Modelos paralelos para o algoritmo de subdivisões sucessivas

Robertino Mendes Santiago Jr
Sistemas Para Internet
Faculdade Alfa de Umuarama
Umuarama, Paraná, Brasil
robertinosantiago@gmail.com

Anderson Faustino da Silva
Departamento de Informática
Universidade Estadual de Maringá
Maringá, Paraná, Brasil
anderson@din.uem.br

Ronaldo Augusto Lara Gonçalves
Departamento de Informática
Universidade Estadual de Maringá
Maringá, Paraná, Brasil
ralgonca@din.uem.br

Resumo - Este trabalho apresenta dois modelos paralelos para um algoritmo iterativo que gera estimativas iniciais para sistemas de equações não lineares, no qual um sistema bem conhecido é o sistema é utilizado na simulação de colunas de destilação reativa. Os modelos utilizam balanceamento de carga estático ou dinâmico, além de escalonamento adjacente ou equidistante. Os experimentos foram realizados em um cluster de computadores com suporte à plataforma MPI. Os modelos obtiveram bons resultados para pequenos e grandes problemas. O modelo dinâmico proporcionou redução no tempo de execução em relação ao modelo estático, alcançando 35,42% para o problema com 5 dimensões e 34,15% para 7 dimensões. O escalonamento equidistante permitiu uma redução de tempo de execução em relação ao escalonamento adjacente, alcançando 40,98% para o modelo estático e 33,89 para o modelo dinâmico.

Palavras-chave: Paralelização de Aplicações Científicas; Modelos Paralelos; Metodologia de Paralelização.

I. INTRODUÇÃO

Diversas áreas científicas são beneficiadas com a utilização de *clusters* de computadores, os quais são usados para simular de forma paralela experimentos que, em muitos casos, demandam muito esforço de processamento, permitindo estimar resultados e comportamentos de forma mais rápida.

A solução iterativa de sistemas de equações não lineares é um dos problemas conhecidos que requer poder computacional elevado. A Engenharia Química é uma das áreas de conhecimento que possui muitos sistemas iterativos de equações não lineares, como exemplo a simulação da produção de ésteres de ácidos graxos (biodiesel) em colunas de destilação reativa, que utiliza um modelo matemático baseado em um sistema iterativo de equações não lineares [1].

A convergência da solução desse sistema depende das estimativas iniciais para as suas incógnitas, o que demanda muito esforço e tempo para serem encontradas. Entretanto, essas estimativas iniciais podem ser geradas por um algoritmo de subdivisões sucessivas [2].

De acordo com o número de incógnitas (dimensões) envolvido no sistema, a geração de estimativas iniciais pelo algoritmo de subdivisões sucessivas pode ser um processo muito demorado devido ao fato de utilizar o próprio sistema iterativo durante sua execução. Além disso, o algoritmo de subdivisões sucessivas gera uma estrutura de dados em forma de árvore *n*-ária, em que *n* equivale ao número de dimensões do problema envolvido.

Esta árvore possui um custo de processamento muito alto para ser armazenada e analisada, principalmente para a aplicação de coluna de destilação reativa aqui mencionada, a qual pode envolver centenas de incógnitas. Neste contexto, o presente trabalho propõe a paralelização deste algoritmo e analisa os resultados sobre dois modelos distintos.

A Seção II apresenta os trabalhos relacionados a esta pesquisa. O algoritmo de subdivisões sucessivas é descrito na Seção III. Os modelos paralelos do algoritmo, bem como as respectivas implementações, são relatados na Seção IV. Os experimentos e resultados são relatados na Seção V. As conclusões são apresentadas na Seção VI.

II. 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 aplicações científicas. Mullenix e Povitsky [3] utilizaram um *cluster* para simular o processo de ablação[1]. A paralelização da aplicação permitiu reduzir o tempo da simulação de 59.95 horas para apenas 3.2 horas quando executado em 32 processadores.

O processo de adsorção de moléculas em superfícies heterogêneas bidimensionais foi simulado por Gomes [4], usando um *cluster* de computadores. A simulação é baseada no método de Monte Carlo para calcular o estado de energia do sistema. Experimentos mostraram uma redução do tempo de processamento em até 83.17%.

Em outro trabalho [5], Costa implementou o método do gradiente conjugado pré-condicionado utilizando quatro pré-condicionadores distintos sobre um *cluster* com 12 processadores. Sobre uma malha com 49.951 elementos o desempenho atingiu *speedup* superlinear.

A estratégia de paralelização desenvolvida por Pinto [6] foi aplicada ao problema de planejamento da operação de sistemas hidrotérmicos. Um *cluster* de computadores foi utilizado para avaliar o desempenho da estratégia e o tempo de processamento passou de 15 horas para 17 minutos utilizando

[1] Processo de rápida remoção de material de uma superfície sólida por meio de reações químicas, sublimação e de outros processos erosivos.

128 processadores, atingindo desta forma a eficiência de 41.10%.

Oliveira [7] usou estratégias para a paralelização de algoritmos do tipo *branch-and-prune* e *branch-and-bound* em ambientes distribuídos compartilhados e dinâmicos. Os resultados mostraram que, dependendo dos valores de níveis de corte realizado na árvore é possível obter uma boa paralelização, obtendo valores próximos ao linear.

Modenesi [8] usou o paralelismo para tratar o problema da análise de agrupamentos em conjuntos e partições na mineração e processamento de grandes volumes de dados. Em experimentos realizados em um *cluster* de computadores, os melhores *speedups* alcançaram valores próximos ao linear.

III. ALGORITMOS DE SUBDIVISÕES SUCESSIVAS

Em 2001, Michael W. Smiley e Changbum Chun [2] apresentaram um algoritmo de subdivisões sucessivas capaz de localizar simultaneamente todas as raízes de sistemas de equações algébricas não lineares, a partir de intervalos de variáveis iniciais. O algoritmo consiste em subdividir cada intervalo de variável em subintervalos de mesmo tamanho, os quais são submetidos a um teste de avaliação para verificar se os mesmos continuam sendo intervalos possíveis, isto é, intervalos que podem conter a solução para a respectiva variável.

O subintervalo submetido ao teste de avaliação é mantido para novas subdivisões somente se o teste retornar um resultado positivo, caso contrário, é descartado. Após a execução de um número finito de subdivisões, os subintervalos são tão pequenos que se aproximam das raízes das respectivas variáveis. Portanto, é um bom gerador de estimativas iniciais para sistemas de equações não lineares.

Este algoritmo utiliza o método de Newton-Raphson e pode ser usado na solução de sistemas fortemente não lineares, como no caso da coluna de destilação reativa [1] que depende de um grande número de estimativas iniciais.

A Figura 1 mostra o algoritmo de subdivisões sucessivas original, o qual é sequencial. A variável *iCov* controla a quantidade de vezes que o teste de cobertura é aplicado. No teste de cobertura, os retângulos[2] que possuem as mesmas coordenadas (adjacentes) em pelo menos uma dimensão são usados para definir um único novo retângulo após a aplicação do teste. A variável *maxSub* controla a quantidade de subdivisões.

A cada subdivisão, o contador de retângulos mantidos na subdivisão atual (*m*) é inicializado. Cada retângulo pode gerar até 2^d sub-retângulos, sendo *d* a dimensão. Desta forma, na linha 5 é feito um laço de repetição com base no produto da quantidade de retângulos mantidos nas subdivisões pela quantidade de sub-retângulos que cada retângulo pode gerar.

Entre as linhas 6 e 16 são obtidas as coordenadas de cada novo sub-retângulo e é aplicado a função do sistema de equações não lineares no ponto central de cada sub-retângulo. Cada sub-retângulo então é avaliado com o auxílio de números aleatórios, que realizam ajustes nos valores do ponto central do sub-retângulo para prever a próxima iteração. O sub-retângulo aprovado pelo critério de seleção é mantido e o contador de retângulos mantidos é incrementado, caso contrário, o sub-retângulo é descartado.

Uma estrutura em árvore, denominada aqui de "Árvore de Estimativas" é gerada para manter os sub-retângulos gerados. O nó raiz da árvore representa o retângulo inicial e cada nó filho representa um único sub-retângulo possível. Para encontrar os retângulos que serão mantidos, a árvore de estimativas é criada nó a nó segundo a técnica "Primeiro em Largura" (*Breadth First*).

Fig. 1. Algoritmo de subdivisões sequencial

```
     Entrada: Arquivo de configuração
     Saída:   Estimativas   aprovadas   no
     teste de convergência
 1.  Inicializa M;
 2.  para l ← 1 até iCov faça
 3.    para i ← 1 até maxSub faça
 4.      Inicializa m;
 5.      para j ← 1 até M * 2^d faça
 6.        Obtêm coordenadas;
 7.        Obtém números aleatórios;
 8.        Aplica f(x);
 9.        Avalia retângulo;
10.        se retângulo aprovado então
11.          Retêm retângulo;
12.          Incrementa m;
13.        senão
14.          Descarta retângulo;
15.        fim
16.      fim
17.      M ← m;
18.    fim
19.    Avalia retângulos adjacentes;
20.    Redefine M;
21.  fim
22.  Aplica o método de Newton;
23.  Verifica soluções encontradas;
24.  Exibe soluções;
```

IV. MODELOS PARALELOS DO ALGORITMO

Devido à forma de criação da árvore, todos os retângulos observados são armazenados, pois eles são usados na descida da árvore até que todos os retângulos da última iteração (folhas) sejam alcançados, os quais são submetidos aos critérios de seleção e convergência. Com isso, a utilização de memória aumenta rapidamente com o avanço das iterações, promovendo a ocorrência intensiva de swap. Para grandes sistemas, em vários experimentos realizados, a execução foi abortada devido ao estouro da capacidade de armazenamento. O tempo de processamento também cresce demasiadamente com essa abordagem original.

Para otimizar o uso da memória, a política de percurso foi alterada para "Primeiro em Profundidade" (*Depth First*), a qual não obriga o armazenamento de todos os retângulos observados. Com essa nova abordagem, somente os retângulos

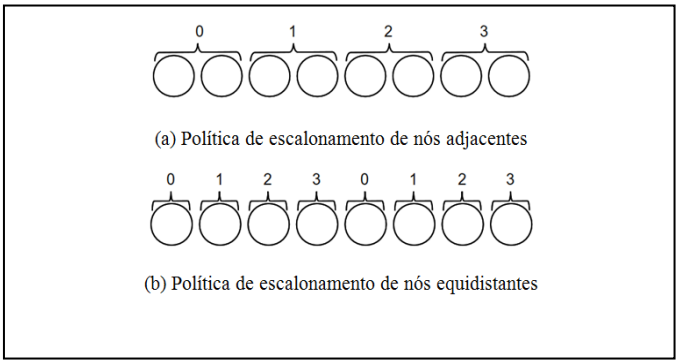
[2] Um retângulo pode ser definido como sendo uma estrutura de dados que possui um conjunto de intervalos (valor inicial e final) para cada dimensão do problema, formando assim uma estimativa.

observados durante a descida rumo a um retângulo folha são armazenados. Além disso, sempre que um retângulo folha é encontrado, o mesmo é imediatamente submetido ao critério de seleção e caso seja aprovado, é submetido a um teste de convergência final, senão, é descartado. Durante o retorno do caminho descido, os retângulos intermediários que não serão mais descidos em seus filhos, são descartados. Dessa forma, somente os nós aprovados no teste de convergência final permanecem armazenados. Após esta otimização de código, o algoritmo de subdivisões sucessivas foi paralelizado.

No presente trabalho, dois modelos paralelos para o algoritmo de subdivisões sucessivas foram desenvolvidos, o BEC e o BDCOC, conforme explicados nas Seções IV(A) e IV(B) respectivamente. Em ambos os modelos, a árvore de subdivisões sucessivas é distribuída entre os processadores do cluster segundo uma política de escalonamento, que pode ser "escalonamento de nós adjacentes" ou "escalonamento de nós equidistantes", conforme mostra a Figura 2. Nesta figura, os índices 0, 1, 2 e 3 indicam o número do processador e os círculos os nós subsequentes da árvore.

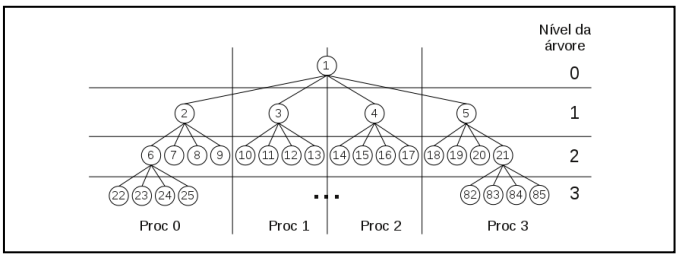
Nas duas políticas de escalonamento, todos os processadores possuem o retângulo inicial, que representa a raiz da árvore. No escalonamento de nós adjacentes (Figura 2(a)), cada processador gera $(2^d)/n$ sub-retângulos adjacentes (subsequentes), onde d é a dimensão e n é o número de processadores envolvidos na computação. No escalonamento de nós equidistantes (Figura 2(b)), cada processador gera $(2^d)/n$ sub-retângulos equidistantes a um intervalo pré-definido de n sub-retângulos.

Fig. 2. Políticas de escalonamento



Em ambos os modelos, todos os retângulos são etiquetados com um valor inteiro, que os identifica individualmente dentro da estrutura, independente de como os sub-retângulos são distribuídos, conforme pode ser observado na Figura 3.

Fig. 3. Árvore de estimativas com elementos enumerados globalmente



Esses valores visam garantir as mesmas condições para todas as execuções, sendo utilizados como semente na geração de números aleatórios a cada sub-retângulo gerado. O exemplo da Figura 3 ilustra uma árvore de estimativas gerada a partir de um retângulo inicial (nó raiz da árvore) com 2 dimensões em 3 níveis de subdivisões. Observe que a distribuição entre os processadores é feita a partir da primeira subdivisão.

A. *Modelo Paralelo com Balanceamento Estático de Carga (BEC)*

Neste modelo, cada processador do *cluster*, gera na primeira subdivisão do retângulo inicial os sub-retângulos de sua competência, obedecendo à política de escalonamento. A partir da segunda subdivisão, cada sub-retângulo pode também gerar até 2^d novos sub-retângulos e assim sucessivamente. Entretanto, um sub-retângulo somente irá gerar novos sub-retângulos caso seja aprovado no critério de seleção que verifica se este sub-retângulo tem potencial para ser uma boa estimativa inicial.

Não há troca explícita de mensagem entre os processadores do *cluster*, apenas sincronização e sinalização. Após avaliar todos os sub-retângulos possíveis, cada processador permanece aguardando em uma barreira de sincronização até que todos os demais finalizem sua parcela de processamento. Quando todos os processadores alcançarem a barreira, cada um envia ao processador mestre os sub-retângulos convergidos, caso possuam. A quantidade de sub-retângulos testados também é enviada a fim de verificar a acurácia do modelo.

O processador mestre agrupa os dados recebidos e finaliza a aplicação. O funcionamento deste modelo pode ser observado na Figura 4, que trabalha conforme já explicado.

É importante destacar que o uso do percurso "Primeiro em Profundidade" permitiu usar recursividade para gerar os sub-retângulos, o que é feito pela função recursiva *SubdivideRecursivo()*, mostrada na Figura 5, a qual realiza as subdivisões de acordo com o escalonamento.

Fig. 4. Algoritmo principal do modelo BEC

```
1.  Inicializa ambiente MPI;
2.  se myId == 0 então
3.     Lê arquivo de configuração e
       aloca estruturas ;
4.  fim
5.  Distribui o retângulo inicial e
    demais valores;
6.  SubdivideRecursivo(maximoSubdivisao,
    retângulo inicial, etiqueta global,
    etiqueta local);
7.  Barreira;
8.  se myId == 0 então
9.     Recebe valores dos nós escravos;
10. Senão
11.    Envia valores para o nó mestre;
12. fim
13. se myId == 0 então
14.    Grava dados no arquivo de saída;
15.    Imprime dados na tela;
16. fim
17. Finaliza ambiente MPI;
```

Na Figura 5, ao realizar a primeira subdivisão, que é controlada pela variável *subDivisaoAtual*, os sub-retângulos são gerados de acordo com a política de escalonamento utilizada. A partir da segunda subdivisão, a quantidade de sub-retângulos que serão testados será 2^d. Na linha 6 inicia o laço de repetição que controla o percurso na árvore.

Após um número finito de subdivisões dos sub-retângulos que foram mantidos, a possível solução pode estar presente apenas em um conjunto restrito de sub-retângulos da árvore, tornando-a assim desbalanceada, o que implica no surgimento de processadores ociosos de forma muito mais rápida, sugerindo desta forma a utilização de balanceamento de carga dinâmico.

Fig. 5. Função SubdivideRecursivo do modelo BEC

```
 1.  se subDivisaoAtual==maximoSubDivisao
     então
 2.      tamanhoTrabalho ← quantidade de
         sub-retângulos de acordo com a
         política de escalonamento;
 3.  senão
 4.      tamanhoTrabalho ← 2ᵈ;
 5.  fim
 6.  para l ← 0 até tamanhoTrabalho faça
 7.      Calcula a etiqueta do retângulo
         de forma global e local;
 8.      Obtém coordenadas;
 9.      Obtém números aleatórios;
10.      Aplica f(x);
11.      Avalia retângulo;
12.      se retângulo aprovado então
13.          se subDivisaoAtual == 1 então
14.              Aplica o método de Newton;
15.              Verifica solução encontrada;
16.          senão
17.              SubdivideRecursivo(
18.              subDivisaoAtual -1,
                 retângulo, etiqueta global,
                 etiqueta local);
                 fim
19.      fim
20.  fim
```

B. *Modelo Paralelo com Balanceamento Dinâmico de Carga por Ordem de Chegada (BDCOC)*

Este modelo realiza a mesma distribuição de carga que o modelo estático. Entretanto, em adição, ele também realiza um balanceamento dinâmico de carga em determinado nível da árvore, do tipo "receptor inicia", não sendo este nível alterado durante a execução da aplicação. A estratégia "receptor inicia" permite que os processadores ociosos procurem por processadores mais sobrecarregados.

Quando um processador termina sua parcela de processamento, ele sinaliza para os demais que está ocioso e fica aguardando carga de trabalho. O nível em que ocorre a verificação de balanceamento refere-se ao momento da subdivisão que a verificação é feita, sendo definido pelo valor do índice da subdivisão.

Quando um processador, que ainda possui retângulos para processar, passa pelo nível em que ocorre a verificação de balanceamento de carga, ele fica sabendo se existe um processador ocioso por meio de um sinal. Caso exista, o nó ciente seleciona um sub-retângulo ainda não processado e envia-o para o processador ocioso, removendo então o sinal que o bloqueava, colocando-o novamente para execução. O atendimento aos processadores ociosos é feito por ordem de chegada ao ponto de verificação.

O processador ocioso recebe o sub-retângulo e os demais dados necessários para o processamento. Ao finalizar o processamento do sub-retângulo, emite um sinal novamente para avisar aos demais processadores que está novamente ocioso e permanece aguardando o recebimento de mais sub-retângulos. Se dois ou mais processadores passarem pelo nível de balanceamento ao mesmo tempo e selecionarem o mesmo processador ocioso, é criada uma fila de sub-retângulos para o processador ocioso. Após ler e processar todos os elementos de sua fila, o processador volta novamente a sinalizar que está ocioso.

O processo de balanceamento encerra quando todos os processadores envolvidos no processamento sinalizarem que estão ociosos. A partir de então, cada processador envia ao processador mestre a quantidade de sub-retângulos processados e os convergidos. O processador mestre então agrupa os dados recebidos e encerra a aplicação. A sincronização por barreira também é usada na finalização do paralelismo. O algoritmo principal (Figura 6) desenvolvido para este modelo possui funcionamento semelhante ao modelo BEC, conforme já explicado.

Fig. 6. Algoritmo principal do modelo BDCOC

```
 1.  Inicializa ambiente MPI;
 2.  se myId == 0 então
 3.    Lê arquivo de configuração e
       aloca estruturas ;
 4.  fim
 5.  Distribui o retângulo inicial e
     demais valores;
 6.  SubdivideRecursivo(maximoSubdivisao,
     retângulo inicial, etiqueta global,
     etiqueta local);
 7.  Sinaliza disponibilidade do nó;
 8.  repita
 9.    Aguarda recebimento de retângulos;
10.    repita
11.      SubdivideRecursivo(
         subdivisaoRecebida -1,
         retângulo, etiqueta global,
         etiqueta local);
12.    até existe fila de retângulos;
13.    Sinaliza disponibilidade do nó;
14.  até quantidadeTrabalhadores !=
       quantidadeNosDisponiveis;
15.  Barreira;
16.  se myId == 0 então
17.    Recebe valores dos nós escravos;
18.  senão
19.    Envia valores para o nó mestre;
20.  fim
21.  se myId == 0 então
22.    Grava dados no arquivo de saída;
23.    Imprime dados na tela;
24.  fim
25.  Finaliza ambiente MPI;
```

Ressalta-se que no arquivo de configuração deste modelo foi inserido um parâmetro para identificar em qual nível da subdivisão ocorrerá o balanceamento de carga. Após a execução de sua parcela de processamento, o algoritmo sinaliza que o processador está disponível e fica aguardando o recebimento de novos retângulos a serem processados. O

processador ocioso então recebe novos retângulos por meio de uma fila própria e após processá-la, indica novamente sua disponibilidade. A função de subdivisão recursiva é mostrada na Figura 7.

Ao enviar o sub-retângulo atual, o algoritmo realiza um salto para o próximo sub-retângulo "irmão" por meio da instrução *continue* da linguagem C. Se o processador, ao passar pelo nível de balanceamento de carga, verificar que não existem processadores disponíveis, continua o processo de recursividade.

Fig. 7. Função SubdivideRecursivo do modelo BDCOC

```
1.  se subDivisaoAtual==maximoSubDivisao
    então
2.    tamanhoTrabalho ← quantidade de
      sub-retângulos de acordo com a
      política de escalonamento;
3.  senão
4.    tamanhoTrabalho ← 2^d;
5.  fim
6.  para l ← 0 até tamanhoTrabalho faça
7.    Calcula a etiqueta do retângulo
      de forma global e local;
8.    Obtém coordenadas;
9.    Obtém números aleatórios;
10.   Aplica f(x);
11.   Avalia retângulo;
12.   se retângulo aprovado então
13.     se subDivisaoAtual ==
        nivelBalanceamento então
14.       se existe nós disponíveis
          então
15.         noDestino ← nó disponível;
16.         envia retângulo ao
            noDestino;
17.         continua;
18.       fim
19.     fim
20.     se subDivisaoAtual == 1 então
21.       Aplica o método de Newton;
22.       Verifica solução encontrada;
23.     senão
24.       SubdivideRecursivo(
          subDivisaoAtual -1,
          retângulo, etiqueta global,
          etiqueta local);
25.     fim
26.   fim
27. fim
```

Cada vez que um processador atinge o nível de balanceamento de carga, como pode ser observado na Figura 7 (linha 13), é verificada a existência de processadores disponíveis. Caso haja, o primeiro processador a atingir este ponto irá selecionar um processador disponível e enviará o retângulo atual para este último, retirando-o da fila de processadores disponíveis.

Vale ressaltar que o processador que estava disponível, após receber um retângulo, não verifica a existência de processadores disponíveis devido ao fato de já ter ultrapassado o ponto de verificação. Desta forma, somente os processadores que possuem maiores quantidades de retângulos a processar disputam os processadores ociosos.

O nível em que ocorre a verificação de processadores ociosos influencia diretamente na quantidade de sub-retângulos a serem testados. Assim, o nível de balanceamento não deve ser muito próximo ao último nível da árvore (folhas) a fim de maximizar o processamento dos processadores ociosos.

V. EXPERIMENTOS E RESULTADOS

Os experimentos paralelos foram executados em um *cluster* composto por 3 computadores com 2 processadores Intel Xeon E5620 2.4 Ghz de 4 núcleos *Hyper-Threading*, e 8 Gb de memória RAM cada um. Desta forma, pode-se obter 24 núcleos físicos ou 48 *threads* com o uso do o *Hyper-Threading*. Os computadores possuem interface *gigabit* e estão conectados por um *switch gigabit*. A plataforma MPI utilizada é a OpenMPI 1.3.2.

Nos experimentos realizados, os tempos coletados foram contabilizados do início ao fim da execução da aplicação, não sendo discriminados tempos de comunicação, tempo de processamento e tempo de ociosidade. É importante salientar que o processador mestre, além de distribuir o retângulo inicial, também realiza o processamento (subdivisões), sendo incluído nos cálculos de *speedup* e eficiência.

Dados de casos reais conhecidos pertencentes a dois sistemas de equações não-lineares, o primeiro com 5 dimensões e o segundo com 7 dimensões, disponíveis em [9] juntamente com suas respectivas soluções, foram utilizados como entrada nos modelos implementados. Para o sistema com 5 dimensões foram aplicados 6 níveis de subdivisões (iterações) e para o sistema com 7 dimensões foram aplicadas 3 níveis de subdivisões. Todos os modelos paralelos aqui desenvolvidos empregam o paralelismo de dados e foram executados sobre 1, 2, 4, 8, 16 e 32 núcleos de processamento, denominados aqui de processadores. Entretanto, cabe ressaltar que foi possível realizar os experimentos com 32 núcleos de processamento devido ao recurso de *Hyper-Threading*.

A. *Tempos de execução*

Cada modelo paralelo foi experimentado com as duas políticas de escalonamento já discutidas. Os tempos de execução foram obtidos pela média aritmética simples sobre 3 execuções para cada conjunto de processadores envolvidos no processamento. Os gráficos dos tempos de execução podem ser observados nas Figuras 8 e 9 para o sistema com 5 dimensões e nas Figuras 10 e 11 para o sistema com 7 dimensões. Nestes gráficos, o eixo x indica o número de processadores e o eixo y o tempo em segundos para 5 dimensões e em minutos para 7 dimensões. Os tempos do algoritmo sequencial foram 798 segundos para o sistema com 5 dimensões e 496 minutos para o sistema com 7 dimensões.

De forma geral, observando as curvas representadas nos gráficos, nota-se que o tempo total de execução da aplicação diminui consideravelmente em função do aumento do número total de processadores envolvidos na computação paralela, satisfazendo assim um dos seus principais objetivos. Entretanto, essa redução não é linear, tendendo a se estabilizar conforme o número de processadores envolvidos aumenta. Analisando estes gráficos, observa-se que a redução no tempo

de execução foi significativamente maior para o sistema de 7 dimensões do que de 5 dimensões.

Fig. 8. Tempo sobre 5 dimensões com escalonamento de nós adjacentes

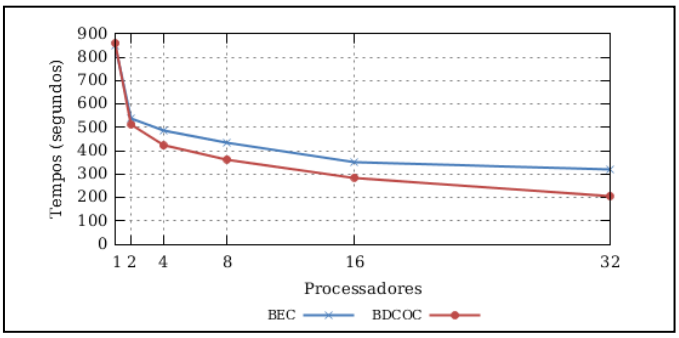


Fig. 9. Tempo sobre 5 dimensões com escalonamento de nós equidistantes

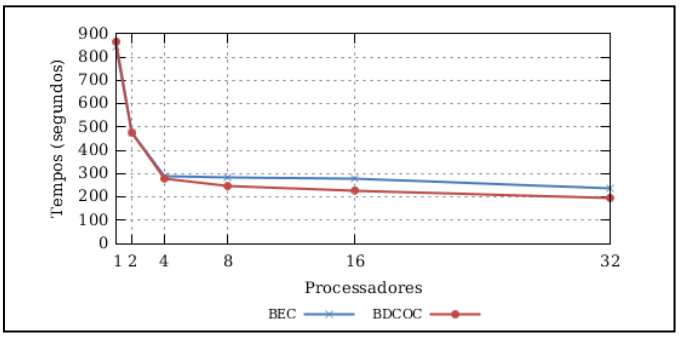
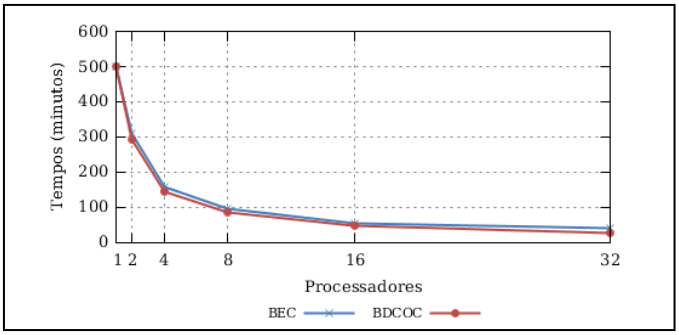


Fig. 10. Tempo sobre 7 dimensões com escalonamento de nós adjacentes

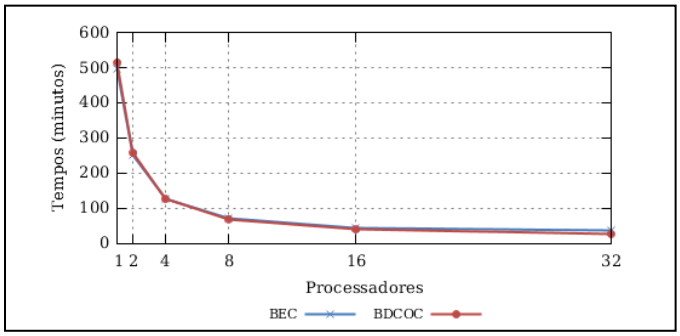


Para 5 dimensões, o menor tempo foi de 196 segundos, alcançado pelo modelo BDCOC com escalonamento de nós equidistantes, utilizando 32 processadores. Para 7 dimensões, o menor tempo foi de 27 minutos, alcançado pelo modelo BDCOC, independente de escalonamento, utilizando 32 processadores.

A análise das planilhas de resultados indicou que a partir de determinado número de processadores a tendência é que os algoritmos paralelos consumam mais tempo, principalmente devido ao custo da comunicação ou à ociosidade dos processadores causado pelo desbalanceamento da árvore de estimativas no caso do modelo que utiliza o balanceamento estático, sendo o grau deste consumo dependente do modelo implementado.

Apesar de não ser muito evidente, o ganho na redução de tempo do modelo dinâmico (BDCOC) sobre o modelo estático (BEC) foi bastante interessante, alcançando 35,42% no experimento com 5 dimensões, escalonamento adjacente e utilizando 32 processadores (206 segundos contra 319 segundos) e 34,15% para 7 dimensões, escalonamento adjacente e utilizando 32 processadores (27 minutos contra 41 minutos).

Fig. 11. Tempo sobre 7 dimensões com escalonamento de nós equidistantes

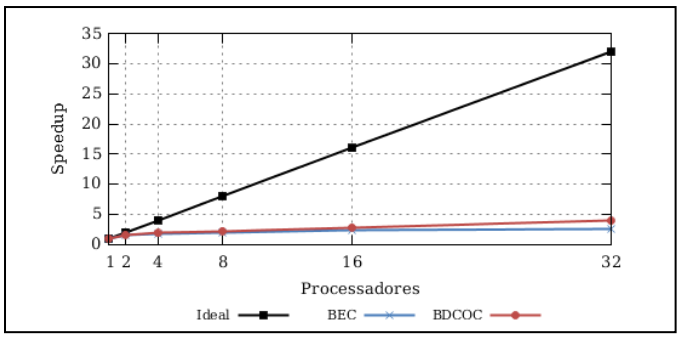


Outro resultado interessante foi o ganho do escalonamento equidistante sobre o adjacente, alcançado pico de 40,98% no modelo BEC para 5 dimensões utilizando 4 processadores (288 segundos contra 488 segundos) e 33,89% no modelo BDCOC para 5 dimensões utilizando 4 processadores (279 segundos contra 422 segundos).

B. *Speedup*

Os gráficos dos *speedups* obtidos podem ser observados nas Figuras 12 e 13 para 5 dimensões e Figuras 14 e 15 para 7 dimensões, onde os valores intermediários foram interpolados e uma curva contendo os valores do *speedup* ideal teórico foi adicionada para ser utilizada como referência. Nestes gráficos, o eixo x indica o número de processadores e o eixo y o *speedup* sobre o algoritmo sequencial.

Fig. 12. Speedup sobre 5 dimensões com escalonamento de nós adjacentes



Para o sistema com 5 dimensões observa-se que o *speedup* cresce mais lentamente do que para 7 dimensões, não sendo beneficiado de forma satisfatória pelo aumento no número de processadores. Esse fato se deve a descompensação causada pelo aumento nos tempos de controle, comunicação e balanceamento, em detrimento da redução no tempo de processamento, uma vez que o tempo de execução da

aplicação é pouco para tirar proveito do aumento no número de processadores.

Fig. 13. Speedup sobre 5 dimensões com escalonamento de nós equidistantes

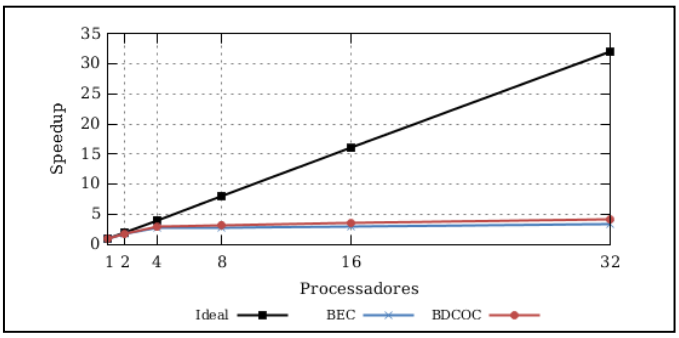


Fig. 14. Speedup sobre 7 dimensões com escalonamento de nós adjacentes

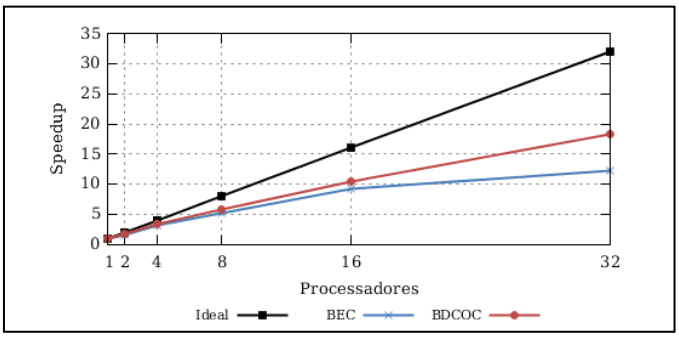
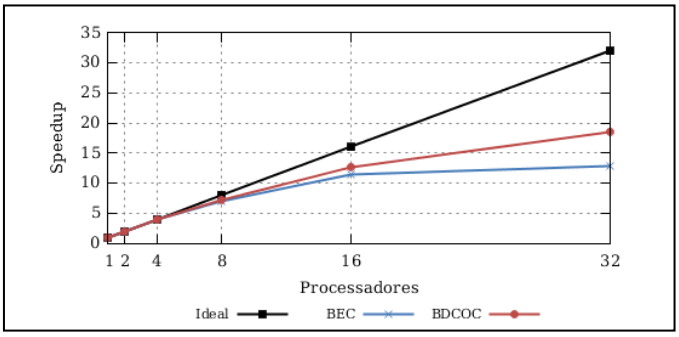


Fig. 15. Speedup sobre 7 dimensões com escalonamento de nós equidistantes



No sistema com 7 dimensões é possível observar que os modelos proporcionaram um ganho de desempenho significativo sobre a versão sequencial, chegando próximo ao linear com até 8 processadores, utilizando o escalonamento de nós adjacentes com 16 processadores e escalonamento de nós equidistantes.

Todos os gráficos apresentam uma queda no ganho de desempenho ao utilizar 32 processadores. Isso se deve ao fato do uso de *Hyper-Threading*, que não equivalem de fato a processadores independentes, mas sim núcleos compartilhados por duas *threads* simultâneas.

C. Eficiência

Os gráficos apresentados nas Figuras 16 e 17 exibem a eficiência obtida pelos modelos paralelos sobre o caso com 5 dimensões, enquanto as Figuras 18 e 19 exibem os gráficos referentes à eficiência obtida pelos modelos paralelos sobre o caso com 7 dimensões. Assim como nos gráficos de *speedup*, os valores intermediários foram interpolados e uma curva com os valores da eficiência ideal teórica foi adicionada para ser utilizada como referência.

Fig. 16. Eficiência sobre 5 dimensões com escalonamento de nós adjacentes

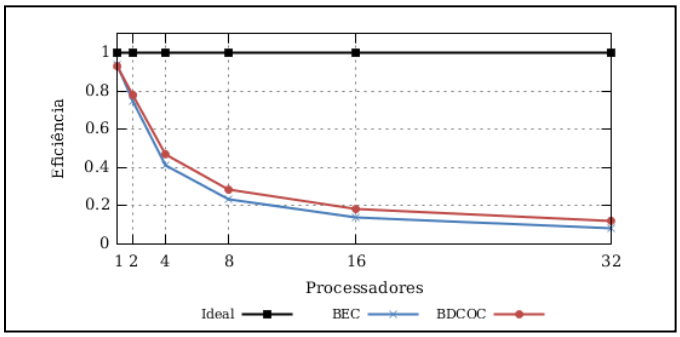


Fig. 17. Eficiência sobre 5 dimensões com escalonamento de nós equidistantes

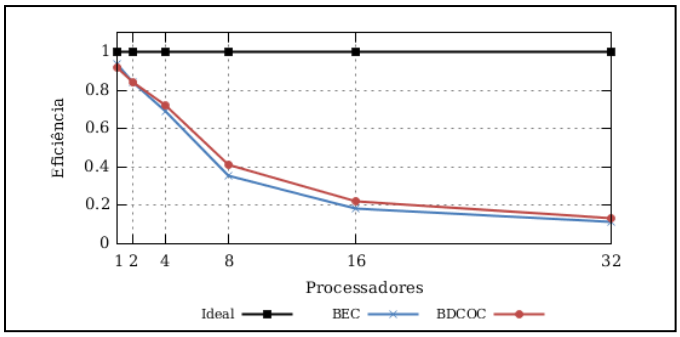
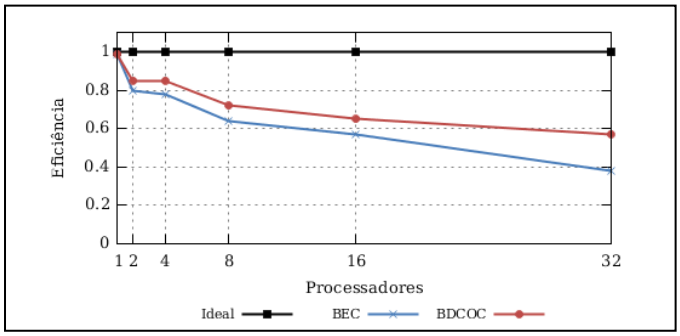


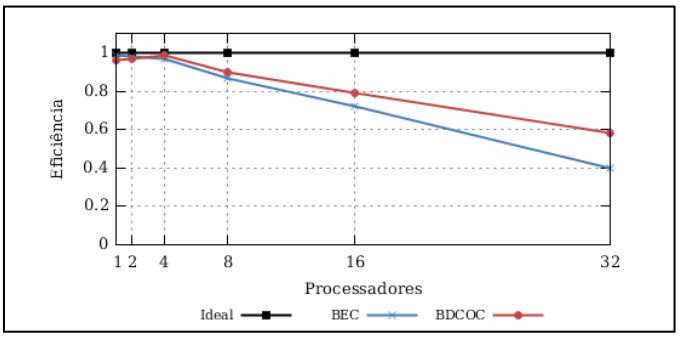
Fig. 18. Eficiência sobre 7 dimensões com escalonamento de nós adjacentes



Apesar dos modelos desenvolvidos apresentarem redução do tempo total de execução sobre o caso com 5 dimensões, é possível observar através das Figuras 16 e 17 que houve uma queda de eficiência dos modelos em ambas as políticas de escalonamento, sendo diretamente relacionada ao aumento do número de processadores envolvidos. Essa queda ocorre devido ao fato da granulosidade da aplicação ser fina, incorrendo em uma maior sobrecarga gerada pela política de balanceamento dinâmico de carga, no caso do modelo BDCOC ou pela ociosidade dos processadores, no caso do Modelo BEC.

Para o caso com 7 dimensões é possível avaliar que os recursos computacionais foram utilizados de forma mais satisfatória, tendo maior destaque o modelo BDCOC que obteve bons índices, na ordem de valores maiores ou iguais a 0,79 com até 16 processadores. Utilizando 32 processadores houve uma pequena queda de eficiência relacionada ao fato dos processos fazerem uso de *Hyper-Threading*.

Fig. 19. Eficiência sobre 7 dimensões com escalonamento de nós equidistantes



VI. CONCLUSÕES

Nos últimos anos, *clusters* têm se mostrado uma excelente alternativa aos supercomputadores multi-processadores devido ao baixo custo associado à sua arquitetura. Nesse sentido, a utilização do paradigma de programação baseado em passagem de mensagens em ambientes de programação paralela tem sido favorecida.

Neste trabalho, foram apresentados dois modelos de algoritmos paralelos para aplicações que são estruturadas em árvores. O problema alvo utilizado foi o algoritmo de subdivisões sucessivas [2] utilizado na geração de estimativas iniciais para sistemas de equações não-lineares que simulam o problema da coluna de destilação reativa, mas podem ser transportados para problemas similares.

Preliminarmente, para otimizar o uso de memória, o algoritmo sequencial original foi alterado de forma que o percurso da árvore de estimativas gerada pela aplicação, que originalmente era "Primeiro em Largura", foi alterado para "Primeiro em Profundidade". Essa modificação foi necessária devido ao fato do algoritmo original produzir estouro de memória à medida que o número de dimensões do sistema de equações não lineares e a quantidade de subdivisões aplicadas ao retângulo inicial aumentavam.

Outra otimização realizada, visando garantir o mesmo ambiente para todas as execuções, foi associar a geração da semente (*seed*) utilizada na geração de números aleatórios a cada sub-retângulo (estimativa) gerado.

Os modelos desenvolvidos obedeceram duas políticas de escalonamento distintas para a divisão do retângulo inicial, chamadas de "escalonamento de nós adjacentes" e "escalonamento de nós equidistantes", gerando assim duas versões para cada modelo. O primeiro modelo desenvolvido utilizou uma política de balanceamento de carga estático, enquanto que o segundo modelo utilizou política de balanceamento dinâmico.

Dados obtidos em dois casos reais conhecidos foram utilizados para a realização dos experimentos, sendo um com 5 dimensões e outro com 7 dimensões. Os resultados foram comparados com os valores reais, o que validou a corretude dos algoritmos.

A partir dos resultados obtidos, observou-se uma redução no tempo total de execução para todos os cenários paralelos em todos os modelos propostos. Sobretudo, apesar dessa redução de tempo na execução do algoritmo, sobre o experimento com 5 dimensões, os valores de *speedup* e eficiência demonstram que os modelos desenvolvidos não tiram proveito dos recursos computacionais de maneira satisfatória, sendo recomendados preferencialmente para execuções com baixo número de processadores envolvidos.

Sobre o caso com 7 dimensões, além da redução do tempo total de execução, os valores obtidos de *speedup* e eficiência foram satisfatórios, atingindo níveis próximos ao linear com até 8 processadores, utilizando o escalonamento de nós adjacentes e 16 processadores utilizando o escalonamento de nós equidistantes. Utilizando 32 processadores, apesar de atingir níveis aceitáveis, ocorre uma queda nos valores. Esse fato ocorre devido ao compartilhamento de recursos de hardware proporcionado pela arquitetura *Hyper-Threading* e a sobrecarga produzida pelo mecanismo de paralelização.

AGRADECIMENTOS

Os autores agradecem as agências de fomento CAPES e Fundação Araucária, pelo apoio financeiro.

REFERÊNCIAS

[1] Machado, G. D. Produção de biodiesel por esterificação em coluna de destilação reativa: modelagem matemática. Dissertação de Mestrado, Universidade Estadual de Maringá, Maringá - PR, 2009.

[2] Corazza, F. C.; Oliveira, J. V.; Corazza, M. L. Application of a subdivision algorithm for solving nonlinear algebraic systems. Acta Scientiarum, v. 30, n. 1, 2008.

[3] Mullenix, N.; Povitsky, A. Parallel implementation of a tightly coupled ablation prediction code using MPI. In: Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on, 2009, p. 1 - 4.

[4] Gomes, J. L. Paralelização de algoritmo de simulação de Monte Carlo para a adsorção em superfícies heterogêneas bidimensionais. Dissertação de Mestrado, Universidade Estadual de Maringá, Maringá, PR, 2009.

[5] Costa, J. F. B. C. Método dos elementos finitos: Análise de desempenho computacional paralelo. Dissertação de Mestrado, Universidade de Aveiro, Aveiro, Portugal, 2010.

[6] Pinto, R. J. Aplicação de Processamento Paralelo ao Problema de Planejamento da Operação de Sistemas Hidrotérmicos Baseado em Cluster de Computadores. Tese de Doutoramento, UFRJ/COPPE, Rio de Janeiro, 2011.

[7] Oliveira, F. G. Aplicações Autônomas para Computação em Larga Escala. Dissertação de Mestrado, Universidade Federal Fluminense, Niterói, 2010.

[8] Modenesi, M. V. Análise de agrupamentos FCM utilizando processamento paralelo. Dissertação de Mestrado, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2008.

[9] Polymath-Software Numerical library problems involving simultaneous nonlinear equations. Disponível em: http://www.polymath-software.com/library/problemlist.shtml Acesso em: 20/01/2011, 2011.