ISABEL CRISTINA MELLO ROSSETI

UMA BIBLIOTECA PARA BALANCEAMENTO DE CARGA EM AMBIENTES DISTRIBUÍDOS

DISSERTAÇÃO DE MESTRADO

DEPARTAMENTO DE INFORMÁTICA

Rio de Janeiro, 05 de Março de 1998

ISABEL CRISTINA MELLO ROSSETI

UMA BIBLIOTECA PARA BALANCEAMENTO DE CARGA EM AMBIENTES DISTRIBUÍDOS

Dissertação apresentada ao Departamento de Informática da PUC-Rio como parte dos requisitos para a obtenção do título de Mestre em Ciências em Informática.
Orientadora: Noemi Rodriguez

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro

Rio de Janeiro, 05 de Março de 1998

Essa dissertação é dedicada à minha família, Delza, José e Cristiane, por todos os esforços dispensados nessa jornada.

AGRADECIMENTOS

À minha orientadora, Noemi Rodriguez, pela excelente orientação. Muito obrigada pelo seus conselhos, por acreditar e incentivar esse trabalho e por me permitir partilhar alguns de seus conhecimentos.

À CAPES, pelo apoio finaceiro.

Aos funcionários da PUC-Rio, LNCC e NCE, pelo apoio técnico.

Ao Flávio Varejão, pelos seus conselhos e pela motivação e apoio ao ingresso no programa de mestrado.

À Simone Martins, pelo suporte técnico em todas as fases do desenvolvimento desse trabalho.

Ao pessoal da sala 8, pela incansáveis discussões a respeito dos problemas tratados nesse trabalho.

À Andréa, Portela e Eliana, pelas nossas programações onde discutíamos a respeito de todos os assuntos, exceto desse trabalho.

À Thais Batista, pelos conselhos em todas as fases do mestrado.

À Soeli Fiorini, pelas longas conversas nas minhas noites de insônia, ao longo desses 3 anos de convivência.

À Alba Sampaio, pelas palavras de incentivo em todos os momentos dessa dissertação.

À Maria, Danilo, Vivian e Rafael, minha segunda família, pelo amor e carinhos dedicados.

Aos amigos, do Rio e de outros estados, o que seria de mim sem vocês?

RESUMO

Diversas técnicas de balanceamento de carga têm sido propostas, a fim de obter melhoria no desempenho de aplicações paralelas. Nesse trabalho, são apresentados alguns algoritmos de balanceamento de carga. Utiliza-se um modelo analítico unificador, o Modelo Iterativo de Matriz, no qual são encaixados os diversos algoritmos. Discute-se a aplicação de alguns desses algoritmos a um problema específico, o problema dos N-corpos. Discute-se a biblioteca de balanceamento de carga desenvolvida para uma implementação paralela desse problema, baseada no algoritmo de Barnes-Hut.

ABSTRACT

Many load balancing techniques have been proposed, in order to improve the performance of parallel applications. This dissertation presents some existing load balancing algorithms. These algorithms are presented according to an analytical model called Matrix Iterative Model. The aplication of some these algorithms to a specific problem, the N-body problem, is discussed. A load balancing library developed for a parallel implementation of this problem, based on Barnes-Hut algorithm, is also discussed.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Atualmente, convive-se com um crescente processo de transformação de ambientes computacionais centralizados em ambientes computacionais distribuídos. Essa transformação é motivada por um conjunto de vantagens que incluem menor custo, maior poder computacional global e maior confiabilidade. No entanto, a maioria das tecnologias distribuídas utilizadas hoje ainda não permite que essas vantagens sejam efetivamente obtidas, impedindo que os usuários atinjam os resultados desejados [Uch95].

Um dos objetivos que não foi devidamente alcançado foi um maior poder computacional a um menor custo, sem que haja para o usuário um aumento no nível de complexidade de utilização do sistema. Assim, um dos problemas mais relevantes que aparece em sistemas distribuídos é o de maximizar o uso dos recursos computacionais, através da redistribuição da execução das tarefas pelo sistema. Essa redistribuição de tarefas é conhecida como balanceamento ou compartilhamento de carga.

Segundo [BF96], o balanceamento de carga é o processo de redistribuir equilibradamente as tarefas entre os nós que compõem o sistema. Já o compartilhamento de carga é o processo de transferir tarefas de máquinas que estão com o número de tarefas maior que um limite pré-definido de tarefas, para outros nós do sistema que estão com o número de tarefas menor que esse limite.

Nesse trabalho, o termo "balanceamento de carga" refere-se ao processo de redistribuir tarefas, não garantindo, porém, um nível de carga equilibrado entre todos os nós participantes do sistema.

Diversas técnicas de redistribuição de carga têm sido propostas [MD94], [FG96], [LLC96] e [ELZ86], visando maximizar o emprego dos recursos e a utilização do sistema. Essas técnicas, geralmente, executam as mesmas operações, em diferentes níveis de complexidade [FG96] e [ELZ86]. Portanto, pode-se padronizar essas operações, a fim de representar um grande intervalo de procedimentos de balanceamento.

Uma das formas utilizadas para estudar o desempenho de algoritmos de balanceamento de carga são os modelos analíticos [ELZ86]. Segundo [Cha80], um modelo analítico é uma representação do sistema onde é possível deduzir as fórmulas ou equações matemáticas que expressam uma solução para um certo problema em estudo. Como muitos dos modelos analíticos existentes são baseados em teoria das filas [MD94], eles geralmente não conseguem derivar resultados gerais aplicáveis a um grande intervalo de procedimentos de balanceamento de carga, porque utilizam aproximações simplificadas para deduzir as equações matemáticas que eles usam, o que limita o número de algoritmos de balanceamento de carga que podem fazer uso deles. Além disso, esses modelos levam a análises de desempenho irreais devido a essas aproximações [MD94]. Esses modelos são úteis, entretanto, quando se quer fazer uma análise geral do comportamento do sistema.

O Modelo Iterativo de Matriz, definido em [Cyb89], é um modelo analítico em que as deduções matemáticas estão todas baseadas em matrizes que representam o sistema. Como esse modelo independe do algoritmo de balanceamento de carga utilizado [FG96], ele pode ser usado para representar uma grande variedade de procedimentos de balanceamento.

Na descrição desse modelo, uma das suposições que deve ser satisfeita é que a aplicação deve pertencer à classe das aplicações iterativas síncronas. Essa classe é formada por algoritmos que geralmente possuem a seguinte heurística: a execução segue por um número determinado de iterações e, o trabalho associado com cada iteração deve ser distribuído entre os nós participantes do sistema. Ao fim de cada iteração, esses nós fazem uma sincronização e, então, iniciam a próxima iteração.

Uma das aplicações síncronas em que é necessário o uso de balanceamento de carga é a simulação paralela do problema dos N-corpos [FG96] e [FG93]. Técnicas clássicas de N-corpos estudam a evolução de um sistema de N partículas (ou corpos) onde existe uma força (por exemplo, a força gravitacional) entre pares de partículas. Essas técnicas são amplamente aplicadas a problemas de astrofísica, de dinâmica dos fluidos, entre outros.

Nesse trabalho, o balanceamento de carga em ambientes distribuídos é investigado. Alguns algoritmos de redistribuição de tarefas são discutidos. Utiliza-se o Modelo Iterativo de Matriz para apresentar os diversos algoritmos. Define-se um protótipo de uma biblioteca de balanceamento de carga básica e extensível. Essa biblioteca, escrita em C e PVM, é composta por rotinas modulares usadas por uma variedade de algoritmos de balanceamento.

A simulação paralela do problema dos N-corpos, descrita em [FG93] e [FG96], foi usada para validar a referida biblioteca. Os experimentos para analisar o desempenho dessa simulação, antes e após a inclusão da biblioteca de balanceamento, foram realizados na máquina paralela IBM SP/2 do Núcleo de Computação Eletrônica (NCE) da Universidade Federal do Rio de Janeiro (UFRJ) e do Laboratório Nacional de Computação Científica (LNCC).

A simulação sequencial discutida nesse trabalho está baseada no algoritmo de Barnes-Hut, descrito em [CT92], [Vel94] e [BH86], para estudar a evolução de um sistema de corpos sob a influência da atração gravitacional Newtoniana. Os corpos consistem de uma massa, posições e velocidades iniciais e eles estão distribuídos sobre um domínio físico finito. A simulação segue sobre um número de iterações, onde, em cada iteração, há o cálculo das forças em cada corpo e há a atualização de sua posição e de outros atributos.

Na simulação paralela do problema dos N-corpos, devida a [FG93] e [FG96], o domínio físico é dividido em pequenas regiões e cada região é alocada a um processador. Inicialmente, os domínios são distribuídos de forma que cada processador tenha o mesmo número de partículas. Cada processador é responsável por todos os cálculos associados com as partículas de sua região. No entanto, para calcular as forças em cada partícula, é necessário conhecer a posição de outras partículas e isso envolve troca de informação entre pares de processadores. Depois da troca de informações, cada processador calcula a força resultante em cada partícula de sua região e atualiza suas posições e velocidades. O processador verifica quais de suas partículas atravessaram as fronteiras de seu domínio e, então, transfere as partículas que não pertencem mais a ele para os processadores apropriados.

Essa transferência causa um desequilíbrio na distribuição de carga entre os processadores. O algoritmo de balanceamento de carga é invocado ao fim de cada iteração. O balanceamento de carga consiste em redesignar domínios para cada processador, tal que o número de partículas associadas aos processadores esteja balanceado.

Os objetivos principais do trabalho são os seguintes:

- realizar um *survey* de alguns algoritmos de balanceamento de carga existentes. Esses algoritmos são apresentados segundo o Modelo Iterativo de Matriz. Esse modelo analítico foi escolhido pela variedade de algoritmos de balanceamento de carga que ele pode representar;
- definir uma biblioteca de balanceamento de carga que contém as rotinas modulares usadas pelos diversos algoritmos. Essa biblioteca é desenvolvida para trabalhar independentemente do algoritmo de balanceamento de carga utilizado;
- obter uma melhoria no desempenho da simulação paralela do problema dos N-corpos usada nesse trabalho, após a inclusão da biblioteca de balanceamento de carga desenvolvida.

O trabalho tem como contribuições principais:

- o estabelecimento de um paralelo entre as características básicas de alguns algoritmos de balanceamento de carga, ressaltando suas similaridades;
- o desenvolvimento de uma biblioteca básica extensível que serve para implementar uma grande variedade de algoritmos de balanceamento de carga;
- a utilização dessa implementação na aplicação de simulação paralela do problema dos N-corpos, validando a referida implementação e também melhorando o desempenho da simulação.

Esse trabalho é organizado como segue. No capítulo 2, são mostradas definições muito usadas em balanceamento de carga, bem como alguns algoritmos de balanceamento existentes. Os diversos algoritmos são apresentados segundo o Modelo Iterativo de Matriz. Além disso, descreve-se esse modelo analítico.

No capítulo 3, são explicadas as etapas do algoritmo de Barnes-Hut e as diferenças desse algoritmo em relação a simulação sequencial. Para paralelizar a referida simulação, foram feitas algumas adaptações que, também, são mostradas nesse capítulo.

No capítulo 4, são descritas, em alto nível, as rotinas que compõem a biblioteca básica extensível implementada nesse trabalho.

No capítulo 5, são analisados os resultados obtidos pelas execuções da simulação paralela do problema dos N-corpos antes e após a inclusão da biblioteca desenvolvida.

No capítulo 6, são apresentadas as considerações finais desse trabalho.

Capítulo 2

Balanceamento de Carga

Para obter benefícios com a computação paralela, a carga computacional de uma aplicação deve ser distribuída de maneira efetiva entre todos os processadores do sistema [FG96]. Essa distribuição de carga computacional é chamada de balanceamento de carga.

O balanceamento de carga é o processo que permite melhor compartilhamento dos recursos computacionais das diversas máquinas interligadas pela redistribuição dessa carga através dos nós que compõem o sistema.

O objetivo desse capítulo é apresentar um *survey* de alguns algoritmos de balanceamento de carga existentes. Esses algoritmos serão apresentados segundo o Modelo Iterativo de Matriz. O Modelo Iterativo de Matriz, definido em [Cyb89], é um modelo analítico em que as deduções matemáticas estão todas baseadas em matrizes que representam o sistema. Como esse modelo independe do algoritmo de balanceamento de carga utilizado [FG96], ele pode ser usado para representar uma grande variedade de procedimentos de balanceamento.

Na seção 2.1, será discutido o balanceamento de carga em ambientes distribuídos. Na seção 2.2, o Modelo Iterativo de Matriz será mostrado. Na seção 2.3, as medidas de desempenho do Modelo Iterativo de Matriz são apresentadas. Na seção 2.4, os algoritmos de balanceamento de carga, apresentados segundo o Modelo Iterativo de Matriz serão mostrados. Na seção 2.5, serão apresentadas algumas considerações sobre os algoritmos discutidos.

2.1 Conceitos Básicos de Balanceamento de Carga

Um problema importante que aparece em sistemas distribuídos é o de melhorar o desempenho de um conjunto de nós através do balanceamento da carga do sistema.

O balanceamento de carga em um sistema distribuído é o processo de compartilhar os recursos computacionais das diversas máquinas interligadas pela redistribuição da carga através dos nós do sistema.

Um esquema de balanceamento de carga pode ser classificado, conforme sua relação com o estado atual de carregamento do sistema, em estático, dinâmico ou adaptativo. Os esquemas de balanceamento de carga estáticos independem do estado do sistema. Os dinâmicos sempre consideram o estado do sistema e os esquemas adaptativos mudam seus parâmetros ou políticas para refletir as mudanças do estado do sistema [LLC96].

Três componentes importantes do balanceamento de carga adaptativo e dinâmico são: a política de transferência, a política de informação e a política de localização. A política de transferência determina se uma tarefa deve ser transferida. A política de informação determina quais informações serão usadas para fazer as redistribuições de carga e como elas serão distribuídas entre os participantes. Isto é, determina as regras para coletar informações sobre os nós participantes do sistema. Finalmente, a política de localização é responsável por achar os nós de destino para onde as tarefas serão transferidas [LLC96].

2.2 Modelo Iterativo de Matriz

O Modelo Iterativo de Matriz, proposto em [FG96], é um modelo analítico que pode ser usado para representar um grande intervalo de algoritmos de balanceamento de carga. O modelo descreve a mudança na distribuição de tarefas sobre os processadores devido ao algoritmo de balanceamento de carga.

Para desenvolver esse modelo assume-se que:

(a) a aplicação pode ser dividida em um grande número de tarefas (comparado ao número de processadores);

(b) as tarefas são idênticas, com relação aos custos computacionais e de comunicação, e podem ser executadas por qualquer processador disponível;

(c) os processadores podem ter diferentes poderes computacionais;

(d) a aplicação deve ser síncrona.

O conjunto de processadores disponíveis é denotado por $P = \{P_1, ..., P_p\}$, onde p é o número de processadores. O poder computacional do processador P_i é denotado por C_i, onde C_i é proporcional ao número médio de operações executadas pelo processador em uma unidade de tempo.

O tempo gasto para executar uma tarefa em P_i é inversamente proporcional a C_i. Para encontrar o balanceamento de carga em um sistema heterogêneo, aloca-se a cada processador um número de tarefas proporcional ao respectivo poder computacional. Por exemplo, se P_i é duas vezes mais rápido que P_j (isto é, $C_i = 2C_j$), aloca-se a P_i o dobro de tarefas de P_j, para o sistema ficar balanceado.

O algoritmo de balanceamento de carga é invocado em instantes de tempo denotados por k ($k = 1, 2, 3, ...$). O algoritmo deve ser invocado em intervalos regulares, no fim de cada iteração em aplicações iterativas, ou deve ser disparado por um evento externo.

O Modelo Iterativo de Matriz descreve a distribuição das tarefas de cada processador do sistema, em cada um dos instantes k, quando o algoritmo de balanceamento de carga é invocado. O estado do sistema em k é representado por um vetor de distribuição de tarefas $w(k)$, onde $w_i(k)$ é o número de tarefas no processador P_i em k.

A distribuição de tarefas no instante de tempo $(k+1)$, $w(k+1)$, pode ser expressa como função de:

1. $w(k)$, a distribuição de tarefas em k;

2. a transferência de tarefas devido ao balanceamento de carga invocado em k;

3. a chegada de tarefas antes da *(k + 1)-ésima* invocação do algoritmo de balanceamento de carga;

4. a saída de tarefas antes da *(k + 1)-ésima* invocação do algoritmo de balanceamento de carga.

O modelo de transferência dinâmico para cada processador, P_i, pode ser representado por:

$$w_i(k+1) = w_i(k) - \textit{tarefas enviadas por } P_i \textit{ em } k \textit{ devido ao balanceamento de carga} + \textit{tarefas recebidas por } P_i \textit{ em } k \textit{ devido ao balanceamento de carga} + \textit{chegada de tarefas durante } [k, k+1] - \textit{saída de tarefas durante } [k, k+1] \qquad \{2.1\}$$

O algoritmo de balanceamento de carga tenta transferir as tarefas para compensar o desequilíbrio resultante da mudança dinâmica da aplicação e pode ser descrito em termos das três políticas seguintes:

1. política de informação $\rightarrow$ os algoritmos de balanceamento de carga diferem na informação de estado do sistema que eles necessitam e também diferem na aquisição dessa informação. Alguns algoritmos exigem uma medida de carga média em todos os processadores (ou num subconjunto de nós) do sistema, enquanto outros exigem a distribuição de carga atual dos processadores do sistema.
   Define-se $\bar{w}(k)$ como o vetor que representa a distribuição de carga ideal, onde o número de tarefas em cada processador é proporcional ao seu poder computacional. Define-se, também, $\tilde{w}(k)$ como o vetor que representa uma aproximação da distribuição ideal.

2. política de transferência $\rightarrow$ quando um algoritmo de balanceamento de carga é invocado, cada processador decide se ele quer participar do balanceamento de carga, baseado na informação de estado disponível. Em algoritmos *Sender-Initiated*[1], os processadores participam no balanceamento de carga se sua carga

[1] No algoritmo Sender-Initiated, descrito em [Qui94], um processador P_i decide enviar uma fração

é maior que o *threshold* de tarefas $w^*(k)$. A participação do processador no balanceamento de carga em k é representado pela matriz $A(k)$. Se o processador i decide participar do balanceamento de carga, o elemento $A_{ii}(k)$ recebe o valor 1; todos os outros elementos recebem o valor 0.

3. política de localização $\rightarrow$ cada processador que decide participar do balanceamento de carga tem que calcular:

   (a) o número de tarefas transferidas;

   (b) os destinos das tarefas.

   As decisões de migração de tarefas são representadas por uma matriz $p \times p$, $M(k)$, onde $M_{ij}(k)$ é a fração de tarefas que o processador P_i decide enviar a P_j (isto é, P_i envia $M_{ij}(k)w_i(k)$ tarefas a P_j). As decisões de migração de tarefas ficam necessariamente restritas à estrutura imposta pelo algoritmo de balanceamento de carga ou pela topologia do sistema. Define-se $H(k)$ como a matriz de topologia. $H(k)$ é uma matriz $p \times p$ que descreve os caminhos de migração possíveis. O elemento $H_{ij}(k)$ é igual a 1, se os processadores P_i e P_j podem trocar tarefas. Caso contrário, $H_{ij}(k) = 0$. Portanto, na construção da matriz M, $M_{ij}(k) \neq 0$ se e somente se $H_{ij}(k) = 1$.

A equação {2.1} pode ser reescrita como segue:

$$w_i(k+1) = w_i(k) - \sum_{j \neq i} M_{ij}(k)A_{ii}(k)w_i(k) + \sum_{j \neq i} M_{ji}(k)A_{jj}(k)w_j(k) + \lambda_i(k) - \mu_i(k) \qquad \{2.2\}$$

onde $\lambda(k)$ e $\mu(k)$ são vetores que denotam a taxa de chegada e a taxa de saída, respectivamente.

As duas somatórias representam a atividade de balanceamento de carga. A primeira corresponde à soma de todas as tarefas transferidas de P_i para os outros processadores e a segunda representa a soma de todas as tarefas transferidas para P_i.

α do excesso de suas tarefas para outro processador, quando o número de tarefas atribuídas a ele é maior que um certo limite de tarefas.

A distribuição "ideal", $\bar{w}(k)$, representa a distribuição de tarefas em um estado de balanceamento ideal de carga, $\bar{w}_i(k)$. A carga "ideal" no processador P_i é dada por:

$$\bar{w}_i(k) = \frac{C_i}{\sum_{\forall j P_j} C_j} \sum_{\forall j P_j} w_j(k) \qquad \{2.3\}$$

O termo $\forall j P_j$ indica que todos os processadores do sistema estão sendo levados em consideração. O primeiro fator da multiplicação é a fração do total de computação associado ao processador P_i e o segundo fator, $\sum_{\forall j P_j} w_j(k)$, é o número total de tarefas no sistema.

Muitos algoritmos usam a informação local (isto é, a distribuição de tarefas nos processadores da vizinhança), como uma aproximação da informação de estado global. A vizinhança 1-passo do processador P_i, $N_i^{(1)}$, é definida como segue: o processador P_j está na vizinhança 1-passo de P_i (isto é, $P_j \in N_i^{(1)}$), se ele é adjacente a P_i, isto é, $H_{ij}(k) = H_{ji}(k) = 1$. P_i coleta a informação de carga na vizinhança 1-passo para calcular sua distribuição de carga, $\tilde{w}_i(k)$, como segue:

$$\tilde{w}_i(k) = \frac{C_i}{C_i + \sum_{\forall j P_j \in N_i^{(1)}} C_j} \left(w_i(k) + \sum_{\forall j P_j \in N_i^{(1)}} w_j(k)\right) \qquad \{2.4\}$$

O primeiro fator da multiplicação denota o poder computacional do conjunto $P_i \cup N_i^{(1)}$ associado com P_i e o segundo fator denota o número total de tarefas no conjunto $P_i \cup N_i^{(1)}$.

2.3 Medidas de Desempenho do Modelo Iterativo de Matriz

O modelo analítico desenvolvido na seção 2.2 pode ser usado para determinar quão bem o algoritmo de balanceamento de carga mantém o nível de tarefas equilibrado nos processadores do sistema. Entretanto, os algoritmos de redistribuição de tarefas aumentam os custos computacionais e os custos de comunicação. Submeter uma aplicação ao balanceamento de carga é válido somente se os custos da redistribuição de tarefas são pequenos comparados aos custos de execução da aplicação.

A essência computacional de uma aplicação iterativa síncrona consiste de uma sequência de iterações com um conjunto de variáveis avaliadas em cada iteração. As variáveis são avaliadas usando alguns ou todos os valores das iterações prévias. Tipicamente, as variáveis são distribuídas entre todos os processadores do sistema e cada processador é responsável pelas variáveis alocadas a ele. No entanto, todos os processadores devem completar uma iteração e trocar os valores das variáveis com os outros processadores antes do início da próxima iteração.

Como cada iteração envolve a execução da aplicação e do algoritmo de balanceamento de carga, o tempo de execução total para max_iter iterações no processador i, onde i pode ser qualquer processador do conjunto $P = \{P_1, ..., P_p\}$, é dado por:

$$T_{total}(i) = \sum_{k=1}^{max\_iter} max_{allP_i}[T_{appl_i}(k) + T_{lb_i}(k)] \qquad \{2.5\}$$

O termo $allP_i$ indica que, para a operação que encontra o maior tempo de execução de um processador P_i pertencente ao conjunto P (max), todos os processadores do sistema estão sendo levados em consideração. O termo max resulta da sincronização de todos os processadores do sistema, ao fim de cada iteração. $T_{appl}(k)$ representa o custo de execução da aplicação na iteração k. $T_{lb}(k)$ representa o custo de execução do algoritmo de balanceamento de carga na k-ésima iteração.

Seja f_{task} a carga de computação e de comunicação associada com uma única tarefa, expressa em termos do número de operações realizadas. Quando se aloca $w_j(k)$ tarefas ao processador P_j na iteração k, o processador P_j gasta $w_j f_{task}/C_j$ para executar todas as tarefas na k-ésima iteração. Então:

$$T_{total}(i) = \sum_{k=1}^{max\_iter} max_{allP_i}[w_i(k) f_{task}/C_i + T_{lb_i}(k)] \qquad \{2.6\}$$

Um algoritmo ideal de balanceamento de carga mantém a distribuição de tarefas equilibrada e possui custo 0. Nesse caso, $\bar{w}_i(k)/C_i = \bar{w}_j(k)/C_j$ e o tempo de execução total é:

$$T_{total}(i) = \sum_{k=1}^{max\_iter} \bar{w}_1(k) f_{task}/C_1 \qquad \{2.7\}$$

Em implementações paralelas, o *speedup* obtido em um processador é uma medida de desempenho muito usada [FG96]. O *speedup* é definido da seguinte maneira:

$$speedup(i) = T_{seq} \ / \ T_{par} \qquad \{2.8\}$$

T_{seq} e T_{par} indicam os melhores tempos sequencial e paralelo obtido para o mesmo problema, respectivamente.

Outra medida de desempenho muito usada é o desequilíbrio de carga em cada iteração. O desequilíbrio $Im(k)$ é definido como segue:

$$Im(k) = max_{all P_i} w_i(k)/\bar{w}_i(k) \qquad \{2.9\}$$

2.4 Algoritmos de Balanceamento de Carga Apresentados Segundo o Modelo Iterativo de Matriz

Nessa seção, alguns algoritmos de balanceamento de carga serão adaptados ao Modelo Iterativo de Matriz. Os procedimentos de balanceamento de carga Random, Difusão e Redistribuição Completa já foram adaptados a esse modelo em [FG96].

2.4.1 Random

No algoritmo Random, descrito em [ELZ86], [FG96], [LLC96] e [MD94], um processador P_i decide enviar uma fração α do excesso de suas tarefas para outro processador, quando o número de tarefas atribuídas a ele, $w_i(k)$, é maior que um certo *threshold*, $w_i^*(k)$. Assim, o processador P_i envia $\alpha(w_i(k) - w_i^*(k))$ tarefas, se $(w_i(k) > w_i^*(k))$. O processador de destino P_e é escolhido randomicamente, do conjunto de processadores, para o envio dessa carga.

Um elemento da matriz de participação de processadores, $A_{ii}(k)$, é igual a 1 se $(w_i(k) > w_i^*(k))$. Todos os outros elementos de A são nulos. O elemento $M_{ie}(k) = \alpha$ e $M_{ij}(k) = 0$ se $j \neq e$.

Para esse algoritmo, as políticas são:

- de transferência $\rightarrow$ sobrecarga de um nó do sistema;
- de informação $\rightarrow$ nenhum nó do sistema divulga seu número de tarefas para os outros processadores;
- de localização $\rightarrow$ seleção randômica do nó de destino.

A equação $\{2.2\}$ pode ser reescrita como:

$$w_i(k+1) = w_i(k) - \alpha A_{ii}(k)\ (w_i(k) - w_i^*(k)) + \textstyle\sum_{j \neq i} A_{jj}(k)\ M_{ji}(k)\ (w_j(k) - w_j^*(k)) + \lambda_i(k) - \mu_i(k) \qquad \{2.10\}$$

2.4.2 Threshold

No algoritmo Threshold, descrito em [ELZ86], [LLC96] e [MD94], um processador P_i decide enviar uma fração α do excesso de suas tarefas para outro processador, quando o número de tarefas atribuídas a ele, $w_i(k)$, é maior que um certo *threshold*, $w_i^*(k)$.

O processador de destino P_e é escolhido randomicamente do conjunto de processadores. Se o número de tarefas atribuídas a P_e, $w_e(k)$, for menor que o valor de *threshold*, $w_e^*(k)$, transfere-se $\alpha(w_i(k) - w_i^*(k))$ tarefas para P_e. Caso contrário, repete-se o processo de seleção de outro nó.

Um elemento da matriz de participação de processadores, $A_{ii}(k)$, é igual a 1 se $(w_i(k) > w_i^*(k))$. Todos os outros elementos de A são nulos. O elemento $M_{ie}(k) = \alpha$ e $M_{ij}(k) = 0$ se $j \neq e$.

Para esse algoritmo, as políticas são:

- de transferência $\rightarrow$ sobrecarga de um nó do sistema;
- de informação $\rightarrow$ o nó de destino divulga seu número de tarefas para o processador que o selecionou [LLC96];
- de localização $\rightarrow$ seleção randômica do nó de destino.

A equação de distribuição de tarefas para esse algoritmo de balanceamento de carga é idêntica à equação $\{2.10\}$.

2.4.3 Shortest

No algoritmo Shortest, descrito em [ELZ86] e [LLC96], um processador P_i decide enviar uma fração α do excesso de suas tarefas para outro processador, quando o número de tarefas atribuídas a ele, $w_i(k)$, é maior que um certo *threshold*, $w_i^*(k)$.

Um conjunto de processadores é escolhido randomicamente do sistema. Desse conjunto, obtém-se o processador, P_e, com o menor número de tarefas atribuídas a ele. Se o número de tarefas atribuídas a P_e, $w_e(k)$, for menor que o valor de *threshold*, $w_e^*(k)$, transfere-se $\alpha(w_i(k) - w_i^*(k))$ tarefas para P_e. Caso contrário, repete-se o processo de seleção de um conjunto de nós.

Um elemento da matriz de participação de processadores, $A_{ii}(k)$, é igual 1 se $(w_i(k) > w_i^*(k))$. Todos os outros elementos de A são nulos. O elemento $M_{ie}(k) = \alpha$ e $M_{ij}(k) = 0$ se $j \neq e$.

Para esse algoritmo, as políticas são:

- de transferência $\rightarrow$ sobrecarga de um nó do sistema;
- de informação $\rightarrow$ o nó de destino divulga seu número de tarefas para o processador que o selecionou [LLC96];
- de localização $\rightarrow$ seleção do nó com o menor número de tarefas, a partir de um conjunto de nós selecionados randomicamente.

A equação de distribuição de tarefas para esse algoritmo de balanceamento de carga é idêntica à equação $\{2.10\}$.

2.4.4 Receiver-Initiated

No algoritmo Receiver-Initiated, descrito em [LLC96] e [MD94], um processador P_i decide obter uma fração α do excesso das tarefas de outro processador, quando o número de tarefas atribuídas a ele, $w_i(k)$, é menor que um certo *threshold*, $w_i^*(k)$.

O processador P_e é escolhido randomicamente do conjunto de processadores. Se o número de tarefas atribuídas a P_e, $w_e(k)$, for maior que o valor de *threshold*, $w_e^*(k)$, transfere-se $\alpha(w_e(k) - w_e^*(k))$ tarefas para P_i. Caso contrário, repete-se o processo de seleção de outro nó.

Um elemento da matriz de participação de processadores, $A_{ii}(k)$, é igual a 1 se $(w_i(k) > w_i^*(k))$. Todos os outros elementos de A são nulos. O elemento $M_{ei}(k) = \alpha$ e $M_{ej}(k) = 0$ se $j \neq i$.

Para esse algoritmo, as políticas são:

- de transferência $\rightarrow$ subcarga de um nó do sistema;
- de informação $\rightarrow$ o nó de destino divulga seu número de tarefas para o processador que o selecionou [LLC96];
- de localização $\rightarrow$ seleção randômica do nó de destino.

A equação de distribuição de tarefas para esse algoritmo de balanceamento de carga é idêntica à equação $\{2.10\}$.

2.4.5 Simétrico

O algoritmo Simétrico, descrito em [MD94], é uma combinação das algoritmos Sender-Initiated e Receiver-Initiated. A estratégia de balanceamento de carga é ajustada de acordo com o número de tarefas atribuídas ao nó, permitindo que esse algoritmo escolha automaticamente entre o algoritmo Sender-Initiated, quando o número de tarefas atribuídas ao nó ultrapassa o valor de *threshold* ou o algoritmo Receiver-Initiated, quando o número de tarefas atribuídas ao nó é menor que o valor de *threshold*.

Os três estados apresentados a seguir são mutuamente exclusivos, isto é, um nó do sistema só pode estar em um dos estados citados abaixo:

- sobrecarregado – o número de tarefas atribuídas a ele, $w_i(k)$, é maior que um certo *threshold*, $w_i^*(k)$.
  ação – o processador P_i decide enviar uma fração α do excesso de suas tarefas para outro processador.
  O processador de destino P_e é escolhido randomicamente do conjunto de processadores. Se o número de tarefas atribuídas a P_e, $w_e(k)$, for menor que o

valor de *threshold*, $w_e^*(k)$, transfere-se $\alpha(w_i(k) - w_i^*(k))$ tarefas para P_e. Caso contrário, repete-se o processo de seleção de outro nó.
Nesse caso, um elemento da matriz de migração $M_{ie}(k)$ é igual a α e $M_{ij}(k)$ é igual a 0, se $j \neq e$.
Para esse estado, a política de transferência é a sobrecarga de um nó do sistema.

- subcarregado – o número de tarefas atribuídas a ele, $w_i(k)$, é menor que um certo *threshold*, $w_i^*(k)$.
ação – o processador P_i decide obter uma fração α do excesso das tarefas de outro processador.
O processador P_e é escolhido randomicamente do conjunto de processadores. Se o número de tarefas atribuídas a P_e, $w_e(k)$, for maior que o valor de *threshold*, $w_e^*(k)$, transfere-se $\alpha(w_e(k) - w_e^*(k))$ tarefas para P_i. Caso contrário, repete-se o processo de seleção de outro nó.
Nesse caso, um elemento da matriz de migração $M_{ei}(k)$ é igual a α e $M_{ej}(k)$ é igual a 0, se $j \neq i$.
Para esse estado, a política de transferência é a subcarga de um nó do sistema.

- balanceado – o número de tarefas atribuídas a ele, $w_i(k)$, é igual a um certo *threshold*, $w_i^*(k)$.
ação – o processador P_i não participa do balanceamento de carga.
Nesse caso, $M_{ij}(k) = 0$, para qualquer j.
Para esse estado, a política de transferência determina que nenhuma tarefa será transferida desse nó.

Para os três casos, um elemento da matriz de participação de processadores, $A_{ii}(k)$, é igual a 1 se $(w_i(k) > w_i^*(k))$. Todos os outros elementos de A são nulos.

A política de informação, para esse algoritmo, é a divulgação do número de tarefas do nó de destino para o processador que o selecionou. A política de localização é a seleção randômica do nó de destino.

A equação de distribuição de tarefas para esse algoritmo de balanceamento de carga é idêntica à equação $\{2.10\}$.

2.4.6 APAL

No algoritmo APAL, descrito em [LLC96], um processador P_i decide enviar uma fração α do excesso de suas tarefas para outro processador, quando o número de tarefas atribuídas a ele, $w_i(k)$, é maior que um certo *threshold*, $w_i^*(k)$.

O processador P_i faz uma previsão dos nós sobrecarregados e subcarregados do sistema e, então elege um processador, P_e, do conjunto de processadores.

O procedimento de previsão de estado tem a seguinte estrutura: na iteração k, cada processador participante do sistema divulga seu número de tarefas para os outros componentes. Cada nó, então, armazena essas informações e as usa, na próxima iteração $(k+1)$, para estimar a carga dos outros processadores [LLC96].

Se o número de tarefas atribuídas a P_e, $w_e(k)$, for menor que o valor de *threshold*, $w_e^*(k)$, transfere-se $\alpha(w_i(k) - w_i^*(k))$ tarefas para P_e. Caso contrário, repete-se o processo de eleição de outro nó, até que o número de eleições ultrapasse um valor de eleição pré-definido, E. Se o número de eleições for maior que E, esse algoritmo para até que um nó torne-se subcarregado.

O algoritmo APAL é similar ao algoritmo de balanceamento de carga Threshold. As diferenças entre eles ocorrem na fase de eleição do nó de destino. No Threshold, não existe uma previsão de carga em cada um dos processadores do sistema e uma limitação no processo de seleção do nó. Se todos os processadores estiverem sobrecarregados, esse algoritmo repete, infinitamente, a seleção randômica de um nó do conjunto de processadores do sistema. Já no APAL, a previsão de carga em cada processador é realizada e as eleições ficam limitadas a um valor pré-determinado de eleições. Se nenhum processador estiver subcarregado e se o número de eleições ultrapassar ao referido valor, esse algoritmo espera até que algum nó do sistema fique subcarregado.

Um elemento da matriz de participação de processadores, $A_{ii}(k)$, é igual a 1 se $(w_i(k) > w_i^*(k))$. Todos os outros elementos de A são nulos. O elemento $M_{ie}(k) = \alpha$ e $M_{ij}(k) = 0$ se $j \neq e$.

Para esse algoritmo, as políticas são:

- de transferência $\rightarrow$ sobrecarga de um nó do sistema;
- de informação $\rightarrow$ cada processador participante do sistema divulga o seu número de tarefas para os outros participantes;
- de localização $\rightarrow$ eleição do nó de destino a partir de uma previsão dos nós subcarregados e sobrecarregados do sistema.

A equação de distribuição de tarefas para esse algoritmo de balanceamento de carga é idêntica à equação $\{2.10\}$.

2.4.7 Redistribuição Completa

No algoritmo Redistribuição Completa, descrito em [FG96], quando a carga em qualquer processador i, $w_i(k)$, ultrapassa o valor de *threshold*, $w_i^*(k)$, todas as tarefas do sistema são agrupadas e distribuídas, igualmente, entre todos os nós desse sistema. Portanto, todos os processadores participam do balanceamento de carga.

Os processadores usam um algoritmo de otimização para calcular a matriz $M(k)$. O cálculo dessa matriz resulta em uma distribuição balanceada das tarefas. Por exemplo, todas as tarefas são agrupadas em um processador mestre e, então, redistribuídas igualmente usando um algoritmo ótimo de alocação de tarefas.

Pelo fato de todos os processadores participarem do balanceamento de carga, a matriz de participação do processador, $A(k)$, será uma matriz com os elementos da diagonal iguais a 1 e os outros elementos iguais a 0.

Para esse algoritmo, as políticas são:

- de transferência $\rightarrow$ sobrecarga de um nó do sistema;
- de informação $\rightarrow$ quando qualquer processador estiver sobrecarregado, cada nó participante do sistema divulga seu número de tarefas para os outros processadores;
- de localização $\rightarrow$ quando qualquer processador estiver sobrecarregado, todos os nós do sistema recebem o mesmo número de tarfeas.

A equação de distribuição de tarefas para esse algoritmo de balanceamento de carga é idêntica à equação $\{2.10\}$.

2.4.8 Difusão

No algoritmo Difusão, descrito em [FG96], quando a carga em qualquer processador i, $w_i(k)$, ultrapassa o valor de *threshold*, $w_i^*(k)$, esse nó decide enviar uma fração α do excesso de suas tarefas para seus vizinhos, examinando, antes, a distribuição de tarefas em cada processador vizinho.

Um elemento da matriz de participação de processadores, $A_{ii}(k)$, é igual a 1 se $(w_i(k) > w_i^*(k))$. Todos os outros elementos de A são nulos.

Os destinos das tarefas são calculados baseados na distribuição de tarefas nos processadores vizinhos. P_i envia tarefas para P_j, se o número de tarefas em P_j, $w_j(k)$, é menor que a carga ideal em P_j, $\bar{w}_j(k)$, isto é, P_i envia $\alpha(\bar{w}_j(k) - w_j(k))$ tarefas para P_j. A matriz $M(k)$ é calculada da seguinte maneira:

(1) $M_{ij}(k) = \alpha[\bar{w}_j(k) - w_j(k)]$;

(2) $M_{ij}(k) = 1$.

O primeiro item fornece um sistema de n equações, os elementos $M_{ij}(k)$ para cada j, e $n+1$ incógnitas, todos os elementos $M_{ij}(k)$ e a constante α. O segundo item indica que a soma de todos os elementos da linha i deve ser 1, o que representa o deslocamento de todo o excesso de tarefas do processador P_i para os seus vizinhos. Como se tem um sistema de $n+1$ equações, os ítens (1) e (2), e $n+1$ incógnitas, define-se cada um dos elementos da linha i da matriz $M(k)$.

Para esse algoritmo, as políticas são:

- de transferência $\rightarrow$ sobrecarga de um nó do sistema;
- de informação $\rightarrow$ cada processador participante do sistema divulga o seu número de tarefas para os seus vizinhos;

- de localização → quando qualquer processador estiver sobrecarregado, todos os processadores vizinhos recebem tarefas, levando-se em consideração as suas distrbuições de tarefas.

A equação de distribuição de tarefas para esse algoritmo de balanceamento de carga é idêntica à equação {2.10}.

2.5 Considerações Finais

Os algoritmos de balanceamento de carga adaptativos que têm sido examinados na literatura [ELZ86], [FG96], [LLC96] e [MD94] coletam informações de estado do sistema e, baseados nessas informações, procuram fazer a melhor redistribuição da execução das tarefas de uma aplicação [ELZ86].

Claramente, a complexidade dessa coleta de informações introduz alguns pontos interessantes:

1. o efeito do *overhead* na coleta das informações → o uso de um algoritmo de balanceamento de carga adaptativo depende criticamente do *overhead* requerido para coletar as informações de estado do sistema. Esse *overhead* pode influir diretamente no tempo de execução de uma aplicação submetida ao procedimento de balanceamento. Assim, excessivos *overheads* podem anular os benefícios de uma redistribuição da execução das tarefas de uma aplicação;

2. o efeito de uma redistribuição das tarefas de uma aplicação → algoritmos de balanceamento de carga complexos contam com informações detalhadas a respeito do estado do sistema. Esse tipo de informação gasta muito tempo para ser coletada e não consegue determinar a sobrecarga ou subcarga de um nó num futuro próximo. Devido a isso, uma redistribuição das tarefas de uma aplicação feita por um algoritmo de balanceamento de carga complexo que supõe estar próxima da redistribuição ótima, pode ser, de fato, muito ruim.

A partir dos pontos levantados anteriormente, surge a seguinte questão com relação aos algoritmos de balanceamento de carga adaptativos: qual é o nível desejável de complexidade desses algoritmos? Em [ELZ86], [MD94] e [FG96] foi mostrado que:

- algoritmos de balanceamento de carga adaptativos simples – algoritmos com pouca ou nenhuma coleta de informações de estado, como Threshold e Random, respectivamente – produzem uma melhoria no desempenho em relação ao caso onde não há a redistribuição das tarefas;
- algoritmos de balanceamento de carga adaptativos simples, como Threshold e Random, têm o desempenho melhor que algoritmos de balanceamento de carga que coletam uma grande quantidade de informação de estado do sistema e que tentam fazer a "melhor" redistribuição da execução, como Shortest, Difusão e Redistribuição Completa;
- esses resultados são válidos sobre um grande intervalo de parâmetros do sistema, como, por exemplo, o seu tamanho.

Em [ELZ86] conclui-se que algoritmos de balanceamento de carga adaptativos simples são mais interessantes do ponto de vista prático e, de modo geral, não existem evidências de que custos de coletar e usar um volume grande de informações de estado tragam algum benefício real. Entretanto, muitos algoritmos de balanceamento de carga adaptativos complexos têm sido propostos na literatura [MD94] e [LLC96], visando a obtenção do mesmo tempo de execução dos algoritmos simples e, além disso, uma melhor redistribuição das tarefas de uma aplicação.

Capítulo 3

O Problema dos N-corpos

Técnicas clássicas de N-corpos estudam a evolução de um sistema de n partículas (ou corpos) onde existe uma força (por exemplo, a força gravitacional) entre pares de partículas. Para os N-corpos, a avaliação direta de todas as forças entre os pares de partículas resulta em uma complexidade média $O(n^2)$. Métodos hierárquicos conseguem reduzir a complexidade média para $O(n \ log \ n)$. Esses métodos são baseados no seguinte princípio de Isaac Newton: "se a magnitude de interação entre corpos cai rapidamente com a distância, então o efeito de um aglomerado de corpos sobre uma partícula deve ser aproximado ao efeito de um simples corpo, se o grupo de corpos estiver longe o bastante dessa partícula".

A simulação sequencial discutida nesse trabalho está baseada no algoritmo de Barnes-Hut, descrito em [CT92], [BH86] e [Vel94], para estudar a evolução de um sistema de corpos sob a influência da atração gravitacional Newtoniana. Os corpos consistem de uma massa, posição e velocidades iniciais e eles estão distribuídos sobre um domínio físico finito. A simulação segue sobre um número de iterações, onde, em cada iteração, há o cálculo das forças em cada corpo e há a atualização de sua posição e de outros atributos.

Na simulação paralela do problema dos N-corpos com o algoritmo de Barnes-Hut, o domínio físico é dividido em pequenas regiões e cada região é alocada a um processador. Cada processador é responsável por todos os cálculos associados com as partículas de sua região. No entanto, para calcular as forças em cada partícula, é necessário conhecer a posição de outras partículas e isso envolve troca de informação entre pares de processadores. Depois da troca de informações, cada processador calcula a força resultante em cada partícula de sua região e atualiza suas posições e velocidades.

Na seção 3.1 o algoritmo de Barnes-Hut é apresentado. Na seção 3.2, é mostrada a simulação sequencial do problema dos N-corpos. Na seção 3.3, é explicada a simulação paralela. Na seção 3.4, é apresentado um estudo do desempenho do algoritmo paralelo.

3.1 O Algoritmo de Barnes-Hut

A figura 3.1 mostra uma coleção de n partículas, ou corpos, randomicamente distribuídos em um quadrado, ou um espaço dimensional. Cada partícula é descrita por um conjunto: massa, posição e velocidade. As partículas estão sujeitas a uma força em virtude das outras partículas no espaço. O objetivo é encontrar algoritmos que simulem o movimento das partículas no tempo.

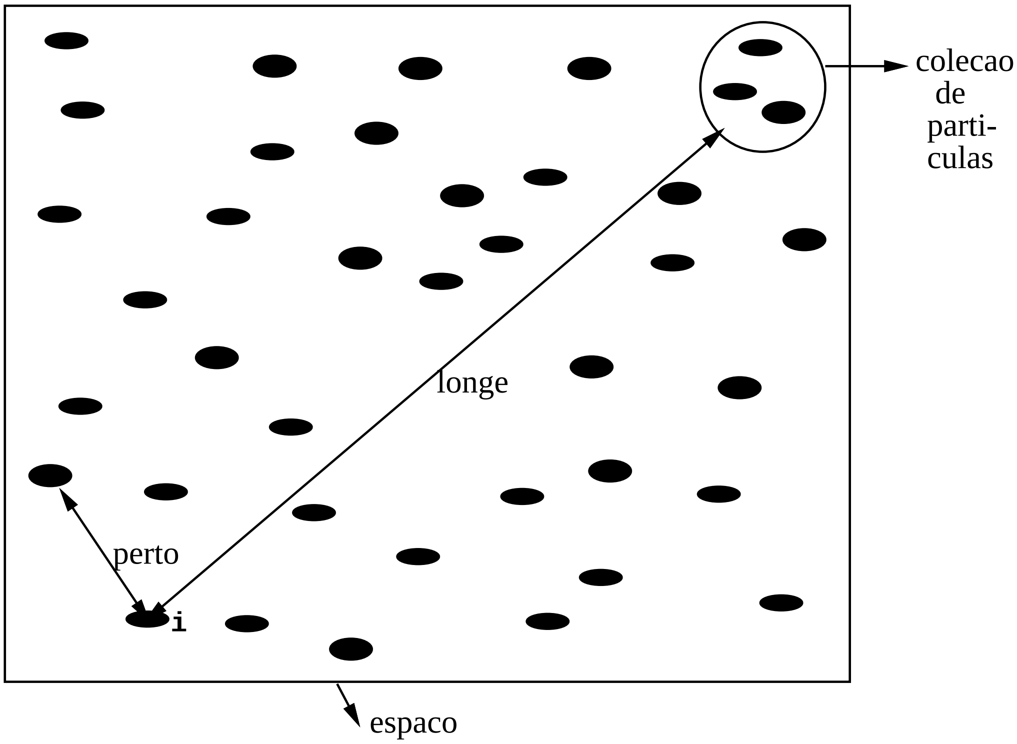

Figura 3.1: Espaço de Partículas

Um método simples é o de calcular o movimento de uma partícula, pelo acúmulo dos efeitos das forças entre todos os pares de partículas. Para cada par de partículas, a força é calculada usando as leis de gravidade de Newton. A aceleração de uma partícula i, a_i, devido a uma partícula j é determinada pela equação:

$$a_i = Gm_j \;/\; r^2$$

onde G é a constante gravitacional universal, m_j é a massa da partícula j e r é o vetor distância de i a j.

Dada a aceleração da partícula i em tempo t, a_i^t, as novas velocidade v e posição

p, no tempo $t + dt$ são calculadas usando as equações:

$$v_i^{t+dt} = v_i^t + dta_i^t$$

$$p_i^{t+dt} = p_i^t + dtv_i^t$$

O cálculo resultante das forças tem complexidade média $O(n^2)$ já que cada partícula é afetada pelo movimento das outras partículas.

O algoritmo de Barnes-Hut reduz a complexidade média para $O(n \; log \; n)$ usando um princípio mostrado na figura 3.1: um grupo de corpos que está longe de uma partícula i pode ser modelado usando uma única partícula j cuja massa é a massa total do grupo de corpos e cuja posição é o centro de massa do grupo.

Em cada passo da simulação, uma árvore quaternária é construída usando as partículas do passo anterior. Inicialmente, a árvore está vazia (isto é, não contém partículas). Como as partículas são adicionadas à árvore, o espaço do problema é recursivamente dividido em quatro partes e essa divisão é feita até que não mais de uma partícula esteja presente em cada divisão do espaço:

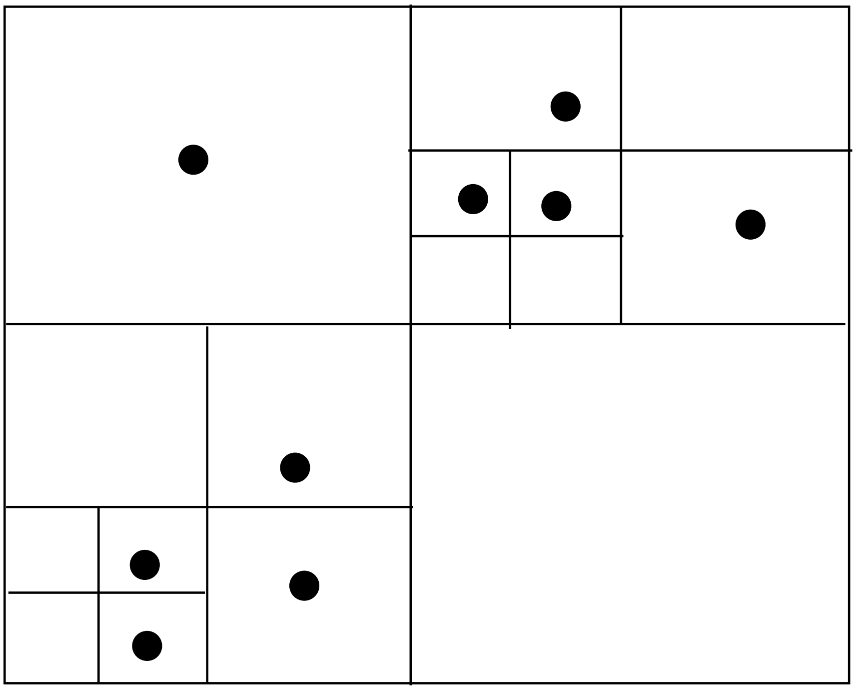

Figura 3.2: Espaço de Solução

Como as partículas são adicionadas na árvore, nós intermediários são usados para acumular a massa total e o centro de massa de um grupo de partículas, como mostra a figura 3.3:

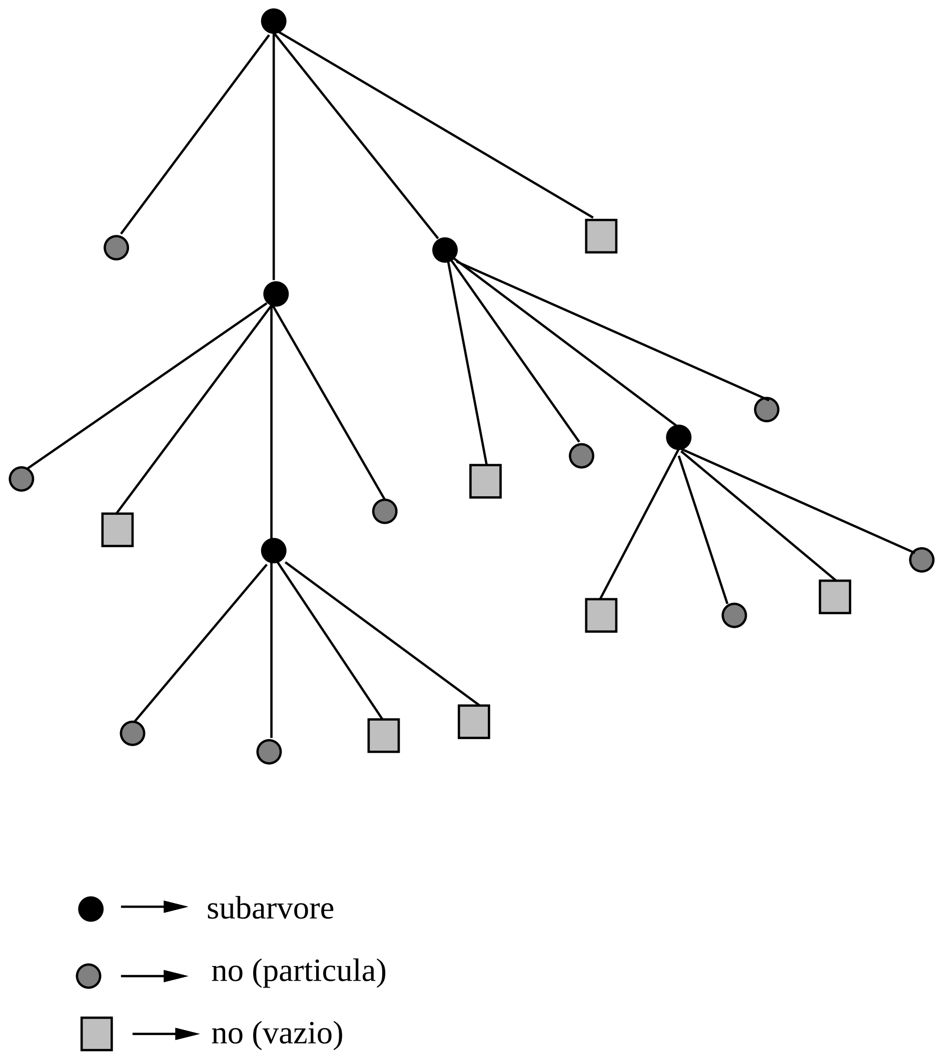

Figura 3.3: Árvore de Solução

Depois que a árvore é construída, a aceleração de uma partícula é calculada descendo na árvore e aplicando o critério de precisão em cada nó. Se um nó provê uma aproximação suficientemente exata dos efeitos em uma partícula, então a aceleração é calculada pela interação do nó e da partícula. Caso contrário, a contribuição do nó é calculada pela somatória das contribuições dos quatro filhos do nó:

```
para cada iteracao k
  arvore = construir_arvore (particulas);
  para cada particula i em k, i_k
```

```
    se (i_k esta_longe_da_arvore) entao
      ac = calcular_efeitos (i, arvore);
    senao
      ac = somatoria (j, ac (filho (j, arvore));
    fim-se;
  fim-para;
fim-para.
```

3.2 Simulação Sequencial do Problema dos N-corpos

Como foi mencionado anteriormente, o algoritmo de Barnes-Hut usa uma abordagem hierárquica e resulta em uma complexidade média $O(n \log n)$. Para conseguir esse valor de complexidade, toda a informação a respeito das partículas é representada na forma de árvore quaternária. Pode-se, contudo, usar uma árvore binária, no lugar da árvore quaternária, para representar as partículas, sem aumentar a complexidade computacional do algoritmo. Com essa mudança, o procedimento *Orthogonal Recursive Bisection* (ORB), responsável pela formação de uma árvore binária de corpos, pode ser usado.

A simulação sequencial discutida nesse trabalho, descrita em [FG93], utiliza árvores binárias. A árvore binária é formada usando o ORB. O procedimento inicia na raiz da árvore binária, que representa o domínio e todas as partículas contidas nele. Os níveis sucessivos da árvore são formados recursivamente pela divisão do domínio em dois subdomínios retangulares (com o número igual de partículas), até que os subdomínios finais contenham somente uma partícula. Na estrutura da árvore resultante, as folhas representam as partículas e os nós intermediários representam grupos de partículas.

O passo seguinte associa com cada nó, o centro de massa e a massa equivalente das partículas contidas na sub-região do nó. Isso é feito pela propagação da informação das folhas para os nós "pais", e assim sucessivamente, até a raiz da árvore. Depois desse passo, as folhas passam a conter as informações a respeito das partículas e os nós internos passam a conter informações sobre os grupos de partículas.

Depois da propagação das informações, a árvore pode ser usada para calcular a força resultante em cada partícula. O algoritmo que calcula a força que age em cada partícula começa na raiz da árvore e segue até chegar nas folhas. Em cada nó interno, o cálculo é feito para determinar se o agregado de partículas, representado pelo nó, pode ser considerado com um único nó. Isso é feito calculando a razão entre o tamanho do sub-domínio que está sendo processado, $tam(no)$, e a distância entre a partícula e o centro de massa do sub-domínio, $dist(no, corpo)$. Se o sub-domínio está muito longe da partícula (isto é, se $tam(no)$ / $dist(no, corpo) \leq \theta$, onde θ é um parâmetro fixo de precisão), a força Newtoniana entre a partícula e o sub-domínio é calculada. Caso contrário, o algoritmo é aplicado recursivamente para cada filho do nó atual:

```
calcular_forca (no, corpo)
inicio
  se ((tam (no) / dist (no, corpo) <= teta) ou (no e_folha))
    forca = forca + newtoniana (no, corpo);
  senao
    para cada filho do no faca
      calcular_forca (filho, corpo);
    fim-para;
  fim-se;
fim.
```

Depois do cálculo das forças, as posições e as velocidades das partículas são atualizadas. Dado que os corpos têm novas posições, uma nova árvore deve ser formada e todo o procedimento é repetido para a nova iteração. A simulação segue por um número pré-determinado de iterações, como mostra o algoritmo a seguir:

```
simulacao_sequencial ()
inicio
  inicializar_particulas (massa, posicao, velocidade);
  para cada iteracao faca
  inicio
    construir_arvore_binaria ();
    calcular_para_cada_no (massa, centro_massa);
```

```
    para cada particula faca
    inicio
      calcular_forca (no, particula);
      atualizar (posicao, velocidade);
    fim-para;
  fim-para;
fim.
```

3.3 Implementação Paralela

Utilizou-se nesse trabalho a simulação paralela do problema dos N-corpos descrita em [FG93].

Duas questões importantes na implementação paralela da simulação sequencial, discutida na seção 3.2, são:

- particionar o problema em módulos e mapear efetivamente esses módulos aos processadores: para o problema dos N-corpos, o domínio físico é quebrado em sub-domínios e cada sub-domínio é associado a um processador. Os processadores são responsáveis por todos os cálculos associados com as partículas contidas no seu sub-domínio;
- impor uma estrutura no padrão de comunicação tal que o número e o tamanho das mensagens sejam minimizados: no problema dos N-corpos, o cálculo da força resultante, em uma partícula, requer que a massa e a posição de todas as outras partículas sejam conhecidas. Quando as partículas são distribuídas sobre todos os processadores, a troca de informação é exigida entre cada par de processadores, resultando em muita comunicação desestruturada.

A decomposição do domínio físico, usando o algoritmo ORB, discutido na seção anterior, garante um balanceamento de carga aproximado, enquanto provê uma estrutura de comunicação eficiente. O domínio físico é dividido, recursivamente, em dois sub-domínios retangulares (com cargas iguais) e metade dos processadores são atribuídos a cada sub-domínio. O procedimento se repete até que haja um processador associado com cada subdomínio retangular. No final da fase de particionamento, o domínio físico está decomposto em p subdomínios, onde p é o número de processadores disponíveis. A decomposição ORB resulta em $log_2 p$ bifurcações espaciais e as folhas da árvore representam os sub-domínios do processador:

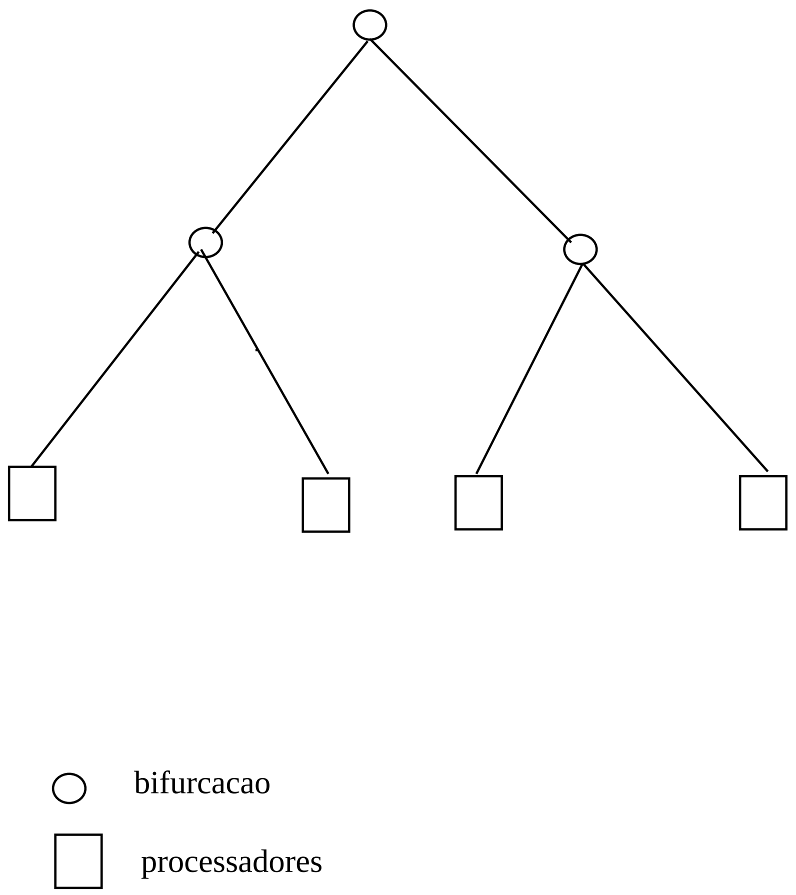

Figura 3.4: Árvore ORB

Cada processador tem um domínio espacial contíguo e é responsável por todos os cálculos associados com as partículas presentes no seu sub-domínio. Os processadores fazem a bifurcação recursiva do seu sub-domínio e representam suas partículas na forma de árvore binária. A informação dos nós intermediários, o centro de massa e a massa equivalente dos seus sub-domínios, são calculados através das informações enviadas das folhas até a raiz da árvore local.

Para calcular o centro de massa e a massa equivalente do domínio físico completo, os processadores necessitam trocar informação do centro de massa e massa equivalente dos seus respectivos domínios. A informação é passada das folhas (no caso, os domínios dos processadores) para a raiz, onde o centro de massa e a massa equivalente do domínio total é calculada e, então, passada para as folhas da árvore ORB, a fim de que todos os processadores tenham essa informação. Essa fase pode ser vista como uma fase de sincronização implícita do algoritmo, já que todos os processadores têm de "cooperar" para o cálculo da massa total e do centro de massa do sistema.

A árvore ORB associada com as partículas locais do processador e com as informações relevantes das partículas de outros processadores é chamada de *Locally Essential Tree* (le-tree). O algoritmo que simula a le-tree é mostrado a seguir:

```
construir_le-tree ()
inicio
  para cada uma das log2 p bifurcacoes
  inicio
    atravessar a arvore e identificar dados que podem ser
    essenciais as outras bifurcacoes;

    destruir dados que nao sao necessarios a essa bifurcacao;

    trocar dados com os processadores correspondentes as outras
    bifurcacoes;

    fundir os dados recebidos na arvore;
  fim-para;
fim.
```

A formação da le-tree envolve troca de informação entre pares de processadores. A decomposição ORB impõe uma estrutura hierárquica e isso, aparentemente, exige comunicações "desestruturadas". O algoritmo le-tree reduz a quantidade de informação transferida (e, portanto, reduz o tamanho da mensagem), por enviar somente o identificador do remetente e os dados para os processadores vizinhos. Um processador vizinho ao processador P_i é o processador cujo domínio físico faz fronteira o de com P_i. Cada processador envia e recebe $log_2 p$ mensagens, ao invés de p mensagens, no caso em que todos os processadores trocam mensagens.

Depois que a le-tree está formada, os processadores têm toda a informação de que necessitam para calcular a força resultante nas partículas do seu domínio. O cálculo da força resultante é essencialmente local, sem qualquer comunicação, usando uma função recursiva similar ao algoritmo de cálculo da força, mostrado na seção anterior. A velocidade e o deslocamento das partículas são atualizados. O processador checa, para suas partículas, quem atravessou o seu domínio e, então, transfere as partículas que não pertencem mais a ele para os processadores apropriados.

Essa transferência causa um desequilíbrio na distribuição de carga entre os processadores. O algoritmo de balanceamento de carga é invocado ao fim de cada iteração, mas só é executado se o desequilíbrio da carga excede ao *threshold* específico. O balanceamento de carga consiste em redesignar domínios para cada processador, tal que o número de partículas associadas aos processadores esteja balanceado. A redesignação de domínios é, basicamente, uma operação de transferência de tarefas. Isto é, se as partículas atravessarem o novo domínio designado a um processador, elas serão transferidas para o processador adequado.

O seguinte procedimento resume a simulação paralela do problema dos N-corpos:

```
simulacao_paralela ()
inicio
  inicializar_particulas (massa, posicao, velocidade);
  distribuir_particulas ();
  para cada iteracao, em cada processador, faca
  inicio
    construir ORB_tree_completa (raiz);
    calcular_para_cada_no (massa, centro_massa);
    trocar_centros_massa ();  /* fase de sincronizacao */
    construir_le-tree ();
    para cada particula faca
    inicio
      calcular_forca (raiz, particula);
      atualizar (posicao, velocidade);
    fim-para;
    enviar_particulas_fujonas ();
    receber_particulas_fujonas ();
```

```
    invocar_balanceamento_carga ();
  fim-para;
fim.
```

3.4 Modelo de Desempenho

O modelo de desempenho é um modelo analítico em que as equações matemáticas deduzidas representam o desempenho de um algoritmo em estudo. Nessa seção, descreve-se um modelo de desempenho geral, proposto em [FG93], que pode ser adaptado para estudar o desempenho de qualquer algoritmo síncrono.

Algoritmos síncronos geralmente possuem a seguinte heurística: a execução segue por um número fixo de iterações e, o trabalho associado com cada iteração é distribuído nos processadores participantes do sistema. No fim de cada iteração, os processadores fazem uma sincronização e, então, iniciam a próxima iteração.

O tempo gasto em cada iteração pode ser dividido em três fases: computação, comunicação e sincronização. Na fase de computação, cada processador avalia um conjunto de variáveis locais através do uso de sua memória local não-compartilhada. Na fase de comunicação, cada processador envia suas informações locais para os outros nós do sistema e recebe informação similar dos demais participantes. Finalmente, na fase de sincronização, cada processador que terminou uma iteração espera os outros nós do sistema terminarem-na, a fim de garantir que todos os processadores completaram a $(k-1)$-ésima iteração, antes que qualquer nó inicie a iteração k.

A fase de sincronização, em um sistema síncrono, obriga os processadores que estão subcarregados esperarem, em cada iteração, pelos sobrecarregados. Portanto, o tempo gasto nessa fase, T_{sync}, é o mesmo para qualquer nó do sistema.

O tempo gasto em cada iteração, T_{iter}, é a soma do tempo de sincronização e o máximo, entre todos os p processadores, da soma do tempo de computação e de comunicação em cada processador:

$$T_{iter} = max_{i=0}^{p-1}[T_{comp}(i) + T_{comm}(i)] + T_{sync} \qquad \{3.1\}$$

T_{comp} e T_{comm} indicam o tempo gasto na fase de computação e de comunicação, respectivamente.

3.4.1 Modelo de Desempenho para a Simulação Paralela do Problema dos N-Corpos

Para adaptar o modelo de desempenho desenvolvido na seção 3.4 à simulação paralela do problema dos N-corpos, assume-se que os processadores do sistema possuem poderes computacioniais iguais.

Os custos de inicialização do algoritmo são considerados desprezíveis. Os vários passos envolvidos em cada iteração podem ser agrupados nas três fases anteriormente descritas – sincronização, comunicação e computação. Cada iteração pode ser dividida em cinco passos:

1. trocar/propagar o centro de massa local – fase de sincronização $\rightarrow$ tempo: T_{sync};
2. formar a árvore local de cada processador – fase de comunicação $\rightarrow$ tempo: T_{le_tree};
3. calcular forças resultantes – parte da fase de computação $\rightarrow$ tempo: T_{force};
4. atualizar os atributos das partículas – parte da fase de computação $\rightarrow$ tempo: T_{update};
5. formar a árvore com as partículas locais – parte da fase de computação $\rightarrow$ tempo: T_{local}.

A equação {3.1} pode ser reescrita como:

$$T_{iter} = max_{i=0}^{p-1}[T_{le\_tree}(i) + T_{force}(i) + T_{update}(i) + T_{local}(i)] + T_{sync} \qquad \{3.2\}$$

Na simulação paralela, descrita na seção 3.3, pode ocorrer transferência de partículas entre os processadores participantes do sistema. Essa transferência de partículas causa um desequilíbrio na carga computacional dos processadores.

O desbalanço no número de partículas aumenta o tempo consumido no cálculo da força resultante em cada partícula. Para os outros passos, o desequilíbrio da carga produz poucas variações nos seus tempos de execução. Portanto, T_{local}, T_{le_tree} e T_{update} podem ser aproximados para os seus respectivos valores médios. Assim:

$$T_{iter} = max_{i=0}^{p-1}[T_{force}(i)] + T_{le\_tree} + T_{update} + T_{local} + T_{sync} \qquad \{3.3\}$$

Um modo conveniente de lidar com o termo $max_{i=0}^{p-1}[T_{force}(i)]$ é definir um fator de desequilíbrio, β, como sendo a razão entre o tempo máximo de cálculo da força e o seu tempo médio:

$$\beta = max_{i=0}^{p-1}[T_{force}(i)]/T_{force} \qquad \{3.4\}$$

A equação $\{3.3\}$ pode ser reescrita como:

$$T_{iter} = \beta T_{force} + T_{le\_tree} + T_{update} + T_{local} + T_{sync} \qquad \{3.5\}$$

3.4.1.1 Sincronização

Na fase de sincronização, os processadores trocam a massa e o centro de massa de seus domínios para calcular a massa total e o centro de massa do sistema. O procedimento de sincronização faz uso da estrutura ORB para calcular e distribuir a informação global. As folhas da árvore ORB enviam as informações a respeito de seu domínio para os seus "pais". Os "pais" calculam a massa total e o centro de massa de seus "filhos" e passam as informações para seus "pais". Esse envio de informações é feito até chegar a "raiz" da árvore ORB. A "raiz" calcula a informação global e passa essa informação para o resto da árvore.

O procedimento de sincronização envia $2log_2p$ mensagens para os log_2p níveis da árvore. Assim:

$$T_{sync} = 2\ log_2p\ T_{msg}(s) \qquad \{3.6\}$$

Onde $T_{msg}(s)$ é o tempo de transmissão de uma mensagem de s bytes.

3.4.1.2 Comunicação

Na fase de comunicação, cada processador identifica quais de seus dados são necessários aos outros processadores participantes do sistema e trocam esses dados com eles.

A primeira troca envolve $(\bar{n}N_{byte})$ bytes onde $\bar{n}$ é o número médio de partículas em um processador e N_{byte} é o número de bytes usados para representar um partícula. Nas outras permutas, os processadores enviam seus dados e os dados adquiridos nas trocas anteriores. Portanto, a k-ésima permuta envolve $(2^k\bar{n})$ partículas.

É razoável admitir que o tempo de busca na árvore, identificando os dados a serem permutados, é proporcional ao número de partículas. Usa-se t_{search_p} e t_{merge_p} para denotar o tempo de busca por partícula e o tempo de fusão de uma partícula recebida na árvore, respectivamente. Então:

$$T_{le\_tree} = \sum_{k=0}^{(log_2p-1)}[\textit{tempo de busca} + \textit{tempo de transmissão de uma mensagem} + \textit{tempo de fusão}] \qquad \{3.7\}$$

Considerando que:

$tempo\ de\ busca = 2^k n t_{search\_p}$

$tempo\ de\ transmissão\ de\ uma\ mensagem = 2^k \bar{n} N_{byte} T_{msg}$

$tempo\ de\ fusão = 2^k n t_{merge\_p}$

Tem-se que:

$$T_{le\_tree} = \sum_{k=0}^{(log_2 p - 1)} [(t_{search\_p} + t_{merge\_p}) 2^k \bar{n} + T_{msg}(2^k \bar{n} N_{byte})] \qquad \{3.8\}$$

Os parâmetros t_{search_p} e t_{merge_p} são dependentes dos poderes computacionais dos nós participantes do sistema.

3.4.1.3 Computação

Na fase de computação, cada processador calcula a força resultante em cada partícula de seu domínio, atualiza os atributos de suas partículas e verifica quais delas atravessaram o seu domínio e, então, transfere as partículas que não pertencem mais a ele para os processadores adequados. Além dessas operações, essa fase é responsável, também, por formar a árvore de partículas locais.

A transferência de partículas causa um desbalanço na carga computacional de cada processador. Como o cálculo da força resultante domina essa fase do algoritmo, é razoável admitir que o desbalanço nos outros passos computacionais dessa fase sejam desprezíveis. O desbalanço na avaliação da força resultante reflete no parâmetro β.

Assim, o tempo dessa fase, para cada iteração é dado por:

$$T_{comp} = \beta T_{force} + T_{le\_tree} + T_{update} + T_{local} + T_{sync} \qquad \{3.9\}$$

O passo de cálculo da força é o que mais consome tempo nessa fase do algoritmo. Para cada partícula, a árvore ORB é visitada a partir da raiz e a força, devida a outras partículas (ou grupo de partículas), é calculada.

Devido a natureza hierárquica do algoritmo, cada partícula tem, em média, $O(log$ $n)$ avaliações de forças associadas a ela. Sendo o número médio de avaliações por partícula igual a $K_{eval} log_2 N$, onde N é o número total de partículas e K_{eval} é uma constante que depende do tipo de distribuição de partículas usada e do parâmtero θ, o tempo consumido no cálculo da força é:

$$T_{force} = \bar{n} K_{eval} log_2 N t_{eval} \qquad \{3.10\}$$

Onde t_{eval} é o tempo de cada avaliação de força.

O passo de atualização faz a correção da posição e da velocidade de cada partícula do processador, baseada na força resultante em cada partícula. O tempo gasto nesse passo é proporcional ao número de partículas de cada processador:

$$T_{update} = \bar{n} t_{update\_p} \qquad \{3.11\}$$

Onde t_{update_p} é o tempo consumido em cada atualização.

O passo de gerência das partículas locais forma uma nova árvore no processador contendo, somente, suas partículas. Essa operação é essencialmente $O(n\ logn)$, onde

n é o número de partículas locais. O tempo gasto nesse passo é dado por:

$$T_{local} = \bar{n} log_2 \bar{n} t_{local\_p} \qquad \{3.12\}$$

Onde t_{local_p} é o tempo consumido na fusão de uma partícula na árvore local.

Os parâmetros t_{eval}, t_{update_p} e t_{local_p} são dependentes dos poderes computacionais dos processadores participantes do sistema.

Capítulo 4

Biblioteca de Balanceamento

Os procedimentos de balanceamento de carga aplicados à simulação paralela do problema dos N-corpos têm um comportamento especial: devem ser encarados como uma operação de redesignação de domínios para os processadores participantes do sistema. Isto é, dependendo do valor de threshold, os processadores mudarão os seus domínios e, com essa mudança, haverá transferência de partículas, com a finalidade de equilibrar a distribuição da carga.

Analisando o balanceamento de carga para o problema dos N-corpos pode-se notar que as diversas políticas de balanceamento de carga apresentadas executam as mesmas operações, com diferentes níveis de complexidade. É possível, portanto, padronizar essas operações, a fim de se representar um grande intervalo de políticas de balanceamento.

O objetivo desse capítulo é apresentar uma biblioteca de balanceamento de carga, validada pela simulação paralela do problema dos N-corpos, para ser usada em ambientes distribuídos. Essa biblioteca é desenvolvida para funcionar independentemente do algoritmo de balanceamento de carga utilizado. Deve-se observar que algumas operações têm implementações especiais para cada algoritmo de balanceamento de carga.

Na seção 4.1, o algoritmo de balanceamento de carga geral, baseado no Modelo Iterativo de Matriz, será apresentado. Na seção 4.2 são tratadas as operações que têm implementações comuns a todas as políticas de balanceamento de carga usadas, em uma biblioteca chamada de Biblioteca Global. Na seção 4.3 são tratadas todas as operações que têm implementações especiais para cada política de balanceamento de carga utilizada, em uma biblioteca chamada de Biblioteca Específica do Balanceamento de Carga. Na seção 4.4 são tratadas todas as operações que têm implementações dependentes da simulação paralela do problema dos N-corpos (ou seja, dependentes da aplicação), em uma biblioteca chamada de Biblioteca Específica da Aplicação.

4.1 Balanceamento de Carga no Modelo Iterativo de Matriz

Nessa seção, o algoritmo de balanceamento de carga geral, desenvolvido a partir dos conceitos do Modelo Iterativo de Matriz, será mostrado. Antes, porém, de se apresentar esse algoritmo, é importante lembrar alguns dos conceitos do modelo.

O conjunto de processadores participantes do sistema é denotado por P, onde $P = \{P_1, P_2, ..., P_p\}$ e p é o número de processadores. O algoritmo de balanceamento de carga é invocado em instantes de tempo denotados por k ($k = 1, 2, ...$). A participação do processador no balanceamento de carga em k é representado pela matriz $A(k)$. Se o processador P_i decide participar do balanceamento de carga, o elemento $A_{ii}(k)$ recebe o valor 1; todos os outros elementos recebem o valor 0.

Cada processador que decide participar do balanceamento de carga tem de tomar decisões de migração de tarefas. As decisões de migração são representadas por uma matriz $p \times p$, $M(k)$, onde $M_{ij}(k)$ indica a quantidade de tarefas que o processador P_i decide enviar para P_j. As decisões de migração de tarefas ficam necessariamente restritas à estrutura imposta pelo algoritmo de balanceamento de carga ou pela topologia do sistema. Define-se $H(k)$ como a matriz de topologia. $H(k)$ é uma matriz $p \times p$ que descreve os caminhos de migração possíveis. O elemento $H_{ij}(k) = 1$ se P_i e P_j trocam tarefas. Caso contrário, $H_{ij}(k) = 0$. Portanto, na construção de $M(k)$, $M_{ij}(k) \neq 0$ se $H_{ij}(k) = 1$.

Um algoritmo de balanceamento de carga geral, que usa os conceitos descritos anteriormente, seria:

```
LoadBalancing (root, mytree, nbody, myparticles)
inicio
  para todos os processadores do sistema, faca
  inicio
    MatrixA (A, myparticles, threshold);
    MatrixH (root, mytree, H);
    MatrixM (A, H, M);
    ExchangeLoad (A, M, root, mytree, nbody, neighbors);
  fim-para;
fim.
```

Os três primeiros procedimentos tratam das matrizes $A(k)$, $H(k)$ e $M(k)$, respectivamente. A última função, $ExchangeLoad$, faz a redistribuição de carga em cada processador.

É necessário enfatizar que todos os algoritmos de balanceamento de carga que podem ser descritos pelo Modelo Iterativo de Matriz têm a mesma descrição do algoritmo geral. Para esses algoritmos, as implementações de $MatrixA$, $MatrixH$ e $ExchangeLoad$ são idênticas. Já a implementação de $MatrixM$ é especial para cada algoritmo de balanceamento.

4.2 Biblioteca Global

A Biblioteca Global permite a comunicação entre nós participantes do sistema, além do envio de cargas através da rede. Essa biblioteca é composta de operações que têm implementações comuns a uma grande variedade de algoritmos de balanceamento de carga, usadas para implementar um algoritmo geral de balanceamento. A partir desse algoritmo, são implementados alguns algoritmos de balanceamento de carga, a saber: Random, Threshold e Shortest.

Nas subseções seguintes, cada uma dessas operações é apresentada.

4.2.1 MatrixH

O procedimento para calcular a matriz de topologia é mostrado abaixo:

```
MatrixH (root, mytree, H)
inicio
  FindNeighbors (root, mytree);
  para todos os processadores, Pj, faca
  inicio
    se Pj e vizinho de Pi, entao Hij = Hji = 1;
    senao Hij = 0;
  fim-para;
fim.
```

A heurística do algoritmo é a seguinte: se o processador P_i troca tarefas com o processador P_j, os elementos da matriz de topologia $H_{ij}(k)$ e $H_{ji}(k)$ receberão o valor 1. Os outros elementos receberão o valor 0.

A função $FindNeighbors$ é usada para encontrar quais são os vizinhos de um processador.

Toda vez que o algoritmo de balanceamento de carga é invocado, ocorrem mudanças nas fronteiras dos processadores, isto é, os processadores podem aumentar ou diminuir os seus domínios. Pode acontecer que, com essas mudanças, os vizinhos de um processador se alterem. Por esse motivo, em cada iteração, cada processador deve descobrir quais são os seus vizinhos.

Uma descrição de alto nível dessa função seria:

```
FindNeighbors (root, mytree)
inicio
 se so existem dois processadores no sistema, P0 e P1, entao
  P0 e vizinho de P1 e vice-versa;

 senao
```

```
 se o processador esta no canto do dominio fisico, entao
  FindTwoNeighbors (xmin, xmax, ymin, ymax, root, neighbors);

 senao
 se o processador tem uma aresta do dominio fisico, entao
  FindThreeNeighbors (xmin, xmax, ymin, ymax, root, neighbors);

 senao
  FindFourNeighbors (xmin, xmax, ymin, ymax, root, neighbors);
fim.
```

A função *FindTwoNeighbors* trata os processadores que estão localizados nos "cantos" do domínio físico, isto é:

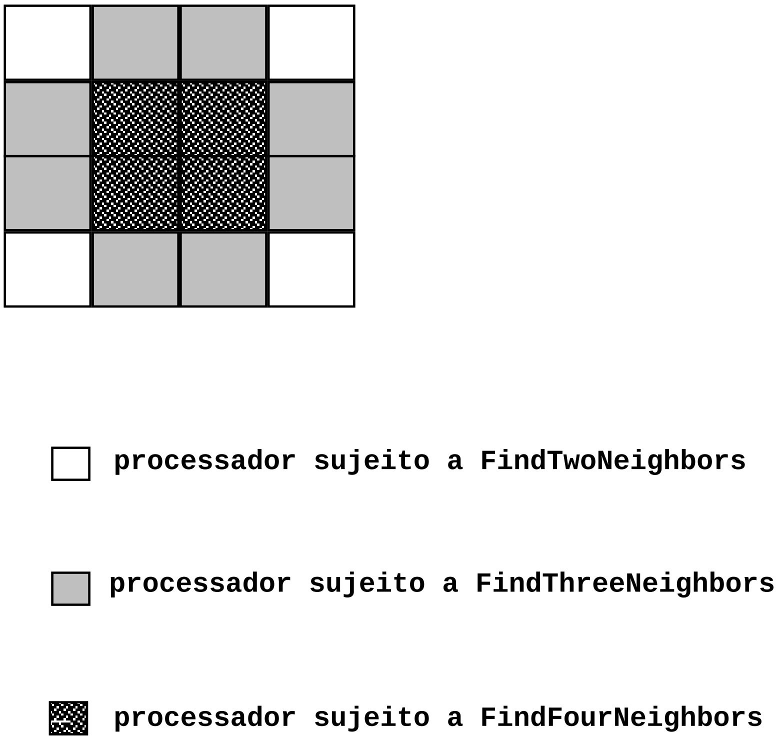

Figura 4.1: Processadores no domínio físico

Esses processadores só podem ter dois vizinhos e a função para encontrar os seus dois vizinhos, $FindTwoNeighbors$, é descrita como:

```
FindTwoNeighbors (xmin, xmax, ymin, ymax, root, neighbors)
inicio
  criacao de uma particula;
  atualizacao das coordenadas x e y da particula;
  vizinho[1] = findnode (root, body, 0);
  atualizacao das coordenadas x e y da particula;
  vizinho[2] = findnode (root, body, 0);
fim.
```

O procedimento $findnode$ descobre a qual processador que a partícula, passada como parâmetro, deve pertencer. Esse procedimento está implementado na simulação paralela dos N-corpos.

Os procedimentos $FindThreeNeighbors$ e $FindFourNeighbors$ são análogos ao procedimento $FindTwoNeighbors$.

4.2.2 MatrixA

Um algoritmo que faz o tratamento à matriz $A(k)$ é apresentado a seguir:

```
MatrixA (A, myparticles, threshold)
inicio
  ModifyElementAii (A, myparticles, threshold);
fim.
```

A implementação de $ModifyElementAii$ é usada por um grupo de algoritmos de balanceamento de carga, que inclui os algoritmos Random, Threshold e Shortest, para calcular quais os processadores que participam do balanceamento de carga. Por se tratar de uma implementação dependente do algoritmo de balanceamento, ela será detalhada na seção 4.3.

4.2.3 ExchangeLoad

O seguinte algoritmo implementa a redistribuição de carga em um processador:

```
ExchangeLoad (A, M, root, mytree, nbody, neighbors)
inicio
  ReassignDomain (A, M, root, mytree, nbody, neighbors);
fim.
```

A heurística do algoritmo é a seguinte: faça a redistribuição de domínios para todos os processadores participantes do sistema.

A implementação de *ReassignDomain* é usada para modificar o domínio de cada processador que envia/recebe partículas. Por se tratar de uma implementação dependente da aplicação, ela será detalhada na seção 4.4.

4.3 Biblioteca Específica de Redistribuição de Tarefas

Como foi mencionado no início desse capítulo, a biblioteca de balanceamento é desenvolvida para funcionar independentemente do algoritmo de balanceamento de carga usado. Deve-se observar que algumas das operações têm implementações especiais para cada algoritmo de balanceamento. Essas operações devem ser apresentadas em uma biblioteca diferente da Biblioteca Global. Elas são descritas na Biblioteca Específica do Balanceamento de Carga.

Nas subseções seguintes, cada uma das operações é apresentada.

4.3.1 MatrixM

A implementação de *MatrixM* é especial para cada algoritmo de balanceamento porque ela descobre, quando um processador está sobrecarregado, o vizinho que receberá o excesso de suas tarefas (política de localização). Como a escolha do destino das tarefas é diferente para cada algoritmo de balanceamento de carga, essa implementação é dependente do algoritmo de balanceamento.

Para o algoritmo Random, a descrição dessa função seria:

```
MatrixM (type, A, H, M)
inicio
  choose = ChooseRandomNeighbor (A, H);
  ModifyElementMij (A, M, choose);
fim.
```

A função $ChooseRandomNeighbor$ é usada para a escolha de vizinhos no algoritmo Random.

A implementação de $ModifyElementMij$ é utilizada por um grupo de algoritmos de balanceamento de carga, que inclui os algoritmos Random, Threshold e Shortest, para alterar o valor do elemento da matriz de migração.

Para o algoritmo Threshold, a descrição dessa função seria:

```
MatrixM (type, A, H, M)
inicio
  choose = ChooseThresholdNeighbor (A, H);
  ModifyElementMij (A, M, choose);
fim.
```

A função $ChooseThresholdNeighbor$ é usada para a escolha de vizinhos no algoritmo Threshold.

Para o algoritmo Shortest, a descrição dessa função seria:

```
MatrixM (type, A, H, M)
inicio
  choose = ChooseShortestNeighbor (A, H);
  ModifyElementMij (A, M, choose);
fim.
```

A função $ChooseShortestNeighbor$ é usada para a escolha de vizinhos no algoritmo Shortest.

4.3.2 ModifyElementAii

A implementação de $ModifyElementAii$ é dependente do algoritmo de balanceamento de carga porque não se aplica a pelo menos um algoritmo de balanceamento, o algoritmo de Redistribuição Completa [FG96].

Essa rotina é responsável por alterar o valor de $A_{ii}(k)$, se o processador P_i participa do balanceamento de carga. A sua implementação é a mesma para os três algoritmos de balanceamento Random, Threshold e Shortest.

A heurística dessa rotina é a seguinte: se o número de tarefas atribuídas ao nó for maior que o valor de threshold, o elemento $A_{ii}(k)$ recebe o valor 1. Caso contrário, o elemento $A_{ii}(k)$ recebe o valor 0.

Uma descrição de alto nível para essa rotina seria:

```
ModifyElementAii (A, myparticles, threshold)
inicio
  se (numero de particulas de Pi > valor de threshold),
    entao Aii = 1;
  senao Aii = 0;
fim.
```

4.3.3 ModifyElementMij

A implementação de $ModifyElementMij$ é dependente do algoritmo de balanceamento de carga porque não se aplica a pelo menos um algoritmo de balanceamento, o algoritmo de Difusão [FG96].

Essa rotina é responsável por alterar o valor de $M_{ij}(k)$, se o processador P_i envia tarefas para P_j. A sua implementação é a mesma para os três algoritmos de balanceamento Random, Threshold e Shortest.

A heurística desse procedimento é a seguinte: Escolhe-se um processador P_e para o recebimento da carga. O elemento $M_{ij}(k) = 1$ se j = e (isto é, se P_j é o processador escolhido P_e). Caso contrário, o elemento M_{ij} recebe o valor 0.

O algoritmo que descreve a função $ModifyElementMij$ seria o seguinte:

```
ModifyElementMij (A, M, choose)
inicio
 se Aii =1, entao
 inicio
   para todos os elementos da linha i, da matriz M, faca
   inicio
     se Pj e o processador escolhido, entao Mij = 1;
     senao Mij = 0;
   fim-para;
 fim-se;
fim.
```

4.3.4 ChooseRandomNeighbor

Essa rotina só se aplica ao algoritmo de balanceamento de carga Random e é responsável por escolher randomicamente um vizinho entre os elementos do conjunto de processadores vizinhos. Esse procedimento escolhe um vizinho, j, do conjunto de processadores vizinhos. A saída será j:

```
ChooseRandomNeighbor (A, H)
inicio
  se Aii = 1, entao
  inicio
    va na linha i da matriz H e escolha j,
sendo que j <> i e Hij = 1;
```

```
    retorne j;
  fim-se;
fim.
```

4.3.5 ChooseThresholdNeighbor

Essa rotina só se aplica ao algoritmo de balanceamento de carga Threshold e é responsável por escolher randomicamente um vizinho entre os elementos do conjunto de processadores vizinhos. Esse procedimento escolhe um vizinho, j, do conjunto de processadores vizinhos. A saída será j se o processador escolhido estiver subcarregado, isto é, o número de tarefas atribuídas a ele for menor que o valor de threshold. Caso contrário, repete-se o processo de seleção do nó:

```
ChooseThresholdNeighbor (A, H)
inicio
  se Aii = 1, entao
  inicio
    (1) va na linha i de H, escolha j (j <> i) e Hij = 1;

    (2) se Ajj = 0, retorne j;

    (3) senao, va para o passo (1);
  fim-se;
fim.
```

4.3.6 ChooseShortestNeighbor

Essa rotina só se aplica ao algoritmo de balanceamento de carga Shortest e é responsável por escolher randomicamente um vizinho entre os elementos do conjunto de processadores vizinhos. Esse procedimento escolhe randomicamente um conjunto de vizinhos, S, do conjunto de processadores vizinhos. De S, obtém-se o nó, j, com o menor número de tarefas. Se o número de tarefas atribuídas a P_j for menor que o valor de threshold, a saída será j. Caso contrário, repete-se o processo de seleção de um outro conjunto de nós:

```
ChooseShortestNeighbor (A, H)
inicio
  se Aii = 1, entao
  inicio
    (1) escolha um conjunto, S, do conjunto de vizinhos;

    (2) escolha de S o no com o menor valor de threshold, j;

    (3) se Ajj = 0, retorne j;

    (4) senao, va para o passo (1);
  fim-se;
fim.
```

4.4 Biblioteca Específica da Aplicação

A Biblioteca Específica da Aplicação é a biblioteca que contém todas as operações que têm implementações dependentes da simulação paralela do problema dos N-corpos.

Nas subseções seguintes, cada uma dessas operações é apresentada.

4.4.1 ReassignDomain

O seguinte algoritmo implementa a redistribuição de domínios em um processador:

```
ReassignDomain (A, M, root, mytree, nbody, neighbors)
inicio
  ModifyDomainBoundaries (A, M, mytree, neighbors)
  collect_particles (mytree, nbody, sendbody);
  runaway (root, sendbody, 0, nbody);
```

```
  receive_runaway (nbody);
fim.
```

A heurística do algoritmo é a seguinte: se o processador está sobrecarregado, há uma mudança no limite do processador. Manda-se a mudança para o vizinho escolhido. O vizinho escolhido faz a mudança no seu limite. Depois das operações de ajuste de fronteiras, faz-se as operações de contabilidade de partículas.

As três últimas funções, *collect_particles*, *runaway* e *receive_runaway*, são usadas para coletar as partículas de um processador, enviar partículas pertencentes a outro processador e receber partículas pertencentes ao processador, respectivamente. Elas são implementadas na simulação paralela do problema dos N-corpos.

A função $ModifyDomainBoundaries$ envia as mudanças a serem feitas no limite do processador escolhido em $MatrixM$. Além disso, ela recebe as mudanças a serem feitas nos limites e efetua essas mudanças.

4.4.2 ModifyDomainBoundaries

A implementação de $ModifyDomainBoundaries$ é dependente da aplicação porque ela altera as fronteiras das árvores dos processadores, discutidas no capítulo 3, criadas pela aplicação. Essa função envia e recebe as mudanças a serem feitas nos limites.

Uma descrição de alto nível seria:

```
ModifyDomainBoundaries (A, M, mytree, neighbors)
inicio
  se Aii = 1, entao
  inicio
    encontrar o limite l;
    para cada k, tal que Mik = 1, faca
    inicio
      ChangeOneDomainBoundary (mytree, neighbors, k, action);
      enviar alteracoes para k;
```

```
    fim-para;
  fim-se;

  para cada k, tal que Mki = 1 e Mik = 1, faca
  inicio
    receber as mudancas a serem feitas em i;
    ChangeOneDomainBoundary (mytree, neighbors, k, action);
  fim-para;
fim.
```

O procedimento $ChangeOneDomainBoundary$ é responsável por alterar o limite de um processador.

Na referida implementação, existem duas maneiras de se "encontrar o limite l":

- l é um valor constante $\rightarrow$ assume-se que um processador vai aumentar ou diminuir a fronteira de seu domínio de um valor pré-determinado, e a implementação de $ModifyDomainBoundaries$ faz as modificações nas fronteiras. Nessa opção, não se sabe a quantidade de partículas que, um nó com sobrecarga, enviará. Um processador sobrecarregado pode diminuir de l a sua fronteira e, mesmo assim, continuar nesse estado, pois a nova fronteira que lhe foi atribuída não é suficiente para o envio do excesso de suas partículas. Ou esse processador pode tornar-se subcarregado porque, com a mudança de sua fronteira, ele pode enviar, para o seu vizinho escolhido, mais corpos que o necessário. Assim, a redesignação de domínios não garante o equilíbrio do número de corpos associados aos processadores;

- l é um valor calculado $\rightarrow$ faz-se uma ordenação de partículas e, a partir dessa ordenação, calcula-se o valor de l. Com esse valor calculado, a implementação de $ModifyDomainBoundaries$ faz as modificações nas fronteiras. Nessa opção, sabe-se a quantidade de partículas que, um nó com sobrecarga, enviará. Um processador sobrecarregado diminui de l a sua fronteira, envia o excesso de corpos e torna-se um nó balanceado. Assim, a redesignação de domínios garante o equilíbrio do número de corpos associados aos processadores.

4.4.3 ChangeOneDomainBoundary

A implementação de *ChangeOneDomainBoundary* é dependente da aplicação porque ela modifica os limites contidos nas árvores de cada processador. Uma descrição de alto nível desse procedimento seria:

```
ChangeOneDomainBoundary (mytree, neighbors, id, action)
inicio
  caso (action)
  INCREASE: incrementar o limite l do processador;
  DECREASE: decrementar o limite l do processador;
fim.
```

Capítulo 5

Análise de Resultados

A fim de validar a implementação descrita no capítulo 4, serão analisados os tempos gastos nas execuções da simulação paralela antes e após a inclusão da biblioteca de balanceamento desenvolvida. Para fazer essa análise, é necessário realizar um projeto de experimentos.

Os experimentos foram realizados na máquina paralela IBM SP/2 do Núcleo de Computação Eletrônica (NCE) da Universidade Federal do Rio de Janeiro (UFRJ) e do Laboratório Nacional de Computação Científica (LNCC). Os dados foram obtidos utilizando os nós da máquina de maneira exclusiva, isto é, a simulação foi executada sem o compartilhamento do poder computacional dos nós com nenhuma outra aplicação.

Na seção 5.1, serão discutidas as variáveis que influenciam o tempo de resposta da simulação paralela. Na seção 5.2, o modelo usado para o projeto de experimentos será explicado. Na seção 5.3, serão analisados, apenas, os resultados obtidos na execução da simulação paralela com o uso da biblioteca de balanceamento desenvolvida. Na seção 5.4, serão comparados os resultados obtidos na execução da simulação paralela do problema dos N-corpos antes e após a inclusão da biblioteca de balanceamento. Na seção 5.5, serão apresentadas algumas considerações sobre os resultados obtidos.

5.1 Descrição do Problema dos N-Corpos

Essa seção revê as idéias básicas da solução do problema dos N-corpos e apresenta os tempos de execução da simulação sequencial e da aplicação paralela.

5.1.1 Simulação Paralela

A simulação paralela do problema dos N-corpos, discutida no capítulo 3, estuda a evolução de um sistema de N corpos (ou partículas) onde existe a força gravitacional Newtoniana entre pares de corpos.

Os corpos consistem de uma massa, posição e velocidades iniciais e eles estão distribuídos sobre um domínio bidimensional físico finito. Nessa simulação, o domínio físico é dividido em 4 pequenas regiões e cada região é alocada a um dos 4 processadores. Cada processador é responsável por todos os cálculos associados com as partículas de sua região. Chama-se de tarefa o conjunto de cálculos associados a uma única partícula.

Para calcular as forças em cada partícula, é necessário conhecer a posição de outras partículas e isso envolve troca de informação entre pares de processadores.

Depois da troca de informações, cada processador calcula a força resultante em cada partícula de sua região e atualiza suas posições e velocidades. O processador verifica quais de suas partículas atravessaram as fronteiras de seu domínio e, então, transfere as partículas que não pertencem mais a ele para os processadores apropriados.

Essa transferência causa um desequilíbrio na distribuição de tarefas entre os processadores. O algoritmo de balanceamento de carga é invocado depois da fase de transferência de corpos. O balanceamento de carga consiste em redesignar domínios para cada processador, de forma que o número de partículas associadas aos processadores fique balanceado.

5.1.2 Execuções das Simulações

Foram usadas no experimento simulações com 4096 e 16384 partículas. Para os valores inferiores a 4096, a paralelização do sistema não é vantajosa, pois o tempo de resposta da simulação paralela é similar ao tempo gasto no algoritmo sequencial (tabela 5.1). Essa similaridade do tempo paralelo é devido ao tempo gasto na fase de troca de informações entre os processadores. Nessa fase, cada processador troca $log\ p$ mensagens com os outros $p - 1$ processadores. Se o tempo gasto com processamento é pequeno, como acontece com números baixos de partículas, o *overhead* da fase de comunicação faz com que haja um aumento do tempo total de execução, em relação a simulação sequencial. Quando o número de corpos cresce, o *overhead* de comunicação é compensado pelo tempo gasto no cálculo das forças resultantes nos corpos (uma das etapas integrantes da fase de processamento).

Todas as posicões e velocidades iniciais das partículas seguem a Distribuição Uniforme e foram geradas a partir de funções existentes na aplicação paralela original.

Partículas	*Simulação* Sequencial	Paralela
512	3682	3633
	3664	3749
	3692	4069
	3659	3858
	3684	4155
	3627	3507
	3699	3445
	3693	3819
	3650	3145
	3667	4142
2048	18590	18498
	17815	18299
	18343	18229
	18475	19835
	18316	18129
	18318	18676
	18341	19227
	18031	18733
	18264	18922
	18565	19133
4096	142696	78010
	142936	76935
	143042	77664
	143220	79344
	143259	76861
	143240	73949
	143127	77941
	143264	72659
	144464	79549
	143424	73439
16384	1114267	472471
	1109664	483067
	1111704	496344
	1127939	473277
	1111381	461799
	1112772	468958
	1114053	479472
	1127399	487782
	1114257	493110
	1113257	498025

Tabela 5.1: Tempos (em ms) de cada uma das 10 execuções das simulações, sequencial e paralela

5.2 Projeto de Experimentos

Antes de conceituar projeto de experimentos é necessário definir fatores e níveis. As variáveis que afetam o tempo de resposta de um experimento são chamadas de fatores. Os valores que um fator pode assumir são chamados de níveis.

A finalidade de um projeto de experimentos é obter, dos dados produzidos pelo menor número de execuções do sistema (ou experimentos), o máximo de informações a respeito de seu comportamento [Jai91].

O projeto de experimentos consiste de um conjunto de especificações, a saber:

(a) o número de experimentos;

(b) as variáveis (e suas combinações) que afetam o tempo de resposta de um experimento;

(c) o número de repetições (ou replicações) de um experimento.

Nesse capítulo, são realizados 2 projetos de experimentos:

- análise das alternativas de balanceamento de carga → nesse caso, um projeto com 2 fatores é usado para selecionar qual foi a simulação paralela com redistribuição de tarefas, gerada a partir da biblioteca de balanceamento, mais eficiente.
  Os fatores a serem considerados na análise desse sistema são o algoritmo de redistribuição de tarefas e o tipo de balanceamento utilizado. Foram implementados 3 algoritmos de balanceamento: *random*, *threshold* e *shortest*. Já o fator balanceamento pode assumir um dos 2 níveis:

  (a) com balanceamento de carga sem certeza de envio (CBS) → essa opção é usada quando se quer medir o tempo de execução da simulação paralela do problema dos N-corpos com o uso da biblioteca de balanceamento de carga. Nessa opção, a redesignação de domínios feita no algoritmo de balanceamento não garante o equilíbrio do número de partículas associadas ao processadores, conforme discutido no capítulo 4;

  (b) com balanceamento de carga com certeza de envio (CBC) → essa opção é usada quando se quer medir o tempo de execução da simulação paralela do problema dos N-corpos com o uso da biblioteca de balanceamento de carga. Nessa opção, a redesignação de domínios feita no algoritmo de balanceamento garante o equilíbrio do número de partículas associadas ao processadores, conforme discutido no capítulo 4.

O projeto de experimentos 2-fatorial com replicações, descrito em [Jai91], é usado, geralmente, para analisar os diferentes fatores considerados no sistema.

O modelo do projeto 2-fatorial utilizado nessa etapa é baseado nos fatores algoritmo de redistribuição de tarefas (A) e o tipo de balanceamento de carga (B). O fator A pode assumir 3 níveis. Já o fator B tem 2 níveis. Nos testes apresentados posteriormente, cada experimento é repetido 10 vezes.

- análise do balanceamento de carga na simulação paralela $\rightarrow$ nesse caso, um projeto com 1 fator, o balanceamento de carga, é usado para comparar os desempenhos da simulação paralela do problema dos N-corpos antes e após a inclusão da biblioteca de balanceamento desenvolvida. Esse fator pode assumir um dos 2 níveis:

  (a) sem balanceamento de carga (SBC) $\rightarrow$ essa opção é usada quando se quer medir o tempo de execução da simulação paralela do problema dos N-corpos sem a intervenção da biblioteca de balanceamento de carga;

  (b) a alternativa selecionada no 1º projeto realizado.

  O projeto de experimentos 1-fatorial com replicações, descrito em [Jai91], é usado, geralmente, para comparar as diferentes alternativas de um único fator.

  O modelo do projeto 1-fatorial utilizado nessa etapa é baseado, unicamente, no tipo de balanceamento de carga (B). Esse fator pode assumir 2 níveis. Nos testes apresentados posteriormente, cada experimento é repetido 10 vezes.

  Esse modelo é uma simplificação do modelo do projeto 2-fatorial porque, quando o fator algoritmo de redistribuição do projeto 2-fatorial é desconsiderado, os dois modelos são similares. Devido a essa similaridade, pode-se desenvolver um modelo geral do projeto 2-fatorial e adaptá-lo aos dois tipos de projetos usados nesse capítulo.

Nas subseções 5.2.1 a 5.2.6, o modelo do projeto 2-fatorial será explicado.

5.2.1 Modelo

Considere um projeto com a níveis do fator A, b níveis do fator B e r replicações do experimento com cada um desses ab níveis. Nesse caso, o modelo é dado por:

$$y_{ijk} = \mu + \alpha_j + \beta_i + \gamma_{ij} + e_{ijk}$$

$i = 1,...,b$

$j = 1,...,a$

$k = 1,...,r$ $\{5.1\}$

onde:

- y_{ijk} é a resposta na k-ésima repetição do experimento com o fator A no nível j e com o fator B no nível i;
- μ é a resposta média do sistema;
- α_j é o efeito do fator A no nível j;
- β_i é o efeito do fator B no nível i;
- γ_{ij} é o efeito da interação entre o fator A no nível j e o fator B no nível i;
- e_{ijk} é o erro experimental obtido na k-ésima replicação do experimento com o fator A no nível j e com o fator B no nível i.

Pela definição do modelo, assume-se que:

- os efeitos são calculados de forma que sua soma seja igual a zero
  $\sum_{j=1}^{a} \alpha_j = 0$
  $\sum_{i=1}^{b} \beta_i = 0$
- as interações entre os fatores são calculados de forma que a soma de suas linhas e colunas seja igual a zero
  $\sum_{i=1}^{b} \gamma_{i1} = \sum_{i=1}^{b} \gamma_{i2} = ... \sum_{i=1}^{b} \gamma_{ia} = 0$
  $\sum_{j=1}^{a} \gamma_{1j} = \sum_{j=1}^{a} \gamma_{2j} = ... \sum_{j=1}^{a} \gamma_{bj} = 0$
- os erros são calculados de forma que sua soma seja igual a zero
  $\sum_{k=1}^{r} \sum_{j=1}^{a} \sum_{i=1}^{b} e_{ijk} = 0$

5.2.2 Cálculo dos Efeitos

Os valores medidos são dispostos em uma matriz $a \times b$ onde cada dimensão corresponde a um fator considerado. O conjunto de repetições de um dado experimento pode ser visto como ocupando a mesma célula dessa matriz. Dessa forma, cada célula da matriz conterá r replicações do mesmo experimento.

Calculando-se a média dos experimentos de uma célula da matriz, obtém-se que:

$$\bar{y}_{ij.} = \mu + \alpha_j + \beta_i + \gamma_{ij} \qquad \{5.2\}$$

O ponto (.) indica que a média é feita entre todas as repetições da célula.

Seja y_{ijk} o elemento (i,j,k) dessa matriz. Substituindo cada y_{ijk} da matriz pela equação {5.1}, obtêm-se abr equações. Adicionando as abr equações, tem-se que:

$$\sum_{k=1}^{r} \sum_{j=1}^{a} \sum_{i=1}^{b} y_{ijk} = abr\mu + br \sum_{j=1}^{a} \alpha_j + ar \sum_{i=1}^{b} \beta_i + r \sum_{j=1}^{a} \sum_{i=1}^{b} \gamma_{ij} + \sum_{k=1}^{r} \sum_{j=1}^{a} \sum_{i=1}^{b} e_{ijk} \qquad \{5.3\}$$

Por definição, a soma dos efeitos e da interação entre eles é igual a zero. Se a soma dos erros experimentais em cada coluna é igual a zero, obtém-se:

$$\sum_{k=1}^{r} \sum_{j=1}^{a} \sum_{i=1}^{b} y_{ijk} = abr\mu \qquad \{5.4\}$$

A resposta média do sistema é dada por:

$$\mu = 1/abr \sum_{k=1}^{r} \sum_{j=1}^{a} \sum_{i=1}^{b} y_{ijk} \qquad \{5.5\}$$

A fração da equação {5.5} é conhecida como média global do sistema e é denotada por $\bar{y}_{...}$. Os 3 pontos (...) indicam que a média é feita em todas as dimensões da matriz. Eles são usados para distinguir a média global da média da coluna, $\bar{y}_{.j.}$, que é obtida fazendo-se a média das respostas pertencentes à coluna j.

A média da coluna j é dada por:

$$\bar{y}_{.j.} = 1/br \sum_{k=1}^{r} \sum_{i=1}^{b} y_{ijk} \qquad \{5.6\}$$

Substituindo a equação {5.1} na equação {5.6}, tem-se que:

$$\bar{y}_{.j.} = 1/br \sum_{k=1}^{r} \sum_{i=1}^{b} (\mu + \alpha_j + \beta_i + \gamma_{ij} + e_{ijk})$$

$$\bar{y}_{.j.} = 1/br \ (br\mu + br\alpha_j + r \sum_{i=1}^{b} \beta_i + r \sum_{i=1}^{b} \gamma_{ij} + \sum_{k=1}^{r} \sum_{i=1}^{b} e_{ijk}) \qquad \{5.7\}$$

Pela definição do modelo, as somatórias da equação {5.7} são iguais a zero. Assim:

$$\bar{y}_{.j.} = \mu + \alpha_j$$

$$\alpha_j = \bar{y}_{.j.} - \mu = \bar{y}_{.j.} - \bar{y}_{...} \qquad \{5.8\}$$

Analogamente, β_i é dado por:

$$\bar{y}_{i..} = \mu + \beta_i$$

$$\beta_i = \bar{y}_{i..} - \mu = \bar{y}_{i..} - \bar{y}_{...} \qquad \{5.9\}$$

Substituindo a equação {5.5}, {5.8} e {5.9} em {5.3}, obtém-se que:

$$\gamma_{ij} = \bar{y}_{ij.} - \bar{y}_{i..} - \bar{y}_{.j.} + \bar{y}_{...} \qquad \{5.10\}$$

5.2.3 Cálculo dos Erros

A resposta estimada para a alternativa j é dada por:

$$\hat{y}_{ij} = \mu + \alpha_j + \beta_i + \gamma_{ij} = \bar{y}_{ij}$$

A diferença entre a resposta medida e a resposta estimada representa um erro experimental. Calculando-se os erros experimentais para as abr observações, obtém-se que o erro médio deve ser zero porque μ foi calculado assumindo-se que as somas dos efeitos, das interações entre eles e dos erros era igual a zero.

A variância dos erros pode ser estimada pela soma dos quadrados dos erros (SSE):

$$SSE = \sum_{k=1}^{r} \sum_{j=1}^{a} \sum_{i=1}^{b} e_{ijk}^2 \qquad \{5.11\}$$

5.2.4 Alocação da Variação

A variação total da observação y em um projeto 2-fatorial é devida aos fatores considerados, a interação entre os fatores e aos erros experimentais. Para calculá-la, eleva-se a equação $\{5.1\}$ ao quadrado, para cada observação y:

$$y_{ijk}^2 = \mu^2 + \alpha_j^2 + \beta_i^2 + \gamma_{ij}^2 + e_{ijk}^2 + 2\mu\alpha_j + 2\mu\beta_i + 2\mu\gamma_{ij} + 2\mu e_{ijk} + \\ 2\alpha_j\beta_i + 2\alpha_j\gamma_{ij} + 2\alpha_j e_{ijk} + 2\beta_i\gamma_{ij} + 2\beta_i e_{ijk} + 2\gamma_{ij} e_{ijk}$$

Adicionando os termos correspondentes das abr equações, tem-se que:

$$\sum_{i,j,k} y_{ijk}^2 = \sum_{i,j,k} \mu^2 + \sum_{i,j,k} \alpha_j^2 + \sum_{i,j,k} \beta_i^2 + \sum_{i,j,k} \gamma_{ij}^2 + \sum_{i,j,k} e_{ijk}^2 + \\ Soma\ dos\ produtos \qquad \{5.12\}$$

Todas as somas dos produtos são iguais a zero devido às seguintes restrições:

(a) a soma dos efeitos é igual a zero;

(a) a soma das interações entre os fatores é igual a zero;

(c) a soma dos erros experimentais é igual a zero.

A equação {5.12}, expressa em termos das somas dos quadrados, pode ser reescrita como:

$$SSY = SSO + SSA + SSB + SSAB + SSE \qquad \{5.13\}$$

Onde SSY é a soma dos quadrados de y, SSO é a soma dos quadrados da média global, SSA é a soma dos quadrados dos efeitos do fator A, SSB é a soma dos quadrados dos efeitos do fator B e $SSAB$ é a soma dos quadrados dos efeitos da interação entre os fatores. Observa-se que SSO, SSA, SSB e $SSAB$ podem ser reescritos como:

$$SSO = \sum_{i,j,k} \mu^2 = abr\mu^2$$

$$SSA = \sum_{i,j,k} \alpha_j^2 = br \sum_j \alpha_j^2$$

$$SSB = \sum_{i,j,k} \beta_i^2 = ar \sum_i \beta_i^2$$

$$SSAB = \sum_{i,j,k} \gamma_{ij}^2 = r \sum_{i,j} \gamma_{ij}^2 \qquad \{5.14\}$$

A variação total de y (SST) é definida como:

$$SST = \sum_{i,j,k} (y_{ijk} - \mu)^2 = \sum_{i,j,k} y_{ijk}^2 \text{ - } abr\mu^2 = SSY - SSO = SSA + SSB + SSAB + SSE \qquad \{5.15\}$$

A variação total pode, portanto, ser dividida em duas partes, SSA, SSB e $SSAB$, e SSE, que representam as partes explicável e inexplicável da variação, respectivamente.

A alta porcentagem da variação explicável indica que o modelo foi empregado de maneira correta. Essa porcentagem é dada pela soma das seguintes percentagens:

% da variação explicada pelo fator A = $100(SSA/SST)$

% da variação explicada pelo fator B = $100(SSB/SST)$

% da variação explicada pela interação AB = $100(SSAB/SST)$

A porcentagem da variação não explicada pelo modelo é dada por:

% da variação não explicada pelo modelo = $100(SSE/SST)$

5.2.5 Análise da Variância

Na alocação da variação, descrita na subseção 5.2.4, qualquer fator que possui uma alta porcentagem de variação é considerado importante.

Essa importância deveria ser distinguida de significância, que é um termo estatístico. Para determinar se um termo tem um efeito significativo na resposta, compara-se sua contribuição na variação com relação aos erros experimentais do sistema.

O procedimento estatístico que analisa a significância de um ou mais fatores é chamado de análise de variância (ANOVA).

Para entender o ANOVA, considere as soma dos quadrados SSY, SSO, SSA, SSB, $SSAB$ e SSE. Cada uma dessas somas tem um grau de liberdade que corresponde ao número de termos independentes requeridos para calculá-los. Os graus de liberdade para as somas são:

$DFY = abr$

$DF0 = 1$

$DFA = (a-1)$

$DFB = (b-1)$

$$DFAB = (a-1)(b-1)$$

$$DFE = ab(r-1) \qquad \{5.16\}$$

A soma SSY consiste de uma soma de abr termos, e todos eles podem ser escolhidos independentemente. Portanto, SSY tem abr graus de liberdade.

A soma SSO consiste de um único termo, μ^2, que é repetido abr vezes. Uma vez que um valor para μ for escolhido, a soma SSO pode ser calculada. Assim, SSO tem 1 grau de liberdade.

A soma SSA contém a soma de a termos, mas somente $a-1$ desses termos são independentes, visto que a soma de todos os efeitos deve ser igual a zero. Portanto, SSB tem $a-1$ graus de liberdade.

A soma SSB contém a soma de b termos, mas somente $b-1$ desses termos são independentes, visto que a soma de todos os efeitos deve ser igual a zero. Portanto, SSB tem $b-1$ graus de liberdade.

A soma $SSAB$ contém a soma de ab termos, mas somente $(a-1)(b-1)$ desses termos são independentes, visto que a soma dos efeitos de A e dos efeitos de B devem ser iguais a zero. Portanto, $SSAB$ tem $(a-1)(b-1)$ graus de liberdade.

A soma SSE consiste de abr termos dos quais somente $ab(r-1)$ podem ser escolhidos independentemente, já que a soma dos r erros experimentais correspondentes as r replicações de cada experimento deve ser zero. Assim, somente $(r-1)$ erros em cada um dos ab experimentos são independentes.

O F-teste deve ser usado para verificar se SSX é significantemente maior que SSE. Assumindo-se que os erros são normalmente distribuídos, SSX e SSE tem distribuições chi-square. A razão $(SSX/DFX)/(SSE/DFE)$ tem uma distribuição F. Se essa razão calculada for maior que o F[1 - α; DFX, DFE] quantile obtido da tabela de quantiles das F-variáveis (ver, por exemplo, as tabelas A.6, A.7 e A.8 de [Jai91]), SSX é considerado significantemente maior que SSE.

A razão SSX/DFX é chamada de quadrado médio de X (MSX), Analogamente, a razão SSE/DFE é chamada de quadrado médio dos erros (MSE).

5.2.6 Intervalos de Confiança para os Efeitos

Os valores estimados dos parâmetros do modelo são variáveis randômicas, já que elas estão baseadas em uma amostra. Os resultados obtidos de uma outra amostra poderiam fornecer os parâmetros com valores diferentes. Para minimizar esse problema, a variância dos parâmetros do modelo deve ser estimada a partir dos erros experimentais do sistema.

Os intervalos de confiança para os efeitos devem ser calculados usando valores da Distribuição Normal para $ab(r-1)$ graus de liberdade (que são os graus de liberdade associados com os erros).

É possível fazer estimativas para os níveis (a partir de $\hat{y}_{ij} = \mu + \alpha_j$ ou de $\hat{y}_{ij} = \mu + \beta_i$) e para as combinações entre eles (a partir de $\sum h_k \alpha_k$ ou de $\sum h_k \beta_k$, onde, por definição, $\sum h_k = 0$).

As estimativas para os níveis isolados podem ser usadas para calcular os intervalos de confiança para as respostas médias.

As estimativas para a combinação dos níveis podem ser usadas para verificar se existe uma diferença significante entre duas alternativas.

O intervalo de confiança para um nível X é dado por:

$$Efeito_X \pm tS_X$$

Onde $Efeito_X$ é a resposta média de X, S_X é o desvio padrão de X e t é o valor da Distribuição Normal.

Os desvios padrões são dados por:

$$S_e = \text{desvio padrão dos erros} = \sqrt{MSE}$$

$$S_{\alpha_j} = \text{desvio padrão de } \alpha_j = S_e \sqrt{(a-1)/abr}$$

$$S_{\beta_i} = \text{desvio padrão de } \beta_i = S_e \sqrt{(b-1)/abr}$$

$$S_{\gamma_{ij}} = \text{desvio padrão de } \gamma_{ij} = S_e \sqrt{(a-1)(b-1)/abr}$$

$S_\mu =$ desvio padrão de $\mu = S_e/\sqrt{br}$

Os efeitos em que os intervalos incluem o zero são significativos; os demais são não significativos.

A fim de se comparar níveis, α_l e α_m, pode-se calcular os intervalos de confiança entre eles, usando a fórmula $\sum h_k \alpha_k$. Para comparar α_l e α_m, deve-se seguir os seguintes passos:

- atribuir valores 1, -1 para α_l e α_m, e 0 para os demais, levando em consideração que $\sum h_k = 0$;
- calcular o valor médio de α_l e α_m, que é dado por $y_{.l.} - y_{.m.}$;
- calcular o desvio padrão de α_l e α_m, que é dado por $S_e\sqrt{\sum h_k^2/br}$.

Na comparação dos níveis α_l e α_m, se o intervalo de confiança inclui o zero, α_l não é superior a α_m e vice-versa. Se o intervalo de confiança não inclui o zero, α_l é superior a α_m.

De maneira análoga, para comparar os níveis β_l e β_m, deve-se:

- atribuir valores 1, -1 para β_l e β_m, e 0 para os demais, levando em consideração que $\sum h_k = 0$;
- calcular o valor médio de β_l e β_m, que é dado por $y_{l..} - y_{m..}$;
- calcular o desvio padrão de β_l e β_m, que é dado por $S_e\sqrt{\sum h_k^2/ar}$.

Na comparação dos níveis β_l e β_m, se o intervalo de confiança inclui o zero, β_l não é superior a β_m e vice-versa. Se o intervalo de confiança não inclui o zero, β_l é superior a β_m.

5.3 Análise das Alternativas de Balanceamento de Carga

Nessa seção serão analisados os tempos das simulações do problema do N-corpos com 3 algoritmos de balanceamento de carga, gerados a partir da biblioteca de balanceamento, a fim de se descobrir qual dessas simulações foi a mais eficiente.

As tabelas 5.2, 5.3 e 5.4 mostram o tempo de execução da simulação com os três algoritmos de balanceamento de carga implementados, a saber: *random*, *threshold* e *shortest*.

A simulação foi executada variando-se o balanceamento de carga para os seguintes níveis:

- a simulação com a biblioteca de balanceamento sem a garantia de equilíbrio do número de partículas de cada um dos processadores participantes (CBS);
- a simulação com a biblioteca de balanceamento com a garantia de equilíbrio do número de partículas dos nós do sistema, ao final da iteração (CBC).

Nas subseções seguintes serão realizadas as análises para o sistema com 4096 e 16384 corpos.

Partículas	*Balanceamento* CBS	CBC
4096	54145	63796
	50372	62724
	55502	62734
	57293	62759
	53624	61480
	56261	64509
	52960	61250
	58263	62655
	58802	64102
	55198	62099
16384	365677	408124
	355702	407559
	344331	425001
	377706	401598
	361188	408636
	360509	408736
	341438	409079
	339157	407894
	365278	409351
	347464	424015

Tabela 5.2: Tempos (em ms) de cada uma das 10 execuções da simulação paralela com o algoritmo de balanceamento de carga *random*

Partículas	*Balanceamento* CBS	CBC
4096	54857	61890
	55212	60784
	55947	62596
	54754	61605
	55507	62143
	56000	62029
	54763	61296
	52372	62123
	55125	65343
	57006	63997
16384	369368	414846
	392752	437041
	378279	409581
	381000	405884
	357954	408678
	357644	409825
	370662	418337
	373869	416590
	373769	408300
	354003	409617

Tabela 5.3: Tempos (em ms) de cada uma das 10 execuções da simulação paralela com o algoritmo de balanceamento de carga *threshold*

Partículas	*Balanceamento* CBS	CBC
4096	52374	59071
	50682	59066
	56388	58037
	54198	57322
	53122	59579
	52851	56409
	52838	59092
	50273	59079
	54158	58672
	54322	56364
16384	352101	423809
	342705	417687
	348132	404408
	342948	418022
	330032	422257
	338566	429812
	345639	422233
	342104	429029
	345361	421760
	350124	411258

Tabela 5.4: Tempos (em ms) de cada uma das 10 execuções da simulação paralela com o algoritmo de balanceamento de carga *shortest*

5.3.1 Análise das Alternativas para o Sistema com 4096 Corpos

A média para cada um dos níveis dos fatores é calculada e apresentada na tabela 5.5. Essas médias são os tempo previstos pelo modelo.

A média global pode ser interpretada como uma indicação de que o tempo de execução de uma simulação com o algoritmo de balanceamento médio e com o tipo de balanceamento médio é 57826,18 ms.

A tabela 5.5 também apresenta os efeitos causados pelo fator algoritmo e pelo fator balanceamento considerados isoladamente. Os resultados podem ser interpretados da seguinte maneira:

- fator algoritmo $\rightarrow$ os tempos gastos na execução da simulação com o algoritmo *random* e com o algoritmo *threshold* são, respectivamente, 1200,22 e 931,12 ms superior ao tempo gasto na execução da simulação com o algoritmo de redistribuição médio. Já a execução da simulação com o algoritmo *shortest* é 2131,33 ms inferior ao tempo gasto na execução da simulação com o algoritmo de balanceamento médio;
- fator balanceamento $\rightarrow$ os tempos gastos na execução da simulação com balanceamento sem certeza de envio é 3324,32 ms inferior ao tempo gasto na execução da simulação com o balanceamento médio. Já o tempo gasto na execução da simulação com balanceamento com certeza de envio é 3324,32 ms superior ao tempo gasto na execução da simulação com balanceamento médio.

A tabela 5.6 apresenta os efeitos causados pela interação entre os dois fatores. Essa interação pode ser interpretada como, por exemplo, para o algoritmo *random* com certeza de envio, uma indicação de que o tempo de execução da simulação com o algoritmo *random* com certeza de envio é 460,08 ms superior ao tempo gasto na execução da simulação com o algoritmo médio com certeza de envio, ou 460,08 ms superior ao tempo gasto na execução da simulação com o algoritmo *random* com balanceamento de carga médio.

Na tabela 5.7 são apresentadas as somas dos quadrados dos fatores, assim como a variação percentual causada por cada fator e pela interação entre eles.

Observa-se que 68% da variação no tempo de execução é devido ao balanceamento utilizado. O fator algoritmo contribui com 14% na variação do tempo de execução da simulação. A interação entre os fatores teve pouca participação na variação dos resultados.

O modelo proposto não é capaz de explicar 16% da variação no tempo de execução da simulação. Esse percentual é relativamente alto, mas pode ser devido ao fato de que os tempos medidos nas 10 replicações de um mesmo experimento apresentaram variações razoáveis. Essas variações podem ser devidas ao fato de que, apesar da exclusividade do poder computacional das máquinas usadas, a simulação foi executada compartilhando a rede de comunicação entre todos os nós do sistema com outras aplicações. Portanto, esses erros podem ser experimentais e não existem indicativos de uma inadequação do modelo para expressar os tempos medidos.

A tabela 5.8 apresenta o grau de significância de cada efeito considerado na análise.

Conforme discutido na subseção 5.2.5, para um dado nível de significância, α, um fator X é estatisticamente significativo se o F calculado for superior ao valor tabelado de $F[\alpha$, grau de liberdade de X, grau de liberdade do erro]. Por indisponibilidade de dados, considerou-se o grau de liberdade do erro igual a 500 (superior ao grau dos erros em todos os testes realizados). Supondo-se que 90% é o nível de significância desejado, os valores de F calculados para os efeitos são superiores aos seus respectivos valores tabelados. Portanto, todos os efeitos são estatisticamente significativos.

Antes de se calcular os intervalos de confiança para o fator considerado, é preciso descobrir o valor da Distribuição Normal (t) a ser usado. Para um nível de significância de 90% ($\alpha = 0,9$), o valor de t[1 - (1 - $\alpha/2$), DFE] é 1,697.

A tabela 5.9 mostra o intervalo de confiança para o fator balanceamento na análise.

Os efeitos em que os intervalos de confiança correspondentes não incluem o zero são significativos, os demais, são não significativos. Dessa forma, observa-se que os fatores e a interação entre eles são significativos.

A fim de comparar os níveis *random*, *threshold* e *shortest* do fator algoritmo, pode-se calcular os intervalos de confiança entre as alternativas, usando a fórmula $\sum h_k \alpha_k$ (onde $\sum h_k = 0$, por definição):

- atribuindo-se para h os valores h_1 = -1, h_2 = 1 e h_3 = 0, tem-se que, para 90% de nível de significância, o intervalo de confiança é $(269.1 \pm 905.4127) = (-636.31270, 1174.51270)$;
- atribuindo-se para h os valores h_1 = -1, $h_2 = 0$ e $h_3 = 1$, tem-se que, para 90% de nível de significância, o intervalo de confiança é $(-3331.55 \pm 905.4127) = (-4236.96270, -2426.13730)$;
- atribuindo-se para h os valores $h_1 = 0$, h_2 = -1 e $h_3 = 1$, tem-se que, para 90% de nível de significância, o intervalo de confiança é $(-3062.45 \pm 905.4127) = (-3967.86270, -2157.03730)$.

Na comparação dos níveis α_l e α_m, se o intervalo de confiança inclui o zero, α_l não é superior a α_m e vice-versa. Caso contrário, α_l é superior a α_m. Observa-se que o nível *shortest* é superior aos demais níveis.

Analogamente, para comparar os níveis CBS e CBC do fator balanceamento, pode-se calcular os intervalos de confiança entre as alternativas, usando a fórmula $\sum h_k \beta_k$ (onde $\sum h_k = 0$, por definição). Atribuindo-se para h os níveis h_1 = -1 e $h_2 = 1$, tem-se que, para 90% de nível de significância, o intervalo de confiança é $(-6648.63333 \pm 739.26637) = (-7387.89970, -5909.36696)$.

Na comparação dos níveis β_l e β_m, se o intervalo de confiança inclui o zero, β_l não é superior a β_m e vice-versa. Caso contrário, β_l é superior a β_m. Observa-se que a alternativa CBS é superior a alternativa CBC.

Fazendo-se uma análise geral da tabelas 5.2, 5.3 e 5.4, já era possível observar uma superioridade do nível *shortest* em relação aos demais níveis, e uma superioridade da alternativa CBS em relação a alternativa CBC. Com os cálculos dos intervalos de confiança entre as alternativas, foi possível comprovar a superioridades observadas nessa análise.

Considerando-se que o objetivo desse capítulo é estudar como o balanceamento de carga pode melhorar o desempenho da aplicação, a simulação paralela com o algoritmo de balanceamento *shortest* sem certeza de envio será usado para a análise proposta na seção 5.4.

5.3.2 Análise das Alternativas para o Sistema com 16384 Corpos

A média para cada um dos níveis dos fatores é calculada e apresentada na tabela 5.10. Essas médias são os tempo previstos pelo modelo.

A média global pode ser interpretada como uma indicação de que o tempo de execução de uma simulação com o algoritmo de balanceamento médio e com o tipo de balanceamento médio é 385910,15 ms.

A tabela 5.10 também apresenta os efeitos causados pelo fator algoritmo e pelo fator balanceamento considerados isoladamente. Os resultados podem ser interpretados da seguinte maneira:

- fator algoritmo $\rightarrow$ o tempo gasto na execução da simulação com o algoritmo *threshold* é 6489,8 ms superior ao tempo gasto na execução da simulação com o algoritmo de redistribuição médio. Já as execuções da simulação com o algoritmo *shortest* e com o algoritmo *random* são, respectivamente, 4010,8 e 2479 ms inferior ao tempo gasto na execução da simulação com o algoritmo de balanceamento médio;

- fator balanceamento $\rightarrow$ os tempos gastos na execução da simulação com balanceamento sem certeza de envio é 29058,41667 ms inferior ao tempo gasto na execução da simulação com o balanceamento médio. Já o tempo gasto na execução da simulação com balanceamento com certeza de envio é 29058,41667 ms superior ao tempo gasto na execução da simulação com balanceamento médio.

A tabela 5.11 apresenta os efeitos causados pela interação entre os dois fatores. Essa interação pode ser interpretada como, por exemplo, para o algoritmo *random* com certeza de envio, uma indicação de que o tempo de execução da simulação com o algoritmo *random* com certeza de envio é 1481,26667 ms superior ao tempo gasto na execução da simulação com o algoritmo médio com certeza de envio, ou 1481,26667 ms superior ao tempo gasto na execução da simulação com o algoritmo *random* com balanceamento de carga médio.

Na tabela 5.12 são apresentadas as somas dos quadrados dos fatores, assim como a variação percentual causada por cada fator e pela interação entre eles.

Observa-se que 86% da variação no tempo de execução é devido ao balanceamento utilizado. O fator algoritmo e a interação entre os fatores tiveram pouca participação na variação dos resultados.

O modelo proposto não é capaz de explicar 12% da variação no tempo de execução da simulação. Esse percentual é relativamente alto, mas pode ser devido ao fato de que os tempos medidos nas 10 replicações de um mesmo experimento apresentaram variações razoáveis. Como já mencionado na subseção 5.3.1, essas variações podem ser devidas ao fato de que, apesar da exclusividade do poder computacional das máquinas usadas, a simulação foi executada compartilhando a rede de comunicação entre todos os nós do sistema com outras aplicações. Portanto, esses erros podem ser experimentais e não existem indicativos de uma inadequação do modelo para expressar os tempos medidos.

A tabela 5.13 apresenta o grau de significância de cada efeito considerado na análise.

Conforme discutido na subseção 5.2.5, para um dado nível de significância, α, um fator X é estatisticamente significativo se o F calculado for superior ao valor tabelado de $F[\alpha$, grau de liberdade de X, grau de liberdade do erro$]$. Por indisponibilidade de dados, considerou-se o grau de liberdade do erro igual a 500 (superior ao grau dos erros em todos os testes realizados). Supondo-se que 90% é o nível de significância desejado, os valores de F calculados para os fatores algoritmo e balanceamento são superiores aos seus respectivos valores tabelados. Portanto, os fatores são estatisticamente significativos. Já a interação entre os fatores não é estatisticamente significativa porque o seu valor de F calculado é inferior ao respectivo valor de F tabelado.

Antes de se calcular os intervalos de confiança para o fator considerado, é preciso descobrir o valor da Distribuição Normal (t) a ser usado. Para um nível de significância de 90% ($\alpha = 0,9$), o valor de t[1 - (1 - $\alpha/2$), DFE] é 1,697.

A tabela 5.14 mostra o intervalo de confiança para o fator balanceamento na análise.

Os efeitos em que os intervalos de confiança correspondentes não incluem o zero são significativos, os demais, são não significativos. Observa-se que a alternativa *random* do fator algoritmo é não significativa e os demais níveis desse fator são significativos. Já fator balanceamento é significativo para todas as suas alternativas.

A fim de comparar os níveis *random*, *threshold* e *shortest* do fator algoritmo, pode-se calcular os intervalos de confiança entre as alternativas, usando a fórmula $\sum h_k \alpha_k$ (onde $\sum h_k = 0$, por definição):

- atribuindo-se para h os valores h_1 = -1, h_2 = 1 e h_3 = 0, tem-se que, para 90% de nível de significância, o intervalo de confiança é $(-8968.8 \pm 6242.359) = (-15211.159, -2726.441)$;
- atribuindo-se para h os valores h_1 = -1, h_2 = 0 e h_3 = 1, tem-se que, para 90% de nível de significância, o intervalo de confiança é (-1531.8 $\pm$ 6242.359) = $(-7774.159, 4710.559)$;
- atribuindo-se para h os valores h_1 = 0, h_2 = -1 e h_3 = 1, tem-se que, para 90% de nível de significância, o intervalo de confiança é (-10500.6 $\pm$ 6242.359) = $(-16742.959, -4258.241)$.

Na comparação dos níveis α_l e α_m, se o intervalo de confiança inclui o zero, α_l não é superior a α_m e vice-versa. Caso contrário, α_l é superior a α_m. Observa-se que os níveis *shortest* e *random* são superiores ao nível *threshold.*

Analogamente, para comparar os níveis CBS e CBC do fator balanceamento, pode-se calcular os intervalos de confiança entre as alternativas, usando a fórmula $\sum h_k \beta_k$ (onde $\sum h_k = 0$, por definição). Atribuindo-se para h os níveis h_1 = -1 e h_2 = 1, tem-se que, para 90% de nível de significância, o intervalo de confiança é $(-58116.8333 \pm 5096.86478) = (-63213.698, -53019.969)$.

Na comparação dos níveis β_l e β_m, se o intervalo de confiança inclui o zero, β_l não é superior a β_m e vice-versa. Caso contrário, β_l é superior a β_m. Observa-se que a alternativa CBS é superior a alternativa CBC.

Fazendo-se uma análise geral da tabelas 5.2, 5.3 e 5.4, já era possível observar as superioridades dos níveis *shortest* e *random* em relação ao nível *threshold*, e uma superioridade da alternativa CBS em relação a alternativa CBC. Com os cálculos dos intervalos de confiança entre as alternativas, foi possível comprovar a superioridades observadas nessa análise.

Considerando-se que o objetivo desse capítulo é estudar como o balanceamento de carga pode melhorar o desempenho da aplicação, a simulação paralela com o algoritmo de balanceamento *shortest* sem certeza de envio será usado para a análise proposta na seção 5.4.

É necessário enfatizar que o algoritmo *shortest* sem certeza de envio foi escolhido, em lugar do *random* sem certeza de envio, porque, além dele ter sido superior na subseção 5.3.1, esse algoritmo obteve melhor desempenho que o *random* no nível CBS (ver coluna CBS das tabelas 5.2 e 5.4).

Balanceamento	*random*	*threshold*	*shortest*	*Média por Linha*	*Efeito Linha*
CBS	55242	55143	53120.6	54501.86667	-3324.31667
CBC	62810.8	62371.6	58269.1	61150.5	3324.31667
Média por Coluna	59026.4	58757.3	55694.85	57286.18333	
Efeito Coluna	1200.21667	931.11667	-2131.33333		

Tabela 5.5: Média das repetições para cada combinação dos níveis

Balanceamento	*random*	*threshold*	*shortest*
CBS	-460.08333	-289.98333	750.06667
CBC	460.08333	289.98333	-750.06667

Tabela 5.6: Efeito da interação entre os fatores nos tempos medidos

Fator	*Soma dos Quadrados*	*Variação Percentual (%)*
	SST = 970951290	
	SSY = $2.01603\ 10^{11}$	
	SS0 = $2.0063205\ 10^{11}$	
Algoritmo	SSA = 137001601.4	14.11
Balanceamento	SSB = 663064879.3	68.29
Algoritmo/Balanceamento	SSAB = 17167340.23	1.77
Erros	SSE = 153717469.1	15.83

Tabela 5.7: Soma dos quadrados e variação percentual para os efeitos considerados

Componente	*Graus de Liberdade*	*Quadrado Médio*	*F Calculado*	*F Tabelado*
Algoritmo	2	68500800.7	24.064	2.31
Balanceamento	1	663064879.3	239.931	2.72
Algoritmo/Balanceamento	2	8583670.115	3.015	2.31
Erros	54	2846619.798		

Tabela 5.8: Valores para a análise de variância para o projeto de testes propostos, onde $S_e^2 = 2846619,798$

Fator	*Desvio Padrão*	*Intervalo de Confiança*
μ	(57826.18333 ± 369.633)	(57456.55033, 58195.81633)
Algoritmo		
random	(1200.21667 ± 522.740)	(677.47667, 1722.95667)
threshold	(931.11667 ± 522.740)	(408.37667, 1453.85667)
shortest	(-2131.33333 ± 522.740)	(-2654.07333, -1608.59333)
Balanceamento		
CBS	(-3324.31667 ± 369.633)	(-3693.94967, -2954.68367)
CBC	(3324.31667 ± 369.633)	(2954.68367, 3693.94967)

Tabela 5.9: Intervalos de confiança para os fatores

Balanceamento	*random*	*threshold*	*shortest*	*Média por Linha*	*Efeito Linha*
CBS	355854	370930	343771.2	356851.73333	-29058.41667
CBC	411008.3	413869.9	420027.5	414968.56667	29058.41667
Média por Coluna	383431.15	392399.95	381899.35	385910.15	
Efeito Coluna	-2479	6489.8	-4010.8		

Tabela 5.10: Média das repetições para cada combinação dos níveis

Balanceamento	*random*	*threshold*	*shortest*
CBS	1481.26667	7588.46667	-9069.73333
CBC	-1481.26667	-7588.46667	9069.73333

Tabela 5.11: Efeito da interação entre os fatores nos tempos medidos

Fator	*Soma dos Quadrados*	*Variação Percentual (%)*
	SST = 5.9541368 10^{10}	
	SSY = 8.99514 10^{12}	
	SS0 = 8.9355986 10^{12}	
Algoritmo	SSA = 1286989234	2.16
Balanceamento	SSB = 5.0663495 10^{10}	85.90
Algoritmo/Balanceamento	SSAB = 284078080.1	0.47
Erros	SSE = 7306805686	12.27

Tabela 5.12: Soma dos quadrados e variação percentual para os efeitos considerados

Componente	*Graus de Liberdade*	*Quadrado Médio*	*F Calculado*	*F Tabelado*
Algoritmo	2	643494617	4.756	2.31
Balanceamento	1	5.0663495 10^{10}	374.422	2.72
Algoritmo/Balanceamento	2	142039040.1	1.050	2.31
Erros	54	135311216.4		

Tabela 5.13: Valores para a análise de variância para o projeto de testes propostos, onde $S_e^2 = 135311216,4$

Fator	*Desvio Padrão*	*Intervalo de Confiança*
μ	(385910.15 $\pm$ 2548.432)	(383361.718, 388458.582)
Algoritmo		
random	(-2479 $\pm$ 3604.028)	(-6083.028, 1125.028)
threshold	(6489.8 $\pm$ 3604.028)	(2885.772, 10093.828)
shortest	(-4010.8 $\pm$ 3604.028)	(-7614.828, -406.772)
Balanceamento		
CBS	(-29058.41667 $\pm$ 2548.432)	(-31606.849, -26509.985)
CBC	(29058.41667 $\pm$ 2548.432)	(26509.985, 31606.849)

Tabela 5.14: Intervalos de confiança para os fatores

5.4 Análise do Balanceamento de Carga na Simulação Paralela

Nessa seção serão analisados os tempos de execução da simulação paralela do problema dos N-corpos antes e após a inclusão da biblioteca de balanceamento, levando-se em consideração, apenas, o tipo de balanceamento de carga utilizado no sistema.

A simulação foi executada variando-se o balanceamento de carga para os seguintes níveis:

- a simulação paralela sem balanceamento de carga (SBC);
- a simulação com o algoritmo de balanceamento de carga *shortest* sem a garantia de equilíbrio do número de partículas de cada um dos processadores participantes (CBS).

Conforme discutido na seção 5.2, é importante enfatizar que o projeto com 1 fator, proposto nessa seção, é uma simplificação do modelo descrito nas seções 5.2.1 a 5.2.6, e, portanto, não há a necessidade de se desenvolver uma nova modelagem para esse tipo de projeto.

Nas subseções seguintes serão feitas as análises para o sistema com 4096 e 16384 corpos.

5.4.1 Análise do Balanceamento para o Sistema com 4096 Corpos

A média para cada um dos níveis é calculada e apresentada na tabela 5.15. Essas médias são os tempos previstos pelo modelo.

A média global pode ser interpretada como uma indicação de que o tempo de execução de uma simulação com um balanceamento de carga médio é 64877,85 ms.

A tabela 5.16 apresenta os efeitos causados pelo fator balanceamento.

Os resultados podem ser interpretados da seguinte maneira:

- o tempo gasto na execução da simulação sem o balanceamento de carga é 11757,25 ms superior ao tempo gasto na simulação com um balanceamento médio;
- o tempo gasto na execução da simulação com balanceamento de carga sem certeza de envio é 11757,25 ms inferior ao tempo gasto na simulação com um balanceamento médio.

Na tabela 5.17 são exibidas as somas dos quadrados do fator, assim como a variação percentual causada por ele.

Observa-se que 97% da variação no tempo de execução é devida ao balanceamento de carga utilizado.

O modelo proposto não é capaz de explicar 3% da variação nos tempos de execução. Esse percentual pode ser devido ao fato de que os tempos medidos nas 10 replicações de um mesmo experimento apresentaram variações razoáveis. Como já mencionado na subseção 5.3.1, essas variações podem ser devidas ao fato de que, apesar da exclusividade do poder computacional das máquinas usadas, a simulação foi executada compartilhando a rede de comunicação entre todos os nós do sistema com outras aplicações. Portanto, esses erros podem ser experimentais e não existem indicativos de uma inadequação do modelo para expressar os tempos medidos.

A tabela 5.18 apresenta o grau de significância do fator considerado na análise.

Conforme discutido na subseção 5.2.5, para um dado nível de significância, α, um fator X é estatisticamente significativo se o F calculado for superior ao valor tabelado de $F[\alpha$, grau de liberdade de X, grau de liberdade do erro]. Supondo-se que 90% é o nível de significância desejado, o valor de F calculado para o fator balanceamento é superior ao seu respectivo valor tabelado. Portanto, esse efeito é estatisticamente significativo.

Antes de se calcular os intervalos de confiança para o fator considerado, é preciso descobrir o valor da Distribuição Normal (t) a ser usado. Para um nível de significância de 90% ($\alpha = 0,9$), o valor de t[1 - (1 - $\alpha/2$), DFE] é 1,734.

A tabela 5.19 mostra o intervalo de confiança para o fator balanceamento na análise.

Os efeitos em que os intervalos de confiança correspondentes não incluem o zero são significativos, os demais, são não significativos. Dessa forma, observa-se que o fator balanceamento é significativo.

A fim de comparar os níveis SBC e CBS do fator balanceamento, pode-se calcular os intervalos de confiança entre as alternativas, usando a fórmula $\sum h_k \beta_k$ (onde $\sum h_k = 0$, por definição). Atribuindo-se para h os valores h_1 = -1 e h_2 = 1, tem-se que, para 90% de nível de significância, o intervalo de confiança é $(-23514.5 \pm 11834.8469) = (-35349.3469, -11679.6531)$.

Na comparação dos níveis β_l e β_m, se o intervalo de confiança inclui o zero, β_l não é superior a β_m e vice-versa. Caso contrário, β_l é superior a β_m. Observa-se que o nível CBS é superior ao SBC.

Fazendo-se uma análise geral da tabela 5.4, já era possível observar uma superioridade do nível CBS em relação ao nível SBC. Com os cálculos dos intervalos de confiança entre as alternativas, foi possível comprovar a superioridade observada nessa análise.

5.4.2 Análise do Balanceamento para o Sistema com 16384 Corpos

A média para cada um dos níveis é calculada e apresentada na tabela 5.20. Essas médias são os tempos previstos pelo modelo.

A média global pode ser interpretada como uma indicação de que o tempo de execução de uma simulação com um balanceamento de carga médio é 412600,85 ms.

A tabela 5.21 apresenta os efeitos causados pelo fator balanceamento.

Os resultados podem ser interpretados da seguinte maneira:

- o tempo gasto na execução da simulação sem o balanceamento de carga é 68829,65 ms superior ao tempo gasto na simulação com um balanceamento médio;
- o tempo gasto na execução da simulação com balanceamento de carga sem certeza de envio é 68829,65 ms inferior ao tempo gasto na simulação com um balanceamento médio.

Na tabela 5.22 são exibidas as somas dos quadrados do fator, assim como a variação percentual causada por ele.

Observa-se que 98% da variação no tempo de execução é devida ao balanceamento de carga utilizado.

O modelo proposto não é capaz de explicar 2% da variação nos tempos de execução. Esse percentual pode ser devido ao fato de que os tempos medidos nas 10 replicações de um mesmo experimento apresentaram variações razoáveis. Como já mencionado na subseção 5.3.1, essas variações podem ser devidas ao fato de que, apesar da exclusividade do poder computacional das máquinas usadas, a simulação foi executada compartilhando a rede de comunicação entre todos os nós do sistema com outras aplicações. Portanto, esses erros podem ser experimentais e não existem indicativos de uma inadequação do modelo para expressar os tempos medidos.

A tabela 5.23 apresenta o grau de significância do fator considerado na análise.

Conforme discutido na subseção 5.2.5, para um dado nível de significância, α, um fator X é estatisticamente significativo se o F calculado for superior ao valor tabelado de $F[\alpha$, grau de liberdade de X, grau de liberdade do erro]. Supondo-se que 90% é o nível de significância desejado, o valor de F calculado para o fator balanceamento é superior ao seu respectivo valor tabelado. Portanto, esse efeito é estatisticamente significativo.

Antes de se calcular os intervalos de confiança para o fator considerado, é preciso descobrir o valor da Distribuição Normal (t) a ser usado. Para um nível de significância de 90% ($\alpha = 0,9$), o valor de t[1 - (1 - $\alpha/2$), DFE] é 1,734.

A tabela 5.24 mostra o intervalo de confiança para o fator balanceamento na análise.

Os efeitos em que os intervalos de confiança correspondentes não incluem o zero são significativos, os demais, são não significativos. Dessa forma, observa-se que o fator balanceamento é significativo.

A fim de comparar os níveis SBC e CBS do fator balanceamento, pode-se calcular os intervalos de confiança entre as alternativas, usando a fórmula $\sum h_k\beta_k$ (onde $\sum h_k = 0$, por definição). Atribuindo-se para h os valores h_1 = -1 e h_2 = 1, tem-se que, para 90% de nível de significância, o intervalo de confiança é $(-137659.3 \pm 53885.31788) = (-191544.61788, -83773.98212)$.

Na comparação dos níveis β_l e β_m, se o intervalo de confiança inclui o zero, β_l não é superior a β_m e vice-versa. Caso contrário, β_l é superior a β_m. Observa-se que o nível CBS é superior ao nível SBC.

Fazendo-se uma análise geral da tabela 5.4, já era possível observar uma superioridade do nível CBS em relação ao nível SBC. Com os cálculos dos intervalos de confiança entre as alternativas, foi possível comprovar a superioridade observada nessa análise.

SBC	CBS	*Média Global*
76635.1	53120.6	64877.85

Tabela 5.15: Média das repetições para cada combinação dos níveis

SBC	CBS
11757.25	-11757.25

Tabela 5.16: Efeito do fator balanceamento nos tempos medidos

Fator	*Soma dos Quadrados*	*Variação Percentual (%)*
	SST = 2847644204.55	
	SSY = 87030352617	
	SS0 = 84182708412.45	
Balanceamento	SSB = 2764658551.25	97.086
Erros	SSE = 82985654.3	2.914

Tabela 5.17: Soma dos quadrados e variação percentual para o fator considerado

Componente	*Graus de Liberdade*	*Quadrado Médio*	*F Calculado*	*F Tabelado*
Balanceamento	1	2764658551.25	599.6682	3.01
Erros	18	4610314.1278		

Tabela 5.18: Valores para a análise de variância para o projeto de testes propostos, onde $S_e^2 = 4610314,1278$

Fator	*Desvio Padrão*	*Intervalo de Confiança*
μ	(64877.85 $\pm$ 832.52895)	(64045.32105, 65710.37895)
Balanceamento		
SBC	(11757.25 $\pm$ 832.52895)	(10924.72105, 12589.77895)
CBS	(-11757.25 $\pm$ 832.52895)	(-12589.77895, 10924.72105)

Tabela 5.19: Intervalos de confiança para o fator balanceamento

SBC	CBS	*Média Global*
481430.5	343771.2	412600.85

Tabela 5.20: Média das repetições para cada combinação dos níveis

SBC	CBS
68829.65	-68829.65

Tabela 5.21: Efeito do fator balanceamento nos tempos medidos

Fator	*Soma dos Quadrados*	*Variação Percentual (%)*
	SST = 96470771585.5	
	SSY = $3.50126\ 10^{12}$	
	SS0 = 3404789228414	
Balanceamento	SSB = 94750414382.45	98.217
Erros	SSE = 1720357203.05	1.783

Tabela 5.22: Soma dos quadrados e variação percentual para o fator considerado

Componente	*Graus de Liberdade*	*Quadrado Médio*	*F Calculado*	*F Tabelado*
Balanceamento	1	94750414382.45	991.3682	3.01
Erros	18	95575400.16944		

Tabela 5.23: Valores para a análise de variância para o projeto de testes propostos, onde $S_e^2 = 95575400,16944$

Fator	*Desvio Padrão*	*Intervalo de Confiança*
μ	(421600.85 ± 3810.2674)	(417790.5826, 425411.1174)
Balanceamento		
SBC	(68829.65 ± 3810.2674)	(65019.3826, 72639.9174)
CBS	(-68829.65 ± 3810.2674)	(-72639.9174, -65019.3826)

Tabela 5.24: Intervalos de confiança para o fator balanceamento

5.5 Considerações Finais

Nas subseções 5.3.1 e 5.3.2, foram feitas análises dos tempos de execução da simulação paralela do problema dos N-corpos com os algoritmos de balanceamento de carga *random*, *threshold* e *shortest*. Essas análises revelam que a simulação com o algoritmo *shortest* obteve melhor desempenho que as simulações com os algoritmos *random* e *threshold*. Esses resultados são coerentes com as conclusões de [ELZ86], considerando-se que a superioridade do algoritmo *shortest* pode ser explicada pela simplicidade do sistema testado. Como somente 4 processadores participaram do sistema, ficou mais fácil escolher, em cada iteração que esse algoritmo era invocado, os processadores que estavam menos sobrecarregados. Essa facilidade na escolha garantiu uma boa redistribuição das tarefas da aplicação e, portanto, diminuiu os tempos do algoritmo em relação aos demais.

Os valores obtidos na execução da simulação paralela com os algoritmos de balanceamento *random*, *difusão* e *redistribuição completa*, discutidos em [FG96], também estão em concordância com o que foi concluído em [ELZ86]. Nessas execuções, prova-se que, para 16 processadores, o algoritmo *random* é 12% mais rápido que o algoritmo de *difusão*, e 30% mais rápido que o algoritmo de *redistribuição completa*. Para um sistema pequeno, 2 ou 4 processadores participantes, as simulações com os algoritmos *random* e *difusão* têm, praticamente, o mesmo tempo de execução. Conforme discutido no parágrafo anterior, a simulação com o algoritmo de balanceamento *difusão* só obtém o mesmo tempo de execução do algoritmo *random* porque um processador sobrecarregado descobre, mais facilmente, a porcentagem de tarefas a serem recebidas por cada um de seus vizinhos.

Nas subseções 5.3.1 e 5.3.2, também foi observado que a simulação paralela com balanceamento de carga sem certeza de envio obteve melhor desempenho que a simulação paralela com balanceamento de carga com certeza de envio. Essa constatação pode ser explicada da seguinte maneira:

- quando ocorre um desequilíbrio nas tarefas dos processadores participantes da simulação paralela com balanceamento de carga com certeza de envio, os processadores sobrecarregados ordenam seus corpos e, depois dessa ordenação, di-

minuem suas regiões de um valor calculado a partir da ordenação e enviam uma quantidade do excesso de seus corpos para os processadores subcarregados (que aumentam suas regiões do referido valor para poder receber esses excessos). É interessante observar que, além da ordenação ser uma operação dispendiosa, esse balanceamento não garante que, ao final da iteração, os processadores do sistema fiquem com o número de tarefas equilibrado, pois um processador que estava subcarregado pode, depois de receber o excesso de tarefas de outro processador, tornar-se sobrecarregado.
Na simulação com balanceamento de carga sem certeza de envio, os processadores sobrecarregados diminuem suas regiões de um valor constante e transferem os corpos que não pertencem mais a eles para os processadores subcarregados (que aumentam suas regiões dessa mesma constante para poder receber esses corpos). Da mesma forma que a simulação com certeza de envio, esse balanceamento não garante que o sistema vai ficar equilibrado ao fim da iteração, pois um processador que estava subcarregado pode, depois de receber o excesso de tarefas de outro processador, tornar-se sobrecarregado. Entretanto, nesse balanceamento não se gasta tempo na fase de ordenação de corpos.

Nas subseções 5.4.1 e 5.4.2 foram realizadas análises que revelam que a simulação paralela com o algoritmo de balanceamento de carga *shortest* sem certeza de envio obteve o melhor desempenho em relação à simulação paralela pura. Essa conclusão pode ser explicada da seguinte maneira:

- quando ocorre um desequilíbrio nas tarefas dos processadores participantes da aplicação paralela original, os processadores que estão subcarregados devem ficar esperando, na fase de sincronização, pelos processadores sobrecarregados, em todas as iterações restantes da simulação, aumentando, assim, o tempo de execução do sistema. Na simulação com balanceamento de carga, tenta-se manter um número de corpos equilibrado entre todos os processadores desse sistema, a fim de minimizar os tempos gastos, desnecessariamente, na fase de sincronização dos processadores e, diminuindo, portanto, os tempos gastos nas execuções das simulações com balanceamento.

Capítulo 6

Conclusões

Com um maior entendimento do problema do balanceamento de carga em ambientes distribuídos, adquirido no desenvolvimento desse trabalho, foi possível estabelecer um paralelo entre as características básicas de alguns algoritmos de balanceamento, e desenvolver uma biblioteca básica extensível que serve para implementar uma grande variedade desses algoritmos.

Grande parte do tempo gasto nesse trabalho foi relacionado com a implementação da biblioteca de balanceamento de carga. Um dos obstáculos encontrados nessa fase foi a dificuldade no entendimento do código da aplicação. Algumas rotinas da implementação da simulação paralela estavam mal documentadas e houve a necessidade de se analisar o funcionamento dessa simulação para entender como essas rotinas funcionavam.

Sanada essa dificuldade, a biblioteca de balanceamento foi desenvolvida usando algumas das rotinas da aplicação, e o uso dessa implementação na simulação paralela tornou-se uma solução simples para o problema do desequilíbrio de tarefas entre os processadores do sistema. Para que o usuário faça uso de um novo algoritmo de redistribuição de tarefas, basta que ele implemente esse algoritmo usando as rotinas já existentes na referida biblioteca.

Informalmente, nos experimentos realizados, a execução da simulação paralela do problema dos N-corpos com a biblioteca de balanceamento foi, em média, 40% mais rápida que a execução da simulação paralela original. Isso se deve ao fato de que, quando ocorre um desequilíbrio de partículas na simulação paralela pura, o *overhead* desse desequilíbrio se propaga até o fim da execução da referida aplicação. Já na simulação com a biblioteca de balanceamento, quando ocorre um desequilíbrio de corpos, o algoritmo de balanceamento é invocado, a fim de equilibrar, entre todos os processadores do sistema, a quantidade de partículas, impedindo que o referido *overhead* aconteça. Assim, o uso da biblioteca de balanceamento na simulação paralela é imprescindível para obter uma melhoria no desempenho dessa simulação.

O fato da execução da simulação paralela com um algoritmo de balanceamento de carga relativamente complexo, *shortest*, ser melhor que a execução da referida simulação com os algoritmos de balanceamento considerados simples, *random* e *threshold*, pode ser explicado pelo tamanho do sistema. Em geral, para sistemas com 2 ou 4 processadores, os algoritmos de balanceamento de carga complexos além de obterem o mesmo tempo de execução dos algoritmos simples, fazem uma melhor redistribuição de tarefas da aplicação. Quando o sistema cresce, como discutido em [FG96], os algoritmos complexos têm *overheads* adicionais na tentativa da melhor redistribuição de tarefas, o que não ocorre com os algoritmos simples. Em sistemas pequenos, 2 ou 4 processadores, se o objetivo é estudar como o balanceamento de carga pode melhorar o desempenho da aplicação, é aconselhável, pelos resultados obtidos, o uso de um algoritmo de balanceamento complexo. Em sistemas com mais de 4 processadores, é aconselhável o uso de algoritmo de balanceamento simples.

O tempo de execução da simulação paralela com o algoritmo *shortest* sem certeza de envio é, em média, 10% mais rápido que o tempo de execução da referida aplicação com o algoritmo *shortest* com certeza de envio (essa observação é válida para os algoritmos de balanceamento de carga *random* e *threshold*). Isso se deve ao fato de que, quando ocorre um desequilíbrio de corpos na simulação com o algoritmo *shortest* com certeza de envio, os processadores sobrecarregados gastam muito tempo na fase de ordenação de seus corpos, o que não acontece na simulação paralela com o algoritmo *shortest* sem certeza de envio. É necessário enfatizar que essas duas alternativas de balanceamento de carga não garantem um número de partículas equilibrado ao final da iteração, pois um processador que estava subcarregado pode, depois de receber o excesso de corpos de outro processador, tornar-se sobrecarregado. Se o objetivo é estudar como o balanceamento de carga pode melhorar o desempenho da aplicação, é aconselhável, pelos resultados obtidos, o uso simulação paralela com o algoritmo de balanceamento sem certeza de envio.

Foram feitas pesquisas sobre o balanceamento de carga que não foram citadas no trabalho, mas que, certamente, são importantes para o entendimento do assunto e, portanto, servem como referências: [HJ87], [RR96], [SH94], [KGR94], [Han95], [GKS94a], [GKS94b], [LMR91], [LM93], [CK88], [Tal92], [TT85], [CHLE80], [Bok81], [APG94], [ZZWD93], [CP96], [LM92] e [ZM94].

6.1 Trabalhos Futuros

Uma série de propostas para trabalhos futuros pode ser apresentada:

- realizar estudos comparativos entre os tempos gastos nas execuções da simulação paralela antes e após a inclusão da biblioteca de balanceamento, aumentando o número de processadores participantes do sistema;
- adaptar a biblioteca de balanceamento e a simulação paralela do problema dos N-corpos para a utilização de um sistema tridimensional de partículas. Nesse caso, a simulação deverá entender as posições e velocidades espaciais dos corpos e a biblioteca deverá alterar a forma de redesignar os domínios entre os processadores, quando o sistema estiver com desequilíbrio de tarefas;
- usar uma outra aplicação paralela síncrona para provar a aplicabilidade da biblioteca desenvolvida. Nesse caso, a biblioteca específica da aplicação, descrita no capítulo 4, deverá ser alterada, para se encaixar à nova aplicação, e deve-se verificar a ocorrência de uma melhoria no tempo dessa aplicação com a inclusão da biblioteca de balanceamento.

Referências Bibliográficas

[APG94] S. Aluru, G. M. Prabhu, and J. Gustafson. Truly Distribution-Independent Algorithms for the N-Body Problem. *IEEE*, pages 420–428, 1994.

[BF96] G. Bernard and B. Folliot. Caractéristiques Générales du Placement Dynamique: Syntèse et Problématique. In *Ecole Placement Dynamique et Répartition de Charge*, 1996.

[BH86] J. Barnes and P. Hut. A Hierarchical $O(NlogN)$ Force Calculation Algorithm. *Nature*, 324:446–449, 1986.

[Bok81] S. H. Bokhary. On the mapping problem. *IEEE Transactionson Computers*, 30(3), 1981.

[Cha80] J. A. Chahon. Modelos Analíticos e de Simulação para Sistemas de Filas com e sem Prioridade. Dissertação de Mestrado, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Outubro de 1980.

[CHLE80] W. W. Chu, L. J. Holloway, M. Lan, and K. Efe. Task Allocation in Distributed Data Processing. *IEEE*, pages 57–69, 1980.

[CK88] T. L. Casavant and J. G. Kuhl. A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems. *IEEE Transactions on Software Engineering*, 14:141–154, 1988.

[CP96] G. Cabillic and I. Puaut. Répartition de Charge dans Stardust: un Environnement pour l'Exécution d'Applications Parallèles en Milieu Hétérogène. In *Ecole Placement Dynamique et Répartition de Charge*, 1996.

[CT92] K. M. Chandy and S. Taylor. *An Introduction to Parallel Programming*. Jones and Bartlett Publishers, Boston, 1992.

[Cyb89] G. Cybenko. Dynamic Load Balancing for Distributed Memory Multiprocessors. *Journal Parallel Distributed Computing*, 7(2):279–301, 1989.

[ELZ86] D. L. Eager, E. D. Lazowska, and J. Zahorjan. Adaptive Load Sharing in Homogeneous Distributed Systems. *IEEE Transactions on Software Engineering*, 12(5):662–675, 1986.

[FG93] M. A. Franklin and V. Govindan. The N-body Problem: Distributed System Load Balancing and Performance Evaluation. In *Proc. 6th International Conference on Parallel and Distributed Computing Systems*, 1993.

[FG96] M. A. Franklin and V. Govindan. A General Matrix Iterative Model for Dynamic Load Balancing. *Parallel Computing*, 22:969–989, 1996.

[GKS94a] A. Grama, V. Kumar, and A. Sameh. N-Body Simulations Using Message Passing Parallel Computers. In *SIAM Conference on Parallel Processing*, 1994.

[GKS94b] A. Grama, V. Kumar, and A. Sameh. Scalable Parallel Formulations of the Barnes-Hut Method for N-Body Simulations. In *Supercomputing'94 Proceedings*, 1994.

[Han95] P. B. Hansen. *Studies in Computational Science: Parallel Programming Paradigms.* Prentice-Hall, New Jersey, 1995.

[HJ87] A. Hac and X. Jin. Dynamic Load Balancing in Distributed System Using a Decentralized Algorithm. In *Proceedings of the 7th International Conference Distributed Computing Systems*, pages 170–184, 1987.

[Jai91] R. Jain. *Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling.* Digital Equipment Corporation, John Wiley & Sons, inc., Littleton, Massachusetts, 1991.

[KGR94] V. Kumar, A. Grama, and V. N. Rao. Scalable Load Balancing Techniques for Parallel Computers. *Journal of Parallel and Distributed Computing*, 22(1):60–79, 1994.

[LLC96] G. Lee, H. Lee, and J. Cho. A Prediction-based Adaptive Location Policy for Distributed Load Balancing. *Journal of Systems Architecture*, 42:1–18, 1996.

[LM92] R. Lüling and B. Monien. Load Balancing for Distributed Branch & Bound Algorithms. In *International Parallel Processing Symposium (IPPS'92)*, 1992.

[LM93] R. Lüling and B. Monien. A Dynamic Algorithm with Provable Good Performance. In *Proceedings of thre 5th ACM Symposium on Parallel Algorithms and Architectures (SPAA'93)*, 1993.

[LMR91] R. Lüling, B. Monien, and F. Ramme. Load Balancing in Large Networks: A Comparative Study. In *Proceedings of thre 3rd IEEE Symposium on Parallel and Distributed Processing (SPDP'91)*, 1991.

[MD94] K. B. Mahieddine and P. M. Dew. A Periodic Symmetrically Initiated Load Balancing Algorithm for Distributed Systems. *Operating Systems Review*, 28(1):66–77, 1994.

[Qui94] M. J. Quinn. *Parallel Computing: Theory and Practice.* Mc-Graw Hill, Oregon State University, second edition edition, 1994.

[RR96] M. R. C. Retiz and P. Rouchon. Étude Quantitative des Stratégies pour la Régulation Dynamique de Charge. In *Ecole Placement Dynamique et Répartition de Charge*, 1996.

[SH94] K. G. Shin and C. J. Hou. Design and Evaluation of Effective Load Sharing in Distributed Real-Time Systems. *IEEE Transactions on Parallel and Distributed Systems*, 5(7):704–719, 1994.

[Tal92] E. Talbi. Etude Expérimentale d'Algorithmes de Placement de Processus. *Letter du Transputer*, 15:7–26, 1992.

[TT85] A. N. Tantawi and D. Towsley. Optimal Static Load Balancing in Distributed Computer Systems. *Journal of the Association for Computing Machinery*, 32(2):445–465, 1985.

[Uch95] R. C. Uchôa. Suporte ao Monitoramento e Controle de Carga em Ambientes Distribuídos. Dissertação de Mestrado, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Agosto de 1995.

[Vel94] E. F. Velde. *Concurrent Scientific Computing.* Spring Verlag, New York, 1994.

[ZM94] T. F. Znati and R. G. Melhem. A Uniform Framework for Dynamic Load Balancing Strategies in Distributed Processing Systems. *Journal of Parallel and Distributed Computing*, 23:246–255, 1994.

[ZZWD93] S. Zhou, X. Zheng, J. Wang, and P. Delisle. Utopia: a Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems. *Software-Practice and Experience*, 23(12):1305–1336, 1993.