos executem estritamente de modo seqüencial, isto é, um não inicia a executar enquanto o outro não tiver terminado, independentemente de dispor ou não dos dados necessários. Isto não pode ser descrito em nossa notação sem a utilização de sinais de sincronização. Para isto introduzimos dois sinais de sincronização: STR (de *start*), entrada de sincronização que indica ao processo que o mesmo pode começar, e END, saída de sincronização que indica que o processo terminou.

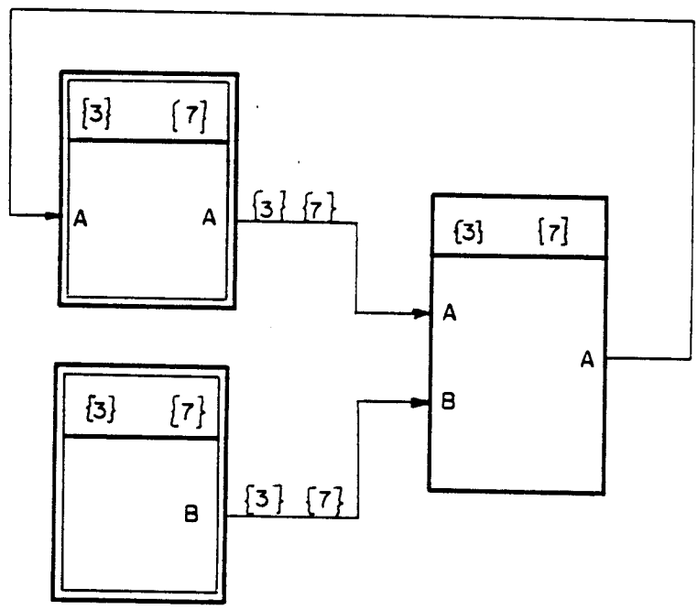


Figura 59: Processo para $A \leftarrow A + B$

Utilizando estes dois sinais podemos desenhar o diagrama da fig. 60 que representa um processo que incrementa uma variável a de 1 e em seguida utiliza seu novo valor para gerar o de uma variável $b \leftarrow 2a$. Note que da forma como os blocos estão interligados, sem os sinais de sincronização poderia ocorrer do processo que multiplica a por 2 acessar a antes da mesma ser incrementada.

O diagrama da fig. 61 representa um processo para o cálculo do fatorial de um número previamente armazenado em um elemento de memória. Este diagrama utiliza um novo tipo de sinal de sincronização, que se ativa de acordo com o valor de certos dados. Neste caso temos um processo que compara o valor de sua entrada com *um*. Se a entrada é diferente de *um*, o dado de entrada é enviado para um novo ciclo, enquanto que se for igual a *um* é disparado um processo que armazena o valor calculado no local de destino do fatorial.

Na fig. 62 temos um outro exemplo de utilização de sinais de sincronização

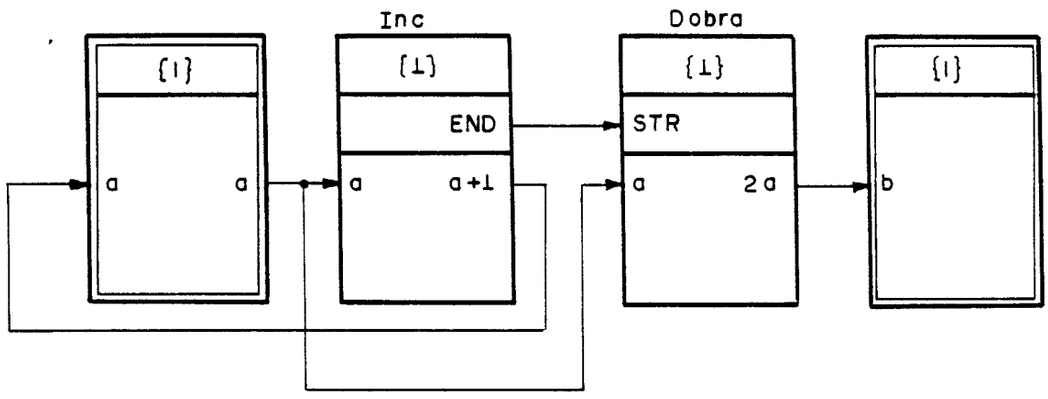


Figura 60: Diagrama para $a \leftarrow a + 1$ seguido de $b \leftarrow 2a$

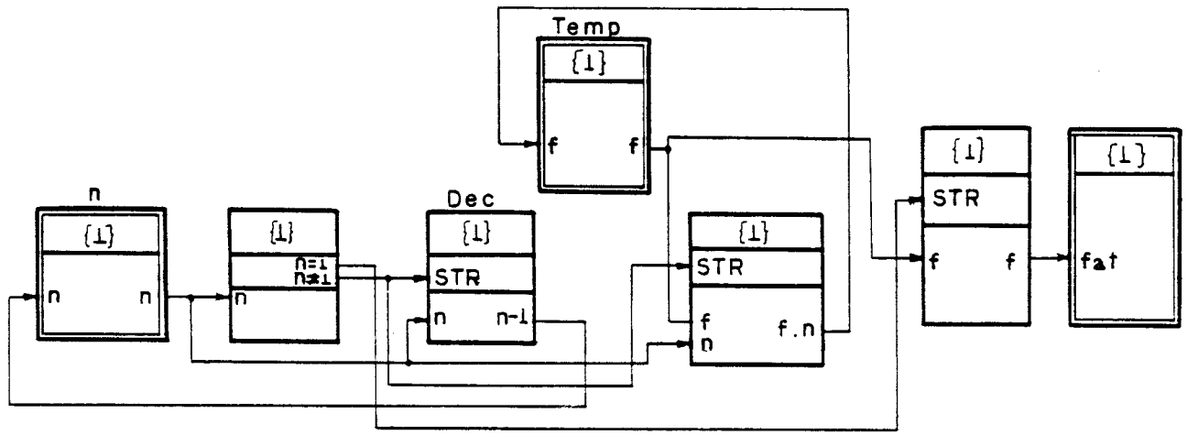


Figura 61: Exemplo de sincronização com valor de dados: cálculo do fatorial

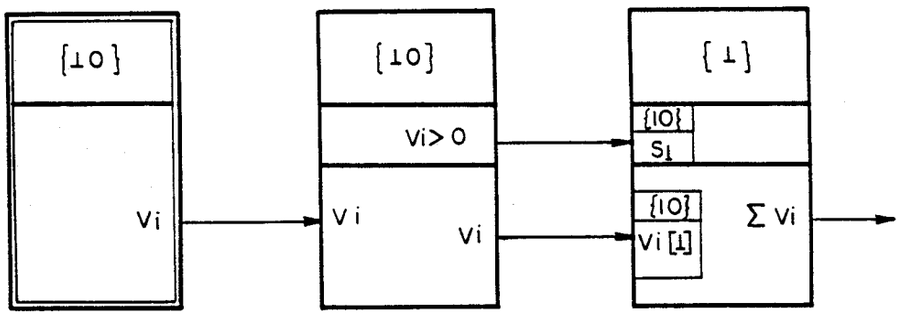


Figura 62: Sincronização por valor de dados: Soma de elementos positivos de um vetor

vinculados a valores dos dados, desta vez utilizado em conjunto com processos múltiplos. Este processo executa a soma dos valores dos componentes de um vetor armazenados em elementos múltiplos de memória, considerando apenas os componentes positivos. Para cada um dos componente do vetor temos um processo que testa se o mesmo é maior que zero, sendo que neste caso o mesmo é enviado para um processo que realiza a soma. O processo de soma se utiliza dos sinais gerados pelo processo de comparação para saber quais de todos os dados possíveis serão somados. Para indicar isso, a entrada do sinal de sincronização é marcada com S (de *Seleção*) seguido de um número. Este número aparece entre parentesis ao lado do dado que será selecionado pela entrada S correspondente.

Um outro tipo de sinal de sincronização determinado por dados é apresentado no exemplo da fig. 63, que exemplifica um processo de decisão de caminhos de processamento. No diagrama da figura, um número (presente na variável x) é multiplicado por ele mesmo tantas vezes quanto indicado por uma variável n.

Outros sinais de sincronização freqüentemente úteis são os que indicam disponibilidade e liberação de dados. Dizemos que um dado está *disponível* quando ele está pronto

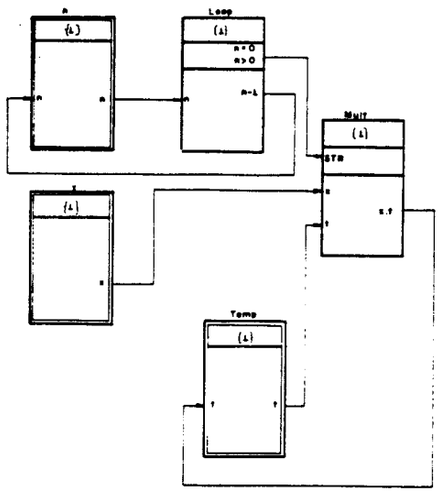


Figura 63: Exemplo de utilização de decisão sobre dados

para ser comunicado entre os processos. Assim, um sinal de sincronização de entrada indicador de diponibilidade, marcado como D seguido de um número, que será associado a um dado de entrada (colocado entre parêntesis após o dado), representará um sinal que será ativo quando o dado indicado estiver disponível para a leitura. Da mesma forma, uma indicação igual como sincronização de saída representará um sinal que será ativo quando o dado de saída indicado estiver pronto para a comunicação. Utilizamos a mesma notação, só que indicando com um L, para indicar a *liberação* de um dado. Só que neste caso uma *saída* de sincronização corresponde a um dado de *entrada*, indicando um sinal que será ativo quando o presente processo já o leu e, portanto, o seu valor pode ser alterado sem prejuízo de seu funcionamento. Da mesma forma, um sinal de sincronização de *entrada* corresponde a umo dado de *saída* que já foi liberado pelo processo que o leu, e portanto o seu valor já pode ser alterado. Essas sincronizações são utilizadas no diagrama da fig.64, onde a comunicação dos dados entre o processo que envia o valor de *b* e o que o lê foi apresentada explicitamente com a utilização dos sinais de sincronização discutidos acima.

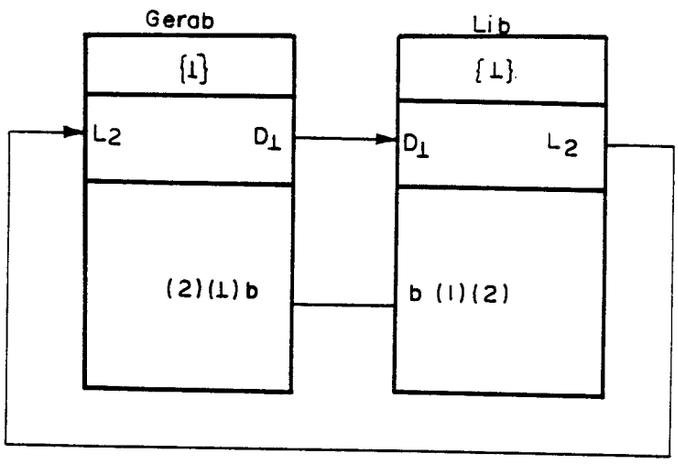


Figura 64: Exemplo da utilização dos sinais D e L

Na fig.65 temos a representação de um processo que utiliza entradas e saídas múltiplas, bem como ampliação de multiplicidade. Corresponde à multiplicação de uma matriz 3×3 por um vetor de três elementos. O vetor é representado por um elemento de memória com saída de multiplicidade 3. A matriz é representada por nove elementos de memória independentes. Como a saída do vetor é ligada a um elemento de comunicação de multiplicidade nove, há uma ampliação de multiplicidade interessante de ser estudada com detalhes pois apresenta um novo fator ainda não introduzido. Neste caso, como o elemento de armazenamento do vetor possui três saídas, consideramos que as mesmas devem ser todas ligadas a elementos de comunicação antes que começemos a realizar repetições, isto é, inicialmente ligamos o primeiro elemento do vetor ao primeiro elemento de comunicação, o segundo elemento do vetor ao segundo da comunicação, o terceiro do vetor ao terceiro da comunicação, e só então começamos a repetição, ligando o primeiro elemento do vetor ao quarto elemento de comunicação, o segundo do vetor ao quinto de comunicação e assim sucessivamente. No caso dos elementos da matriz as ligações são na relação de um para um, bem como na entrada dos elementos de processamento. Veja que temos três elementos de processamento, cada um com entradas de multiplicidade três. Na fig. 66 temos o diagrama do mesmo processo com a multiplicidade explicitamente mostrada.

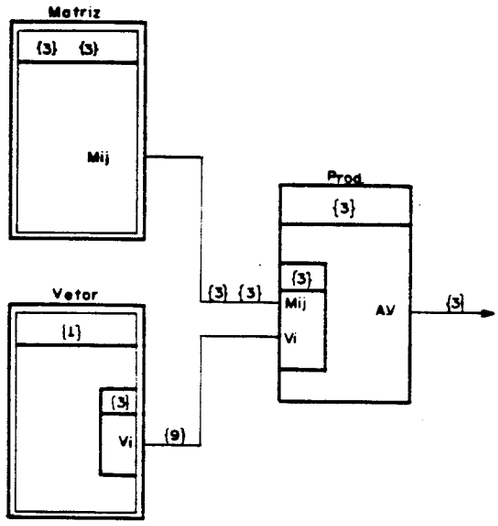


Figura 65: Produto de matriz por vetor

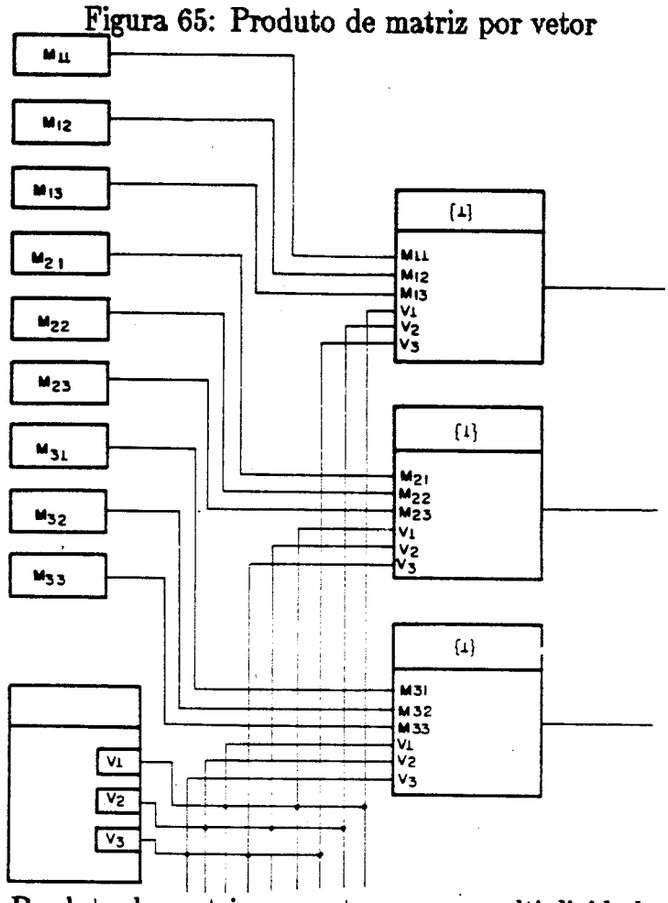


Figura 66: Produto de matriz por vetor com a multiplicidade explicitada

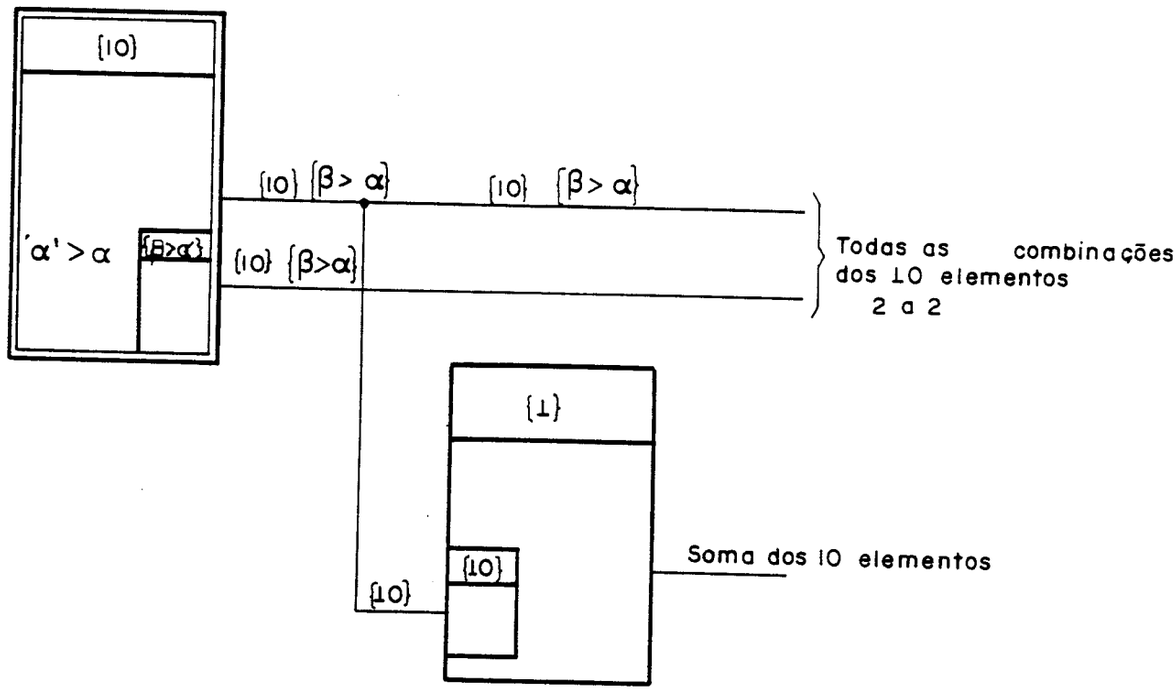


Figura 67: Combinação de 10 elementos dois a dois e soma dos 10 elementos

Como exemplo final, temos na fig. 67 o diagrama de um processo que fornece simultaneamente todas as combinações dois a dois de 10 elementos dados e a soma desses elementos. Note a utilização de entradas e saídas múltiplas, a utilização de multiplicidade com critério de seleção, e a redução de multiplicidade na ligação entre um conjunto de elementos de comunicação (o que vai fornecer as combinações) e outro (o que vai fornecer a soma dos elementos).

Apêndice B

Descrição em OCCAM dos processos avaliados

Apresentamos nas páginas seguintes os códigos OCCAM para os processos que foram utilizados na apresentação e avaliação da proposta (cap. 4 e cap. 5).

Bibliografia

[1] Anders Wallqvist, Bruce Berne, and Chani Pangali. Exploiting physical parallelism using supercomputers: two examples from chemical physics. *Computer*, 20(5):9–21, may 1987.

[2] Charles L. Seitz. The cosmic cube. *Communications of the ACM*, 28(1):22–33, january 1985.

[3] J. R. Gurd, C. C. Kirkham, and I. Watson. The Manchester prototype dataflow computer. *Communication of the ACM*, 28(1):34–52, january 1985.

[4] Geoffrey C. Fox and Steve W. Otto. Algorithms for concurrent processors. *Physics Today*, 50–59, may 1984.

[5] Hassan M. Ahmed. Highly concurrent computing structures for matrix arithmetic and signal processing. *Computer*, 65–82, january 1982.

[6] H. T. Kung. Why systolic architectures? *Computer*, 37–46, january 1982.

[7] Leonard Uhr. *Algorithm-Structured Computer Arrays and Networks.* Academic Press, Inc, Orlando, Florida, 1984.

[8] A. F. Bakker, C. Bruin, F. van Dieren, and H. J. Hilhorst. Molecular dynamics of 16000 Lennard-Jones particles. *Phys. Letters*, 93A(2), 1982.

[9] Farid. F. Abraham. Computation statistical mechanics — metodology, properties and supercomputing. *Adv. in Physics*, 35(1):1–111, 1986.

[10] Rajiv Kalia. Special purpose molecular dynamics machine. 1986. Comunicação privada.

[11] L. Verlet. Computer experiments on classical fluids: I. thermodynamical properties of Lennard-Jone molecules. *Physical Review*, 159(1):98–103, july 1967.

[12] *Transputer Reference Manual.* INMOS Ltd., Bristol, september 1985.

[13] C. Askew. *T800 evaluation results.* Internal Note 5, ESPRIT WP7, 1987.

[14] Rajiv Kalia. Simulações de dinâmica molecular no Cray X-MP. Comunicação privada.

[15] James C. Browne. Parallel architectures for computer systems. *Physics Today*, 28–35, may 1984.

[16] *IMS T800 Transputer.* INMOS Ltd., Bristol.

[17] D. Pountain. *A Tutorial Introduction to OCCAM Programming.* INMOS Ltd., Bristol, march 1986.

[18] *The Transputer Instruction Set — A Compiler Writer's Guide.* INMOS Ltd., Bristol, february 1987.

[19] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, Aug. 1978.

```
-- Descricao OCCAM para processos de dinamica molecular (granulacao grosseira)

-- NOTA: constantes marcadas com ??? sao as que devem ser adaptadas ao problema
--       especifico do usuario ou a estrutura dos elementos fisicos utilizados
--       para o processamento.

-- Constantes uteis para diversos processos:
VAL Num.Graos IS ??? :              -- Numero total de graos
VAL ncoord IS 2 :                   -- caso bidimensional
VAL ngx IS ??? :                    -- numero de graos na direcao x
VAL ngy IS ??? :                    -- numero de graos na direcao y
VAL ng IS [ngx, ngy] :              -- vetor com numeros de graos
VAL Num.Div.Proc IS ??? :           -- numero de divisoes de processamento
VAL Num.max.Part IS 16*1024 :       -- Numero maximo de particulas por D.P.
VAL Num.Graos.Div IS Num.Graos/Num.Div.Proc :
VAL NQ IS ??? :                     -- Numero de quadros
VAL Nulo IS -1 :                    -- particula nula
VAL Grao.Nulo IS -1 :               -- grao nulo
VAL Grao.term IS -2 :               -- grao indicador de termino

-- Definicao dos canais
PROTOCOL grao IS INT :
PROTOCOL particula IS INT :
PROTOCOL coordenadas IS [ncoord]INT :
PROTOCOL coordenadas.auxiliar IS INT; [ncoord]INT :
PROTOCOL saida.prtc IS 1; [ncoord]INT; [ncoord]INT OR
          2; [ncoord]INT; [ncoord]REAL32; [ncoord]REAL32 OR
          3; [ncoord]INT OR
          4 OR
          5 :
PROTOCOL fila.saida IS 1; [ncoord]INT; [ncoord]INT OR
          2; [ncoord]INT; [ncoord]INT; [ncoord]INT OR
          3; [ncoord]INT OR
          4 OR
          5 :
PROTOCOL estado IS 1; [ncoord]INT OR
          2; [ncoord]REAL32; [ncoord]REAL32 :
PROTOCOL entrada.prtc IS 1; [ncoord]INT; [ncoord]INT OR
          2; [ncoord]INT; [ncoord]REAL32; [ncoord]REAL32 :
CHAN (grao) nova.grao, grao.viz, nova.aux.grao, grao.aux, grao.cent :
CHAN (INT; [ncoord]INT) nova.prtc, coord.le, nova.paux :
CHAN (BOOL) fim.quad, fim.grao, fim.aux, fim.tst, pede.prtc, fim.junta :
CHAN (BOOL) pot.fim.quad, cin.fim.quad, fim.ren, pot.fim.prog, cin.fim.prog :
CHAN (BOOL) outro.grao, terminou, fim.varr.quad, fim.varr.grao, outro.i :
CHAN (BOOL) fim.varr, fim.ac, fim.coord, fim.vel, pede.scin :
CHAN (coordenadas) caux.le, ant.le.coord, ant.esc.coord, nova.coord, coord.ext :
CHAN (coordenadas) coord.cnt, coord.perif, coords, xnovo, xanter :
CHAN (particula) paux.dis, ant.le.prtc, ant.esc.prtc, icalc, prtc.ext :
CHAN (particula) prtc.fora, prtc.lb, prtc.sai, prtc.dis, prtc.cnt, prtc.perif :
CHAN (particula) prtc.ent :
CHAN (coordenadas.auxiliar) aux.ent :
CHAN (INT; [ncoord]INT) coord.fora, xatual :
CHAN (INT) tipo.quad, a.tipo.quad, v.tipo.quad :
CHAN (saida.prtc) xsai.esq, xsai.dir :
CHAN (estado) est.prtc.sai, est.prtc.ent :
CHAN (entrada.prtc) xent :
[tamfila+1]CHAN (fila.saida) fila.esq, fila.dir :
CHAN (fila.saida) xsai.l&, xsai.l4, xent.l3, xent.l7 :
CHAN (REAL32) dist.quad, ac, iró, Epot, Ecin, Scin :
CHAN ([ncoord]REAL32) dist.coord, ac.prtc, ac.ren, vel, vel.ren :
CHAN (INT; REAL32) svel.esq, svel.dir :
CHAN ([ncoord]REAL32) media.esq, media.dir :
CHAN (REAL32; REAL32) Ener.ent, Ener.sai :

-- Rotina para o controle das listas de armazenamento atual
PROC cont.list.arm()
  INT g, rd, wr, i :
  [Num.max.Part][ncoord]INT x :
  PLACE x AT ????? :
  [Num.max.Part]INT proxpart :
  [2][Num.Graos.Div]INT primpart :
  PLACE proxpart AT ??? :
  PLACE primpart AT ??? :
  SEQ
    rd:=0
    wr:=1
    prim.esc IS primpart[wr] :
    SEQ g=0 FOR Num.Graos.Div
      prim.esc[g]:=Nulo
    WHILE g>Grao.term
      ALT
        nova.grao ? g
          IF
            g>=0
              primpart.grao IS primpart[wr][g] :
              SEQ
                nova.prtc ? i; x[i]
```

```
              proxpart[i]:=primpart.grao
              primpart.grao:=i
          g=Grao.Nulo
            SEQ
              wr:=wr><1
              rd:=rd><1
              prim.esc IS primpart[wr] :
              SEQ k=0 FOR Num.Graos.Div
                prim.esc[k]:=Nulo
              fim.quad ! TRUE
          TRUE
            SKIP
        grao.viz ? g
          SEQ
            i:=primpart[rd][g]
            WHILE i>=0
              SEQ
                coord.le ! i; x[i]
                i:=proxpart[i]
            fim.grao ! TRUE
:

-- Controle das listas de armazenamento das particulas externas
PROC cont.aux()
  VAL Num.Graos.Aux IS 2*ngy :
  INT g, rd, wr, i :
  [Num.max.Part][ncoord]INT xaux :
  [Num.max.Part]INT proxpart :
  [2][Num.Graos.Aux]INT primpart :
  [Num.Graos.Aux]INT contvez :
  PLACE xaux AT ??? :
  PLACE proxpart AT ??? :
  PLACE primpart AT ??? :
  PLACE contvez AT ??? :
  SEQ
    rd:=0
    wr:=1
    prim.esc IS primpart[wr] :
    SEQ g=0 FOR Num.Graos.Aux
      SEQ
        prim.esc[g]:=Nulo
        contvez[g]:=3
    WHILE g>Grao.term
      ALT
        nova.aux.grao ? g
          IF
            g>=0
              primpart.grao IS primpart[wr][g] :
              SEQ
                nova.paux ? i; xaux[i]
                proxpart[i]:=primpart.grao
                primpart.grao:=i
            g=Grao.Nulo
              SEQ
                wr:=wr><1
                rd:=rd><1
                prim.esc IS primpart[wr] :
                SEQ k=0 FOR Num.Graos.Aux
                  SEQ
                    prim.esc[k]:=Nulo
                    contvez[k]:=3
            TRUE
              SKIP
        grao.aux ? g
          conta IS contvez[g] :
          SEQ
            conta:=conta-1
            i:=primpart[rd][g]
            WHILE i>=0
              SEQ
                caux.le ! xaux[i]
                IF
                  conta<>0
                    SKIP
                  TRUE
                    paux.dis ! i
                i:=proxpart[i]
            fim.aux ! TRUE
  :

  -- Rotina para encontrar qual grao auxiliar a que a particula pertence
  PROC calc.g.aux()
    VAL max.aux IS ??? :            -- Numero maximo de particulas auxiliares
    BOOL fim :
    [max.aux]INT filaux :
    INT ind, g :
    [ncoord]INT x :
    PLACE filaux AT ??? :
```

```
  SEQ
    fim:=FALSE
    SEQ k=0 FOR max.aux
      filaux[k]:=k
    ind:=0
    WHILE NOT fim
      ALT
        (ind<max.aux) & aux.ent ? lado; x
          SEQ
            g:=x[1]/ngy + lado*ngy
            i:=filaux[ind]
            ind:=ind+1
            nova.aux.grao ! g
            nova.paux ! i; x
        paux.dis ? i
          IF
            i>=0
              SEQ
                ind:=ind-1
                filaux[ind]:=i
            TRUE
              fim:=TRUE
:

-- Rotina para encontrar o grao ao qual uma particula pertence
PROC calc.grao()
  VAL Num.Col.Div IS ngx/Num.Div.Proc :  -- num. de graos em x por divisao
  VAL conv IS [ngy, 1] :                 -- fatores para calculo de num. do grao
  BOOL fim, inter, exter :
  INT i, k, g, n :
  [ncoord]INT x, ga :
  [Num.Div.Proc][ncoord]INT gb :
  PLACE gb AT ??? :
  SEQ
    fim:=FALSE
    SEQ n=0 FOR Num.Div.Proc
      gbdiv IS gb[n] :
      SEQ k=0 FOR ncoord
        IF
          k=0
            gbdiv[k]:=n*Num.Col.Div
          TRUE
            gbdiv[k]:=0
    WHILE NOT fim
      SEQ
        inter:=TRUE
        exter:=TRUE
        WHILE (inter OR extern) AND NOT fim
          SEQ
            ALT
              (inter) & icalc ? i
                IF
                  (i>=0)
                    nova.coord ? x
                  TRUE
                    inter:=FALSE
              (exter) & prtc.ext ? i
                IF
                  (i>=0)
                    coord.ext ? x
                  TRUE
                    exter:=FALSE
              . fim.tst ? fim
                i:=Nulo
            IF
              (i>=0)
                SEQ
                  gbdesta IS gb[dp] :
                  SEQ k=0 FOR ncoord
                    ga[k]:=x[k]/ng[k]-gbdesta[k]
                  IF
                    (ga[0]>=0) AND (ga[0]<Num.Col.Div)
                      SEQ
                        g:=0
                        SEQ k=0 FOR ncoord
                          g:=g+ga[k]*conv[k]
                        nova.grao ! g
                        nova.prtc ! i; x
                        IF
                          ga[0]=0
                            coord.fora ! 3; x
                          ga[0]=Num.Col.Div-1
                            coord.fora ! 4; x
                          TRUE
                            SKIP
                    ga[0]<0
                      SEQ
                        prtc.fora ! i
```

```
                    coord.fora ! 1; x
                TRUE
                  SEQ
                    prtc.fora ! i
                    coord.fora ! 2; x
              TRUE
                SKIP
        IF
          NOT fim
            SEQ
              nova.grao ! Grao.Nulo
              prtc.fora ! Nulo
          TRUE
            SKIP
    nova.grao ! Grao.term
    prtc.fora ! Nulo
:

-- Rotina que envia uma particula se a mesma sair da divisao de processamento
PROC envia()
  BOOL fim :
  INT i, tag, tquad :
  [ncoord]INT x, xa :
  [ncoord]REAL32 a, v :
  SEQ
    fim:=FALSE
    tipo.quad ? tquad
    WHILE NOT fim
      SEQ
        prtc.fora ? i
        IF
          (i>=0)
            coord.fora ? tipo; x
            IF
              tipo=3
                xsai.esq ! 3 ; x
              tipo=4
                xsai.dir ! 3 ; x
              tipo=1
                SEQ
                  prtc.lb ! i
                  prtc.sai ! i
                  IF
                    tquad<>0
                      SEQ
                        est.prtc.sai ? tag; xa
                        xsai.esq ! 1 ; x; xa
                    TRUE
                      SEQ
                        est.prtc.sai ? tag; a; v
                        xsai.esq ! 2; x; a; v
              tipo=2
                SEQ
                  prtc.lb ! i
                  prtc.sai ! i
                  IF
                    tquad<>0
                      SEQ
                        est.prtc.sai ? tag; xa
                        xsai.dir ! 1 ; x; xa
                    TRUE
                      SEQ
                        est.prtc.sai ? tag; a; v
                        xsai.dir ! 2; x; a; v
          TRUE
            SEQ
              tipo.quad ? tquad
              xsai.esq ! 4
              xsai.dir ! 4
              IF
                tquad<0
                  PAR
                    fim.tst ! TRUE
                    fim:=TRUE
                    prtc.lb ! i
                    xsai.esq ! 5
                    xsai.dir ! 5
                TRUE
                  SKIP
:

-- Rotina para o controle da fila de numeros de particulas disponiveis
PROC cont.fila()
  INT k, i, ind, ANY :
  [Num.max.Part]INT fila :
  PLACE fila AT ??? :
  SEQ
    SEQ k=0 FOR Num.max.Part
```

```
      fila[k]:=k
    ind:=0
    i:=0
    WHILE i>=0
      ALT
        (ind<Num.max.Part) & pede.prtc ? ANY
          SEQ
            prtc.dis ! fila[ind]
            ind:=ind+1
        prtc.lb ? i
          SEQ
            fila[ind]:=i
            ind:=ind-1
    fim.junta ! TRUE
:

-- Rotina para juntas as particulas que veem da direita e da esquerda
PROC junta.ext()
  BOOL fim, cont.esq, cont.dir :
  INT tag, i :
  [ncoord]INT x, xa :
  [ncoord]REAL32 a, v :
  SEQ
    fim:=FALSE
    cont.esq:=TRUE
    cont.dir:=TRUE
    WHILE NOT fim
      SEQ
        ALT
          xent ? tag
            IF
              tag=1
                xent ? x; xa
              tag=2
                xent ? x; a; v
              TRUE
                cont:=FALSE
          fim.junta ? fim
            SKIP
        IF
          NOT fim
            IF
              tag<>3
                SEQ
                  pede.prtc ! TRUE
                  prtc.dis ? i
                  prtc.ext ! i
                  coord.ext ! x
                  IF
                    tag=1
                      SEQ
                        prtc.ent ! i
                        est.prtc.ent ! 1; xa
                    TRUE
                      SEQ
                        prtc.ent ! i
                        est.prtc.ent ! 2; a; v
              TRUE
                IF
                  cont
                    SKIP
                  TRUE
                    SEQ
                      prtc.ext ! Nulo
                      cont:=TRUE
          TRUE
            SKIP
:

-- Rotina para conversao e envio dos dados das particulas pelos links
PROC envio.dados()
  VAL Tam.Fila IS ????? :
  VAL Lim.Ind IS Tam.Fila-1-2*ncoord :
  BOOL fim, fim.esq, fim.dir, cont.esq, cont.dir :
  INT ind.esq, ind.dir, tag :
  [ncoord]INT x, xa, na, nv :
  []REAL32 a RETYPES na :
  []REAL32 v RETYPES nv :
  [Tam.Fila]INT fila.esq, fila.dir :
  SEQ
    fim:=FALSE
    cont.esq:=TRUE
    cont.dir:=TRUE
    ind.esq:=0
    ind.dir:=0
    fim.esq:=FALSE
    fim.dir:=FALSE
    WHILE NOT fim
```

```
PRI ALT
  (cont.esq AND (ind.esq<Lim.Ind)) & xsai.esq ? tag
    IF
      tag=3
        SEQ
          xsai.esq ? x
          fila.esq[ind.esq]:=tag
          ind.esq:=ind.esq+1
          [fila.esq FROM ind.esq FOR ncoord]:=x
          ind.esq:=ind.esq+ncoord
      tag=1
        SEQ
          xsai.esq ? x; xa
          fila.esq[ind.esq]:=tag
          ind.esq:=ind.esq+1
          [fila.esq FROM ind.esq FOR ncoord]:=x
          ind.esq:=ind.esq+ncoord
          [fila.esq FROM ind.esq FOR ncoord]:=xa
          ind.esq:=ind.esq+ncoord
      tag=2
        SEQ
          xsai.esq ? x; a; v
          fila.esq[ind.esq]:=tag
          ind.esq:=ind.esq+1
          [fila.esq FROM ind.esq FOR ncoord]:=x
          ind.esq:=ind.esq+ncoord
          [fila.esq FROM ind.esq FOR ncoord]:=na
          ind.esq:=ind.esq+ncoord
          [fila.esq FROM ind.esq FOR ncoord]:=nv
          ind.esq:=ind.esq+ncoord
      TRUE
        SEQ
          fila.esq[ind.esq]:=tag
          ind.esq:=ind.esq+1
          cont.esq:=FALSE
          IF
           tag=5
             fim.esq:=TRUE
           TRUE
             SKIP
  (cont.dir AND (ind.dir<Lim.Ind)) & xsai.dir ? tag
    IF
      tag=3
        SEQ
          xsai.dir ? x
          fila.dir[ind.dir]:=tag
          ind.dir:=ind.dir+1
          [fila.dir FROM ind.dir FOR ncoord]:=x
          ind.dir:=ind.dir+ncoord
      tag=1
        SEQ
          xsai.dir ? x; xa
          fila.dir[ind.dir]:=tag
          ind.dir:=ind.dir+1
          [fila.dir FROM ind.dir FOR ncoord]:=x
          ind.dir:=ind.dir+ncoord
          [fila.dir FROM ind.dir FOR ncoord]:=xa
          ind.dir:=ind.dir+ncoord
      tag=2
        SEQ
          xsai.dir ? x; a; v
          fila.dir[ind.dir]:=tag
          ind.dir:=ind.dir+1
          [fila.dir FROM ind.dir FOR ncoord]:=x
          ind.dir:=ind.dir+ncoord
          [fila.dir FROM ind.dir FOR ncoord]:=na
          ind.dir:=ind.dir+ncoord
          [fila.dir FROM ind.dir FOR ncoord]:=nv
          ind.dir:=ind.dir+ncoord
      TRUE
        SEQ
          fila.dir[ind.dir]:=tag
          ind.dir:=ind.dir+1
          cont.dir:=FALSE
          IF
           tag=5
             fim.dir:=TRUE
           TRUE
             SKIP
  (ind.esq > 0) & SKIP
    SEQ
      xsai.l& ! ind.esq; [fila.esq FROM 0 FOR ind.esq]
      IF
        NOT cont.esq
          IF
            fim.esq AND fim.dir
              fim:=TRUE
            NOT fim.esq
```

```
                cont.esq:=TRUE
              TRUE
                SKIP
          TRUE
            SKIP
        ind.esq:=0
      (ind.dir > 0) & SKIP
        SEQ
          xsai.14 ! ind.dir; [fila.dir FROM 0 FOR ind.dir]
          IF
            NOT cont.dir
              IF
                fim.esq AND fim.dir
                  fim:=TRUE
                NOT fim.dir
                  cont.dir:=TRUE
                TRUE
                  SKIP
            TRUE
              SKIP
          ind.dir:=0
:

-- Rotina para a recepcao de dados pelos links
PROC recep()
  INT total.dir, total.esq :
  [Tam.Fila]INT fila.dir, fila.esq :
  PROC esvazia(INT total.lado, []INT fila.lado, BOOL fim.lado)
    INT ind, tag :
    [ncoord]INT x, xa, nac, nv  :
    []REAL32 a RETYPES na :
    []REAL32 v RETYPES nv :
    SEQ
      ind:=0
      WHILE ind<total.lado
        SEQ
          tag:=fila[ind]
          ind:=ind+1
          IF
            tag=3
              SEQ
                x:=[fila.lado FROM ind FOR ncoord]
                ind:=ind+ncoord
                aux.ent ! 1; x
            tag=1
              SEQ
                x:=[fila.lado FROM ind FOR ncoord]
                ind:=ind+ncoord
                xa:=[fila.lado FROM ind FOR ncoord]
                ind:=ind+ncoord
                xent ! 1; x; xa
            tag=2
              SEQ
                x:=[fila.lado FROM ind FOR ncoord]
                ind:=ind+ncoord
                na:=[fila.lado FROM ind FOR ncoord]
                ind:=ind+ncoord
                nv:=[fila.lado FROM ind FOR ncoord]
                ind:=ind+ncoord
                xent ! 2; x; a; v
            tag=4
              xent ! 3
            TRUE
              fim.lado:=TRUE
  :
  SEQ
    fim.dir:=FALSE
    fim.esq:=FALSE
    WHILE NOT (fim.dir AND fim.esq)
      ALT
        (total.dir=0) & xent.13 ? total.dir; [fila.dir FROM 0 FOR total.dir]
          esvazia(total.dir, fila.dir, fim.dir)
        (total.esq=0) & xent.17 ? total.esq; [fila.esq FROM 0 FOR total.esq]
          esvazia(total.esq, fila.esq, fim.esq)
:

-- Rotina que controla a execucao dos diversos quadros
PROC quadros()
  VAL Max0 IS ngy :                    -- Num. de graos em uma lateral
  INT q, g :
  SEQ
    q:=0
    WHILE q<NQ
      SEQ
        SEQ g=0 FOR Num.Graos.Div
          grao.cent ! g
        fim.quad ?= TRUE
        q:=q+1
```

```
        SEQ
          pot.fim.quad ! TRUE
          cin.fim.quad ! TRUE
          grao.cnt ! Grao.Nulo
    PAR
      grao.cnt ! Grao.term
      pot.fim.prog ! TRUE
      cin.fim.prog ! TRUE
      fim.ren ! TRUE
:

-- Rotina para encontrar os vizinhos de um dados grao
PROC vizinhos()
  BOOL fim :
  INT g, l, c, lin, col, gviz :
  SEQ
    fim:=FALSE
    WHILE NOT fim
      SEQ
        grao.cent ? g
        IF
          g>=0
            SEQ
              grao.viz ! g
              l:=g/ngx
              c:=g REM ngx
              SEQ lin=l-1 FOR 3
                SEQ col=c-1 FOR 3
                  SEQ
                    IF
                      lin=-1
                        lin:=ngy-1
                      lin=ngy
                        lin:=0
                      TRUE
                        SKIP
                    IF
                      col=-1
                        SEQ
                          outro.grao ?= TRUE
                          terminou ! FALSE
                          grao.aux ! lin
                      col=ngx
                        SEQ
                          outro.grao ?= TRUE
                          terminou ! FALSE
                          grao.aux ! lin+ngy
                      TRUE
                        SEQ
                          gviz:=lin*ngx+col
                          IF
                            gviz>g
                              SEQ
                                outro.grao ?= TRUE
                                terminou ! FALSE
                                grao.viz ! gviz
                            TRUE
                              SKIP
              outro.grao ?= TRUE
              terminou ! TRUE
          g=Grao.Nulo
            fim.varr.quad ! TRUE
          TRUE
            SEQ
              fim:=TRUE
              fim.varr.grao ! TRUE
:


-- Rotina que gera todos os pares de particulas para todas as particulas de um
-- grao
PROC pares()
  BOOL fim, fim.viz, grao.cnt :
  [Max.Part.Viz][ncoord]INT x :
  [Max.Part.Viz]INT prtc :
  INT i, ult.cnt, ultima, cent, k :
  -- Rotina que testa distancia (se menos que o raio de corte)
  PROC teste.dis([ncoord]INT xi, [ncoord]INT xj, VAL INT j)
    VAL rc2 IS ??? :            -- Quadrado do raio de corte
    VAL rc2.sigma IS ??? :      -- Quadrado do raio de corte em coord. sigma
    VAL cnvsigma2 IS [???,???] : -- fatores de conversao quadratico para coordenadas de sigma
    [ncoord]REAL32 xij :
    INT k :
    REAL32 rij2 :
    SEQ
      xij[0]:=(REAL32 (xi[0] MINUS xj[0]))
      xij2[0]:=xij[0]*xij[0]
      IF
```

```
        xij2[0]<rc2
          SEQ
            xij[1]:=(REAL32 (xi[1] MINUS xj[1]))
            xij2[1]:=xij[1]*xij[1]
            IF
              xij2[1]<rc2
                SEQ
                  rij2:=xij2[0]*cnvsigma2[0]+xij2[1]*cnvsigma2[1]
                  IF
                    rij2<rc2.sigma
                      SEQ
                        prtc.perif ! j
                        dist.quad ! rij2
                        dist.coord ! xij
                    TRUE
                      SKIP
              TRUE
                SKIP
        TRUE
          SKIP
:
SEQ
  fim:=FALSE
  WHILE NOT fim
    ALT
      coord.le ? prtc[0]; x[0]
        SEQ
          prtc.cnt ! prtc[0]
          xatual ! prtc[0]; x[0]
          icalc ! prtc[0]
          ind:=1
          fim.viz:=FALSE
          grao.cnt:=TRUE
          WHILE NOT fim.viz
            ALT
              coord.le ? prtc[ind]; x[ind]
                SEQ
                  teste.dis(x[0], x[ind], prtc[ind])
                  ind:=ind+1
              caux.le ? x[ind]
                SEQ
                  teste.dis(x[0], x[ind], Num.max.Part+1)
                  prtc[ind]:=Num.max.Part+1
                  ind:=ind+1
              fim.grao ?= TRUE
                SEQ
                  IF
                    NOT grao.cnt
                      SKIP
                    TRUE
                      SEQ
                        ult.cnt:=ind
                        grao.cnt:=FALSE
                  outro.grao ! TRUE
                  terminou ? fim.viz
              fim.aux ?= TRUE
                SEQ
                  outro.grao ! TRUE
                  terminou ? fim.viz
          ultima:=ind
          SEQ cent=1 FOR ult.cnt-1
            x.cent IS x[cent] :
            prtc.cent IS prtc[cent] :
            SEQ
              prtc.perif ! Nulo
              prtc.cnt ! prtc.cent
              xatual ! prtc.cent; x.cent
              icalc ! prtc.cent
              SEQ k=cent+1 FOR ultima-cent-1
                teste.dis(x.cent, x[k], prtc[k])
          prtc.perif ! Nulo
      fim.varr.quad ? fim.quad
        SEQ
          prtc.cnt ! Nulo
          icalc ! Nulo
          xatual ! Nulo; x[ind]
      fim.varr.grao ? fim
        dist.quad ! -1.
:

-- Rotina para o calculo do fator comum da aceleracao
PROC acel()
  VAL kac IS ??? :              -- constante para o calculo da aceleracao
  REAL32 rij2, inrij2, inrij6 :
  SEQ
    dist.quad ? rij2
    WHILE rij2>=0.
      SEQ
```

```
        inrij2:=1./rij2
        inrij6:=(inrij2*inrij2*inrij2)
        ir6 ! inrij6
        ac ! inrij6*(inrij6-1.)*inrij2*kac
        dist.quad ? rij2
    fim.ac ! TRUE
:

-- Rotina para o calcula da energia potencial
PROC poten()
  BOOL fim :
  REAL32 u, inrij6 :
  SEQ
    fim:=FALSE
    u:=0.
    WHILE NOT fim
      ALT
        ir6 ? inrij6
          u:=u+inrij6*(inrij6-1.)
        pot.fim.quad ?= TRUE
          SEQ
            Epot ! u
            u:=0.
        pot.fim.prog ? fim
          SKIP
:

-- Rotina para o calculo da aceleracao em cada direcao
PROC ac.coord()
  BOOL fim :
  INT i, j, k, l, g :
  [ncoord]INT dx :
  REAL32 aclr :
  [Num.max.Part+1][ncoord]REAL32 acx :
  REAL32 act :
  PLACE acx AT ????? :
  SEQ
    fim:=FALSE
    WHILE NOT fim
      ALT
        prtc.cnt ? i
          SEQ
            SEQ l=0 FOR Num.max.Part
              acx.prtc IS acx[l] :
              SEQ k=0 FOR ncoord
                acx.prtc[k]:=0.
            a.tipo.quad ? tq
            WHILE i>=0
              SEQ
                prtc.perif ? j
                WHILE j>=0
                  SEQ
                    PAR
                      dist.coord ? dx
                      ac ? aclr
                    aci IS acx[i] :
                    acj IS acx[j] :
                    SEQ k=0 FOR ncoord
                      acik IS aci[k] :
                      acjk IS acj[k] :
                      SEQ
                        act:=aclr*dx[k]
                        acik:=acik+act
                        acjk:=acjk-act
                    prtc.perif ? j
                PAR
                  ac.prtc ! acx[i]
                  IF
                    tq<>0
                      SKIP
                    TRUE
                      ac.ren ! acx[i]
                prtc.cnt ? i
        fim.ac ? fim
          fim.coord ! TRUE
  :

  -- Rotina para o calculo das novas coordenadas
  PROC calc.coord()
    BOOL fim :
    [ncoord]REAL32 ac :
    [ncoord]INT dx, xn :
    SEQ
      fim:=FALSE
      WHILE NOT fim
        ALT
          ac.prtc ? ac
            SEQ
```

```
            coords ? dx
            SEQ k=0 FOR ncoord
              xn[k]:=dx[k] PLUS (INT ROUND ac[k])
            xnovo ! xn
          fim.coord ? fim
            fim.vel ! TRUE
:

-- Rotina para o calculo da velocidade da particula
PROC veloc()
  VAL kvel IS ??? :       -- constante para calculo de velocidade
  BOOL fim :
  [ncoord]INT x, xa, v :
  [ncoord]REAL32 v :
  INT k :
  SEQ
    fim:=FALSE
    v.tipo.quad ? tq
    WHILE NOT fim
      ALT
        xnovo ? x
          SEQ
            xanter ? xa
            SEQ k=0 FOR ncoord
              v[k]:=(REAL32 (xn[k] MINUS xa[k]))*kvel
            vel ! v
            IF
              tq<>0
                SKIP
              TRUE
                vel.ren ! v
            nova.coord ! xn
          v.tipo.quad ? tq
            SKIP
          fim.vel ? fim
            SKIP
:

-- Rotina para calculo da energia cinetica
PROC cinet()
  REAL32 SEc, Ec :
  BOOL fim, fim.quadro :
  INT k :
  [ncoord]INT v :
  SEQ
    SEc:=0.
    Ec:=0.
    fim:=FALSE
    WHILE NOT fim
      ALT
        vel ? v
          SEQ k=0 FOR ncoord
            Ec:=Ec+v[k]*v[k]
        cin.fim.quad ?= TRUE
          SEQ
            SEc:=SEc+Ec
            Ecin ! Ec
            Ec:=0.
        pede.scin ?= TRUE
          SEQ
            Scin ! SEc
            SEc:=0.
        cin.fim.prog ? fim
          Ecin ! -1.
:

-- Rotina de renormalizacao das velocidades
PROC renorm()
  VAL dt IS ??? :             -- delta t
  VAL m.dt2 IS ??? :          -- meio delta t ao quadrado
  VAL txrnm IS ??? :          -- numero de quadros para renormalizacao
  VAL Trq IS ??? :            -- temperatura requerida
  INT tq, k, i, np, npe, k :
  BOOL fim :
  [ncoord]REAL32 vmed, velf, svm :
  [ncoord]INT x, dif, xat :
  [Num.max.Part][ncoord]REAL32 v, a :
  [Num.max.Part][ncoord]INT xa :
  [ncoord]REAL32 med :
  PLACE v AT ???? :
  PLACE a AT ???? :
  PLACE xa AT ???? :
  REAL32 s :
  SEQ
    tq:=0
    fim:=FALSE
    SEQ k=0 FOR ncoord
      vmed[k]:=0.
```

```
WHILE NOT fim
  PRI ALT
    xatual ? i; x
      IF
        (i>=0)
          SEQ
            IF
              tq<>1
                xat:=xa[i]
              TRUE
                ai IS a[i] :
                vi IS v[i] :
                SEQ k=0 FOR ncoord
                  xat[k]:=x[k]-(INT (vi[k]-med[k])*dt*velf)-(INT ai[k]*m.dt2)
            xanter ! xat
            SEQ k=0 FOR ncoord
              dif[k]:=x[k]<<1 MINUS xat[k]
            coords ! dif
            IF
              tq<>0
                SKIP
              TRUE
                PAR
                  ac.ren ? a[i]
                  vi IS v[i] :
                  SEQ
                    vel.ren ? vi
                    SEQ k=0 FOR ncoord
                      vmed[k]:=vmed[k]+vi[k]
                    np:=np+1
        TRUE
          SEQ
            tq:=tq+1
            IF
              (tq<>1) AND (tq<>txrnm)
                SKIP
              tq=1
                SEQ
                  IF
                    dp<>0
                      svel.esq ? npe; svm
                    TRUE
                      SEQ
                        SEQ k=0 FOR ncoord
                          svm[k]:=0.
                        npe:=0
                  SEQ k=0 FOR ncoord
                    svmk IS svm[k] :
                    svmk:=svmk+vmed[k]
                  npe:=npe+np
                  IF
                    dp<>Num.Div.Proc
                      svel.dir ! npe; svm
                    TRUE
                      SEQ
                        SEQ k=0 FOR ncoord
                          med[k]:=svm[k]/npe
                        media.esq ! med
                  IF
                    (dp<>0) AND (dp<>Num.Div.Proc)
                      SEQ
                        media.dir ? med
                        media.esq ! med
                    p=0
                      media.dir ? med
                    TRUE
                      SKIP
                  pede.scin ! TRUE
                  Scin ? s
                  velf:=sqrt(Trq*txrnm/s)
              TRUE
                SEQ
                  tq:=0
                  SEQ i=0 FOR Num.max.Part
                    vi IS v[i] :
                    SEQ k=0 FOR ncoord
                      vi[k]:=0.
      PAR
        tipo.quad ! tq
        v.tipo.quad ! tq
        a.tipo.quad ! tq
  prtc.ent ? i
    SEQ
      est.prtc.ent ? tag
      IF
        tag=1
          SEQ
            est.prtc.ent ? xa[i]
```

```
            TRUE
              SEQ
                est.prtc.ent ? i; a[i]; v[i]
                vi IS v[i] :
                SEQ k=0 FOR ncoord
                  vmk IS vmed[k] :
                  vmk:=vmk+vi[k]
                np:=np+1
        prtc.sai ? i
          IF
            tq<>0
              est.prtc.sai ! 1; xat[i]
            TRUE
              SEQ
                est.prtc.sai ! 2; a[i]; v[i]
                np:=np-1
                vi IS v[i] :
                SEQ k=0 FOR ncoord
                  vmk IS vmed[k] :
                  vmk:=vmk-vi[k]
        fim.ren ? fim
          tipo.quad ! -1
:

-- Rotina de totalizacao e comunicacao das energias
PROC energias()
  REAL32 ecint, ec7, u7, uint, u, ec :
  SEQ
    Ecin ? ecint
    WHILE ecint >= 0.
      SEQ
        u:=0.
        ec:=0.
        Ener.ent ? ec7; u7
        Epot ? uint
        ec:=ec7+ecint
        u:=u7+uint
        Ener.sai ! ec; u
        Ecin ? ecint
:
```