X Simpósio Brasileiro de Arquitetura de Computadores

Balanceamento de Carga em uma Implementação Distribuída do Problema dos N-Corpos

Isabel Cristina Mello Rosseti
Noemi Rodriguez
Departamento de Informática — PUC-Rio
Rua M. S. Vicente 225, Gávea
22453-900 — Rio de Janeiro — RJ
rosseti,noemi@inf.puc-rio.br

Resumo
O balanceamento de carga é um fator fundamental no uso efetivo de ambientes de memória distribuída para execução de aplicações paralelas. Este trabalho estuda a aplicação de três algoritmos de balanceamento a uma simulação paralela de um sistema de N-corpos. Um modelo analítico unificador, o Modelo Iterativo da Matriz, é usado como base para a implementação de uma biblioteca única de balanceamento de carga. O trabalho utiliza técnicas de projeto e análise de experimentos para examinar o desempenho da aplicação com uso dessa biblioteca.

Palavras-chave
Balanceamento de carga, algoritmos de balanceamento de carga, Modelo Iterativo de Matriz, problema dos N-corpos.

Abstract
Load balancing has become a fundamental factor in the effective use of distributed-memory environments for parallel computation. This paper studies the application of load-balancing algorithms to a parallel simulation of the N-body problem. The Matrix Iterative Model is used as an unifying basis for the implementation of a load-balancing library. Experimental design and analysis techniques are used to examine application performance with the use of this library.

Keywords
Load balancing, load balancing algorithms, Matrix Iterative Model, N-body problem.

1 Introdução

Nesse trabalho, o problema de balanceamento de carga é estudado no contexto de uma aplicação paralela particular; descreve-se a utilização de três algoritmos de balanceamento na simulação paralela da evolução de um sistema de N-corpos, baseada no algoritmo de Barnes-Hut.

Franklin e Govindan propõem um modelo [FG96], denominado *Modelo Iterativo de Matriz* (MIM), que permite representar uma gama bastante grande de algoritmos de balanceamento. O MIM foi utilizado neste trabalho como base para a organização da implementação do balanceamento de carga. Essa abordagem permitiu que fossem minimizadas as diferenças entre as implementações dos diversos algoritmos. Foi desenvolvida uma biblioteca onde é enfatizada a separação da funcionalidade comum a todos os algoritmos.

A seção 2 apresenta a terminologia utilizada e discute, de forma resumida, algumas questões relevantes para o problema de balanceamento de carga. A seção 3 descreve o problema dos N-corpos e a simulação paralela utilizada. Na seção 4, discute-se a implementação da biblioteca de balanceamento. A seção 5 apresenta os resultados dos experimentos realizados. Finalmente, a seção 6 discute algumas conclusões deste trabalho.

2 Balanceamento de Carga

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

2.1 Modelo Iterativo de Matriz

O Modelo Iterativo de Matriz (MIM), proposto em [FG96], é um modelo analítico que pode ser usado para representar um conjunto de algoritmos de balanceamento de carga. O modelo descreve a distribuição de tarefas entre os processadores devido ao algoritmo de balanceamento.

O modelo assume 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 e (d) a aplicação pertence à classe das aplicações iterativas síncronas.

O conjunto de processadores disponíveis é denotado por $P = \{P_1, ..., P_p\}$, onde p é o número de processadores.

O MIM descreve a distribuição das tarefas de cada processador do sistema, em cada uma das iterações k ($k = 1, 2, ...$), 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 na iteração $(k+1)$, $w(k+1)$, 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 e (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:

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

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.

A decisão de migração de tarefas é representada por uma matriz $p \times p$, $M(k)$, onde $M_{ij}(k)$ indica a quantidade de tarefas, α, que o processador P_i decide enviar para P_j. Essa decisão fica necessariamente restrita 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, $H_{ij}(k) = 0$.

A equação {2.1} pode ser reescrita como segue:

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

onde $\lambda(k)$ e $\mu(k)$ são vetores que denotam a taxa de chegada e a taxa de saída, respectivamente. Na equação {2.2}, a primeira somatória 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.

2.2 Algoritmos de Balanceamento de Carga Apresentados Segundo o MIM

Em [FG96], os procedimentos de balanceamento de carga *random*, *difusão* e *redistribuição completa* foram adaptados ao MIM. Nessa subseção, os algoritmos de balanceamento de carga *random*, *threshold* e *shortest* serão adaptados a esse modelo. Esses foram os algoritmos escolhidos para os experimentos com a simulação paralela dos N-corpos, por representarem políticas bastante distintas de informação e de localização.

Nos três algoritmos a seguir, um processador P_i decide enviar uma fração α do excesso de suas tarefas para outro processador, P_e, quando o número de tarefas atribuídas a ele, $w_i(k)$, é maior que um valor pré-definido, $w_i^*(k)$:

- *random* [ELZ86], [FG96], [LLC96] e [MD94]→ nesse algoritmo, 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;

- *threshold* [ELZ86], [LLC96] e [MD94] → nesse algoritmo, 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;

- *shortest* [ELZ86] e [LLC96] → nesse algoritmo, 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.

Para esses algoritmos, 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 $j \neq e$.

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

$$w_i(k+1) = w_i(k) - \alpha A_{ii}(k)(w_i(k) - w_i^*(k)) + \sum_{j \neq i} A_{jj}(k) M_{ji}(k)(w_j(k) - w_j^*(k)) + \lambda_i(k) - \mu_i(k) \quad \{2.3\}$$

3 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. Essas técnicas são amplamente aplicadas a problemas de astrofísica, de dinâmica dos fluidos, entre outros.

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. Essa simulação segue sobre um número pré-determinado 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 usada neste trabalho, 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. 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. Nessa fase, cada processador troca $log\ p$ mensagens com os outros $p - 1$ 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.

4 Biblioteca de Balanceamento de Carga

Utilizando o MIM como base, podemos capturar as partes comuns entre os vários algoritmos de balanceamento de carga, descrevendo um algoritmo de balanceamento genérico através dos seguintes passos:

1. calcula matriz A (define participação de processadores)
2. calcula matriz H (determina vizinhanças correntes)

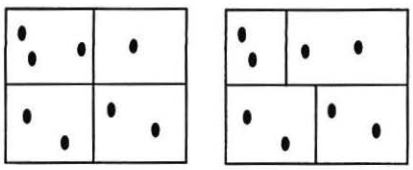

Figura 1: Simulação dos N-corpos antes e após a invocação do algoritmo de balanceamento de carga

3. calcula matriz M (determina destino de tarefas transferidas)

4. executa trocas de carga entre processadores descritas por A, H e M

Os três primeiros passos tratam das matrizes $A(k)$, $H(k)$ e $M(k)$, e foram implementados por funções $MatrixA$, $MatrixH$ e $MatrixM$, respectivamente. O último passo foi implementado pela função $ExchangeLoad$.

É necessário enfatizar que todos os algoritmos de balanceamento de carga que podem ser descritos pelo MIM se encaixam nessa mesma descrição do algoritmo geral. Para esses algoritmos, as implementações de $MatrixH$ e $ExchangeLoad$ são idênticas. Já as implementações de $MatrixA$ e $MatrixM$ são especiais para cada algoritmo de balanceamento. Do ponto de vista da aplicação, a escolha de algoritmo de balanceamento é transparente. Ao final de cada iteração da simulação, a aplicação chama um procedimento de balanceamento de carga, que basicamente chama as funções $MatrixA$, $MatrixH$, $MatrixM$ e $ExchangeLoad$. No momento de ligação, o programador escolhe qual instância específica de $MatrixA$ e $MatrixM$ ele deseja utilizar (através de um arquivo *Makefile*).

Para estender a biblioteca desenvolvida com um novo algoritmo, seria necessário apenas desenvolver uma nova implementação das funções $MatrixM$ e $MatrixA$. Em particular, nos três algoritmos estudados a implementação de $MatrixA$ utilizada foi a mesma, uma vez que todos apresentam a mesma política de transferência, baseada no fato do número corrente de tarefas ultrapassar um valor pré-definido (essa política é bastante comum nos assim chamados algoritmos *iniciados pelo remetente*).

5 Análise de Resultados

Os experimentos foram realizados na máquina paralela IBM SP/2 do Núcleo de Computação Eletrônica (NCE) da Universidade Federal do Rio de Janeiro (UFRJ). Os dados foram obtidos utilizando 4 nós da máquina de maneira exclusiva, isto é, a simulação foi executada sem o compartilhamento do poder computacional dos nós com nenhuma outra aplicação. Cada experimento foi repetido 10 vezes.

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 1). Essa similaridade do tempo paralelo é devido ao tempo gasto na fase de troca de informações entre

os processadores. Conforme mencionado na seção 3, 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 custo 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 custo 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.

A fim de validar a implementação descrita na seção 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, foram realizados 2 projetos de experimentos [Jai91]. Esses projetos serão tratados nas subseções seguintes.

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

Tabela 1: Tempos (em ms) de cada uma das 10 execuções das simulações, sequencial e paralela.

5.1 Análise das Alternativas de Balanceamento de Carga

O projeto com 2 fatores, descrito em [Jai91], foi 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 balanceamento de carga 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 ou vai aumentar ou vai diminuir a

fronteira de seu domínio de um valor pré-determinado, l, e vai fazer as modificações nas fronteiras desse valor. Nesse caso, não se sabe a quantidade de partículas que um nó com sobrecarga enviará. É importante observar que esse balanceamento não garante que o sistema vai ficar equilibrado ao fim da iteração;

(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. Nesse opção, a redesignação de domínios feita no algoritmo de balanceamento faz uma ordenação de partículas e, a partir dessa ordenação, calcula-se o valor l. Com esse valor calculado, faz-se as modificações nas fronteiras. Nesse caso, sabe-se a quantidade de partículas que um nó com sobrecarga enviará. É importante observar que 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.

As análises do projeto 2-fatorial, descritas em [Ros98], revelaram que a simulação com o algoritmo *shortest* obtiveram melhor desempenho que as simulações com os algoritmos *random* e *threshold* (ver tabela 2). 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.

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, diminuem 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. 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. Entretanto, nesse balanceamento não se gasta tempo na fase de ordenação de corpos.

Considerando-se que o objetivo desse trabalho 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 comparado com a simulação paralela original na subseção 5.2.

5.2 Análise do Balanceamento de Carga na Simulação Paralela

Nessa subseção, o projeto com 1 fator [Jai91], 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) → 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, isto é, o algoritmo *shortest* sem certeza de envio.

O modelo do projeto 1-fatorial 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 adaptar o modelo descrito na subseção 5.1 para realizar a análise descrita nessa subseção.

A análise do projeto de experimentos 1-fatorial, descrita em [Ros98], revelou 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.

6 Conclusões

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 trabalho adicional 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 trabalho adicional aconteça. Assim, o uso da biblioteca de balanceamento na simulação paralela é imprescindível para obter uma melhoria no desempenho dessa simulação.

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

partículas	*random*		*threshold*		*shortest*	
	CBS	CBC	CBS	CBC	CBS	CBC
4096	54114	63796	54857	61890	52374	59071
	50372	62724	55212	60784	50682	59066
	55502	62734	55947	62596	56388	58037
	57293	62759	54754	61605	54198	57322
	53624	61480	55507	62143	53122	59579
	56261	64509	56000	62029	52851	56409
	52960	61250	54763	61296	52838	59092
	58263	62655	52372	62123	50273	59079
	58802	64102	55125	65343	54158	58672
	55198	62099	57006	63997	54322	56364
16384	365677	408124	369368	414846	352101	423809
	355702	407559	392752	437041	342705	417687
	344331	425001	378279	409581	348132	404408
	377706	401598	381000	405884	342948	418022
	361188	408636	357954	408678	330032	422257
	360509	408736	357644	409825	338566	429812
	341438	409079	370662	418337	345639	422233
	339157	407894	373869	416590	342104	429029
	365278	409351	373769	408300	345361	421760
	347464	424015	354003	409617	350124	411258

Tabela 2: Tempos (em ms) de cada uma das 10 execuções da simulação com a biblioteca de balanceamento.

Fator	*Variação Percentual (%)*	*Fator*	*Variação Percentual (%)*
Algoritmo	14.11	Algoritmo	2.16
Balanceamento	68.29	Balanceamento	85.90
Alg/Bal	1.77	Alg/Bal	0.47
Erros	15.83	Erros	12.27

Tabela 3: Variação percentual dos efeitos considerados para o sistema com 4096 e 16384 corpos.

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 nenhuma das duas alternativas de balanceamento de carga garante 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. Essa possibilidade é uma decorrência inevitável da localidade de decisões nos algoritmos estudados.

Agradecimentos

Agradecemos ao Núcleo de Computação Eletrônica (NCE) da UFRJ, por permitir e apoiar a utilização de sua máquina paralela, e ao Prof. Celso Ribeiro, pelas sugestões dadas durante a condução deste trabalho.

Referências Bibliográficas

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

Fator	*Variação Percentual (%)*	*Fator*	*Variação Percentual (%)*
Balanceamento	97.086	Balanceamento	98.217
Erros	2.914	Erros	1.783

Tabela 4: Variação percentual do fator balanceamento para o sistema com 4096 e 16384 corpos.

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

[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.

[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.

[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.

[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.

[Ros98] I. C. M. Rosseti. Uma Biblioteca para Balanceamento de Carga em Ambientes Distribuídos. Dissertação de Mestrado, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Março de 1998.

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