USP / IFQSC / SBI
8-2-000984

Universidade de São Paulo

Instituto de Física e Química de São Carlos

Proposta e implementação de um sistema de Processamento Paralelo para Dinâmica Molecular

Gonzalo Travieso

Tese apresentada ao
Instituto de Física e Química de São Carlos,
Para a obtenção do título de
Doutor em Ciências (Física Aplicada)

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

Departamento de Física e Ciência dos Materias
São Carlos - 1993

SERVIÇO DE BIBLIOTECA E INFORMAÇÃO - IFQSC
FÍSICA

UNIVERSIDADE DE SÃO PAULO
Instituto de Física e Química de São Carlos

Fone (0162) 72-6222
Fax (0162) 72-2218

Av. Dr. Carlos Botelho, 1465
Caixa Postal 369
CEP 13560.970 - São Carlos - SP
Brasil

MEMBROS DA COMISSAO JULGADORA DA TESE DE DOUTORADO DE **GONZALO TRAVIESO** APRESENTADA AO INSTITUTO DE FISICA E QUIMICA DE SAO CARLOS, DA UNIVERSIDADE DE SAO PAULO, EM 25/05/1993

COMISSAO JULGADORA:

Prof.Dr.Jan Frans Willem Slaets

Prof.Dr.Roland Koberle

Prof.Dr.Luciano da Fontoura Costa

Prof.Dr.José Pedro Rino

Prof.Dr.Eduardo W.Beroamini

Resumo

Neste trabalho, propomos um método de paralelização de simulações de dinâmica molecular para execução em máquinas de memória distribuída, trabalhando sob passagem de mensagens. Nos limitamos à análise de implementação de sistemas com ensemble microcanônico de partículas de Lennard-Jones, desenvolvendo no entanto um sistema que pode ser expandido para incluir outras características. Mostramos que o sistema apresenta bom desempenho com relação à paralelização, representando uma alternativa viável para a simulação de sistemas com muitas partículas.

Abstract

A method for parallelization of molecular dynamics simulations in distributed-memory machines operating under the message-passing paradigm is proposed. Even though the analysis and implementation presented in this work were primarily restricted to the system model known as microcanonical ensemble of Lennard-Jones particles, the obtained system can easily be adapted for inclusion of other characteristics. It is shown that the developed system is well suited for parallelization, resulting in a feasible alternative for simulation of many-particle systems.

Conteúdo

1 Introdução **7**

2 Dinâmica Molecular **9**

2.1 Conceitos básicos 9
2.2 Simulações de mecânica estatística 12
2.3 Tipos de partículas 13
2.4 Equações de movimento 15
2.5 Simulações dos diferentes ensembles 17
2.6 Levantamento de valores 20
2.6.1 Quantidades termodinâmicas 20
2.6.2 Quantidades estruturais 21
2.6.3 Correções de longo alcance 21
2.7 Condições de contorno 22
2.8 Considerações adicionais 25
2.8.1 Configuração inicial 25
2.8.2 Termalização 25
2.8.3 Conservação da energia 26
2.9 Conclusão 26

3 Discretização e representação de dados **27**

3.1 Integração das equações de movimento 27
3.1.1 Método de Runge-Kutta 28
3.1.2 Método preditor-corretor de Gear 28
3.1.3 Método de diferença central de Störmer 29
3.1.4 Comparação dos métodos 30
3.2 Representação dos dados e cálculo de valores 32
3.3 Implementação das condições de contorno periódicas 33
3.4 Considerações adicionais 35
3.5 Conclusões 36

4 Paralelização **37**
4.1 Algoritmos para a otimização do cálculo de forças 37
4.1.1 Algoritmo de tabela de vizinhos 38
4.1.2 Algoritmo de células 38
4.1.3 Combinação dos algoritmos 40
4.2 Propostas de implementação paralela 41
4.2.1 Implementações em supercomputadores de propósito geral . . 41
4.2.2 Implementações em arquiteturas paralelas de propósito específico 42
4.2.3 Implementações em arquiteturas paralelas de propósito geral . 43
4.3 Forma proposta de paralelização 44
4.3.1 *Transputers* e occam 44
4.3.2 Separação em anel 46
4.3.3 Duplicação de células 48
4.3.4 Divisão homogênea ou não-homogênea 51
4.3.5 Estrutura global do programa 51
4.3.6 Limitações da paralelização 53
4.3.7 Estrutura das ligações 53
4.4 Conclusão 55

5 Avaliação e resultados **57**
5.1 Validação 57
5.2 Tempo de processamento 57
5.2.1 Cálculo do tempo de processamento 59
5.2.2 Levantamento do tempo de processamento 61
5.2.3 Variação com o número de partículas 63
5.3 Conclusão 66

6 Conclusões e futuros trabalhos **69**
6.1 Conclusão 69
6.2 Futuros trabalhos 70

A Arquivos de configuração para a placa B008 **75**
A.1 Um processador 75
A.2 Dois processadores 75
A.3 Três processadores 75
A.4 Quatro processadores 76

B Código occam de dinâmica molecular **77**

Lista de Figuras

2.1 Interações de partículas dentro de um raio de corte r_c 14
2.2 Repetição do espaço de simulação em condições de contorno periódicas 23
2.3 Convenção da imagem mínima sob condições periódicas: no caso r_a, a distância se calcula no espaço normal, enquanto que no caso r_b a distância utiliza uma imagem de uma das partículas 24

4.1 Divisão do espaço de simulação em células e busca das interagentes nas células vizinhas 39
4.2 Varredura de metade das células vizinhas a uma dada célula. O hachurado simples indica a célula central, e o hachurado duplo indica as células vizinhas que devem ser varridas. 39
4.3 Representação esquemática de transputers. Em (a), temos um *transputer* e seus quatro *links*; em (b) uma representação simplificada e em (c), uma possível forma de interligação de um conjunto de *transputers* 45
4.4 Divisão do espaço de simulação e anel de processamento 47
4.5 Representação das interligações entre os processadores, para o caso de 4 processadores de cálculo (Pn) e 1 de controle separado (C) 48
4.6 Duplicação das células nas bordas. As linhas tracejadas indicam as colunas de células duplicadas 50
4.7 Descrição em pseudo-código do programa implementado 52
4.8 Representação das interligações da placa B008 54
4.9 Representação das interconexões e distribuição utilizada neste trabalho (no exemplo, para 4 *transputer* de cálculo (Pn) e um de controle (C)) 55

5.1 Variação de $\langle \Delta H^2 \rangle^{1/2}$ com Δt, para o caso bidimensional 58
5.2 Variação de $\langle \Delta H^2 \rangle^{1/2}$ com Δt, para o caso tridimensional 58
5.3 Tempos de um passo de simulação para o caso bidimensional 63
5.4 Tempos de um passo de simulação para o caso bidimensional (extrapolados) 64
5.5 Tempos de um passo de simulação para o caso tridimensional 64
5.6 Tempos de um passo de simulação para o caso tridimensional (extrapolados) 65
5.7 $T \times N$ para o caso bidimensional 65

5.8 $T \times N$ para o caso tridimensional 66
5.9 *Speedups* para o caso bidimensional 67
5.10 *Speedups* para o caso bidimensional (com extrapolação) 67
5.11 *Speedups* para o caso tridimensional 68
5.12 *Speedups* para o caso tridimensional (com extrapolação) 68

Lista de Tabelas

3.1 Levantamento da flutuação média quadrática da energia para simulações com 32 e 64 bits. O ponto de simulação corresponde a $\rho = 0.8442$ e $T = 0.524$, com 2048 partículas. A simulção envolveu um tempo total $t = 1.0$. 34

3.2 Comparação de armazenamento de posições em inteiro ou ponto flutuante (real). $\rho = 0.8442$, $\Delta t = 0.001$, 1000 passos de simulação, $N = 2048$ 35

3.3 Comparação de precisão e tempo de processamento entre tabela de *look-up* e cálculo direto para o potencial de Lennard-Jones. 36

4.1 Tempos de processamento médios em 128 passos com $\rho = 0.8442$ e $N = 2048$ num sistema bidimensional, para o algoritmo completo e o algoritmo sem redistribuição. 40

4.2 Comparação entre o método de células e o método combinado, com relação ao tempo de processamento, tanto para 2 como 3 dimensões. $N = 2048$, $\rho = 0.8442$. 41

5.1 Tempos de simulação (em segundos) para o caso bidimensional 62

5.2 Tempos de simulação (em segundos) para o caso tridimensional . . . 62

Capítulo 1

Introdução

O uso de computadores como ferramentas de estudo em física, e como um elemento de ligação entre teoria e experimento já é amplamente difundido [Abr86, AT87].

Um dos campos de aplicação é no estudo de *dinâmica molecular*, onde o computador é utilizado para realizar simulaçôes a fim de examinar o comportamento e determinar fatores a nível molecular em materiais.

As demandas computacionais desse problema são extremamente elevadas [Abr86]. Para citar os pontos principais envolvidos diremos que é simultaneamente requerido um tempo de processamento relativamente curto para cada simulação, e uma grande disponibilidade do sistema computacional, de forma a que o pesquisador possa testar diversas possibilidades, até encontrar os parâmetros adequados para suas simulações.

Essas elevadas demandas levaram a diversas pesquisas, tanto em termos de algoritmo como de processadores, visando à determinação de formas de execução eficiente. Notadamente, podemos citar os métodos de implementação em supercomputadores [GDK89], os processadores de propósito específico [BGGT90, APB+87] e as implementações em máquinas paralelas de propósito geral [GKPV88, PP89].

O presente trabalho visa à discussão de uma forma de implementação de dinâmica molecular em uma rede de *transputers*, programada em `occam`, conectada a um computador pessoal do tipo IBM-PC que serve de hospedeiro.

No capítulo 2 temos uma apresentação breve dos aspectos de dinâmica molecular relevantes ao presente estudo, juntamente com a delimitação do problema de dinâmica molecular com o qual lidaremos

O capítulo 3 apresenta um resumo de técnicas de implementação numérica e representação de dados, dando ênfase aos métodos mais utilizados no campo de dinâmica molecular. São também discutidos alguns aspectos da implementação.

O capítulo 4 apresenta e justifica o tipo de algoritmo que será utilizado e o método de paralelização escolhido, juntamente com uma revisão de métodos propostos de paralelização. Também é dada a descrição da forma de interligação dos diversos *transputers*.

O capítulo 5 apresenta a validação (teste de correção da implementação), o desenvolvimento de expressões para o tempo e o levantamento de tempos de processamento

sob diversas condições de número de partículas e processadores.

Capítulo 2

Dinâmica Molecular

Introduziremos neste capítulo os conceitos básicos de mecânica estatística e dinâmica molecular que serão necessários para a compreensão do trabalho realizado e para discussões posteriores. Gostaríamos de indicar que a introdução apresentada não pretende ser abrangente nem no campo de mecânica estatística nem de dinâmica molecular, mas apenas servir como base para as apresentações dos resultados a seguir. [LL58] é uma boa introdução aos conceitos de mecânica estatística, e [AT87] apresenta um estudo abrangente de técnicas de dinâmica molecular e Monte Carlo.

Outro ponto a ser ressaltado é que tratamos apenas de sistemas em equilíbrio. Para estudos de sistemas fora de equilíbrio, alguns dos elementos aqui apresentados podem ser utilizados, mas tomando-se o cuidado com a análise de seu campo de validade. [AT87] e [Hoo86] apresentam estudos de sistemas fora de equilíbrio.

2.1 Conceitos básicos

Num estudo de mecânica estatística, cada sistema pode ser enfocado à partir de dois pontos de vista distintos, que chamaremos de *macroscópico* e *microscópico*, respectivamente.

Do ponto de vista macroscópico, o sistema em equilíbrio é completamente determinado por certo número de variáveis termodinâmicas, como o número de partículas N, a pressão P e a temperatura T. Um conjunto de variáveis desse tipo define o chamado *estado termodinâmico* do sistema, à partir do qual diversas outras quantidades termodinâmicas do sistema podem ser calculadas, como por exemplo, densidade ρ, potencial químico μ, capacidade térmica C_V, etc.

Do ponto de vista microscópico, o sistema é determinado pelo estado de todas as partículas de que é constituido. Assim, num sistema com N partículas em três dimensões, temos $3N$ valores para as posições e $3N$ valores para as velocidades que caracterizam o *estado microscópico*[1] do sistema.

[1] Onde não houver possibilidade de confusão, utilizaremos a palavra *estado* tanto para o estado termodinâmico como para o estado microscópico do sistema.

Se considerarmos todos os possíveis estados que um sistema com N partículas pode assumir, veremos que se forma um espaço com 3 dimensões para cada possível posição e 3 dimensões para cada possível velocidade de cada uma das partículas, totalizando $6N$ dimensões. Chamamos esse espaço de *espaço de estado* do sistema. Portanto, o estado microscópico do sistema em um dado instante será representado por um ponto no espaço de estado. Chamaremos esse ponto de Γ, e a função $\Gamma(t)$ descreve o desenvolvimento do sistema no tempo.

Os valores medidos a nível macroscópico de variáveis como a temperatura, pressão, etc., estão relacionados com valores médios de funções do estado microscópico [LL58]. Assim, seja X uma quantidade que é função do estado microscópico do sistema, $X(\Gamma(t))$, e $\hat{X}$ a correspondente variável termodinâmica. Devemos então ter:

$$\hat{X} = \langle X \rangle_t = \langle X(\Gamma(t)) \rangle_t = \lim_{\tau \to \infty} \frac{1}{\tau} \int_0^\tau X(\Gamma(t)) dt \tag{2.1}$$

onde $\langle X \rangle_t$ indica média da quantidade X no tempo.

Em mecânica estatística é mais utilizada uma outra forma de realização da média que apresentaremos agora. Ao invés de lidar com apenas um sistema, e seguir toda a evolução temporal do mesmo, pressupomos uma grande quantidade de sistemas semelhantes quanto ao número e tipo de partículas e interação com o ambiente, mas estando cada qual em um estado microscópico distinto, e cada sistema sendo completamente independente dos outros. Esse conjunto de sistemas forma o que é chamado de *ensemble*.

Obviamente, cada um dos sistemas será representado por um ponto distinto no espaço de estado. Associamos então, a cada ponto do espaço de estado, uma probabilidade de que esse ponto seja ocupado por um sistema, e formamos com isso uma *função distribuição de probabilidade*, ϱ.

Conforme os sistemas evoluem independentemente, os pontos que os representam descrevem trajetórias nesse espaço de estado. Desta forma, a função distribuição de probabilidade é, em geral, uma função do tempo. No entanto, para sistemas em equilíbrio, essa função tende a uma distribuição de equilíbrio, sendo que a forma dessa distribuição é determinada pelo tipo de interação com o ambiente. Desta forma, temos uma grande quantidade de possíveis ensembles de equilíbrio, sendo que os mais importantes serão indicados abaixo. Nas descrições abaixo, assumiremos que os sistemas podem ser descritos por uma hamiltoniana (ver seção 2.4) $\mathcal{H} = \mathcal{H}(\Gamma)$, k é a constante de Boltzmann, T é a temperatura absoluta, N é o número de partículas, V o volume do sistema, P a pressão, E a energia total, μ o potencial químico específico e δ representa o delta de Dirac.

Ensemble microcanônico : Neste caso, são considerados constantes N, V e E. A função de distribuição para este ensemble é:

$$\varrho_{NVE} = \delta(\mathcal{H} - E) \tag{2.2}$$

Ensemble canônico : Aqui, consideramos constantes N, V e T, e a função distribuição de probabilidade fica:

$$\varrho_{NVT} = e^{-\mathcal{H}/kT} \tag{2.3}$$

Ensemble isotérmico-isobárico : Mantém constantes os valores de N, T e P, com função distribuição dada por:

$$\varrho_{NPT} = e^{-(\mathcal{H}+PV)/kT} \tag{2.4}$$

Ensemble grande canônico : Aqui são mantidos constantes μ, V e T. A função distribuição de probabilidades fica:

$$\varrho_{\mu VT} = e^{-(\mathcal{H}-\mu N)/kT} \tag{2.5}$$

Devemos indicar que as expressões "mantidos constantes" acima não significam necessariamente que os valores das quantidades fiquem absolutamente invariáveis, mas sim que as suas médias termodinâmicas permaneçam constantes, isto é, não apresentem tendências de longo prazo. Um ponto importante a ressaltar é que os diversos ensembles são equivalente no limite termodinâmico de número muito grande de partículas, sendo as diferenças entre os valores de propriedades levantadas em todos eles equivalente dentro de um erro da ordem $\mathcal{O}(1/N)$ [LL58].

Quanto à questão da equivalência entre as médias temporais e a média nos ensembles, ela é abordada nos textos clássicos de mecânica estatística, envolvendo o conceito de *ergodicidade*. Somente queremos frisar aqui um ponto importante relativamente a isso para as simulações por computador. Dizemos que um sistema é *ergódico* quando, dado tempo suficientemente longo, ele passa arbitrariamente próximo de todos os pontos no espaço de estado onde ϱ é diferente de zero. Seja $\mathcal{P}$ a região do espaço de estado onde ϱ é diferente de zero. Se um sistema parte de condições iniciais dadas, e evolue de acordo com um conjunto fixo de equações de movimento, então é possível que ele seja levado ao ponto de partida sem haver passado por toda a região $\mathcal{P}$[2]. Em tal caso não-ergódico, os resultados levantados não seriam representativos do sistema sob estudo. Além disso, mesmo em sistemas teoricamente ergódicos, o tempo para a varredura de uma fração apropriada do espaço de estado pode ser extremamente alto, e portanto os resultados de uma simulação podem ser parciais. Isso indica o cuidado que o pesquisador deve ter ao planejar as simulações e interpretar seus resultados.

Indiquemos aqui também que o primeiro caso acima, em que o sistema volta para suas condições iniciais, é extremamente improvável, tanto no caso real, como no caso de simulação. Dado o caráter caótico do sistema [Hoo86], por mais particulares que sejam as condições iniciais, as interações com o ambiente em um sistema real, que

[2] Neste caso, o sistema se encontra num estado *metaestável*.

representam contínuas alterações aleatórias no estado microscópico do sistema, garantem a ergodicidade [LL58][3]. Já no caso de simulação, temos o efeito pseudo-aleatório dos arredondamentos, que possuem ação semelhante. No entanto, esses efeitos não são totalmente aleatórios[4], e portanto, apesar de diminuída, a probabilidade de ocorrer um ciclo fechado não é zero.

Além disso, os resultados das simulações serão prejudicados não só se um estado totalmente metaestável for atingido, mas também se um estado "aproximadamente metaestável" for atingido. Com isso queremos dizer um estado que apesar de não se repetir exatamente, repete-se com pequenas modificações, de forma que a maior parte da amostragem se faz em torno de uma região limitada do espaço de estado, devido ao tempo finito de simulação.

2.2 Simulações de mecânica estatística

Apresentaremos agora os pontos básicos relativos às simulações para estudos de mecânica estatística.

Como vimos, existem fundamentalmente dois tipos de realização de médias, que são as médias temporais e as médias no espaço de estado. Isto gerou duas formas de simulação por computador, que são comumente chamadas *dinâmica molecular* e *método de Monte Carlo*.

O método de Monte Carlo, também chamado *método de Metropolis*, por ter sido utilizado pela primeira vez nas simulações descritas em [MRR+53], parte de uma tentativa de amostragem direta no espaço de estado. Assim, são geradas, de acordo com uma prescrição estatística, diversas configurações, e os valores das variáveis de interesse são calculados para essas diversas configurações.

Em dinâmica molecular, procura-se executar uma média no tempo, de forma que um sistema tem uma configuração inicial gerada, e à partir desse ponto, sua evolução temporal é simulada de acordo com as equações de movimento. À partir disso se geram os dados que podem ser utilizados para o cálculo das quantidades de interesse.

Assim, para a execução de uma simulação de dinâmica molecular, devemos estabelecer alguns pontos:

Tipo de partículas : Quais características das partículas devem ser introduzidas na simulação? Qual seu formato? Quais os potencias de interação? Existem algumas restrições a serem aplicadas?

[3]Dito de outra forma: a probabilidade de o sistema ficar preso num ciclo sem percorrer alguma região do espaço de estado é muito menor que a de passar arbitrariamente próximo de todos os pontos da região $\mathcal{P}$.

[4]O erro de arredondamento age de forma aleatória apenas na medida em que ele é um fator *externo* ao problema que está sendo simulado. Entretanto, como os cálculos e arredondamentos são determinísticos, se o sistema por alguma razão voltar para o estado inicial, então ele prosseguirá da mesma forma que anteriormente, isto é, entrará num estado metaestável.

Equações de movimento : Quais as equações que determinam a forma de movimento das partículas?

Levantamento de valores : Como se fazem os cálculos das quantidades termodinâmicas ou outras de interesse?

Condições de contorno : Devem ser efetivamente estabelecidas condições de contorno para a integração das equações de movimento das partículas.

Passaremos agora a abordar essas questões, e mais algumas de carater prático, nas próximas seções.

2.3 Tipos de partículas

Podemos classificar os principais tipos de partículas utilizadas em simulações de dinâmica molecular da seguinte forma:

Esferas rígidas : Uma das formas mais simples de introduzir as interações entre as partículas é considerá-las esféricas e rígidas, movendo-se retilineamente entre colisões e sofrendo de tempos em tempos colisões perfeitamente elásticas com outras partículas. Exemplos de simulações com esferas rígidas são apresentados numa série de artigos de Alder e Wainwright ([AW59], [AW70]).

Potenciais suaves : Outra forma de introduzir as interações entre as partículas em uma simulação é considerando que essas interações sejam descritas por uma função potencial contínua. Entre os diversos tipos de potenciais utilizados, citemos o Lennard-Jones, dado pela fórmula abaixo [AT87]:

$$\mathcal{V}_{ij} = 4\epsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right] \tag{2.6}$$

onde σ é a distância para a qual o potencial se anula, e ϵ é o valor mínimo do potencial, e o de Stillinger-Weber [SW85]. Absteremo-nos de apresentar aqui a expressão do potencial de Stillinger-Weber, mas notaremos apenas os seguintes pontos: é um potencial que considera, além dos pares de partículas, também as triplas de partículas; e envolve o cálculo de exponenciais.

Exemplos de simulações com o potencial de Lennard-Jones são as de [Ver67]. O potencial de Stillinger-Weber é utilizado freqüentemente em simulações em silício, onde os potenciais de pares não atingem bons resultados [BGGT90].

Sistemas moleculares : Em sistemas que consistem de moléculas com átomos ligados quimicamente, a simulação quântica das ligações seria muito dispendiosa

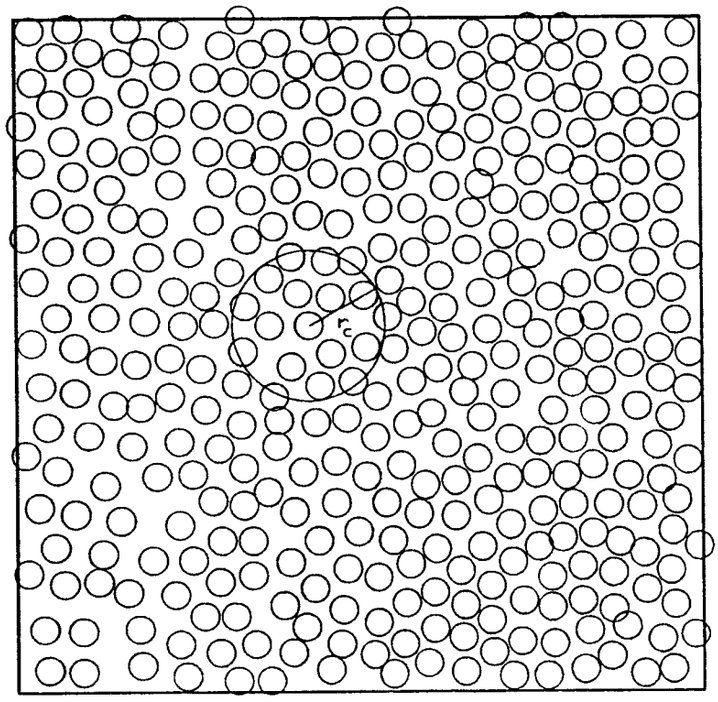

Figura 2.1: Interações de partículas dentro de um raio de corte r_c

computacionalmente, de forma que em geral simulam-se as ligações químicas como ligações rígidas entre os átomos componentes da molécula. Exemplos podem ser encontrados em [BLQ73] e [STS77].

Um fator de extrema importância para a organização da simulação relativo ao potencial utilizado é a sua distância de atuação. Diremos que um potencial é de *curta distância* se ele decai com rapidez suficiente para que as interações entre partículas precisem ser calculadas apenas dentro de um distância r_c, chamada *raio de corte*. Podemos portanto considerar, no cálculo da força sobre dada partícula, apenas as interações dessa partícula com aquelas que se encontram a uma distância menor ou igual a r_c (fig. 2.1).

O truncamento do potencial em r_c apresenta diversos efeitos na simulação, por transformar o potencial numa função não-contínua. Um dos mais marcantes é a flutuação da energia total quando da saída ou entrada de partículas do raio de influência de outras. Estes fatores devem ser levados em consideração na análise dos resultados (ver página 26).

Os potenciais de Lennard-Jones e Stillinger-Weber são potencias de curta distância. No entanto, as forças eletrostáticas são de *longa distância*. Se é necessária a inclusão de forças eletrostáticas na simulação, então precisam-se considerar as interações entre

todas as partículas simuladas, e ainda, no caso de condições de contorno periódicas (veja seção 2.7), é necessária também a utilização da soma de Ewald [AT87] para levar em conta os efeitos das diversas imagens periódicas de cada partícula.

Para o potencial de Lennard-Jones costumam ser utilizadas unidades *reduzidas*, onde a unidade para energias é de forma que $\epsilon = 1$, a de distâncias de forma que $\sigma = 1$, e as massas das partículas são unitárias, $m = 1$. Normalmente, para indicar que uma quantidade está expressa em unidades reduzidas, indicamo-la com um asterisco. Por exemplo: ρ^*, T^*, etc. No entanto, neste trabalho, como lidaremos apenas com quantidades reduzidas, estas serão expressas sem o asterico e, à partir deste ponto, assumimos que todas as quantidades apresentadas são reduzidas (a menos que expressamente indicado em contrário).

Neste trabalho, consideraremos apenas potenciais contínuos de curta distância, particularmente o de Lennard-Jones, e trabalharemos com sistemas de partículas idênticas.

2.4 Equações de movimento

Para a descrição do movimento de um sistema de partículas, dispomos de diversos conjuntos de equações. Em primeiro lugar, temos as equações de Newton, que em coordenadas cartesianas ficam:

$$m_i \ddot{\mathbf{r}}_i = \mathbf{f}_i \tag{2.7}$$

onde o índice i se extende por todas as N partículas do sistema, m_i é a massa da partícula i, $\mathbf{r}_i$ o vetor que indica sua posição, e $\mathbf{f}_i$ a força que age sobre essa partícula. A força pode ser calculada por:

$$\mathbf{f}_i = -\nabla_{\mathbf{r}_i} \mathcal{V} \tag{2.8}$$

onde $\mathcal{V}$ é a energia potencial do sistema.

Uma outra formulação é a de Lagrange. Neste caso, definimos um conjunto de G *coordenadas generalizadas* q_k, onde G é o número de graus de liberdade do sistema, e uma função $\mathcal{L}$ chamada *lagrangeana* que depende das coordenadas generalizadas e das velocidades generalizadas que lhe correspondem, $\dot{q}_k$, além do tempo: $\mathcal{L} = \mathcal{L}(q, \dot{q}, t)$.[5]. Para o presente trabalho, têm interesse apenas lagrangeanas independente do tempo. As equações de movimento ficam, então:

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_k} - \frac{\partial \mathcal{L}}{\partial q_k} = 0 \tag{2.9}$$

Constituem-se portanto num sistema de G equações diferenciais.

[5] q indica o conjunto de todas as q_k. O mesmo vale para $\dot{q}$.

A lagrangeana $\mathcal{L}$ pode ser decomposta em duas funções, uma dependente unicamente das velocidades generalizadas, e outra dependente unicamente das coordenadas generalizadas:

$$\mathcal{L}(q,\dot{q}) = \mathcal{K}(\dot{q}) - \mathcal{V}(q) \tag{2.10}$$

$\mathcal{K}$ representa a *energia cinética* do sistema, e $\mathcal{V}$ sua *energia potencial*.

Se consideramos um sistema simples de partículas livres, e utilizamos coordenadas cartesianas, então as equações (2.9) se reduzem às equações (2.7). A vantagem na utilização das equações de Lagrange surge quando desejamos impor restrições holonômicas ao movimento de algumas ou todas as partículas. Restrições *holonômicas* são aquelas que estão relacionadas apenas com a geometria do sistema. Este é o caso, por exemplo, na simulação de sistemas moleculares com ligações rígidas.

Na formulação de Hamilton, utilizamos coordenadas generalizadas q_k e os *momentos* associados com elas p_k, definidos por:

$$p_k = \frac{\partial \mathcal{L}}{\partial \dot{q}_k} \tag{2.11}$$

A *hamiltoniana* $\mathcal{H}$ é então definida por:

$$\mathcal{H}(q,p) = \sum_k \dot{q}_k p_k - \mathcal{L}(q,\dot{q}) \tag{2.12}$$

sendo que a somatória se extende por todos os graus de liberdade. As equações de movimento ficam:

$$\dot{q}_k = \frac{\partial \mathcal{H}}{\partial p_k} \tag{2.13}$$

$$\dot{p}_k = -\frac{\partial \mathcal{H}}{\partial q_k} \tag{2.14}$$

A hamiltoniana definida pela equação (2.12) normalmente se reduz à energia total do sistema:

$$\mathcal{H}(q,p) = \mathcal{K}(p) + \mathcal{V}(q)$$

Para sistemas sem restrição e utilizando coordenadas cartesianas as equações ficam:

$$\dot{r}_i = p_i/m \tag{2.15}$$

$$\dot{p}_i = f_i \tag{2.16}$$

Gostariamos de salientar dois aspectos da formulação das equações de movimento de Hamilton:

- São equações de *primeira ordem* no tempo, ao contrário das equações de Newton e Lagrange, que são de segunda ordem

- Os papéis desempenhados pelas coordenadas q e momentos p são simétricos, e independentes. Isto facilita generalizações que serão apresentadas adiante, pois a derivada de q não precisa ser necessariamente igual a p/m.

Finalmente, citemos a formulação das equações de movimento de acordo com o princípio de Gauss [Hoo86]. O princípio de Gauss é utilizado quando desejamos a introdução de restrições ao movimento das partículas. Como vimos, as equações de Lagrange, e portanto também as de Hamilton, podem lidar com restrições holonômicas aos movimentos das partículas. Quando queremos lidar com restrições *não-holonômicas*, podemos aplicar o princípio de Gauss, que consiste em estabelecer que as forças f_c necessárias para garantir as restrições indicadas devem ser mínimas no sentido de *mínimos quadrados*. Assim, alteramos as equações de Newton:

$$m_i\ddot{\mathbf{r}}_i = \mathbf{f}_i + \mathbf{f}_{ci} \tag{2.17}$$

e exigimos que a soma:

$$\sum \frac{\mathbf{f}_{ci}{}^2}{2m_i}$$

seja mínima.

2.5 Simulações dos diferentes ensembles

Como veremos adiante, as simulações de dinâmica molecular lidam, em geral, com sistemas isolados, sem interação com o ambiente. As equações de movimento de Newton (2.7), apresentadas acima, são capazes de gerar, para um sistema isolado, apenas o ensemble microcanônico. Isto fica patente pelo fato de que a energia total é conservada pelas equações de Newton, e de que o número de partículas N e o volume V são mantidos constantes durante a simulação.

Se desejarmos realizar simulações sob condições de temperatura constante, pressão constante, etc., devemos introduzir esquemas artificiais nas equações de movimento, de forma a garantir as conservações desejadas. Veremos alguns métodos empregados com esse fim. Apresentaremos apenas métodos para temperatura constante, e forneceremos indicações da literatura sobre métodos para pressão constante.

Re-escalamento de velocidades Neste método, as velocidades são re-escaladas após cada movimento por um fator de $(T/\mathcal{T})^{1/2}$, onde T é a temperatura desejada, e $\mathcal{T}$ é a temperatura instantânea, calculada de acordo com a equação (2.29). Esse método é utilizado freqüentemente durante a termalização (ver página 25). No entanto, seu uso para dinâmica molecular isotérmica não é adequado devido às constantes e maciças intervenções no equilíbrio do sistema [AT87].

O método similar para pressão constante seria o contínuo re-escalamento do volume ocupado pelo sistema (e consequentemente das posições das partículas) de modo a garantir a constância do valor da pressão.

Método estocástico Este método pretende simular as interações entre as partículas do sistema e o ambiente. A certos intervalos, é introduzida uma "colisão" com o reservatório, através da seleção da velocidade de uma molécula aleatoriamente de acordo com a distribuição de Maxwell-Boltzmann na temperatura desejada [And80]. A densidade de probabilidade no limite para um grande número de passos de simulação é a do ensemble canônico [And80].

O problema com este método é na escolha do intervalo médio entre colisões. Se esse intervalo for alto, as flutuações de energia serão suaves, mas a dinâmica da temperatura será dominada pelas características próprias das equações de movimento, resultando em uma amostragem que não difere significativamente daquela realizada por essas equações, que resultam no ensemble microcanônico. Por outro lado, se tornamos as colisões mais freqüentes, a dinâmica da temperatura e do sistema será dominada pelas colisões, e não pelas interações entre as partículas [AT87].

Para atingir pressão constante, o método consistiria em realizar mudanças estocásticas no volume do sistema a certos intervalos. Não temos conhecimento de nenhuma simulação que utilize esse método.

Método da restrição Um método de dinâmica molecular isotérmica baseado nas equações de movimento de Gauss (2.17) é apresentado em [HLM82], e resulta nas equações de movimento:

$$\dot{\mathbf{r}}_i = \mathbf{p}_i/m_i \tag{2.18}$$

$$\dot{\mathbf{p}}_i = \mathbf{f}_i - \xi(\mathbf{r},\mathbf{p})\mathbf{p} \tag{2.19}$$

$\xi(\mathbf{r},\mathbf{p})$ representa o papel de um "coeficiente de fricção", que é calculado de modo a manter o valor da temperatura instantânea $\mathcal{T}$ num valor constante. O valor de ξ de acordo com o princípio de Gauss será:

$$\xi = \frac{\sum_i \mathbf{p}_i \cdot \mathbf{f}_i}{\sum_i |\mathbf{p}_i|^2} \tag{2.20}$$

[EM83] mostram que estas equações amostram um ensemble $NV\mathcal{T}$-constante, cuja distribuição configuracional, isto é, das coordenadas das partículas, é identica às do ensemble canônico. A diferenças na distribuição de momentos são da ordem $\mathcal{O}(1/N)$, devido à equivalência entre ensembles.

Também em [EM83] encontramos uma aplicação do princípio de Gauss para dinâmica molecular com pressão constante.

Método de Sistema extendido Um outro método foi introduzido por Nosé [Nos84], onde as interações com um reservatório térmico são simuladas através da introdução de um outro grau de liberdade s, que representa o reservatório. Associada ao reservatório também existe uma "inércia térmica", Q, que controla a taxa de flutuação da temperatura. Associado ao grau de liberdade s devemos introduzir também um "momento" p_s. As velocidades das partículas serão relacionadas com as derivadas temporais e momentos por:

$$\mathbf{v}_i = s\dot{\mathbf{r}}_i = \mathbf{p}_i/m_i s \tag{2.21}$$

O potencial associado a s será:

$$\mathcal{V}_s = (f+1)k_B T \ln s \tag{2.22}$$

onde f é o número de graus de liberdade e T é a temperatura especificada. O termo de energia cinética é:

$$\mathcal{K}_s = \frac{1}{2}Q\dot{s}^2 = p_s^2/2Q \tag{2.23}$$

As equações de movimento para as partículas ficam:

$$\ddot{\mathbf{r}}_i = \mathbf{f}_i/ms^2 - 2\dot{s}\dot{\mathbf{r}}_i/s \tag{2.24}$$

e para o parâmetro s:

$$Q\ddot{s} = \sum_i m_i \dot{\mathbf{r}}_i^2 s - (f+1)k_B T/s \tag{2.25}$$

A hamiltoniana do sistema: $\mathcal{H}_s = \mathcal{K} + \mathcal{K}_s + \mathcal{V} + \mathcal{V}_s$ é conservada, o que é útil na verificação do algoritmo (pag. 26). Em [Nos84], é demonstrado que o sistema apresenta distribuição canônica nas variáveis $\mathbf{r}$ e $\mathbf{p}/s$.

O ajuste do valor do parâmetro Q deve ser cuidadoso. Um valor muito alto tende a levar o sistema de volta ao ensemble microcanônico, enquanto que um valor muito baixo gera fortes oscilações de energia, acarretando problemas na termalização.

Um método similar para pressão constante foi proposto em [And80], através da inclusão de uma variável externa V, que representa o volume. Associada a essa variável existe uma "massa" Q. A energia cinética dessa nova variável é dada por:

$$\mathcal{K}_V = \frac{1}{2}Q\dot{V}^2 \tag{2.26}$$

e a energia potencial:

$$\mathcal{V}_V = PV \tag{2.27}$$

Uma generalização desse método é o apresentado em [PR80], que possibilita uma variação do tamanho de cada um dos lados do espaço de simulação, sendo adequado para o estudo de mudanças estruturais em sólidos.

2.6 Levantamento de valores

Abordaremos brevemente o método utilizado para o levantamento de características macroscópicas ou estruturais do sistema à partir dos dados produzidos durante a simulação. Abordaremos também as correções que podem ser aplicadas para melhora dos valores encontrados. Não serão citados aqui os métodos de cálculo de quantidades que envolvem flutuações ou funções de correlação temporal (veja-se [AT87]).

2.6.1 Quantidades termodinâmicas

Com exceção da energia total, o problema principal para o levantamento de valores termodinâmicos é que eles correspondem a valores médios sobre intervalos grandes de tempo do ponto de vista microscópico. Assim, é necessária a introdução de definições de valores instantâneos, que possam ser calculados em termos do estado microscópico do sistema, e que sejam coerentes com a definição macroscópica das quantidades. Apresentaremos, a seguir, as fórmulas mais utilizadas (ver, por exemplo, [Abr86], [Hoo86] e [AT87]):

Energia total

A energia total é simplesmente a soma da energia cinética com a energia potencial:

$$E = \mathcal{K} + \mathcal{V} \tag{2.28}$$

Temperatura Instantânea

A temperatura instantânea é normalmente definida por:

$$\mathcal{T} = \frac{1}{DNk_B} \sum_{i=1}^{N} |\mathbf{p}_i|^2/m_i \tag{2.29}$$

onde D é o número de dimensões (2 ou 3), e k_B a constante de Boltzmann. Pode-se mostrar que a temperatura termodinâmica T é a média (temporal ou no espaço de estado) da temperatura instantânea $\mathcal{T}$ dentro de um erro da ordem $\mathcal{O}(1/N)$.

O termo DN utilizado acima corresponderia mais precisamente ao número de graus de liberdade do sistema. Assim, para sistemas com restrições, devemos alterar adequadamente a equação acima.

Pressão Instantânea

Para o cálculo da pressão, costuma ser utilizada a expressão da pressão instantânea $\mathcal{P}$ dada por:

$$\mathcal{P} = \rho k_B \mathcal{T} + \mathcal{W}/V \tag{2.30}$$

onde V 'e o volume, ρ 'e a densidade N/V e $\mathcal{W}$ é o "virial interno", dado por [AT87]:

$$\mathcal{W} = \frac{1}{3}\sum_i \sum_{j>i} \mathbf{r}_{ij} \cdot \mathbf{f}_{ij} \tag{2.31}$$

onde $\mathbf{r}_{ij}$ é a distância entre as partículas do par i e j, $\mathbf{f}_{ij}$ é a força entre as partículas desse par e as somatórias garantem que cada par seja considerado apenas uma vez.

A pressão P será então a média da pressão instantânea $\mathcal{P}$.

2.6.2 Quantidades estruturais

A quantidade estrutural mais comumente calculada em simulações de dinâmica molecular é a *função de distribuição radial*, $g(r)$, definida como fornecendo a probabilidade de encontrar um par de átomos distanciados de r, relativamente à probabilidade esperada para uma distribuição completamente aleatória.

Para o cálculo por computador, consideramos as partículas presentes em uma faixa pequena Δr de distância do valor r [Rah64]:

$$g(r) = \frac{V}{N}\frac{n(r)}{4\pi r^2 \Delta r} \tag{2.32}$$

onde $n(r)$ é o número de partículas situadas a uma distância entre r e $r + \Delta r$ de uma partícula dada.

O *fator de estrutura*, denotado por[6] $S(a)$, que reflete os componentes de Fourier das flutuações da densidade em líquidos, pode ser calculado através da transformada de Fourier, $G(a)$ da função de distribuição radial:

$$S(a) = 1 + \rho G(a) \tag{2.33}$$

A função de distribuição radial é relacionada com a *função de correlação de pares*, $h(r)$, por:

$$h(r) = g(r) - 1 \tag{2.34}$$

2.6.3 Correções de longo alcance

Uma das fontes de erros comuns em simulações de dinâmica molecular é a utilização de potenciais de curto alcance com um raio de corte. Muitas vezes é util introduzir correções para compensar a parte correspondente ao longo alcance. As contribuições para energia, pressão, etc, podem ser estimadas para $r > r_c$ usando a aproximação $g(r) \approx 1$ nessa região [HM86].

[6] a é o espaço recíproco das distancias em relação à transformada de Fourier, assim como a freqüencia é o espaço recíproco do tempo.

SERVIÇO DE BIBLIOTECA ... O - IFQSC

Para o potencial de Lennard-Jones isto nos dá:

$$E_{cla} = \frac{8}{9}\pi N \rho r_c^{-9} - \frac{8}{3}\pi N \rho r_c^{-3} \tag{2.35}$$

$$P_{cla} = \frac{32}{9}\pi N \rho^2 r_c^{-9} - \frac{16}{3}\pi N \rho^2 r_c^{-3} \tag{2.36}$$

onde E_{cla} é a correção de longo alcance para a energia total reduzida e P_{cla} é a correção de longo alcance para a pressão reduzida.

Próximo ao ponto triplo[7], as correções de longo alcance, principalmente na pressão, são extremamente importantes [HM86].

2.7 Condições de contorno

Devido ao grande número de partículas presentes em sistemas macroscópicos ($\mathcal{O}(10^{23})$), mesmo nos mais rápidos computadores atuais tem-se de realizar simulações com quantidades comparativamente muito pequenas de partículas ($\mathcal{O}(10^3)$). Assim, se tentarmos lidar com esse sistema em condições de contorno correspondentes a sistemas reais (por exemplo, considerando as partículas presas em um recipiente fechado, com ou sem interação térmica com o ambiente), toda a dinâmica do sistema será dominada pelas interações com o ambiente, a menos que lidemos com situações especiais, como por exemplo no estudo de uma pequena gota de líquido, onde as forças coesivas entre as moléculas são suficientes para manter o sistema durante o curso da simulação [AT87].

O modo mais utilizado de minimizar esse problema é estabelecer condições de contorno periódicas, onde o volume do espaço de simulação é repetido em todas as direções (fig. 2.2). Note que essas condições implicam que, quando uma partícula sai por uma das faces, uma imagem sua penetra pela face oposta. Vemos que isso leva a que o momento angular do sistema não seja conservado [Hoo86].

Quando se utilizam condições de contorno periódicas, então todas as distâncias calculadas no sistema devem obedecer à convenção de *imagem mínima*, na qual a distância entre as partículas i e j é calculada entre a partícula i e a imagem da partícula j mais próxima de i (ver fig. 2.3).

Fica claro pela fig. 2.2 que o sistema simulado consiste em um sistema periódico infinito, o que não é o caso de sistemas reais. Um efeito notório disso é que a função de distribuição de pares não é mais perfeitamente esférica. Um estudo teórico sobre os efeitos das condições de contorno periódicas é apresentado em [PH81].

Outras formas de condições de contorno podem ser úteis, de acordo com o tipo de simulação desejada. Veja-se por exemplos os levantamentos em [Abr86] e [AT87].

[7] O ponto triplo é o ponto de co-existência das fases gasosa, líquida e sólida.

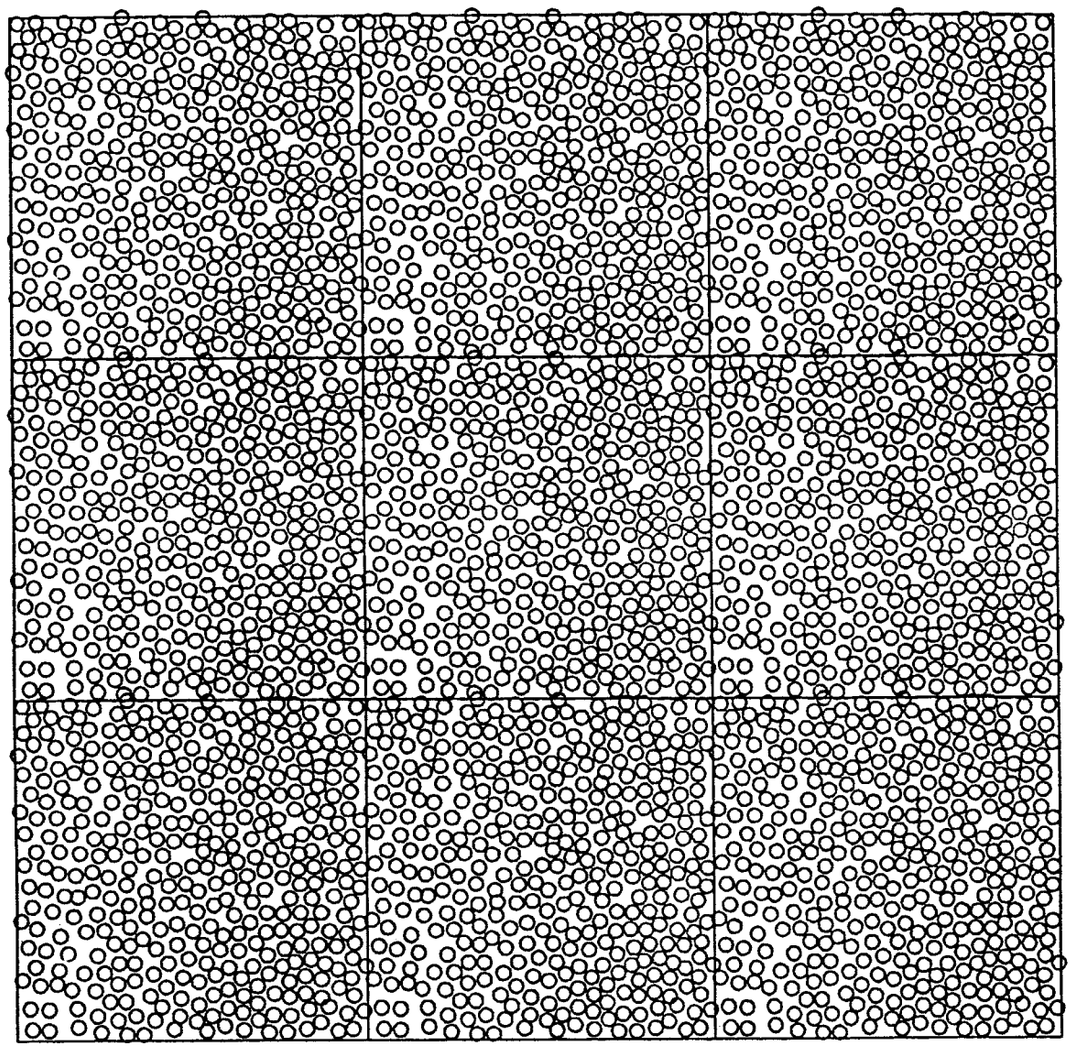

Figura 2.2: Repetição do espaço de simulação em condições de contorno periódicas

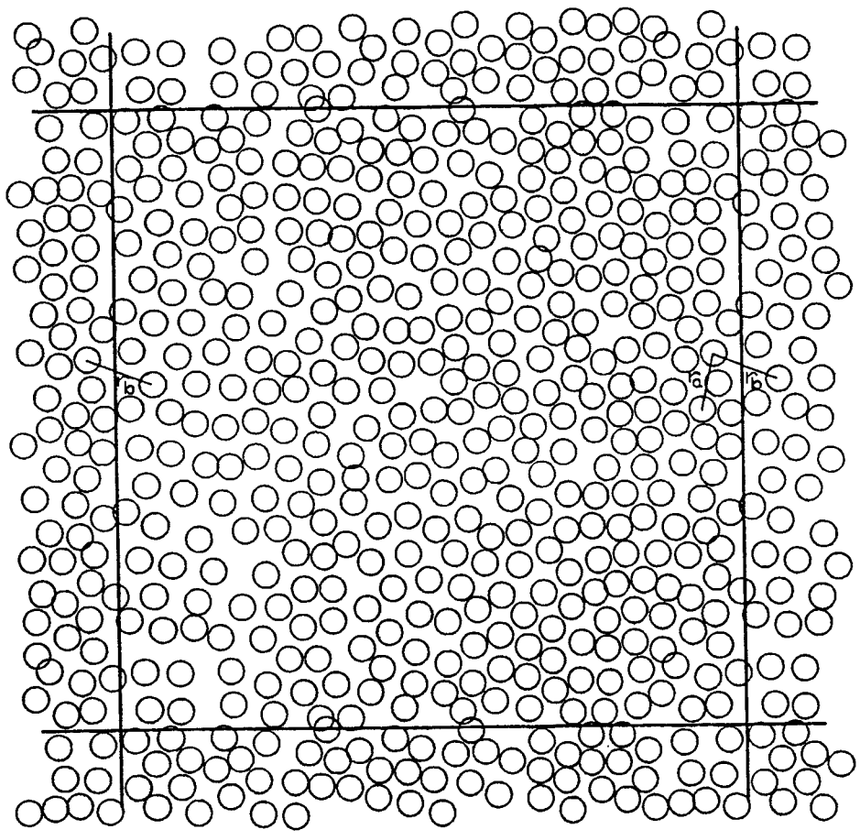

Figura 2.3: Convenção da imagem mínima sob condições periódicas: no caso r_a, a distância se calcula no espaço normal, enquanto que no caso r_b a distância utiliza uma imagem de uma das partículas

2.8 Considerações adicionais

Para finalizar nossa introdução às simulações de dinâmica molecular, enfocaremos alguns aspectos do processo de simulação.

2.8.1 Configuração inicial

A integração das equações de movimento implica no fornecimento de um conjunto de condições iniciais para as equações diferencias. Normalmente isso é fornecido como posições e velocidades iniciais das partículas, ou então como posições em dois instantes consecutivos.

Pode-se mostrar que, a menos para condições muito especiais (ver seção 2.1) as condições iniciais fornecidas não são importantes [LL58], desde que se simule uma extensão de tempo suficiente para o sistema amostrar uma região significativa do espaço de estado.

Normalmente, as partículas são distribuídas inicialmente numa distribuição cúbica de face centrada, com as velocidades escolhidas aleatoriamente dentro de uma distribuição gaussiana. A distribuição de Maxwell-Boltzmann é atingida após um período de termalização.

2.8.2 Termalização

Apesar de, como citado acima, as condições iniciais do sistema não serem importantes, ainda assim durante um certo intervalo de tempo imediatamente após o inicio da simulação as condições do sistema serão dominadas pelas condições iniciais, isto é, haverá uma forte correlação entre o estado do sistema e o estado inicial.

Desta forma, é necessário permitir-se um período de *termalização*, durante o qual não serão realizados levantamentos nos valores do sistema. Esse período é relacionado com o *tempo de relaxação* do sistema [LL58].

A determinação do tempo de duração do período de termalização não é simples. Sabe-se [AT87] que as quantidades termodinâmicas atingem os valores de equilíbrio antes das quantidades estruturais. Assim, um método de determinar o período de termalização pode ser através da monitoração destas últimas. Em geral, escolhe-se *a priori* um intervalo que seja suficientemente grande para se ter certeza de que o sistema entrou em equilíbrio, e realiza-se a simulação por esse intervalo sem realização de nenhum levantamento.

Um método que pode facilitar a termalização é o re-escalamento de velocidades durante a fase inicial desse período, de forma a manter uma alta temperatura (por exemplo, $T = 5$, para um sistema de Lennard-Jones). Isto facilita os processo de desmanchamento das configurações iniciais das particulas. Após uma tal fase com temperatura elevada, as velocidades são re-escaladas uma última vez de acordo com

a temperatura desejada para a simulação, e então é deixado um período menor de termalização para a estabilização da distribuição de velocidades.

2.8.3 Conservação da energia

Após o desenvolvimento do algoritmo de simulação, é útil realizar-se alguns testes para verificar a sua validade. O teste mais comumente utilizado é a verificação da conservação da energia total.

Numa simulação microcanônica, a energia total deve ser conservada, e em alguns outros esquemas de simulação (como o de Nosé), dispomos de uma hamiltoniana que deve ter o valor conservado. Obviamente, o algoritmo utilizado para a integração numérica apresenta problemas de aproximação, conforme veremos no capítulo seguinte, que fazem com que o valor da energia não seja completamente conservado. No entanto, ela deve apenas flutuar em torno de um valor médio. Também a introdução de um raio de corte faz com que haja uma mudança descontínua no valor da energia potencial quando duas partículas se afastam mais do que r_c. Esse problema pode ser contornado fazendo o cálculo de um *potencial sem descontinuidade*, que consiste no mesmo valor do potencial mas subtraído do valor de potencial em r_c. Esse potencial sem descontinuidade, que pode ser facilmente calculado à partir do potencial normal se sabemos o número de pares interagentes em um dado instante [AT87], pode então ser utilizado para o teste da conservação de energia.

O teste da conservação da energia total não é infalível, como demonstrado em [Gre90], mas é ainda assim o mais utilizado devido à facilidade de implementação e grande sensibilidade a diversos tipos de erros no algoritmo.

2.9 Conclusão

Vimos os pontos principais a serem considerados numa simulação de dinâmica molecular, quais sejam: o tipo de partículas utilizadas, as equações de movimento, as condições de contorno e a forma de cálculo de alguns valores e características do sistema. A partir deste ponto, nos restringiremos ao estudo de partículas de Lennard-Jones que se movem de acordo com as equações de Newton, sujeitas a condições de contorno periódicas.

No próximo capítulo, estudaremos os fatores ligados à implementação numérica e ao algoritmo de integração das equações de movimento escolhido para as simulações neste trabalho.

Capítulo 3

Discretização e representação de dados

Considerando o problema de interesse estabelecido no capítulo anterior, procederemos agora a uma análise de sua discretização, representação dos dados e forma de cálculos para a implementação em computador.

3.1 Integração das equações de movimento

Como vimos no capítulo anterior, as equações de movimento constituem-se em um sistema de equações diferenciais. Para a simulação por computador, precisamos, portanto, estabelecer uma forma de integração numérica dessas equações. Dentre os diversos métodos descritos na literatura (veja por exemplo [DB74]) comentaremos aqui somente três: o de Runge-Kutta, por ser muito difundido fora do campo de dinâmica molecular, e os métodos preditores-corretores de Gear e de diferença central de Störmer, por serem os mais utilizados em dinâmica molecular.

Durante a discussão, simplificaremos a notação do sistema de equações diferenciais a serem resolvidas, indicando-as nas seguintes formas vetoriais. Para as equações de primeira ordem (por exemplo as de Hamilton, eqs. (2.13) e (2.14)):

$$\frac{d\mathbf{y}}{dt} = \mathbf{f}(\mathbf{y}) \tag{3.1}$$

onde $\mathbf{y}$ representa um vetor com coordenadas e momentos das partículas: $\mathbf{y} = (q_1, q_2, ..., q_N, p_1, p_2, ..., p_N)^T$ e $\mathbf{f}$ uma função desse vetor, que corresponde às velocidades e às forças. Para equações de segunda ordem, como as de Newton (eq. (2.7)):

$$\frac{d^2\mathbf{y}}{dt^2} = \mathbf{f}(\mathbf{y}) \tag{3.2}$$

Naturalmente, o sistema de equações diferenciais será resolvido recorrendo-se a aproximações por intermédio de equações de diferença. Assim, o tempo será discretizado e representado por:

$$t_n = n\Delta t \tag{3.3}$$

onde t_n é o tempo discreto e Δt é o intervalo de discretização. Também, os sucessivos valores das variáveis serão representados por $\mathbf{y}_n$, indicando uma aproximação para $\mathbf{y}(t_n)$.

Comentaremos para os diversos métodos os seus *erros globais*, que consistem nos erros na avaliação de $\mathbf{y}$ durante o desenrolar de uma simulação completa, isto é, para dado intervalo de tempo simulado. Este valor é chamado *erro global* em contraposição ao *erro local*, que é o erro envolvido no cálculo de um passo do valor de $\mathbf{y}$ devido à aproximação da equação diferencial por uma equação de diferenças.

3.1.1 Método de Runge-Kutta

O método de Runge-Kutta consiste em calcular o valor de $\mathbf{f}(\mathbf{y})$ em pontos próximos à curva de solução, e combiná-los de forma apropriada.

Uma forma possível é apresentada nas equações abaixo:

$$\begin{aligned}
\mathbf{k}_1 &= \Delta t \mathbf{f}(\mathbf{y}_n) \\
\mathbf{k}_2 &= \Delta t \mathbf{f}(\mathbf{y}_n + \frac{1}{2}\mathbf{k}_1) \\
\mathbf{k}_3 &= \Delta t \mathbf{f}(\mathbf{y}_n + \frac{1}{2}\mathbf{k}_2) \\
\mathbf{k}_4 &= \Delta t \mathbf{f}(\mathbf{y}_n + \mathbf{k}_3) \\
y_{n+1} &= y_n + \frac{1}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)
\end{aligned} \tag{3.4}$$

O erro global de aproximação deste método é $\mathcal{O}(\Delta t^4)$.

3.1.2 Método preditor-corretor de Gear

Os métodos do tipo *preditor-corretor* se baseiam no seguinte processo:

1. É realizada uma *previsão* do valor de $\mathbf{y}_{n+1}$, levando-se em consideração o comportamento do sistema em instantes anteriores
2. Com base no valor previsto, é calculada a função $\mathbf{f}$
3. O valor de $\mathbf{f}$ é utilizado para realizar uma *correção* do valor previsto
4. Os dois passos anteriores podem ser repetidos iterativamente, usando de cada vez o último valor calculado como valor previsto, até que se consiga uma convergência, isto é, até que a correção necessária seja suficientemente pequena

Os diversos métodos preditores-corretores se distinguem tanto pela forma de realizar a predição como a correção. Comentaremos aqui a formulação de Nordsieck para o método de Gear [AT87] de ordem 4, aplicado a uma equação diferencial de

segundo grau (eq. 3.2). Neste caso, as informações sobre o comportamento anterior do sistema são representadas pelo armazenamento de $\mathbf{y}$ e suas derivadas primeira, segunda e terceira, que representaremos respectivamente por $\mathbf{y}'$, $\mathbf{y}''$ e $\mathbf{y}'''$. A função $\mathbf{f}$ é relacionada com a "aceleração" ($\mathbf{y}''$).

O valor previsto será:

$$\begin{aligned}
\mathbf{y}_p &= \mathbf{y}_n + \Delta t \mathbf{y}'_n + \frac{\Delta t^2}{2}\mathbf{y}''_n + \frac{\Delta t^3}{6}\mathbf{y}'''_n \\
\mathbf{y}'_p &= \mathbf{y}'_n + \Delta t \mathbf{y}''_n + \frac{\Delta t^2}{2}\mathbf{y}'''_n \\
\mathbf{y}''_p &= \mathbf{y}''_n + \Delta t \mathbf{y}'''_n \qquad (3.5)\\
\mathbf{y}'''_p &= \mathbf{y}'''_n \qquad (3.6)
\end{aligned}$$

e a correção aplicada:

$$\begin{aligned}
\mathbf{y}_{n+1} &= \mathbf{y}_p + \frac{\Delta t^2}{12}\Delta \mathbf{y}'' \\
\mathbf{y}'_{n+1} &= \mathbf{y}'_p + \frac{5\Delta t}{12}\Delta \mathbf{y}'' \\
\mathbf{y}''_{n+1} &= \mathbf{y}''_p + \Delta \mathbf{y}'' \\
\mathbf{y}'''_{n+1} &= \mathbf{y}'''_p + \frac{1}{\Delta t}\Delta'' \mathbf{y} \qquad (3.7)
\end{aligned}$$

sendo que denotamos por $\Delta\mathbf{y}''$ a diferença entre o valor previsto $\mathbf{y}''_p$ e o valor calculado à partir de $\mathbf{f}(\mathbf{y}_p)$.

As fórmulas acima de preditores e corretores apresentam precisão suficiente para que um único passo de correção seja suficiente para a maioria dos casos. De fato, a utilização de mais de um passo de correção não acrescenta melhora de precisão suficiente para justificar o aumento do número de cálculos de $\mathbf{f}$, sendo mais justificada uma diminuição no valor de Δt [VH87]. Também, o aumento do grau do polinômio de predição, eqs. 3.5, não compensa para dinâmica molecular [AT87].

O erro global de um preditor-corretor como acima é $\mathcal{O}(\Delta t^4)$.

3.1.3 Método de diferença central de Störmer

O método de diferença central de Störmer é utilizado para a resolução numérica de equações diferenciais de *segunda ordem*, da forma (3.2). Consiste em aproximar a derivada $\mathbf{y}'(t_n)$ por:

$$\mathbf{y}'(t_n) \approx \frac{y_{n+1} - y_n}{2\Delta t} \qquad (3.8)$$

A equação de diferenças para o sistema fica:

$$\mathbf{y}_{n+1} = 2\mathbf{y}_n - \mathbf{y}_{n-1} + \Delta t^2 \mathbf{f}(\mathbf{y}_n) \tag{3.9}$$

Este método foi introduzido em simulações de dinâmica molecular por Verlet [Ver67], e desde então tem sido muito utilizado, em diversas variantes.

Uma das variantes, conhecida como *leap-frog*, procura contornar um problema numérico que pode ser reconhecido quando verificamos que na eq. (3.9) temos a soma de quantidades $\mathcal{O}(\Delta t^0)$ com uma quantidade $\mathcal{O}(\Delta t^2)$, o que pode resultar em imprecisões numéricas quando o valor de Δt é pequeno. No esquema *leap-frog* isto é contornado pelo uso das seguintes equações:

$$\begin{aligned} \mathbf{y}'_{n+1/2} &= \mathbf{y}'_{n-1/2} + \Delta t \mathbf{f}(\mathbf{y}_n) \\ \mathbf{y}_{n+1} &= \mathbf{y}_n + \Delta t \mathbf{y}'_{n+1/2} \end{aligned} \tag{3.10}$$

Neste caso, a soma de valores sempre envolve quantidade $\mathcal{O}(\Delta t^0)$ com quantidades $\mathcal{O}(\Delta t)$.

No entanto, as duas variantes acima ainda apresentam um problema: a forma de gerenciamento da velocidade $\mathbf{y}'$. No primeiro caso, as velocidade não são introduzidas, a pesar de poderem ser calculadas de forma indireta por (3.8). No segundo caso, as velocidades são introduzidas, mas não estão em fase com os valores das posições e de $\mathbf{f}$. Uma variante, chamada *método da velocidade*, que elimina as duas dificuldades simultaneamente foi proposta em [SAHW82]:

$$\begin{aligned} \mathbf{y}'_{n+1/2} &= \mathbf{y}'_n + \frac{\Delta t}{2}\mathbf{f}(\mathbf{y}_n) \\ \mathbf{y}_{n+1} &= \mathbf{y}_n + \Delta t \mathbf{y}'_{n+1/2} \\ \mathbf{y}'_{n+1} &= \mathbf{y}'_{n+1/2} + \frac{\Delta t}{2}\mathbf{f}(\mathbf{y}_{n+1}) \end{aligned} \tag{3.11}$$

Todas essas variantes são numericamente equivalentes, a menos de erros de truncamento e arredondamento, apresentando um erro global $\mathcal{O}(\Delta t^2)$.

3.1.4 Comparação dos métodos

Não existe, em princípio, nenhuma vantagem completa de um dos métodos sobre todos os outros, o que fica demonstrado pelo fato de que todos esses métodos, bem como ainda outros, têm sido utilizados com sucesso em simulações de dinâmica molecular por diversos pesquisadores. No entanto, vamos apresentar agora alguns fatores que podem levar à escolha entre os métodos apresentados, bem como indicar as razões do método escolhido para este trabalho.

Os critérios normalmente empregados na escolha de um método de integração das equações de movimento são as seguintes:

Precisão A ordem de grandeza dos erros introduzidos na aproximação das equações

Espaço Espaço de memória necessário para armazenar os valores utilizados

Facilidade de implementação que redunda em simplificação do código e diminuição da carga computacional

Fidelidade ou ausência de introdução de características não existentes no sistema físico sob estudo

A precisão, ou dependência do método com o valor de Δt é extremamente importante, pois, se o método permite valores grandes de Δt, o número de passos para a simulação de um dado intervalo de tempo pode ser diminuido, mantendo-se a mesma precisão nos resultados. Isto está relacionado com o erro global do método, que foi indicado acima para cada um deles. Como vimos, o que apresenta maior erro global é o de Störmer. Isto não ocorre, no entanto, para todos os valores de Δt. Quando o Δt utilizado é acima de certo valor, o método de Störmer passa a ser vantajoso, conforme indicado em [AT87] e [VH87], que apresentam avaliações da precisão de alguns métodos.

Com relação a espaço, o mais econômico deles é o de Störmer, que exige o armazenamento simultâneo de valores de $\mathbf{y}$, $\mathbf{y}'$ e $\mathbf{f}$. Já os de Runge-Kutta e Gear de ordem 4 exigem o armazenamento simultâneo de 5 conjuntos de dados: $\mathbf{y}$, $\mathbf{y}'$, $\mathbf{y}''$, $\mathbf{y}'''$ e $\mathbf{f}$ para o método de Gear e $\mathbf{y}$, $\mathbf{k}_1$, $\mathbf{k}_2$, $\mathbf{k}_3$, $\mathbf{k}_4$, para o de Runge-Kutta.

O método que apresenta implementações e cálculos mais simples é sem dúvida o de Störmer. O de Gear apresenta uma implementação um pouco mais complexa, e introduz o cálculo de uma quantidade maior de variáveis. Já o de Runge-Kutta é o mais complexo dos três, além de implicar uma quadruplicação do cálculos das forças, que representa a parte de maior demanda computacional em dinâmica molecular. Deixemos claro no entanto que os cálculos adicionais do algoritmo de Gear em relação ao de Störmer não implicam em um aumento significativo no tempo de processamento, justamente devido ao fato de que a maior parte desse tempo será dispendida no cálculo das forças, em qualquer dos casos.

Com relação à introdução de características não presentes nos processos físicos sob simulação, isto é algo cujo estudo ainda não se encontra completo. Notemos apenas alguns fatores: o método de Störmer é reversível no tempo, o mesmo não acontecendo com o método de Runge-Kutta, ou mesmo com o de Gear quando realizado um único passo de correção [VH87]. Ainda, foram encontradas anomalias na simulação de um sistema simples (pêndulo físico) com um algoritmo preditor-corretor (diferente do apresentado), que não surgiram com a utilização do algoritmo de Störmer [Mil91].

Com relação às diversas variantes do método de Störmer, elas são, como foi dito acima, numericamente equivalentes, de forma que a escolha entre uma delas deve ser

determinada por outros fatores. A forma mais atrativa parece ser o método da velocidade, pela forma como gerencia simultaneamente o problema do arredondamento e do cálculo da velocidade, o que o torna mais adequado para a implementação dos diversos esquemas de simulação dos ensembles não-microcanônicos [And83].

Neste trabalho, onde o enfoque principal é no estudo dos fatores relativos a paralelização, a escolha de um ou outro método de integração é indiferente. Optamos então pela implementação da variante da velocidade do método de Störmer, por ser de implementação extremamente simples e bastante versátil. Além disso, a forma como os cálculos são realizados nessa variante é semelhante à que ocorre nos métodos preditores-corretores, podendo inclusive ser considerado como um método preditor-corretor simples, onde a correção é aplicada apenas sobre as velocidades [AT87]. Isto facilita a transição do programa para um método preditor-corretor mais preciso caso haja necessidade.

3.2 Representação dos dados e cálculo de valores

Analisaremos agora alguns aspectos relativos à representação dos dados no computador e considerações afins.

Em primeiro lugar, vamos discutir o número de bits utilizado na representação dos dados. Para a implementação deste trabalho, escolhemos uma representação de 32 bits para posições, velocidades e acelerações, e 64 bits para energias (cinética e potencial). Apresentaremos as razões que nos levaram a essa escolha.

A representação padrão IEEE-754 para números de ponto flutuante de 32 bits garante um máximo de 7 dígitos decimais corretos em qualquer operação aritmética. Esta precisão é insuficiente para o cálculo das energias em sistemas com muitas partículas. Assim, por exemplo, para o cálculo da energia cinética, num sistema com $N = 10^5$ teremos apenas dois dígitos significativos do valor da energia cinética de cada partícula considerados na soma total. O caso é ainda pior para o cálculo da energia potencial, onde precisamos considerar o potencial de cada *par* de partículas interagentes. Por exemplo, numa densidade de $\rho = 0.8$, o número de pares interagentes (para o valor de N acima) será aproximadamente $2.5 \cdot 10^6$ (caso tridimensional), ocasionando sérios problemas no cálculo. Assim, decidimo-nos pela utilização de uma precisão de 64 bits para essas variáveis.

A escolha de uma precisão de 64 bits para variáveis escalares como a energia não representa grande demanda de memória. No entanto, o mesmo já não ocorre para variáveis vetoriais, como as posições, velocidades e acelerações. O armazenamento dessas variáveis é responsável por praticamente toda a utilização de memória de um programa de dinâmica molecular. Assim, a utilização de uma precisão de 64 bits implica na possibilidade de armazenamento da metade do número de partículas com relação à utilização de uma representação de 32 bits, para uma mesma quantidade

de memória. Esta é uma das principais razões que nos levaram à escolha de uma precisão de 32 bits para essas variáveis. A outra é o fato de que a grande maioria dos cálculos realizados são operações sobre os valores dessas variáveis, e portanto a utilização de 32 bits representa grande economia de tempo de computação em relação a 64 bits. Devemos, é claro, justificar essa escolha mostrando que não implica em erros inaceitáveis na simulação.

Com relação às posições, raramente se justifica a utilização de uma precisão maior do que 32 bits, visto que o caracter caótico do sistema implica em que as trajetórias calculadas para as partículas irão apresentar divergências exponencialmente crescentes com o tempo [Hoo86], de forma a rapidamente eliminar a vantagem de algumas casas a mais de precisão. Somente é necessário manter-se a precisão nos valores da posição devido à necessidade de cálculo de fatores de correlação temporal, e portanto, soluções aproximadamente exatas são necessárias apenas por tempos comparáveis com os tempos de correlação de interesse. Para o cálculo de médias estatísticas, basta que o sistema faça uma amostragem correta do ensembles correspondente (em geral o microcanônico) [AT87]. Obviamente, maiores erros nos cálculos das trajetórias são introduzidos pelo método de integração utilizado do que pelo arredondamento que ocorre na aproximação de 32 bits.

Já com relação às velocidades e acelerações, deve ser tomado mais cuidado. Isto decorre do fato de que os seus valores aparecem nas equações de integração multiplicados por fatores que envolvem potências de Δt. Desta forma, se a precisão utilizada no cálculo dessa equação for baixa, e o valor de Δt muito pequeno, então a influência da aceleração pode ser desprezada no arredondamento. No entanto ainda aqui notamos que, de qualquer forma, diversas casas decimais do valor da aceleração e da velocidade serão perdidos no arredondamento desse cálculo devido à diferença de escala determinada pelos fatores de Δt.

Para ilustrar os pontos indicados, veja a tabela 3.1, onde são apresentados os resultados da flutuação média quadrática da energia para uma mesma simulação, em 32 e 64 bits. Veja que os valores são praticamente idênticos. De fato, as diferenças ocorreram no máximo na oitava casa decimal[1].

3.3 Implementação das condições de contorno periódicas

Normalmente, para a implementação de condições de contorno periódicas, o processo utilizado é o seguinte:

1. As coordenadas de todas as partículas são normalizadas de forma que a "caixa" de simulação (o espaço disponível para as partículas) tenha suas dimensões

[1] Em ambas as simulações, foram utilizadas precisões de 64 bits para o armazenamento dos totalizadores, conforme discutido acima. Também os valores de posições foram armazenados numa representação inteira, conforme será discutido no próximo ítem.

Tabela 3.1: Levantamento da flutuação média quadrática da energia para simulações com 32 e 64 bits. O ponto de simulação corresponde a $\rho = 0.8442$ e $T = 0.524$, com 2048 partículas. A simulção envolveu um tempo total $t = 1.0$.

Δt	N_p	32 bits	64 bits
0.03125	32	0.0057136688	0.0057136758
0.015625	64	0.0011171688	0.0011171694
0.0078125	128	0.0002785567	0.0002785585
0.00390625	256	0.0000699587	0.0000699578
0.001953125	512	0.0000178181	0.0000178170
0.0009765625	1024	0.0000043881	0.0000043839
0.00048828125	2048	0.0000010991	0.0000010944
0.000244140625	4096	0.0000002649	0.0000002546

unitárias. Isto é feito distribuindo as coordenadas no intervalo $[-1/2, 1/2)$, ou, mais frequentemente, $[0, 1)$[2]. Escolheremos aqui este último caso.

2. Após o cálculo da nova posição das partículas, o resultado deve ser novamente normalizado, o que é feito pelo cálculo:

$$\tilde{x} = x - \lfloor x \rfloor$$

onde $\lfloor x \rfloor$ indica o maior inteiro menor que x.

No caso da linguagem **occam** utilizada neste trabalho, existe um método mais eficiente e preciso de lidar com as condições de contorno periódicas que consiste no seguinte:

1. Normalizar as coordenadas de forma que fiquem distribuídas entre -2^{31} e $2^{31}-1$.

2. Representar os valores das coordenadas normalizadas pelos valores *inteiros* de 32 bits correspondentes

3. Ao realizar o cálculo das novas posições, utilizar as operações `PLUS` e `MINUS`, que realizam respectivamente a soma e a subtração de forma circular sobre 32 bits, isto é, temos $(2^{31} - 1)$ `PLUS` $1 = -2^{31}$.

Desta forma, não é necessária a contínua renormalização das coordenadas, que já se apresentam normalizadas após os cálculos, evitando assim as conversões para

[2] A notação $[0, 1)$ indica que o 0 é incluído no intervalo, enquanto o 1 é excluído, de acordo com convenção padrão na matemática

Tabela 3.2: Comparação de armazenamento de posições em inteiro ou ponto flutuante (real). $\rho = 0.8442$, $\Delta t = 0.001$, 1000 passos de simulação, $N = 2048$

	Inteira	Real
Tempo de processamento (min)	17.1	18.8
Flutuação da energia	$4.49 \cdot 10^{-6}$	$4.60 \cdot 10^{-6}$

inteiro e operações extras. Outra vantagem é que, como nas posições o expoente não é significativo, sendo relevante apenas o valor da mantissa do número de ponto flutuante, então este método representa um aumento de precisão na representação das posições, pelo aproveitamento das posi,c oes normalmente ocupadas pelo expoente para a representa,c ao da mantissa.

Para demonstrar esse pontos, realizamos duas simulações idênticas em todos os aspectos, a não ser que uma utilizou a representação inteira, e outra a representação em ponto flutuante tradicional. Os resultados estão na tabela 3.2. Note-se que apesar da vantagem representada pela utilização de inteiros, ela não é tão significativa, pois a diferença de tempos entre os cálculos inteiros e reais no *transputer* utilizado neste trabalho não é grande.

3.4 Considerações adicionais

Uma forma comumente utilizada de agilizar o cálculo em simulações de dinâmica molecular é o uso de uma tabela de *look-up* para o cálculo do potencial interpartículas e a força.

É construída uma tabela que dá o valor do potencial em função do quadrado da distância entre as partículas. O espaçamento da tabela, que corresponde à diferença entre os valores da distância quadrática para duas entradas consecutivas na tabela é em geral fixo, mas pode também ser feito variável, de acordo com o método de interpolação utilizado. O valor desse espaçamento deve ser determinado de acordo com o erro aceitável para a simulação em vista, e pode ser calculado através de expressões de erro máximo, como apresentadas por exemplo em [DB74], ou através da realização de algumas simulações de teste.

Para o cálculo da força, pode também ser construída uma tabela adicional, mas em geral é mais eficiente o uso da mesma tabela. e a diferenciação da fórmula de interpolação para o potencial, como sugerido em [AT87].

O uso de uma tabela de *look-up* é particularmente útil quando o potencial é descrito por uma função complexa, por exemplo, envolvendo exponenciais, como o potencial de Stillinger-Weber. No caso do potencial de Lennard-Jones, o seu uso resulta inefi-

Tabela 3.3: Comparação de precisão e tempo de processamento entre tabela de *look-up* e cálculo direto para o potencial de Lennard-Jones.

	Calculado	*Look-up*
Energia	-1.76196	-1.76194
Desvio da energia	0.00028	0.00027
Tempo (s)	1.01	1.21

ciente, devido à simplicidade de cálculo desse potencial. Para demonstrar esse ponto, realizamos uma simulação utilizando interpolação quadrática em uma tabela de *look-up*, e cálculo das forças por derivação. A separação entre entradas da tabela foi de $0.01\sigma^2$. A tabela 3.3 mostra os valores de energia média, desvio padrão da energia e tempo de processamento médio por passo de simulação, tanto para uma simulação com cálculos diretos como por tabela de *look-up*. Note-se que os valores de energia e desvio-padrão são praticamente idênticos, o que indica que o uso da tabela não afetou adversamente o desempenho do sistema. No entanto, o tempo de processamento cresce aproximadamente 20% com o uso da tabela.

3.5 Conclusões

Estabelecemos neste capítulo a forma de aproximação numérica a ser utilizada neste trabalho e a representação dos dados. Façamos um resumo dos pontos principais:

- Para a solução das equações diferenciais será utilizada a variante da velocidade do algoritmo de Störmer, devido a combinar adequadamente fatores de simplicidade, versatilidade e precisão
- A representação de posições, velocidade e acelerações utilizará 32 bits, devido à pouca vantagem na inclusão de uma representação de 64 bits, que não compensa o grande aumento de memória requerido. Deixamos uma representação de 64 bits apenas para os totalizadores de energias e virial.
- As posições serão armazenadas em inteiro, ao invés de ponto flutuante, o que representa uma melhora de precisão, além de simplificar o processo de implementação de condições de contorno periódicas em `occam`
- Não será utilizada uma tabela de *look-up* para o cálculo de potencial e força

Com esses fatos estabelecidos, passaremos no próximo capítulo ao estudo do algoritmo a ser utilizado e a forma de paralelização.

Capítulo 4

Paralelização

Nos capítulos anteriores procedemos à delimitação do problema e à escolha dos algoritmos e representações numéricas da implementação. Este capítulo trata do algoritmo utilizado e da paralelização para a implementação em uma rede de computadores que trabalha sob o paradigma de passagem de mensagens, como é o caso da rede de *transputers* escolhida.

4.1 Algoritmos para a otimização do cálculo de forças

Em um trabalho anterior [Tra89], discutimos e analisamos diversos algoritmos para dinâmica molecular. Apresentaremos aqui um breve resumo dos principais pontos dessas discussões, de forma a basear a escolha do algoritmo para este trabalho.

Primeiramente, deve-se notar que o ponto crucial em termos de consumo de tempo de computação numa simulação de dinâmica molecular se refere ao cálculo das forças. Assim, um bom algoritmo deve se aproveitar ao máximo do curto alcance das forças envolvidas e realizar o cálculo das forças apenas entre os pares de partículas que se encontram a distâncias menores que o raio de corte r_c.

Mas mesmo com o cálculo das interações apenas de partículas a distâncias menores que r_c, ainda resta um problema: como encontrar as partículas que efetivamente interagem. O método direto seria calcular a distância entre todos os possíveis pares de partículas, mas isso é insatisfatório, pois esse cálculo de distância cresceria com N^2.

Os dois métodos mais conhecidos de lidar com esses problemas são os de *tabela de vizinhos* e o *algoritmo de células*[1], que passaremos a descrever.

[1] Em [Tra89] chamamos este algoritmo de *algoritmo de granulação grosseira*. Neste trabalho estamos utilizando o termo *células*, mais de acordo com o costumeiro no campo de dinamica molecular: *linked cells method*. O *linked* foi retirado por não fazer parte essencial do método, mas apenas representar uma forma de implementação.

4.1.1 Algoritmo de tabela de vizinhos

O algoritmo chamado *tabela de vizinho* ou, mais extensamente, *tabela de vizinhos com atualização infreqüente*, foi inicialmente proposto por Verlet [Ver67]. A idéia central desse algoritmo é formar uma lista das partículas que interagem com cada partícula dada. Como as partículas não apresentam grandes deslocamentos por diversos passos de simulação, essas listas não precisam ser alteradas a cada passo, mas apenas a certos intervalos, isto é, a atualização da tabela de vizinhos é infreqüente.

A atualização pode ser realizada de maneira infreqüente desde que as listas de vizinhos não incluam apenas as partículas dentro de uma distância r_c da partícula dada, mas sim de uma distância $r_v > r_c$. A diferença $r_v - r_c$ determina o número de intervalos que a simulação pode prosseguir sem atualização da tabela. Esse número de intervalos pode ser tanto fixo como variável. No trabalho original de Verlet, esse número era fixo, escolhido de acordo com uma avaliação com ampla margem de segurança [Ver67]. No entanto, um método também possível é ficar monitorando os deslocamentos das partículas, e realizar uma atualização quando o maior deslocamento chegar no valor da diferença $r_v - r_c$ [AT87].

Assim, o tempo de processamento é, durante *a maior parte* da simulação, proporcional a N, sendo que somente nos instantes em que é necessária a atualização da tabela de vizinhos temos um tempo proporcional a N^2.

Como vimos em [Tra89], os pontos fracos deste método são:

- Resta uma fase do algoritmo proporcional a N^2, o que significa que essa fase será dominante para grande número de partículas, apesar de ser pequena quando lidamos com poucas partículas

- A fase de atualização da tabela de vizinhos é uma fase global, isto é, são necessárias informações de todas as partículas, o que dificulta a paralelização

4.1.2 Algoritmo de células

Um outro método, proposto originalmente por Quentrec e Brot (citado em [AT87]), e também estudado em [Tra89], é o de *células*. Consiste em dividir o espaço simulado em células, que têm como lado um valor igual ou maior que r_c, de forma que, ao se desejar encontrar as partículas interagentes com dada partícula, basta procurá-las na mesma célula da partícula ou em uma das células vizinhas imediatas. A fig. 4.1 demonstra isso. Como as forças entre duas partículas interagentes são simétricas, basta calcular uma única vez para cada par de partículas. Desta forma, apenas metade dos vizinhos precisam ser considerados, como por exemplo indicado na figura 4.2.

O número médio de partículas por célula depende apenas da densidade, em geral constante, de forma que o tempo de busca de interagentes será proporcional a N.

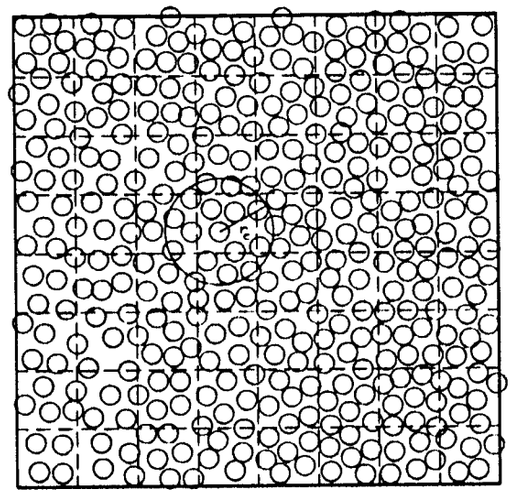

Figura 4.1: Divisão do espaço de simulação em células e busca das interagentes nas células vizinhas

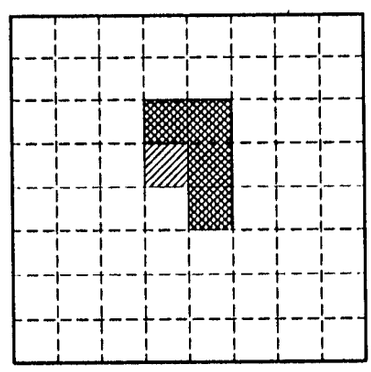

Figura 4.2: Varredura de metade das células vizinhas a uma dada célula. O hachurado simples indica a célula central, e o hachurado duplo indica as células vizinhas que devem ser varridas.

Tabela 4.1: Tempos de processamento médios em 128 passos com $\rho = 0.8442$ e $N = 2048$ num sistema bidimensional, para o algoritmo completo e o algoritmo sem redistribuição.

Algoritmo	Tempo de um passo (s)
Completo	1.017
Sem distribuição	0.953

Como demonstramos em [Tra89], este algoritmo é preferível ao de tabela de vizinhos para grandes quantidades de partículas, apresentando ainda a vantagem de facilitar a paralelização, pois cada processador pode ficar encarregado de uma parte das células, necessitando comunicar-se apenas com processadores vizinhos, como veremos com mais detalhes adiante.

Naturalmente, podemos utilizar atualização infreqüente também para este método, desde que o tamanho de cada célula seja feito suficientemente maior que o valor de r_c, de forma que a distribuição das partículas pelas células seja realizada apenas esporadicamente. No entanto, neste algoritmo isto não representa vantagem tão grande como no de tabela de vizinhos, pois a distribuição das partículas pelas células depende apenas do valor das coordenadas de cada partícula, e portanto é proporcional a N. Além disso, isto poderia implicar a necessidade de aumentar o tamanho da célula o que resultaria num aumento do tempo de execução. Como veremos no capítulo 5, a determinação do tamanho de cada célula depende de diversos fatores, de forma que pode ocorrer de não ser necessário o aumento do tamanho da célula.

Para comparação, realizamos duas simulações idênticas, mas em uma delas eliminando a parte de redistribuição das partículas pelos grão, durante todo o processamento, somente com o intuito de comparação de tempos, e sem nos preocuparmos com a possibilidade de introdução de erros no resultado. A tabela 4.1 apresenta os resultados. Como vemos, o tempo gasto com a redistribuição é da ordem de apenas 6% do tempo total. Não são esperadas alterações significativas deste valor com o aumento do número de partículas.

4.1.3 Combinação dos algoritmos

Propostas têm sido apresentadas de combinação dos dois métodos acima, de forma que:

1. Inicialmente se faria a distribuição das partículas por células
2. Em seguida, com base na distribuição em células, se levantaria uma tabela de vizinhos

Tabela 4.2: Comparação entre o método de células e o método combinado, com relação ao tempo de processamento, tanto para 2 como 3 dimensões. $N = 2048$, $\rho = 0.8442$.

Método	Tempo	
	2 dim.	3 dim.
Células	1.13	8.36
Combinado	0.77	3.56

3. Os dois processos acima seriam realizados infreqüentemente

Essa proposta foi apresentada por exemplo em [Abr86] e [BGGT90]. A idéia é diminuir, em relação ao algoritmo de tabela de vizinhos, a quantidade de pares que precisam ser considerados ao se calcular a tabela, de forma que também o passo de atualização da tabela seja proporcional a N.

Para avaliar o efeito dessa combinação, realizamos simulações com o método combinado e com o método de células. Os resultados são apresentados na tabela 4.2. Como vemos, os resultados são bastante significativos, principalmente no caso tridimensional. Desta forma, optamos pela utilização do método combinado neste trabalho.

4.2 Propostas de implementação paralela

Devido às grandes demandas computacionais envolvidas na simulação de dinâmica molecular, diversos pesquisadores têm apresentado propostas de implementação em arquiteturas paralelas de diversos tipos. Classificaremos as diversas propostas apresentadas nas seguintes categorias:

1. Implementações em supercomputadores de propósito geral
2. Implementações em arquiteturas paralelas de propósito específico
3. Implementações em arquiteturas paralelas de propósito geral

Veremos agora algumas propostas nessas categorias.

4.2.1 Implementações em supercomputadores de propósito geral

A execução de simulações de dinâmica molecular em supercomputadores é uma das formas mais simples de se conseguir uma diminuição dos tempos de processamento envolvidos. Isto se deve principalmente aos dois fatores seguintes:

- À alta velocidade de processamento *escalar* desses processadores
- À existência de ferramentas automáticas de vetorização, que permitem a extração de um certo nível de paralelismo do programa sem nenhuma necessidade de re-escrita do programa

Assim, a simples transferência do código de um processador normal para um supercomputador pode representar um grande ganho em tempo de processamento.

No entanto, essa simples transferência não é capaz de utilizar os recursos do computador de uma forma eficiente, pois os códigos de dinâmica molecular não apresentam uma estrutura muito regular de distribuição de operações, tendo condicionais inseridos em diversas partes do processamento dentro de ciclos. A capacidade de vetorização automática para esses códigos é portanto extremamente baixa, determinando baixa eficiência na utilização do supercomputador.

Diversas alterações são necessárias para que a eficiência seja aumentada, e essas alterações requerem bom conhecimento tanto do código como da arquitetura específica da máquina em uso.

Em [GDK89] temos um estudo de implementação de dinâmica molecular de partículas de Lennard-Jones em um computador Cray XMP. É utilizada uma combinação do algoritmo de células com o de tabela de vizinhos com atualização infreqüente e diversas técnicas são introduzidas para permitir a vetorização dos ciclos maiores, e também eliminar a vetorização de alguns ciclos muito curtos, que apresentariam menor desempenho processados vetorialmente devido ao tempo de latência para a inicialização das unidades vetorias. Os resultados apresentados nesse trabalho são claramente superiores a propostas anteriores de vetorização.

4.2.2 Implementações em arquiteturas paralelas de propósito específico

Uma outra possibilidade de aceleração dos cálculos de dinâmica molecular é o desenvolvimento de arquiteturas paralelas especialmente adequadas a esse problema. A grande vantagem desse método é que a arquitetura do processador pode ser projetada de forma a que:

1. Os recursos da máquina estejam completamente ocupados durante todo o ciclo de processamento
2. A máquina apresente maior quantidade de recursos justamente nos pontos onde são mais necessários para o algoritmo em questão.

Certamente, os dois pontos acima são somente um ideal e, em alguns casos, conflitantes entre sí, entretanto, eles podem mais facilmente ser conseguidos numa máquina de propósito específico do que em qualquer máquina de propósito geral.

Exemplos de processadores de propósito específico para dinâmica molecular são os propostos em [BBvDH82], para a simulação de até 16000 partículas de Lennard-Jones, em [Abr86] e [APB+87], onde é proposta a utilização de um processador semi-dedicado, chamado SPARK, numa configuração paralela de memória distribuída numa topografia semelhante à determinada pelo algoritmo de células. Esses processadores são interligados através de uma rede reconfigurável do tipo Benes. Uma proposta para processador dedicado para dinâmica molecular de partículas de Stillinger-Weber é apresentada em [BGGT90], para simulações da fase sólida de silício. Neste caso, é utilizado um conjunto especialmente construído de processadores de ponto flutuante, que são interligados numa arquitetura de memória compartilhada. O duto do sistema é capaz de suportar até 16 processadores, tendo cada um deles um desempenho de pico de 13.6 Mflops, e médio de aproximadamente 4.7 Mflops para o problema a que foi desenvolvido.

4.2.3 Implementações em arquiteturas paralelas de propósito geral

Nesta categoria incluímos as propostas de implementação de dinâmica molecular em arquiteturas paralelas cujos elementos básicos de processamento não foram desenvolvidos especialmente para o problema.

Podemos citar a proposta em [BS88] de cálculo da interação de todos os pares num sistema com até 4096 partículas no ICL-DAP, e o seu estudo da forma de levar em consideração todos os pares. [GKPV88] propõe o uso da *Connection Machine* para o cálculo de interações de todos os pares, e estuda o método mais eficiente de realizar a comunicação dos dados. [BF91] propõe uma estratégia *sistólica* de cálculo de todos os pares em uma rede de *transputers*. [PP89] apresenta um método de paralelização da dinâmica molecular de *poucas* partículas numa rede de *transputers*. [BBK91] apresenta uma forma de realizar balanço de carga dinâmico em uma rede de *transputers*, para o caso de problemas em que há separação de fase, o que poderia induzir grande desbalanceamento. [GV89] apresenta um método de paralelização do cálculo de energia de biomoléculas numa rede de *transputers*.

Comparativamente com a implementação em supercomputadores de propósito geral, a vantagem deste método é o mais fácil acesso e maior disponibilidade do equipamento, enquanto que sua desvantagem é uma necessidade maior de re-estruturação do código do programa em relação aos programas seqüenciais prontos. Comparando-se com a implementação em máquinas dedicadas, este método apresenta maior facilidade de implementação, pois não é necessário o desenvolvimento de processadores específicos. No entanto, a eficiência tende a ser menor, devido à arquitetura do processador não ser especialmente adequada ao problema em questão.

4.3 Forma proposta de paralelização

Neste trabalho, optamos pela implementação em uma rede de *transputers* [INM85], programada na linguagem occam [INM88]. As razões que nos levaram a essa escolha foram as seguintes:

- A versatilidade da implementação em um sistema de processamento paralelo de propósito geral, que nos permite, apenas com alterações de *software*, realizar diversos desenvolvimentos diferentes, e mesmo para programas distintos, sem queda muito significativa na eficiência
- A maior facilidade de implementação do que através de processadores dedicados, que ainda apresentariam a desvantagem de não ser tão facilmente adaptáveis a outros tipos de programas
- A maior disponibilidade e mais fácil acesso em relação a supercomputadores, presentes em poucas instituições de pesquisa do mundo
- A estrutura integrada desenvolvida no *transputer* própria para o processamento paralelo [INM85], ao contrário de outros processadores que exigem *hardware* especial para processamento paralelo
- A adaptação entre a linguagem occam e a arquitetura do *transputer*. A linguagem e o processador foram desenvolvidos em conjunto, de forma que ambos refletem o mesmo paradigma de programação, conhecido como *Communicating Sequential Processes* (CSP) [Hoa78]

Apresentaremos agora uma **breve** descrição do *transputer* T800 e de occam, e passaremos então a apresentar a forma escolhida de paralelização.

4.3.1 *Transputers* e occam

Em [Hoa78], Hoare apresentou um paradigma de programação denominado *Communicating Sequential Processes*, por se basear na programação paralela de diversos processos seqüenciais, cada qual interagindo com os outros unicamente através de comunicações explicitamente indicadas.

A linguagem occam se propõe a ser uma linguagem que trabalha com o paradigma de CSP, e os *transputers* apresentam uma arquitetura adequada ao processamento de programas desenvolvidos nessa linguagem. Faremos agora uma exposição dos pontos principais relativos ao T800 e a occam, sem tentar uma apresentação técnica dos mesmos, mas apenas indicando os pontos mais importantes com relação ao que será apresentado posteriormente neste trabalho. Mais detalhes podem ser encontrados em [INM88], [PM88] e [INM85].

As características básicas do *transputer* T800 podem ser indicadas como:

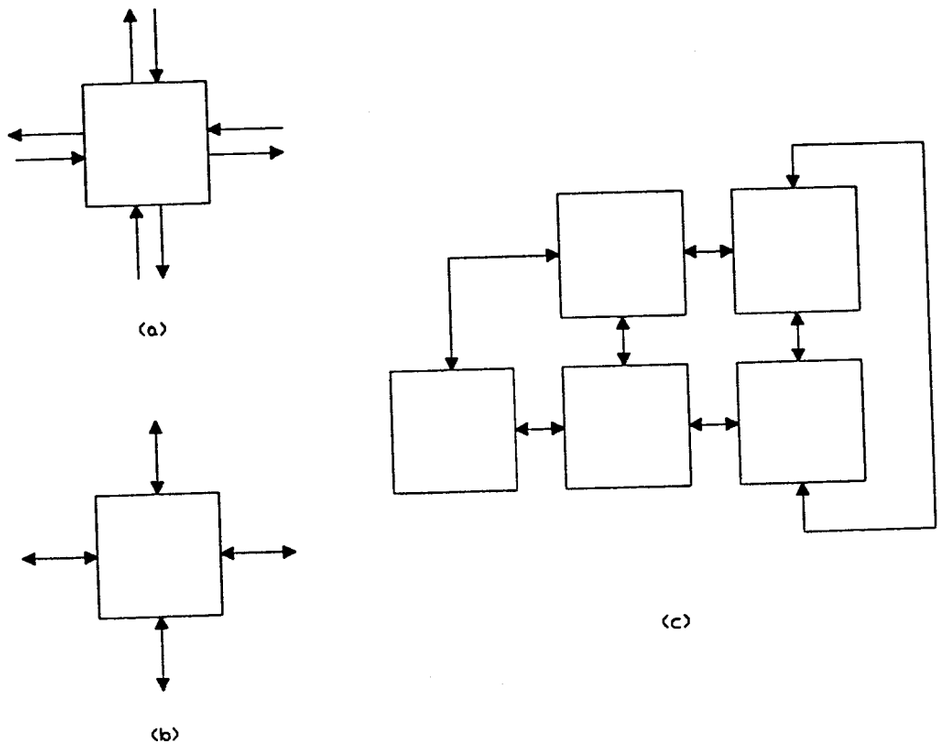

Figura 4.3: Representação esquemática de transputers. Em (a), temos um *transputer* e seus quatro *links*; em (b) uma representação simplificada e em (c), uma possível forma de interligação de um conjunto de *transputers*

- Unidade de processamento inteiro de 32 bits
- Unidade de processamento em ponto flutuante de 32 ou 64 bits
- Memória interna no processador de 4 Kbytes
- 4 linhas bidirecionais de ligação (*links*) com outros processadores, operando em 10 ou 20 Mbits/s
- Suporte interno para a multiprogramação, com chaveamento relativamente rápido entre processos (da ordem de $1\mu s$)

Estas duas últimas características são as que o tornam adequado ao multiprocessamento no paradigma de memória distribuída próprio de CSP.

O processador é normalmente representado como na fig. 4.3.

Com relação às características próprias da linguagem occam, gostaríamos de salientar:

SERVIÇO DE B... E INFORMAÇÃO - IFOSC

- Permite a expressão de um programa em forma explicitamente paralela. De fato, a expressão em forma paralela é *requerida* quando se pretende explorar o paralelismo do sistema de processamento, e o compilador occam não realizará nenhuma tentativa de extração automática de paralelismo

- Utiliza o paradigma de comunicação entre processos seqüenciais, tanto para a comunicação de dados, como para a sincronização entre os processos. Os elementos através dos quais os dados passam de um processo a outro são chamados *canais*.

- Permite (na realidade *requer*) a associação explícita de processadores a determinados processos, e de *links* a determinados canais

- Permite a execução de programas concorrentes em um mesmo processador, sem alteração quanto à funcionalidade com relação à execução em diversos processadores, facilitando assim o processo de desenvolvimento e depuração do programa.

4.3.2 Separação em anel

O fator básico que permitiu o desenvolvimento da forma de paralelização adotada é o fato das interações entre partículas serem de curto alcance. Desta forma, para um processador determinar a força total atuante em uma partícula ele necessita informações de apenas um conjunto restrito de partículas, distribuídas espacialmente perto da partícula considerada. Assim, se fizermos uma distribuição das partículas por diversos processadores de forma que partículas próximas[2] sejam alocadas no mesmo processador ou em processadores vizinhos, então os únicos dados necessários para o cálculo das forças sobre essas partículas estarão presentes no mesmo processador ou em um processador vizinho. Este tipo de paralelização, que se baseia numa distribuição dos dados pelos diversos processadores, é conhecido como *divisão geométrica* do trabalho [Hey89].

Como o *transputer* dispõe de apenas 4 linhas de conexão física (*links*) com processadores vizinhos, e considerando que além das comunicações com os vizinhos para informações sobre dados das partículas são necessárias comunicações com um processo central, para informação de resultados parciais, possíveis alterações de valores, etc., optamos, para evitar a multiplexação de mais de um canal por um mesmo *link*, por realizar a divisão em apenas uma das dimensões, formando uma estrutura em anel com na fig. 4.4.

Representando conjuntamente as comunicações de dados de partículas e as comunicações de controle e resultados, a interligação entre os diversos processadores ficará

[2] Entendemos aqui por *partículas próximas* aquelas que apresentam entre si uma distância menor que o raio de corte r_c.

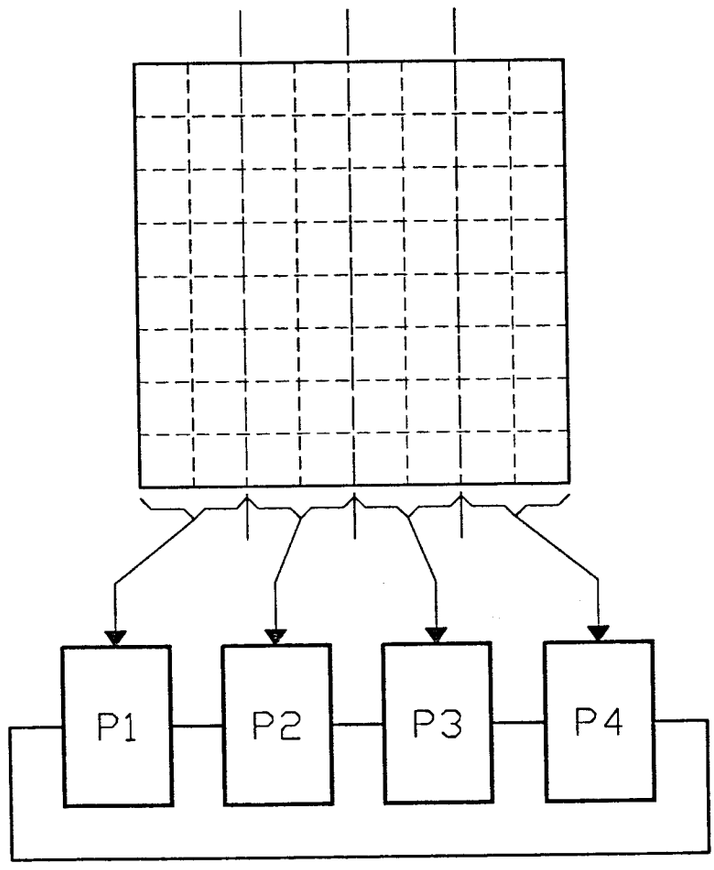

Figura 4.4: Divisão do espaço de simulação e anel de processamento

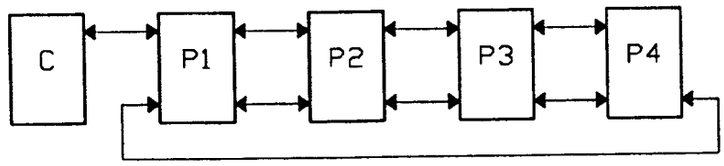

Figura 4.5: Representação das interligações entre os processadores, para o caso de 4 processadores de cálculo (Pn) e 1 de controle separado (C)

da forma indicada na fig. 4.5 (que representa o caso com 4 processadores de cálculo e 1 separado para controle). A utilização de um processador separado para as tarefas centralizadas de controle não é necessária, mas foi utilizada neste trabalho por razões que ficarão claras adiante.

Seria possível, se fosse utilizada a multiplexação dos dados de controle e resultados pelos mesmos *links* dos dados de partículas, realizar ainda a divisão em mais uma dimensão. No entanto, isso não foi tentado aqui por ter relevância apenas nos casos em que se deseja trabalhar com um número de processadores grande em relação ao número de partículas (isto ficará mais claro durante as análises dos resultados, cap. 5).

4.3.3 Duplicação de células

Vimos que, para o cálculo da força sobre uma partícula de dada célula, devemos considerar os dados das partículas de todas as células vizinhas, e que apenas metade dessas células precisa ser utilizada na busca de pares interagentes, devido à terceira lei de Newton. Assim, se adotamos o modo de varredura sugerido na fig. 4.2, então podemos notar os seguintes pontos:

- Para as células que ficam à extrema direita de um dos processadores, o cálculo da varredura de pares dessas células exige conhecimento das posições de partículas de células alocadas ao processador vizinho à direita
- Para as células que ficam à extrema esquerda de um dos processadores, é necessária a comunicação dos valores de força calculados durante a varredura das células à extrema direita do processador vizinho à esquerda

O método direto de resolver esse problema seria agir de acordo com o seguinte esquema:

1. Quando um processador chegasse na fase de cálculo das interações de uma célula da sua extrema direita, requisitaria os valores das posições das partículas para o seu vizinho à direita
2. O vizinho à direita atenderia a essa requisição, enviando os dados pedidos

3. O processador então realizaria os cálculos de interação com essas partículas

4. Após calculadas as interações, os valores das forças sobre as partículas que vieram do vizinho seriam comunicadas a esse vizinho, para ser acumuladas com os valores de aceleração calculados nesse vizinho

No entanto, esse esquema apresenta um grave inconveniente: a necessidade constante de sincronização e comunicação entre os processadores durante a fase crítica de cálculo das forças. Note-se que, como as comunicações entre *transputers* em occam são bloqueantes, e como não é permitido a um processo alterar dados que estão sendo utilizados por outro processo, essas sincronizações implicariam em que os processadores deveriam ficar alguns instantes sem calcular, aguardando comunicações dos vizinhos.

Para resolver esse problemas, desenvolvemos um método que consiste na duplicação de dados de células. Assim, cada processador, além das posições das suas próprias partículas, tem conhecimento também das posições das partículas que são necessárias para a varredura de suas células à extrema direita (isto é, tem conhecimento também das posições das partículas das células à extrema esquerda de seu vizinho da direita[3], veja fig 4.6).

Com isso, ficamos com o seguinte método:

1. Cada processador comunica os dados sobre as partículas nas suas células à extrema esquerda para os seus vizinhos à esquerda

2. Durante o cálculo das forças, cada processador age de forma completamente independente, usando os dados sobre as partículas de suas células e os dados recebidos do vizinho à direita

3. Após terminados os cálculos de forças, os valores de forças acumulados sobre as partículas recebidas do vizinho à direita são comunicadas a esse vizinho, e concomitantemente são recebidos os valores de força calculados pelo seu vizinho à esquerda sobre suas próprias partículas

4. Os valores de força são combinados, e as novas posições são calculadas à partir delas, começando-se um novo ciclo.

Como se pode notar, esse método exige comunicações apenas em instantes bem delimitados, fora do ciclo crítico de cálculo de forças, e ainda permite que todos os valores a serem comunicados sejam reunidos em um mesmo vetor, o que diminui o tempo de transmissão, por requerer apenas um tempo de latência de comunicação pelos *links*.

[3]Quando falamos de vizinhos à direita e à esquerda, não podemos nos esquecer que o problema é completamente periódico e, correspondendo a isso, os processadores estão organizados em anel. Assim, todos os processadores possuem tanto um vizinho à esquerda quanto um à direita.

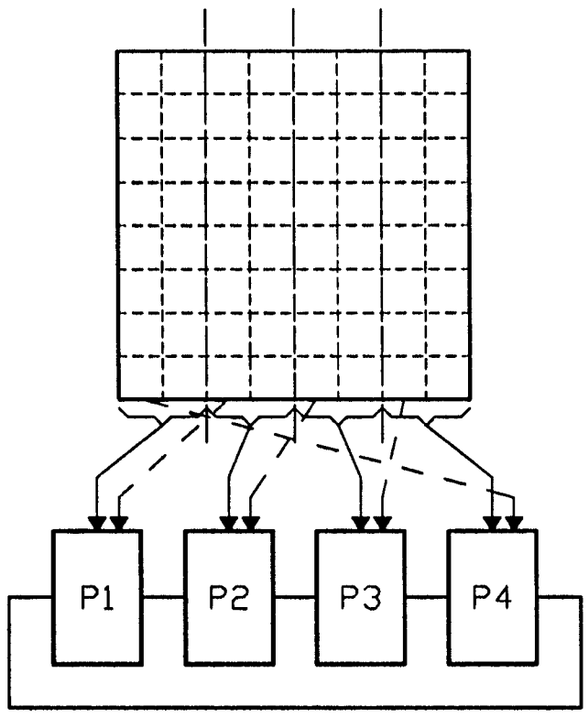

Figura 4.6: Duplicação das células nas bordas. As linhas tracejadas indicam as colunas de células duplicadas

4.3.4 Divisão homogênea ou não-homogênea

Quando realizamos a divisão do processamento entre os diversos processadores, nem sempre o número de células é divisível pelo número de processadores, ainda mais que as divisões devem ser feitas em apenas uma das direções.

Abordaremos aqui dois métodos de lidar com esse problema dentro da forma proposta de paralelização:

Divisão homogênea : Neste caso, para forçar uma divisão aproximadamente homogênea de partículas entre os processadores, mudamos a dimensão da célula na direção da divisão, de forma que o número de células nessa direção seja múltiplo do número de processadores

Divisão não-homogênea : Aqui, realizamos uma divisão das células em sua forma original, e alguns dos processadores ficam então com uma carga maior que os restantes

O primeiro método permite uma melhor distribuição de carga entre os processadores, mas aumenta o tamanho de cada célula. O segundo método não aumenta as células, mas introduz um desbalanço de carga entre os diversos processadores.

Conforme ficará mais claro no capítulo 5, o método de divisão homogênea é preferível ao de divisão não-homogênea, pois a parte que mais demanda tempo do cálculo é a varredura das tabelas de vizinhos durante o cálculo das forças, que não é afetada pelo tamanho das células. A parte afetada pelo tamanho das células é a *construção* da tabela de vizinhos, o que não é realizado em todos os intervalos de tempo, e que tem cálculo simples, dependendo apenas da posição de cada partícula.

4.3.5 Estrutura global do programa

Para tornar mais clara a operação conjunta dos diversos fatores indicados acima, apresentamos na fig. 4.7 um pseudo-código do programa. Nessa figura, indicamos por `rij` a distância entre as particulas i e j, `rv` o valor do raio de vizinhança (r_v), `rc` o raio de corte (r_c) e utilizamos a notação de `j > i` para indicar que o par formado por duas partículas de uma mesma célula deve ser considerado apenas uma vez. O termo "primeira parte do movimento" refere-se ao cálculo de $v_{n+1/2}$ e r_{n+1} do método da velocidade de Störmer, e o termo "segunda parte do movimento" refere-se ao cálculo de v_{n+1} (ver eq. 3.11). A expressão "vizinha posterior de uma célula" é utilizada para representar a escolha dos vizinhos de acordo com o esquema demonstrado na figura 4.2. No cálculo da interação são feitos os cálculos de força e de energia potencial de cada interacao.

```
LEIA dados iniciais das particulas
PARA passo DE 1 ATE Numero_de_Passos FACA
  PARA CADA particula i FACA
    CALCULE primeira parte do movimento de i
  FIM PARA
  SE passo MULTIPLO DE Intervalo_de_Redistribuicao
    PARA CADA particula i FACA
      CALCULE celula C correspondente a particula i
      GUARDE particula i EM celula C
    FIM PARA
  FIM SE
  EM PARALELO FACA
    ENVIE dados das celulas da primeira coluna
    RECEBA dados das celulas da coluna duplicada
  FIM PARALELO
  SE passo MULTIPLO DE Intervalo_de_Redistribuicao
    PARA CADA celula C FACA
      PARA CADA particula i DA celula C FACA
        PARA CADA particula j DA celula C COM j > i FACA
          SE rij > rv
            GUARDE particula j EM Tabela_de_Vizinhos de i
          FIM SE
        FIM PARA
        PARA CADA celula V vizinha posterior de C FACA
          PARA CADA particula j DE V FACA
            SE rij < rv
              GUARDE particula j EM Tabela_de_Vizinhos de i
            FIM SE
          FIM PARA
        FIM PARA
      FIM PARA
    FIM PARA
  FIM SE
  PARA CADA particula i FACA
    PARA CADA particula j DE Tabela_de_Vizinhos de i FACA
      SE rij < rc
        CALCULE interacao entre particulas i e j
      FIM SE
    FIM PARA
  FIM PARA
  EM PARALELO FACA
    ENVIE aceleracoes das celulas da coluna repetida
    RECEBA aceleracoes das celulas da primeira coluna
  FIM PARALELO
  PARA CADA particula i FACA
    CALCULE segunda parte do movimento de i
  FIM PARA
  CALCULE energia cinetica
  ENVIA totais deste passo para controle
FIM PARA
```

Figura 4.7: Descrição em pseudo-código do programa implementado

4.3.6 Limitações da paralelização

O modo de paralelização proposto apresenta duas limitações principais, que descreveremos agora:

Tamanho mínimo : A forma como a varredura em busca de pares interagentes é realizada limita os problemas que podem ser simulados com relação a um mínimo: o número de partículas e sua distribuição (densidade) deve ser de tal modo que seja possível a divisão em pelo menos 3 células, em cada dimensão. Assim, o número mínimo de células para o caso bidimensional é de 9, e para o caso tridimensional de 27. Isto não é um problema grave, pois simulações menores podem facilmente ser manuseadas com apenas um *transputer*. Entretanto, um método para lidar com simulações pequenas em um sistema de processamento paralelo foi apresentado em [PP89]

Número máximo de processadores : Dada uma simulação que possa ser dividida em n^d células, onde d é o número de dimensões, a paralelização apresentada permite uma implementação em *no máximo* n processadores. Isto porque uma divisão maior do problema seguindo o mesmo princípio de divisão geométrica somente seria possível com a divisão em mais de uma dimensão, o que não foi executado, conforme explicado acima (pag. 48). Esta já é uma limitação mais grave, mas não foi determinante dentro dos estudos realizados, conforme veremos no capítulo 5.

4.3.7 Estrutura das ligações

Passaremos a apresentar a descrição da implementação do sistema discutido. Primeiro, apresentaremos as características básicas da placa IMS-B008 e outras, utilizadas neste trabalho, e em seguida apresentaremos o modo como o sistema foi implementado nesse *hardware*.

Hardware disponível

Para a implementação do programa de dinâmica molecular, dispunhamos do seguinte *hardware*:

- Placa-base IMS-B008, com a capacidade de instalação de até 10 módulos TRAM[4], incluindo uma chave de interligação configurável C004 para escolha por software das interconexões entre as TRAMs. A TRAM instalada no *slot* 0 é a responsável pela programação da chave C004, e tem o *link* 1 de seu *transputer* conectado

[4]TRAM é um padrão da INMOS para placas a serem utilizadas em sistemas de processamento paralelo.

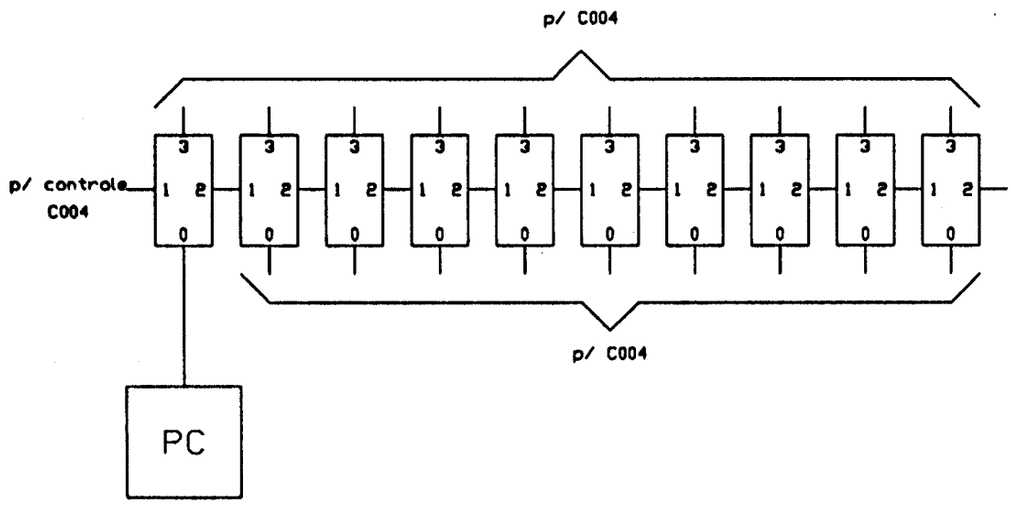

Figura 4.8: Representação das interligações da placa B008

com essa chave. Todas as TRAMs instaladas têm os *links* 1 e 2 de seus processadores interligados por *hardware*: *link* 2 no *link* 1 do processador seguinte, com exceção do processador do *slot* 0, que tem o *link* 1 ligado na C004, conforme dito, e do processador do *slot* 10, que tem o seu *link* 2 interligado num conector de borda. Os *links* 0 e 3 de todos os processadores são interligados à chave C004, para escolha por *software* das interconexões, com exceção do *link* 0 do *slot* 0, que é interligado ao hospedeiro (no caso um computador tipo IBM-PC). Veja fig. 4.8 para uma representação das interligações da placa B008. Um programa foi fornecido juntamente com a placa para a configuração da chave C004. Mais detalhes podem ser encontrados em [INM90].

- 1 módulo TRAM IMS-B404, com um processador T800 de 25 MHz, 2 Mbytes de memória, e *hardware* para controle de um subsistema de *transputers*

- 4 módulos TRAM IMS-B411, cada qual com 1 processador T800 de 20 MHz, e 1 Mbytes de memória.

Interconexão

Como vimos na fig. 4.5, a estrutura de interligação dos processadores consiste basicamente num anel, usado para a comunicação dos dados de partículas e acelerações entre processadores, ao qual se acrescenta um ligação em cadeia para a comunicação de informações de controle e resultados. Vemos também nessa figura, que cada processador de cálculo utiliza deve utilizar todos os seus 4 *links* para as comunicações.

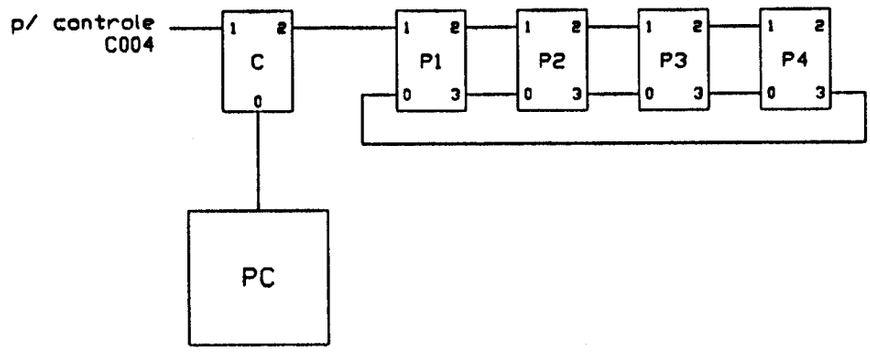

Figura 4.9: Representação das interconexões e distribuição utilizada neste trabalho (no exemplo, para 4 *transputer* de cálculo (Pn) e um de controle (C))

No *hardware* disponível, o processador alocado no *slot* 0 tem automaticamente seu *link* 1 ligado à chave C004 e seu *link* 0 ao hospedeiro. Isto impossibilita a execução em um mesmo processador da parte de controle com uma das partes de processamento, o que seria em princípio viável, dado que a parte de controle não apresenta grande gasto de tempo de computação nas suas tarefas, pois é responsável apenas por comunicações e alguns cálculos simples. Além disso, o processador de que dispomos com 2 Mbytes de memória (necessária durante o processo de desenvolvimento do programa), e que deve ser ligado no *slot* 0, opera a 25 MHz, o que difere da operação dos demais processadores, a 20 MHz. Assim, a sua utilização implicaria numa distorção no levantamento de dados relativos à paralelização. Em vista disso, decidimos fazer uma distribuição de forma que o processador do *slot* 0 ficou responsável apenas pela parte de controle, sem realizar nenhum cálculo da dinâmica molecular. A estrutura das interconexões ficou então como indicado na fig. 4.9.

Os arquivos utilizados para a programação da chave C004 para 1, 2, 3 e 4 processadores estão apresentados no apêndice A.

O apêndice B apresenta o código occam de dinâmica molecular, incluindo todos os detalhes apresentados.

4.4 Conclusão

Vimos que uma implementação que combine os algoritmos de células e de tabela de vizinhos com atualização infreqüente representa a uma boa escolha para o gerenciamento do cálculo de forças, e que apresenta também grandes vantagens quando se trata de efetuar uma paralelização.

Realizamos a implementação em um conjunto de *transputers*, com interligações em forma de anel e separação geométrica dos dados em apenas uma das direções, utilizando um sistema com base em computador do tipo IBM-PC e módulos do padrão

TRAM. Vimos que essa forma de implementação apresenta limitações tanto para quantidades muito pequenas de partículas quanto para quantidades grandes de processadores.

No próximo capítulo, realizaremos uma análise quantitativa do sistema proposto.

Capítulo 5

Avaliação e resultados

Neste capítulo apresentaremos análises *a priori* e levantamento de resultados para avaliação da proposta apresentada. Focalizaremos principalmente os aspectos relativos a eficiência de paralelização, mas apresentaremos antes um levantamento demonstrando a correção dos resultados, de acordo com a verificação da conservação da energia total, conforme discutido no capítulo 2.

5.1 Validação

Realizamos os testes de validação através do cálculo da flutuação média quadrática da energia, que deve ser proporcional ao quadrado de Δt, conforme vimos no capítulo 2. Fizemos então um levantamento da variação da flutuação com Δt, para o caso de $N = 2048$, $\rho = 0.6$, $r_c = 2.5$, $r_v = 2.8$, e atualizações da tabela de vizinhos a intervalos de aproximadamente 0.092, com a simulação se extendendo por um tempo simulado total de aproximadamente 2.0, tanto para o caso bidimensional como para o caso tridimensional. Os resultados são apresentados respectivamente nas figuras 5.1 e 5.2. Vemos que na faixa de Δt entre 0.001 e 0.03, o comportamento é exatamente o esperado, apresentando uma discrepância apenas fora dessa faixa, para baixo devido à perda de precisão nas operações numéricas, e para cima devido à divergência da integração das equações de movimento.

5.2 Tempo de processamento

Apresentaremos aqui cálculos do tempo de processamento em função do número de partículas (com apenas 1 processador) e do número de processadores (tendo fixo o número de partículas). Desenvolveremos antes uma expressão, com base em aproximações médias, do tempo de processamento em função de alguns parâmetros da simulação, e então apresentaremos os tempos levantados.

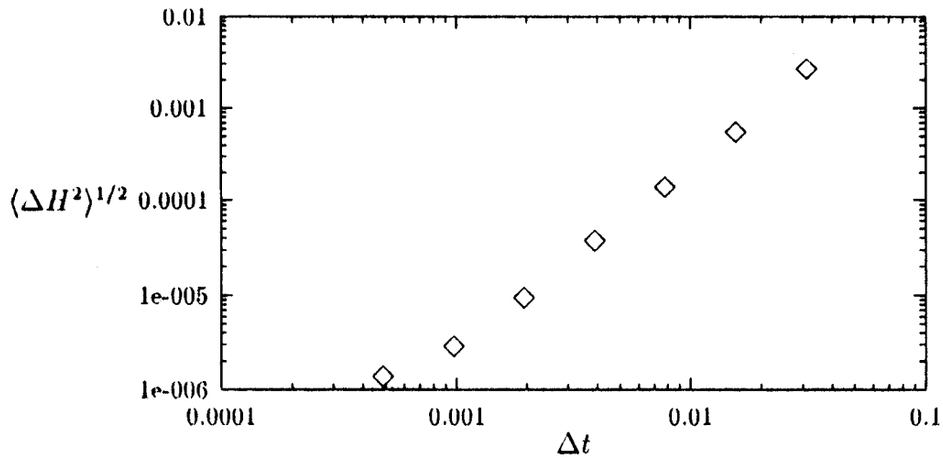

Figura 5.1: Variação de $\langle \Delta H^2 \rangle^{1/2}$ com Δt, para o caso bidimensional

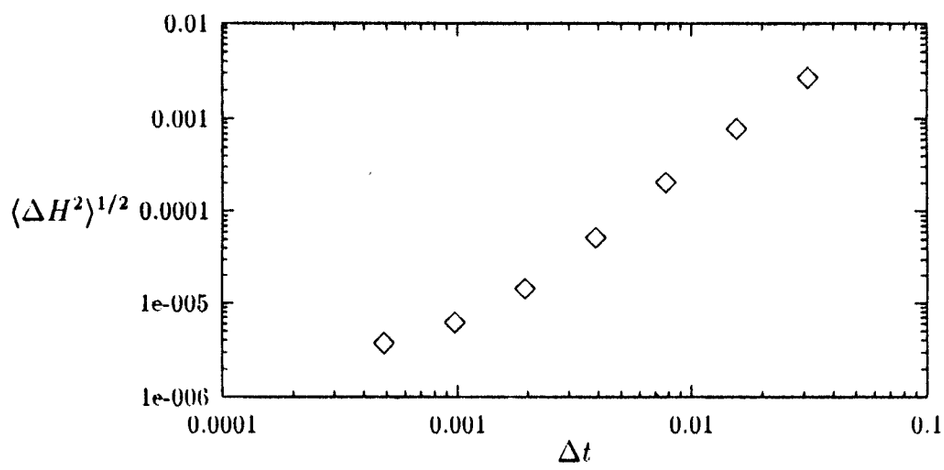

Figura 5.2: Variação de $\langle \Delta H^2 \rangle^{1/2}$ com Δt, para o caso tridimensional

5.2.1 Cálculo do tempo de processamento

Para o levantamento de uma expressão que permita o cálculo do tempo de processamento, dados os parâmetros da simulação e o número de processadores, utilizaremos continuamente valores médios em locais onde há variação, e desprezaremos alguns fatores relativos a inicializações e similares, tomando em consideração apenas a parte do processamento que realiza o ciclo de atualização de coordenadas e os cálculos executados simultaneamente.

Um único processador

Inicialmente, calcularemos o tempo de processamento com um único processador, e depois apresentaremos uma generalização da expressão para maior número de processadores.

Podemos discriminar o tempo de processamento nas seguintes partes:

- Controle geral do ciclo e inicializações, que envolvem um tempo total constante, que chamaremos de t_i
- Parte relativa a inicializações da estrutura de células, envolvendo um fator proporcional ao número de células. Denotaremos este fator por Mt_c, onde M é o número de células e t_c é um fator de tempo. Podemos calcular M por:

$$M = \left\lfloor \frac{\sqrt[d]{N/\rho}}{r_v} \right\rfloor^d \tag{5.1}$$

  d é o número de dimensões, e $\lfloor x \rfloor$ indica o maior inteiro menor que x.

- Partes relativas a comunicações de dados sobre células duplicadas[1] (recebimento de posições e envio de forças calculadas). Esta parte é proporcional ao número de células duplicadas e ao número médio de partículas por célula. Denotando por D o número de células duplicadas, e notando que o número médio de partículas por célula é N/M, esta parte pode ser representada por

$$D\frac{N}{M}t_d$$

  onde t_d é um fator de tempo. D pode ser calculado como:

$$D = \left\lfloor \frac{\sqrt[d]{N/\rho}}{r_v} \right\rfloor^{d-1} \tag{5.2}$$

[1]Notemos aqui que para utilizar o mesmo algoritmo não importa o número de processadores, realizamos a duplicação de células mesmo quando temos apenas um processador.

- Uma parte proporcional ao número de partículas no sistema, incluindo as partes de atualização das posições, cálculo da célula-destino da partícula e mesmo cálculo da força total sobre cada partícula contando-se já pronta a tabela de vizinhos. Representaremos esta parte por Nt_p, onde t_p é um fator de tempo

- A parte de atualização da tabela de vizinhos, que envolve a busca nas células vizinhas de uma dada célula. Assim, esse tempo é proporcional ao número de células M e ao quadrado do número de partículas por célula $(N/M)^2$. Podemos representar essa parte então por:

$$\left(\frac{N}{M}\right)^2 Mt_v$$

  onde t_v é um fator de tempo.

Juntando todos esses fatores encontramos:

$$T_s = t_i + Mt_c + D\frac{N}{M}t_d + Nt_p + \left(\frac{N}{M}\right)^2 Mt_v \tag{5.3}$$

É necessário frisar que os fatores t_i, t_c, t_d, t_p e t_v terão valores distintos conforme se trate de simulação em 2 ou 3 dimensões.

Notemos que N/M é aproximadamente constante para uma dada densidade (apresentando apenas pequenas flutuações devido à necessidade de arredondar o número de células para inteiro). M é por sua vez aproximadamente proporcional a N, apenas com flutuações determinadas pelo truncamento para inteiro. Assim, a expressão como um todo é aproximadamente proporcional a N, conforme já indicado no capítulo 4.

Mais de um processador

Quando estamos lidando com mais de um processador, devemos considerar a forma de distribuição do processamento entre esses processadores, ao calcular o valor do tempo de processamento.

Basicamente as mesmas expressões apresentadas para o caso de apenas um processador servem para o caso de diversos processadores, se fizermos os seguintes ajustes:

- O número de partículas que deve entrar nos cálculos é o número de partículas no processador que apresenta mais partículas. Assim, representaremos por $\hat{N}$ esse número. Notemos aqui que no caso de divisão homogênea $\hat{N} \approx N/P$, onde P é o número de processadores.

- O fator M deve ser ajustado, pois agora, ao determinarmos o número de células em que o sistema deve ser dividido, devemos levar em consideração que em

uma das direções (a direção em que estamos realizando a divisão pelos processadores), teremos algum ajuste. Para o caso de divisão não-homogênea, alguns processadores ficarão com uma quantidade maior de células, e no caso de divisão homogênea, devemos re-calcular o número de células de forma a conseguir uma divisão inteira na direção considerada. Denotaremos então por $\hat{M}$ o maior número de células em um processador. No caso de divisão homogênea teremos:

$$\hat{M} = \left\lfloor \frac{\sqrt[d]{N/\rho}}{r_v} \right\rfloor^{d-1} \left\lfloor \frac{\sqrt[d]{N/\rho}}{r_v P} \right\rfloor \tag{5.4}$$

Com essas alterações, a expressão fica:

$$T_s = t_i + \hat{M} t_c + D\frac{\hat{N}}{\hat{M}} t_d + \hat{N} t_p + \left(\frac{\hat{N}}{\hat{M}}\right)^2 \hat{M} t_v \tag{5.5}$$

Note-se que, como $\hat{N}$ e $\hat{M}$ tendem a ser divididos de forma aproximadamente igual entre os processadores, o *speedup*, definido na equação (5.6), pag. 66, tende a ser aproximadamente linear.

5.2.2 Levantamento do tempo de processamento

Para o levantamento da dependência do tempo de processamento com o número de partículas, e com o número de processadores, realizamos diversas simulações, em apenas um processador, com os seguintes parâmetros:

- $\Delta t = 0.00462$
- $\rho = 0.6$
- $r_c = 2.5$
- $r_v = 2.8$
- $f = 0.05$, isto é, atualização da tabela de vizinhos a cada 20 intervalos de tempo
- N foi variado nos valores 512, 968, 1458, 2048, 2450 e 2888 para a simulação bidimensional e 864, 1372, 2048, 2916 e 4000 para a simulação tridimensional.
- P foi variado de 1 a 4.

Com os dados conseguidos, montamos as tabelas 5.1 e 5.2

Podemos notar que os tempos de execução com divisão não-homogênea são sempre maiores ou iguais aos tempos com divisão homogênea, de acordo com o discutido

Tabela 5.1: Tempos de simulação (em segundos) para o caso bidimensional

		Homogênea			Não-homogênea		
N	$P=1$	$P=2$	$P=3$	$P=4$	$P=2$	$P=3$	$P=4$
512	0.143	0.074	0.051	0.041	0.074	0.057	0.046
968	0.268	0.138	0.094	0.073	0.138	0.099	0.082
1458	0.404	0.209	0.140	0.108	0.211	0.145	0.118
2048	0.566	0.287	0.194	0.148	0.287	0.200	0.148
2450	0.677	0.342	0.231	0.178	0.342	0.250	0.192
2888	0.795	0.401	0.268	0.205	0.401	0.268	0.205

Tabela 5.2: Tempos de simulação (em segundos) para o caso tridimensional

		Homogênea			Não-homogênea		
N	$P=1$	$P=2$	$P=3$	$P=4$	$P=2$	$P=3$	$P=4$
864	1.041	0.537	0.384	0.290	0.538	0.513	0.291
1372	1.727	0.881	0.630	0.478	0.882	0.867	0.479
2048	2.508	1.309	0.917	0.681	1.507	1.022	1.025
2916	3.504	1.807	1.203	0.970	1.807	1.212	1.176
4000	4.930	2.496	1.684	1.376	2.499	1.696	1.683

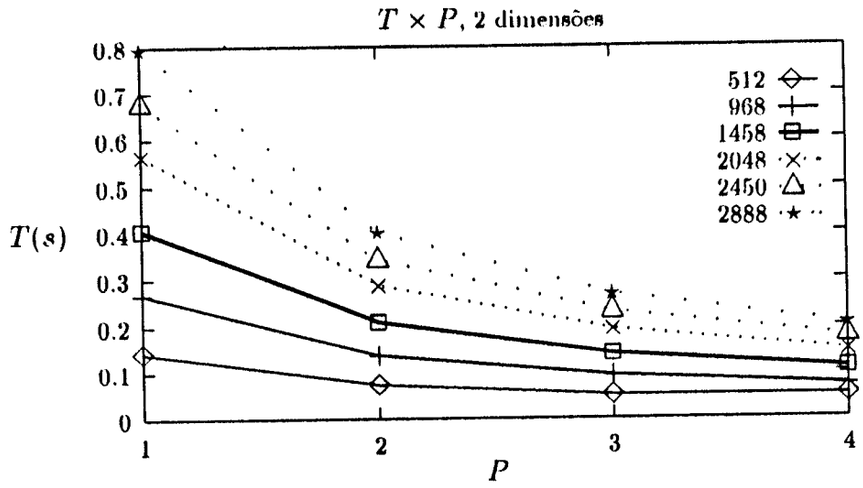

Figura 5.3: Tempos de um passo de simulação para o caso bidimensional

na pag. 51. Assim, à partir deste ponto, consideraremos apenas o caso de divisão homogênea.

Com os valores dessas tabelas, construímos os gráficos das figuras 5.3 e 5.5 (nesses gráficos e nos seguintes, ligamos os pontos através de linhas para facilitar a visualização). Também utilizando esses valores, realizamos um ajuste de parâmetros por mínimos quadrados na equação (5.5), e utilizamos o resultado para extrapolação, montando os gráficos das figuras 5.4 e 5.6. Note-se que as extrapolações do caso tridimensional são interrompidas nos pontos onde uma divisão adicional se tornaria impossível.

5.2.3 Variação com o número de partículas

Para uma mais fácil análise da variação do tempo de processamento com o número de partículas, observe os gráficos das figs. 5.7 e 5.8, onde estão apresentados os resultados para os casos bi- e tri-dimensionais, respectivamente. Como vemos, o crescimento com o tempo é praticamente linear em ambos os casos, conforme esperado.

Variação com o número de processadores

Para melhor apreciar o desempenho do sistema com relação à paralelização, preparamos os gráficos das figuras 5.9 e 5.11, onde são apresentados os levantamentos dos *speedups* em função do número de processadores. Também mostramos os valores de *speedup* de acordo com os valores extrapolados calculados, nas figs. 5.10 e 5.12. O valor do

SERVIÇO DE ... INFORMAÇÃO - IFOSC

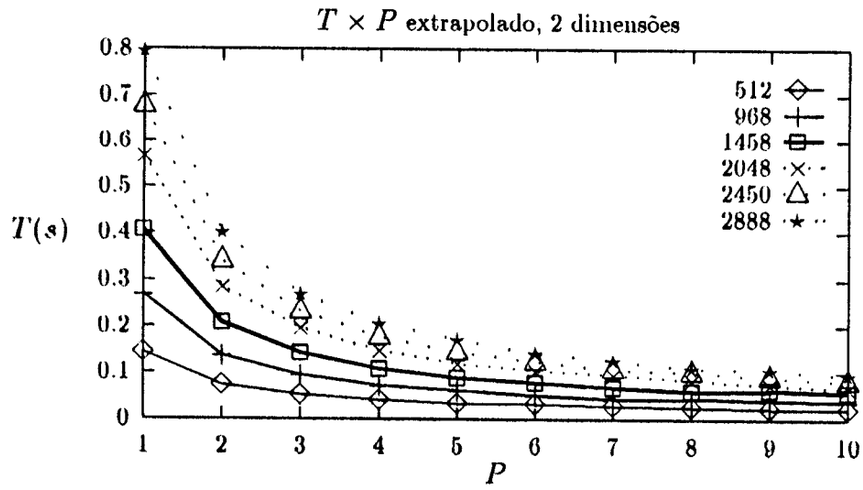

Figura 5.4: Tempos de um passo de simulação para o caso bidimensional (extrapolados)

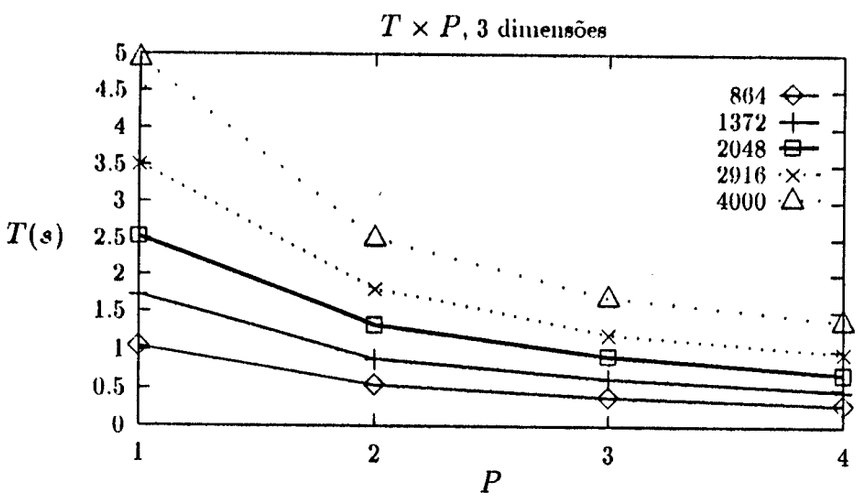

Figura 5.5: Tempos de um passo de simulação para o caso tridimensional

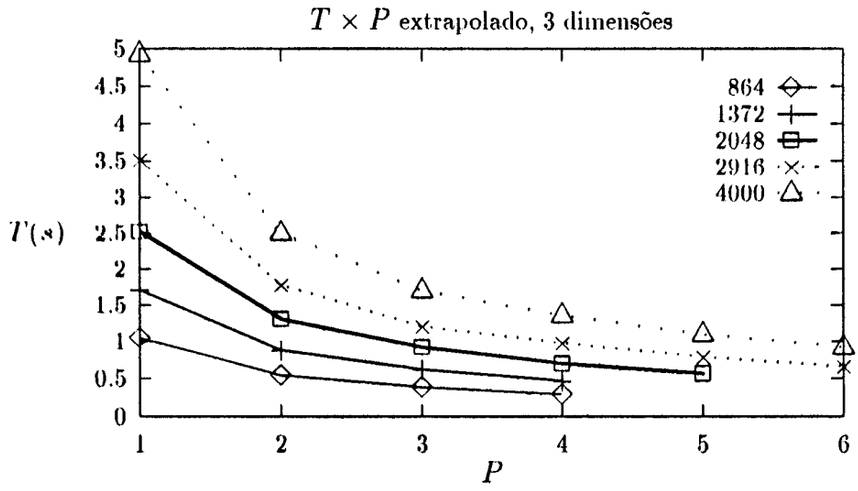

Figura 5.6: Tempos de um passo de simulação para o caso tridimensional (extrapolados)

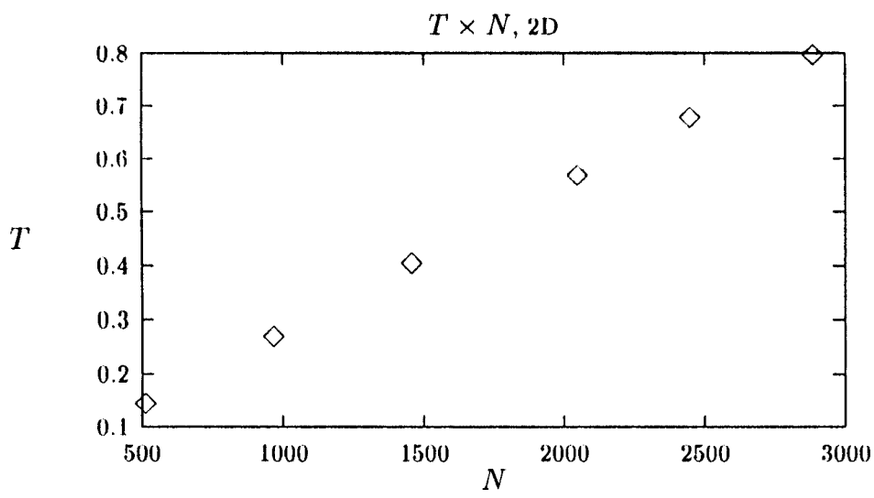

Figura 5.7: $T \times N$ para o caso bidimensional

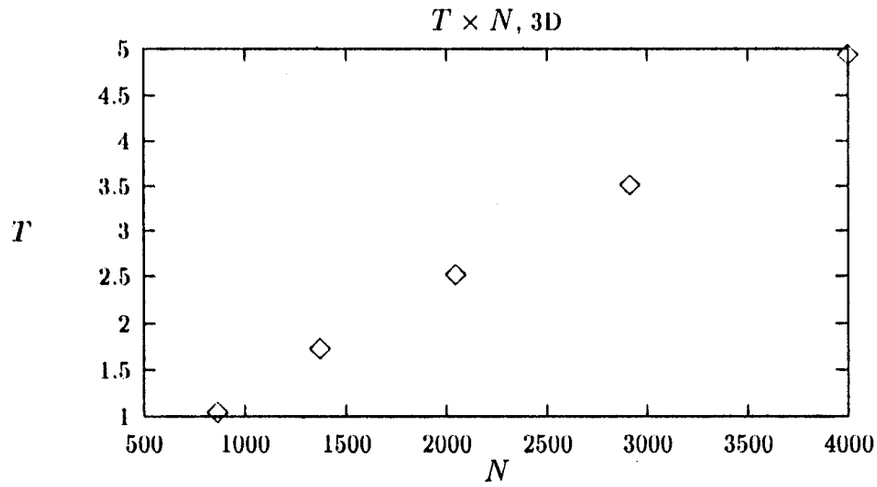

Figura 5.8: $T \times N$ para o caso tridimensional

speedup S é definido por:

$$S = \frac{T_1}{T_P} \tag{5.6}$$

onde T_1 é o tempo de execução em um processador, e T_P é o tempo em P processadores.

5.3 Conclusão

Analisando os resultados apresentados, bem como as equações levantadas, podemos concluir o seguinte:

1. O tempo de processamento cresce de forma aproximadamente linear com o número de partículas
2. A paralelização é mais eficiente para o caso bidimensional do que para o tridimensional
3. Os *speedups* são melhores quando se aumenta o número de partículas
4. Não existe uma tendência específica de queda da eficiência de utilização dos processadores com o aumento do número de processadores, mas apenas flutuações determinadas pelas diversas condições de arredondamentos nas distribuições

Assim, o método proposto representa uma boa opção de paralelização para situações onde se deseja trabalhar com grandes quantidades de partículas.

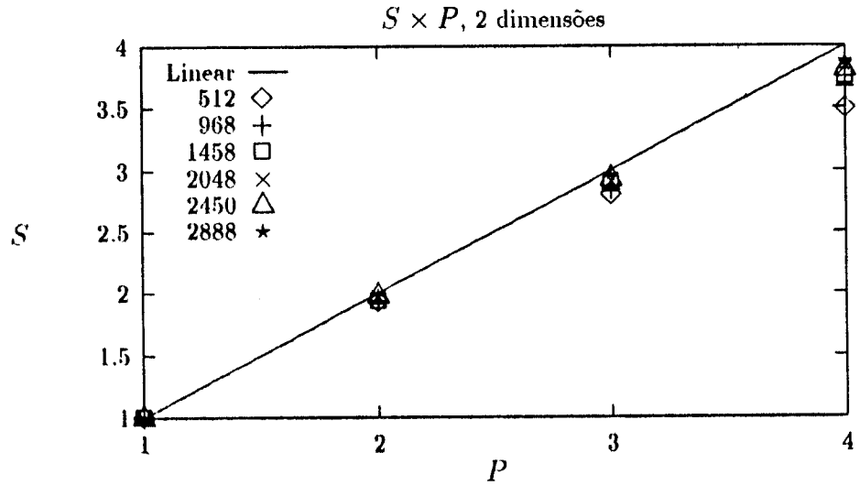

Figura 5.9: *Speedups* para o caso bidimensional

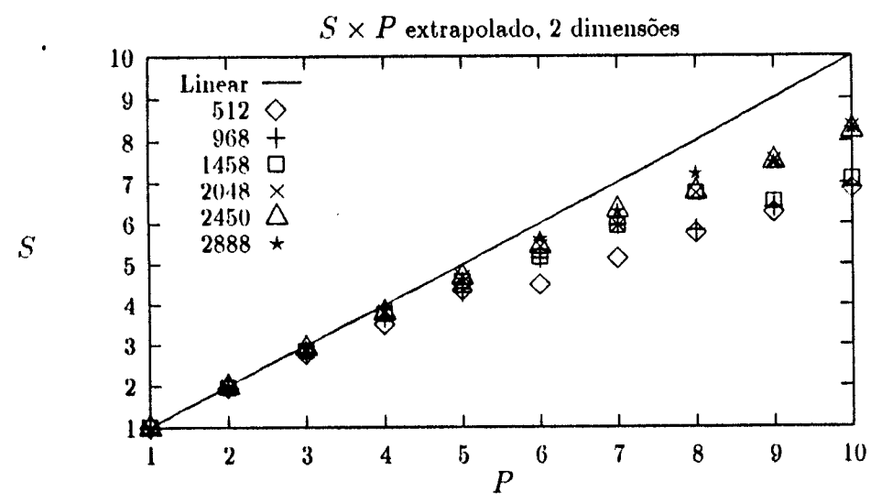

Figura 5.10: *Speedups* para o caso bidimensional (com extrapolação)

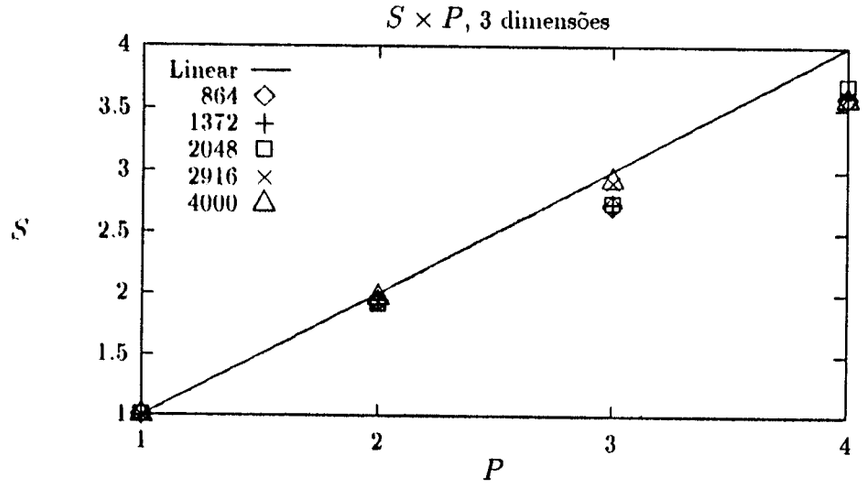

Figura 5.11: *Speedups* para o caso tridimensional

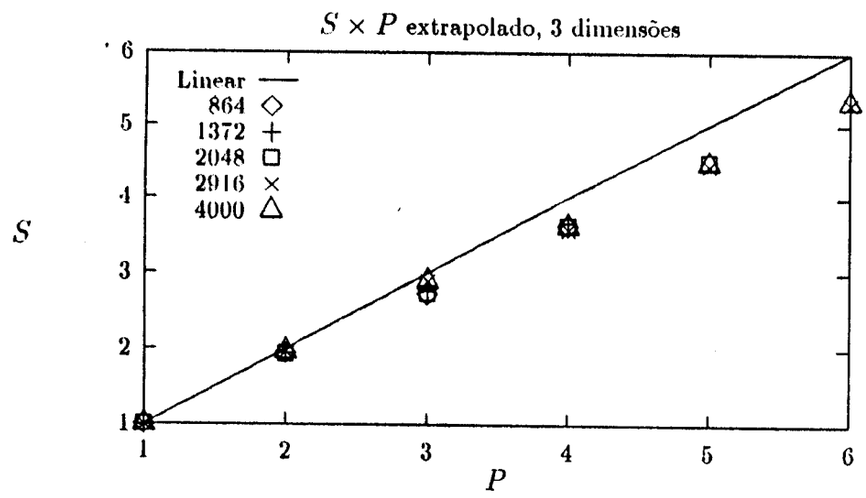

Figura 5.12: *Speedups* para o caso tridimensional (com extrapolação)

Capítulo 6

Conclusões e futuros trabalhos

Apresentaremos agora de forma concisa as principais conclusões deste trabalho, e indicaremos direções possíveis para futuras pesquisas.

6.1 Conclusão

De acordo com o exposto nos capítulos anteriores, podemos apresentar as seguintes conclusões e pontos a serem destacados:

- O estudo realizado se restringiu a partículas de Lennard-Jones, com movimento determinado pelas equações de Newton (isto é, simulação no ensemble microcanônico) com condições de contorno periódicas;
- O algoritmo de integração utilizado para a solução das equações de movimento foi o de Störmer, variante da velocidade. Nenhum levantamento foi feito com outros algoritmos;
- Foram utilizadas representações de 32 bits para posições, acelerações e velocidades, o que se mostrou adequado dentro dos estudos efetuados;
- Uma nova técnica de gerenciamento das condições de contorno periódicas através do uso de operadores especiais de occam foi introduzida, que permitiu sensível ganho no tempo de processamento e na precisão;
- O uso de tabela de *look-up* se mostrou inadequado para o potencial em questão;
- Para o gerenciamento dos pares de partículas interagentes durante o cálculo de forças, foi utilizado um algoritmo que combina os algoritmos de células e de tabela de vizinhos com atualização infreqüente, que tem bom desempenho e facilita o processo de paralelização ;
- A forma de interligação escolhida para os processadores foi um anel, com ligações adicionais para comunicação de parâmetros e totais. Esta forma se apresentou

adequada, desde que o número de processadores utilizados não seja excessivamente alto ou o número de partículas muito baixo;

- A implementação efetuada apresenta crescimento aproximadamente linear com o número de partículas;
- O aumento do *speedup* não apresenta nenhuma tendência definida de afastamento do *speedup* linear ideal, mas apenas flutuações, determinadas pelos arredondamentos durante a distribuição das partículas;
- A implementação é mais eficiente quando o número de partículas é maior;
- A forma de paralelização proposta apresenta limites de número mínimo de partículas (dados os outros fatores da simulação), e de número máximo de processadores (dados todos os fatores da simulação).

6.2 Futuros trabalhos

Dentro da linha do trabalho aqui apresentado, podemos apresentar a proposta dos seguintes estudos:

- Adaptação do método para potencias de triplas, como o de Stillinger-Weber;
- Adaptação do método para sistemas moleculares, onde se necessita considerar restrições determinadas pelas ligações entre átomos;
- Testes de simulações em outros tipos de ensemble, que não o microcanônico;
- Teste de outros algoritmos de integração e seus efeitos na precisão e no tempo de processamento;
- Paralelização da parte de levantamento de valores, como cálculos de fatores estruturais, etc.;
- Estudos de formas de contornar os problemas relativos a número máximo de processadores, através de nova forma de divisão geométrica do problema ou de alguma divisão algorítmica.

Bibliografia

[Abr86] Farid. F. Abraham. Computation statistical mechanics — metodology, properties and supercomputing. *Adv. in Physics*, 35(1):1–111, 1986.

[And80] H. C. Andersen. Molecular dynamics simulations at constant pressure and/or temperature. *Journal of Chemical Physics*, 72(4):2384–2393, 1980.

[And83] H. C. Andersen. Rattle: A "velocity" version of the Shake algorithm for molecular dynamics calculations. *Journal of Computational Physics*, 52:24–34, 1983.

[APB+87] D. J. Auerbach, W. Paul, A. F. Bakker, C. Lutz, W. E. Rudge, and F. F. Abraham. A special purpose parallel computer for molecular dynamics: Motivation, design, implementacion and application. *Journal of Physical Chemistry*, 91(19):4881–4890, 1987.

[AT87] M. P. Allen and D. J. Tildesley. *Computer simulation of liquids.* Claredon Press, 1987.

[AW59] B. J. Alder and T. E. Wainwright. Studies in molecular dynamics I. General method. *Journal of Chemical Physics*, 31(2):459–466, 1959.

[AW70] B. J. Alder and T. E. Wainwright. Studies in molecular dynamics II. Behavior of a small number of hard spheres. *Journal of Chemical Physics*, 33(5):1439–1451, 1970.

[BBK91] J. E. Boillat, F. Brugè, and P. G. Kropf. A dynamic load balancing algorithm for molecular dynamics simulation on multi-processor systems. *Journal of Computational Physics*, 96(1):1–14, 1991.

[BBvDH82] A. F. Bakker, C. Bruin, F. van Dieren, and H. J. Hilhorst. Molecular dynamics of 16000 Lennard-Jones particles. *Phys. Letters*, 93A(2), 1982.

[BF91] F. Brugè and S. L. Fornili. On the sistolic calculation of all-pairs interactions using transputer arrays. *Journal of Computational Physics*, 96:224–228, 1991.

[BGGT90] A. F. Bakker, G. H. Gilmer, M. H. Grabow, and K. Thompson. A special purpose computer for molecular dynamics calculations. *Journal of Computational Physics*, 90:313–335, 1990.

[BLQ73] J. Barojas, D. Levesque, and B. Quentrec. Simulation of diatomic homonuclear liquids. *Physical Review A*, 7(3):1092–1105, 1973.

[BS88] L. L. Boyer and Pawley G. S. Molecular dynamics of clusters of particles interacting with pairwise forces using a massively parallel computer. *Journal of Computational Physics*, 78:405–423, 1988.

[DB74] G. Dahlquist and Åke Björck. *Numerical Methods.* Prentice-Hall, 1974.

[EM83] D. J. Evans and G. P. Morris. The isothermal/isobaric molecular dynamics ensemble. *Physics Letters*, 98A(8,9):433–436, 1983.

[GDK89] G. S. Gret, B. Dünweg, and K. Kremer. Vectorized link cell FORTRAN code for molecular dynamics simulations for a large number of particles. *Computer Physics Communications*, 55:269–285, 1989.

[GKPV88] D.L. Greenwell, R. K. Kalia, J. C. Patterson, and P. D. Vashishta. Molecular dynamics algorithm on the connection machine. In H. D. Simon, editor, *Proceedings of the Conference on Scientific Aplications of the Connection Machine*, september 1988.

[Gre90] D. Greenspan. A counterexample of the use of energy as a mesure of computational accuracy. *Journal of Computational Physics*, 91:490–494, 1990.

[GV89] J. M. Goodfellow and F. Vovelle. Biomolecular energy calculations using transputer technology. *Eur. Biophys. Journal*, 17:167–172, 1989.

[Hey89] A. J. G. Hey. The role of MIMD arrays of transputers in computational physics. *Computer Physics Communications*, 56(1):1–24, 1989.

[HLM82] W. G. Hoover, A. J. C. Ladd, and B. Moran. High-strain-rate plastic studied via nonequilibrium molecular dynamics. *Physical Review Letters*, 48(26):1818–1820, 1982.

[HM86] J. P. Hansen and I. R. McDonald. *Theory of simple liquids.* Academic Press, second edition edition, 1986.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, Aug. 1978.

[Hoo86] Wm. G. Hoover. Molecular dynamics. In *Lecture Notes in Physics*. Springer-Verlag, 1986.

[INM85] INMOS Ltd., Bristol. *Transputer Reference Manual*, september 1985.

[INM88] INMOS Ltd. *occam2 Reference Manual*, 1988.

[INM90] INMOS Ltd., Bristol. *S708 User's Guide*, january 1990.

[LL58] L. D. Landau and E. M. Lifshitz. *Statistical Physics*. Pergamon Press, 1958.

[Mil91] R. H. Miller. A horror story about integration methods. *Journal of Computational Physics*, 93:469–476, 1991.

[MRR+53] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[Nos84] S. Nosé. A molecular dynamics method for simulation in the canonical ensemble. *Molecular Physics*, 52(2):255–268, 1984.

[PH81] L. R. Pratt and S. W. Haan. Effects of periodic boundary conditions on equilibrium properties of computer simulated fluids. I. Theory. *Journal of Chemical Physics*, 74(3):1864–1872, 1981.

[PM88] D. Pountain and D May. *A tutorial introduction to occam programming*. BSP, Oxford, march 1988.

[PP89] H. G. Petersen and J. W. Perram. Molecular dynamics on transputer arrays I. Algorithm design, programming issues, timing experiments and scaling projections. *Molecular Physics*, 67(4):849–860, 1989.

[PR80] M. Parrinello and A. Rahman. Cristal structure and pair potentials: a molecular dynamics study. *Physical Review Letters*, 45(14):1196–1199, 1980.

[Rah64] A. Rahman. Correlations in the motion of atoms in liquid argon. *Physical Review*, 136(2A):A405–A411, october 1964.

[SAHW82] W. C. Swope, H. C. Andersen, Berens P. H., and K. R. Wilson. A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters. *Journal of Chemical Physics*, 76(1):637–649, 1982.

[STS77] K. Singer, A. Taylor, and J. V. L. Singer. Thermodynamic and structural properties of liquids modeled by "two Lennard-Jones centres" pair potentials. *Molecular Physics*, 33(6):1757–1795, 1977.

[SW85] F. H. Stillinger and T. A. Weber. Computer simulation of local order in condensed phases of silicon. *Physical Review B*, 31(8):5262–5271, 1985.

[Tra89] Gonzalo Travieso. Estudo de processamento paralelo para dinâmica molecular. Master's thesis, Instituto de Física e Química de São Carlos, march 1989.

[Ver67] L. Verlet. Computer experiments on classical fluids: I. Thermodynamical properties of Lennard-Jones molecules. *Physical Review*, 159(1):98–103, july 1967.

[VH87] G. D. Venneri and W. G. Hoover. Simple exact test for well-known molecular dynamics algorithms. *Journal of Computational Physics*, 73:468–477, 1987.

Apêndice A

Arquivos de configuração para a placa B008

A seguir estão listados os arquivos de configuração para 1, 2, 3 e 4 *transputers*, respectivamente.

A.1 Um processador

```
SOFTWIRE
 PIPE 0

   SLOT 1, LINK 3 TO SLOT 1, LINK 0

END
```

A.2 Dois processadores

```
SOFTWIRE
 PIPE 0

   SLOT 1, LINK 3 TO SLOT 2, LINK 0
   SLOT 2, LINK 3 TO SLOT 1, LINK 0

END
```

A.3 Três processadores

```
SOFTWIRE
 PIPE 0

   SLOT 1, LINK 3 TO SLOT 2, LINK 0
   SLOT 2, LINK 3 TO SLOT 3, LINK 0
```

```
   SLOT 3, LINK 3 TO SLOT 1, LINK 0

END
```

A.4 Quatro processadores

```
SOFTWIRE
 PIPE 0

   SLOT 1, LINK 3 TO SLOT 2, LINK 0
   SLOT 2, LINK 3 TO SLOT 3, LINK 0
   SLOT 3, LINK 3 TO SLOT 5, LINK 0
   SLOT 5, LINK 3 TO SLOT 1, LINK 0

END
```

Apêndice B

Código occam de dinâmica molecular

O código seguinte apresenta a implementação de dinâmica molecular para 2 dimensões. A implementação para 3 dimensões é similar. As implementações foram feitas separadamente de forma a permitir algumas otimizações de tempo de processamento e utilização de espaço.

```
**Listed On  04-07-93  12:34A  **
**List of Fold**       "2D (com tabela de vizinhos)"
**List all lines  with Fold Headers
**Excluding : NO LIST folds
--{{{  Bibliotecas comuns (para tabela de vizinhos)
--{{{F comuns (constantes e protocolos)
--:::F comuns06.tsr
--{{{  LIB comuns
--:::A  4 14
--{{{  Library version  09:07:49 13-Feb-93
--:::A  1 1
121
--}}}
--{{{  constantes
VAL NDp IS 4:                          -- 1 | 2 | 3 | 4
VAL Ndim IS 2:
VAL Max.grao IS 12:                    -- maximo de particulas por grao
VAL Nmax IS 12000:                     -- maximo de particulas numa DP
VAL Dois.max.INT IS (REAL32 ROUND (MOSTPOS INT))-(REAL32 ROUND (MOSTNEG INT)):
VAL Max.muda IS 1024:                -- maximo de particulas que mudam de DP
VAL Nviz IS 4:
--}}}
--{{{  Protocolos
PROTOCOL Sinais.controle
  CASE
    constantes; REAL32; REAL32; REAL32; [Ndim]REAL32; [NDp+1]INT;
                                [Ndim]INT; [Ndim]REAL32; [NDp]INT
    nova.particula; INT; [Ndim]INT; [Ndim]REAL32; [Ndim]REAL32
    redistribui.particulas
    nao.redistribui
    comunica.dados
    altera.velocidades; REAL32
    prossegue
    fim.da.simulacao
:
PROTOCOL Comunicacao.vizinhos
  CASE
    particulas.que.mudam; INT::[][2+(3*Ndim)]INT
    graos.repetidos; INT::[]INT; INT::[][Ndim]INT
    aceleracoes.repetidos; INT::[]INT; INT::[][Ndim]REAL32
:
PROTOCOL Devolve.resultados
  CASE
    dados; INT; [Ndim]INT; [Ndim]REAL32; [Ndim]REAL32
    terminaram.dados
    totalizadores; INT; REAL64; REAL64; REAL64; REAL64
:
--}}}
--}}}
--}}}F
--}}}
--{{{  PROGRAM rodando numa rede como um stand-alone
--:::A  4 11
--{{{F rodando numa rede como um stand-alone
--:::F rodand08.tsr
#USE strmhdr
#USE linkaddr
#USE comuns2Dtv
--{{{  SC Calculos
--:::A  4 10
--{{{F Calculos
--:::F calcul15.tsr
#USE comuns2Dtv
#USE snglmath
PROC calculos (CHAN OF Sinais.controle De.controle, Para.proximo,
```

```
            CHAN OF Comunicacao.vizinhos Para.direita, Para.esquerda,
            Da.direita, Da.esquerda,
            CHAN OF Devolve.resultados Para.controle, De.proximo,
            VAL INT Dp)
--{{{  constantes
VAL Dois.Real.Max.Int IS 2.0(REAL64)*((REAL64 ROUND (MOSTPOS INT))+
                                 1.0(REAL64)):
VAL Max.graos.div IS 3000:
VAL Mostneg.int32 IS REAL64 ROUND (MOSTNEG INT):
VAL Viz.por.part IS 12:
--}}}
--{{{  variaveis
BOOL redistribuiu:
[Max.graos.div][Nviz]INT16 Vizinho:
[NDp+1]INT Grao.base:
[Nmax]INT Numero.particula:
[Nmax][Ndim]INT r:
[Nmax][Ndim]REAL32 v, a:
[NDp]INT Min.r:
[Max.graos.div][Max.grao]INT Particula:
[Max.graos.div]INT Np:

[Ndim]REAL32 Box, Tam.grao:
[Ndim]INT Num.graos:
[Ndim]REAL32 Para.box:
[Ndim]REAL32 Dt.box:
[Ndim]REAL64 ITam.grao:
[Nmax]INT16 Contadores.Viz:
[Nmax*Viz.por.part]INT16 Tabela.Viz:

BOOL continua, espera.controle:
INT Num.graos.div:
INT Nint, Ntotal:
INT Num.vizinhos:
INT Numero.de.repetidos:
REAL64 Ec:
REAL32 Dt:
REAL32 Dts2.24:
INT Ncut:
REAL64 V, W:
REAL32 Rm2, Rm, Rc2, Rc:
[Ndim]REAL32 Para.sigma:
PLACE Para.sigma IN WORKSPACE:

--}}}
SEQ
  --{{{  inicializacao
  --{{{  le constantes
  Dw.controle ? CASE constantes; Dt; Rc; Rm; Box; Grao.base; Num.graos;
                        Tam.grao; Min.r
  Para.proximo ! constantes; Dt; Rc; Rm; Box; Grao.base; Num.graos;
                        Tam.grao; Min.r
  --}}}
  --{{{  calculo de constantes internas
  Num.graos.div:=Grao.base[Dp+1]-Grao.base[Dp]
  Numero.de.repetidos:=Num.graos[1]
  Para.box:=[Dois.max.INT/Box[0], Dois.max.INT/Box[1]]
  Dt.box:=[Para.box[0]*Dt, Para.box[1]*Dt]
  Para.sigma:=[Box[0]/Dois.max.INT, Box[1]/Dois.max.INT]
  Rc2:=Rc*Rc
  Rm2:=Rm*Rm
  Dts2.24:=12.0(REAL32)*Dt --24*Dt/2
  ITam.grao:=[REAL64 ROUND (INT TRUNC
      ((Dois.Real.Max.Int/(REAL64 ROUND Num.graos[0]))+0.5(REAL64))),
              REAL64 ROUND (INT TRUNC
```

```
    ((Dois.Real.Max.Int/(REAL64 ROUND Num.graos[1]))+0.5(REAL64)))]
--}}}
--{{{  inicializa vetor de graos vizinhos
VAL Primeiro.usado IS Nviz+1:
SEQ
  SEQ g=0 FOR Num.graos.div
    [Ndim]INT gd:
    SEQ
      gd:=[g/Num.graos[1], g\Num.graos[1]]
      SEQ ng=Primeiro.usado FOR Nviz
        [Ndim]INT dg, gv:
        SEQ
          dg:=[(ng/3)-1, (ng\3)-1]
          SEQ k=0 FOR Ndim
            SEQ
              gv[k]:=gd[k]+dg[k]
              --{{{  considera condicoes de contorno periodicas
              IF
                gv[k]>=Num.graos[k]
                  gv[k]:=gv[k]-Num.graos[k]
                gv[k]<0
                  gv[k]:=gv[k]+Num.graos[k]
                TRUE
                  SKIP
              --}}}
          Vizinho[g][ng-Primeiro.usado]:=INT16 ((gv[0]*Num.graos[1])+gv[1])
--}}}
--{{{  le dados iniciais sobre particulas
continua:=TRUE
Nint:=0
WHILE continua
  --{{{  variaveis locais
  INT pp:
  [Ndim]INT rp:
  [Ndim]REAL32 vp, ap:
  --}}}
  --{{{  FUNCTION interna
  BOOL FUNCTION interna (VAL []INT r)
    BOOL b:
    VALOF
      IF
        Dp<(NDp-1)
          b:=(r[0]<Min.r[Dp+1])
        TRUE
          b:=TRUE
      RESULT b
  :
  --}}}
  SEQ
    De.controle ? CASE
      nova.particula; pp; rp; vp; ap
        IF
          interna (rp)
            SEQ
              Numero.particula[Nint]:=pp
              r[Nint]:=rp
              v[Nint]:=vp
              a[Nint]:=ap
              Nint:=Nint+1
          TRUE
            Para.proximo ! nova.particula; pp; rp; vp; ap
      prossegue
        SEQ
          Para.proximo ! prossegue
          continua:=FALSE
```

```
--}}}
--}}}
continua:=TRUE
WHILE continua
  SEQ
    SEQ
      --{{{  calcula proxima posicao
      SEQ
        SEQ part=0 FOR Nint
          ri IS r[part]:
          vi IS v[part]:
          VAL ai IS a[part]:
          SEQ
            vi[0]:=vi[0]+(Dts2.24*ai[0])
            vi[1]:=vi[1]+(Dts2.24*ai[1])
            ri[0]:=ri[0] PLUS (INT ROUND (Dt.box[0]*vi[0]))
            ri[1]:=ri[1] PLUS (INT ROUND (Dt.box[1]*vi[1]))
      --}}}
      --{{{  distribui pelos graos
      De.controle ? CASE
        redistribui.particulas
          SEQ
            Para.proximo ! redistribui.particulas
            --{{{  recalcula os graos das particulas
            --{{{  constantes
            VAL Grao IS 0:
            VAL Part IS 1:
            VAL Coord IS 2:
            VAL Vel IS Coord+Ndim:
            VAL Ac IS Vel+Ndim:
            VAL Esquerda IS 0:
            VAL Direita IS 1:
            --}}}
            --{{{  variaveis
            [2]INT Np.sai:
            [2]INT Np.entra:
            [2][Max.muda][2+(3*Ndim)]INT Para.fora, De.fora:
            INT grao:
            INT g0,g1:
            --}}}
            --{{{  PROC retira
            PROC retira (VAL INT i, INT Nint, VAL INT Lado, grao)
              n IS Np.sai[Lado]:
              SEQ
                Armazem IS Para.fora[Lado][n]:
                VAL []INT v.INT RETYPES v[i]:
                VAL []INT a.INT RETYPES a[i]:
                SEQ
                  Armazem[Grao]:=grao
                  Armazem[Part]:=Numero.particula[i]
                  [Armazem FROM Coord FOR Ndim]:=r[i]
                  [Armazem FROM Vel FOR Ndim]:=v.INT
                  [Armazem FROM Ac FOR Ndim]:=a.INT
                n:=n+1
                Nint:=Nint-1
                IF
                  i<Nint
                    SEQ
                      Numero.particula[i]:=Numero.particula[Nint]
                      r[i]:=r[Nint]
                      v[i]:=v[Nint]
                      a[i]:=a[Nint]
                  i=Nint  -- ultima particula e' que saiu !!
                    SKIP
            :
```

```
--}}}
INT part:
SEQ
  redistribuiu:=TRUE
  --{{{  zera estrutura de graos
  SEQ g=0 FOR Num.graos.div
    Np[g]:=0
  --}}}
  part:=0
  Np.sai:=[0, 0]
  WHILE part<Nint
    SEQ
      --{{{  calcula novo grao
      SEQ
        g0:=(INT TRUNC (((REAL64 TRUNC r[part][0])-
                Mostneg.int32)/ITam.grao[0]))
        g1:=(INT TRUNC (((REAL64 TRUNC r[part][1])-
                Mostneg.int32)/ITam.grao[1]))
        grao:=(g0*Num.graos[1])+g1
      --}}}
      --{{{  envia para Dp correspondente
      IF
        (grao>=Grao.base[Dp]) AND (grao<Grao.base[Dp+1])
          --{{{  guarda no grao interno correspondente
          INT g:
          SEQ
            g:=grao-Grao.base[Dp]
            Particula[g][Np[g]]:=part
            Np[g]:=Np[g]+1
            part:=part+1
          --}}}
        grao < Grao.base[Dp]
          IF
            (Dp<>(NDp-1)) OR ((Dp=(NDp-1)) AND
                                (grao>=Grao.base[1]))
              retira (part, Nint, Esquerda, grao)
            (Dp=(NDp-1)) AND (grao<Grao.base[1])
              retira (part, Nint, Direita, grao)
        grao >= Grao.base[Dp+1]
          IF
            (Dp<>0) OR ((Dp=0) AND (grao<Grao.base[NDp-1]))
              retira (part, Nint, Direita, grao)
            (Dp=0) AND (grao>=Grao.base[NDp-1])
              retira (part, Nint, Esquerda, grao)
      --}}}
    PAR
      --{{{  envia dados para vizinhos
      Para.esquerda ! particulas.que.mudam;
                    Np.sai[Esquerda]::Para.fora[Esquerda]
      Para.direita ! particulas.que.mudam;
                    Np.sai[Direita]::Para.fora[Direita]
      --}}}
      --{{{  le de vizinhos
      Da.esquerda ? CASE particulas.que.mudam;
                    Np.entra[Esquerda]::De.fora[Esquerda]
      Da.direita ? CASE particulas.que.mudam;
                    Np.entra[Direita]::De.fora[Direita]
      --}}}
    --{{{  coloca lidos nos graos
    SEQ lado=0 FOR 2
      SEQ i=0 FOR Np.entra[lado]
        VAL Armazem IS De.fora[lado][i]:
        VAL []REAL32 ve RETYPES [Armazem FROM Vel FOR Ndim]:
        VAL []REAL32 ae RETYPES [Armazem FROM Ac FOR Ndim]:
        INT grao:
```

```
            SEQ
              Numero.particula[Nint]:=Armazem[Part]
              r[Nint]:=[Armazem FROM Coord FOR Ndim]
              v[Nint]:=ve
              a[Nint]:=ae
              grao:=Armazem[Grao]-Grao.base[Dp]
              Particula[grao][Np[grao]]:=Nint
              Np[grao]:=Np[grao]+1
              Nint:=Nint+1
        --}}}
      --}}}
  nao.redistribui
    Para.proximo ! nao.redistribui
--}}}
--{{{  recebe e envia graos repetidos
VAL Max.repetidos IS 50:
VAL Primeiro.a.enviar IS 0:
VAL Primeiro.repetido IS Num.graos.div:
[Max.repetidos]INT Np.grao.entra, Np.grao.sai:
[Max.repetidos*Max.grao][Ndim]INT r.entra, r.sai:
INT ne:
SEQ
  --{{{  forma lista de particulas repetidas a enviar
  SEQ
    ne:=0
    SEQ grao=Primeiro.a.enviar FOR Numero.de.repetidos
      VAL g IS grao-Primeiro.a.enviar:
      SEQ
        Np.grao.sai[g]:=Np[grao]
        SEQ j=0 FOR Np[grao]
          SEQ
            r.sai[ne]:=r[Particula[grao][j]]
            ne:=ne+1
  --}}}
  --{{{  envia para esquerda e recebe da direita
  PAR
    Para.esquerda ! graos.repetidos;
       Numero.de.repetidos::Np.grao.sai; ne::r.sai
    INT nr, n:
    Da.direita ? CASE graos.repetidos; nr::Np.grao.entra; n::r.entra
  --}}}
  --{{{  organiza as recebidas na estrutura interna
  INT i:
  SEQ
    Ntotal:=Nint
    i:=0
    SEQ grao=Primeiro.repetido FOR Numero.de.repetidos
      VAL g IS grao-Primeiro.repetido:
      SEQ
        Np[grao]:=Np.grao.entra[g]
        SEQ j=0 FOR Np.grao.entra[g]
          SEQ
            r[Ntotal]:=r.entra[i]
            Particula[grao][j]:=Ntotal
            Ntotal:=Ntotal+1
            i:=i+1
  --}}}
--}}}
--{{{  forma tabela de vizinhos
IF
  redistribuiu=TRUE
    SEQ
      --{{{  atualiza
      INT ic, iv:
      [(Nviz+1)*Max.grao]INT Juntas:
```

```
SEQ
  ic:=0
  iv:=0
  SEQ g=0 FOR Num.graos.div
    INT n, ncentral:
    SEQ
      --{{{  forma lista de particulas dos graos que interagem
      n:=0
      SEQ i=0 FOR Np[g]
        SEQ
          Juntas[n]:=Particula[g][i]
          n:=n+1
      ncentral:=n
      SEQ v=0 FOR Nviz
        VAL viz IS INT Vizinho[g][v]:
        SEQ i=0 FOR Np[viz]
          SEQ
            Juntas[n]:=Particula[viz][i]
            n:=n+1
      --}}}
      --{{{  busca pares nessa lista
      SEQ central=0 FOR ncentral
        VAL i IS Juntas[central]:
        VAL ri IS r[i]:
        SEQ
          Contadores.Viz[ic]:=i(INT16)
          Tabela.Viz[iv]:=INT16 i
          iv:=iv+1
          SEQ periferica=central+1 FOR n-(central+1)
            VAL j IS Juntas[periferica]:
            VAL rj IS r[j]:
            --{{{  verifica par ij
            --{{{  variaveis temporarias
            [Ndim]REAL32 dij:
            PLACE dij IN WORKSPACE:
            REAL32 rij2:
            --}}}
            SEQ
              dij[0]:=Para.sigma[0]*(REAL32 ROUND
                        (ri[0] MINUS rj[0]))
              IF
                ABS(dij[0])>Rm
                  SKIP
                TRUE
                  SEQ
                    dij[1]:=Para.sigma[1]*(REAL32 ROUND
                        (ri[1] MINUS rj[1]))
                    IF
                      ABS(dij[1])>Rm
                        SKIP
                      TRUE
                        SEQ
                          rij2:=(dij[0]*dij[0])+(dij[1]*dij[1])
                          IF
                            rij2 < Rm2
                              --{{{  guarda j na tabela de i
                              SEQ
                                Tabela.Viz[iv]:=INT16 j
                                iv:=iv+1
                                Contadores.Viz[ic]:=
                                    Contadores.Viz[ic]+1(INT16)
                              --}}}
                            TRUE
                              SKIP
            --}}}
```

```
                ic:=ic+1
            --}}}
        --}}}
        redistribuiu:=FALSE
    TRUE
      SKIP
--}}}
--{{{  calcula interacoes
INT it, i, j:
SEQ
  --{{{  inicializa totalizadores
  V:=0.0(REAL64)
  W:=0.0(REAL64)
  Ncut:=0
  []REAL32 ajunto RETYPES a:
  SEQ i=0 FOR Ntotal*Ndim
    ajunto[i]:=0.0(REAL32)
  --}}}
  --{{{  calcula interacoes internas
  it:=0
  SEQ cent=0 FOR Nint
    SEQ
      i:=INT Tabela.Viz[it]
      it:=it+1
      VAL ri IS r[i]:
      ai IS a[i]:
      SEQ viz=0 FOR (INT Contadores.Viz[cent])-1
        SEQ
          j:=INT Tabela.Viz[it]
          it:=it+1
          VAL rj IS r[j]:
          aj IS a[j]:
          --{{{  interage particulas i e j
          --{{{  variaveis temporarias
          [Ndim]REAL32 dij:
          PLACE dij IN WORKSPACE:
          REAL32 rij2:
          --}}}
          SEQ
            dij[0]:=Para.sigma[0]*(REAL32 ROUND (ri[0] MINUS rj[0]))
            dij[1]:=Para.sigma[1]*(REAL32 ROUND (ri[1] MINUS rj[1]))
            rij2:=(dij[0]*dij[0])+(dij[1]*dij[1])
            IF
              rij2 < Rc2
                --{{{  calcula valores da interacao e acumula
                REAL32 acij, aij, wij, vij, sr12, sr6, sr2:
                SEQ
                  Ncut:=Ncut+1
                  sr2:=1.0(REAL32)/rij2
                  sr6:=sr2*(sr2*sr2)
                  sr12:=sr6*sr6
                  vij:=sr12-sr6
                  V:=V+(REAL64 vij)
                  wij:=vij+sr12
                  W:=W+(REAL64 wij)
                  acij:=wij*sr2
                  aij:=acij*dij[0]
                  ai[0]:=ai[0]+aij
                  aj[0]:=aj[0]-aij
                  aij:=acij*dij[1]
                  ai[1]:=ai[1]+aij
                  aj[1]:=aj[1]-aij
                --}}}
              TRUE
                SKIP
```

```
            --}}}
      --}}}
      --{{{  comunica dados de colunas repetidas
      VAL Max.repetidos IS 50:
      VAL Primeiro.a.enviar IS Num.graos.div:
      VAL Primeiro.a.receber IS 0:
      [Max.repetidos]INT Np.grao.entra, Np.grao.sai:
      [Max.repetidos*Max.grao][Ndim]REAL32 a.entra, a.sai:
      INT ne:
      SEQ
        --{{{  forma lista de particulas repetidas a enviar
        SEQ
          ne:=0
          SEQ grao=Primeiro.a.enviar FOR Numero.de.repetidos
            VAL g IS grao-Primeiro.a.enviar:
            SEQ
              Np.grao.sai[g]:=Np[grao]
              SEQ j=0 FOR Np[grao]
                SEQ
                  a.sai[ne]:=a[Particula[grao][j]]
                  ne:=ne+1
        --}}}
        --{{{  envia para direita e recebe da esquerda
        PAR
          Para.direita ! aceleracoes.repetidos;
                    Numero.de.repetidos::Np.grao.sai; ne::a.sai
          INT nr, n:
          Da.esquerda ? CASE aceleracoes.repetidos;
                    nr::Np.grao.entra; n::a.entra
        --}}}
        --{{{  organiza as recebidas na estrutura interna
        INT i:
        SEQ
          i:=0
          SEQ grao=Primeiro.a.receber FOR Numero.de.repetidos
            VAL g IS grao-Primeiro.a.receber:
            SEQ j=0 FOR Np.grao.entra[g]
              SEQ
                SEQ k=0 FOR Ndim
                  a[Particula[grao][j]][k]:=a[Particula[grao][j]][k]+
                                                a.entra[i][k]
                i:=i+1
        --}}}
      --}}}
    --}}}
    --{{{  termina movimento
    SEQ
      Ec:=0.0(REAL64)
      SEQ i=0 FOR Nint
        vi IS v[i]:
        VAL ai IS a[i]:
        SEQ
          vi[0]:=vi[0]+(Dts2.24*ai[0])
          vi[1]:=vi[1]+(Dts2.24*ai[1])
          Ec:=Ec+(REAL64 ((vi[0]*vi[0])+(vi[1]*vi[1])))
    --}}}
  --{{{  comunica-se com Controle
  REAL64 Vc, Ecp, Vp, Vcp, Wp:
  INT Ncutp:
  SEQ
    --{{{  calcula energia potencial sem descontinuidade
    REAL32 sc2, sc6, sc12, vij:
    SEQ
      ------------------------------------------------------------------
      ---------------  Encontra en. pot. livre de descontinuidades  ---
```

```
  sc2:=1.0(REAL32)/Rc2          -- Calcula energia de um par a Rc
  sc6:=(sc2*sc2)*sc2            --
  sc12:=sc6*sc6                 --
  vij:=sc12-sc6                 --
  Vc:=V-(REAL64 ((REAL32 ROUND Ncut)*vij))
                                -- Desconta de V para todos os
                                -- interagentes
  ---------------------------------------------------------------
--}}}
--{{{  multiplica por constantes
Vc:=Vc*4.0(REAL64)
V:=V*4.0(REAL64)
Ec:=Ec*0.5(REAL64)
W:=W*8.0(REAL64)
--}}}
--{{{  espera totalizadores e envia para controle
De.proximo ? CASE totalizadores; Ncutp; Ecp; Vp; Vcp; Wp
Para.controle ! totalizadores; Ncutp+Ncut; Ecp+Ec; Vp+V; Vcp+Vc; Wp+W
--}}}
espera.controle:=TRUE
WHILE espera.controle
  REAL32 fat.vel:
  De.controle ? CASE
    prossegue
      --{{{  avisa prosseguimento e termina loop
      SEQ
        Para.proximo ! prossegue
        espera.controle:=FALSE
      --}}}
    comunica.dados
      --{{{ envia dados das particulas internas e serve como passagem
      BOOL tem.dados:
      SEQ
        Para.proximo ! comunica.dados
        SEQ i=0 FOR Nint
          Para.controle ! dados; Numero.particula[i];
                                 r[i]; v[i]; a[i]
        tem.dados:=TRUE
        WHILE tem.dados
          --{{{  variaveis auxiliares
          INT np:
          [Ndim]INT rp:
          [Ndim]REAL32 vp, ap:
          --}}}
          De.proximo ? CASE
            dados; np; rp; vp; ap
              Para.controle ! dados; np; rp; vp; ap
            terminaram.dados
              SEQ
                Para.controle ! terminaram.dados
                tem.dados:=FALSE
      --}}}
    altera.velocidades; fat.vel
      --{{{  altera o valor de todas as velocidades por fat.vel
      SEQ
        Para.proximo ! altera.velocidades; fat.vel
        SEQ i=0 FOR Nint
          SEQ k=0 FOR Ndim
            v[i][k]:=v[i][k]*fat.vel
      --}}}
    constantes; Dt; Rc; Rm; Box; Grao.base;
                                        Num.graos; Tam.grao; Min.r
      --{{{
      SEQ
        Para.proximo ! constantes; Dt; Rc; Rm; Box; Grao.base;
```

```
                                                Num.graos; Tam.grao; Min.r
                --{{{  recalcula constantes internas
                Num.graos.div:=Grao.base[Dp+1]-Grao.base[Dp]
                Numero.de.repetidos:=Num.graos[1]
                Para.box:=[Dois.max.INT/Box[0], Dois.max.INT/Box[1]]
                Dt.box:=[Para.box[0]*Dt, Para.box[1]*Dt]
                Para.sigma:=[Box[0]/Dois.max.INT, Box[1]/Dois.max.INT]
                Rc2:=Rc*Rc
                Rm2:=Rm*Rm
                Dts2.24:=12.0(REAL32)*Dt --24*Dt/2
                ITam.grao:=[REAL64 ROUND (INT64 TRUNC
                        ((Dois.Real.Max.Int/(REAL64 ROUND
                                Num.graos[0]))+0.5(REAL64))),
                             REAL64 ROUND (INT64 TRUNC
                        ((Dois.Real.Max.Int/(REAL64 ROUND
                                Num.graos[1]))+0.5(REAL64)))]
                --}}}
              --}}}
            fim.da.simulacao
              --{{{  sinaliza final
              SEQ
                Para.proximo ! fim.da.simulacao
                espera.controle:=FALSE
                continua:=FALSE
              --}}}
        --}}}
:
--}}}F
--...F code HT8
--:::A  1 2
--:::F calcul15.dcd
--...F descriptor
--:::A  1 4
--:::F calcul15.dds
--...F link
--:::A  1 9
--:::F calcul15.dlk
--...F debug
--:::A  1 5
--:::F calcul15.ddb
--...F CODE SC Calculos 09:08:01 13-Feb-93
--:::A  2 10
--:::F calcul15.csc
--}}}
--{{{  SC Vazio
--:::A  3 10
--{{{F Vazio
--:::F vazio010.tsr
#USE comuns2Dtv
PROC vazio (CHAN OF Sinais.controle a.consumir,
            CHAN OF Devolve.resultados a.enviar)
  --{{{  variaveis
  REAL32 lixo1, lixo2, lixo3:
  [Ndim]REAL32 vlixo1, vlixo2:
  [NDp+1]INT vilixo1:
  [Ndim]INT vilixo2:
  [NDp]INT vilixo3:
  INT ilixo:
  BOOL continua:
  --}}}
  SEQ
    continua:=TRUE
    WHILE continua
      a.consumir ? CASE
        constantes; lixo1; lixo2; lixo3; vlixo1; vilixo1;
```

```
                                    vilixo2; vlixo2; vilixo3
          SKIP
        nova.particula; ilixo; vilixo2; vlixo1; vlixo2
          STOP
        redistribui.particulas
          a.enviar ! totalizadores; 0; 0.0(REAL64); 0.0(REAL64);
                                          0.0(REAL64); 0.0(REAL64)
        nao.redistribui
          a.enviar ! totalizadores; 0; 0.0(REAL64); 0.0(REAL64);
                                          0.0(REAL64); 0.0(REAL64)
        comunica.dados
          a.enviar ! terminaram.dados
        altera.velocidades; lixo1
          SKIP
        prossegue
          SKIP
        fim.da.simulacao
          continua:=FALSE
:
--}}}F
--...F code HT8
--:::A  1 2
--:::F vazio010.dcd
--...F descriptor
--:::A  1 4
--:::F vazio010.dds
--...F debug
--:::A  1 5
--:::F vazio010.ddb
--...F CODE SC Vazio 09:08:08 13-Feb-93
--:::A  2 10
--:::F vazio010.csc
--}}}
--{{{  SC Novo controle (com exit!)
--:::A  4 10
--{{{F Controle
--:::F control4.tsr
--{{{  bibliotecas utilizadas
#USE comuns2Dtv
#USE strmhdr
#USE sphdr
#USE solib
#USE sklib
#USE splib
#USE snglmath
#USE dblmath
#USE spinterf
#USE streamio
#USE strings
--}}}
PROC controle (CHAN OF SP from.isv, to.isv,
               CHAN OF Sinais.controle Para.calculo,
               CHAN OF Devolve.resultados De.calculo)
  --{{{  variaveis e canais locais
  CHAN OF KS keyboard:
  CHAN OF SS screen:
  [3]CHAN OF SP fs, ts:
  CHAN OF BOOL parar.mult:
  [20]BYTE arquivo.cnt, arquivo.ini, arquivo.tot, arquivo.sav,
          arquivo.sto, arquivo.tmp, arquivo.nnn:
  INT tam.cnt, tam.ini, tam.tot, tam.sav, tam.sto, tam.tmp, tam.nnn:
  --}}}
  SEQ
    --{{{  leitura do nome dos arquivos
    --{{{  PROC junta.extensao
```

```
PROC junta.extensao ([]BYTE com.extensao, VAL []BYTE sem.extensao,
                     VAL []BYTE extensao, VAL INT posicao.ponto,
                     INT tamanho)
  SEQ
    tamanho:=0
    append.text (tamanho, com.extensao, sem.extensao)
    tamanho:=posicao.ponto
    append.text (tamanho, com.extensao, extensao)
:
--}}}
--{{{  variaveis
BYTE resultado:
BOOL existe:
INT posicao.ponto:
[1]BOOL opcao.existe:
[1][2]INT parametros.opcao:
INT erro.comp:
[100]BYTE linha:
--}}}
SEQ
  --{{{  pega nome do arquivo da linha de comando
  tam.cnt:=0
  so.parse.command.line (from.isv, to.isv, [" "], [spopt.always],
                         opcao.existe, parametros.opcao, erro.comp, linha)
  IF
    erro.comp>0
      SEQ
        so.write.string (from.isv, to.isv, [linha FROM 0 FOR erro.comp])
        so.exit (from.isv, to.isv, sps.failure)
    TRUE
      SKIP
  append.text (tam.cnt, arquivo.cnt,
      [linha FROM parametros.opcao[0][0] FOR parametros.opcao[0][1]])
  --}}}
  --{{{  acrescenta extensao se necessario
  posicao.ponto:=char.pos ('.', [arquivo.cnt FROM 0 FOR tam.cnt])
  IF
    posicao.ponto=(-1)
      SEQ
        posicao.ponto:=tam.cnt
        append.text (tam.cnt, arquivo.cnt, ".cnt")
    TRUE
      SKIP
  --}}}
  --{{{  verifica a existencia
  so.test.exists (from.isv, to.isv, [arquivo.cnt FROM 0 FOR tam.cnt],
                  existe)
  IF
    (NOT existe)
      SEQ
        so.write.string (from.isv, to.isv,
                         ">>>Arquivo inexistente. Forneca outro nome.")
        so.exit (from.isv, to.isv, sps.failure)
    TRUE
      SKIP
  --}}}
  so.write.nl (from.isv, to.isv)
  VAL nome.dado IS [arquivo.cnt FROM 0 FOR tam.cnt]:
  SEQ
    junta.extensao (arquivo.tot, nome.dado, ".tot", posicao.ponto, tam.tot)
    junta.extensao (arquivo.ini, nome.dado, ".ini", posicao.ponto, tam.ini)
    junta.extensao (arquivo.sav, nome.dado, ".sav", posicao.ponto, tam.sav)
    junta.extensao (arquivo.sto, nome.dado, ".sto", posicao.ponto, tam.sto)
    junta.extensao (arquivo.tmp, nome.dado, ".tmp", posicao.ponto, tam.tmp)
    junta.extensao (arquivo.nnn, nome.dado, ".$$$", posicao.ponto, tam.nnn)
```

```
--}}}
PAR
  --{{{  entrada provindo de arquivo
  CHAN OF SP from.isv IS fs[0]:
  CHAN OF SP to.isv IS ts[0]:
  BYTE resultado:
  SEQ
    so.keystream.from.file (from.isv, to.isv, keyboard,
                            [arquivo.cnt FROM 0 FOR tam.cnt], resultado)
  --}}}
  --{{{  controle
  CHAN OF SP from.isv IS fs[1]:
  CHAN OF SP to.isv IS ts[1]:
  INT32 saida:
  BYTE resultado:
  --{{{  entrada e saida
  -------------- Rotinas de escrita -------------
  --{{{  escreve cadeia
  PROC wr.str (VAL []BYTE string)
    SEQ
      so.fwrite.string (from.isv, to.isv, saida, string, resultado)
      --{{{
      IF
        resultado<>spr.ok
          SEQ
            so.write.string (from.isv, to.isv,
                     "*c*n>> Erro no arquivo de saida. Erro=")
            so.write.int (from.isv, to.isv, (INT resultado), 0)
            STOP
        TRUE
          SKIP
      --}}}
  :
  --}}}
  --{{{  escreve INT
  PROC wr.int (VAL INT value, width)
    SEQ
      so.fwrite.int (from.isv, to.isv, saida, value, width, resultado)
      --{{{
      IF
        resultado<>spr.ok
          SEQ
            so.write.string (from.isv, to.isv,
                     "*c*n>> Erro no arquivo de saida. Erro=")
            so.write.int (from.isv, to.isv, (INT resultado), 0)
            STOP
        TRUE
          SKIP
      --}}}
  :
  --}}}
  --{{{  escreve REAL32
  PROC wr.r32 (VAL REAL32 value, VAL INT before, after)
    SEQ
      so.fwrite.real32 (from.isv, to.isv, saida, value, before,
                        after, resultado)
      --{{{
      IF
        resultado<>spr.ok
          SEQ
            so.write.string (from.isv, to.isv,
                             "*c*n>> Erro no arquivo de saida. Erro=")
            so.write.int (from.isv, to.isv, (INT resultado), 0)
            STOP
        TRUE
```

```
        SKIP
    --}}}
:
--}}}
--{{{  escreve REAL64
PROC wr.r64 (VAL REAL64 value, VAL INT before, after)
  SEQ
    so.fwrite.real64 (from.isv, to.isv, saida, value, before,
                      after, resultado)
    --{{{
    IF
      resultado<>spr.ok
        SEQ
          so.write.string (from.isv, to.isv,
                           "*c*n>> Erro no arquivo de saida. Erro=")
          so.write.int (from.isv, to.isv, (INT resultado), 0)
          STOP
      TRUE
        SKIP
    --}}}
:
--}}}
--{{{  envia resultado INT (com mensagem indicativa)
PROC type.int (VAL []BYTE name, VAL INT result, width)
  SEQ
    wr.str (name)
    wr.int (result, width)
:
--}}}
--{{{  envia resultado REAL32 (com mensagem indicativa)
PROC type.r32 (VAL []BYTE name, VAL REAL32 result, VAL INT before, after)
  SEQ
    wr.str (name)
    wr.r32 (result, before, after)
:
--}}}
--{{{  envia resultado REAL64 (com mensagem indicativa)
PROC type.r64 (VAL []BYTE name, VAL REAL64 result, VAL INT before, after)
  SEQ
    wr.str (name)
    wr.r64 (result, before, after)
:
--}}}
--{{{  muda de linha
PROC wr.nl ()
  so.fwrite.nl (from.isv, to.isv, saida, resultado)
:
--}}}
-----------------------------------------------
-------------- Rotinas de leitura -------------
--{{{  le INT
PROC rd.int (INT value, INT c)
  SEQ
    ks.read.int (keyboard, value, c)
    IF
      c=ft.number.error
        SEQ
          so.write.string (from.isv, to.isv,
                           "*c*n>>>Erro na leitura de dados")
          STOP
      TRUE
        SKIP
:
--}}}
--{{{  le REAL32
```

```
PROC rd.r32 (REAL32 value, INT c)
  SEQ
    ks.read.real32 (keyboard, value, c)
    IF
      c=ft.number.error
        SEQ
          so.write.string (from.isv, to.isv,
                           "*c*n>>>Erro na leitura de dados")
          STOP
      TRUE
        SKIP
:
--}}}
--{{{  pede nova entrada
PROC request.retype ()
  wr.str (">> Erro durante a leitura. Digite novamente: ")
:
--}}}
--{{{  le INT (com mensagem de pedido)
PROC expect.int (VAL []BYTE prompt, INT value, INT c)
  SEQ
    wr.str (prompt)
    rd.int (value, c)
    WHILE (c=ft.number.error) OR (c<>(INT'*c'))
      SEQ
        request.retype ()
        ks.read.echo.char (keyboard, screen, c)
        rd.int (value, c)
:
--}}}
--{{{  le REAL32 (com mensagem de pedido)
PROC expect.r32 (VAL []BYTE prompt, REAL32 value, INT c)
  SEQ
    wr.str (prompt)
    rd.r32 (value, c)
    WHILE (c=ft.number.error) OR (c<>(INT'*c'))
      SEQ
        request.retype ()
        ks.read.echo.char (keyboard, screen, c)
        rd.r32 (value, c)
:
--}}}
-----------------------------------------------
--}}}
--{{{  constantes
VAL Tamanho.de.uma IS (4*Ndim)*3:  -- r, v e a para uma particula
VAL seg.tick IS 64.0E-6(REAL32):
VAL min.tick IS seg.tick/60.0(REAL32):
VAL livre IS (REAL64 ROUND Ndim):
--}}}
--{{{  variaveis e canais locais
BOOL criado.temp:
INT32 temporario, armazena:
INT quadro.inicial:
INT Intervalo.armazenamento, Intervalo.salvamento:
INT Intervalo.totais, Intervalo.redistribuicao, flag.equilibracao:
TIMER tempo:
INT t1, t2, t3, t4:
REAL32 Ac.t, Ac.t2:
REAL64 Ac.E, Ac.V, Ac.Ec, Ac.P, Ac.Eo:
REAL64 Ac.E2, Ac.V2, Ac.Ec2, Ac.P2, Ac.Eo2:
REAL32 Dt, Rc, Rm:
INT Num.quadros:
INT Num.Part:
[Ndim]REAL32 Box:
```

```
[Ndim]INT Num.graos:
[Ndim]REAL32 Tam.grao:
[NDp]INT Min.r:
[NDp+1]INT Primeiro.grao:
--}}}
SEQ
  --{{{  abre arquivo de saida
  so.open (from.isv, to.isv, [arquivo.tot FROM 0 FOR tam.tot], spt.text,
           spm.output, saida, resultado)
  --}}}
  --{{{  le constantes e envia para calculo
  INT c, Nd:
  SEQ
    --{{{  le e testa numero de dimensoes
    c:=INT'*c'
    rd.int (Nd, c)
    IF
      Nd<>Ndim
        SEQ
          so.write.string (from.isv, to.isv, "*c*n>>Erro no arquivo de
                                dados: numero incompativel de dimensoes")
          STOP
      TRUE
        SKIP
    --}}}
    --{{{  le e envia constantes de calculos
    rd.int (Num.quadros, c)
    rd.r32 (Dt, c)
    rd.r32 (Rc, c)
    rd.r32 (Rm, c)
    rd.int (Num.Part, c)
    SEQ k=0 FOR Ndim
      rd.r32 (Box[k], c)
    SEQ k=0 FOR Ndim
      rd.int (Num.graos[k],c)
    SEQ k=0 FOR Ndim
      rd.r32 (Tam.grao[k], c)
    SEQ n=0 FOR NDp+1
      rd.int (Primeiro.grao[n],c)
    SEQ n=0 FOR NDp
      rd.int (Min.r[n],c)
    --{{{  envio das constantes
    Para.calculo ! constantes; Dt; Rc; Rm; Box; Primeiro.grao;
                   Num.graos; Tam.grao; Min.r
    --}}}
    --}}}
    --{{{  le constantes de controle
    rd.int (Intervalo.armazenamento, c)
    rd.int (Intervalo.salvamento, c)
    rd.int (Intervalo.totais, c)
    rd.int (Intervalo.redistribuicao, c)
    rd.int (flag.equilibracao, c)
    --}}}
  --}}}
  --{{{  testa se existe salvamento para esta simulacao
  BOOL existe.sav, existe.nnn:
  SEQ
    so.test.exists (from.isv, to.isv, [arquivo.sav FROM 0 FOR tam.sav],
                    existe.sav)
    so.test.exists (from.isv, to.isv, [arquivo.nnn FROM 0 FOR tam.nnn],
                    existe.nnn)
    IF
      (Intervalo.salvamento<>0) AND (existe.sav OR existe.nnn)
        INT32 salva:
        SEQ
```

```
so.write.string (from.isv, to.isv,
                 "*c*nRetomando simulacao interrompida!")
--{{{  abre arquivo de salvamento
IF
  existe.sav   -- se existe arquivo.sav, usa como salvamento
    so.open (from.isv, to.isv,
             [arquivo.sav FROM 0 FOR tam.sav], spt.binary,
             spm.input, salva, resultado)
  existe.nnn   -- se existe arquivo.nnn,
               -- entao arquivo.tmp e' um salvamento
               -- mais novo
    so.open (from.isv, to.isv,
             [arquivo.tmp FROM 0 FOR tam.tmp], spt.binary,
             spm.input, salva, resultado)
IF
  resultado<>spr.ok
    SEQ
      so.write.string (from.isv, to.isv,
         "*c*n>>>Erro na leitura do arquivo de salvamento")
      STOP
  TRUE
    SKIP
--}}}
--{{{  le r, v e a de cada particula e envia para calculos
SEQ i=0 FOR Num.Part
  [4*Ndim]BYTE rb, vb, ab:
  INT lidos1, lidos2, lidos3:
  SEQ
    so.read (from.isv, to.isv, salva, lidos1, rb)
    so.read (from.isv, to.isv, salva, lidos2, vb)
    so.read (from.isv, to.isv, salva, lidos3, ab)
    IF
      ((lidos1+lidos2)+lidos3)<>(12*Ndim)
        SEQ
          so.write.string (from.isv, to.isv,
                 "*c*n>>Arquivo de salvamento inconsistente")
          STOP
      TRUE
        SKIP
    VAL []INT r RETYPES rb:
    VAL []REAL32 v RETYPES vb:
    VAL []REAL32 a RETYPES ab:
    Para.calculo ! nova.particula; i; r; v; a
--}}}
--{{{  inicializa totais e quadro inicial com arq. de acumul.
INT tam.lido:
--{{{  redefinicao dos acumuladores
[]BYTE q.aux RETYPES quadro.inicial:
[]BYTE E.aux RETYPES Ac.E:
[]BYTE Eo.aux RETYPES Ac.Eo:
[]BYTE Ec.aux RETYPES Ac.Ec:
[]BYTE V.aux RETYPES Ac.V:
[]BYTE P.aux RETYPES Ac.P:
[]BYTE E2.aux RETYPES Ac.E2:
[]BYTE Eo2.aux RETYPES Ac.Eo2:
[]BYTE Ec2.aux RETYPES Ac.Ec2:
[]BYTE V2.aux RETYPES Ac.V2:
[]BYTE P2.aux RETYPES Ac.P2:
--}}}
SEQ
  --{{{  le numero do quadro
  so.read (from.isv, to.isv, salva, tam.lido, q.aux)
  --}}}
  --{{{  le cada um dos acumuladores
  so.read (from.isv, to.isv, salva, tam.lido, E.aux)
```

```
        so.read (from.isv, to.isv, salva, tam.lido, Eo.aux)
        so.read (from.isv, to.isv, salva, tam.lido, Ec.aux)
        so.read (from.isv, to.isv, salva, tam.lido, V.aux)
        so.read (from.isv, to.isv, salva, tam.lido, P.aux)
        so.read (from.isv, to.isv, salva, tam.lido, E2.aux)
        so.read (from.isv, to.isv, salva, tam.lido, Eo2.aux)
        so.read (from.isv, to.isv, salva, tam.lido, Ec2.aux)
        so.read (from.isv, to.isv, salva, tam.lido, V2.aux)
        so.read (from.isv, to.isv, salva, tam.lido, P2.aux)
        --}}}
      quadro.inicial:=quadro.inicial+1
      --}}}
      --{{{  fecha arquivo de salvamento
      so.close (from.isv, to.isv, salva, resultado)
      IF
        resultado<>spr.ok
          SEQ
            so.write.string (from.isv, to.isv,
               "*c*n>>>Erro ao fechar arquivo de salvamento")
            STOP
        TRUE
          SKIP
      --}}}
      IF
        Intervalo.armazenamento<>0
          --{{{ abre arq. de armaz. e posiciona no local correto.
          VAL armazenamentos.feitos IS
                      (quadro.inicial-1)/Intervalo.armazenamento:
          VAL posicao IS INT32
               ((Tamanho.de.uma*Num.Part)*armazenamentos.feitos):
          SEQ
            so.open (from.isv, to.isv,
               [arquivo.sto FROM 0 FOR tam.sto], spt.binary,
               spm.existing.update, armazena, resultado)
            IF
              resultado<>spr.ok
                SEQ
                  so.write.string (from.isv, to.isv,
                       "*c*n>>> Erro no arquivo de armazenamento")
                  STOP
              TRUE
                SKIP
            so.seek (from.isv, to.isv, armazena,
                       posicao, spo.start, resultado)
            IF
              resultado<>spr.ok
                SEQ
                  so.write.string (from.isv, to.isv,
                    "*c*n>>> Inconsistencia do arquivo de armazenamento")
                  STOP
              TRUE
                SKIP
          --}}}
        TRUE
          SKIP
TRUE
  SEQ
    so.write.string (from.isv, to.isv, "*c*nIniciando simulacao")
    --{{{  inicializa totalizadores e quadro inicial
    SEQ
      quadro.inicial:=1
      Ac.E:=0.0(REAL64)
      Ac.Eo:=0.0(REAL64)
      Ac.Ec:=0.0(REAL64)
      Ac.V:=0.0(REAL64)
```

```
  Ac.P:=0.0(REAL64)
  Ac.E2:=0.0(REAL64)
  Ac.Eo2:=0.0(REAL64)
  Ac.Ec2:=0.0(REAL64)
  Ac.V2:=0.0(REAL64)
  Ac.P2:=0.0(REAL64)
--}}}
--{{{ le posicoes, velocidades e aceleracoes do arquivo inicial
INT32 iniciais:
SEQ
  --{{{  abre arquivo de valores iniciais
  so.open (from.isv, to.isv, [arquivo.ini FROM 0 FOR tam.ini],
           spt.binary, spm.input, iniciais, resultado)
  IF
    resultado<>spr.ok
      SEQ
        so.write.string (from.isv, to.isv,
          "*c*n>>>Erro na leitura do arquivo de valores iniciais")
        STOP
    TRUE
      SKIP
  --}}}
  --{{{  le r, v e a de cada particula e envia para calculos
  SEQ i=0 FOR Num.Part
    [4*Ndim]BYTE rb, vb, ab:
    INT lidos1, lidos2, lidos3:
    SEQ
      so.read (from.isv, to.isv, iniciais, lidos1, rb)
      so.read (from.isv, to.isv, iniciais, lidos2, vb)
      so.read (from.isv, to.isv, iniciais, lidos3, ab)
      IF
        ((lidos1+lidos2)+lidos3)<>(12*Ndim)
          SEQ
            so.write.string (from.isv, to.isv,
             "*c*n>>Arquivo de valores iniciais inconsistente")
            STOP
        TRUE
          SKIP
      VAL []INT r RETYPES rb:
      VAL []REAL32 v RETYPES vb:
      VAL []REAL32 a RETYPES ab:
      Para.calculo ! nova.particula; i; r; v; a
  --}}}
  --{{{  fecha arquivo de valores iniciais
  so.close (from.isv, to.isv, iniciais, resultado)
  IF
    resultado<>spr.ok
      SEQ
        so.write.string (from.isv, to.isv,
          "*c*n>>>Erro ao fechar arquivo de valores iniciais")
        STOP
    TRUE
      SKIP
  --}}}
--}}}
IF
  Intervalo.armazenamento<>0
    --{{{  abre novo arquivo de armazenamento
    SEQ
      so.open (from.isv, to.isv,
               [arquivo.sto FROM 0 FOR tam.sto], spt.binary,
               spm.output, armazena, resultado)
      IF
        resultado<>spr.ok
          SEQ
```

```
                    so.write.string (from.isv, to.isv,
                        "*c*n>>> Erro no arquivo de armazenamento")
                    STOP
                TRUE
                  SKIP
            --}}}
          TRUE
            SKIP
--}}}
--{{{  cria arquivo temporario para dados
criado.temp:=FALSE
IF
  (Intervalo.armazenamento<>0) OR (Intervalo.salvamento<>0)
    SEQ
      so.open (from.isv, to.isv, [arquivo.tmp FROM 0 FOR tam.tmp],
               spt.binary, spm.new.update, temporario, resultado)
      criado.temp:=TRUE
  TRUE
    SKIP
--}}}
--{{{  ajusta valores de intervalo para uso com REM
--{{{  Intervalo de armazenamento
IF
  Intervalo.armazenamento = 0
    Intervalo.armazenamento:=Num.quadros+1
  TRUE
    SKIP
--}}}
--{{{  Intervalo de salvamento
IF
  Intervalo.salvamento = 0
    Intervalo.salvamento:=Num.quadros+1
  TRUE
    SKIP
--}}}
--{{{  Intervalo de escrita de totais
IF
  Intervalo.totais = 0
    Intervalo.totais:=Num.quadros+1
  TRUE
    SKIP
--}}}
--{{{  Intervalo de redistribuicao de particulas por grao
IF
  Intervalo.redistribuicao = 0
    Intervalo.redistribuicao:=Num.quadros+1
  TRUE
    SKIP
--}}}
--}}}
so.write.string (from.isv, to.isv, "*c*nProcessando quadro ")
Ac.t:=0.0(REAL32)
Ac.t2:=0.0(REAL32)
tempo ? t1
--{{{  constantes auxiliares
VAL volume IS REAL64 (Box[0]*Box[1]):
VAL Rho IS (REAL64 ROUND Num.Part)/volume:
--}}}
SEQ q=quadro.inicial FOR ((Num.quadros-quadro.inicial)+1)
  --{{{  variaveis locais ao loop
  REAL64 P, T, E, Eo, Ec, V, Vc, W:
  INT Ncut:
  REAL32 t:
  --}}}
  SEQ
```

```
so.write.int (from.isv, to.isv, q, 5)
--{{{  envia mensagem para inicio de calculos
Para.calculo ! prossegue
tempo ? t2
--}}}
--{{{  testa se precisa redistribuir particulas
IF
  ((q REM Intervalo.redistribuicao)=0) OR (q=1)
    Para.calculo ! redistribui.particulas
  TRUE
    Para.calculo ! nao.redistribui
--}}}
--{{{  recebe totalizadores
De.calculo ? CASE totalizadores; Ncut; Ec; V; Vc; W
tempo ? t3
--{{{  acumula totais
Ec:=Ec/(REAL64 ROUND Num.Part)
T:=Ec*(2.0(REAL64)/livre)
Ac.Ec:=Ac.Ec+Ec
Ac.Ec2:=Ac.Ec2+(Ec*Ec)
V:=V/(REAL64 ROUND Num.Part)
Ac.V:=Ac.V+V
Ac.V2:=Ac.V2+(V*V)
E:=Ec+V
Ac.E:=Ac.E+E
Ac.E2:=Ac.E2+(E*E)
Vc:=Vc/(REAL64 ROUND Num.Part)
Eo:=Vc+Ec
Ac.Eo:=Ac.Eo+Eo
Ac.Eo2:=Ac.Eo2+(Eo*Eo)
P:=(Rho*T)+(W/volume)
Ac.P:=Ac.P+P
Ac.P2:=Ac.P2+(P*P)
--}}}
--{{{  calcula e acumula tempo de calculos
t:=seg.tick*(REAL32 ROUND (t3 MINUS t2))
Ac.t:=Ac.t+t
Ac.t2:=Ac.t2+(t*t)
--}}}
--}}}
--{{{  testa se precisa escrever totais
IF
  (q REM Intervalo.totais) = 0
    --{{{  envia totais
    SEQ
      wr.int (q, 6)
      wr.int (Ncut, 6)
      wr.r64 (E, 2, 5)
      wr.r64 (Eo, 2, 5)
      wr.r64 (Ec, 2, 5)
      wr.r64 (V, 2, 5)
      wr.r64 (T, 2, 5)
      wr.r64 (P, 2, 5)
      wr.r32 (t, 2, 3)
      wr.nl ()
    --}}}
  TRUE
    SKIP
--}}}
--{{{  testa se precisa fazer armazenamento
IF
  (q REM Intervalo.armazenamento) = 0
    SEQ
      --{{{  le dados das particulas e escreve em temporario
      --{{{  variaveis e constantes locais
```

```
BOOL fim.das.particulas:
INT tr, tv, ta:
[Ndim]INT r:
[Ndim]REAL32 v, a:
INT Numero.particula:
--}}}
SEQ
  Para.calculo ! comunica.dados
  fim.das.particulas:=FALSE
  WHILE NOT fim.das.particulas
    SEQ
      De.calculo ? CASE
        dados; Numero.particula; r; v; a
          --{{{  grava dados no arquivo temporario
          --{{{  redefine r, v e a como vetore de BYTE
          VAL []BYTE r.INT RETYPES r:
          VAL []BYTE v.INT RETYPES v:
          VAL []BYTE a.INT RETYPES a:
          --}}}
          SEQ
            --{{{ busca posicao onde a part. deve ser armaz.
            so.seek (from.isv, to.isv, temporario,
               (INT32 (Numero.particula*Tamanho.de.uma)),
               spo.start, resultado)
            IF
              resultado<>spr.ok
                SEQ
                  so.write.string (from.isv, to.isv,
                    "*c*n>>>Inconsistencia entre arquivo e simulacao")
                  STOP
              TRUE
                SKIP
            --}}}
            --{{{  escreve dados da particula
            so.write (from.isv, to.isv, temporario,
                      r.INT, tr)
            so.write (from.isv, to.isv, temporario,
                      v.INT, tv)
            so.write (from.isv, to.isv, temporario,
                      a.INT, ta)
            IF
              ((tr+tv)+ta)<Tamanho.de.uma
                SEQ
                  so.write.string (from.isv, to.isv,
                    "*c*n>>>Erro ao armazenar valores")
                  STOP
              TRUE
                SKIP
            --}}}
          --}}}
        terminaram.dados
          fim.das.particulas:=TRUE
--}}}
--{{{  reposiciona temporario no inicio do arquivo
so.seek (from.isv, to.isv, temporario, 0(INT32),
               spo.start, resultado)
--}}}
--{{{  copia dados do arquivo temporario em armazenamento
BYTE resultado:
[Tamanho.de.uma]BYTE temp:
INT tamanho.lido, tamanho.escrito:
SEQ
  SEQ i=0 FOR Num.Part
    SEQ
      --{{{  le do arquivo temporario para temp
```

```
            so.read (from.isv, to.isv, temporario,
                     tamanho.lido, temp)
            IF
              tamanho.lido <> Tamanho.de.uma
                SEQ
                  so.write.string (from.isv, to.isv,
                    ">>Inconsistencia no arquivo temporario gerado")
                  STOP
              TRUE
                SKIP
            --}}}
            --{{{  escreve de temp no arquivo de armazenamento
            so.write (from.isv, to.isv, armazena, temp,
                      tamanho.escrito)
            IF
              tamanho.escrito <> Tamanho.de.uma
                SEQ
                  so.write.string (from.isv, to.isv,
                    ">>Erro na escrita de arquivo de armazenamento")
                  STOP
              TRUE
                SKIP
            --}}}
      --}}}
  TRUE
    SKIP
--}}}
--{{{  testa se precisa fazer salvamento
IF
  (q REM Intervalo.salvamento) = 0
    BYTE resultado:
    INT32 salva:
    SEQ
      --{{{  le dados das particulas e escreve em temporario
      --{{{  variaveis e constantes locais
      BOOL fim.das.particulas:
      INT tr, tv, ta:
      BYTE resultado:
      [Ndim]INT r:
      [Ndim]REAL32 v, a:
      INT Numero.particula:
      --}}}
      SEQ
        Para.calculo ! comunica.dados
        fim.das.particulas:=FALSE
        WHILE NOT fim.das.particulas
          SEQ
            De.calculo ? CASE
              dados; Numero.particula; r; v; a
                --{{{  grava dados no arquivo temporario
                --{{{  redefine r, v e a como vetore de BYTE
                VAL []BYTE r.INT RETYPES r:
                VAL []BYTE v.INT RETYPES v:
                VAL []BYTE a.INT RETYPES a:
                --}}}
                SEQ
                  --{{{  busca pos. onde a part. deve ser armaz.
                  so.seek (from.isv, to.isv, temporario, (INT32
                     (Numero.particula*Tamanho.de.uma)),
                     spo.start, resultado)
                  IF
                    resultado<>spr.ok
                      SEQ
                        so.write.string (from.isv, to.isv,
                          "*c*n>>>Inconsistencia entre arquivo e simulacao")
```

```
                STOP
              TRUE
                SKIP
            --}}}
            --{{{  escreve dados da particula
            so.write (from.isv, to.isv, temporario,
               r.INT, tr)
            so.write (from.isv, to.isv, temporario,
               v.INT, tv)
            so.write (from.isv, to.isv, temporario,
               a.INT, ta)
            IF
              ((tr+tv)+ta)<Tamanho.de.uma
                SEQ
                  so.write.string (from.isv, to.isv,
                    "*c*n>>>Erro ao armazenar valores")
                  STOP
              TRUE
                SKIP
            --}}}
          --}}}
        terminaram.dados
          fim.das.particulas:=TRUE
--}}}
--{{{  reposiciona temporario no local dos acumuladores
so.seek (from.isv, to.isv, temporario, (INT32
      (Num.Part*Tamanho.de.uma)), spo.start, resultado)
--}}}
--{{{  salva acumuladores e numero do quadro
INT tam.escrito:
--{{{  redefinicao dos acumuladores
VAL []BYTE q.aux RETYPES q:
VAL []BYTE E.aux RETYPES Ac.E:
VAL []BYTE Eo.aux RETYPES Ac.Eo:
VAL []BYTE Ec.aux RETYPES Ac.Ec:
VAL []BYTE V.aux RETYPES Ac.V:
VAL []BYTE P.aux RETYPES Ac.P:
VAL []BYTE E2.aux RETYPES Ac.E2:
VAL []BYTE Eo2.aux RETYPES Ac.Eo2:
VAL []BYTE Ec2.aux RETYPES Ac.Ec2:
VAL []BYTE V2.aux RETYPES Ac.V2:
VAL []BYTE P2.aux RETYPES Ac.P2:
--}}}
SEQ
  --{{{  escreve numero do quadro
  so.write (from.isv, to.isv, temporario, q.aux, tam.escrito)
  --}}}
  --{{{  escreve cada um dos acumuladores
  so.write (from.isv, to.isv, temporario, E.aux, tam.escrito)
  so.write (from.isv, to.isv, temporario, Eo.aux,tam.escrito)
  so.write (from.isv, to.isv, temporario, Ec.aux,tam.escrito)
  so.write (from.isv, to.isv, temporario, V.aux, tam.escrito)
  so.write (from.isv, to.isv, temporario, P.aux, tam.escrito)
  so.write (from.isv, to.isv, temporario, E2.aux,tam.escrito)
  so.write (from.isv, to.isv, temporario, Eo2.aux,tam.escrito)
  so.write (from.isv, to.isv, temporario, Ec2.aux,tam.escrito)
  so.write (from.isv, to.isv, temporario, V2.aux,tam.escrito)
  so.write (from.isv, to.isv, temporario, P2.aux,tam.escrito)
  --}}}
--}}}
--{{{  muda o nome de arquivo.sav para um nome temporario
IF
  q<>Intervalo.salvamento
    so.rename (from.isv, to.isv,
               [arquivo.sav FROM 0 FOR tam.sav],
```

```
                    [arquivo.nnn FROM 0 FOR tam.nnn], resultado)
          TRUE
            INT32 temp:
            SEQ
              so.open (from.isv, to.isv,
                       [arquivo.nnn FROM 0 FOR tam.nnn],
                       spt.binary, spm.output, temp, resultado)
              so.close (from.isv, to.isv, temp, resultado)
        --}}}
        --{{{  muda o nome de arquivo.tmp para arquivo.sav
        so.close (from.isv, to.isv, temporario, resultado)
        so.rename (from.isv, to.isv,
                       [arquivo.tmp FROM 0 FOR tam.tmp],
                       [arquivo.sav FROM 0 FOR tam.sav], resultado)
        --}}}
        --{{{  muda o nome do nome temporario para arquivo.tmp
        so.rename (from.isv, to.isv,
                       [arquivo.nnn FROM 0 FOR tam.nnn],
                       [arquivo.tmp FROM 0 FOR tam.tmp], resultado)
        so.open (from.isv, to.isv,
                       [arquivo.tmp FROM 0 FOR tam.tmp], spt.binary,
                       spm.existing.update, temporario, resultado)
        --}}}
      TRUE
        SKIP
    --}}}
    so.write.string (from.isv, to.isv, "*#08*#08*#08*#08*#08")
--{{{  ajustes de finalizacao
tempo ? t4
Para.calculo ! fim.da.simulacao
--{{{  calcula e comunica valores globais de toda simulacao
REAL64 Media.T, Media.E, Media.V, Media.Ec, Media.P, Media.Eo:
REAL64 Desvio.T, Desvio.E, Desvio.V, Desvio.Ec, Desvio.P, Desvio.Eo:
SEQ
  --{{{  calcula medias
  SEQ
    Media.E:=Ac.E/(REAL64 ROUND Num.quadros)
    Media.V:=Ac.V/(REAL64 ROUND Num.quadros)
    Media.Ec:=Ac.Ec/(REAL64 ROUND Num.quadros)
    Media.P:=Ac.P/(REAL64 ROUND Num.quadros)
    Media.Eo:=Ac.Eo/(REAL64 ROUND Num.quadros)
    Media.T:=Media.Ec*(2.0(REAL64)/livre)
  --}}}
  --{{{  calcula desvios-padrao
  SEQ
    Desvio.E:=DSQRT((Ac.E2/(REAL64 ROUND Num.quadros))-
                (Media.E*Media.E))
    Desvio.V:=DSQRT((Ac.V2/(REAL64 ROUND Num.quadros))-
                (Media.V*Media.V))
    Desvio.Ec:=DSQRT((Ac.Ec2/(REAL64 ROUND Num.quadros))-
                (Media.Ec*Media.Ec))
    Desvio.P:=DSQRT((Ac.P2/(REAL64 ROUND Num.quadros))-
                (Media.P*Media.P))
    Desvio.Eo:=DSQRT((Ac.Eo2/(REAL64 ROUND Num.quadros))-
                (Media.Eo*Media.Eo))
    Desvio.T:=Desvio.Ec*(2.0(REAL64)/livre)
  --}}}
  --{{{  comunica resultados
  SEQ
    type.r64 ("*n*nMedia da energia sem descontinuidade: ",
         Media.Eo, 3, 10)
    type.r64 ("*n                          Desvio padrao: ",
         Desvio.Eo, 3, 10)
    type.r64 ("*n*n                      Media da Energia: ",
         Media.E, 3, 10)
```

```
    type.r64 ("*n                          Desvio padrao: ",
         Desvio.E, 3, 10)
    type.r64 ("*n*n                   Media do potencial: ",
         Media.V, 3, 10)
    type.r64 ("*n                          Desvio padrao: ",
         Desvio.V, 3, 10)
    type.r64 ("*n*n             Media da Energia cinetica: ",
         Media.Ec, 3, 10)
    type.r64 ("*n                          Desvio padrao: ",
         Desvio.Ec, 3, 10)
    type.r64 ("*n*n                 Media da Temperatura: ",
         Media.T, 3, 10)
    type.r64 ("*n                          Desvio padrao: ",
         Desvio.T, 3, 10)
    type.r64 ("*n*n                      Media da pressao: ",
         Media.P, 3, 10)
    type.r64 ("*n                          Desvio padrao: ",
         Desvio.P, 3, 10)
  --}}}
--}}}
--{{{  calcula e comunica tempo de calculos e respectivo desvio
REAL32 Media.t, Desvio.t:
SEQ
  Media.t:=Ac.t/(REAL32 ROUND Num.quadros)
  Desvio.t:=SQRT((Ac.t2/(REAL32 ROUND Num.quadros))-(Media.t*Media.t))
  type.r32 ("*n*n          Media do tempo de calculos: ",
                Media.t, 2, 5)
  type.r32 ("s.*n                          Desvio padrao: ",
                Desvio.t, 2, 5)
--}}}
--{{{  calcula e comunica tempo total
SEQ
  type.r32 ("*c*nTempo total de simulacao:",
                min.tick*(REAL32 ROUND (t4 MINUS t1)), 3, 3)
  wr.str ("min")
--}}}
--{{{  fecha e apaga arquivo temporario
IF
  criado.temp
    SEQ
      --{{{  fecha
      so.close (from.isv, to.isv, temporario, resultado)
      IF
        resultado <> spr.ok
          SEQ
            so.write.string (from.isv, to.isv,
                 ">>Erro ao fechar aquivo temporario")
            STOP
        TRUE
          SKIP
      --}}}
      --{{{  apaga
      so.remove (from.isv, to.isv, [arquivo.tmp FROM 0 FOR tam.tmp],
                         resultado)
      IF
        resultado <> spr.ok
          SEQ
            so.write.string (from.isv, to.isv,
                 ">>Erro ao apagar aquivo temporario")
            STOP
        TRUE
          SKIP
      --}}}
  TRUE
    SKIP
```

```
        --}}}
        --{{{  fecha arquivo de armazenamento
        IF
          Intervalo.armazenamento<Num.quadros
            SEQ
              so.close (from.isv, to.isv, armazena, resultado)
              IF
                resultado <> spr.ok
                  SEQ
                    so.write.string (from.isv, to.isv,
                          "*c*n>>>Erro ao fechar arquivo de armazenamento")
                    STOP
                TRUE
                  SKIP
          TRUE
            SKIP
        --}}}
        --{{{  fecha arquivo de saida
        IF
          Intervalo.totais<Num.quadros
            so.close (from.isv, to.isv, saida, resultado)
          TRUE
            SKIP
        --}}}
        --{{{  apaga arquivo de salvamento
        IF
          Intervalo.salvamento<Num.quadros
            so.remove (from.isv, to.isv, [arquivo.sav FROM 0 FOR tam.sav],
                          resultado)
          TRUE
            SKIP
        --}}}
        --{{{  termina processos auxiliares
        INT lixo:
        SEQ
          keyboard ? lixo
          WHILE lixo<>ft.terminated
            keyboard ? lixo
        parar.mult ! FALSE
        --}}}
        --}}}
      --}}}
      so.multiplexor (from.isv, to.isv, ts, fs, parar.mult)
    so.exit (from.isv, to.isv, sps.success) -- termina o server
:
--}}}F
--...F code HT8
--:::A  1 2
--:::F control4.dcd
--...F descriptor
--:::A  1 4
--:::F control4.dds
--...F link
--:::A  1 9
--:::F control4.dlk
--...F debug
--:::A  1 5
--:::F control4.ddb
--...F CODE SC Novo controle (com exit!) 09:08:29 13-Feb-93
--:::A  2 10
--:::F control4.csc
--}}}
--{{{  definicao de canais
[NDp]CHAN OF Sinais.controle controles:
[NDp]CHAN OF Devolve.resultados resultados:
```

```
[NDp+1]CHAN OF Comunicacao.vizinhos para.esquerda, para.direita:
--}}}
PLACED PAR
  PROCESSOR 0 T8
    --{{{  processador de controle (substituido o EXE do TDS)
    --{{{  definicao de canais
    CHAN OF SP from.isv, to.isv:
    PLACE from.isv AT link0.in:
    PLACE to.isv AT link0.out:
    PLACE controles[0] AT link2.out:
    PLACE resultados[0] AT link2.in:
    --}}}
    controle (from.isv, to.isv, controles[0], resultados[0])
    --}}}
  PLACED PAR dp=0 FOR NDp-1
    --{{{  somente calculos nos primeiros n-1 processadores
    PROCESSOR dp+1 T8
      --{{{  definicao de canais
      PLACE controles[dp] AT link1.in:
      PLACE resultados[dp] AT link1.out:
      PLACE controles[dp+1] AT link2.out:
      PLACE resultados[dp+1] AT link2.in:
      PLACE para.direita[dp+1] AT link3.out:
      PLACE para.esquerda[dp] AT link0.out:
      PLACE para.esquerda[dp+1] AT link3.in:
      PLACE para.direita[dp] AT link0.in:
      --}}}
      calculos (controles[dp], controles[dp+1],
                para.direita[dp+1], para.esquerda[dp],
                para.esquerda[dp+1], para.direita[dp],
                resultados[dp], resultados[dp+1], dp)
    --}}}
  PROCESSOR NDp T8
    --{{{  calculos e vazio no ultimo processador de calculos
    --{{{  definicao de canais
    PLACE controles[NDp-1] AT link1.in:
    PLACE resultados[NDp-1] AT link1.out:
    PLACE para.direita[NDp] AT link3.out:
    PLACE para.esquerda[NDp-1] AT link0.out:
    PLACE para.direita[NDp-1] AT link0.in:
    PLACE para.esquerda[NDp] AT link3.in:
    CHAN OF Sinais.controle controle.interno:
    CHAN OF Devolve.resultados resultado.interno:
    --}}}
    PAR
      calculos (controles[NDp-1], controle.interno,
                para.direita[NDp], para.esquerda[NDp-1],
                para.esquerda[NDp], para.direita[NDp-1],
                resultados[NDp-1], resultado.interno, NDp-1)
      vazio (controle.interno, resultado.interno)
    --}}}
--}}}F
--...F code HT4
--:::A  1 2
--:::F rodand08.dcd
--...F descriptor
--:::A  1 4
--:::F rodand08.dds
--...F CODE PROGRAM rodando numa rede como um stand-alone 09:08:47 13-Feb-93
--:::A  2 11
--:::F rodand08.cpr
--}}}
```