Geração paralela de estimativas iniciais para sistemas iterativos*

Robertino M. Santiago Jr[1], Guilherme D. Machado[2], Vladimir F. Cabral[2], Lúcio Cardozo Filho[2], Ronaldo Augusto de Lara Gonçalves[1]

[1]UEM - Universidade Estadual de Maringá
Departamento de Informática
Av. Colombo, 5.790, Jd. Universitário
CEP 87020-900 - Maringá, PR

robertino@facinor.br, ronaldo@din.uem.br

[2]UEM - Universidade Estadual de Maringá
Departamento de Engenharia Química
Av. Colombo, 5.790, Jd. Universitário
CEP 87020-900 - Maringá, PR

guilherme.duenhas@gmail.com, vladimir@deq.uem.br, cardozo@deq.uem.br

Resumo. *Computação paralela tem sido muito utilizada em várias áreas de conhecimento para resolver problemas de alta complexidade, os quais normalmente exigem tempo de processamento elevado quando solucionados sequencialmente. A solução iterativa de sistemas de equações não lineares é considerada um desses problemas para um grande número de aplicações, principalmente quando o sistema possui muitas dimensões, o que justifica sua paralelização. Na engenharia química, por exemplo, sistemas de equações não lineares estão sendo aplicados à simulação computacional de uma coluna de destilação reativa, nos quais a obtenção de estimativas iniciais também é um trabalho muito árduo, pois precisa ser resolvida usando o próprio sistema iterativo. Neste trabalho, um algoritmo para gerar essas estimativas foi paralelizado em 2 modelos: regular e balanceado. Os resultados apresentam as vantagens do modelo balanceado em várias situações.*

1. Introdução

Nas mais diversas áreas científicas, existem experimentos que são muito difíceis de serem realizados, muitos dos quais são caros e até perigosos, requerendo equipamentos especiais e procedimentos rígidos para evitar acidentes. Através do uso de simulação computacional, é possível estimar resultados e comportamentos desses experimentos evitando desta forma danos e gastos desnecessários. Entretanto, essa simulação computacional pode exigir um tempo de processamento muito elevado se executada sobre um único processador, de modo sequencial. Como solução para reduzir este tempo, dividi-la e distribuí-la entre várias unidades de processamento para ser executada paralelamente, pode proporcionar a obtenção de resultados de forma bem mais rápida.

*Projeto apoiado pela Fundação Araucária (PAC-Clusters)

Nesse contexto, um dos problemas que demanda grande quantidade de poder computacional é a solução iterativa de sistemas de equações não lineares. O nível de exigência do poder de processamento aumenta em função do tamanho dos sistemas envolvidos. Uma das áreas de conhecimento que possui muitos sistemas iterativos de equações não lineares é a Engenharia Química, em que podemos citar a simulação da produção de ésteres de ácidos graxos (biodiesel) em colunas de destilação reativa, a qual utiliza um modelo matemático baseado em um sistema iterativo de equações não lineares [1]. Sabe-se que convergência da solução desse sistema depende das estimativas iniciais para as suas incógnitas, o que demanda muito esforço e tempo para serem encontradas. Entretanto, esse conjunto de estimativas iniciais pode ser gerado por meio de um algoritmo de subdivisão [2].

Porém, a geração de estimativas iniciais pelo algoritmo de subdivisão também pode ser um processo muito demorado quando envolve sistemas com um elevado número de incógnitas (dimensões), porque utiliza o próprio sistema iterativo para ser executado. Desta forma, o presente trabalho tem como objetivo paralelizar o algoritmo utilizado para gerar as estimativas iniciais para permitir que os resultados sejam obtidos em um tempo menor. Dois métodos de paralelização foram experimentados: regular e balanceado.

O presente artigo está organizado conforme segue. Na Seção 2 são descritos alguns conceitos sobre computação paralela. A Seção 3 apresenta o algoritmo utilizado neste trabalho. Na Seção 4 são apresentados trabalhos relacionados. Na Seção 5 é descrita a metodologia de paralelização utilizada neste trabalho. Os resultados são apresentados na Seção 6 e na Seção 7 são relatadas as conclusões obtidas.

2. Computação Paralela

Existem essencialmente dois tipos de paralelismo [3]: funcional e de dados. O paralelismo funcional ocorre quando uma aplicação é formada por segmentos de códigos que possuem funções diferentes e que podem ser executados em paralelo, atuando ou não sobre o mesmo conjunto de dados. O paralelismo de dados [4] ocorre quando os dados são divididos entre os elementos de processamento para serem processados paralelamente, por um mesmo segmento do código ou não. Os segmentos de códigos que executam em paralelo normalmente são denominados tarefas.

O número e o tamanho das tarefas em que um problema é dividido determinam a granulosidade da aplicação [5] e pode ser dividida em fina e grossa. Granulosidade fina é dita quando a divisão dos dados origina um grande número de pequenas tarefas, requerendo uma quantidade maior de comunicação entre os processadores envolvidos. A granulosidade grossa é o obtida quando a divisão do problema gera um número pequeno de grandes tarefas, requerendo uma quantidade menor de comunicação.

Tradicionalmente, as arquiteturas paralelas são classificadas em multiprocessadores e multicomputadores, que segundo Tanenbaum [6] a diferença fundamental entre as duas é a presença ou ausência de memória compartilhada. Multiprocessadores compartilham um único espaço de endereçamento de memória, podendo a memória estar fisicamente centralizada ou distribuída. Dois processadores podem se comunicar apenas escrevendo e lendo dados na memória compartilhada. Os multicomputadores não possuem um único espaço de endereçamento de memória compartilhado. Cada processador possui sua própria memória, sendo essa acessível

somente a ele mesmo. A comunicação entre dois processadores ocorre através do uso de mensagens que são enviadas através da rede de interconexão. No contexto dos multicomputadores aparecem os *clusters*, muitas vezes contendo elementos multiprocessadores.

Um *cluster* pode ser definido como um conjunto de computadores, com um ou mais elementos de processamento, conhecidos como nós, interligados por meio de uma rede de interconexão, possuindo software de suporte acessível ao usuário para organizar e controlar as tarefas de computação concorrente que colaboram no processamento de uma aplicação [7]. Os *clusters* de computadores têm se mostrado uma excelente alternativa de baixo custo para os tradicionais supercomputadores em laboratórios científicos, universidades e centros de supercomputação, além de serem empregados para fins comerciais. Atualmente, conforme a lista dos 500 maiores supercomputadores do mundo [8], 82.20% são *clusters* de computadores.

Em um *cluster* de computadores, as memórias estão localizadas fisicamente em cada computador e a única forma de um processador acessar dados na memória de outro computador é através da rede de comunicação. Para realizar essa comunicação é utilizado o recurso de passagem de mensagem. MPI (*Message Passing Interface*) é um padrão que especifica a semântica e a sintaxe de funções de comunicação definidos pelo Fórum MPI, com o propósito de atender às necessidades de aplicações paralelas típicas [9]. Em MPI todo o paralelismo é explícito, sendo o desenvolvedor responsável em especificar como e quando o paralelismo será utilizado.

Normalmente duas medidas de desempenho são utilizadas para avaliar um sistema paralelo: *speedup* e eficiência. *Speedup* pode ser definido como a relação entre o tempo gasto na execução de uma determinada tarefa utilizando um único processador e o tempo gasto com N processadores, indicando assim quantas vezes o programa paralelo é mais rápido que a versão sequencial [10]. A eficiência demonstra se os recursos foram bem aproveitados. A eficiência é o produto da divisão do *speedup* pelo número de processadores utilizados no processamento.

3. Algoritmo de geração de estimativas iniciais

Uma equação não linear envolve um conjunto de incógnitas (variáveis) e de operações sobre estas incógnitas. Encontrar as raízes desta equação é descobrir um conjunto de valores (conjunto solução) que aplicados à equação satisfaça a igualdade desta equação a um determinado valor. Uma mesma equação pode ter vários conjuntos solução. Quando dispomos de várias equações sujeitas aos mesmos conjuntos solução temos um sistema de equações não lineares. Uma forma de encontrar um conjunto solução para o sistema é partir de um conjunto inicial estimado (estimativas iniciais) e usar um processo iterativo em que, a cada iteração, o conjunto estimado é aplicado as equações do sistema e, de acordo com o método utilizado, transformado em um conjunto mais próximo de um conjunto solução. Após a aplicação de uma sequência de iterações o conjunto estimado poderá convergir ou não para um conjunto solução. De fato, a eficiência do método iterativo depende das estimativas iniciais.

Em 2001, Michael W. Smiley e Changbum Chun [2], da Universidade do Estado de Iowa nos Estados Unidos, apresentaram um algoritmo, denominado SubDivN1, que através de subdivisões sucessivas pode localizar bons conjuntos de estimativas iniciais

para sistemas de equações algébricas não lineares, a partir de um determinado conjunto de intervalo de variáveis. Um conjunto de intervalos de variáveis contém um intervalo de solução para cada incógnita onde se espera encontrar uma ou mais solução. O método apresentado subdivide sucessivamente os intervalos fazendo-os se aproximar das soluções.

No presente trabalho, usamos o algoritmo SubDivN1 para gerar as estimativas iniciais de um sistema de equações não lineares aplicado na simulação da destilação reativa usada na produção de biodiesel. O método convencional de produção do biodiesel é realizado através da transesterificação de óleos vegetais através da catálise homogêneo em processo batelada descontínuo realizado em duas etapas distintas, a reação química e a separação dos produtos [1]. A destilação reativa é uma operação em que ocorre simultaneamente uma reação química e a separação dos produtos no mesmo equipamento. O uso de coluna de destilação reativa permite reduzir em aproximadamente 80% os custos do processo através da eliminação de unidades, de 28 equipamentos e 11 etapas diferentes para apenas uma coluna de destilação reativa [1], justificando assim a sua simulação e estudos que auxiliem o seu entendimento.

O modelo matemático utilizado para simular computacionalmente a coluna de destilação reativa é formado por um sistema de equações não lineares que, para a obtenção de resultados, depende de um bom conjunto de estimativas iniciais, permitindo assim a convergência do método de resolução. Usamos o algoritmo SubDivN1 com este propósito, mas o mesmo pode levar dias de execução quando sujeito a um sistema de muitas variáveis. Por esse motivo realizamos a sua paralelização.

4. Trabalhos relacionados

Muitas pesquisas têm sido feitas sobre o uso de processamento paralelo para produção de resultados de forma rápida, eficiente e econômica em diferentes áreas científicas. Na pesquisa realizada por Mullenix e Povitsky [11], desenvolvida no Ohio Supercomputer Center's, foi utilizado um *cluster* para simular o processo de ablação. A ablação é um processo de rápida remoção de material de uma superfície sólida através de reações químicas, sublimação e de outros processos erosivos. Através da paralelização da aplicação foi possível reduzir o tempo computacional necessário para a execução do programa sequencial, que necessitava de 59.95 horas, para apenas 3.2 horas quando executado em paralelo utilizando 32 processadores.

O trabalho produzido por Kawabata, Venturini e Coda [12] realizado em São Carlos, tem o objetivo de criar um programa computacional paralelo para ser utilizado como plataforma para futuras pesquisas em métodos numéricos e mecânica estrutural, permitindo que o processamento paralelo seja realizado tanto em *clusters* quanto em computadores de múltiplos núcleos. Nos experimentos realizados em um *cluster*, primeiramente com seis nós e posteriormente doze nós, foi utilizado um toróide com 201.600 graus de liberdade como objeto de experimentação. O processamento foi dividido em partes, onde a parte que obteve melhor desempenho foi a montagem da matriz, alcançando *speedup* de 5,5 e 11, respectivamente.

O trabalho realizado por Gomes [13] discute questões relacionadas à paralelização de algoritmos sequenciais utilizados na solução de problemas científicos, mais especificamente, a paralelização de um algoritmo da área de Engenharia

Química que simula o processo de adsorção de moléculas em superfícies heterogêneas bidimensionais e que possa ser executado em um *cluster* de computadores. Esse algoritmo utiliza o método de Monte Carlo para calcular o estado de energia do sistema após movimentos das moléculas e os resultados são utilizados para a obtenção de gráficos de isotérmicas, comparando-os com os experimentos reais e dados conhecidos. Segundo, a execução sequencial do referido algoritmo consome muitas horas de processamento. Nesse trabalho foram obtidos redução do tempo de processamento de até 83,17%.

5. Paralelização regular e balanceada

Os experimentos paralelos foram executados no *cluster* existente no Laboratório Experimental de Computação de Alto Desempenho (LECAD) do Departamento de Informática da Universidade Estadual de Maringá. O *cluster* do LECAD é composto por 10 computadores sendo: 1 servidor com processador Intel Core 2 Quad 2.4 Ghz, 4 núcleos, 2 Gb de memória RAM, 6 computadores com, cada um, processador AMD Opteron 1218 2.6 Ghz, 2 núcleos, 4 Gb de memória RAM, e 3 computadores com, cada um, 2 processadores Intel Xeon E5620 2.4 Ghz, 4 núcleos *Hyper-Threading*[1], 8 Gb de memória RAM. Todos os computadores possuem interface de rede gigabit conectados por uma switch gigabit. A distribuição Linux utilizada é a Rocks Cluster com Kernel 2.6.18. A implementação da plataforma MPI utilizada é a OpenMPI 1.3.2.

O algoritmo de subdivisão [2] funciona conforme pode ser observado no Algoritmo 1. Pode-se observar que o algoritmo ao inicializar faz a leitura do arquivo de configuração e inicializa as variáveis. A variável $iCov$ controla a quantidade de vezes que será aplicado o teste de cobertura e a variável $maxSub$ controla a quantidade de subdivisões. A cada subdivisão, o contador de retângulos retidos na subdivisão atual (m) é inicializado.

Cada retângulo pode gerar até 2^d sub-retângulos, desta forma, na linha 7 é feito um laço de repetição com base no produto da quantidade de retângulo retidos pelas subdivisões até o momento pela quantidade de sub-retângulos que cada estimativa pode gerar. Entre as linhas 8 e 18 são obtidas as coordenadas de cada novo sub-retângulo e é aplicado a função do sistema de equações não linear no ponto central de cada sub-retângulo. Esse sub-retângulo então é avaliado e se for aprovado pelo critério de seleção será retido e incrementado o contador de retângulos retidos na atual subdivisão, caso contrário, o sub-retângulo é descartado.

Após avaliar todas as possibilidades (linha 19) o contador da quantidade total de retângulos retidos é atualizado. Após o máximo de subdivisões ser atingido, é aplicado um teste de cobertura, caso tenha sido definido, para verificar a existência de de retângulos adjacentes. Após ter sido estimado todos os sub-retângulos possíveis do retângulo inicial, é aplicado o método de Newton nos retângulos retidos para verificar a existência de raízes (convergência). Caso haja convergência de algum retângulo, será exibido na tela.

Uma estrutura em árvore é gerada para manter os sub-retângulos gerados, em que cada nó da árvore representa um único conjunto de sub-retângulos possíveis. Para encontrar os retângulos a serem retidos essa árvore era originalmente percorrida segundo a

[1] *Hyper-Threading*: permite que cada núcleo execute dois processos ao mesmo tempo. [6]

técnica "Primeiro em Largura" (*Breadth First*[2]). Usando este caminhamento, o algoritmo armazena todos os retângulos observados para no final submetê-los aos critérios de seleção e convergência. Com isso, a utilização de memória aumenta rapidamente, devido ao crescimento acelerado da árvore com o avanço das iterações, promovendo a ocorrência intensiva de *swap*[3]. Para grandes sistemas, em vários testes realizados, quase sempre a execução foi abortada devido ao estouro da capacidade de armazenamento. O tempo de processamento também cresce demasiadamente com essa abordagem.

Algoritmo 1 Algoritmo da aplicação

```
1: Lê o arquivo de configuração
2: Atribui valores às variáveis
3: Inicializa M
4: Para l de 1 até iCov faça
5:     Para i de 1 até o maxSub faça
6:         Inicializa m
7:         Para j de 1 até M * 2^d faça
8:             Obtém coordenadas
9:             Obtém número aleatórios
10:            Aplica f(x)
11:            Avalia retângulo
12:            Se retângulo aprovado então
13:                Retêm retângulo
14:                Incrementa m
15:            Senão
16:                Descarta retângulo
17:            Fim se
18:        Fim para
19:        M = m
20:    Fim para
21:    Avalia retângulos adjacentes
22:    Redefine M
23: Fim para
24: Aplica o método de Newton
25: Verifica soluções encontradas
26: Exibe soluções
```

Para otimizar o uso de recursos, optou-se por utilizar o caminhamento "Primeiro em Profundidade" (*Depth First*[4]) para não precisar armazenar todos os retângulos observados. Com essa nova abordagem, cada nó da árvore visitado já é submetido ao critério de seleção e caso seja aprovado, o algoritmo continua o caminhamento descendo pelos nós filhos, caso contrário, ele é descartado. Somente após atingir o número

[2]*Breadth First*: visita todos os sucessores de um nó visitado antes de visitar quaisquer sucessores de qualquer um desses sucessores [14].

[3]*Swap*: consiste em trazer totalmente cada processo para a memória, executá-la durante um determinado tempo e então devolvê-lo ao disco [15]

[4]*Depth First*: visita todos os sucessores de um nó visitado antes de visitar qualquer um de seus "irmãos" [14].

finito de subdivisões pré-determinado o método de convergência pode ser aplicado. A paralelização foi realizada a partir deste modelo otimizado.

A paralelização do algoritmo foi baseada no modelo mestre-escravo, onde o nó mestre efetua a leitura do arquivo de configuração e distribui para os nós escravos as variáveis de configuração bem como o retângulo inicial, e posteriormente recebe as soluções encontradas pelos nós escravos. Sobretudo, é válido salientar que o nó mestre também é utilizado no processamento dos sub-retângulos.

Como dito anteriormente, cada retângulo a ser testado pode gerar até 2^d sub-retângulos, sendo d a dimensão (número de incógnitas). Entretanto, cada nó executa apenas os sub-retângulos que lhe compete gerar e processar, segundo uma política de distribuição que denominamos de "grupos vizinhos". Nessa política, o total de sub-retângulos deve ser fracionado em grupos contendo, cada um, uma mesma quantidade de sub-retângulos vizinhos. Entretanto, dois modelos de algoritmos paralelos foram experimentados: regular e balanceado.

No modelo regular o retângulo inicial é propagado para todos os nós do *cluster* e cada nó então gera somente os $\frac{2^d}{n}$ sub-retângulos referentes ao grupo de vizinhos que lhe compete executar, onde n é o número de nós envolvidos no processamento.

O modelo balanceado funciona de maneira semelhante ao modelo regular, no que se refere a distribuição inicial de carga, entretanto, realiza um balanceamento dinâmico de carga segundo a estratégia "receptor inicia". Nesse caso, quando um nó termina sua parcela de processamento definida inicialmente, sinaliza que está disponível. Os demais nós, em determinado nível da árvore pré-definido através do arquivo de configuração, realizam uma checagem verificando se existe algum nó livre e, caso haja, envia uma estimativa pra que o nó disponível processe-a. Para a realização dos testes foi utilizado um sistema de equação não linear com 7 dimensões disponível em [16] juntamente com sua respectiva solução, sendo definido um número máximo de 3 iterações.

6. Resultados

Nos experimentos realizados os tempos coletados foram contabilizados do início ao fim da execução, não sendo discriminados tempos de comunicação, tempo de processamento e tempo ocioso. É importante salientar que o nó mestre além de distribuir o retângulo inicial para os demais nós também realiza as subdivisões, deste modo, ele é incluído no cálculo do *speedup*. Na Tabela 1 são listados os tempos de processamento de cada versão do algoritmo.

Nº de Nós	Algoritmo Sequencial	Modelo Regular	Modelo Balanceado
1	495	—	—
2	—	311	250
4	—	175	134
8	—	96	68
16	—	72	52

Tabela 1: Tempos de processamento (em minutos)

O *speedup* de cada modelo paralelo foi obtido dividindo o tempo gasto pelo programa na versão sequencial pelo tempo gasto na versão paralela. Os valores dos

speedups obtidos podem ser visualizados na Tabela 2, a qual mostra também o ganho percentual do modelo balanceado sobre o modelo regular. O gráfico é ilustrado na Figura 1(a), onde os valores intermediários foram interpolados e adicionado também o valor do *speedup* ideal teórico para ser utilizado como referência. Pode-se observar nesse gráfico que ambos os modelos proporcionaram um aumento de desempenho significativo sobre a execução sequencial. Os valores obtidos deixam claro que o desempenho de ambos os modelos não cresce proporcionalmente ao número de nós utilizados, os quais dobram a cada medição.

É evidente que o modelo balanceado foi melhor, apresentando ganhos sobre o modelo regular que atingem o pico de 42%, no caso de 8 nós de processamento. Esses ganhos se mostraram crescentes com a variação no número de nós de 2 a 8, mas teve uma queda quando se usou 16 nós. Ressalta-se entretanto que a plataforma de hardware utilizada não contém exatamente 16 cores, mas sim 8 cores dual (*hyper-thread*) o que pode ter justificado esta queda uma vez que os recursos de hardware são compartilhados entre os processos que executam em *hyperthreading*. Outro fator que pode ter influenciado nessa perda é o *overhead*[5] criado pelo processo de balanceamento de carga, que aumenta com o aumento no número de nós, ao mesmo tempo que a granularidade da carga de trabalho diminui para cada nó. A Tabela 3 exibe a eficiência obtida pelos modelos paralelos e seu respectivo gráfico pode ser observado na Figura 1(b).

Nº de Nós	Modelo Regular	Modelo Balanceado	Ganho $\frac{Balanceado}{Regular}$
2	1,59	1,98	24,5%
4	2,83	3,69	30,4%
8	5,16	7,28	41,1%
16	6,88	9,52	38,4%

Tabela 2: Speedups obtidos

Nº de Nós	Modelo Regular	Modelo Balanceado
2	0,8	0,99
4	0,71	0,92
8	0,64	0,91
16	0,43	0,59

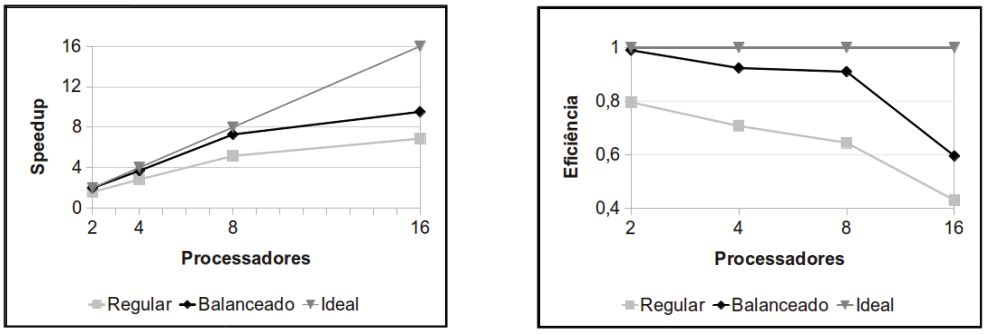
Tabela 3: Eficiência

7. Conclusões

Para a simulação do processo de coluna de destilação reativa, utilizado no processo de produção de ésteres de ácidos graxos, é necessário o conhecimento de estimativas iniciais. O algoritmo utilizado para a geração destas estimativas pode exigir um tempo de processamento muito elevado quando o sistema de equações não lineares possuir muitas dimensões, justificando assim sua paralelização.

Como parte da otimização do código, foram realizadas modificações no código original do algoritmo de subdivisão, entre elas, a principal modificação foi a forma

[5] *Overhead*: custo das operações existentes na versão paralela que não existe na versão sequencial.



(a) Speedup (b) Eficiência

Figura 1: Speedup e Eficiência obtidos

de caminhamento dentro a árvore, que originalmente utilizava a técnica "Primeiro em Largura" passou a utilizar a técnica "Primeiro em Profundidade". Essa modificação foi necessária devido o fato do algoritmo original promover a ocorrência intensiva de *swap* à medida que o número de dimensões do sistema de equações não lineares e a quantidade de subdivisões aplicadas ao retângulo inicial aumentam. Em diversos testes realizados com grandes sistemas, normalmente a execução era abortada devido ao estouro da capacidade de armazenamento.

Cada retângulo pode gerar até 2^d sub-retângulos, sendo d a dimensão. Para a divisão do retângulo inicial, foram utilizados dois modelos de algoritmos paralelos. No primeiro modelo cada nó do *cluster* gera em sua primeira subdivisão apenas os sub-retângulos adjacentes que lhe compete processar. O segundo modelo distribui os sub-retângulo de maneira idêntica ao primeiro modelo, entretanto possui balanceamento de carga sob a política "receptor inicia", em que o nó que termina sua parcela de processamento, sinaliza sua disponibilidade em receber mais retângulos para subdividir.

Os testes utilizaram um sistema de equação não linear com 7 dimensões, realizando um número máximo de 3 subdivisões. A execução dos experimentos foi realizada no *cluster* heterogêneo existente no Laboratório Experimental de Computação de Alto Desempenho (LECAD) do Departamento de Informática da Universidade Estadual de Maringá utilizando o modelo mestre-escravo.

A partir dos resultados obtidos pela execução dos dois modelos paralelos desenvolvidos neste trabalho, observou-se a redução no tempo total de processamento das versões paralelas em relação à versão sequencial. Podemos observar que o modelo balanceado utilizou os recursos computacionais de maneira mais eficiente que o modelo regular, obtendo ganhos crescentes com a utilização de 2, 4 e 8 nós. Sobretudo, com a utilização de 16 nós, houve uma redução no ganho e, dentre os fatores que possam produzir essa característica, podemos destacar o compartilhamento de recursos de hardware proporcionado pela arquitetura *hyper-thread* e o *overhead* produzido pelo mecanismo de balanceamento de carga, ambos fatores sendo mais expressivos quando se usa mais que 8 nós de processamento.

Apesar do trabalho ainda estar em fase de desenvolvimento, os experimentos foram válidos para identificar os pontos frágeis das abordagens utilizadas até o momento, permitindo assim que novos modelos possam ser produzidos.

Referências

[1] Machado, G. D., *Produção de biodiesel por esterificação em coluna de destilação reativa: modelagem matemática*, Master's thesis, Universidade Estadual de Maringá, Maringá - PR, 2009.

[2] Corazza, F. C., Oliveira, J. V., and Corazza, M. L., "Application of a subdivision algorithm for solving nonlinear algebraic systems," *Acta Scientiarum*, Vol. 30, No. 1, 2008.

[3] Yero, E. J. H., *Estudo sobre Processamento Maciçamente Paralelo na Internet*, Ph.D. thesis, Universidade Estadual de Campinas, Campinas, SP, 2003.

[4] Foster, I., *Designing and Building Parallel Programs*, Addison-Wesley, 1995, Disponivel em: http://www.mcs.anl.gov/ itf/dbpp/.

[5] Grama, A., Gupta, A., Karypis, G., and Kumar, V., *Introduction to Parallel Computing*, Addison-Wesley, 2003.

[6] Tanenbaum, A. S., *Organização Estruturada de Computadores*, Prentice Hall, 5th ed., 2007.

[7] Sterling, T., *Beouwlf Cluster Computing with Linux*, MIT Press, 2001.

[8] TOP500, "TOP 500 Supercomputer Site," 2011, Acesso em: 28/07/2011.

[9] Barreto, M. E., Ávila, R. B., and Oliveira, F. A. D., "Execução de aplicações em ambientes concorrentes," *Anais da 1a. Escola Regional de Alto Desempenho*, SBC, Gramado, RS, 2001.

[10] Cortes, O. A. C. and Mendez, O. R. S., "Determinação do Ranking de Contingências em Sistemas de Energia Elétrica Utilizando MPI (Message Passing Interface)," *INFOCOMP Journal of Computer Science*, 1999.

[11] Mullenix, N. and Povitsky, A., "Parallel implementation of a tightly coupled ablation prediction code using MPI," *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, aug. 2009, pp. 1 –4.

[12] Kawabata, C. L. O., Venturini, W. S., and Coda, H. B., "Cadernos de Engenharia de Estruturas," *Desenvolvimento e implementação de um método de elementos finitos paralelo para análise não linear de estruturas*, Vol. 11, No. 53, 2009, pp. 151 –155.

[13] Gomes, J. L., *Paralelização de algoritmo de simulação de Monte Carlo para a adsorção em superfícies heterogêneas bidimensionais*, Master's thesis, Universidade Estadual de Maringá, Maringá, PR, 2009.

[14] Tenenbaum, A. A., Langsam, Y., and Augenstein, M. J., *Estruturas de Dados Usando C*, Makron Books, 1995.

[15] Tanenbaum, A. S., *Sistemas Operacionais Modernos*, Prentice Hall, 2nd ed., 2003.

[16] Polymath-Software, "Numerical Library Problems Involving Simultaneous Nonlinear Equations," 2011, Disponível em: http://www.polymath-software.com/library/problemlist.shtml Acesso em: 20/01/2011.