UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

GUSTAVO ROMANO

Paralelização do Algoritmo de Geração de Redes Aleatórias Contínuas por *Simulated Annealing*

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Nicolas Bruno Maillard
Orientador

Porto Alegre, março de 2008

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Romano, Gustavo

Paralelização do Algoritmo de Geração de Redes Aleatórias Contínuas por *Simulated Annealing* / Gustavo Romano. – Porto Alegre: PPGC da UFRGS, 2008.

69 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2008. Orientador: Nicolas Bruno Maillard.

1. Simulated Annealing. 2. Paralelização. 3. Redes aleatórias contínuas. I. Maillard, Nicolas Bruno. II. Título.

AGRADECIMENTOS

Primeiramente agradeço a minha família, em especial aos meus pais, Sidnei e Teresinha, e a minha irmã, Gabriela por todo apoio, dedicação, carinho e amor.

Agradeço a todos professores que direta ou indiretamente contribuíram para conclusão do meu mestrado. E um especial agradecimento ao meu orientador Nicolas Maillard pelo incentivo, pelo apoio e por ter sempre acreditado na minha capacidade.

Agradeço todos amigos que de alguma forma contribuíram para alcançar essa conquista.

Agradeço ao CNPq pelo auxílio em forma de bolsa.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AG	Algoritmo Genético
CRN	*Continuous Random Network* (Redes Aleatórias Contínuas)
FC	Função Custo
FLOPS	*FLoating point Operations Per Second*
GM	*Global Minimum* (Mínimo Global)
HT	*Hyper Threading*
LabTeC	Laboratório de Tecnologia em Clusters
LM	*Local Minimum* (Mínimo Local)
MPI	*Message Passing Interface*
NP	*Non-Deterministic Polynomial time*
PVM	*Parallel Virtual Machine*
RAM	*Random Access Memory*
SA	*Simulated Annealing*
UCS	Universidade de Caxias do Sul
UFRGS	Universidade Federal do Rio Grande do Sul
VLSI	*Very Large-Scale Integration*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Esse trabalho tem dois objetivos principais: o primeiro deles consiste em apresentar o estado da arte sobre processos de otimização combinatorial dando uma ênfase especial ao método *Simulated Annealing* (SA). São apresentados seu histórico, funcionalidades, algoritmo genérico e propostas de paralelização presentes na literatura. Além disso, é apresentado o algoritmo de geração de redes aleatórias contínuas, algoritmo, esse, projetado por pesquisadores do Instituto de Física da UFRGS que utiliza o método SA para gerar redes que atendam certas restrições.

O segundo objetivo consiste em propor a paralelização desse algoritmo visando diminuir significativamente o tempo de geração de cada rede, que com o algoritmo seqüencial chega a demorar mais de um mês. Nessa etapa foi utilizada uma adaptação de um dos métodos propostos pela literatura juntamente com a técnica de divisão de domínio. Os resultados obtidos mostraram-se satisfatórios tanto em relação à qualidade numérica quanto à diminuição do tempo de processamento. Além disso, discute-se no trabalho a genericidade da proposta de paralelização a outros problemas baseados em SA.

Palavras-chave: Simulated Annealing, paralelização, redes aleatórias contínuas.

Paralelization of the Algorithm to Generate Continuous Random Network using Simulated Annealing

ABSTRACT

This work has two main goals: the first one is to present the state of the art on combinatorial optimization processes, with a special emphasis to the Simulated Annealing (SA) method. The work presents its history, features, generic algorithm and proposed parallelization present in the literature. Moreover, the algorithm to generate random networks continued is presented. This algorithm was designed by researchers of the UFRGS Physics Institute and it uses the SA method.

The second goal of this work is to propose a parallelization for this algorithm in order to decrease significantly the generation time of each network, that with the sequential algorithm reaches more than months. To do that was used an adaptation of one of the methods proposed by literature together with the domain partitioning technical. The results were satisfactory in terms of the numerical quality and in the decrease of the processing time. In addition, this work discusses the genericity of the proposed parallelization to other problems based on SA.

Keywords Simulated Annealing, paralelization, continuous random network.

1 INTRODUÇÃO

Processos de otimização combinatorial (busca do melhor resultado) podem ser utilizados para resolver boa parte dos problemas do dia-a-dia que envolvam um grande número de variáveis. Esses problemas se caracterizam pelo fato de que, normalmente, todas as soluções possíveis devem ser testadas até que a melhor solução seja encontrada. Isso faz com que, quando um grande número de variáveis está envolvido, sua solução demore um tempo extremamente grande para ser encontrada.

Dessa forma, existe a necessidade de encontrar outras maneiras de executar essas tarefas de modo que o tempo de execução seja reduzido para um patamar viável. Uma opção para tal é não buscar a melhor solução em si, mas sim, buscar uma solução suficientemente perto dela que atenda as necessidades. Essa solução pode ser encontrada através do uso de algoritmos baseados em heurísticas e meta-heurísticas.

Simulated Annealing (SA) é um exemplo de algoritmo baseado em meta-heurísticas que pode ser implementado em problemas de otimização de forma a reduzir o seu tempo de processamento. Ele recebe um destaque especial frente a outras técnicas pois facilmente foge de mínimos locais, garantindo que o resultado esteja próximo ao mínimo global. Porém, embora a complexidade numérica possa ser reduzida significativamente com o seu uso, para sistemas que dependam de muitas variáveis, esse método pode ainda consumir um tempo inviável até convergir em um resultado satisfatório.

Um exemplo de problema baseado em SA que exige um grande tempo de processamento é o “Algoritmo de Geração de Redes Aleatórias Contínuas (CRN)”. Esse algoritmo foi proposto por pesquisadores do Instituto de Física da UFRGS e visa encontrar novos materiais cuja dureza e módulo volumétrico sejam superiores ao do diamante. Para que se possa encontrar esses materiais, um grande número de átomos deve ser simulado. Isso faz com que simulações levem meses de processamento até convergir em um estado estável.

Para tentar reduzir esse tempo de processamento, muitas propostas de paralelização para o método SA vêm sendo feitas. Porém, devido às características seqüenciais do mesmo (cada passo depende do passo anterior), essa tarefa não é trivial.

Visando buscar maneiras eficientes de minimizar o tempo do processamento do método de SA, esse trabalho propõe a paralelização do mesmo a partir de adaptações de técnicas propostas pela literatura no contexto de SA. Essa proposta foi aplicada ao algoritmo de geração de CRN com resultados discutidos no decorrer do texto. Além disso, foi demonstrado que a proposta apresentada é versátil, podendo ser facilmente adaptada a outros problemas. Para ilustrar isso foi feita a aplicação da mesma ao algoritmo do Caixeiro Viajante baseado em SA.

Organização

Os capítulos que seguem estão divididos em duas partes principais. A primeira delas compreende o contexto científico desse trabalho e está dividido da seguinte forma: o capítulo 2 apresenta o processo de otimização combinatória, o método *Simulated Annealing* e as técnicas propostas pela literatura para paralelização de problemas baseados nele; o capítulo 3 apresenta o algoritmo de geração de Redes Aleatórias Contínuas por *Simulated Annealing*.

Na segunda parte do trabalho estão contidas as contribuições deste. Essa parte está dividida da seguinte forma: o capítulo 4 apresenta as estratégias adotadas nesse trabalho para a paralelização do algoritmo de geração de Redes Aleatórias Contínuas; o capítulo 5 apresenta a verificação numérica, análise teórica e experimental do algoritmo adotado; e, por fim, no capítulo 6 são apresentadas as conclusões obtidas nesse trabalho.

I

Contexto Científico

2 OTIMIZAÇÃO COMBINATÓRIA E O MÉTODO *SIMULATED ANNEALING*

A partir desse capítulo será apresentado o contexto científico do trabalho. Este capítulo está dividido da seguinte forma: na seção 2.1 é feita uma breve apresentação sobre o processo de otimização combinatória passando por noções sobre algoritmos probabilísticos e algoritmo de Metropolis e na seção 2.2 é apresentado o método de otimização combinatória *Simulated Annealing*, suas propostas de paralelização e ferramentas para tal.

2.1 Otimização Combinatória

O processo de otimização consiste em buscar a melhor forma de realizar uma determinada tarefa. Sua teoria é conhecida há séculos, porém, somente após o advento dos computadores ficou possível aplicá-la de forma mais eficiente (HROMKOVIC, 2001).

Grande parte dos problemas do dia-a-dia que possuem um número finito ou infinito de contáveis soluções alternativas podem ser formulados como problema de otimização combinatória. Essa, por sua vez, pode ser definida como a parte do estudo matemático que busca encontrar um ótimo arranjo, agrupamento, ordenação ou seleção de objetos discretos usualmente de número finito (KELLY, 1996).

Em um problema de otimização tem-se uma função objetivo e um conjunto de restrições, ambos relacionados às variáveis de decisão. O problema pode ser de minimização ou de maximização da função objetivo. A resposta para o problema, ou seja, o ótimo global, será o menor (ou maior) valor possível para a função objetivo para o qual o valor atribuído às variáveis não viole nenhuma restrição (HROMKOVIC, 2001).

A grande parte desses problemas são NP-Completos, sua solução para grande instâncias é uma tarefa que necessita computação intensiva e sua complexidade é dada por uma função exponencial de grau igual ao número de variáveis envolvidas (tamanho da entrada) (GAREY; JOHNSON, 1990). Por exemplo, a solução de um problema de caixeiro viajante com um universo de 100 cidades necessitaria 25×10^{136} anos para ser solucionado com uma busca completa (testando-se todas as possíveis soluções) em uma máquina com $12,5$ Tera-FLOPS (para se ter uma idéia, o universo tem aproximadamente 15×10^8 anos) (MOHARIL; LEE, 2005).

Para encontrar uma solução para esse problema em tempo hábil geralmente é necessário relaxar o significado de resolver. Para (HOROWITZ; SAHNI, 1978), relaxar significa admitir uma solução com o valor próximo ao da solução ótima que possa ser alcançado em um tempo viável.

Uma forma de alcançar essa solução é através do uso de heurísticas. Entre as técnicas

de heurísticas mais utilizadas estão as chamadas técnicas de busca local. Isto é, técnicas que visam minimizar a função custo em núcleos locais, fazendo com que o mínimo local seja encontrado de forma rápida. Dentre os algoritmos que utilizam essa técnica, pode-se destacar: algoritmos genéticos, busca tabu, algoritmo da colônia de formigas, busca por algoritmos gulosos aleatórios e redes neurais.

Embora esses algoritmos sejam relativamente rápidos, eles apresentam como desvantagem o fato de geralmente chegarem ao mínimo local de forma prematura, isto é, chegam a esse mínimo sem saber se ele é ou não o mínimo global.

O uso de metaheurísticas pode ser a solução para esse problema. Uma metaheurística consiste em um processo mestre iterativo que guia e modifica as operações de heurísticas subordinadas com o objetivo de produzir eficientemente soluções de alta qualidade, utilizando procedimentos de busca que evitam a parada prematura em ótimos locais que sejam distantes do ótimo global (KELLY, 1996). Pode-se destacar o algoritmo de Simulated Annealing como um representante dos algoritmos que utilizam metaheurísticas. Mais detalhes sobre ele serão apresentados na seção 2.2.

Grande parte dos algoritmos baseados em heurísticas e metaheurísticas são considerados algoritmos probabilísticos (ou aleatórios). Isto é, são algoritmos determinísticos que fazem escolhas aleatórias ao longo de sua execução, as quais auxiliam em seu processo de convergência.

2.1.1 Algoritmos probabilísticos

Os algoritmos probabilísticos podem ser do tipo Las Vegas e Monte Carlo. Um algoritmo do tipo Las Vegas usa a aleatoriedade para tentar agilizar o processo de resolução do problema. Ele sempre fornece uma solução correta para o problema, ou termina fornecendo uma resposta do tipo “não sei”, isso é, não pode realizar o cálculo. Sendo assim, pode-se considerá-lo um algoritmo confiável. Um algoritmo Monte Carlo, por sua vez, tem como característica utilizar da aleatoriedade para buscar o resultado e não apenas para agilizar a convergência. Ao término da execução ele fornece uma solução possivelmente correta ou retorna uma mensagem do tipo “não sei”. Sendo assim, algoritmos de Las Vegas são mais confiáveis que de Monte Carlo, porém, dependendo dos dados de entrada, podem levar um tempo extremamente maior para alcançar o resultado (HROMKOVIC, 2001).

Embora o método de Monte Carlo não obtenha resultados extremamente precisos, ele tem sido utilizado há bastante tempo como forma de obter aproximações numéricas de funções complexas em tempo hábil. Esse método, tipicamente envolve a geração de observações de alguma distribuição de probabilidades e o uso da amostra obtida para aproximar a função de interesse (MITZENMACHER; UPFAL, 2005).

De acordo com (HAMMERSELEY; HANDSCOMB, 1964) o nome “Monte Carlo” surgiu durante o projeto Manhattan na Segunda Guerra Mundial. No projeto de construção da bomba atômica, Ulam, von Neumann e Fermi consideraram a possibilidade de utilizar o método, que envolvia a simulação direta de problemas de natureza probabilística relacionados com o coeficiente de difusão do neutron em certos materiais.

O algoritmo de Metropolis, também conhecido por Algoritmo de Metropolis-Hastings é provavelmente o método Monte Carlo mais utilizado na Física. Detalhes sobre esse algoritmo serão apresentados na próxima subseção

2.1.2 Algoritmo de Metropolis

O algoritmo de Metropolis (METROPOLIS et al., 1953) é um método Monte Carlo para determinar propriedades macroscópicas de um sistema através de uma média sobre as configurações simuladas. Normalmente é implementado sob o formalismo canônico, no qual, o sistema simulado possui uma temperatura bem definida. Dessa forma, cada configuração do sistema pode ocorrer com uma freqüência proporcional à distribuição de Boltzmann. Segundo a distribuição de Boltzmann, em qualquer sistema em equilíbrio térmico a uma temperatura T, a probabilidade de se encontrar um estado a uma energia particular E é proporcional a $e^{\frac{-E}{k_b T}}$, em que k_b é a constante de Boltzmann. (HROMKOVIC, 2001).

Classicamente, para se determinar a probabilidade de uma configuração, precisa-se saber a ocorrência de todas as outras configurações, ou seja, é necessário o conhecimento da função de partição canônica (quantidade suficientemente grande de dados que represente as propriedades estatísticas de um sistema inteiro em equilíbrio termodinâmico). Porém, esse procedimento torna-se muito custoso quando o sistema depende de um número grande de variáveis. A eficiência do algoritmo de Metropolis reside no fato dele não precisar calcular a probabilidade das configurações em si, mas sim a relação entre as probabilidades. Sendo assim, a razão entre as probabilidades de duas configurações pode ser determinada independentemente das outras. Dadas duas configurações m e n quaisquer, a razão entre a probabilidade da configuração m (P_m) e a probabilidade da configuração n (P_n) pode ser escrita como (HROMKOVIC, 2001):

$$\frac{P_m}{P_n} = \frac{\frac{exp\left(-\frac{C_m}{k_b T}\right)}{Z}}{\frac{-exp\left(-\frac{C_n}{k_b T}\right)}{Z}} = exp\left(-\frac{C_m - C_n}{k_b T}\right)$$

em que k_b é a constante de Boltzmann, T a temperatura do sistema, C_m é o valor da função custo da configuração m, C_n é o valor da função custo da configuração n e Z é a função de partição canônica do sistema.

Nota-se, portanto, que, com as simplificações, o cálculo da função de partição Z não é mais necessário. Assim sendo, pode-se escrever o algoritmo de Metropolis que gera uma amostra seguindo a distribuição de Boltzman da seguinte forma (também representadas no fluxograma da figura 2.1) (HROMKOVIC, 2001):

1. Gera-se uma configuração inicial m com valores aleatórios para todos os graus de liberdade do sistema, respeitando as suas restrições;

2. Gera-se uma nova configuração-tentativa n, resultado de pequenas alterações nas coordenadas da configuração anterior;

3. Se a função custo da nova configuração for menor que a da configuração anterior, inclui-se a configuração na amostra, e essa passa a ser a configuração m. Caso contrário:

   (a) Gera-se um número aleatório entre 0 e 1;

   (b) Se esse número for menor que $e^{\frac{-\Delta C}{k_b T}}$ (em que ΔC é a variação da função custo), aceita-se na amostra a nova configuração, tornando-a a configuração m. Caso contrário, continua-se com a configuração original.

4. Repete-se os passos 2 e 3 até que algum critério de parada seja satisfeito. Cada uma dessas repetições é chamada de "passo **Monte Carlo**".

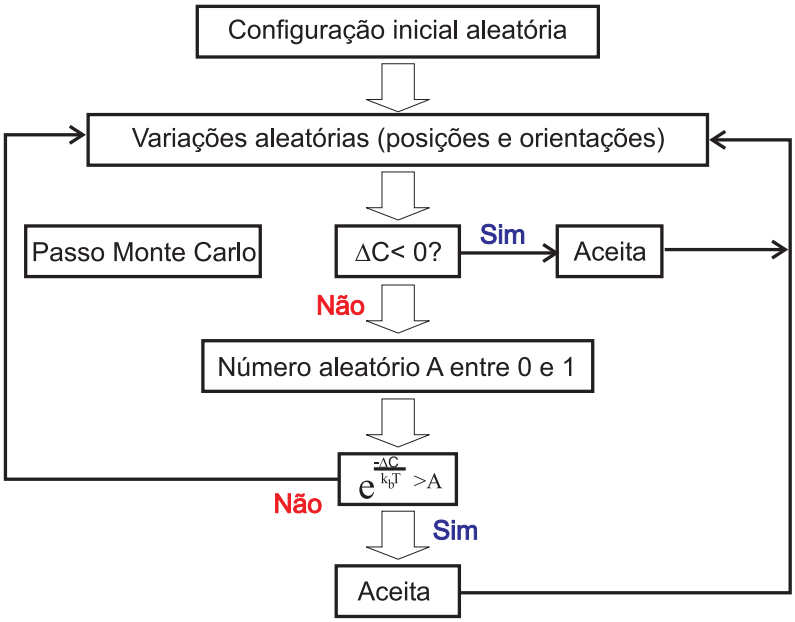


Figura 2.1: Fluxograma para o algoritmo de Metropolis

Em outras palavras, a cada iteração do algoritmo de Metropolis aplicado a um problema de otimização, gera-se o valor da função custo para duas soluções (a atual e uma nova solução). Esses valores são comparados, e caso a nova solução tenha melhorado o resultado, essa é aceita. Caso a nova solução seja pior que a atual, essa será aceita com uma probabilidade de $e^{\frac{-\Delta C}{k_b T}}$, em que ΔC é a variação da função custo e T é a temperatura atual do sistema.

2.2 Simulated Annealing

Simulated Annealing (SA) é um algoritmo genérico probabilístico utilizado para solucionar problemas de otimização combinatorial global de uma função em um grande espaço (KLIEWER, 2000; LEE; LEE, 1996). Suas características permitem que ele alcance de forma eficiente um mínimo global fugindo de mínimos locais (HENDERSON; JACOBSON; JOHMSON, 2003). Sua fácil implementação, propriedades de convergência e características de fuga de mínimos locais fizeram esse método muito popular nas últimas décadas (HENDERSON; JACOBSON; JOHMSON, 2003). Várias pesquisas demonstraram que SA pode ser muito eficiente para vários problemas de otimização, tais como, caixeiro viajante, design de VLSI, minimização lógica, design de código, processamento de imagem, entre outros (LEE; LEE, 1996).

2.2.1 Histórico e motivação

Simulated Annealing foi proposto em 1983 por (KIRKPATRICK; GELATT C. D.; VECCHI, 1983) e independentemente em 1985 por (CERNÝ, 1985), sendo baseado no trabalho pioneiro de (METROPOLIS et al., 1953). Seu nome é uma analogia ao processo físico de enrijecimento de sólidos, no qual um material é rapidamente aquecido e

lentamente resfriado para que suas falhas estruturais sejam removidas. De acordo com a termodinâmica, se o processo de resfriamento for suficientemente lento, a configuração final do material corresponderá à situação de mínima energia do sistema. Essa idéia é utilizada pelo algoritmo de SA para minimizar uma função qualquer.

2.2.2 Algoritmo

Para determinar o mínimo de energia, (KIRKPATRICK; GELATT C. D.; VECCHI, 1983) e (CERNÝ, 1985) utilizaram o método de Metropolis para explorar uma função custo em questão, e adicionaram uma idéia nova: a temperatura T deve variar lentamente de um valor elevado até um valor suficientemente baixo. Tem-se, então, um algoritmo simples e robusto para a localização de mínimos globais.

Na Figura 2.2, pode-se observar um pseudocódigo do algoritmo de SA. Ele começa com uma solução inicial (geralmente corresponde a uma distribuição aleatória dos dados) e uma temperatura. Essa temperatura deve ser suficientemente alta para que se possam explorar várias configurações. Em seguida, dois laços determinam o número de iterações que o algoritmo terá. O primeiro deles, determina o número de vezes que a temperatura será reduzida e o segundo o número de iterações com cada temperatura (quando se deseja que a temperatura diminua a cada iteração, basta deixar esse valor igual a 1 ou eliminar esse laço). A cada iteração do algoritmo um novo estado do sistema é gerado através de uma perturbação aleatória do estado atual (linha 5). Após o cálculo da variação da função custo (linha 6), a solução é aceita de acordo com o critério de aceitação de Metropolis descrito acima (linha 7). Caso aceita, essa passa ser a nova solução (linha 8). Por fim, na linha 11 é feita a alteração da temperatura com um fator λ. Esse fator deve ser definido de tal forma que a temperatura seja diminuída lentamente para garantir a convergência da execução (BEVILACQUA, 2002; HENDERSON; JACOBSON; JOHMSON, 2003). Alguns autores, tais como (HROMKOVIC, 2001), sugerem que o fator de redução fique entre $0,8$ e $0,99$, e que o número de passos com cada temperatura seja igual ao tamanho médio das vizinhanças, porém, esses valores não são rígidos, podendo variar de acordo com a simulação.

```
 1: S ← SolucaoInicial()
 2: T ← TemperaturaInicial
 3: for i ← 1 to NumeroDeReducoesDeTemperatura do
 4:     for j ← 1 to TamanhoDaEpoca do
 5:         S' ← NovaSolucao(S)
 6:         ΔC ← FuncaoCusto(S') − FuncaoCusto(S)
 7:         if ΔC < 0 ou Aceita(ΔC, T) then
 8:             S ← S'
 9:         end if
10:     end for
11:     T ← λT
12: end for
```

Figura 2.2: Algoritmo *Simulated Annealing*

Na representação do algoritmo de SA, pode-se observar com clareza que a única grande diferença entre esse algoritmo e o de Metropolis está na variação da temperatura. Se isolarmos a linha 11, teremos novamente um algoritmo de Metropolis.

A chave do algoritmo de SA está na possibilidade de fuga de mínimos locais através de aceitação de soluções que aumentem a função custo, como visto na Figura 2.3. Além disso, como a temperatura decresce no decorrer da execução, nas etapas iniciais existe

uma maior probabilidade de aceitação de novas soluções. Essa probabilidade vai diminuindo no decorrer da execução até chegar ao ponto (quando a temperatura estiver próxima de zero) em que apenas movimentos que melhorem a função custo sejam aceitos e, por fim, o mínimo global seja encontrado.

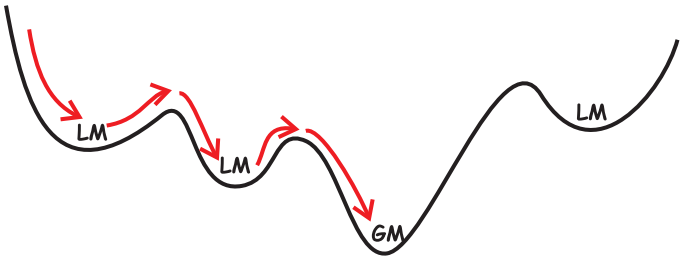


Figura 2.3: Possibilidade de exploração de soluções com função custo maior, faz com que o SA fuja de mínimos locais (LM) e encontre o mínimo global (GM).

Pode-se usar cadeias de Markov para modelar o comportamento do algoritmo de SA, e essas cadeias convergem de forma que todas as probabilidades estejam concentradas no conjunto de soluções globais ótimas. Isso prova que o algoritmo é convergente, caso contrário, ele convergiria para um mínimo local ótimo, que poderia ser ou não o mínimo global (HENDERSON; JACOBSON; JOHMSON, 2003). Mais detalhes sobre essa convergência podem ser vistos em (HENDERSON; JACOBSON; JOHMSON, 2003; ROMEO; SANGIOVANNI-VINCENTELLI, 1991; AARTS; KORST, 1988; MITRA; ROMEO; SANGIOVANNI-VINCENTELLI, 1985).

As provas apresentadas na literatura baseiam-se em execuções com um número de passos tendendo para ao infinito. Sabe-se que esse número de passos é inviável de ser processado. Portanto, as simulações devem ser realizadas com um número de passos suficientemente grande, porém, dentro do tempo disponível de processamento. Isso pode acarretar em não obrigatoriamente encontrar o valor ótimo, mas sim, um valor muito próximo a ele.

2.2.3 Paralelizações propostas na literatura para o método *Simulated Annealing*

Como foi visto na seção anterior, a necessidade de execução do laço principal do SA uma grande quantidade de vezes, para garantir a convergência do sistema junto ao mínimo global, faz com que sua execução seja de grande custo computacional. Uma abordagem direta para redução do tempo que esse custo gera é a paralelização do método. Entretanto, devido ao fato do SA ser inerentemente seqüencial (a cada iteração necessita-se dos dados da iteração anterior) e o custo de cada iteração ser relativamente baixo, a obtenção de uma paralelização eficiente não é fácil. Apesar disso, muitas implementações paralelas foram sugeridas e implementadas (MAHFOUD; GOLDBERG, 1995).

(MOHARIL; LEE, 2005) classificou as diferentes abordagens para paralelização do SA a partir das características das cadeias de Markov utilizadas para modelá-las. A figura 2.4 apresenta essas classificações e as mesmas são explicadas nas próximas subseções.

2.2.3.1 Cadeia de Markov Única

Existem diversas implementações (KRAVITZ; RUTENBAR, 1986; NATARAJAN, 1991; AARTS; KORST, 1988; GREENING, 1991; DELAMARRE; VIROT, 1993; SUN; SECHEN, 1994; BANERJEE; JONES; SARGENT, 1990) da paralelização do método de

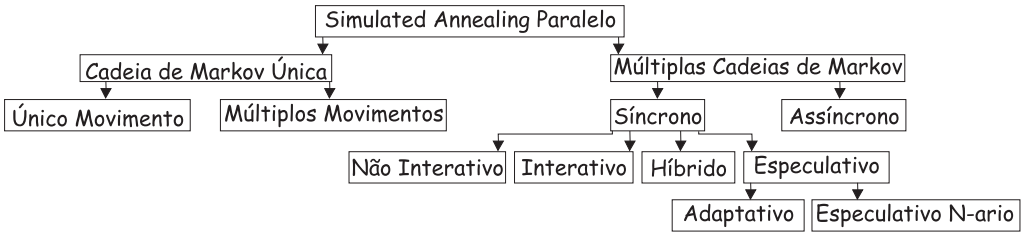


Figura 2.4: Classificações de *Simulated Annealing* Paralelos

SA, em que apenas uma cadeia de Markov é seguida. Nesse grupo, perturbações e avaliações de possíveis soluções candidatas são feitas em paralelo, mas apenas uma solução global é aceita ao fim de cada iteração. Esse grupo, pode ser dividido em dois subgrupos (CHANDY; BANERJEE, 1996):

1. **Abordagem de movimento único:** nessa abordagem, a avaliação de um movimento individual é paralelizada através da quebra em pequenas sub-tarefas. Devido à sua granularidade, essa abordagem de paralelização é restrita a arquiteturas de memória compartilhada;

2. **Abordagem de múltiplos movimentos:** cada processo gera e avalia um movimento de forma independente a cada iteração. Os movimentos de todos os processos são comparados e apenas uma das soluções segue para a próxima iteração.

Essas abordagens geram resultados muito semelhantes aos da execução seqüencial em termos de comportamento de convergência, porém, exploram apenas um pequeno nível de paralelismo. Além disso, a decomposição em grão-fino do problema acarreta uma grande sobrecarga de comunicação que degrada significativamente a eficiência em um sistema de multiprocessadores de larga escala (CHEN; FLANN; WATSON, 1998).

2.2.3.2 Múltiplas Cadeia de Markov

Uma alternativa para contornar os problemas do uso de uma única cadeia de Markov (paralelismo limitado e grande sobrecarga) é eliminar o requisito de que todos os processos tenham que seguir o mesmo caminho. Ou seja, todos os processos têm permissão para seguir sua própria cadeia de Markov. As várias abordagens propostas para o uso de múltiplas cadeias de Markov podem ser classificadas em dois grandes grupos:

1. **Múltiplas Cadeia de Markov Síncronas:** nessa abordagem a temperatura inicial, o coeficiente de resfriamento e as condições de equilíbrio são decididas independentemente em cada processo. Interações entre processos ou troca de soluções intermediárias são feitas de forma síncrona entre todos os processos. Pode-se dividir essa abordagem em:

   (a) **Não Interativa:** processos perturbam e avaliam as soluções de forma totalmente independente, sem nenhum tipo de comunicação para troca de soluções entre eles durante a simulação. Ao fim dos processos de simulação, a melhor solução é apresentada (LEE; LEE, 1996);

   (b) **Interativa:** processos perturbam e avaliam as soluções locais e periodicamente trocam essas soluções com outros processos. Essa troca acarreta em uma barreira de sincronização entre todos os processos. Implementações

dessa abordagem foram feitas, porém, o *speedup* conseguido foi baixo e não linear (LEE; LEE, 1996);

(c) **Híbrido:** uma das abordagens de processamento híbrido é o uso de SA juntamente com Algoritmos Genéticos (AG). Essa abordagem combina características de ambos os algoritmos para superar os pontos fracos que cada um possui quando usados separadamente. Uma característica interessante dessa abordagem é que o desempenho escala linearmente com o aumento do número de processadores, porém, tem-se um decréscimo na qualidade numérica da solução final (CHEN; FLANN; WATSON, 1998; MAHFOUD; GOLDBERG, 1995);

(d) **Especulativo:** por envolver computação especulativa, a concorrência é introduzida no algoritmo seqüencial. Isso é alcançado através da predição de cada movimento gerado antes que esse seja avaliado. Baseado nisso, movimentos subseqüentes podem ser propostos e avaliados antes que a decisão de aceitar ou rejeitar o movimento seja feita (CASOTTO; ROMEO; SANGIOVANNI-VINCENTELLI, 1987; CHANDY et al., 1997; CHANDY; BANERJEE, 1996). Esse subgrupo pode ainda ser classificado em duas subclasses:

i. **Adaptativo:** o algoritmo usado para baixas temperaturas é uma versão adaptada do algoritmo especulativo, em que, o número de processos alocados para solucionar o problema e o número de movimentos avaliados por processo entre os pontos de sincronismos muda de acordo com a temperatura. Nas temperaturas mais altas, a abordagem interativa é adotada (KNOPMAN; AUDE, 1997);

ii. **Especulativo N-ario:** em vez de usar uma árvore binária como o modelo especulativo convencional, essa implementação usa uma árvore especulativa N-aria, em que N é o número de processadores usados (SOHN, 1995).

2. **Assíncrono:** essa versão é similar à implementação síncrona interativa, exceto por acessar a solução global intermediária de forma assíncrona. Devido a isso, o tempo ocioso presente na sincronização é completamente eliminado (LEE; LEE, 1996; CHANDY et al., 1997; CHANDY; BANERJEE, 1996). O problema dessa implementação é conseguir um estado global atualizado, visto que cada processador estará em uma etapa da paralelização.

2.2.4 Ferramentas para *Simulated Annealing* paralelo

Existem poucas ferramentas genéricas (que possam ser adaptadas a uma aplicação diferente da projetada) para simulações de SA em paralelo. Uma dessas ferramentas é a *parSA-Library* (KLIEWER, 2000).

Segundo os autores, *parSA-Library* foi projetada para ser um *framework* paralelo confortável e eficiente para a simulação de diferentes aplicações baseadas em SA. Ela foi implementada em linguagem C++ e utiliza conceitos de orientação a objeto. A comunicação entre os processos é feita através da biblioteca de troca de mensagens MPI, o que confere uma grande portabilidade para diferentes arquiteturas sem a necessidade de mudança no código. Ainda segundo os autores, a paralelização do método SA é transparente para o usuário, adaptando-se a diferentes plataformas e problemas. O usuário deve apenas informar como será o processo de resfriamento, critérios de avaliação da função custo, forma

de iteração entre vizinhos e critério de término da simulação (KLIEWER; TSCHöKE, 1999).

parSA-Library implementa o método de múltiplas cadeia de Markov síncronas interativas (KLIEWER; TSCHöKE, 2000). Isso é, cada processador envolvido na simulação executa um certo número de passos independente e de tempos em tempos os dados são trocados entre eles. Uma vez de posse de todas as soluções, a melhor delas é selecionada e essa passa a ser a solução base de todos os processos.

O desempenho do algoritmo depende principalmente do tempo de agendamento entre cada sincronização. Quanto mais freqüente, menor o desempenho, porém melhor a qualidade numérica. Um grande inconveniente dessa biblioteca é que ela não permite outras formas de paralelização, ficando o programador restrito apenas a esse método.

2.3 Message Passing Interface (MPI)

Message Passing Interface (MPI) é um padrão para comunicação de dados na computação paralela (GROPP et al., 1996). Seu principal objetivo é disponibilizar uma interface que seja largamente utilizada no desenvolvimento de programas baseados em troca de mensagens. Além de garantir a portabilidade dos programas paralelos, a interface MPI possibilita a implementação eficiente de sua especificação para diversos tipos de máquinas paralelas existentes (reais ou *clusters* de *workstations*) (SNIR et al., 1996). MPI define apenas o modelo de troca de mensagens tais como nomes de funções, seqüências de chamadas e resultados de sub-rotinas. Existem diversas implementações do padrão MPI, cada qual com suas características e otimizações de código (OPENMPI, 2007; MPICH, 2007; LAMMPI, 2007). Considerada uma evolução de *Parallel Virtual Machine* (PVM) (GEIST et al., 1994), a biblioteca pode ser utilizada em programas escritos em linguagem FORTRAN, C ou C++.

No padrão MPI, uma aplicação é constituída por um ou mais processos que podem ser executados em máquinas distintas, os quais se comunicam através de funções de envio e recebimento de mensagens via interface de rede. Assim, as implementações do padrão oferecem uma infra-estrutura para a computação paralela na qual é possível trocar informações entre vários processadores (GROPP; LUSK; THAKUR, 1999).

MPI disponibiliza diferentes formas de comunicação. Os mecanismos de comunicação mais simples que podem ser utilizados são a comunicação ponto a ponto, onde ocorrem operações de troca de mensagens de um determinado processo com outro. Estruturas mais refinadas de comunicação são obtidas usando um grupo de processos que invocam operações coletivas de comunicação (SNIR et al., 1996).

Os recursos encontrados em MPI são muito importantes pois garantem implementações paralelas com mecanismos de comunicação eficientes e uma maior independência entre as execuções dos processos. Além desses, MPI possui ainda mecanismos para a criação de estruturas cartesianas, bem como as funções necessárias para o mapeamento e acesso aos processos.

Nesse capítulo foram apresentados conceitos sobre otimização combinatória e o algoritmo genérico probabilístico utilizado para solucionar problemas de otimização combinatorial global de uma função em um grande espaço, conhecido como *Simulated Annealing*. Foram apresentadas suas características, seu histórico, seu algoritmo genérico, propostas de paralelização e ferramentas para tal.

No próximo capítulo será apresentado um trabalho prático que utiliza o método *Simulated Annealing* para gerar redes aleatórias contínuas.

3 GERAÇÃO DE REDES ALEATÓRIAS CONTÍNUAS POR *SIMULATED ANNEALING*

Neste capítulo será apresentado o trabalho realizado por (JORNADA, 2007) para geração de redes aleatórias contínuas utilizando o método *Simulated Annealing*. Esse trabalho serviu de base para a paralelização realizada no presente trabalho. Essa paralelização será apresentada no capítulo 4.

Esse capítulo está dividido da seguinte forma: na seção 3.1 o trabalho é contextualizado; na seção 3.2 é apresentado o algoritmo proposto; na seção 3.3 é apresentada a implementação seqüencial do algoritmo; na seção 3.4 são apresentadas as estruturas de dados utilizadas; na seção 3.5 são apresentadas as definições de parâmetros para as simulações; na seção 3.6 são apresentados os resultados obtidos com essa versão; e na seção 3.7 são apresentadas considerações finais sobre o capítulo.

3.1 Contexto do trabalho

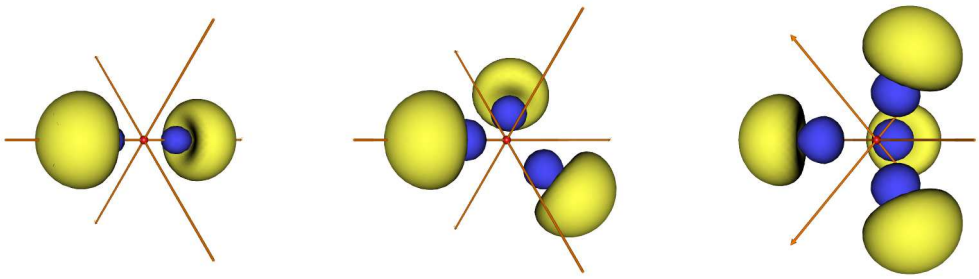
Recentemente, uma das áreas da ciência dos materiais em que se tem tido um grande interesse é a nanotecnologia. A possibilidade de se obter compostos de propriedades diversas apenas por diferentes configurações estruturais abre inúmeras possibilidades de pesquisas. Nessa perspectiva, o elemento que tem se destacado é o carbono. Diamante, grafite e nanotubos são alguns exemplos de materiais formados unicamente por esse elemento, os quais possuem propriedades físicas distintas determinadas pela forma com que os átomos arranjam-se.

A possibilidade do carbono hibridizar-se[1] de diversas formas, permite que ele constitua diversos materiais com características distintas. Um átomo de carbono isolado tem quatro orbitais de valência, cada qual ocupado por um elétron: um orbital s, de simetria esférica, e três orbitais p (px, py, pz) em forma de haltere. Uma representação das diferentes hibridizações do carbono é fornecida na figura 3.1, em que observa-se um átomo de carbono sp realizando 2 ligações, um sp^2 realizando 3 ligações e um átomo sp^3 realizando 4 ligações.

Das diferentes fases que o carbono pode assumir, a amorfa (mista) tem se mostrado de alta utilidade prática. Filmes finos com *Diamond-like carbon* exibem alto grau de dureza, sendo empregados para o recobrimento de peças ópticas. Pela possibilidade desse elemento se apresentar em diferentes hibridizações (sp^3, sp^2 e sp), estruturas amorfas de carbono podem ter propriedades muito diferentes, dependendo da quantidade de cada fase no material.

Um modelo relativamente realístico para descrever uma grande gama desses mate-

[1] Por hibridização, entende-se o processo de formação de orbitais eletrônicos híbridos.



(a) Hibridização sp (b) Hibridização sp^2 (c) Hibridização sp^3

Figura 3.1: Exemplo de átomos com diferentes hibridizações. Figura disponível em (IMMEL, 2007).

riais não-cristalinos (amorfos) é o das redes aleatórias contínuas (CRN), proposto por (ZACHARIASEN, 1932) originalmente para explicar a organização molecular no vidro. Trata-se da idealização de que o material não possui *dangling-bonds*, isto é, que cada átomo realiza o número de ligações que sua hibridização permite. Dessa forma, tem-se um material sem nenhum tipo de defeito químico. A estrutura possuirá uma ordem de curta distância, mas não apresentará periodicidade cristalina. Um exemplo de CRN pode ser visto na Figura 3.2.

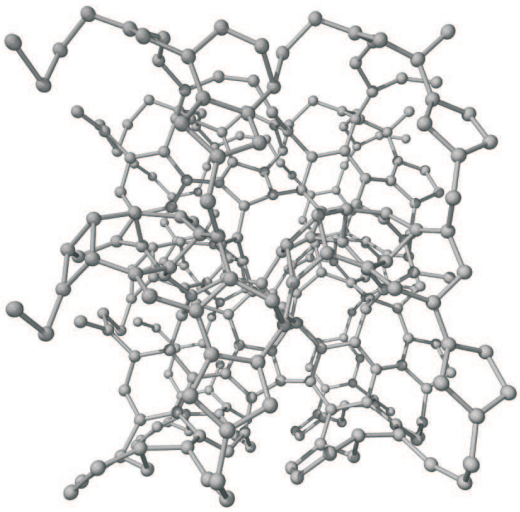


Figura 3.2: Exemplo de CRN

Na pesquisa de novos materiais, pesquisadores do Instituto de Física da Universidade Federal do Rio Grande do Sul (UFRGS) propuseram a geração de redes aleatórias contínuas formadas de átomos de carbono com hibridização mista sp^3/sp^2 (JORNADA, 2007). Essas redes podem modelar novos materiais (reais ou hipotéticos) cuja dureza e módulo volumétrico sejam significativamente superiores aos do diamante (PEROTTONI; JORNADA, 2001).

Embora as CRN tenham sido utilizadas com sucesso para a descrição de semicondutores amorfos desde meados da década de 1930 (BARKEMA; MOUSSEAU, 2000), os algoritmos que são usados para gerar essas redes normalmente funcionam apenas para

aquelas formadas somente por átomos com hibridização sp^3 (como é o caso do algoritmo WWW (WOOTEN; WINER; WEAIRE, 1985)).

3.2 Algoritmo de Geração de Redes Aleatórias Contínuas por *Simulated Annealing*

Para tentar contornar a limitação na escolha da hibridização presente nos algoritmos descritos na seção anterior, pesquisadores da física (JORNADA, 2007) da UFRGS desenvolveram uma versão seqüencial de um algoritmo capaz de gerar CRN formadas por átomos com hibridização sp^3 e sp^2. Esse algoritmo tem por objetivo gerar um sistema estável que compreenda certas restrições. Para alcançar essa estabilidade, o algoritmo utiliza-se do método de SA.

Basicamente, tem-se como estado inicial do algoritmo um conjunto de átomos posicionados de forma aleatória em um espaço. Cada um desses átomos representa uma variável no sistema, e cada uma delas possui 3 possíveis graus de mudança (eixos x, y e z). Esses átomos inicialmente posicionados não precisam necessariamente respeitar as restrições impostas, podendo conter configurações bem diferentes das esperadas.

Para que se pudesse gerar uma função custo de avaliação para o problema, e essa pudesse ser aplicada no método SA, foi implementado o conceito de hibridização. Isto é, dada a distribuição de átomos, verificava-se os átomos que estavam a uma distância menor que 1,75 Å[2] (máxima distância física permitida entre dois átomos). Para esses, uma ligação é criada e seu grau de hibridização é atualizado. Na figura 3.3 pode-se observar um plano bidimensional em que são apresentados alguns átomos, a forma com que eles se ligam e os seus graus de hibridização. Observa-se também, que o átomo posicionado na primeira coluna liga-se com o átomo posicionado na última. Isso acontece, pois se deseja criar uma noção de continuidade ao sistema.

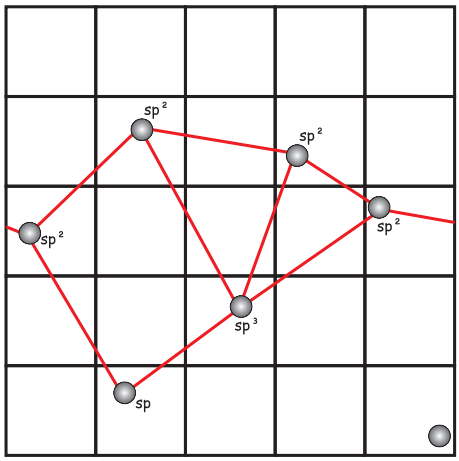


Figura 3.3: Exemplo da função de hibridização dos átomos

A partir da hibridização dos átomos, pôde-se calcular a função custo do estado atual do sistema. Para tal, utilizou-se de duas componentes. A primeira foi a energia de ligação dos átomos do sistema. Essa energia E é calculada através da soma da energia de cada átomo E_i, que, por sua vez, é dada em função da distância entre ele e os vizinhos, bem

[2]O Ångström (Å) é uma medida de comprimento que equivale a $10^{-10}m$.

como dos ângulos formados em cada uma de suas ligações. A fórmula do cálculo da energia de cada átomo i pode ser representada por:

$$E_i = \sum_{r_{ij}<r_{max}} \left[k_r(r_{ij} - r_{0_{ij}})^2 + \sum_{r_{ik}<r_{max}} k_a(\Theta_{jik} - \Theta_{0_i})^2 \right], \tag{3.1}$$

em que r_{ij} é a distância entre o átomo i e o átomo j, $r_{0_{ij}}$ é a distância de equilíbrio interatômica entre os dois átomos, Θ_{jik} é o ângulo formado pelas ligações dos átomos i, j e k e Θ_{0_i} é o ângulo de equilíbrio característico do átomo i.

A segunda componente da função custo é o custo de hibridização. Para esse cálculo, são verificados os valores de hibridização (com quantos átomos cada átomo esta ligado) de todos os átomos do sistema e comparados com os valores esperados no início da simulação. Caso esses valores sejam diferentes, um custo é aplicado a essa diferença. Pode-se expressar esse custo por:

$$H = \frac{1}{N} \sum_i C_i(N_i - N_{0_i})^2, \tag{3.2}$$

em que N é o número de átomos do sistema, C_i é o custo de cada hibridização do tipo i errada no sistema, N_i é o número de átomos com hibridização i presentes no sistema e N_{0_i} é o número de átomos esperados com essa hibridização.

De posse das duas, pode-se calcular a função custo através da aplicação de um peso (p_h e $(1 - p_h)$) a cada uma das componentes. A fórmula para a função custo (FC) pode ser expressa por:

$$FC = p_h.H + (1 - p_h).E. \tag{3.3}$$

Para o conceito de perturbação de uma configuração, considerou-se um pequeno deslocamento na posição de um determinado átomo nos seus 3 eixos. Esse deslocamento faz com que ligações entre átomos sejam desfeitas e novas sejam criadas. Alterando, assim, o custo de hibridização. Além disso, o deslocamento faz com que as distâncias e os ângulos entre os átomos ligados se alterem. A variação da função custo pode ser utilizada justamente com o critério de Metropolis para validar ou não cada mudança.

3.3 Implementação

O sistema baseado no algoritmo foi desenvolvido em linguagem C, seu pseudocódigo está representado na Figura 3.4 e será explicado em detalhes nos próximos parágrafos.

O algoritmo começa pela leitura do arquivo que contém os parâmetros da simulação (linha 1). Nele estão contidas informações tais como: número de átomos, átomos com hibridização sp^2 e sp^3 desejados, densidade do sistema, representatividade do custo de hibridização, definição de constantes, número de passos do SA e temperatura inicial e final do sistema (um exemplo do arquivo de entrada pode ser visto na figura 3.6). Em seguida, os átomos informados nos parâmetros são posicionados de forma aleatória no espaço (linha 2).

Para tornar mais fácil a localização dos átomos e executar de forma mais eficiente cálculos locais, criou-se o conceito de células. Cada célula consiste em uma região endereçada do espaço x, y, z que pode conter nenhum, um ou vários átomos. A quantidade de células existente no sistema depende do número de átomos e a densidade do mesmo (a densidade varia de acordo com o número de átomos de cada hibridização), podendo

```
 1: le_parametros();{lê parâmetros de entrada do sistema}
 2: posiciona_atomos();{posiciona os átomos aleatoriamente no sistema}
 3: do_cells();{coloca cada átomo em sua célula do sistema}
 4: hibri_all();{hibridiza todos os átomos}
 5: tote();{calcula a energia total do sistema}
 6: for proc = 0 to NR_PROC − 1 do
 7:    temperatura = temp[proc];{inicializa a temperatura para o processo}
 8:    for iter = 0 to MAX_ITER[proc] − 1 do
 9:       if iter%RESIZE_UNI_EACH == 0 then
10:          muda_tamanho_universo();{muda o tamanho do universo}
11:       end if
12:       at = atomos[rand(NATM)];{pega um átomo aleatoriamente}
13:       E_local_old = local_energy(at);{calcula a energia local de onde o átomo está inser.}
14:       Hibri_old = hibri_cost(at);{calcula o custo de hibridização}
15:       at→x = at→x + randf()*MAX_DESLOC;{desloca o átomo na direção x}
16:       at→y = at→y + randf()*MAX_DESLOC;{desloca o átomo na direção y}
17:       at→z = at→z + randf()*MAX_DESLOC;{desloca o átomo na direção z}
18:       atm_hibri(at);{re-hibridiza o átomo atual}
19:       E_local_new = local_energy(at);{calc. a nova energia local de onde o átomo está}
20:       Hibri_new = hibri_cost(at);{calcula o novo custo de hibridização}
21:       dE = E_local_new - E_local_old;{calcula a variação de energia}
22:       dH = Hibri_new - Hibri_old;{calcula a variação do custo de hibridização}
23:       dfc = P_HIB * (dH) + (1 − P_HIB) * (dE) ;{calcula a variação da função custo}
24:       if aceita(dfc, temperatura) then
25:          atualiza_posicao();{atualiza a posição do átomo}
26:       else
27:          desfaz_movimento();{volta o átomo para posição anterior}
28:       end if
29:       temperatura = temperatura * λ {diminui a temperatura em um fator λ < 1}
30:    end for
31: end for
```

Figura 3.4: Algoritmo seqüencial para geração de CRN

variar de 125 $(5 \times 5 \times 5)$ até mais de 6859 $(19 \times 19 \times 19)$. O passo seguinte do algoritmo consiste em posicionar cada átomo em uma dessas células do sistema (linha 3). A Figura 3.5 mostra um exemplo bidimensional do funcionamento do sistema de células, nela estão representados alguns átomos posicionados aleatoriamente e as células em que eles estão armazenados. Observa-se que com o sistema de células pode-se localizar de forma rápida os átomos posicionados na mesma célula, bem como nas células vizinhas. Por exemplo, sem precisar percorrer todo o universo de átomos, pode-se saber que $A7$, $A9$ e $A10$ estão nas proximidades de $A1$. Também se pode notar que, a distribuição dos átomos nas células se faz através da posição dos átomos. Dessa forma algumas células terão mais átomos que outras, porém a medida que o sistema tende a convergir para um estado estável, essas diferenças tendem a diminuir.

A seguir, hibridiza-se todos os átomos (linha 4) (para isso o sistema de célula é bastante útil) e cálcula-se a energia total do sistema (linha 5).

Após posicionados e calculados os valores da distribuição inicial, inicia-se um laço (linha 6) que controla o número de processos de resfriamento que irá ocorrer no sistema (pode-se optar um resfriamento uniforme do início ao fim, ou várias etapas de resfri-

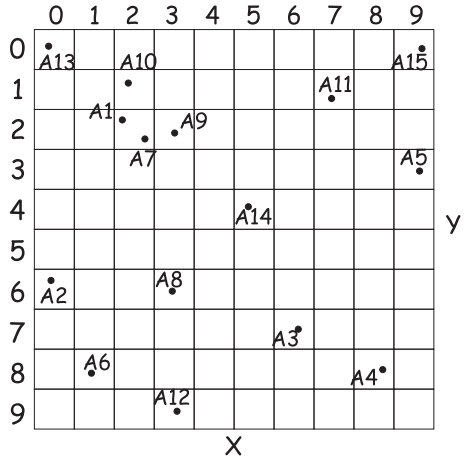


Figura 3.5: Exemplo da função de posicionamento dos átomos nas células

amento, em que cada uma delas pode acontecer em velocidades diferentes). Em cada processo uma temperatura inicial é definida (linha 7) e a seguir um conjunto de iterações (linhas 8 a 30) de SA são executadas.

A cada iteração um átomo é selecionado aleatoriamente (linha 12), e a energia local (linha 13) de onde ele está inserido é calculada (entende-se como energia local a energia da célula onde o átomo está inserido e das células vizinhas a ela). Esse cálculo é feito de forma local para diminuir os custos de processamento.

O passo seguinte é fazer uma pequena perturbação no sistema através da alteração aleatória da posição do átomo selecionado (linhas 15 a 17). Após isso, a hibridização do átomo é refeita e a nova energia local (linha 19) e o novo custo de hibridização (linha 20) são calculados, bem como a variação desses valores (linhas 21 e 22). A partir da variação desses valores, a variação da função custo é calculada (linha 23).

Uma vez obtida a função custo, verifica-se se esse movimento é aceito ou não através do critério de Metropolis (linha 24) e da temperatura atual do sistema. Se for aceito, os novos valores são atualizados. Caso contrário, o átomo é colocado em sua posição anterior. Por fim, a temperatura do sistema é atualizada (linha 29) e finaliza-se o laço.

Cabe notar que na linha 9 existe um teste que verifica se a iteração atual corresponde a uma iteração de mudança do tamanho do universo. Caso seja, essa operação, que consiste em alterar o tamanho do espaço onde os átomos estão inseridos, é executada. Uma vez feito isso, a célula em que cada átomo está inserido, bem como as ligações entre eles podem ser alteradas. Essas alterações gerarão uma nova função custo, que será avaliada pelo critério de Metropolis. Caso seja aceita, a nova configuração fará parte do sistema, caso contrário, volta-se à configuração anterior.

A figura 3.7 ilustra um exemplo do funcionamento do laço interno do algoritmo. Nesse exemplo está representado um corte bi-dimensional de um sistema com sete átomos devidamente hibridizado (a). O primeiro passo é selecionar um átomo aleatoriamente (b). Em seguida, é verificada a energia dos átomos localizados nas células próximas a ele (c). No caso do átomo selecionado, por exemplo, por ele ser um átomo com hibridização sp^3 espera-se que os átomos que estejam ligados a ele tenham uma distância de 1,544 Å e que os ângulos formados sejam de 109,471°. Caso os valores sejam diferentes, um custo é aplicado a essa diferença.

Uma vez calculada a energia local, calcula-se o custo de hibridização do sistema (d). Para tal é necessário percorrer todos os átomos e verificar quantos átomos tem-se de cada

hibridização. Após isso, verificam-se quantos divergem dos parâmetros e se aplica um custo sobre essa diferença. No exemplo, caso deseja-se um sistema totalmente sp^2 (todos átomos com três ligações), deve-se aplicar um custo ao átomo selecionado e aos outros dois átomos que não possuem hibridização correta.

Após calcular os valores do sistema atual, faz-se um pequeno deslocamento na posição do átomo selecionado (e). E refaz-se a hibridização do átomo (f), isso é, criam-se novas ligações com os átomos que ficaram mais próximos do átomo selecionado após o deslocamento e retiram-se as ligações entre os átomos que se distanciaram em demasia.

Em seguida, calcula-se a nova energia local (g), e o novo custo de hibridização (h), bem como a variação entre eles. Caso seja aprovado pelo critério de Metropolis, essa passa a ser a nova configuração do sistema (i), caso contrário volta-se a configuração anterior (a).

Nota-se que, para avaliar se um movimento é aceito ou não, depende-se de dois cálculos: o da energia e o da hibridização. O primeiro é local, isso é, depende apenas dos átomos posicionados nas proximidades do selecionado. O segundo, por sua vez, depende de todos os átomos presentes no sistema.

```
<main>
title=Simulação teste  # Título da simulação

dump=15           # Intervalo (em minutos) entre cada Dump

<system>
atoms=1024        # Número de átomos

cost_hibri=0.30   # Parte do custo final referente a hibridização

prop_sp3=0.8      # Proporção de átomos sp3
prop_sp2=0.2      # Proporção de átomos sp2

<params>
resize=210        # Intervalo entre cada mudança do tamanho do universo

cost_nb=3.4       # Custo para átomos sem ligações
cost_sp0=3.2      # Custo para átomos com uma ligação
cost_sp=2.0       # Custo para átomos com duas ligações
cost_sp2=1.0      # Custo para átomos com três ligações
cost_sp3=1        # Custo para átomos com quatro ligações
cost_sp4=2.0      # Custo para átomos com mais do que quatro ligações

k_a=2.0           # Valor da constante angular

<annealing>
steps=1e8(0.05, 0.88, 0.07)  # Número de passos de cada etapa
temp=(1e4, 4, 0.5, 1e-3)     # Temperatura inicial e final de cada etapa
```

Figura 3.6: Exemplo do arquivo de entrada para o sistema

3.4 Estruturas de dados

As duas principais estruturas de dados do sistema são responsáveis pela representação dos átomos (`Vatomo`) e pela representação das ligações entre eles (`Vbond`). Uma

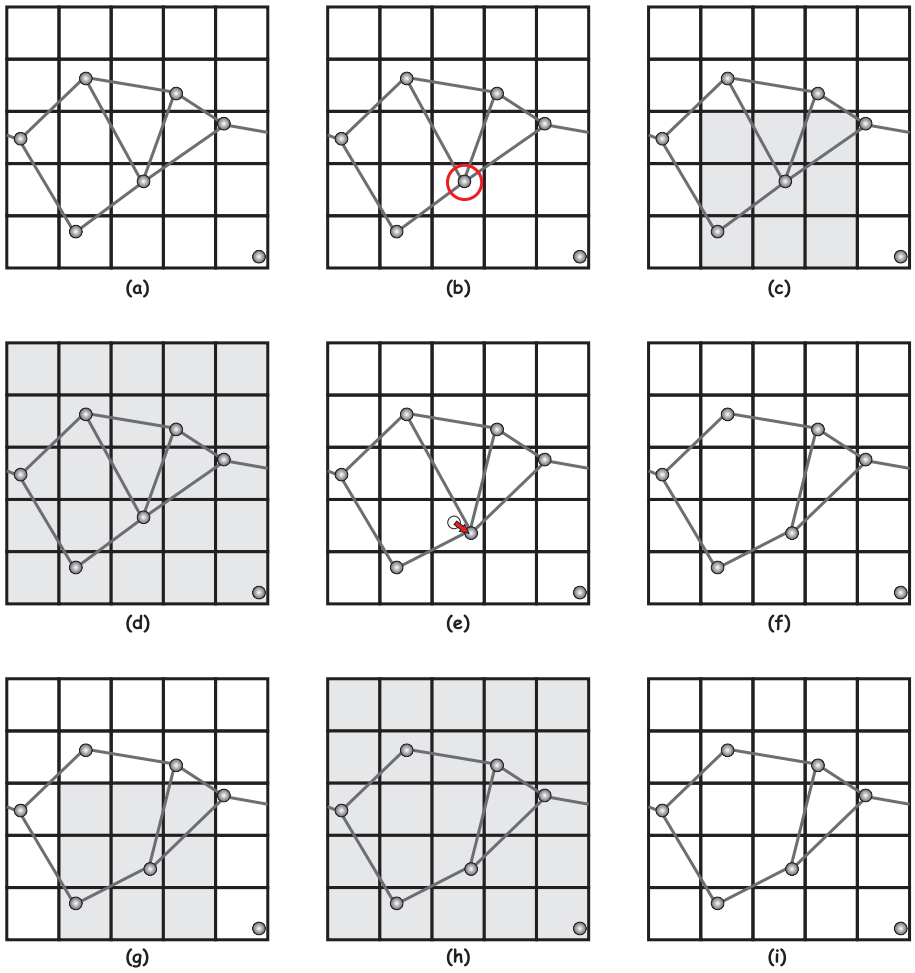


Figura 3.7: Exemplo do funcionamento do algoritmo de geração de CRN

representação dessas estruturas pode ser vista na Figura 3.8. Pode-se observar que na estrutura do átomo são armazenadas as coordenadas do mesmo, a última posição deste (usada quando o movimento não é aceito e tem-se que voltar à posição anterior), uma referência à primeira ligação deste, referências para o próximo átomo dentro da mesma célula, bem como para o anterior. Além disso, são armazenados a posição do átomo no vetor de átomos e a célula em que esse átomo está localizado. Já na estrutura de ligação, são armazenados a próxima ligação e a ligação anterior do átomo, a ligação conjugada (ligação oposta a essa ligação), o átomo que esta sendo ligado e o vetor de separação entre os 2 átomos.

A partir dessas duas estruturas é possível de forma fácil armazenar os dados e realizar os principais cálculos necessários para a simulação. Com elas, não é necessário percorrer todo o universo de átomos para realizar cálculos locais.

3.5 Definição de parâmetros

Antes de iniciar as simulações, um conjunto de parâmetros precisam ser definidos. Entre eles estão: número de átomos a serem simulados; átomos com hibridização sp^2 e sp^3 desejados; número de etapas; número de passos do SA e temperatura inicial e final de cada etapa.

A escolha correta desses valores implica em uma maior facilidade de encontrar o

```
struct Vatomo{
   float p[3];             // posição do átomo
   float p_old[3];         // última posição do átomo
   short int hibri;        // número de ligações (2=sp, 3=sp2, 4=sp3)
   Tbond *B0;              // referência à primeira ligação do átomo
   Tatomo *cell_next;      // próximo átomo na célula
   Tatomo *cell_previous;  // átomo anterior na célula
   int idx;                // índice do átomo no vetor de átomos
   int cell_idx[3]         // célula em que o átomo se encontra
}

struct Vbond{
   Tbond *previous;        // ligação anterior do átomo
   Tbond *next;            // próxima ligação do átomo
   Tbond *conjugate;       // ligação conjugada
   Tatomo *atm;            // átomo ao qual se liga
   float sep[3];           // vetor de separação entre os átomos
}
```

Figura 3.8: Principais estruturas de dados do sistema

mínimo global para a simulação. Valores como número de átomos e hibridização são definidos de acordo com a simulação que se deseja fazer. Já o número de etapas geralmente é definido como três para poder representar inicialmente um sistema líquido, depois uma etapa de congelamento e por fim uma etapa sólida. Os valores representados na tabela 3.1, são sugestões relativas ao percentual de iterações e da temperatura inicial e final de cada etapa. Esses valores foram obtidos através de diversos testes executados por (JORNADA, 2007).

Tabela 3.1: Valores para diferentes etapas da simulação (as temperaturas estão em uma escala arbitrária).

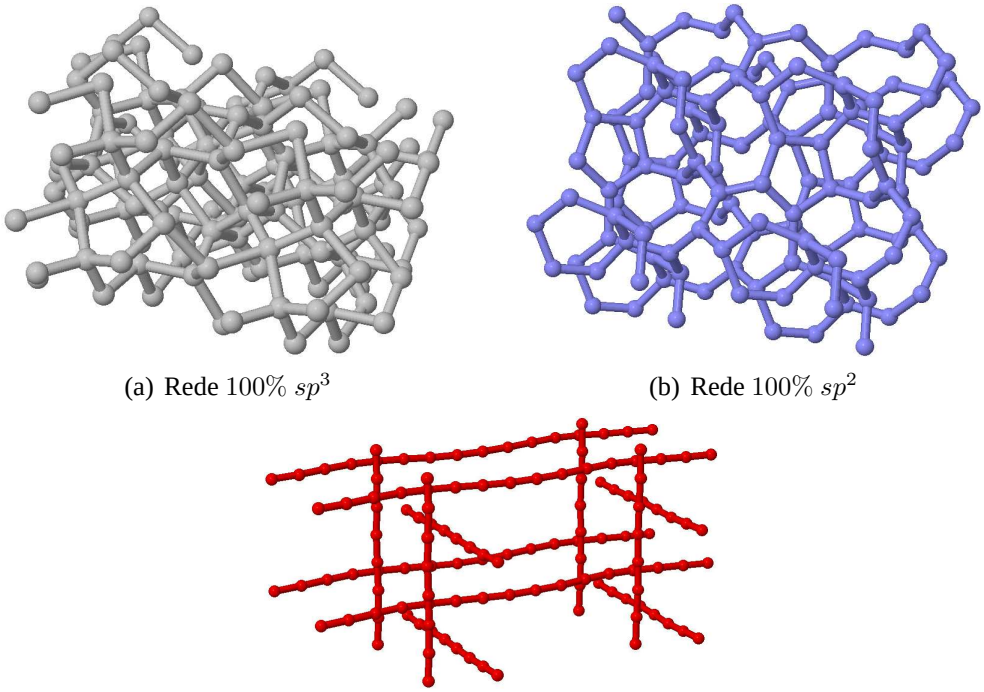
Etapa	Fração de Tempo	Temp. Inicial	Temp. Final
1	5%	10^4	4
2	88%	4	$0,5$
3	7%	$0,5$	10^{-4}

O número de iterações total do algoritmo é o fator mais complicado de ser definido. Para garantir a convergência o número de iterações deveria ser próximo ao infinito, porém, como esse valor é inviável, deve-se definir um valor suficientemente grande. Esse valor deve ser exponencialmente proporcional ao número de variáveis envolvidas (átomos, no problema em questão).

3.6 Resultados obtidos

Uma vez criado o programa, foram feitas simulações para validar seu funcionamento. Primeiramente as redes testadas tinham configurações 100% sp^3 (diamante), 100% sp^2 (grafeno - um plano do cristal de grafite) e 100% sp para poder verificar a eficiência do programa em gerar estruturas estáveis. Foi definido que cada rede possuiria 16 átomos e que o número de passos executados seria igual a 10^6. A partir dessas execuções foram geradas as redes representadas na figura 3.9 (Para facilitar a visualização, uma vez que

são redes contínuas, as estruturas foram replicadas nos seu eixos). O número de passos definido foi suficiente para que as redes fossem geradas em menos de dois minutos. O sucesso na geração dessas redes mostrou que o método desenvolvido é flexível para criar estruturas distintas (com quaisquer concentrações de carbono sp^3, sp^2 e sp) (JORNADA, 2007).



(a) Rede 100% sp^3 (b) Rede 100% sp^2

(c) Rede 100% sp

Figura 3.9: CRNs simuladas com 16 átomos

Novas redes (agora mistas) foram geradas com um universo de 128 átomos. Para tais, o programa também se mostrou eficiente em convergir de forma satisfatória na configuração especificada, porém, foi necessário um número muito maior de passos, fazendo com que a simulação durasse aproximadamente uma semana. Esse tempo, embora um pouco grande, é aceitável visto o número de possibilidades de combinações existentes com esse número de átomos.

Uma vez validado o funcionamento do programa, partiu-se para a geração das redes maiores (1024 e 2048 átomos) as quais poderiam atender o propósito esperado. A geração de cada uma dessas redes desprendeu cerca de um mês e meio de processamento. Defronte desta demora e da necessidade de que mais de 100 redes fossem geradas para os primeiros testes, partiu-se para a estratégia de computação em grade. A grade utilizada foi a GridUCS, uma grade institucional da Universidade de Caxias do Sul formada atualmente por 241 computadores não dedicados, isto é, computadores que são utilizados pela grade somente quando estão ociosos. Com o uso da estrutura de grade foi possível gerar o conjunto total de redes em um tempo muito menor em comparação com a execução em uma única máquina. Porém, como essas redes eram geradas através de uma estratégia de paralelização *bag-of-task*, um grande problema persistia: quando uma rede com características específicas precisava ser gerada, o tempo para obtê-la continuava igual ao tempo serial (quando não mais).

Para tentar resolver esse problema, foi proposta a paralelização da geração de cada uma dessas redes. Essa paralelização será apresentada em detalhes no capítulo 4.

3.7 Considerações finais

Nesse capítulo foi apresentado o trabalho base para a paralelização apresentada no capítulo 4. Foi apresentada sua descrição e o pseudo-código da versão seqüencial do seu algoritmo.

A partir do próximo capítulo serão apresentadas as contribuições desse trabalho. Serão apresentadas as estratégias de paralelização do algoritmo de geração de redes aleatórias contínuas (capítulo 4), a verificação numérica, análise teórica e experimental desse algoritmo (capítulo 5) e as conclusões obtidas (capítulo 6).

II

Contribuições

4 PARALELIZAÇÃO DO ALGORITMO DE GERAÇÃO DE REDES ALEATÓRIAS CONTÍNUAS POR *SIMULATED ANNEALING*

Com o intuito de contornar o problema de tempo apresentado no capítulo anterior foi proposta a paralelização do algoritmo de geração das CRN. Essa proposta será apresentada nesse capítulo que está dividido da seguinte forma: na seção 4.1 é apresentada a metodologia para desenvolvimento da versão paralela; na seção 4.2 é apresentada a forma como os dados foram distribuídos entre os processadores; na seção 4.3 são apresentados detalhes sobre a forma de sincronização dos dados; na seção 4.4 é apresentada a implementação feita para este trabalho; na seção 4.5 é apresentada a fórmula para o cálculo do número máximo de processadores que podem ser utilizados com cada configuração; na seção 4.6 é apresentada a generalização do método proposto a fim de ser aplicado a outros problemas baseados em SA e em especial é apresentada a utilização deste no problema do caixeiro viajante. Por fim, na seção 4.7 são apresentadas considerações finais sobre o capítulo.

4.1 Metodologia

O primeiro passo no desenvolvimento da versão paralela do algoritmo concentrou-se no teste de viabilidade de uso da biblioteca *parSA-Library*. Para tal, baixou-se a versão disponível no site indicado nas publicações (KLIEWER; TSCHöKE, 1999).

Uma vez carregada a versão, tentou-se compilar a aplicação teste disponível junto a biblioteca. Entretanto, pode-se notar que a portabilidade expressa nos artigos não estava totalmente funcional. Algumas das instruções presentes no código eram exclusivas para a plataforma *Solaris*. Após realizado um estudo, foi possível substituir essas instruções por outras compatíveis com a plataforma presente nos *clusters* do instituto.

Após compilar a biblioteca, partiu-se para a adaptação do código seqüencial do algoritmo de geração de CRN para o uso da biblioteca. Nesse passo, viu-se que com o uso dessa biblioteca não seria possível implementar diversos recursos presentes na versão seqüencial do algoritmo, tais como mudança de tamanho do universo, resfriamento em várias etapas, entre outros. Por esse motivo, concluiu-se que a utilização da biblioteca não seria possível.

O segundo passo foi analisar as diversas propostas de paralelizações sugeridas pela literatura. Para ajudar na escolha, foi verificada qual seria a plataforma alvo para a execução. Feito isso, chegou-se à conclusão de que os ambientes mais prováveis seriam *clusters*, com no máximo dois processadores cada nó. Visto isso, eliminaram-se as propostas que se utilizava de uma única cadeia de Markov por essas serem mais adequadas

para ambientes de memória compartilhada.

Dentre as propostas que utilizavam múltiplas cadeias de Markov, a abordagem não iterativa foi descartada, pois ela possibilitaria apenas que mais cadeias fossem exploradas ao mesmo tempo. Porém, não diminuiria o tempo de execução (que era o objetivo almejado). O uso de computação especulativa também foi descartado devido ao fato de ser possível apenas um pequeno grau de paralelismo e devido à necessidade de grande número de comunicações.

Sendo assim, sobraram três abordagens: assíncrona, síncrona interativa e síncrona híbrida. Destas, foram feitos alguns teste de comportamento de convergência com a abordagem síncrona híbrida (utilizando-se de uma simplificação dos algoritmos genéticos). Esses testes mostraram que, devido ao grande número de variáveis envolvidas e devido à grande dependência de uma configuração global (custo de hibridização) para obtenção da função custo do problema em questão, esse método não seria apropriado. Essa grande dependência da hibridização no cálculo da função custo fez com que a abordagem assíncrona também fosse descartada.

Deste modo, a abordagem de Múltiplas Cadeias de Markov Síncrona mostrou ser a mais apropriada para o problema em questão. Uma vez selecionada a abordagem, partiu-se para a adaptação do algoritmo a ela. Nessa etapa verificou-se que, como o domínio da simulação era conhecido, poder-se-ia explorar um maior paralelismo através do uso conjunto da abordagem selecionada com a de decomposição de domínio (AZENCOTT, 1992).

4.2 Distribuição dos Dados

Para utilizar essa técnica, aproveitou-se do conceito de células citado na seção 3.2, em que o universo onde os átomos estão inseridos é dividido em unidades menores. Na hora da paralelização do problema, cada processador fica responsável por certa região do espaço, isso é, fica responsável por executar suas iterações de SA sobre um conjunto de átomos inseridos nas células sobre sua responsabilidade. Na figura 4.1 (a) pode-se observar uma representação dessa divisão do espaço onde um universo de $10 \times 10 \times 10$ células é dividido entre 8 processadores.

Com essa divisão do espaço, é possível que todos os processadores envolvidos executem iterações de SA independentemente. Porém, essa divisão gera alguns inconvenientes. Um deles diz respeito à mudança de posição dos átomos localizados próximo a fronteira entre os processadores. Esse tipo de mudança interfere imediatamente no processamento dos átomos localizados nas células sob responsabilidade de outros processadores. Para evitar esse problema, optou-se por criar uma "região neutra" nas fronteiras da região de cada processador. Isso é, cada processador pode apenas modificar as posições dos átomos localizados nas células do "interior" de sua região. Uma representação dessas "regiões neutras" pode ser vista na figura 4.1 (b).

Com o uso de regiões neutras, pôde-se garantir que movimentos de átomos próximos as fronteiras não interferissem em outros domínios, entretanto essa restrição não possibilitava a simulação de movimentos dos átomos localizados nessas regiões. Para resolver esse problema, a cada sincronização dos dados entre os processadores, essa região é deslocada nos eixos x, y e z. A Figura 4.2 ilustra o funcionamento desse esquema, em que se observa que a cada passo as regiões neutras mudam de lugar, fazendo com que todos os átomos tenham direito a se movimentar.

Uma dúvida proveniente do uso dessa técnica é se a distribuição das probabilidades

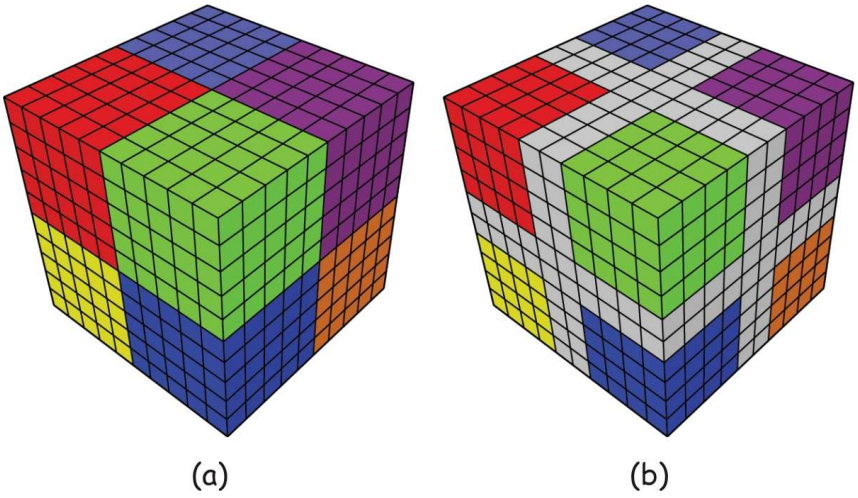


Figura 4.1: Distribuição da região do espaço entre os processadores

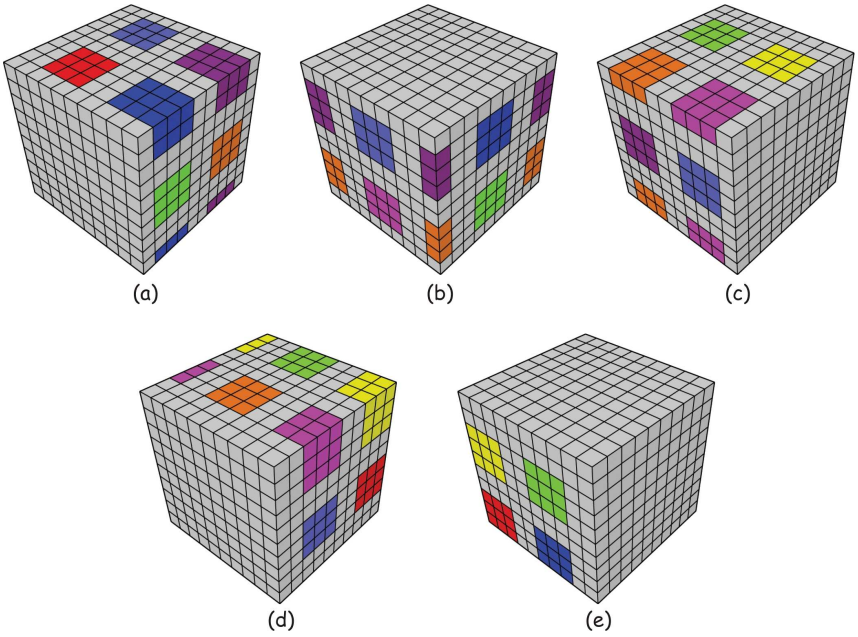


Figura 4.2: Funcionamento do deslocamento da área de responsabilidade de cada processo

de escolha entre cada átomo continua uniforme. Para verificar isso, realizou-se um estudo teórico sob a probabilidade de cada escolha.

Primeiramente verificou-se a probabilidade de escolha de cada elemento em uma distribuição sem uso de regiões neutras. Nesse tipo de distribuição, a probabilidade de cada valor ser escolhido é inversamente proporcional ao número de valores possíveis. Por exemplo, em um caso em que se possa escolher entre 10 valores distintos, a probabilidade de cada valor ser escolhido é de $\frac{1}{10}$. Sendo assim, pode-se definir a probabilidade de cada valor ser escolhido como:

$$P_{(i)} = \frac{1}{v}, \tag{4.1}$$

em que v é o número de possíveis valores a serem escolhidos.

Considerando-se que sejam feitas E escolhas, o número de escolhas prováveis de um

determinado valor é de:

$$P_{(i)} = \frac{1}{v} \times E \tag{4.2}$$

ou

$$P_{(i)} = \frac{E}{v} \tag{4.3}$$

Para calcular a probabilidade de escolha de cada valor em um sistema com regiões neutras e deslocamento, considerou-se que as E escolhas citadas acima seriam divididas em v etapas. Portanto, em cada uma dessas etapas seriam realizadas $\frac{E}{v}$ escolhas. Além disso, em cada etapa um dos valores estaria na região neutra, isso é, esse valor seria impossibilitado de ser escolhido. Sendo assim, cada um dos outros $v - 1$ valores teriam a probabilidade de escolha igual a $\frac{1}{v-1}$ e o valor na região neutra teria a probabilidade de escolha igual a zero.

Considerando-se, então, que cada valor tem a probabilidade de escolha $\frac{1}{v-1}$ em $v - 1$ etapas e de 0 em outra etapa, e que a cada etapa $\frac{E}{v}$ escolhas são efetuadas, ao término das v etapas a quantidade de escolhas de cada valor pode ser dado por:

$$P_{(i)} = (v - 1) \times \frac{1}{v - 1} \times \frac{E}{v} \tag{4.4}$$

ou simplificando

$$P_{(i)} = \frac{E}{v} \tag{4.5}$$

Portanto, como a equação 4.5 é igual a 4.3 pode-se garantir que a probabilidade de escolha de determinado valor na versão com regiões neutras e deslocamento é a mesma que na versão sem regiões neutras e deslocamento para toda simulação cujo número de etapas é divisível pelo número de valores. Quando esse número não é divisível, um pequeno resíduo restará para alguns valores, porém, como geralmente o número de etapas é muito maior que o número de valores esse resíduo será muito pequeno, não interferindo o balanceamento.

Para ilustrar a análise teórica foi realizado um teste prático. Esse teste consistiu em fazer 10000 escolhas aleatórias entre 10 valores possíveis, representando o sistema sem regiões neutras. O gráfico 4.3 mostra o histograma dessas escolhas. Pode-se observar que a distribuição é bastante uniforme, ficando todos os valores próximos a 1000.

Posteriormente foi realizado um teste para simular as regiões neutras. Para tal, foram feitas 5 simulações com 2000 escolhas cada uma. Em cada escolha 2 valores foram tornados indisponíveis, isso é, as escolhas aleatórias ficaram restritas apenas a 8 valores. Na figura 4.4 são apresentadas as distribuições das escolhas nas 5 simulações. Nota-se que as escolhas estão distribuídas de forma uniformes nos 8 valores disponíveis e os outros 2 valores, referentes a região neutras, estão zerados.

Por fim, somou-se as escolhas obtidas em cada uma das 5 simulações, obtendo-se os valores representados na figura 4.5. Nela pode-se observar que a soma das simulações gera um histograma uniforme, com valores concentrados próximo ao valor 1000. Esses dados foram representados juntamente com os dados obtidos na versão sem regiões neutras através de um gráfico quantile-quantile[1] (JAIN, 1991). Esse gráfico pode ser visto na figura 4.6. Observa-se que os pontos do gráfico estão praticamente alinhados com a

[1]Em estatística o quantile-quantile é um método para diagnosticar graficamente as diferenças entre duas

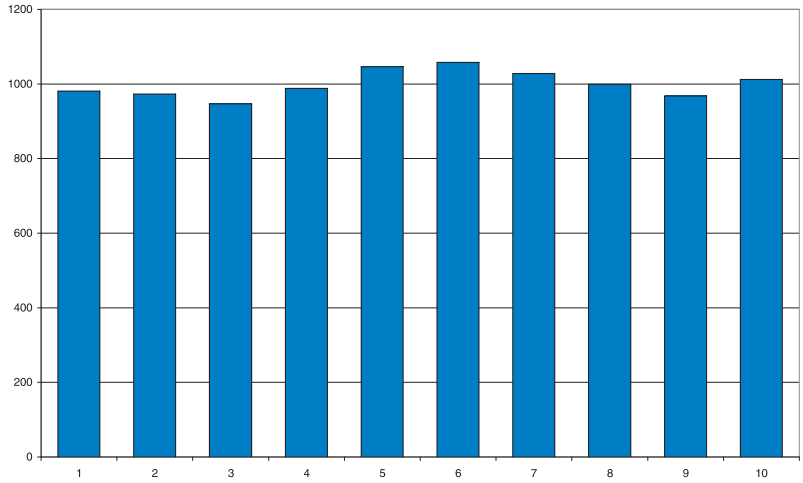


Figura 4.3: Histogramas das escolhas em um sistema sem regiões neutras

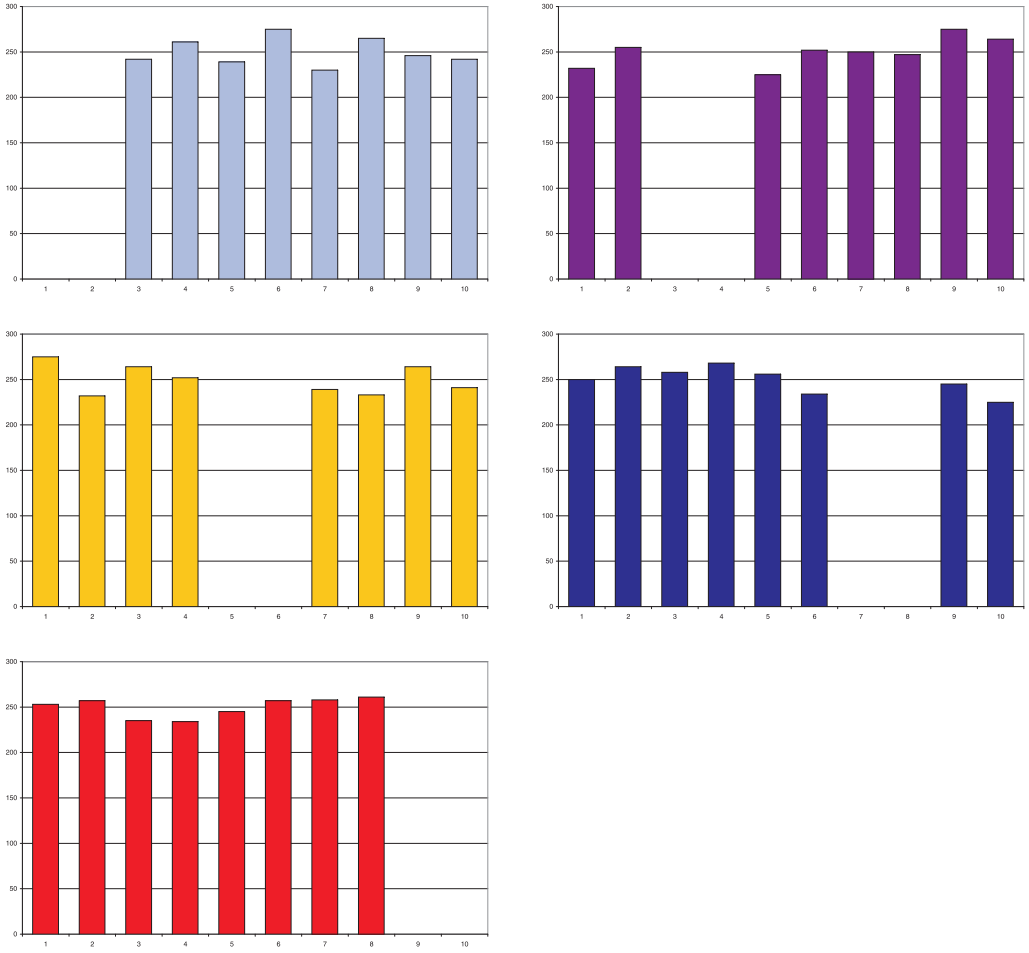


Figura 4.4: Histogramas das escolhas em um sistema com regiões neutras (as diferentes cores referem-se a cada uma das etapas mostradas na figura 4.5).

reta que forma 45^o com os eixos e que seu coeficiente angular é de $1,009$, mostrando assim, que as duas distribuições são equivalentes. Sendo assim, com o teste prático pode-se

distribuição probabilísticas. Quando as duas distribuições são equivalentes, o gráfico será representado por uma reta.

ilustrar o que a análise teórica previa, o sistema com regiões neutras juntamente com o deslocamento permite que os valores sejam escolhidos com a mesma probabilidade do que com o sistema sem regiões neutras.

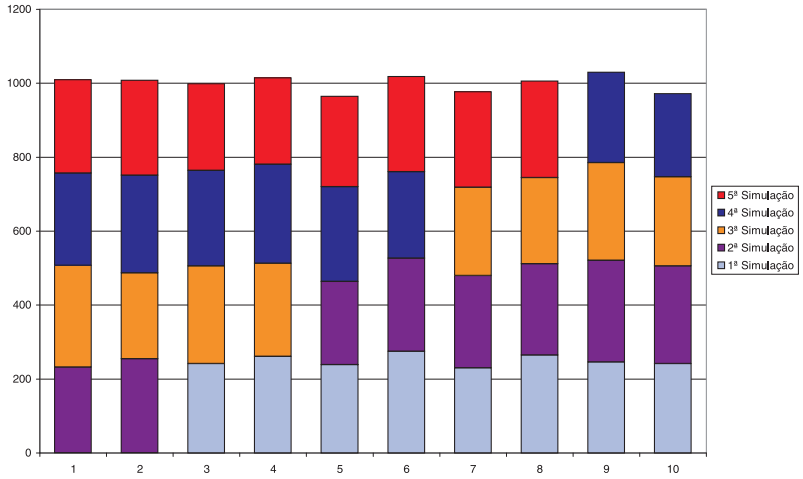


Figura 4.5: Histogramas da soma das escolhas com regiões neutras nas diversas etapas

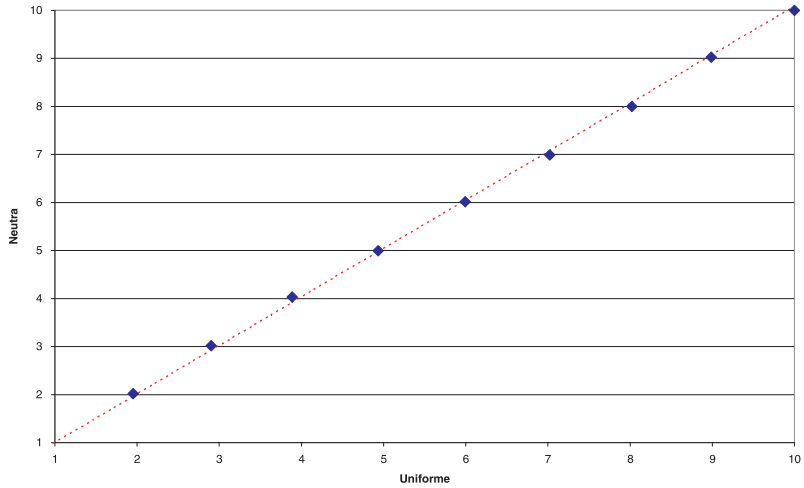


Figura 4.6: Gráfico quantile-quantile entre os resultados da versão sem regiões neutras (uniforme) e com regiões neutras

4.3 Sincronização dos Dados

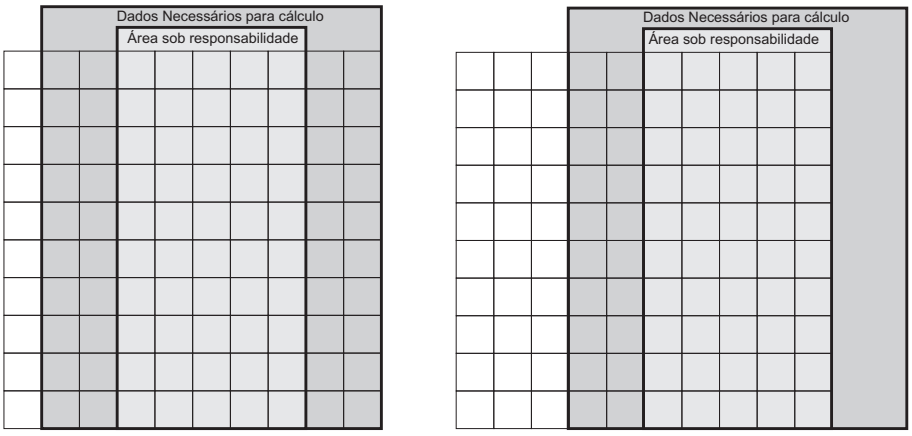
A primeira abordagem para a sincronização das posições dos átomos entre os processadores foi feita de maneira centralizada. Isso é, a cada conjunto de passos, todos os processadores enviavam a posição dos átomos sobre sua responsabilidade para um processador central. Esse, por sua vez, era responsável pela união de todo o domínio e redistribuía esse domínio para os demais processadores. Devido ao fato de que, para se garantir a convergência dos dados, eram necessárias comunicações com grande freqüência, essas comunicações tornaram-se custosas, gerando uma eficiência paralela inferior à desejada.

Essa menor eficiência tinha dois fatores principais. O primeiro deles era a comunicação em si. Uma vez que existia um processador centralizado, conforme o número de

processos ia aumentando, maior era esse custo. Outro fator impactante dizia respeito à necessidade de refazer a hibridização dos átomos de todo o sistema. Esse processo tornava-se muito custoso à medida que se aumentava o número de átomos.

Para melhorar o desempenho sem perder qualidade numérica foi necessário alterar o funcionamento das sincronizações. Para tal, criaram-se dois níveis delas. Um deles (o que ocorre com menos freqüência) é o mesmo existente na versão anterior, entretanto ele passou a ocorrer em espaços de tempos maiores. Na freqüência em que este ocorria anteriormente foi introduzida uma nova sincronização. Essa, em vez de ser centralizada em um único processador ocorre de forma distribuída. Nela, cada processador envia para os processadores vizinhos o conjunto de átomos localizados na fronteira entre eles. Deste modo, não se tem mais um processador centralizado, nem mesmo a necessidade de hibridização de todos os átomos do sistema, basta realizar essa tarefa com os recebidos.

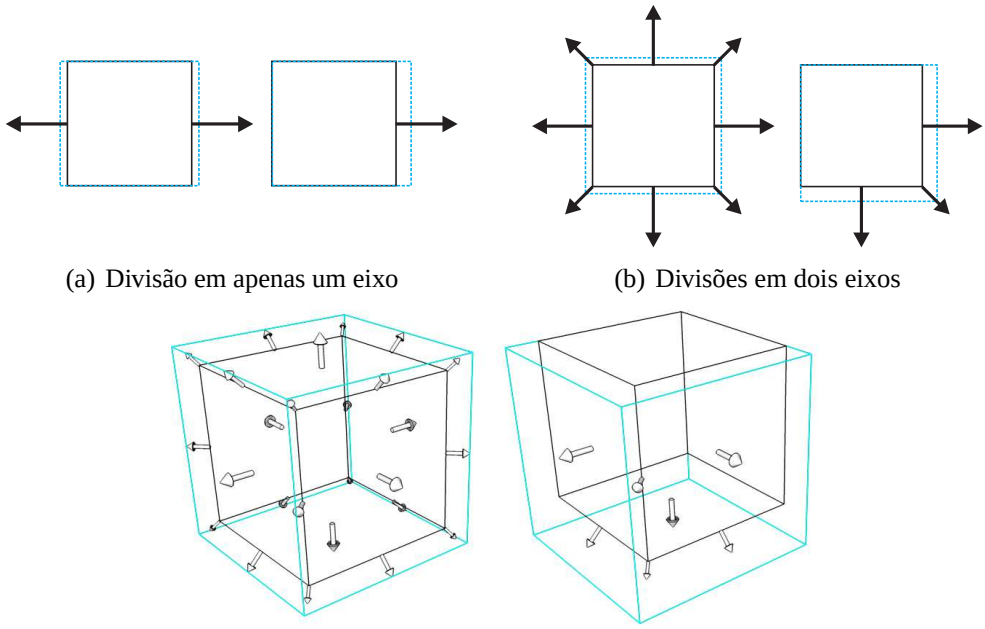
Uma vez introduzida essa nova sincronização, pode-se observar que o sistema de deslocamento citado anteriormente além de evitar que átomos posicionados nas regiões neutras não sejam explorados, permite que o número de mensagens trocadas entre os processos seja muito menor. Isso se deve ao fato de que as mensagens precisam ser enviadas apenas para uma direção, visto que os dados da fronteira da outra direção já pertenciam ao mesmo processo. Na figura 4.7 está ilustrado um exemplo de um corte bidimensional de um sistema que demonstra porque os dados precisam ser enviados apenas para uma direção. Nele observa-se a área sobre responsabilidade e a área de cálculo antes e depois do deslocamento, e, pode-se notar que à esquerda, não houve a necessidade de dados provenientes de outros processadores. Sendo assim, para o funcionamento do sistema, basta que os dados do processador posicionado a direita sejam enviados. Cabe lembrar também que, esse sistema de deslocamento torna-se eficiente para esse problema, devido ao fato do acesso aos dados ser cíclico, isso é, os átomos posicionados nas últimas colunas do sistema fazem ligações com os átomos localizados nas primeiras colunas como se estas estivessem diretamente ligadas entre si.



(a) Antes do Deslocamento (b) Depois do Deslocamento

Figura 4.7: Áreas necessárias para o cálculo antes e depois do deslocamento

Na figura 4.8 estão ilustradas as comunicações necessárias para troca de informações num sistema convencional e num sistema com deslocamento. Nota-se que, para casos em que existe divisão em apenas um eixo 4.8(a), o número de mensagens cai de 2 para 1; no caso de divisões em dois eixos 4.8(b), de 8 para 3; e no caso de divisões nos três eixos 4.8(c), de 26 para apenas 7.



(a) Divisão em apenas um eixo

(b) Divisões em dois eixos

(c) Divisões nos três eixos

Figura 4.8: Mensagens trocadas no sistema convencional e no com deslocamento

Nessa etapa de sincronização, além de trocar os dados com os processadores vizinhos, também é feita a atualização da energia global do sistema e de seu custo de hibridização. Para isso, cada processador calcula os valores da sua área de atuação e através do comando `MPI_Allreduce` o somatório de todos os valores parciais são enviados a todos os processadores. Assim sendo, os processadores tem uma visão atualizada da posição dos átomos de sua área de atuação e o valor global da energia e hibridização do sistema.

A sincronização centralizada, embora menos eficiente, não foi retirada do sistema devido à necessidade de execução da rotina de mudança do tamanho do universo. Por essa função necessitar obrigatoriamente do conhecimento de todos os átomos do sistema, tornava-se inviável realizá-la em paralelo. Portanto, aproveitou-se essa sincronização para executá-la. Além disso, essa sincronização torna-se útil, pois, a partir dela todos os processadores compartilham do mesmo estado global e podem seguir suas execuções com um estado mais atualizado.

Cabe salientar que, embora a estrutura de cada átomo seja formada por vários campos, optou-se por fazer o envio apenas de sua posição no espaço (coordenadas x, y e z). Isso se deve ao fato de que mesmo que as demais informações fossem enviadas, as mesmas precisariam ser atualizadas no novo processador. Sendo assim, enviando-se apenas as posições, o custo do envio dos dados ficou menor e o tempo de processamento das informações não enviadas permaneceu igual.

4.4 Implementação

A paralelização foi implementada em linguagem C com o uso de biblioteca de troca de mensagens MPI. Na Figura 4.9 está representado um pseudocódigo dessa implementação. Nele vê-se que um processo principal realiza a leitura dos parâmetros (linha 2) e o posicionamento dos átomos (linha 3). No arquivo de parâmetros, além das informações contidas no programa seqüencial, foram introduzidas informações referentes ao número

```
1: if rank==0 then
2:     le_parametros();{lê parâmetros de entrada do sistema}
3:     posiciona_atomos();{posiciona os átomos}
4: end if
5: for proc = 0 to NR_PROC − 1 do
6:     temperatura = temp[proc];{inicializa a temperatura}
7:     Max_iters = (Max_Steps[proc]/size/Pas_Independ);
8:     for iter = 0 to Max_iters do
9:         if iter%RESIZE_UNI_EACH == 0 then
10:            recolhe_dados(); {processadores enviam dados processo 0}
11:            muda_tamanho_universo();{muda tamanho do universo}
12:            distribui_dados();{processo 0 distribui posição dos átomos para demais processos}
13:            do_cells();{coloca cada átomo em sua célula}
14:            hibri_all();{refaz a hibridização dos átomos}
15:        else
16:            troca_dados();{troca posição com vizinhos}
17:            hibri_novos();{hibridiza átomos recebidos}
18:            desloca_area();{desloca a área de atuação}
19:            sincroniza_energia();{Sincroniza a energia do sistema}
20:            sincroniza_hibri();{Sincroniza a hibridização do sistema}
21:        end if
22:        tote();{calcula a energia total do sistema}
23:        for passo = 0 to Passos_Independ do
24:            at = atomos[rand(NATM)];{pega um átomo aleatoriamente}
25:            E_local = local_energy(at);{calcula a energia local}
26:            Hibri = hibri_cost(at);{calcula o custo de hibridização}
27:            at→x = at→x + randf()*MAX_DESLOC;{desloca o átomo na direção x}
28:            at→y = at→y + randf()*MAX_DESLOC;{desloca o átomo na direção y}
29:            at→z = at→z + randf()*MAX_DESLOC;{desloca o átomo na direção z}
30:            atm_hibri(at);{re-hibridiza o átomo atual}
31:            E_local_new = local_energy(at);{calcula a energia local}
32:            Hibri_new = hibri_cost(at);{calcula o custo de hibridização}
33:            dE = E_new - E_old;{calcula a variação de energia}
34:            dH = Hibri_new - Hibri_old;{calcula a variação do custo de hibridização}
35:            dfc = (1.0f - COST_HIBRI)*(dE) + COST_HIBRI*(dH); {calcula a variação da FC}
36:            if aceita(dfc) then
37:                atualiza_posicao();{atualiza a posição do átomo}
38:            else
39:                desfaz_movimento();{volta o átomo posição anterior}
40:            end if
41:            temperatura = temperatura * λ; {diminui a temperatura do sistema}
42:        end for
43:    end for
44: end for
45: recolhe_dados(); {processadores enviam dados processo 0}
46: if rank==0 then
47:     gera_saida(); {gera arquivos de saída}
48: end if
```

Figura 4.9: Algoritmo paralelo para geração de CRN

de divisões em cada eixo e o intervalo entre cada sincronização parcial (um exemplo do arquivo de entrada pode ser visto na figura 4.10). Após a leitura e posicionamento dos átomos, cada processo inicia um conjunto de etapas de resfriamento do sistema (linha 5). Define-se a temperatura inicial da etapa (linha 6) e calcula-se o número de iterações executadas por cada processo (linha 7). O número de iterações é dado em função da quantidade de passos informados no arquivo de entrada, dos processos envolvidos e o dos passos independentes do sistema.

O passo seguinte consiste em um laço que controla as iterações do sistema (linha 8 a 43). A cada iteração do laço verifica-se se a iteração atual é em uma iteração de mudança do tamanho do universo (linha 9). Caso seja, os dados são recolhidos para o processo de *rank* 0 (linha 10), esse executa o procedimento de mudança (linha 11) e envia os dados de volta aos demais processos (linha 12). Após recebidos os dados, os processos colocam cada átomo em uma célula (linha 13) e fazem a hibridização dos mesmos (linha 14). Caso não seja uma iteração de mudança do tamanho do universo, os processos trocam as posições dos átomos apenas com os processos vizinhos (linha 16). Em seguida, realiza-se a hibridização dos novos átomos recebidos (linha 17), desloca-se a área de atuação (linha 18) e calcula-se a energia total do sistema (linha 19) e o custo de hibridização global (linha 20). A partir desse ponto, cada processo faz um conjunto de passos independentes de SA (linhas 23 a 42), selecionando os átomos de suas respectivas regiões. Após isso, volta-se ao início do laço onde novos passos são executados. Ao fim da execução, todos processadores enviam as posições dos seus átomos para o processador central (linha 45) que os une e gera os arquivos de saída (linha 47).

```
<main>
title=Simulação teste paralela  # Título da simulação

<system>
atoms=1024        # Número de átomos
cost_hibri=0.30   # Parte do custo final referente a hibridização
prop_sp3=0.8      # Proporção de átomos sp3
prop_sp2=0.2      # Proporção de átomos sp2

DIV_X=2           # Divisões no eixo X
DIV_Y=2           # Divisões no eixo Y
DIV_Z=2           # Divisões no eixo Z

troca_atm=100     # Número de iterações entre cada sincronização parcial

<params>
resize=500        # Intervalo entre cada mudança do tamanho do universo

<annealing>
steps=1e8(0.05, 0.88, 0.07)  # Número de passos de cada etapa
temp=(1e4, 4, 0.5, 1e-3)     # Temperatura inicial e final de cada etapa
```

Figura 4.10: Exemplo do arquivo de entrada para o sistema paralelo

4.5 Número de processadores em função do número de átomos

Como foi citado anteriormente, o método proposto de paralelização trabalha com a divisão do domínio do sistema. Essa divisão gera restrições quanto ao número máximo

de processadores que podem ser utilizados com certa quantidade de átomos. Isso deve-se ao fato de que, se o sistema a ser simulado possuir um número pequeno de variáveis e um número grande de processadores forem utilizados, cada processador ficará reponsavel por um pequeno número de variáveis a cada conjunto de iterações, podendo, assim, não conseguir realizar sua tarefa com eficiência.

Além dos dados serem divididos entre vários processadores, a existência de regiões neutras também diminui o número de variáveis que cada processador pode manipular. Dessa forma, ao escolher o número de processadores a ser utilizado, esses cuidados devem ser tomados.

Para o problema de geração de redes aleatórias contínuas de carbono, é interessante que o número de átomos que cada processador possa manipular seja pelo menos a metade do número de passos independentes que ele realiza. Pode-se obter o número de átomos manipuláveis por cada processador através da seguinte fórmula:

$$a = \frac{s - 2D_x}{D_x} \times \frac{s - 2D_y}{D_y} \times \frac{s - 2D_z}{D_z} \times \delta, \tag{4.6}$$

em que s é o número de células em cada dimensão, D_x é o número de divisões no eixo x, D_y é o número de divisões no eixo y, D_z é o número de divisões no eixo z e δ é a densidade do sistema.

s, por sua vez, pode ser definido como

$$s = \sqrt[3]{\frac{A}{\delta}}, \tag{4.7}$$

em que, A é o número de átomos presentes na simulação.

4.6 Generalização da técnica de paralelização

Embora a solução apresentada nesse capítulo tenha sido desenvolvida especificamente para a paralelização do algoritmo de geração das CRN ela pode ser facilmente generalizada a fim de ser aplicada na paralelização de outros problemas baseados em SA. Para que isso possa ser possível, é necessário que o problema tenha algumas características especiais. São elas:

- **Possibilidade de agrupamentos de variáveis**: para que se possa aplicar a divisão do domínio é necessário que as variáveis do problema possam ser divididas em subconjuntos, que serão distribuídos entre os processadores. É interessante, também, que seja possível descobrir quais variáveis possuem interação direta com as variáveis localizadas nos demais processadores.

- **Possibilidade de mudanças de curto alcance**: é de suma importância que o problema a ser paralelizado possibilite a realização de mudanças de curto alcance, isso é, mudanças que interfiram apenas (ou principalmente) na região local a cada processador. Essa restrição permite que as sincronizações entre os processadores sejam feitas com um intervalo maior, aumentando assim a eficiência da paralelização.

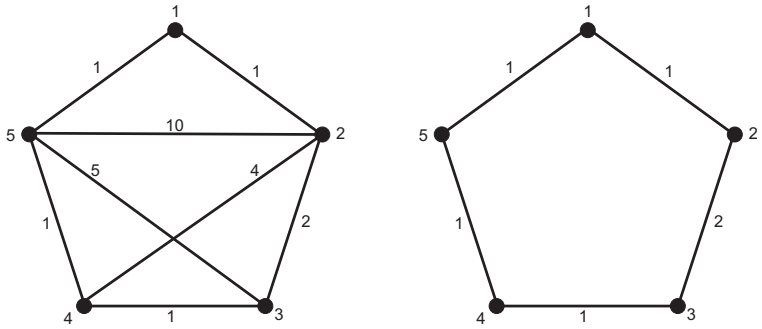
- **Possibilite acesso aos dados de forma cíclica**: essa característica permite que se possa aplicar o deslocamento da área de atuação de forma eficiente, possibilitando, assim, que variáveis presentes nas regiões neutras sejam exploradas.

Nas próximas subseções será descrito como seria possível adaptar a paralelização proposta nesse trabalho ao problema do Caixeiro Viajante. Na subseção 4.6.1 será apresentada a descrição do problema de Caixeiro Viajante, na subseção 4.6.2 será descrito o algoritmo seqüencial para o Caixeiro Viajante utilizando SA e por fim na subseção 4.6.3 será apresentada a proposta de paralelização utilizando a técnica proposta nesse trabalho.

4.6.1 Problema do Caixeiro Viajante

O problema do Caixeiro Viajante consiste em: dado um certo número de cidades e um conjunto de custos referentes ao deslocamento entre duas cidades, definir o melhor caminho possível (menor custo) que passe exatamente uma vez em cada cidade e retorne à origem (CAMPELLO; MACULAN, 1994).

Na figura 4.11(a) é apresentado um conjunto de cidades e os custos associados ao deslocamento entre elas. Pode-se observar que o melhor caminho que satisfaz o problema é o caminho $C = \{1, 2, 3, 4, 5, 1\}$ que tem um custo igual a 6 e está representado na figura 4.11(b).



(a) Caminhos disponíveis entre as cidades

(b) Caminho ótimo

Figura 4.11: Exemplo do funcionamento do problema do Caixeiro Viajante

4.6.2 Algoritmo seqüencial do Caixeiro Viajante utilizando SA

A implementação seqüencial do problema do Caixeiro Viajante utilizando SA pode ser descrita através do seguinte conjunto de passos:

- A partir das informações das ligações entre as cidades (figura 4.12(a)), gera-se uma configuração inicial através da escolha de um caminho aleatório que passe por todas as cidades (figura 4.12(b));
- Inicia-se um laço para controlar os passos do SA;
- A cada passo do laço seleciona-se duas ligações e faz-se a permutação dessa ligação, como representado na figuras 4.12(c) e 4.12(d);
- Uma vez feita a permutação, a função custo da nova configuração é calculada. Para esse problema, pode-se considerar a função custo como sendo o somatório dos custos do caminho percorrido.
- Após calcular a função custo, verifica-se se o novo caminho é aprovado pelo critério de Metropolis;
- A simulação deve ser executada até que algum critério de parada seja satisfeito.

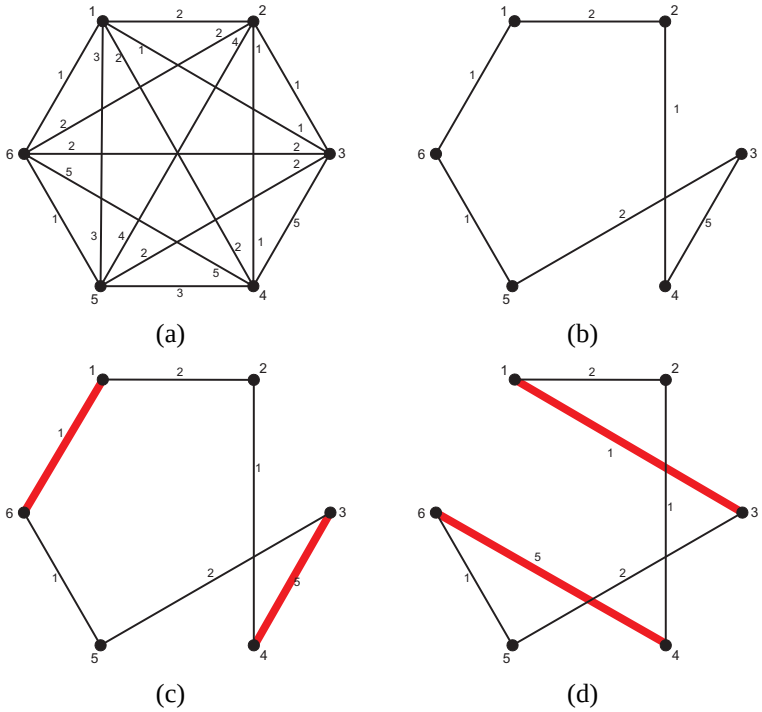


Figura 4.12: Caixeiro Viajante Seqüencial

4.6.3 Proposta de paralelização do problema do Caixeiro Viajante

Para aplicar a proposta de paralelização descrita nesse trabalho ao problema do caixeiro viajante alguns conceitos devem ser adaptados. São eles:

- Como no problema do caixeiro viajante não existe exatamente um domínio a ser dividido, pode-se mudar esse conceito através da divisão de um caminho. Isso é, um processador principal cria um caminho inicial aleatório que passe por todas as cidades. Esse caminho, por sua vez, é subdividido em N partes, e cada uma dessas partes é enviada para um processador que tratará essa parte como seu "domínio" (figura 4.13(a));

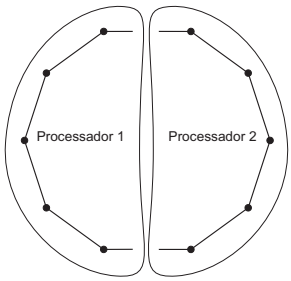
- As regiões neutras, para esse problema, são as cidades localizadas na fronteira entre dois processadores, isso é, cidades que fazem ligação com outras localizadas em processadores diferentes (figura 4.13(b)). Essas ligações não devem ser alteradas no processo paralelo;

- Cada processador faz as alterações no percurso e as avalia somente no subdomínio que a ele compete (mudança de curto alcance);

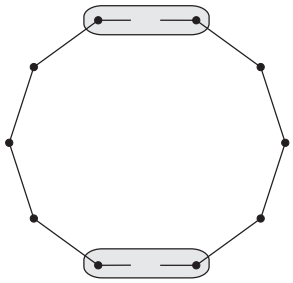
- As mensagens trocadas em cada sincronização intermediária dizem respeito às ligações entre as cidades próximas às fronteiras (figura 4.13(c));

- Uma vez realizada a sincronização intermediária, a área de cada processador deve ser deslocada (figura 4.13(d));

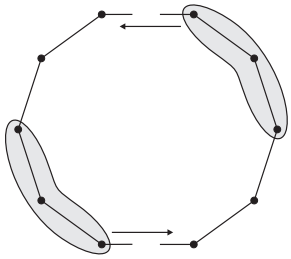
- No processo de sincronização global, por sua vez, cada processador envia seu percurso atual para todos os outros processadores. Dessa forma, todos processadores passam a conhecer todo o percurso, tendo assim, uma visão mais atualizada do problema.



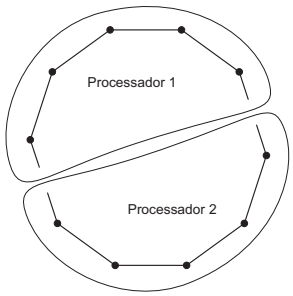
(a) Domínio dividido entre dois processadores



(b) Regiões neutras em destaque



(c) Dados enviados na sincronização parcial



(d) Nova área de atuação após deslocamento

Figura 4.13: Caixeiro Viajante Paralelo

Observa-se, portanto, que com pequenos ajustes, pode-se aplicar a técnica descrita nesse trabalho em outros problemas que utilizam SA e atendam às restrições descritas no início dessa seção.

Como restrição de paralelização para esse problema, pode-se considerar que o número de cidades manipulável por cada processador não deva ser menor que a metade do número de passos independentes. A fórmula para obtenção do número de cidades c manipulável por processador é:

$$c = \frac{C}{p} - 2, \tag{4.8}$$

em que, C é o número total de cidades e p o número de processadores.

4.7 Considerações finais

Nesse capítulo foi apresentada a paralelização do algoritmo de geração de redes aleatórias contínuas de carbono. A abordagem de paralelização adotada não se enquadra diretamente em nenhuma das citadas na seção 2.2.3, mas, pode ser considerada uma adaptação do método de Múltiplas Cadeias de Markov Síncronas Interativa. A diferença entre elas está relacionada à área de atuação, sendo que, na apresentada anteriormente todos os processos podem modificar todas as variáveis e, na adotada, a atuação de cada processo é limitada a certas variáveis.

Essa escolha possibilitou que um paralelismo maior fosse explorado (todos os movimentos feitos são efetivos e não apenas o do processador que se saiu melhor, como nos casos citados na literatura). Entretanto, para tal, foi necessária a criação de novos pontos de sincronização, os quais tornaram o sistema não tão linear quanto o esperado.

Também foi apresentada nesse capítulo a generalização da proposta de paralelização,

para que esta possa ser utilizada em outros problemas baseados em SA. Pôde-se observar, através do exemplo da paralelização do problema do Caixeiro Viajante, que a adaptação a novos problemas pode ser feita de forma simples, sendo apenas necessário a mudança de alguns conceitos.

No próximo capítulo será apresentada uma verificação dos resultados numéricos obtidos com a versão paralela, bem como uma análise teórica sobre a complexidade do algoritmo.

5 VERIFICAÇÃO NUMÉRICA, ANÁLISE TEÓRICA E EXPERIMENTAL

Nesse capítulo são apresentados dados obtidos através da utilização da versão paralela do algoritmo de geração de redes aleatórias contínuas. Esses dados são utilizados para comprovar a equivalência da qualidade numérica da solução paralela em relação à solução seqüencial. Também será apresentada a análise do custo computacional agregado com a paralelização. Através dessa análise pode-se verificar qual a escalabilidade máxima dessa solução, bem como a eficiência em cada configuração.

Esse capítulo está dividido da seguinte forma: na seção 5.1 são apresentados os dados que comprovam a qualidade numérica da versão paralela em relação à versão seqüencial; na seção 5.2 são definidas as fórmulas para obtenção do custo computacional existente nas execuções da simulações; na seção 5.3 são apresentadas as fórmulas do sobrecusto computacional proveniente da paralelização e os possíveis valores para as variáveis de controle; na seção 5.4 é realizada a análise das fórmulas e feita a verificação experimental de seu comportamento; por fim na seção 5.5 são feitas algumas considerações finais sobre o capítulo.

5.1 Verificação numérica

Para ser verificada a equivalência numérica dos resultados obtidos através do uso da versão paralela do algoritmo foi realizado um conjunto de simulações com essa versão. Essas simulações foram realizadas nos *clusters* do Departamento de Informática e Departamento de Física e Química da Universidade de Caxias do Sul (UCS) e nos *clusters* do Instituto de Informática da Universidade Federal do Rio Grande do Sul (UFRGS). Esses *clusters* têm as seguintes configurações:

- **Polentao:** *cluster* do Departamento de Informática da UCS, composto de 21 computadores, sendo 1 servidor e 20 nodos de processamento. A configuração de cada um dos computadores do *cluster* é a seguinte: Intel Pentium IV HT 2.8Ghz HT, 512Kb de memória cache, 1024Mb de memória RAM, 40Gb de disco rígido. Todos os computadores possuem Sistema Operacional Scientific Linux IV com versão do Kernel 2.6.9-42.0.10 e a interconexão dos mesmos é feita através de uma rede local do tipo Fast-ethernet.

- **Hal:** *cluster* do Departamento de Física e Química da UCS. Esse *cluster* é composto por 12 computadores sendo: 1 servidor Intel Pentium IV HT 3.0Ghz, 1024Kb memória cache, 1,5Gb de memória RAM, 40Gb de disco rígido, 10 nodos Intel Celeron 2.66Ghz, 256Kb de memória cache, 40Gb de disco rígido e 1 nodo Intel

Pentium IV 2.8Ghz Dual Processor, 1024Kb memória cache, 1,5Gb de memória RAM, 40Gb de disco rígido. Todos os computadores possuem de Sistema operacional Scientific Linux IV com Kernel 2.6.9-42.0.10 e a interconexão dos mesmos é feita através de uma rede local do tipo Fast-ethernet.

- **LabTeC:** *cluster* do Instituto de Informática da UFRGS. Esse *cluster* é composto por 20 nodos biprocessados Intel Pentium III 1.13Ghz, 1024Kb memória cache, 1,0Gb de memória RAM, sistema operacional Debian com Kernel 2.6 e a interconexão dos mesmos é feita através de uma rede local do tipo Fast-ethernet.

- **Corisco:** *cluster* do Instituto de Informática da UFRGS. Esse cluster é composto por 14 nodos biprocessados Intel Pentium III 1.2Ghz, 1024Kb memória cache, 512Mb de memória RAM, sistema operacional Debian com Kernel 2.6 e a interconexão dos mesmos é feita através de uma rede local do tipo Gigabit-ethernet.

O motivo da utilização de recursos da UCS para a validação numérica deu-se devido à parceria firmada entre os pesquisadores dessas duas instituições para a realização desse projeto.

Para essa etapa de verificação, fatores referentes à eficiência, tais como tempo de execução e *speedup*, não foram levados em conta devido à heterogeneidade dos recursos utilizados, ficando, essa etapa, concentrada apenas em comparar os resultados numéricos obtidos na versão paralela com os obtidos na versão seqüencial. Fatores de eficiência serão tratados nas seções 5.2 a 5.4 onde os testes foram executados em um ambiente mais homogêneo.

Sabe-se que, certas propriedades estatísticas das posições atômicas podem ser comparadas com o intuito de validar a versão paralelizada do algoritmo. Em particular, as distribuições angular e radial dos átomos fornecem importantes informações da microestrutura das redes, e podem apontar para uma eventual distorção criada pela paralelização.

Para comparar os resultados entre a versão seqüencial e a versão paralela do sistema, gerou-se, com a versão paralela, um conjunto de redes contendo 1024 e 2048 átomos. Essas redes foram distribuídas entre 1×10^8, 5×10^8 e 1×10^9 iterações, e foi definido que as redes geradas deveriam ser $100\%\ sp^3$. Em seguida, selecionou-se a rede com 2048 átomos gerada com 1×10^9 iterações, já que uma rede serial com 2048 átomos, gerada com 2×10^9 iterações, tinha sido gerada anteriormente. Apesar desta última estrutura ter sido obtida com o dobro do iterações, a única grandeza física das redes que deve mudar é a largura da distribuição angular, e não a forma das distribuições. Por isso, essa comparação é válida.

A distribuição angular mede a freqüência que um determinado ângulo é observado entre duas ligações adjacentes. Para o caso de redes $100\%\ sp^3$, é de se esperar que essa grandeza tenha uma forma sinodal em torno do ângulo de equilíbrio $109.4°$. A largura da distribuição mede a aleatoriedade da rede gerada e, consequentemente, a estabilidade do material criado. Se a estrutura fosse cristalina, como no caso do diamante, o único ângulo observado seria o de $109.4°$, e a distribuição seria apenas um pico nesse ângulo.

Já a distribuição radial informa o número de átomos observados ao redor de um átomo, para uma determinada distância. Nesse caso, para redes cristalinas, deve-se observar apenas uma seqüência de picos em posições bem definidas, já que, em materiais cristalinos, os átomos estão espaçados de forma regular uns dos outros. Para redes aleatórias, esses picos devem se alargar, a distâncias suficientemente grandes, se sobrepor completamente.

Já que o algoritmo utilizado para gerar as redes é estocástico, flutuações nos valores das distribuições radial e angular são inevitáveis. Para reduzir esse ruído, as distribuições foram submetidas a um filtro gaussiano.

Como se pode observar na figura 5.1, a distribuição angular obtida com o algoritmo paralelizado apresenta uma largura um pouco superior àquela do obtido em redes geradas de forma serial. Porém, esse fato deve-se basicamente ao número reduzido de iterações utilizadas para gerar a rede paralela. A forma da distribuição é a mesma nas duas situações, o que indica que a paralelização do algoritmo não favoreceu a uma determinada configuração de ângulos.

De forma semelhante, a figura 5.2 indica uma grande semelhança entre a rede serial e a paralela. A diferença principal entre elas parece ser originária de flutuações estatísticas de posições atômicas, e não de uma distorção na geometria dos átomos introduzida pela paralelização.

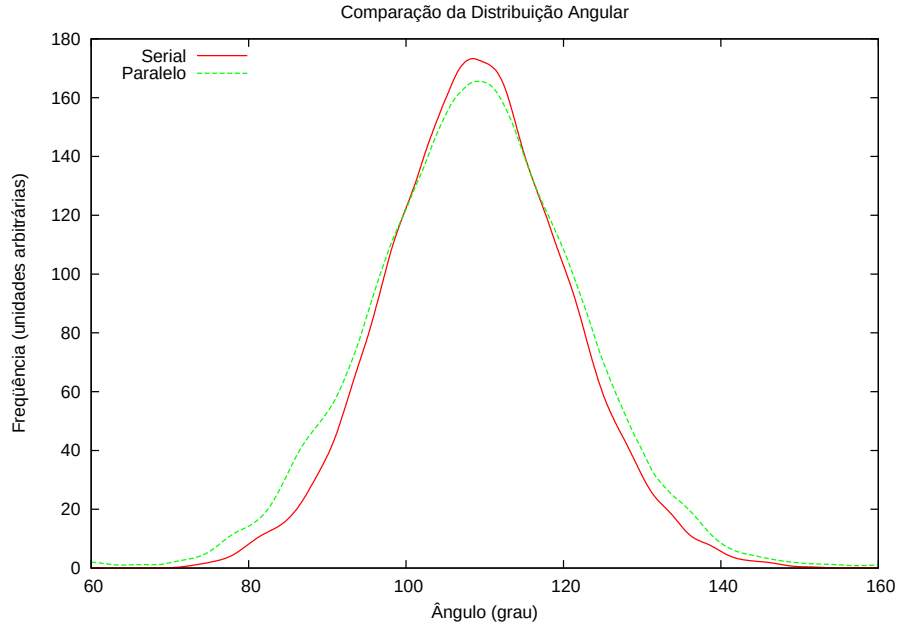


Figura 5.1: Comparação da distribuiçào angular entre a versão seqüencial e a versão paralela

Diante desses resultados, pode-se garantir que o comportamento encontrado na versão paralela do sistema é equivalente ao da versão seqüencial. Sendo assim, a versão paralela pode ser utilizada pelos físicos sem que ela introduza distorções observáveis ao sistema.

5.2 Custo computacional

Nessa seção e nas que seguem, será realizada a análise do custo computacional agregado com a paralelização do algoritmo de geração de CRN por SA. Para dar inicio à definição das fórmulas que descrevem esse custo, deve-se definir:

- p: número de processadores envolvidos na paralelização;
- n: número total de iterações;
- A: número de átomos pertencentes à simulação.

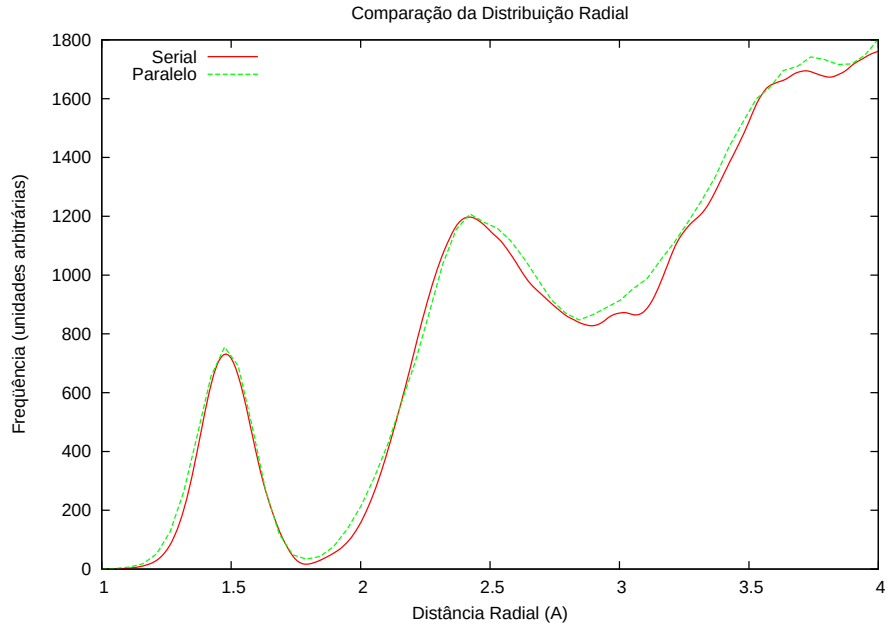


Figura 5.2: Comparação da distribuições radial entre a versão seqüencial e a versão paralela

A partir disso, o tempo total da simulação pode ser dado em função do somatório dos tempos das quatro fases do algoritmo. São elas:

- **Execução de cálculos**

  O tempo gasto por cada processador para executar as iterações que são de sua responsabilidade ($T_c(A, p, n)$) é dado em função do número de iterações que ele deve executar (n/p) e o tempo de cada iteração ($T_i(A)$).

  $$T_c(A, p, n) = (n/p).T_i(A),$$

  em que $T_i(A)$ é o tempo médio de uma iteração.

  $T_i(A)$, por sua vez, pode ser decomposto no somatório de:

  $$T_i(A) = v.\tau + v.h + \alpha.A,$$

  em que:

  - τ é o tempo de cálculo da energia de cada átomo;
  - h é o tempo necessário para verificar se um átomo está suficientemente próximo a outro a ponto de poderem se ligar;
  - α é o tempo necessário para verificar a hibridização de um átomo;
  - v é o número de átomos presentes nas células vizinhas ao átomo em questão.

  Considerando-se uma distribuição relativamente homogênea, uma densidade média de δ átomos por célula e que são consideradas como células vizinhas, as posicionadas até 2 células de distância nos eixos x, y, z, temos:

$$v \cong 5 \times 5 \times 5 \times \delta = 125.\delta.$$

Considerando-se que v é relativamente constante e independente de outras variáveis, podemos substituir, para efeito de simplificação, na fórmula $v.\tau$ por Γ e $v.h$ por H. Dessa forma, podemos descrever o tempo de cada iteração como sendo:

$$T_i(A) = \Gamma + H + \alpha.A.$$

- **Comunicação entre vizinhos**

  O tempo gasto em cada comunicação entre os processos vizinhos é dado em função do número de átomos e do número de divisões nos eixos x, y, z (D_x, D_y, D_z respectivamente).

  Para facilitar a explicação das fórmulas usadas para obtenção do número de átomos envolvidos nas comunicações, define-se o número de células em cada eixo (s) como:

$$s \cong \sqrt[3]{\frac{A}{\delta}},$$

  em que δ é a densidade média de átomos por células do sistema.

  Nos próximos itens serão mostrados o número de mensagens enviadas de acordo com o número de eixos divididos e o tamanho dessas mensagens. A figura 5.3 ajuda identificar qual região cada mensagem representa.

  – Em simulações em que exista divisão em apenas 1 dos eixos, será necessário o envio de apenas 1 mensagem (m) contendo:
  $s^2 \times 2.\delta$ átomos (figura (a)).
  ou

$$a = 2.\delta.s^2.$$

  – Em simulações em que exista divisão em 2 dos eixos, será necessário o envio de 3 mensagens (m), sendo que elas contêm, respectivamente:
  $\frac{s^2}{D_x} \times 2.\delta$ átomos (figura (b));
  $\frac{s^2}{D_y} \times 2.\delta$ átomos (figura (c));
  $s.4.\delta$ átomos (figura (d)).
  Simplificando:

$$a = 2.\delta.s.\left(\frac{s}{D_x} + \frac{s}{D_y} + 2\right)$$

  ou

$$a = 2.\delta.s.\left(s.\frac{D_y + D_x}{D_x.D_y} + 2\right).$$

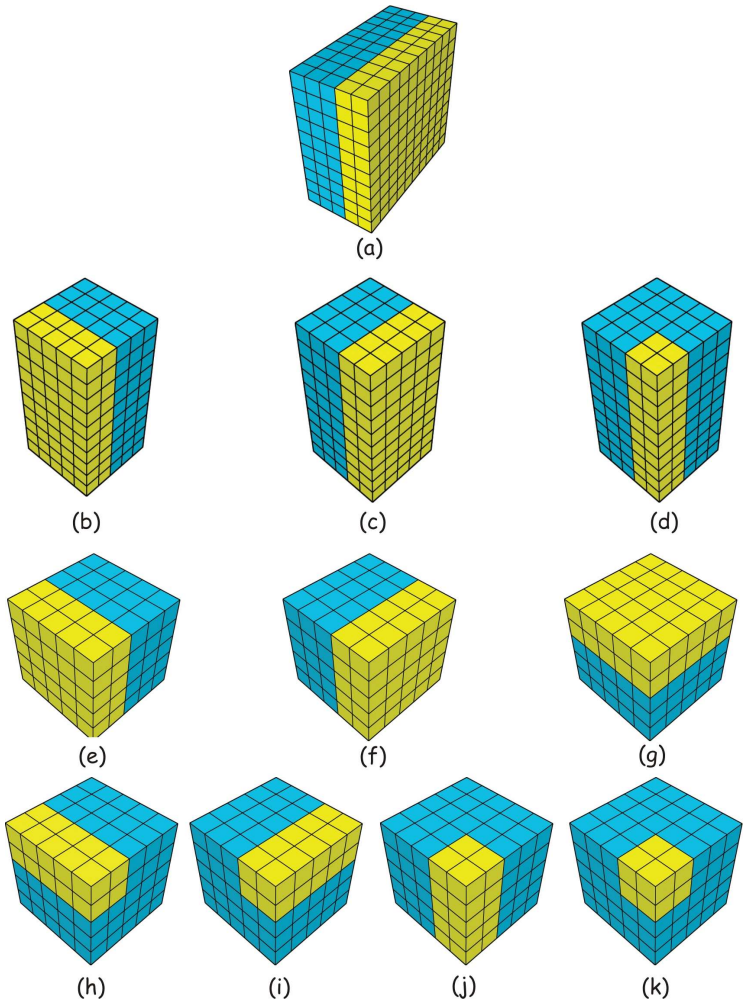


Figura 5.3: Representação dos átomos enviados em cada mensagem

Como o produto das divisões nos 2 eixos é igual ao número de processos envolvidos, podemos simplificar ainda para

$$a = 2.\delta.s.\left(s.\frac{D_y + D_x}{p} + 2\right).$$

– Em simulações em que exista divisão em 3 dos eixos, será necessário o envio de 7 mensagens (m), sendo que elas contêm, respectivamente:
$\frac{s}{D_x}.\frac{s}{D_z} \times 2.\delta$ átomos (figura (e));
$\frac{s}{D_y}.\frac{s}{D_z} \times 2.\delta$ átomos (figura (f));
$\frac{s}{D_x}.\frac{s}{D_y} \times 2.\delta$ átomos (figura (g));
$\frac{s}{D_x} \times 4.\delta$ átomos (figura (h));
$\frac{s}{D_y} \times 4.\delta$ átomos (figura (i));
$\frac{s}{D_z} \times 4.\delta$ átomos (figura (j));
$8.\delta$ átomos (figura (k));
Simplificando:

$$a = 2.\delta.\left((\frac{s}{D_x}.\frac{s}{D_y} + \frac{s}{D_x}.\frac{s}{D_z} + \frac{s}{D_y}.\frac{s}{D_z}) + 2.(\frac{s}{D_x} + \frac{s}{D_y} + \frac{s}{D_z}) + 4\right)$$

ou

$$a = 2.\delta.\left(s^2.(\frac{D_x + D_y + D_z}{D_x.D_y.D_z}) + 2.s.(\frac{D_x.D_y + D_y.D_z + D_x.D_z}{D_x.D_y.D_z}) + 4\right).$$

Como o produto das divisões nos 3 eixos é igual ao número de processos envolvidos, podemos simplificar ainda para

$$a = 2.\delta.\left(s^2.(\frac{D_x + D_y + D_z}{p}) + 2.s.(\frac{D_x.D_y + D_y.Dz + D_x.D_z}{p}) + 4\right).$$

Considerando-se que para transmitir cada átomo são comunicados 4 *floats* (1 para identificar o átomo que está sendo comunicado e 3 para representar sua posição no espaço x, y, z) e que, para cada mensagem enviada, tem-se um custo da latência (μ) e um custo da vazão (β)(floats/segundo), o custo de envio de cada mensagem pode ser obtido através da fórmula:

$$\mu + \frac{4.atomos}{\beta}.$$

Além do custo de envio das mensagens, existe o custo de hibridização dos átomos recebidos e o custo de colocação dos átomos em suas respectivas células. O primeiro é dado em função do somatório dos átomos recebidos em todas as mensagens (a) multiplicado pelo custo de hibridização de cada átomo (H). Já o segundo é dado em função do número total de átomos (A) multiplicado pelo custo de colocação de cada átomo em sua célula (c). Tem-se assim:

$$a.H + A.c.$$

Baseado nesses dados, a fórmula do tempo gasto em comunicação entre os vizinhos pode ser dado por:

$$T_c(A, a, m, n, p, k) = \frac{(n/p)}{k}\left(m.\mu + \frac{4.a}{\beta} + a.H + A.c\right),$$

em que k é o intervalo de iterações entre cada comunicação.

- **Sincronização parcial da energia e da hibridização**

  O tempo gasto para a sincronização parcial da energia e da hibridização é dado em função do número de átomos e o número de processadores envolvidos. Pode-se dividi-lo em:

  – Cálculo da energia local e custo de hibridização de cada processador ($C_p(A, p)$):

  $$C_p(A, p) = \frac{A}{p}.(\Gamma + \alpha).$$

  – Tempo de sincronização dos dados parciais e distribuição dos mesmos:

  $$(p - 1)\left(2\mu + \frac{4}{\beta}\right).$$

Para evitar que exista um novo ponto de sincronização, optou-se por executar essa sincronização juntamente com a troca de dados entre vizinhos. Assim o tempo gasto para a sincronização parcial da energia pode ser dado por:

$$\frac{n/p}{k}\left(\frac{A}{p}.(\Gamma+\alpha)+(p-1)\left(2\mu+\frac{4}{\beta}\right)\right).$$

- **Mudança de tamanho do universo e sincronização total**

  O tempo gasto com a mudança de tamanho do universo e com a sincronização total dos dados é dado em função do número de átomos e do número de processos envolvidos, sendo ele dividido em:

  – tempo para enviar os dados para o processo principal

  $$(p-1).\left(\mu+\frac{4.A}{\beta.p}\right).$$

  – tempo de mudança do universo

  $$r.A+\tau.A+\alpha.A+H.A.$$

  em que r é o tempo de mudança de posição de um átomo.

  – tempo de distribuição dos dados

  $$(p-1).\left(\mu+\frac{3.A}{\beta}\right).$$

  – tempo para hibridização dos dados recebidos

  $$H.A.$$

  Pode-se assim definir o tempo para mudar o tamanho do universo e sincronização ($T_s(A,n,p,k')$) como sendo:

$$T_s(A,n,p,k')=\frac{(n/p)}{k'}\left((p-1).\left(2.\mu+\frac{4.A}{\beta.p}+\frac{3.A}{\beta}\right)+A(r+\tau+\alpha+2H)\right)$$

em que k' é o intervalo de iterações entre cada mudança do tamanho do universo.

Somando o tempo das quatro operações acima tem-se:

$$\begin{aligned}T_p =&(n/p).(\Gamma+H+\alpha.A)+\\&\frac{(n/p)}{k}\left(m.\mu+a.\left(\frac{4}{\beta}+H\right)+A.c\right)+\\&\frac{(n/p)}{k}\left(\frac{A}{p}.(\Gamma+\alpha)+(p-1)\left(2\mu+\frac{4}{\beta}\right)\right)+\\&\frac{(n/p)}{k'}\left((p-1).\left(2.\mu+\frac{4.A}{\beta.p}+\frac{3.A}{\beta}\right)+A(r+\tau+\alpha+2H)\right)\end{aligned}$$

ou

$$
\begin{aligned}
T_p =& (n/p).(\Gamma + H + \alpha.A) + \\
& \frac{(n/p)}{k}\left(m.\mu + a.\left(\frac{4}{\beta} + H\right) + A.c + \frac{A}{p}.(\Gamma + \alpha) + (p-1)\left(2\mu + \frac{4}{\beta}\right)\right) + \\
& \frac{(n/p)}{k'}\left((p-1).\left(2.\mu + \frac{4.A}{\beta.p} + \frac{3.A}{\beta}\right) + A(r + \tau + \alpha + 2H)\right).
\end{aligned}
$$

5.3 Sobrecusto

Sabe-se que, quando apenas um processo (p) está envolvido (execução seqüencial), o número de mensagens trocadas pelos processos vizinhos (m), bem como o tamanho dessas mensagens (a) é igual a zero. Portanto, pode-se escrever a fórmula para o tempo seqüencial como sendo:

$$
\begin{aligned}
T_1 =& n.(\Gamma + H + \alpha.A) + \\
& \frac{n}{k}(A.c) + \\
& \frac{n}{k'}(A(r + \tau + \alpha + 2H)),
\end{aligned}
$$

portanto, o sobrecusto da paralelização pode ser dado por:

$$
\begin{aligned}
p.T_p - T_1 =& \frac{n}{k}\left(m.\mu + a.\left(\frac{4}{\beta} + H\right) + \frac{A}{p}.(\Gamma + \alpha) + (p-1)\left(2\mu + \frac{4}{\beta}\right)\right) + \\
& \frac{n}{k'}\left((p-1).\left(2.\mu + \frac{4.A}{\beta.p} + \frac{3.A}{\beta}\right)\right).
\end{aligned}
$$

Como se pode observar, o sobrecusto da paralelização pode ser dividido em duas partes principais. A primeira (que pode ser dividida em duas subpartes) está relacionada com as comunicações entre vizinhos e a sincronização parcial da energia e da hibridização. Na subparte referente à comunicação entre os vizinhos observa-se que o fator principal de crescimento está ligado ao número de átomos trocados (a), sendo que, quanto mais átomos são trocados, maior o custo. Deve-se lembrar que a é definido em função do número total de átomos e o número de processos envolvidos, portanto, à medida que o número de processos cresce, decresce o número de átomos trocados. Sendo assim, essa parte torna-se bastante escalável em termos de processos envolvidos.

Já na subparte referente a sincronização parcial da energia e da hibridização e na segunda parte do sobrecusto (referente a sincronização total dos dados), observa-se um comportamento diferente. Pode-se notar que os termos com maior representação no custo são $(p-1)\,(2\mu)$ e $(p-1)(\frac{3.A}{\beta})$ respectivamente. Esses termos têm seu crescimento diretamente ligado com o número de processos envolvidos (p). Devido a isso, quando o número de processos cresce muito, essas partes tornam-se pouco escaláveis.

Defronte a esses dados, tem-se a necessidade de definir os valores para k e k' de forma que respeitem as características de cada parte, gerem o menor sobrecusto possível e que não interfiram na qualidade numérica do problema.

Para garantir a qualidade numérica, optou-se por definir o valor para k' como sendo algum valor entre $5.k \leq k' \leq 10.k$. Esses valores além de colaborar para a convergência do sistema, também permitem que os dois termos tenham um peso parecido no sobrecusto da paralelização.

O valor de k, por sua vez, pode ser definido pelo número médio de átomos sobre responsabilidade de cada processo. Esse número pode ser obtido através da divisão do número total de átomos (A) pelo número de processos (p).

$$k = \frac{A}{p}$$

Porém, quando o número de processos é muito pequeno, esse número pode ser muito grande (principalmente em simulações com grande número de átomos), podendo, assim, fazer com que a simulação tenda para mínimos locais e não para o mínimo global, como esperado. Já quando o número de processos é muito grande, as comunicações tornam-se muito constantes, fazendo, assim, com que a eficiência da paralelização seja prejudicada.

Uma alternativa é definir k em função de um percentual do número total de átomos. Testes mostram que, para obter uma boa eficiência na paralelização e um bom resultado numérico, esse percentual deve estar entre 10% e 20% e não deve exceder em mais de 5 vezes o número de átomos sobre responsabilidade de cada processo. Essas limitações, estão novamente relacionada ao fato de evitar que o algoritmo convirja para mínimos locais.

Baseado nesses dados, pode-se definir o sobrecusto mínimo para $p <= 25$ (quando $k = \frac{A}{5}$ e $k' = 10.k = 2.A$), como sendo:

$$\begin{aligned} p.T_p - T_1 =& \frac{n}{\frac{A}{5}} \left(m.\mu + a. \left(\frac{4}{\beta} + H \right) + \frac{A}{p}. (\Gamma + \alpha) + (p-1) \left(2\mu + \frac{4}{\beta} \right) \right) + \\ & \frac{n}{2.A} \left((p-1). \left(2.\mu + \frac{4.A}{\beta.p} + \frac{3.A}{\beta} \right) \right), \end{aligned}$$

ou ainda, pode-se simplificar a fórmula, utilizando apenas os termos de maior representatividade. Assim o sobrecusto aproximado pode ser definido como:

$$p.T_p - T_1 \cong \frac{5.a.n}{A} \left(\frac{4}{\beta} + H \right) + \frac{10.n.\mu.(p-1)}{A} + \frac{3.n.(p-1)}{2.\beta}.$$

E para $p > 25$ (quando $k = 5.\frac{A}{p}$ e $k' = 10.k = 50.\frac{A}{p}$) como sendo:

$$\begin{aligned} p.T_p - T_1 =& \frac{n}{5.\frac{A}{p}} \left(m.\mu + a. \left(\frac{4}{\beta} + H \right) + \frac{A}{p}. (\Gamma + \alpha) + (p-1) \left(2\mu + \frac{4}{\beta} \right) \right) + \\ & \frac{n}{50.\frac{A}{p}} \left((p-1). \left(2.\mu + \frac{4.A}{\beta.p} + \frac{3.A}{\beta} \right) \right). \end{aligned}$$

Simplificando a fórmula:

$$p.T_p - T_1 \cong \frac{a.n.p}{5.A} \left(\frac{4}{\beta} + H \right) + \frac{2.n.\mu.p.(p-1)}{5.A} + \frac{3.n.p.(p-1)}{50.\beta}.$$

5.4 Análise

Diante das fórmulas obtidas na seção anterior, pode-se observar que o comportamento da paralelização está dividido em duas partes principais. A primeira delas acontece até o uso de 25 processadores. Nela observa-se que, mantendo-se o mesmo número de átomos, o sobrecusto varia apenas em função do número de processadores envolvidos. Por sua vez, quando mais de 25 processadores estão envolvidos, além do sobrecusto devido ao maior número de processadores, tem-se também um sobrecusto causado pelo fato de que as sincronizações são mais próximas umas das outras.

Para corroborar esses dados, foi realizada uma simulação no *cluster* LabTeC. Esse *cluster* pertence ao Instituto de Informática da Universidade Federal do Rio Grande do Sul. Ele é composto por 20 nós bi-processados Pentium III 1.1 GHz com 1 Gb de memória RAM cada. A interconexão dos mesmos é feita através de uma rede Fast-Ethernet.

A simulação foi feita utilizando-se 2048 átomos variando-se o número de processadores. O número de iterações foi limitado a 8×10^6. Esse número de iterações não é suficiente para convergir em um estado estável, mas é satisfatório para analisar o comportamento da paralelização. Os testes foram realizados 20 vezes com cada configuração, sendo apresentadas as médias desses valores, descartando-se o maior e menor tempo. Os valores para o desvio padrão ficaram abaixo de 1% da média, exceto na distribuição $2 \times 1 \times 1$ que teve esse valor igual a $2,7\%$.

Os dados obtidos na simulação podem ser observados na tabela 5.1. Nela está representado o número de processadores, a distribuição dos mesmos (quantas divisões no eixo x, y e z respectivamente), o tempo de execução, o sobrecusto (obtido pela multiplicação do tempo paralelo pelo número de processadores envolvidos, subtraído do tempo seqüencial), o *speedup* (obtido pela divisão do tempo seqüencial pelo tempo paralelo) e a eficiência (obtida pela divisão do *speedup* pelo número de processadores envolvidos).

Tabela 5.1: Valores referentes às execuções da versão paralela do algoritmo com diferentes configuração

Processadores	**Distribuição**	**Tempo (s)**	**Sobrecusto (pTp-T1)**	***Speedup***	**Eficiência**
1	$1 \times 1 \times 1$	$1670,8$	-	$1,00$	$100,0\%$
2	$2 \times 1 \times 1$	$1025,8$	$370,4$	$1,64$	$81,9\%$
4	$4 \times 1 \times 1$	$463,6$	$182,3$	$3,55$	$90,2\%$
4	$2 \times 2 \times 1$	$470,3$	$209,7$	$3,64$	$88,9\%$
8	$2 \times 2 \times 2$	$237,3$	$226,1$	$7,05$	$88,1\%$
8	$4 \times 2 \times 1$	$237,4$	$226,6$	$7,04$	$88,1\%$
12	$3 \times 2 \times 2$	$160,2$	$251,2$	$10,43$	$86,9\%$
16	$4 \times 2 \times 2$	$124,0$	$312,3$	$13,48$	$84,3\%$
24	$4 \times 3 \times 2$	$92,7$	$554,6$	$18,02$	$75,1\%$
27	$3 \times 3 \times 3$	$88,1$	$706,7$	$18,97$	$70,3\%$
32	$4 \times 4 \times 2$	$80,0$	$889,2$	$20,88$	$65,3\%$
36	$4 \times 3 \times 3$	$77,3$	$1112,0$	$21,62$	$60,1\%$

Para facilitar a visualização dos dados, nas figuras 5.4, 5.5, 5.6 e 5.7 são respectivamente representados os gráficos de variação do tempo de processamento, do sobrecusto, da eficiência e do *Speedup* em função do número de processadores.

Pode-se observar que, a partir de 4 processos, os dados coletados nessa simulação validam as fórmulas apresentadas na seção anterior. Eles mostram que o sobrecusto tente a crescer a medida que o número de processos aumenta, sendo que, a partir de 25 pro-

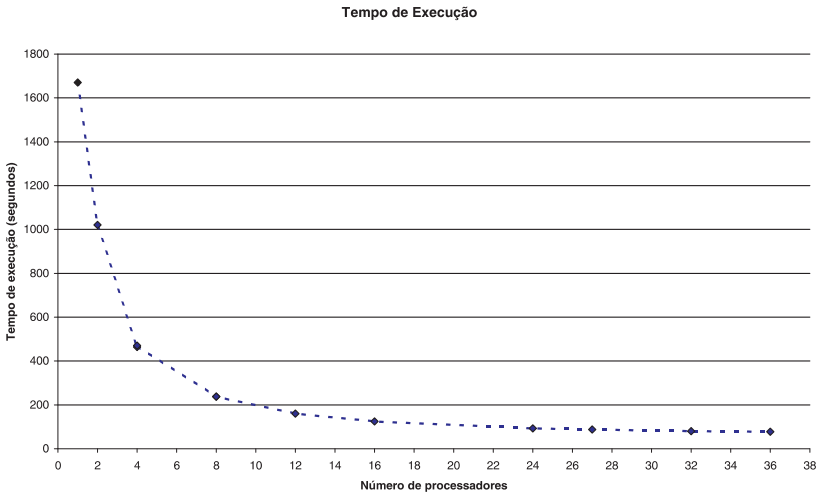


Figura 5.4: Gráfico do tempo de execução em função do número de processadores

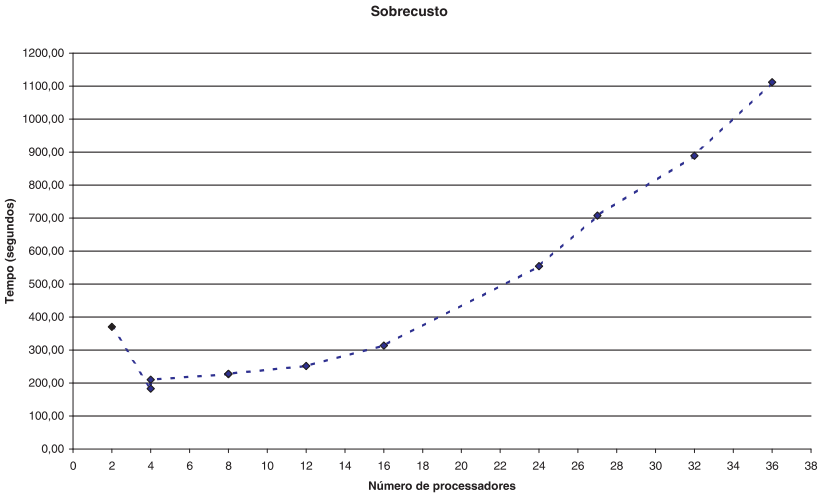


Figura 5.5: Gráfico do sobrecusto da paralelização em função do número de processadores

cessadores esse crescimento se acentua ainda mais. Nas simulações com 2 processadores os dados divergem significativamente do esperado. O motivo dessa divergência não foi detectado, mas crê-se que este esteja ligado à dependência mútua de dados entre os 2 processadores.

Também observa-se que, a partir de 25 processadores, os ganhos provenientes da inclusão de novos recursos tornam-se cada vez menores. Por exemplo, ao aumentar de 24 para 36 processadores tem-se um acréscimo de 50% nos recursos utilizados, porém, o ganho de desempenho é de apenas 20%. Dessa forma pode-se observar que existe uma limitação da paralelização que está concentrada pouco acima de 25 processadores.

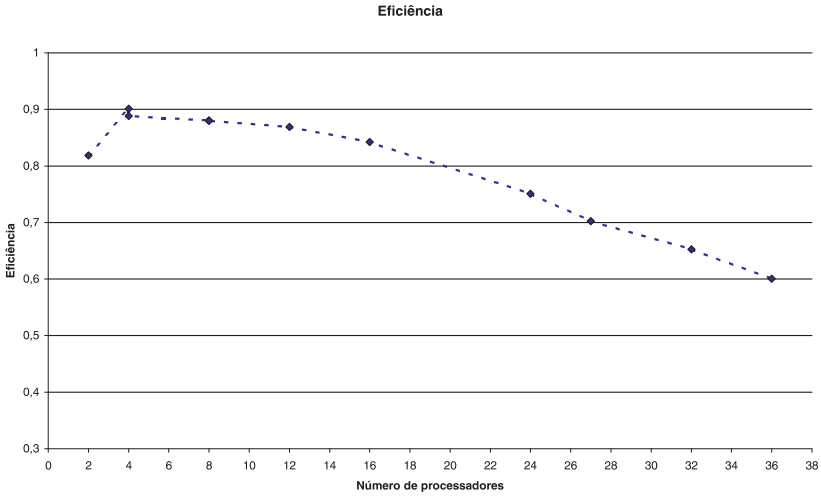


Figura 5.6: Gráfico da eficiência em função do número de processadores

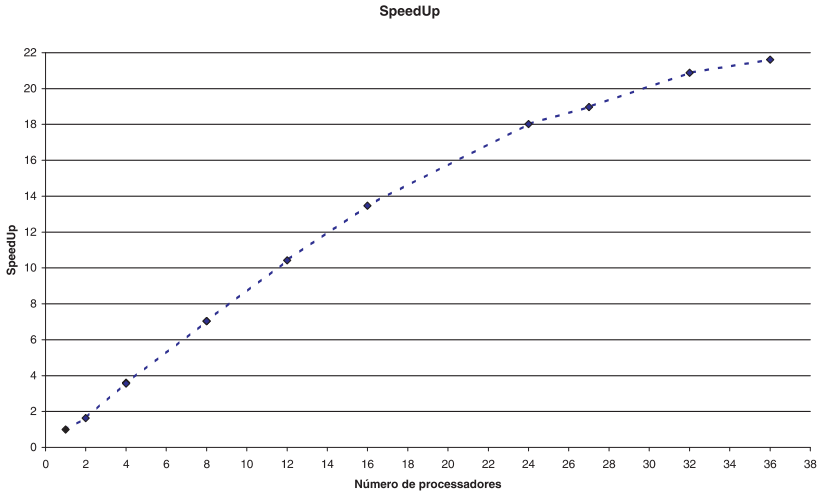


Figura 5.7: Gráfico de *Speedup* em função ao número de processadores

5.5 Considerações finais

Nesse capítulo foi realizada a validação dos resultados obtidos na versão paralela do algoritmo de geração de redes aleatórias contínuas através da geração de redes nos *clusters* Polentão e Hal da Universidade de Caxias do Sul e nos *clusters* Corisco e LabTeC da Universidade Federal do Rio Grande do Sul e a comparação destas com as redes previamente geradas com a versão seqüencial do mesmo algoritmo. Essa comparação mostrou que a versão paralela é equivalente à versão seqüencial. Além disso, foi feita uma análise teórica do custo computacional agregado com a paralelização do algoritmo proposto nesse trabalho. Essa análise permitiu que, através das formulas obtidas, possa-se analisar o comportamento do programa em diferentes situações. Também, foram realizados testes práticos de execuções da solução paralela no *cluster* LabTeC. Nesses testes, pode-se observar que, devido as características do problema, tem-se uma diminuição da eficiência à medida que novos processadores são utilizados. Essa diminuição torna-se mais crítica a partir de 25 processadores, momento o qual se torna necessárias comunicações mais freqüentes entre os processadores.

6 CONCLUSÕES

Nesse trabalho foi apresentado o estado da arte sobre processos de otimização combinatorial dando uma ênfase especial ao método *Simulated Annealing*. Foi apresentado seu histórico, funcionalidades, algorítmo genérico e propostas de paralelização presentes na literatura. Também foi apresentado o "Algoritmo de Geração de Redes Aleatórias Contínuas", algoritmo esse, projetado por pesquisadores do instituto de física da Universidade Federal do Rio Grande do Sul e que utiliza-se do método *Simulated Annealing* para gerar redes que atendam certas restrições.

Como foi visto anteriormente, devido ao grande número de variáveis envolvidas nas simulações com esse algoritmo, essas podem levar meses de processamento até convergir em um estado estável que atendam as restrições impostas. Uma vez de posse dessa limitação, foi proposta a busca na literatura de programas e técnicas de paralelização que pudessem ser aplicadas ao método *Simulated Annealing* a fim de reduzir esse tempo de processamento.

Alguns destes foram testados, porém nenhum se mostrou 100% eficiente para a resolução do problema em questão. Frente a isso, foi proposta uma abordagem que mescla a técnica de Múltiplas Cadeias de Markov Síncronas Interativa presente na literatura com a técnica de divisão de domínio. Dessa forma foi possível que um maior grau de paralelismo fosse explorado (todos os movimentos feitos são efetivos e não apenas o do processador que se saiu melhor, como nos casos citados na literatura). Em contrapartida, foi necessária a criação de novos pontos de sincronização, os quais tornaram o *speedup* não tão linear quanto o esperado.

Embora o *speedup* não tenha sido tão próximo ao linear como esperado em uma aplicação paralela convencional, ele mostrou-se bastante satisfatório em comparação a outros resultados obtidos em paralelizações do método *Simulated Annealing*. Além disso, com a versão paralela desse algoritmo foi possível, para os pesquisadores da física, que novas redes fossem geradas com um tempo muito inferior aos obtidos com a versão seqüencial, sendo, assim, de grande utilidade prática.

Outro ponto que pôde ser observado é que a proposta de paralelização apresentada nesse trabalho mostra-se bastante versátil, podendo ser facilmente adaptada a outros problemas baseados em SA sendo apenas necessário a mudança de alguns conceitos. Isso pode ser observado através do exemplo que ilustra os passos necessários para utilizá-la na paralelização do problema do Caixeiro Viajante.

Como foi citado, a arquitetura alvo da paralelização desse trabalho é formada de *clusters*. Sendo assim, devido ao crescente avanço das novas arquiteturas multi-cores, um possível próximo passo a ser seguido seja adaptar a paralelização proposta nesse trabalho de forma que ela possa explorar com maior eficiência detalhes presentes nessa arquitetura, tais como memória compartilhada.

Outro detalhe que pode ser implementado é o refinamento do trabalho de cada processador em regiões onde um maior número de cálculos é exigido. Isso pode ser feito através da criação de novos processos/*threads* que dividirão as tarefas nessas regiões. Dessa forma, poder-se-ia garantir que um melhor balanceamento de carga fosse efetuado.

REFERÊNCIAS

AARTS, E.; KORST, J. **Simulated Annealing and Boltzmann Machines**: a stochastic approach to combinatorial optimization and neural computing. New York: John Wiley & Sons, 1988.

AZENCOTT, R. **Simulated Annealing**: parallelization techniques. New York: John Wiley & Sons, 1992.

BANERJEE, P.; JONES, M. H.; SARGENT, J. S. Parallel Simulated Annealing Algorithms for Standard Cell Placement on Hypercube Multiprocessors. **IEEE Transactions on Parallel and Distributed Systems**, Piscataway, NJ, USA, v.1, p.91–106, Jan. 1990.

BARKEMA, G. T.; MOUSSEAU, N. High-quality continuous random networks. **Phys. Rev. B**, [S.l.], v.62, n.8, p.4985–4990, Aug. 2000.

BEVILACQUA, A. A methodological approach to parallel simulated annealing on an SMP System. **J. Parallel Distrib. Comput.**, Orlando, FL, USA, v.62, n.10, p.1548–1570, 2002.

CAMPELLO, R. E.; MACULAN, N. **Algoritmos e heuristicas** : desenvolvimento e avaliacao de performance. Niteroi: Eduff, 1994.

CASOTTO, A.; ROMEO, F.; SANGIOVANNI-VINCENTELLI, A. L. A Parallel Simulated Annealing Algorithm for the Placement of Macro-Cells. **IEEE Trans. on CAD of Integrated Circuits and Systems**, [S.l.], v.6, n.5, p.838–847, 1987.

CERNÝ, V. Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. **Journal of Optimization Theory and Application**, [S.l.], v.45, p.41–51, 1985.

CHANDY, J. A.; BANERJEE, P. Parallel simulated annealing strategies for VLSI cell placement. In: INTERNATIONAL CONFERENCE ON VLSI DESIGN: VLSI IN MOBILE COMMUNICATION, 1996. **Proceedings...** [S.l.]: IEEE Computer Society, 1996. p.37.

CHANDY, J. A. et al. An evaluation of parallel simulated annealing strategies with application to standard cell placement. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, New York, v.16, n.4, p.398–410, Apr. 1997.

CHEN, H.; FLANN, N. S.; WATSON, D. W. Parallel Genetic Simulated Annealing: a massively parallel simd algorithm. **IEEE Trans. Parallel Distrib. Syst.**, Piscataway, NJ, USA, v.9, n.2, p.126–136, 1998.

DELAMARRE, D.; VIROT, B. **Simulated annealing algorithm**: amelioration techniques. [S.l.]: Laboratoire d Informatique Fondamentale d Orléans - Université d Orleans, 1993. Tech. Report.

GAREY, M. R.; JOHNSON, D. S. **Computers and Intractability**: a guide to the theory of np-completeness. San Francisco: W. H. Freeman, 1990.

GEIST, A. et al. **PVM**: a users' guide and tutorial for networked parallel computing. [S.l.]: MIT Press, 1994.

GREENING, D. R. Parallel Simulated Annealing Techniques. **Physica D: Nonlinear Phenomena**, Amsterdam, v.42, n.1-3, p.293–306, Aug. 1991. Trabalho apresentado na Annual International Conference of the Center for Nonlinear Studies and Self-Organizing, Collective, and Cooperative Phenomena in Natural and Artificial Computing Networks, 1990.

GROPP, W. et al. **A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard**. [S.l.]: Argonne National Laboratory, 1996. Technical Report, Disponível em: <http://www.mcs.anl.gov/mpi/mpich>. Acesso em: fev.2008.

GROPP, W.; LUSK, E.; THAKUR, R. **Using MPI-2**. [S.l.]: MIT Press, 1999.

HAMMERSELEY, J. M.; HANDSCOMB, D. C. **Monte Carlo method**. London: [s.n.], 1964.

HENDERSON, D.; JACOBSON, S. H.; JOHMSON, A. W. **Handbook of Metaheuristics**. [S.l.]: Kluwer Academic Publishers, 2003.

HOROWITZ, E.; SAHNI, S. **Fundamentals of Computer Alori**. New York, NY, USA: W. H. Freeman, 1978.

HROMKOVIC, J. **Algorithmics for hard problems**: introduction to combinatorial optimization, randomization, approximation, and heuristics. New York, NY, USA: Springer-Verlag, 2001.

IMMEL, D. S. **Dr. Stefan Immel Webpage**. Disponível em: <http://csi.chemie.tu-darmstadt.de/ak/immel/tutorials/orbitals/hydrogenic.html>. Acesso em: nov. 2007.

JAIN, R. **The Art of Computer Systems Performance Analysis**. [S.l.]: Wiley & Sons, 1991.

JORNADA, F. H. **Geração de Redes Contínuas Aleatórias de Carbono por Simulated Annealing**. 2007. Trabalho de Conclusão de Curso (Física) - Instituto de Física, UFRGS, Porto Alegre.

KELLY, J. P. **Meta-Heuristics**: theory and applications. Norwell, MA, USA: Kluwer Academic Publishers, 1996.

KIRKPATRICK, S.; GELATT C. D., J.; VECCHI, M. P. Optimization by Simulated Annealing. **Science**, [S.l.], v.220, n.4598, p.671–680, 1983.

KLIEWER, G. A General Software Library for Parallel Simulated Annealing. In: EURO WINTER INSTITUTE ON METAHEURISTICS IN COMBINATORIAL OPTIMISATION, 2000, Lac Noir, Switzerland. **Proceedings...** [S.l.: s.n.], 2000. Submitted to the European Jornaul of Operational Research (EJOR).

KLIEWER, G.; TSCHöKE, S. **Parallel Simulated Annealing Library (parSA)**: user manual. [S.l.]: University of Paderborn, 1999. Relatório Técnico.

KLIEWER, G.; TSCHöKE, S. A General Parallel Simulated Annealing Library and Its Application in Airline Industry. In: INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM, IPDPS, 14., 2000, Cancun, Mexico. **Proceedings...** [S.l.: s.n.], 2000. p.55–61.

KNOPMAN, J.; AUDE, J. S. Parallel Simulated Annealing: an adaptive approach. In: INTERNATIONAL PARALLEL PROCESSING SYMPOSIUM, IPPS, 11., 1997, Geneva, Switzerland. **Proceedings...** Los Alamitos CA: IEEE Computer Society, 1997. p.522–526.

KRAVITZ, S. A.; RUTENBAR, R. A. Multiprocessor-based placement by simulated annealing. In: ACM/IEEE DESIGN AUTOMATION CONFERENCE , DAC 23., 1986, Las Vegas. **Proceedings...** IEEE Press, 1986. p.567–573.

LAMMPI. **LAM/MPI Parallel Computing**. Disponível em: <http://www.lam-mpi.org/>. Acesso em: nov. 2007.

LEE, S.-Y.; LEE, K. G. Synchronous and Asynchronous Parallel Simulated Annealing with Multiple Markov Chains. **IEEE Trans. Parallel Distrib. Syst.**, Piscataway, NJ, USA, v.7, n.10, p.993–1008, 1996.

MAHFOUD, S. W.; GOLDBERG, D. E. Parallel recombinative simulated annealing: a genetic algorithm. **Parallel Computing**, [S.l.], v.21, n.1, p.1–28, 1995.

METROPOLIS, N. et al. Equation of state calculations by fast computing machines. **Journal of Chemical Physics**, [S.l.], v.21, n.1087, p.1087–1092, 1953.

MITRA, D.; ROMEO, F.; SANGIOVANNI-VINCENTELLI, A. L. **Convergence and Finite-Time Behavior of Simulated Annealing**. [S.l.]: EECS Department, University of California, Berkeley, 1985. (UCB/ERL M85/23).

MITZENMACHER, M.; UPFAL, E. **Probability and Computing**: randomized algorithms and probabilistic analysis. New York, NY, USA: Cambridge University Press, 2005.

MOHARIL, S.; LEE, S.-Y. **Parallel Simulated Annealing with Load Balancing on Temporally Heterogeneous Cluster of Workstations**. [S.l.]: Electrical and Computer Engineering Department, Auburn University, 2005. Tech. Report.

MPICH. **MPICH2**: high-performance and widely portable mpi. Disponível em: <http://www-unix.mcs.anl.gov/mpi/mpich/>. Acesso em: nov. 2007.

NATARAJAN, K. S. Graph-Partitioning on Shared-Memory Multiprocessor Systems. In: ICPP, 3., 1991. **Proceedings...** [S.l.: s.n.], 1991. p.120–124.

OPENMPI. **Open MPI**: open source high performance computing. Disponível em: <http://www.open-mpi.org/>. Acesso em: nov. 2007.

PEROTTONI, C. A.; JORNADA, J. A. H. The carbon analogues of type-I silicon clatrates. **J. Phys.: Condens. Matter**, [S.l.], v.13, p.5981–5998, 2001.

ROMEO, F.; SANGIOVANNI-VINCENTELLI, A. A Theoretical Framework for Simulated Annealing. **Algorithmica**, [S.l.], v.6, n.3, p.302–345, 1991.

SNIR, M. et al. **MPI**: the complete reference. [S.l.]: MIT Press, 1996.

SOHN, A. Parallel N-ary Speculative Computation of Simulated Annealing. **IEEE Trans. Parallel Distrib. Syst.**, Piscataway, NJ, USA, v.6, n.10, p.997–1005, 1995.

SUN, W.-J.; SECHEN, C. A loosely coupled parallel algorithm for standard cell placement. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, ICCAD, 1994, San Jose. **Proceedings...** Los Alamitos CA: IEEE Computer Society Press, 1994. p.137–144.

WOOTEN, F.; WINER, K.; WEAIRE, D. Computer Generation of Structural Models of Amorphous Si e Ge. **Physical Review Letters**, [S.l.], v.54, n.13, p.1392–1395, 1985.

ZACHARIASEN, W. H. The Atomic Arrangement in Glass. **J. American Ceramic Society**, [S.l.], v.54, n.9, p.3841–3851, 1932.