4

Aplicações de Alto Desempenho Trivialmente Paralelizáveis

Ney Lemke[1] *(UNISINOS, lemke@exatas.unisinos.br)*

Resumo:

Neste curso aplicações com alta demanda computacional que podem ser executados de forma eficaz em máquinas de arquitetura paralela são apresentadas e duas classes de métodos são enfocadas: simulações de Monte Carlo e heurísticas genéticas. Em cada caso uma fundamentação teórica é apresentada além de diferentes implementações e alguns problemas que podem ser atacados por estes métodos. Nas conclusões um balanço sobre o estado da arte é feito e algumas áreas interessantes de investigação são apresentadas.

[1]Doutor em Física junto ao IF da UFRGS. Bacharel em Física pela UFRGS. Professor e Pesquisador do PIPCA Unisinos. Orientador de Mestrado do PIPCA. Áreas de Interesse: Sistemas complexos e desordenados, Inteligência Artificial, Processamento de Alto Desempenho e Bioinformática.
PipCA – Centro de Ciências Exatas e da Terra – Unisinos, Av. Unisinos, 950, 93022-000 – São Leopoldo – RS – Brazil

4.1. Introdução

Os computadores de hoje são insuficientes para resolver os problemas da ciência do século XXI. Apesar da revolução que vem ocorrendo na Informática, a velocidade dos computadores é ainda insuficiente para resolver os principais problemas em áreas como meteorologia, medicina, biociências, fusão controlada e mesmo em muitas aplicações comerciais. Sterling [STE 01] em um artigo recente para a *Scientific American* estima que estes problemas necessitam de computadores 1000 vezes mais rápidos que os super-computadores atuais. Como exemplo considere a determinação da estrutura tridimensional de macro-moléculas biológicas, ingrediente essencial para a compreensão de sua funcionalidade. Estima-se que para realizar esta tarefa serão necessários computadores capazes de executar um quatrilhão de operações de ponto flutuante por segundo.

Para atingir esta performance existem duas soluções possíveis: o desenvolvimento de hipercomputadores com arquiteturas revolucionárias (usando computação quântica, por exemplo) e custos extremamente elevados ou a construção de computadores de arquitetura paralela usando computadores pessoais com custos menores. Historicamente a primeira solução tem se mostrado inadequada, pois o tempo de desenvolvimento de hipercomputadores é muito longo, o que faz com que uma vez que o sistema esteja operante, agregados de máquinas de custo muito mais baixo já estão disponíveis com uma performance comparável.

Infelizmente a programação de agregados de máquinas paralelas é um desafio considerável e os ganhos obtidos em muitas aplicações é pífio. Contudo existem técnicas numéricas que são extremamente poderosas e que podem ser executados nesta arquitetura de forma extremamente eficiente, nosso objetivo neste curso será descrever estes métodos e suas inúmeras aplicações. Dois métodos serão enfocados: algoritmos de Monte Carlo e heurísticas genéticas. Eles foram escolhidos porque apesar de serem simples podem ser aplicados a um espectro amplo de problemas em quase todas as áreas do conhecimento.

Para colocar em perspectiva a importância da computação de alto desempenho listo abaixo algumas áreas do conhecimento que serão beneficiadas com seu desenvolvimento.

Meteorologia Computadores mais potentes serão capazes de integrar dados provenientes de satélites gerando mapas detalhados que possibilitarão prever o comportamento do clima com maior precisão por períodos mais longos de tempo.

Medicina e Biociências O sequênciamento genético, a determinação da estrutura tridimensional de biomoléculas e do mapa de interação dos genes são problemas complexos que nos permitirão desenvolver remédios mais eficazes e aumentar o nosso conhecimento sobre a vida.

Comércio e Finança O desenvolvimento de modelos mais acurados do comportamento do consumidor e do funcionamento da economia facilitará o desenvolvimento de políticas e a criação de sistemas comerciais mais eficientes.

Astronomia A modelagem da galáxia e o tratamento de dados provenientes de telescópios como o Hubble nos permitirá testar modelos tanto em astronomia como em física fundamental.

O plano do curso é na seção 4.2. apresentar a técnica de Monte Carlo e suas aplicações, na seção 4.3. discutimos as heurísticas genéticas e na seção 4.4. discutimos algumas perspectivas e apresentamos nossas conclusões.

4.2. Algoritmos de Monte Carlo

O Método de Monte Carlo apresenta soluções aproximadas para uma grande variedade de problemas físicos e matemáticos através da realização de experimentos numéricos que utilizam números aleatórios ou pseudo-aleatórios. Estes métodos podem ser aplicados tanto em problemas de natureza probabilística quanto problemas determinísticos. O nome do método é inspirado na famosa cidade de Monte Carlo no principado de Mônaco e nas roletas dos cassinos, que são um exemplo de geradores de números aleatórios [PLL].

A teoria e o desenvolvimento dos métodos de Monte Carlo foi iniciada em 1944 apesar de existirem exemplos de aplicações de métodos probabilísticos na solução de problemas matemáticos desde o século XVIII. O exemplo mais célebre é o da determinação de π proposto pelo conde de Buffon. Uma agulha é atirada em tabuleiro que consiste de linhas paralelas igualmente espaçadas, π pode ser estimado contando a freqüência com que a agulha cruza as linhas (para um teste prático do método visite a página http://www.geocities.com/CollegePark/Quad/2435/buffon.html).

Ainda que muitos outros tenham utilizado métodos de Monte Carlo entre eles Lorde Rayleigh e Kolmogorov, o uso destas técnicas somente se popularizou depois de 1930 quando Enrico Fermi o utilizou para calcular a difusão de nêutrons em reatores nucleares. Na década de 40 uma fundação formal do método foi desenvolvida por von Neumann em colaboração com Stanislaw Ulam. As primeiras simulações foram executadas no computador ENIAC durante a realização do projeto Manhattan [MET 87, ECK 87, ULA 49, ULA 47]. Desde então com o surgimento de computadores cada vez mais potentes o método se popularizou e hoje é uma das técnicas numéricas mais utilizadas.

4.2.1. Cálculo de integrais

Uma aplicação usual do método de Monte Carlo é o cálculo de integrais. Isto parece de início um contra-senso como usar números aleatórios para estimar uma quantidade não aleatória?

Suponha que queremos calcular a integral:

$$F = \int_a^b f(x)dx. \tag{4.1}$$

Admitindo que f é sempre positiva no intervalo (a, b), podemos interpretar a integral como sendo a área sob a curva $f(x)$. O primeiro passo fundamental é determinar uma cota superior para f (um número maior do que todos os valores que f assume no intervalo (a, b)). Esta quantidade denotamos por H. Agora sabemos que a curva definida por f está inteiramente contida no retângulo de lados H e $(b - a)$, conforme está representado na fig. 4.1.

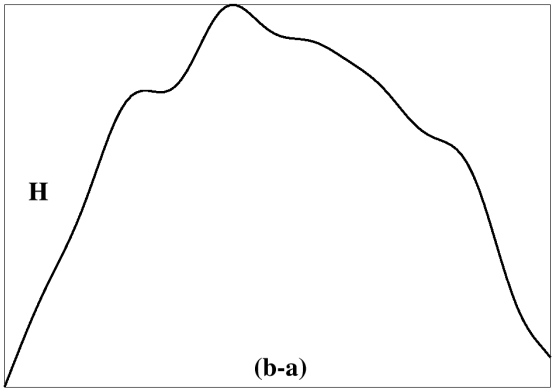

Figura 4.1: A função f e a caixa de lados $(b-a)$ e H.

Para estimar a área sob a curva $f(x)$ calculamos pares de números x_i e y_i que satisfaçam $a \leq x_i \leq b$ e $0 \leq y_i \leq H$. A fração de pontos x_i, y_i que satisfaz a condição $y_i \leq f(x_i)$ é um estimador da razão entre a integral de $f(x)$ e a área do retângulo. Assim temos que o estimador F_n é dado por:

$$F_n = A\frac{n_s}{n}$$

onde n_s são os pontos abaixo da curva, n o número total de pontos e $A = H(b-a)$. A medida que n cresce temos estimativas cada vez mais precisas para a integral.

Outro método de integração usando técnicas de Monte Carlo está baseado no teorema dos grandes números. Considere uma variável aleatória distribuída uniformemente no intervalo (a, b) e de uma função f definida neste intervalo. O teorema garante que o valor médio de f dado por:

$$\langle f \rangle = \frac{1}{n}\sum_{i=1}^{n} f(x_i) \tag{4.2}$$

e o valor esperado definido como:

$$E(f(x)) = \frac{1}{b-a}\int_a^b f(x)dx \tag{4.3}$$

no limite em que n tende a infinito são iguais com probabilidade 1. O termo $(b-a)$ foi incluído na expressão 4.3 para garantir a normalização.

Um exemplo para facilitar a compreensão deste resultado é considerar uma variável aleatória S que assume com igual probabilidade os valores 0 e 1. Sabemos que o valor esperado de S $E(S)$=1/2.0+1/2.1=1/2, se agora geramos uma seqüência de n valores de S intuitivamente esperamos que o valor médio desta seqüência esteja próximo de 1/2 quando n for bastante grande.

Agrupando os resultados das expressões 4.2 e 4.3 obtemos um estimador para a integral:

$$F_n = (b-a)\langle f \rangle = \frac{(b-a)}{n} \sum_{i=1}^{n} f(x_i) \tag{4.4}$$

onde x é uma variável aleatória uniformemente distribuída no intervalo (a, b) e n é o número de tentativas.

Este método pode ser facilmente estendido para um número arbitrário de dimensões, para ilustrar este procedimento consideramos apenas o caso bidimensional. Neste caso queremos estimar a integral:

$$I = \int_a^b \int_c^d f(x)\, dx \tag{4.5}$$

um estimador para esta integral é:

$$F_n = \frac{(b-a)(d-c)}{n} \sum_{i=1}^{n} f(x_i, y_i) \tag{4.6}$$

onde x_i e y_i são variáveis aleatórias uniformemente distribuídas nos intervalos (a, b) e (c, d) e n é o número de tentativas.

O método de Monte Carlo não é eficaz para problemas com poucas variáveis mas é o método mais adequado quando temos de resolver problemas com um número muito grande de variáveis.

No apêndice 4.6.2. apresentamos um exemplo de cálculo de integrais utilizando o método de Monte Carlo e a interface MPI. O algoritmo é bastante simples, dividimos a média entre k processos, uma vez que estes cálculos são finalizados os resultados são adicionados utilizando a função `MPIReduce` e temos uma estimativa para a integral.

4.2.2. Métodos de Monte Carlo em mecânica estatística

A mecânica estatística é a área da física que investiga o comportamento de sistemas formados por um número muito grande de partículas. Tais como um gás em uma sala, um sólido desordenado, ou uma galáxia. Dada a complexidade destes sistemas a determinação de suas propriedades não pode ser realizada através da descrição exata do comportamento de seus componentes mas sim através de ferramentas estatísticas.

Quando investigamos um gás em uma sala não queremos saber qual é a velocidade de cada uma de suas moléculas, mas sim qual é a distribuição de probabilidade das velocidades, ou ainda qual é a velocidade média ou o desvio padrão da velocidade. Estas propriedades estão relacionados com propriedades observáveis do sistema, por exemplo a temperatura de um gás está diretamente relacionada com a velocidade média.

Os sistemas estudados em mecânica estatística são formados por muitas unidades interagentes, o que torna estes problemas particularmente difíceis de serem tratados analiticamente. O que nos leva ao desenvolvimento de técnicas numéricas capazes de comparar nossos modelos teóricos com a realidade experimental. As técnicas de Monte Carlo são extremamente populares nesta área e são para muitos sistemas a única ferramenta disponível.

Neste curso estamos interessados em investigar sistemas estatísticos que estejam em contato com um reservatório térmico (um sistema muito maior do que o sistema investigado que se mantém a uma temperatura T). Esta situação é interessante pois é fácil de ser comparada com a experiência e ao mesmo tempo simples de ser investigada teoricamente. Nesta situação o nosso sistema troca energia com o reservatório mas mantém fixa sua temperatura.

Para estudar esta situação imaginamos uma grande quantidade de cópias do sistema todos com o mesmo volume V e o mesmo número de partículas N em equilíbrio a uma temperatura T. A probabilidade que o sistema esteja em um estado s com energia E é dada por:

$$P_s = \frac{e^{-\beta E_s}}{Z} \tag{4.7}$$

onde $\beta = 1/kT$, k é a constante de Boltzmann que por simplicidade nas simulações assumimos como sendo 1 e Z é uma constante de normalização dada por:

$$Z = \sum_{s=1}^{M} e^{-\beta E_s}. \tag{4.8}$$

Z é chamada de função partição do sistema. Z é a chave para a descrição do sistema, através dela podemos conhecer todas as grandezas de interesse.

Estas grandezas podem ser obtidas realizando médias sobre a distribuição de probabilidades:

$$\langle A \rangle = \frac{1}{Z} \sum_{s} A_s e^{-\beta E_s} \tag{4.9}$$

onde a soma se estende por todos os estados possíveis do sistema e A é uma grandeza como energia, velocidade etc.

Para descobrir mais sobre mecânica estatística eu recomendo o livro clássico de Kerson Huang "Statistical Mechanics" [HUA 87] que contém o essencial para a compreensão dos fundamentos da mecânica estatística, para uma visão mais atualizada sugiro o livro "A Modern Course in Statistical Physics" [REI 98] de Linda Reichl que além de fornecer uma visão sobre a base da teoria também apresenta alguns dos avanços obtidos nós últimos anos.

4.2.2.1. O algoritmo de Metropolis

Vamos assumir que estamos interessados em investigar um sistema físico descrito por alguma variável discreta, S. Em física é usual associarmos esta variável com o spin, que é o momento magnético intrínseco de uma partícula, mesmo em situações em que esta variável tem outra interpretação. Por simplicidade então assumimos que S só pode assumir os valores 1 e -1. O nosso problema pode ser descrito usando N variáveis S, onde N é um número muito grande.

Temos então uma função E que depende do estado de cada um dos spins e representa a energia do sistema. Para calcular o valor de $\langle A \rangle$ uma função qualquer dos spins precisamos então realizar as somas na equação (4.9). O número de estados possíveis de nosso sistema é 2^N, e mesmo para valores pequenos como 200, realizar estas somas

é computacionalmente proibitivo. A alternativa é utilizar técnicas de Monte Carlo para estimar $\langle A \rangle$.

Um plano bastante simples consiste em realizar as médias considerando não todos os estados possíveis mas apenas m estados tomados aleatoriamente. Inocentemente poderíamos pensar que bastaria tomar um número suficientemente grande de estados aleatórios. A verdade é que devido a natureza do problema apenas uma fração diminuta dos estados colabora de forma significativa para as somas 4.9. Quando tomamos estados aleatórios estes tem uma participação muito pequena na soma e acabamos concentrando todo nosso esforço computacional em estados cuja contribuição é desprezível.

Para minimizar este problema a solução é escolher os estados aleatórios usando uma distribuição de probabilidade π que favoreça os estados que de fato colaboram para a soma. A média neste caso é:

$$A_m = \frac{\sum_{s=1}^{m} (A_s/\pi_s) e^{-\beta E_s}}{\sum_{s=1}^{m} (1/\pi_s) e^{-\beta E_s}} \tag{4.10}$$

Este procedimento é chamado de amostragem por importância.

O próximo passo é escolher π de forma a minimizar o custo computacional de calcular $\langle A \rangle$. A melhor escolha possível é:

$$\pi_s = \frac{e^{-\beta E_s}}{\sum_{i=1}^{m} e^{-\beta E_s}} \tag{4.11}$$

Neste caso temos:

$$\langle A \rangle = \frac{1}{m} \sum_{s=1}^{m} A_s. \tag{4.12}$$

Ou seja com esta distribuição π, calcular $\langle A \rangle$ se reduz ao cálculo de uma média tradicional!

A questão agora é como produzir estados com distribuição de probabilidade π.

A idéia é gerar uma seqüência de estados do sistema $\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_n$ de forma recursiva com uma probabilidade de transição de um estado para outro dada por $W(\mathbf{S}_i \rightarrow \mathbf{S}_{i+1})$. Observe que $\mathbf{S}$ é um vetor com todos os valores S que caracterizam o sistema e W fornece a probabilidade de irmos de um estado $\mathbf{S}$ para outro. Esta seqüência de estados aleatórios é chamada de uma cadeia de Markov.

Para garantir que esta cadeia produza estados distribuídos de acordo com π temos que garantir duas condições básicas:

acessibilidade de um dado estado inicial deverá sempre ser possível atingir qualquer outro estado aplicando a regra de transição um número suficiente de vezes

balanço detalhado a transição de probabilidade deve satisfazer a seguinte condição:

$$\pi_s W(s \rightarrow s') = \pi_{s'} W(s' \rightarrow s) \tag{4.13}$$

Podemos demonstrar que qualquer W que satisfaça estas condições possui como distribuição de equilíbrio π. Ou seja se escolhemos estados iniciais com uma distribuição qualquer, a medida que geramos novos estados, a distribuição de probabilidades destes estados tende a distribuição π [NEW 99].

O algoritmo de Metropolis é um dos métodos que nos possibilita produzir uma cadeia de Markov com a distribuição π. O algoritmo pode ser descrito pelos passos:

1. escolha um estado inicial;
2. faça uma mudança no estado S invertendo um dos spins;
3. calcule $\Delta E = E_{tent} - E_{atual}$, a mudança na energia devido a mudança proposta;
4. se ΔE for menor que zero aceite a mudança e vá para o passo 8;
5. se ΔE for positivo calcule $w = e^{-\beta \Delta E}$;
6. gere um número aleatório em [0,1];
7. se $r \leq w$ aceite a mudança caso contrário mantenha o estado atual;
8. repita os passos de (2) até (7) até o sistema atingir o equilíbrio, o tempo necessário para atingir o equilíbrio é denotado por τ_{eq};
9. repita os passos de (2) até (7) e a cada τ passos calcule as quantidades físicas de interesse.

O passo (2) pode ser realizado de várias formas, ou escolhendo aleatoriamente o spin ou seqüencialmente atualizando cada spin do sistema. Definimos um MCS (*Monte Carlo Step*), passo de Monte Carlo) como sendo N passos de atualizações de spins.

Dois parâmetros importantes para a realização das simulações são τ_{eq} e τ, que dependem do problema que estamos interessados em simular e do número de spins que dispomos. Existem inúmeras técnicas para determinar estes tempos. A mais simples é medir alguma quantidade de interesse do sistema, digamos E, e esperar até que esta quantidade se estabilize, ou ainda simular o sistema em dois estados iniciais diferentes e esperar que os dois sistemas se equiparem [NEW 99]. Estimar τ pode ser mais complexo e depende do cálculo de funções correlação (maiores detalhes consulte [NEW 99]), mas a correta estimativa de τ é menos crítica para as simulações. Assim podemos realizar alguns testes para comparar a robustez de nossos modelo testando vários valores para τ. Em uma simulação do modelo de Ising que descrevemos a seguir com 10000 spins, τ_{eq} é da ordem de 5000 MCS e τ é da ordem de 100 MCS.

A maior dificuldade computacional de realizarmos simulações de Monte Carlo é o fato que τ_{eq} cresce de forma exponencial com o tamanho do sistema, tornando proibitivo a simulação de sistemas com muitos spins.

O método de Monte Carlo foi desenvolvido para simular sistemas em equilíbrio, contudo em qualquer simulação de Monte Carlo temos de nos preocupar com a dependência temporal de grandezas como a energia e a magnetização do sistema. Uma questão natural para nos colocarmos é se a dependência temporal destas quantidades está relacionada com a dependência temporal dos sistemas físicos que queremos investigar. Ainda que não exista nenhuma prova formal, apenas indícios empíricos, acredita-se que o método de Monte Carlo é capaz também de caracterizar a dinâmica do sistema. Fisicamente a dinâmica do algoritmo de Monte Carlo pode ser pensada como sendo a dinâmica do próprio sistema, que tende a diminuir a energia, mas que devido a temperatura eventualmente pode aumentá-la. O peso w é uma estimativa realística para a probabilidade deste último evento ocorrer.

4.2.2.2. Modelo de Ising

O modelo de Ising é o modelo mais simples que apresenta uma transição de fase, ele foi inventado por W. Lenz em 1920 e foi resolvido por E. Ising em 1925 que resolveu o problema em 1d. Em 1944 Onsager [ONS 44] resolveu o modelo em 2d e foram necessários mais 45 anos para que Zamalodchikov resolvesse o modelo em 2d com a aplicação de um campo magnético [ZAM 89]. Até os dias de hoje não é conhecida sua solução em 3d. Sem sombra de dúvida, mais homens-horas foram investidos neste modelo do que em qualquer outro. As técnicas computacionais e estatísticas mais sofisticadas foram aplicadas na elucidação de suas propriedades. Apesar da importância dos resultados analíticos uma parte substancial de nosso conhecimento sobre ele foi obtida através de simulações de Monte Carlo.

Lenz propôs o modelo para descrever as propriedades de um ferromagneto, ou seja de um ímã. A sua premissa fundamental é a de que as propriedades magnéticas de um ferromagneto decorrem das interações entre os momentos de dipolo magnético dos átomos que compõe o material. O modelo assume uma rede, que pode possuir a geometria e o número de dimensões que quisermos, com um momento de dipolo ou spin em cada sítio. No caso mais simples todas as interações possuem a mesma intensidade J e a energia do sistema é dada por:

$$E = -J \sum_{\langle i\, j \rangle} S_i S_j - H \sum_i S_i \tag{4.14}$$

onde H é um campo magnético aplicado e $\langle i\, j \rangle$ significa que as somas são tomadas apenas entre primeiros vizinhos. Admitindo que J e H sejam positivos os spins tendem a se alinhar com o campo. Na fig. 4.2 representamos esquematicamente o modelo.

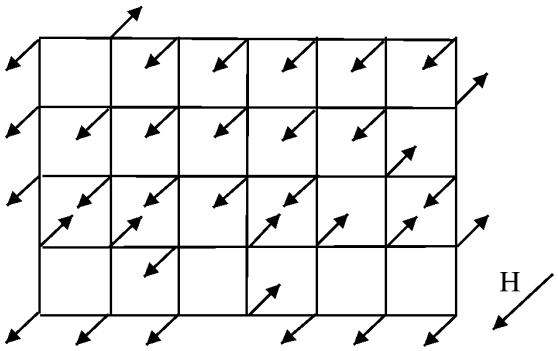

Figura 4.2: Representação esquemática do Modelo de Ising em 2d com um campo aplicado H. .

As quantidades de interesse do modelo de Ising são basicamente a energia, a magnetização por spin

$$M = \sum_i \langle S_i \rangle, \tag{4.15}$$

o calor específico

$$C = \frac{1}{T^2}(\langle E^2 \rangle - \langle E \rangle^2), \tag{4.16}$$

e a suscetibilidade

$$\chi = \frac{1}{T}(\langle M^2 \rangle - \langle M \rangle^2). \tag{4.17}$$

As médias $\langle \dots \rangle$ são chamadas de médias térmicas e são tomadas ao longo do algoritmo de Monte Carlo. A magnetização está relacionada com a imantação do ferromagneto, o calor específico mede a capacidade de absorver calor do sistema e finalmente a suscetibilidade χ mede como a magnetização varia quando aplicamos um campo magnético no sistema.

O maior interesse do modelo de Ising reside no fato de que ele possui uma transição de fase em duas dimensões ou mias dimensões para $T = T_c$. Em $2d$ temos que $T_c \sim 2.26$ para $H = 0$. Para temperaturas superiores a T_c a magnetização do sistema é nula enquanto que para temperaturas inferiores a magnetização admite valores finitos. O comportamento interessante do modelo ocorre nas proximidades de T_c, o calor específico e a suscetibilidade divergem nesta temperatura e a dinâmica do sistema se torna muito lenta. Os interesses atuais de pesquisa se concentram na caracterização do comportamento do sistema nesta região [BIN 92, NEW 99].

É importante observar que do ponto de vista das simulações só é possível simular sistemas pequenos, infinitamente menores do que sistemas experimentais. Para podermos comparar os resultados de nossa simulação com os experimentos, técnicas sofisticadas de análise foram desenvolvidas e permitem relacionar os resultados numéricos com os valores de algumas grandezas que podem ser medidas experimentalmente, estas metodologias são conhecidas como técnicas de reescalonamento finito. Infelizmente sua descrição foge do escopo deste curso e sugerimos a consulta do livro de Neumann e Barkema [NEW 99] ou para aqueles que preferem português a leitura da tese do autor [LEM 98].

4.2.3. Aplicações em biociências

O desenvolvimento veloz da genética e da bioquímica nas últimas décadas tem fornecido um campo importante de aplicações para novas técnicas de simulação. Para ilustrar uma das inúmeras aplicações das técnicas de Monte Carlo nestas áreas vamos investigar o modelo repton que modela a eletroforese de moléculas de DNA.

4.2.3.1. Eletroforese

Um problema tecnológico importante é a obtenção da seqüência de bases que compõe o DNA e obter assim o código genético de um determinado ser vivo. Parte vital deste processo é a separação de uma mistura de moléculas de DNA de diferentes tipos de acordo com sua massa. Para realizar esta separação nos valemos do fato que as moléculas de DNA são carregadas negativamente. O plano consiste em colocar estas moléculas em um gel e aplicar um campo elétrico. O gel pode ser visualizado como um líquido repleto de obstáculos em forma de varetas e a molécula de DNA como uma cobra (esta é a origem do nome do modelo) que se move através deste meio. Esta técnica é conhecida como eletroforese e vem sendo utilizada para separar não apenas DNA mas também outros polímeros carregados negativamente.

O procedimento experimental consiste em injetar moléculas de DNA no gel e aplicar um campo elétrico sobre o gel. Geometricamente o gel forma uma tira bastante fina. As moléculas começam a vagar através do gel e a se afastar do local onde foram injetadas. As moléculas mais extensas encontram mais resistência enquanto se movem através do gel e portanto se movem mais lentamente. Após esperar um intervalo de tempo, tipicamente algumas horas, os fragmentos de DNA mais curtos viajaram uma distância maior ao longo da tira. Ao longo do gel podemos encontrar diversas faixas, cada um contendo DNA com um determinado comprimento. Um resultado experimental está representado na fig. 4.3.

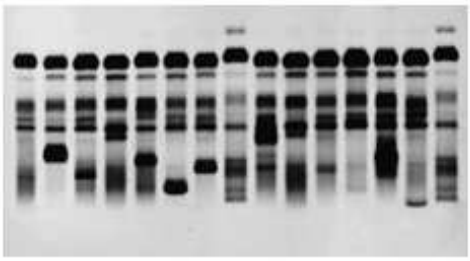

Figura 4.3: Eletroforese de uma mistura de fragmentos de DNA.

4.2.3.2. Modelo repton

O modelo repton foi criado para fornecer uma compreensão dos mecanismos físicos que governam a eletroforese. Particularmente gostaríamos de compreender a relação entre a velocidade de difusão v das moléculas e seu comprimento.

As moléculas de DNA se movem como cobras através da rede de obstáculos do gel. Os movimentos laterais da molécula são bloqueados pelo gel e o principal modo de movimento é ao longo da própria molécula. As pontas da molécula realizam movimentos aleatórios e o resto da molécula segue após a cabeça. O modelo repton foi introduzido por Rubinstein para descrever esta situação [RUB 87], neste curso tratamos uma versão simplificada do problema chamado modelo repton projetado descrito por [NEW 99]. Este modelo apesar de ser mais simples produz os mesmos resultados que o modelo original.

A molécula de DNA é representada como sendo formada por uma cadeia de N polímeros ou reptons. Ainda que o polímero real se mova no espaço tridimensional no nosso modelo estamos interessados apenas na posição dos polímeros ao longo do eixo x que é a direção do campo elétrico aplicado. Assim temos que associar a cada repton um único número, sua posição no eixo x. É conveniente representar o polímero em um gráfico indicando a posição de cada um dos reptons conforme esquematizado na fig. 4.4. A cada passo de tempo escolhemos um repton e tentamos movê-lo com uma determinada probabilidade ao longo do eixo x, devido a rigidez do polímero um único tipo de movimento é possível: se o repton possui um vizinho no mesmo nível e um vizinho em um nível abaixo ou acima, então ele pode subir ou descer um nível respectivamente. As pontas são mais livres podem subir ou descer desde que permaneçam no máximo um nível de distância de seus vizinhos. Na fig. 4.4 ilustramos os movimentos possíveis. Devido ao campo elétrico sempre aceitamos um movimento possível que aumente o nível de

um repton enquanto que um movimento que diminua o nível de um repton é aceito com probabilidade $\exp(-E/2)$, onde E é o campo elétrico aplicado.

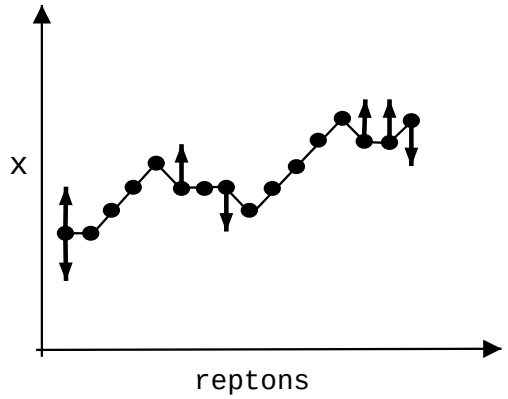

Figura 4.4: Visão esquemática do modelo repton. Os movimentos permitidos são indicados por setas.

Para mostrar que este modelo consegue explicar os mecanismos físicos da eletroforese mostramos na fig. 4.5 uma comparação entre resultados do modelo e dados experimentais. Na fig. temos representado a velocidade de difusão v contra o campo elétrico E, p é um parâmetro que descreve o gel. Os dados foram obtidos por Heller et al [HEL 94].

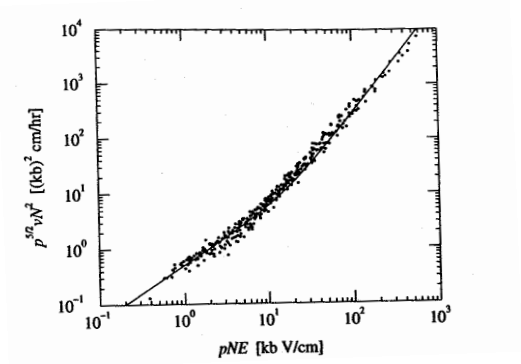

Figura 4.5: Dados experimentais em comparação com resultados do modelo repton. A curva cheia foi obtida através da análise dos resultados da simulação, enquanto que os pontos são experimentais. O gráfico mostra a dependência da velocidade de difusão v em função do campo elétrico aplicado E. A variável p está relacionada com propriedades do gel .

4.2.4. Algoritmos paralelos para técnicas de Monte Carlo

As simulações de Monte Carlo podem consumir muito tempo computacional, as simulações de Kawashima e Young [KAW 96] consumiram 1 ano de computação em um agregado de estações de trabalho em 1996. E mesmo assim ainda não foram conclusivas. O grande desafio é que τ_{eq}, o tempo para equilibrar o sistema, é muito longo para sistemas grandes. A solução para mitigar este custo computacional é executar as simulações em máquinas de arquitetura paralela.

Existem três classes de técnicas para criar um algoritmo paralelo: paralelização trivial, decomposição funcional e decomposição por domínio. Problemas trivialmente paralelizáveis são aqueles que podem ser divididos em tarefas idênticas que estão fracamente correlacionadas e que podem ser realizadas por programas independentes sendo executados em processadores isolados sem necessidade de comunicação inter-processos. Decomposição funcional se refere a dividir um problema em diferentes tarefas, sendo que cada tarefa deve ser executadas por um processador diferente. Ao final do processamento o resultado de cada uma das tarefas é integrado para se obter o resultado final. Decomposição por domínios significa dividir o domínio de aplicação do problema em regiões e fazer com que cada processador se dedique a uma região em particular [NEW 99].

Em simulações de Monte Carlo raramente usamos decomposição funcional, pois temos que calcular basicamente uma tarefa, um número gigantesco de vezes. Com as simulações de Monte Carlo ocorrem tipicamente em redes é bastante simples utilizarmos decomposição em domínios, ou ainda como temos de tomar médias sobre muitas amostras as simulações de Monte Carlo são problemas trivialmente paralelizáveis.

4.2.4.1. Algoritmos trivialmente paralelizáveis

O objetivo das simulações de Monte Carlo é produzir um grande número de configurações estatisticamente independentes do sistema de interesse. Nada impede que estas configurações sejam obtidas através de programas independentes rodando em processadores diferentes. Estas técnicas vem sendo utilizadas desde a idade da pedra da computação. Problemas deste tipo são chamados de trivialmente paralelizáveis ou até mesmo de embaraçosamente paralelizáveis. Newman e Barkema [NEW 99] chegam a afirmar que estas aplicações são ideais para computadores paralelos, o argumento básico é que devido ao baixo número de comunicação inter processos podemos atingir excelente desempenho, dificilmente igualado por qualquer outra aplicação.

Pessoalmente acredito que o baixo custo de máquinas paralelas construídas usando computadores pessoais e a facilidade de implementar algoritmos trivialmente paralelizáveis favorecerá o uso de técnicas de Monte Carlo em detrimento de outras técnicas numéricas que são menos performáticas nestas arquiteturas.

Eventualmente temos situações em que não podemos utilizar esta estratégia, um exemplo é a simulação de sistemas muito grandes, que precisem de mais memória que a disponível em cada máquina do agregado ou mesmo em casos que queremos simular uma única amostra que demorá muito tempo para atingir o equilíbrio. Nestas situações é mais vantajosos utilizar decomposição por domínios.

No apêndice 4.6.3. apresento uma implementação do modelo de Ising em 2 dimensões com condições de contorno periódicas usando a interface MPI e o método de Metropolis. Para facilitar a simulação todos os spins são armazenados em um vetor unidimensional e as matrizes `up`, `down`, `left` e `right` contém seus vizinhos. Várias cópias

do sistema são evoluídas separadamente, inicialmente até atingirem o equilíbrio, depois por mais t passos. A cada τ passos a magnetização e a energia são medidos. Na última etapa os resultados provenientes de todos os processos são somados e o valor da magnetização e da energia são calculados. Por comodidade estas quantidades são medidas para vários valores da temperatura.

4.2.4.2. Decomposição por domínios

Para facilitar a discussão sobre esta técnica de paralelização iremos nos concentrar na discussão do modelo de Ising em 2d. A idéia é dividir a rede quadrada em várias regiões conforme esquematizado na fig. 4.6 e fazer com que cada processador trabalhe em um domínio específico. Esta estratégia apresenta um problema toda a vez que tivermos de atualizar um dos spins da borda teremos que realizar uma comunicação entre os processadores, o que inviabilizaria a simulação. A solução para este problema é não atualizar os spins das fronteiras! Ainda que isto não viole a condição de balanço detalhado viola o requerimento que todos os estados do sistema sejam acessíveis. A solução do impasse é fazermos as fronteiras variarem ao longo do tempo. Isto nos garante que o sistema irá atingir o equilíbrio e que o custo computacional não será elevado.

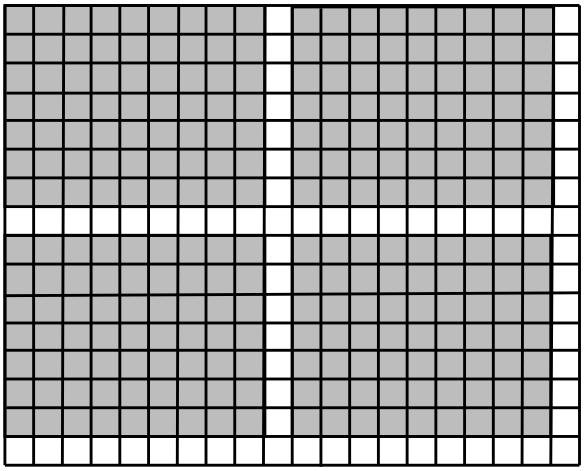

Figura 4.6: Representação dos diferentes domínios de uma simulação do modelo de Ising em 2 dimensões .

Existem ainda outras decomposições em domínios que generalizam os algoritmos de Wolff [WOL 89] e Swendsen-Wang [SWE 87]. Estes algoritmos dividem os spins em classes de spins fortemente correlacionados que são atualizados de forma simultânea. Generalizações destes algoritmos para arquiteturas paralelas são possíveis e estão descritos em [NEW 99].

Como balanço final sobre algoritmos paralelos para técnicas de Monte Carlo gostaria de afirmar apesar de já dispormos algoritmos eficazes estes ainda são meras generalizações de técnicas seqüenciais. Acredito que técnicas genuinamente paralelas deverão

surgir em um futuro próximo e que farão um uso mais criativo da arquitetura paralela.

4.3. Heurísticas genéticas

Em ciência e tecnologia nos deparamos muitas vezes com problemas complexos de otimização. O planejamento e a construção de qualquer artefato tecnológico implica na escolha de diversas variáveis com o objetivo de garantir o desempenho do artefato ou minimizar seu custo. A complexidade do espaço de busca destas soluções que incluem muitas variáveis sujeitas a vínculos inviabilizam o uso de técnicas de otimização tradicionais. Nesta situação temos de recorrer a heurísticas, que nada mais são do que uma adaptação sofisticada do universal método de tentativa e erro.

Uma inspiração importante para o desenvolvimento de heurísticas é a seleção natural que tem se mostrado capaz de gerar soluções intrincadas e eficazes em uma infinidade de situações. Tradicionalmente as heurísticas inspiradas na seleção natural tem sido nomeadas ¨Algoritmos Genéticos¨ desde o trabalho seminal de J. H. Holland [HOL 75]. Neste curso eu utilizo a denominação Heurística Genética pela singela razão de que o termo algoritmo é inadequado para descrever estas técnicas de otimização. O termo algoritmo é utilizado para designar procedimentos univocamente definidos que conduzam em tempo finito para a solução de um problema.

As heurísticas genéticas combinam a sobrevivência dos mais aptos com uma troca aleatória de informação entre membros de uma população composta por cadeias de caracteres. A cada geração um novo conjunto de criaturas, (cadeias de caracteres) é criada usando características dos elementos mais adaptados da anterior, isto permite um uso eficiente das informações contidas na população para especular novas possíveis soluções.

O poder destas heurísticas decorre diretamente da composição de dois processos: a mutação e o *crossover*. O primeiro é equivalente a um processo de busca localizada e permite que uma vez que as seqüências estejam próximas a um máximo local da função f, que queremos maximizar, elas convirjam para ele, já o *crossover* permite uma busca muito mais completa no espaço de parâmetros permitindo que máximos globais sejam encontrados.

É importante ressaltar que o algoritmo não pressupõe nenhum conhecimento sobre a função f e em principio o algoritmo está apto a encontrar ótimos para qualquer função f. Contudo a experiência tem mostrado que em muitos casos a convergência do método é muito lenta e técnicas mais sofisticadas devem ser empregadas. Um problema em aberto consiste em determinar quando as heurísticas genéticas são inadequadas e que modificações devem ser realizadas no paradigma central para torná-los eficientes nos casos mais difíceis.

Uma solução para diminuir o custo computacional é utilizar computadores de arquitetura paralela. Na seção 4.3.2. descrevemos algumas das heurísticas desenvolvidas para estas arquiteturas.

4.3.1. Operadores evolutivos

Suponhamos que queremos maximizar a função f que depende de vários parâmetros x_i que podem ser tanto números inteiros, reais ou variáveis binárias. Para simplificar a discussão supomos que f seja sempre positiva e denominamos f de fitness. O primei-

ro passo que adotamos é codificar todos os parâmetros x_i em uma única seqüência de caracteres binários, que a partir de agora será denominada indivíduo.

A evolução da heurística pode ser decomposta em diversos operadores: seleção, mutação, *crossover*, reescalonamento entre outros. Cada um dos operadores é praticamente independente dos demais o que garante uma boa modularidade para o problema, ou seja, sempre podemos adequar a heurística para o problema que queremos resolver através de uma escolha judiciosa dos operadores que iremos aplicar.

4.3.1.1. Evolução

A evolução consiste em escolher os indivíduos cujo material genético será transmitido para a próxima geração. Esta escolha deve depender do valor do fitness dos indivíduos. Mas temos uma grande variedade de opções. A alternativa clássica proposta por Goldberg [GOL 87] é conhecida como seleção de roda de roleta, inspirado uma vez mais no tradicional jogo de roleta. Nesta modalidade cada indivíduo é escolhido com uma probabilidade p_i dada por:

$$p_i = \frac{f_i}{F} \tag{4.18}$$

onde f_i é o fitness do indivíduo e F é a soma do fitness de toda a população. Esta sistemática pode ser facilmente implementada alocando para cada individuo uma região do intervalo $[0, 1]$ igual a p_i e sorteando um número aleatório, o individuo correspondente passa para a próxima etapa do processo. Na proposta inicial a cada geração N_{pop} (número de indivíduos da população) indivíduos eram selecionados, mas em uma variante popular chamada de estado fixo, uma porcentagem h de indivíduos é mantida de uma geração até a próxima.

Existem outros operadores possíveis para a seleção um dos mais populares é a seleção por torneios, onde N_{pop} torneios envolvendo dois indivíduos são realizados, saindo vencedor o indivíduo com maior fitness.

4.3.1.2. *Crossover* e mutação

Uma vez escolhidos os indivíduos que irão se reproduzir devemos gerar novos indivíduos a partir de seu ¨material genético¨. Dispomos de dois operadores a mutação e o *crossover*.

O *crossover* consiste em tomar dois indivíduos da população e com probabilidade p_{cross}, escolher um número aleatório $\bar{l}$ que pode assumir os valores 1, 2, ... , l e tomar os $\bar{l}$ primeiros bits de sua seqüência genética e os restantes $l - \bar{l}$ bits do outro indivíduo para formar uma nova seqüência.

Esta nova seqüência pode ainda sofrer mutações, ou seja, inversões aleatórias das variáveis binárias ao longo da seqüência. na fig. 4.7 esquematizamos estes processos.

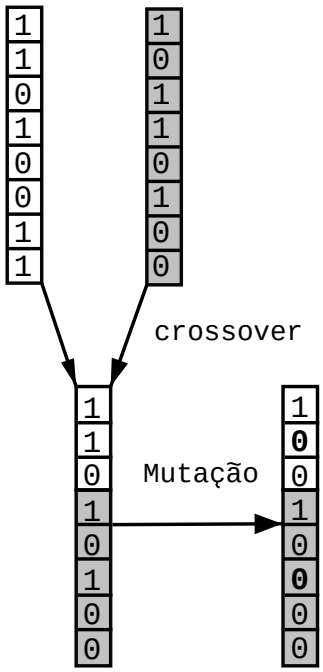

Figura 4.7: Representação esquemática da reprodução de um indivíduo em uma heurística genética.

Existem muitas variantes para estes dois processos. O *crossover* pode ser generalizado tomando porções não contíguas de cada um dos indivíduos pais. A mutação pode ser modificada adicionando um mecanismo de busca local além do processo de mutação como por exemplo, descida de gradiente ou pesquisa tabu.

4.3.1.3. Outros operadores

Ainda podemos aplicar outros operadores, entre eles o de rescalonamento da função de fitness com o objetivo de prevenir a perda de variabilidade nas gerações iniciais. Nas primeiras gerações é comum surgir um indivíduo com um fitness muito maior que os demais e que eventualmente irá dominar completamente a população, assim modificamos o fitness de forma a diminuir a diferença entre o melhor e o pior indivíduo [GOL 89].

Para evitar o colapso da população em torno de um único máximo local podemos utilizar o operador de super-população ou de divisão de fitness que diminui o fitness dos indivíduos se existem muito pouca variabilidade genética na população [JON 75, GOL 87].

4.3.1.4. Descrição de uma heurística genética

Para finalizar esta seção esquematizamos o funcionamento de uma heurística genética.

1. Inicie com uma população de N_{pop} indivíduos descritos por l bits escolhidos aleatoriamente.

2. Calcule a função de fitness para cada um dos indivíduos.

3. Repita cada um dos passos até que N_{pop} indivíduos tenham sido escolhidos

   (a) Selecione um par de indivíduos da população.

   (b) Aplique o operador de *crossover*.

   (c) Aplique o operador de mutação.

   (d) Coloque o indivíduo na nova população.

4. Substitua a população atual pela nova população.

5. Repita o passo 2 até que o critério de parada tenha sido atingido.

Nesta descrição omitimos a aplicação de outros operadores ao problema, para manter a discussão o mais simples e genérica possível.

4.3.2. Heurísticas genéticas em paralelo

As heurísticas genéticas seqüenciais mostraram sua eficácia em um sem número de aplicações. Mas em muitos casos os tempos necessários para atingir o ótimo global são demasiadamente longos. Neste tipo de situação uma alternativa importante a ser considerada é o uso de heurísticas genéticas paralelas. Alguns casos em que esta alternativa é vantajosa são listadas abaixo [NOW 99]:

- Em alguns casos para garantir que a convergência seja rápida é importante utilizarmos populações muito grandes, nestes casos pode ser impossível executar a simulação em uma única máquina.

- Se a determinação da função de fitness consumir muito tempo a distribuição do processo por vários processadores é vantajosa.

- Heurísticas genéticas seqüenciais podem ficar aprisionadas em máximos locais implementações paralelas podem percorrer de forma concorrente diferentes regiões do espaço de busca.

As heurísticas genéticas são versáteis e portanto existem diferentes formas de se construir uma implementação paralela, esta liberdade pode ser utilizada para tirarmos vantagem da particular arquitetura de que dispomos.

De acordo com Cantú-Paz [CP 97b] podemos classificar em quatro tipos básicos as implementações paralelas para estas heurísticas: paralelização global, paralelização de granularidade grossa, paralelização de granularidade fina e paralelização híbrida.

4.3.2.1. Paralelização global

Neste método mantemos uma única população, a aplicação dos operadores e a determinação da função de fitness é feita em paralelo. Da mesma forma que na formulação tradicional cada indivíduo compete com todos os demais e pode se reproduzir com qualquer outro.

A forma mais tradicional de paralelização é avaliar a função de fitness de forma paralela restringindo as comunicações apenas no envio de indivíduos aos diferentes processadores e no recebimento do valor do fitness.

Este modelo de paralelização não assume nada sobre a arquitetura computacional subjacente e pode ser implementado de forma eficiente tanto em máquinas de memória partilhada como de memória distribuída. No primeiro caso os processadores tem acesso aos indivíduos da população e a determinação do fitness pode ser feita sem que haja conflito entre os processadores. No segundo caso usamos uma arquitetura mestre-escravo, na qual o processo mestre se encarrega de distribuir as tarefas.

Este modelo foi aplicado com sucesso por Abramson e Abela [ABR] para procurar cronogramas eficazes para escolas.

4.3.2.2. Paralelização de granularidade grossa

Neste modelo de paralelização a população de indivíduos é dividida em subpopulações ou demes e os indivíduos podem migrar entre elas. A migração é crítica para este modelo, se esta é muito limitada os demes se comportam como populações isoladas e temos um ganho marginal com a implementação paralela, por outro lado se permitimos que os indivíduos migrem mais livremente atingimos valores mais elevados de fitness em um número menor de gerações, mas devido a problemas de comunicação o custo computacional pode ser maior nestes casos. Assim temos que estabelecer taxas de migração ótimas para cada caso.

Para implementar uma heurística neste modelo temos de escolher a topologia dos demes e a forma como a migração será realizada. Os demes podem estar todos conectados com todos (o que é conveniente se temos um agregado de computadores) ou ainda os demes podem estar organizados como um hipercubo, um anel ou ainda uma malha, neste caso a migração só pode ocorrer entre demes vizinhos. A escolha de uma particular topologia para os demes está diretamente relacionada com a topologia de nossa máquina paralela.

Temos ainda que definir de que forma se dará a migração, podemos escolher aleatoriamente os indivíduos que irão migrar ou ainda podemos escolher entre os melhores ou os piores, quantos deverão migrar e com qual periodicidade. Todos estes fatores devem ser escolhidos e podem ser determinantes no desempenho do algoritmo.

Este modelo foi aplicado em diferentes problemas: em particionamento de grafos [COH 91, TAL 91, LEV 94], na distribuição de cargas em computadores de arquitetura paralela [NEU 91, SER 94, MUN 91] e no desenho de circuitos [DAV 94].

4.3.2.3. Paralelização de granularidade fina

Neste modelo a população é dividida em muitos demes cada um deles contendo um pequeno número de indivíduos com intensa comunicação entre eles. Este modelo é especialmente adequado para computadores massivamente paralelos. Como estas arqui-

teturas estão bastante distantes de nossa realidade remeto os interessados para a referência [CP 97b].

4.3.2.4. Paralelização híbrida

Na paralelização híbrida mesclamos dois modelos anteriores para implementar nossa heurística. Neste caso temos que conviver com sistemas ainda mais complexos que os anteriores. Contudo devido ao surgimento de agregados formados por máquinas bi-processadas este modelo deverá se desenvolver para se adaptar a arquitetura disponível.

Um modelo promissor consiste em conjugar uma heurística de granularidade grossa com um modelo de paralelização global. Neste caso temos vários demes cuja função de fitness é avaliada de forma paralela, conforme esquematizado na fig. 4.8

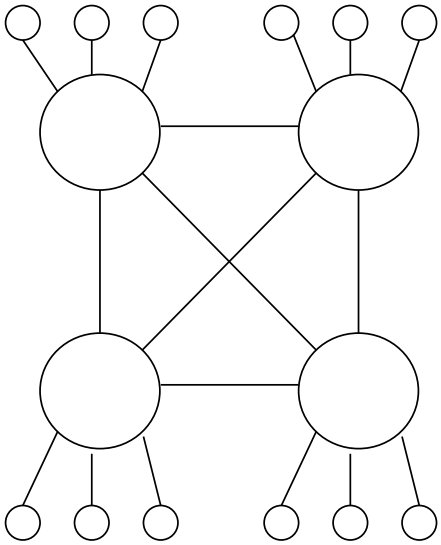

Figura 4.8: Modelo de paralelização mesclando uma paralelização global e uma de granularidade grossa. Os círculos maiores representam os demes enquanto que os círculos menores indicam que o cálculo do fitness é também feito de forma seqüencial.

Bianchinni e Brown [BIA 93] experimentaram com este modelo e mostraram experimentalmente que ele consegue encontrar soluções de mesma qualidade e em menos tempo que outras arquiteturas.

4.3.2.5. Perspectivas

O desenvolvimento de heurísticas genéticas para computadores de arquitetura paralela é uma área promissora, principalmente com o surgimento de agregados de computadores de baixo custo. A paralelização dos algoritmos é simples do ponto de vista da implementação, um desafio maior está na escolha adequada do modelo que devemos

empregar e na especificação dos parâmetros da heurística. Este último tópico não pode ser tratado nestas notas mas remetemos os leitores interessados para as referências [CP 97a, CP 97c, CP 99, CP 00].

4.3.3. Aplicações de heurísticas genéticas

Gostaríamos de finalizar nossa discussão sobre as heurísticas paralelas descrevendo uma aplicação desta técnica. Esta aplicação é sem dúvida a aplicação mais popular e foi desenvolvida por Karl Sims [SIM 94b, SIM 94a], o objetivo é evoluir criaturas virtuais formadas por blocos unidos através de juntas. As criaturas são desenvolvidas com um determinado objetivo, nadar, andar ou competir para quem consegue permanecer mais tempo próxima a um bloco quadrado. Na fig. 4.9 apresentamos algumas das criaturas obtidas por Karl.

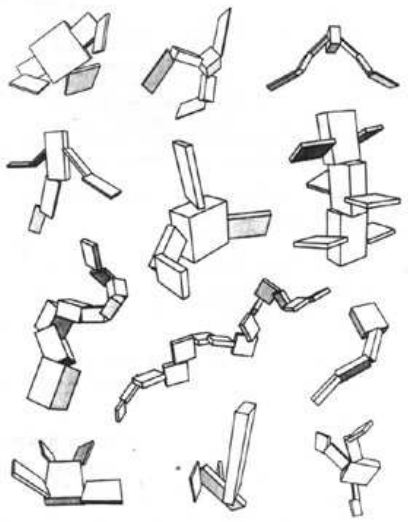

Figura 4.9: Criaturas virtuais evoluídas utilizando heurísticas genéticas.

Devido a popularidade do trabalho de Karl existem algumas páginas dedicadas a suas criaturas que merecem ser consultadas:

- `http://www.genarts.com/karl`
- `http://biota.org/ksims`
- `http://dynamics.org/%7Ealtenber/GA_ART/Sims.mpg` (Link para um filme mostrando as criaturas)
- `132.248.59.55/basilisc/basilisc.html` (veja um robô desenvolvido usando as mesma técnicas)

A morfologia destas criaturas foi obtida através de um algoritmo genético sofisticado que permitia evoluir não apenas os corpos das criaturas, mas também seus sistemas nervosos que incluíam sensores e controladores para os movimentos. As simulações também incluíam uma modelagem do meio ambiente onde se movimentavam as criaturas com inclusão de efeitos como a viscosidade da água, gravidade entre outros. Diferentes funções de fitness foram utilizadas uma para cada tipo de comportamento esperado, por exemplo, no caso de criaturas que nadavam e andavam a função de fitness estava relacionada com a velocidade.

As criaturas foram modeladas como grafos direcionados, o que permitia o surgimento de morfologias arbitrariamente complexas. Os vértices do grafo representavam as partes rígidas do corpo enquanto que as arestas representavam as juntas. Foram consideradas vários tipos de juntas com diferentes possibilidades de movimento, esféricas rígidas de revolução, etc.

O cérebro das criaturas era composto por sensores, por efetores e neurônios internos. Os primeiros eram de três tipos: sensores de junta que e indicavam o ângulo de cada junta, os sensores de contato que indicavam se duas partes do corpo estavam em contato e finalmente foto-sensores que reagiam a iluminação do ambiente. Os neurônios internos permitiam as criaturas possuir um comportamento arbitrariamente complexo. Eles eram capazes de realizar operações matemáticas como somas, produtos e cálculo de funções como seno e cosseno que permitiam interpretar uma determinada informação sensorial e enviar comandos motores para os efetores.

Devido ao fato de que a complexidade das criaturas pode ser arbitraria, os indivíduos são representados por um conjunto variável de parâmetros. O que inviabiliza o uso dos operadores tradicionais de *crossover* e mutação. A solução encontrada por Sims foi a de introduzir operadores de mutação que permitiam aumentar a complexidade dos indivíduos incluindo novas partes aos organismos; variar algum dos parâmetros, como o tipo de junta utilizada a operação realizada pelo neurônio; gerar ou retirar novas conexões e finalmente excluir partes.

O operador de *crossover* também teve de ser generalizado. Foram utilizados dois operadores um dos quais misturava aleatoriamente as partes de dois indivíduos e outro que levava uma parte do organismo e as partes ligadas a esta para o outro indivíduo.

Para realizar as simulações foi utilizada um computador massivamente paralelo chamado de Connection Machine CM-5 [2] usando um modelo de paralelização de granularidade fina. Uma das dificuldades encontradas foi que a determinação do fitness de determinados indivíduos tomava muito tempo, o que acabava por gerar processadores inativos por longos períodos. A solução encontrada foi desconsiderar estes indivíduos até que a avaliação de seu fitness estivesse concluída. Usando esta técnica, simular 100 gerações de 300 indivíduos demorava aproximadamente 3 horas na CM-5.

Os resultados obtidos são bastantes impressionantes. Diferentes técnicas de nado e locomoção foram desenvolvidas várias delas extremamente complexas e que seriam muito difíceis de serem implementadas por um programador humano. Muitas criaturas conseguiram realizar outras tarefas como perseguir um ponto luminoso ou disputar um cubo verde. Os resultados obtidos são uma evidência de que as heurísticas genéticas são um método poderoso para a descoberta de soluções inusitadas para problemas de otimização e que o uso de máquinas paralelas pode permitir a obtenção de bons resultados

[2] A máquina incluí 512 processadores com 16 gigabytes de memória, um sistema de discos com capacidade total de 140 gigabytes e velocidade de processamento de 64 gigaflops

com custos computacionais suportáveis.

4.4. Conclusões

Neste curso apresentamos duas técnicas de simulação que necessitam de processamento de alto desempenho e que permitem o desenvolvimento de implementações paralelas que são ao mesmo tempo simples de serem feitas e com boa performance.

Ainda que tanto para as heurísticas genéticas quanto para as técnicas de Monte Carlo já existam algoritmos eficazes, acredito que nos próximos anos devido a popularização dos agregados de computadores deveremos assistir a um grande avanço nesta área, em especial no desenvolvimento de algoritmos adequados a ambientes heterogêneos e compostos por máquinas com mais de um processador.

Os problemas tratados aqui são em sua maioria bastante simples mas já apresentam a estrutura básica de problemas de ponta em física, engenharia e biologia. As técnicas desenvolvidas para o seu tratamento podem ser estendidas com facilidade a problemas mais sofisticados. Assim acredito que este curso possa facilitar a comunicação entre os usuários de computação de alto desempenho e os especialistas em computação distribuída e paralela fornecendo um terreno comum onde novas técnicas possam ser geradas e aplicadas em problemas de relevância tecnológica e científica.

4.5. Agradecimentos

Este trabalho foi parcialmente financiado pelo Conselho de Desenvolvimento Científico e Tecnológico através do projeto no.469445/2000-9.

4.6. Apêndices

4.6.1. Números aleatórios

A geração de números aleatórios é o coração de qualquer simulação de Monte Carlo. Existem diferentes algoritmos que podem ser utilizados entre eles podemos citar os algoritmos elencados no clássico “Numerical Recipes” [PRE 93]. Ainda que eu pessoalmente tenha utilizado estes algoritmos em inúmeras aplicações, recentemente resolvi adotar uma biblioteca mais adequada para simulação de Monte Carlo em máquinas de arquitetura paralela. A escolha recaiu sobre a biblioteca SPRNG (*Scallable Parallel Random Number Generator* que pode ser obtida em `http://sprng.cs.fsu.edu/`) desenvolvida pela universidade estadual da Florida que fornece uma biblioteca que garante números aleatórios com boas propriedades estatísticas (longos períodos, independência, etc), reprodutiveis (simulações podem ser reproduzidas facilmente), portabilidade (existem versões para inúmeras arquiteturas) e finalmente o código é aberto e pode ser estendido.

As bibliotecas incluem os algoritmos: *bit Linear Congruential, 64 bit Linear Congruential, modified Additive Lagged Fibonacci, Multiplicative Lagged Fibonacci, Combined Multiple Recursive* e *Prime Modulus Linear Congruential*. Para uma revisão dos

diferentes algoritmos consulte a documentação da biblioteca. Como última vantagem cito que a biblioteca possui interfaces para C, Fortran e para a interface MPI (*Message Passing Interface*). Para rodar os exemplos destas notas esta biblioteca deve estar instalada em seu sistema.

4.6.2. Cálculo de Integrais

```
//=========================================================
// Calcula a integral de uma funcao em (a,b)
//         usando técnica a de Monte Carlo
// Ney Lemke
// Data: 29/09/2001
//=========================================================
/* para compilar use por exemplo
mpiCC -O3 -DLittleEndian  -DSPRNG_MPI -DINTEL
      -I/usr/local/mpi/include
      -I/usr/local/include -o mcmpi  mcmpi.cpp  -lsprng
      -L/usr/local/lib -lmpi
   para rodar use
mpirun -np 4 mcmpi
*/
#include <iostream.h>    // Cabecalho padrao do C++
#include <math.h>        // Cabecalho de funcoes matematicas
#include "mpi.h"         //
#include <stdlib.h>      // Cabecalho padrao do C++
#define SIMPLE_SPRNG
// Cabecalhos do gerador de numeros aleatorios
#define USE_MPI   // Nao modifique
#include "sprng.h"
#define SEED 985456376   // semente dos numeros aleatorios
// Calcula Integral
void mcmpi( int numProcs, int myRank, int num_random);
double f( double);        // define funcao
int main( int argc, char* argv[] ) {
int myRank, numProcs;
// Rank do processo e numero de processos
MPI_Init(&argc,&argv);     // Linhas tradicionais do MPI
MPI_Comm_rank( MPI_COMM_WORLD,&myRank );
MPI_Comm_size( MPI_COMM_WORLD, &numProcs );
// Informacoes essencias para rodar a simulacao
int num_random;
if (myRank == 0)num_random = 10000000;
// Envia o numero de pontos aleatorios para os processos
MPI_Bcast(&num_random, 1, MPI_INT, 0, MPI_COMM_WORLD);
mcmpi( numProcs, myRank, num_random);
MPI_Finalize();
return 0;
}
```

```
// mcmpi
void mcmpi( int numProcs, int myRank, int num_random) {
    double integral = 0;        // valor da integral
    double myintegral = 0;      // valor da integral local
    double ran;                 // numero aleatorio
    double lim_inf=0;      // limite superior
    double lim_sup=1;      // limite inferior da integral
    int i;
    int gtype=0;
// Algoritmo para gerar numeros aleatorios: 0,1,2,3,4,5
    init_sprng(gtype,SEED,SPRNG_DEFAULT);
    // inicializa corrente de n. aleatorios
    for (i=0; i< num_random; i++){
    ran = (lim_sup-lim_inf)*sprng()+lim_inf;
    myintegral += f(ran);
    }
    // Calcula a soma das variaveis myintegral e retorna
    // o valor na variavel integral no processo 0
    MPI_Reduce(&myintegral, &integral, 1, MPI_DOUBLE,
               MPI_SUM, 0, MPI_COMM_WORLD);
    if (myRank == 0) {
       cout << "o valor da integral de  f(x) e  = "
            << integral/num_random/numProcs
            << endl;
    }
}
// A funcao que deve ser integrada
double f( double x) {
   return x*x;
}
```

4.6.3. Modelo de Ising

```
//=========================================================
//    Simulacao do Modelo de Ising em 2d
//    Ney Lemke
//    16/10/2001
//=========================================================
#include <iostream.h>    // Cabecalho padrao do C++
#include <math.h>        // Cabecalho de funcoes matematicas
#include <mpi.h>         // Cabecalho para o mpi
#include <stdlib.h>      // Cabecalho padrao do C++
#include <time.h>
#define SIMPLE_SPRNG     // Cabecalhos e variaveis do gerador
                         // de numeros aleatorios
#define USE_MPI          // Nao modifique
#include <sprng.h>       //
void evolui(int n); // evolui a rede n passos
```

```
void atualiza(void);     // atualiza um unico spin
void inicializa(double tempe, double h);
// inicializa a rede e as probabilidades para uma determina-
da temperatura
void inicializa_geral(void);
// Inicializa variaveis comuns a todas as simulacoes
//=========================================================
//                  Declaracao Constantes
//=========================================================
#define LSIZE             20 // Dimensao do Sistema
#define L2                (LSIZE*LSIZE) // Number of sites
#define TEMPEI             2.0 // Temperatura Inicial
#define TEMPEF             3.0 // Temperatura Final
#define TEMPED             0.1 // Incremento de Temperatura
#define H                 0.0 // Campo Magnético
#define TIME              100 // Tempo de Medida
#define TAU               10  // Frequencias das medidas
#define TAUEQ             100  // Tempo de equilibrio
int right[L2],left[L2], // Vizinhos dos sitios
    up[L2],down[L2];
double prob[9][1];       // Probabilidades de inversao
int s[L2];                    // configuracao
int  my_m;                    // magnetizacao local
double my_E;                  // energia local
// Programa principal
int main( int argc, char* argv[] ) {
int myRank, numProcs;
// Rank do processo e numero de processos
int seed;
// semente dos numeros aleatorios
int namelen;
// comprimento do nome do processador
char processor_name[MPI_MAX_PROCESSOR_NAME];
// nome do processador
int m;                 // magnetizacao media
double E;                // energia media
double my_mediaE=0.;
// energia media em cada processador
int  my_mediam=0;
// magnetizacao media em cada processador
int i,j;
double tempe;           // temperatura atual
MPI_Init(&argc,&argv);     // Linhas tradicionais do MPI
MPI_Comm_rank( MPI_COMM_WORLD,&myRank );
MPI_Comm_size( MPI_COMM_WORLD, &numProcs );
MPI_Get_processor_name(processor_name, &namelen);
// obtemos nome do processador
cout << "#Processo " << myRank << " rodando no processador "
```

```
     << processor_name << "\n";
int gtype=0;
// Algoritmo para gerar numeros aleatorios: 0,1,2,3,4,5
seed = 2*(int) time(NULL) + 1;
        /* inicializacao dos numeros aleatorios */
init_sprng(gtype,seed,SPRNG_DEFAULT);
inicializa_geral();          // inicializa matrizes de vizinhos
// inicializa corrente de n. aleatorios
 for(tempe=TEMPEI; tempe<TEMPEF; tempe+=TEMPED){
   if(myRank!=0){
     inicializa(tempe, H);
       // inicializa estado inicial e matriz de probabilidades
     evolui(TAUEQ);               // equilibra o sistema
     my_mediaE=0.;
     my_mediam=0;
     for (i=0; i<TIME*L2; i+=TAU){
       evolui(TAU);
     // evolui o sistema alguns passos
       my_mediaE+=my_E;            // atualiza medias
       my_mediam+=my_m;
     }
   }
 // Linhas que executam a soma de todas as variaveis
 // locais e produzem
 // a variavel media
   MPI_Reduce(&my_mediaE, &E, 1, MPI_DOUBLE,
              MPI_SUM, 0, MPI_COMM_WORLD);
   MPI_Reduce(&my_mediam, &m, 1, MPI_INT,
              MPI_SUM, 0, MPI_COMM_WORLD);
    // So o processo mestre escreve os resultados
   if (myRank==0){
     // Para facilitar o uso dos dados imprimimos os valores da
     // temperatura, E e a magnetizacao
     cout << tempe << "\t"
          << E/(TIME/TAU)/(L2*L2)/(numProcs-1) << "\t"
          << ((double) m)/(TIME/TAU)/(L2*L2)/(numProcs-1)
          << "\n";
   // Temos que normalizar corretamente os resultados
   }
 }
MPI_Finalize();
return 0;
}
// atualiza um unico spin
void atualiza(void){
int  site;
int  spin;
int s1;
```

```
 site = (int) (sprng()*L2); // sitio tomado aleatoriamente
 s1 =   s[right[site]]  // soma entre os primeiros vizinhos
     + s[left[site]]
     + s[down[site]]
     + s[up[site]];
s1 *= s[site];
// inverte o spin
 if (sprng() < prob[s1+4][(s[site]+1)/2]) {
           s[site] *= -1;
             my_m+=2*s[site];
             my_E-=2*s1;
}
return;
}
// inicializacao da matriz de vizinhos
void inicializa_geral(){
int i,j;
for (i=0;i<L2;++i)   // sítios vizinhos: condicoes de
                     // contorno periodicas
    {
    if (i % LSIZE==LSIZE-1) right[i] = i - LSIZE + 1;
                            else right[i] = i+1;
    if (i % LSIZE==0) left[i] = i + LSIZE - 1;
                       else left[i] = i-1;
    if (i<LSIZE) up[i] = L2 - LSIZE + i;
                 else up[i] = i - LSIZE;
    if (i>=L2-LSIZE) down[i] = (i % LSIZE);
                     else down[i] = i + LSIZE;
    }
return;
}
void inicializa(double temp, double h){
int i,j;
double tmp;
 for (i=0;i<9;i+=2){
     for (j=0;j<2;j++){
             tmp = (i-4)*1.0+ (2*j-1)*h;
             if (tmp <= 0.0) prob[i][j]= 1.0;
             else prob[i][j] = exp(-2.*tmp/temp);
             // fator de metropolis
         }
     }
//iniciamos sempre no estado ordenado
my_m=L2;
my_E=2.*L2+L2*h;
for (i=0; i<L2; ++i)
   {
    s[i] = 1;
```

```
   }
return;
}
void evolui (int n){
    int i;
    for (i=0;i<L2*n;i++){
      atualiza();
    }
}
```

4.7. Bibliografia

[ABR] ABRAMSON, D. A parallel genetic algorithm for solving the school timetable problem. In: PROCEEDINGS OF THE FIFTEENTH AUSTRALIAN COMPUTER SCIENCE CONFERENCE, In: PROCEEDINGS OF THE FIFTEENTH AUSTRALIAN COMPUTER SCIENCE CONFERENCE. [s.n.], 1992.

[BIA 93] BIANCHINNI, R. Parallel genetic algorithms on distrubet-memory architectures. In: Atkins, S., editors, TRANSPUTER RESEARCH AND APPLICATIONS 6, 1993. **Proceedings...** Amsterdan:IOS Press, 1993. p.67.

[BIN 92] BINNEY, J. J. et al. **The theory of critical phenomena**. Oxford:Oxford Science Publications, 1992.

[COH 91] COHOON, J. P.; MARTIN, W. N. A multi-population genetic algorithm for solving the k-partition problem on hypercubes. In: Belew, R. K., editors, PROCEEDINGS OF THE FOURTH INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS, 1991. **Proceedings...** San Mateo:Morgan Kauffmann Publishers, 1991.

[CP 97a] CANTÚ-PAZ, E. Modeling idealized bounding cases of parallel genetic algorithms. In: Koza, J. et al., editors, GENETIC PROGRAMMING 1997: PROCEEDINGS OF THE SECOND ANNUAL CONFERENCE, 1997. **Proceedings...** San Francisco:Morgan Kaufmann, 1997. p.353.

[CP 97b] CANTÚ-PAZ, E. A survey of parallel genetic algorithms. Illinois Genetic Algorithms Laboratory, 1997. Relatório técnico.

[CP 97c] CANTÚ-PAZ, E. Predicting speedups of ideal bounding cases of parallel genetic algorithms. In: Bäck, T., editor, PROCEEDINGS OF THE SEVENTH INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS (ICGA97), 1997. **Proceedings...** , CA:Morgan Kaufmann, 1997.

[CP 99] CANTÚ-PAZ, E. **Designing efficient and accurate parallel genetic algorithms**. Urbana, IL:Univ. Illinois, Urbana-Champaign, 1999. Tese de Doutorado.

[CP 00] CANTú-PAZ, E. **Efficient and accurate parallel genetic algorithms (Genetic Algorithms and Evolutionary Computation Volume 1)**. New York:Kluwer Academic Publishers, 2000.

[DAV 94] DAVIS, M.; LIU, L. Vlsi circuits circuit syntesis using a parallel genetic algorithm. In: Schaffer, J. D.; Schwefel, H. P., editors, PROCEEDINGS OF THE FIRST IEEE CONFERENCE ON EVOLUTIONARY COMPUTATION, 1994. **Proceedings...** Piscataway:[s.n.], 1994. v.2, p.104.

[ECK 87] ECKHARD, R. Stan Ulam, John von Neumann and the Monte Carlo method. **Los Alamos Science**, [S.l.], p.15, 1987.

[GOL 87] GOLDBERG, D. E. Genetic algorithms with sharing for multimodal sharing functions. In: Grefenstette, J. J., editor, GENETIC ALGORITHMS AND THEIR APPLICATIONS: PROCEEDINGS OF THE SECOND INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS, 1987. **Proceedings...** Erlbaum:[s.n.], 1987.

[GOL 89] GOLDBERG, D. E. **Genetic algoritms in search, optimization and learning**. USA:Addison-Wesley, 1989.

[HEL 94] HELLER, C.; DUKE, T. A.; VIOVY, J. L. Electrophoretic mobility of DNA in gels. I **Biopolymers**, [S.l.], v.25, p.431, 1994.

[HOL 75] HOLLAND, J. H. **Adaptation in natural and artificial systems**. Massachussets:MIT Press, 1975.

[HUA 87] HUANG, K. **Statistical mechanics**. John Wiley and Sons, 1987.

[JON 75] JONG, K. A. D. **An analisys of the behavior of a class of genetic adaptative systems**. University of Michigan, 1975. Tese de Doutorado.

[KAW 96] KAWASHIMA, N. Phase transition in the three-dimensional $\pm j$ Ising spin glass. **Phys. Rev B**, [S.l.], v.53, p.R484, 1996.

[LEM 98] LEMKE, N. **Simulação numérica de vidros de spin e redes neurais**. Instituto de Física – UFRGS, 1998. Tese de Doutorado.

[LEV 94] LEVINE, D. A parallel genetic algorithm for the set partioning problem. Argone National Laboratory, Mathematicas and Computer Science Division, 1994. Relatório técnico.

[MET 87] METROPOLIS, N. The beginning of the Monte Carlo method. **Los Alamos Science**, [S.l.], p.15, 1987.

[MUN 91] MUNTEAN, T. A parallel genetic algorithm for process-processor mapping. In: Durand, M., editors, PROCEEDINGS OF THE SECOND SYMPOSIUM II. HIGH PERFORMANCE COMPUTING, 1991. **Proceedings...** Amsterdã:[s.n.], 1991. p.71.

[NEU 91] NEUHAUS, P. Solving the mapping problem - experiences with a genetic algorithm. In: Schwefel, H.-P., editors, PARALLEL PROBLEM SOLVING FROM NATURE, 1991. **Proceedings...** Berlim:Springer-Verlag, 1991. p.170.

[NEW 99] NEWMAN, M. E. J. **Monte Carlo methods in statistical physics**. Oxford:Clarendon Press, 1999.

[NOW 99] NOWOSTAWSKI, M. Review and taxonomy of parallel genetic algorithms. Information Science Departament of the University of Otago, 1999. Relatório técnico.

[ONS 44] ONSAGER, L. The first solution of the d=2 Ising model. **Phys. Rev.**, [S.l.], v.65, p.117, 1944.

[PLL] PLLANA, S. History of Monte Carlo method. http://www.geocities.com/CollegePark/Quad/2435/index.html. Acessado em 5/12/2001.

[PRE 93] PRESS, W. H. et al. **Numerical recipes in C : The art of scientific computing**. Cambridge Univ Press, 2. ed., 1993.

[REI 98] REICHL, L. **A modern course in statistical physics**. Wiley-Interscience, 1998.

[RUB 87] RUBINSTEIN, M. Discretized model of entangled-polymer dynamics. **Phys. Rev. Lett.**, [S.l.], v.59, p.1946, 1987.

[SER 94] SERENDYNSKI, F. Dynamic mapping and load balancing with parallel algorithms. In: Schaffer, J. D.; Schwefel, H. P., editors, PROCEEDINGS OF THE FIRST IEEE CONFERENCE ON EVOLUTIONARY COMPUTATION, 1994. **Proceedings...** Piscataway:[s.n.], 1994. v.2, p.834.

[SIM 94a] SIMS, K. Evolving 3d morphology and behavior by competition. In: Maes, B., editor, ARTIFICIAL LIFE IV PROCEEDINGS, 1994. MIT Press, 1994. p.28.

[SIM 94b] SIMS, K. Evolving virtual creatures. In: COMPUTER GRAPHICS (SIGGRAPH '94 PROCEEDINGS), 1994. [s.n.], 1994. p.15.

[STE 01] STERLING, T. How to build a hypercomputer. **Scientific American**, [S.l.], p.28, 2001.

[SWE 87] SWENDSEN, R. H. Non-universal critical dynamics in Monte Carlo simulations. **Phys. Rev. Lett.**, [S.l.], v.58, p.86, 1987.

[TAL 91] TALBI, E.-G. A parallel genetic algorithm for the graph partioning problem. In: PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON SUPERCOMPUTING, 1991. **Proceedings...** Cologne:[s.n.], 1991.

[ULA 47] ULAM, S.; RICHTMEYER, R. D. Statistical methods in neutron diffusion. Los Alamos National Laboratory, 1947. Relatório técnico.

[ULA 49] ULAM, S. The Monte Carlo method. **Journal of American Statistical Association**, [S.l.], v.44, p.335, 1949.

[WOL 89] WOLFF, U. Collective Monte Carlo updating for spin systems. **Phys. Rev. Lett.**, [S.l.], v.62, p.361, 1989.

[ZAM 89] ZAMALODCHIKOV, A. B. Solution of the d=2 Ising model with non-zero external magnetic field. **Int. J. Mod. Phys.**, [S.l.], v.A4, p.4235, 1989.