Placement por simulated annealing em um cluster de estações de trabalho.

Jonas Knopman[1] e Júlio Salek Aude[2]

Universidade Federal do Rio de Janeiro

Resumo

A resolução de problemas de otimização combinatória com base no método de Simulated Annealing mostrou produzir resultados de extrema qualidade mas, tipicamente, a um custo de processamento muito alto. Visando reduzir este custo diversas soluções têm sido propostas com o uso de hardware dedicado, processadores velozes ou explorando possibilidades de paralelizar o algoritmo. Neste artigo é apresentada uma implementação paralela do algoritmo de simulated annealing visando resolver o problema do placement de células no projeto de circuitos impressos ou de circuitos VLSI. É mostrado que a escolha do mecanismo de paralelização é influenciada pela temperatura. Em particular introduz-se a idéia do uso de estratégias adaptativas que, dinamicamente, mudam o esquema de partição do problema visando obter o máximo de desempenho. O algoritmo foi implementado em um cluster de workstations usando PVM (Parallel Virtual Machine) formando assim uma máquina paralela virtual com comunicação através de troca de mensagens. O algoritmo seqüencial é apresentado juntamente com algumas técnicas de paralelização. São apresentadas medidas extensivas da qualidade dos algoritmos propostos bem como são levantados uma série de problemas, alguns dos quais levaram a modificações dos algoritmos inicialmente testados enquanto outros mantêm-se ainda não resolvidos.

Abstract

The solution of combinatorial optimization problems based on simulated annealing algorithms have been shown to produce results of extremely high quality, but typically at a very high cost in processing time. Aiming to reduce this cost several solutions have been proposed using dedicated hardware, fast processors or exploring different approaches of parallelization. This paper presents an annealing application - placement of cells in the design of printed circuits or VLSI circuits - and develops parallel annealing algorithms to solve it. It is shown that the choice of the parallelization strategy is influenced by temperature. In particular, the paper introduces the idea of adaptive strategies that dynamically change the parallel decomposition scheme to achieve maximum performance. The algorithm was implemented in a cluster of workstations under PVM (Parallel Virtual Machine) communicating through message passing. The serial annealing algorithm is shown in addition to a detailed explanation of the parallel algorithms. Extensive measurements of efficiency are presented as well as a discussion of the problems found that have led to modifications in the original algorithm or that remain still unsolved.

[1] MsC Pesquisador; E-mail: jonas@nce.ufrj.br. NCE/UFRJ, Caixa Postal 2324, Rio de Janeiro, RJ, CEP 20001-970

[2] PhD Pesquisador NCE/UFRJ; Adjunto IM/UFRJ; E-mail: salek@nce.ufrj.br

1. Introdução

Simulated annealing (SA) foi primeiro proposto por Kirkpatrik et al. [5] para a solução de problemas complexos de otimização combinatória e se mostrou eficiente para um conjunto grande de aplicações [4, 7]. Sua principal característica é a possibilidade de explorar o espaço de estados do problema permitindo movimentos *hill climbing*. A desvantagem desta solução probabilística é o tempo necessário para atingir uma solução próxima da ótima.

Algumas tentativas de paralelizar o algoritmo SA podem ser encontradas na literatura para máquinas de memória compartilhada [1] ou distribuída [2]. Na maioria dos casos a paralelização é implementada ao nível dos dados. Os dados descrevendo o problema são divididos em pequenos subconjuntos e distribuídos entre os processadores que realizam de forma concorrente o SA seqüencial.

O algoritmo SA não é estático: o parâmetro temperatura que controla os movimentos de *hill-climbing* altera profundamente o comportamento do algoritmo durante a sua execução. Este artigo mostra que o comportamento dinâmico do algoritmo fornece novas oportunidades de explorar o paralelismo inerente ao processo. A estratégia de paralelização empregada consiste em mudar o algoritmo usado de acordo com a fase do processo. O objetivo é construir um algoritmo paralelo eficiente, fornecendo uma solução de qualidade igual ao do algoritmo seqüencial.

Uma das estratégias de paralelização empregadas consiste em manter vários processadores trabalhando simultaneamente na avaliação de uma única cadeia de Markov preservando, dessa forma, as propriedades de convergência do algoritmo seqüencial. Para aplicações típicas de SA cerca de 90% de todos os movimentos gerados são rejeitados. A observação de que os movimentos rejeitados são independentes uns dos outros leva a uma implementação direta do método, uma vez que esses movimentos podem ser rejeitados independentemente em cada um dos processadores. Os movimentos aceitos levam a uma sincronização de todos os processadores envolvidos no cálculo da cadeia de Markov. Especialmente no início do algoritmo um número muito grande de movimentos é aceito levando a uma eficiência muito baixa.

Um outro tipo de estratégia consiste em cadeias de Markov independentes, uma computada por cada processador. Como veremos adiante esta é uma estratégia adequada ao regime de temperaturas altas. O número de movimentos aceitos fornece a indicação do momento ideal de trocar as estratégias.

Todos os algoritmos aqui descritos foram implementados em um cluster de workstations rodando PVM (Parallel Virtual Machine) [3]. PVM é um ambiente onde uma coleção heterogênea de computadores seriais, paralelos e vetoriais interligados em rede funcionam como se constituíssem um único recurso computacional.

2. Definição do problema

Fisicamente, o projeto de um circuito impresso ou de um circuito integrado envolve o posicionamento de um conjunto de módulos eletrônicos em uma placa ou substrato. Os elementos do circuito existentes nos módulos levam à existência de pinos de conexão nas bordas dos módulos. Vários subconjuntos destes pinos devem ser interconectados para formar *redes*. Uma vez que os módulos contêm vários pinos, eles, geralmente, pertencem a mais de uma rede. Formalmente, o problema de *placement* consiste em encontrar a localização ótima dos módulos de modo a minimizar uma função custo segundo alguma norma.

Na prática, existem uma série de requisitos (geralmente conflitantes) que devem ser minimizados no projeto de um circuito eletrônico: o comprimento da fiação, ruído, dissipação de calor, área, etc. Entretanto, acima de tudo, o posicionamento utilizado deve prover fácil roteamento dos sinais para formar as redes.

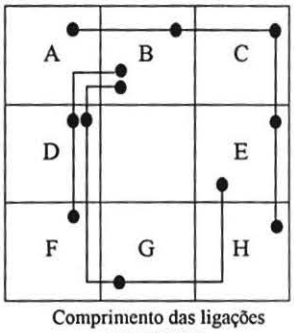
Em termos práticos, é impossível incorporar todos os objetivos de projeto em uma única função custo. Entretanto, minimizar o comprimento total da fiação parece ser uma aproximação razoável dos demais objetivos de projeto. De fato, circuitos com ligações curtas entre pinos têm, em geral, baixa dissipação de calor, pequena área e baixo nível de ruído.

Finalmente, devemos notar que a distância usada para medir o custo de um *placement* é a distância retilínea ou distância *Manhattan*, isto é, se (x_1, y_1) e (x_2, y_2) são dois pontos no plano a distância retilínea d é dada por:

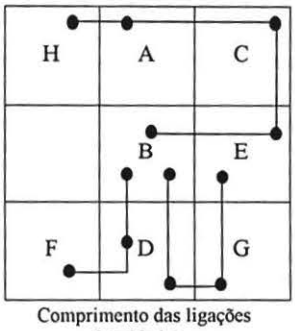
$$d = |x_1 - x_2| + |y_1 - y_2|$$

Seja o seguinte exemplo composto por três redes[1] S_1, S_2 e S_3 e oito módulos A, B,..., H.
$S_1 = \{A, B, C, E, H\}$
$S_2 = \{B, D, E, G\}$
$S_3 = \{B, D, F\}$

A figura a seguir mostra duas alternativas para o *placement* dos módulos e a diferença resultante na função custo



Comprimento das ligações
13 unidades



Comprimento das ligações
9 unidades

3 - Simulated Annealing

3.1 - Introdução:

Quando um pedaço de metal é moldado a estrutura cristalina interna é alterada. Uma parte da energia empregada para moldar o metal é absorvida pelo material sobre a forma de imperfeições na cadeia cristalina. A nova estrutura é, tipicamente, mais sujeita a fraturas. Dessa forma, se for necessário moldar o metal ainda mais, a estrutura original deve ser recuperada. Isso é feito aquecendo-se o material a uma temperatura próxima à metade do seu ponto de fusão e mantendo-o nesta temperatura por um certo tempo. O metal é então resfriado lentamente. A energia térmica em altas temperaturas e o

[1]Em problemas reais o número dos pinos deveria aparecer ao lado da identificação dos módulos, i.e., A_1, B_5 e H_7. Para fins de clareza, a numeração dos pinos é omitida neste exemplo e as redes são definidas como conexões de módulos e não de pinos.

resfriamento lento possibilitam aos átomos do metal caminhar em direção à configuração de energia mínima, nominalmente cristais perfeitos. Os métodos computacionais, em analogia, permitem, na fase inicial do algoritmo, que os parâmetros do problema variem de uma grande quantidade (simulando os efeitos de alta temperatura). O parâmetro controlando a quantidade de alteração é freqüentemente chamado de T ou temperatura. Nas fases seguintes do algoritmo (temperaturas mais baixas) reduz-se a variação permitida para os parâmetros do problema. A idéia é que em altas temperaturas evite-se que a solução caia num mínimo local. Então, uma vez no entorno da solução global, as fases de baixa temperatura permitem o refinamento da solução.

O algoritmo SA pode ser visto como um algoritmo probabilístico de busca por um mínimo global. Ao contrário dos métodos determinísticos de *hill climbing*, configurações que piorem o custo também podem ser aceitas. Partindo de uma configuração inicial, o algoritmo repetidamente gera novas configurações vizinhas. Configurações que contribuam para diminuir a função custo são aceitas. Configurações que aumentem a função custo de uma quantidade ΔC são aceitas com probabilidade $e^{-\Delta C/T}$. Valores altos de T levam à aceitação de grandes deteriorações na função custo ao passo que, em baixas temperaturas, somente configurações melhores são aceitas. Em cada uma das temperaturas devem ser gerados estados suficientes para atingir o equilíbrio térmico.

SA não é um algoritmo propriamente dito [6]. Ele é melhor definido como um paradigma para se construir algoritmos para a solução de problemas específicos de otimização combinatória. Para os nossos propósitos, o projeto de um algoritmo de annealing compreende 5 partes:

1. **Espaço de estados**: A representação dos estados possíveis para o sistema deve ser simples e facilitar a fácil geração de perturbações.
2. **Geração de novos estados:** As regras para a geração de novos estados devem ser flexíveis o bastante para permitir que qualquer solução seja atingida a partir de transformações no estado inicial. Além disso, os novos estados têm de ser gerados a um custo relativamente baixo uma vez que muitos deles têm de ser gerados durante a execução do algoritmo.
3. **Cálculo do custo de uma configuração:** O custo de uma configuração deve ser calculado incrementalmente de modo que o tempo para avaliar uma nova configuração seja mínimo.
4. **Resfriamento:** O esquema de resfriamento da temperatura T é crucial para o bom desempenho do algoritmo: temperaturas iniciais muito baixas ou um resfriamento muito rápido podem levar a soluções não satisfatórias. Temperaturas iniciais muito altas ou um resfriamento muito lento levam a um custo de processamento muito alto.
5. **Estruturas de dados:** A habilidade de propor e avaliar novas configurações eficientemente depende de uma representação eficiente dos dados do problema. Para o problema de placement, isto significa estruturas projetadas de modo que a localização dos módulos e a conectividade do circuito sejam facilmente acessadas.

O algoritmo SA pode ser descrito, matematicamente, por meio de cadeias de Markov. Pode-se mostrar que, se forem executados um número infinito de passos em cada temperatura, e o resfriamento for muito lento ($\alpha \approx 1$), o algoritmo converge para a solução ótima com probabilidade 1. Ainda que esta certeza forneça pouca informação em como escolher os parâmetros adequados para cada aplicação específica, ela serve para dar confiança no bom comportamento do algoritmo.

3.2 - O algoritmo básico

```
SimulatedAnnealing()        {
        Começa com um estado inicial s_i; k = 0;
        T_k = T_0;                  /* temperatura inicial */
        Repita
                Repita
                        gera uma nova configuração s_j;
                        calcula ΔC = C(s_i) - C(s_j);
                        Se aceita(ΔC, T_k)
                                s_i = s_j;
                Até equilíbrio();
                T_{k+1} = α_k * T_k;        /* redução de temperatura */
                k = k + 1;
        Até congelamento();
}
```

A aceitação de um novo estado s_j é determinado pela função *aceita()* cuja estrutura é mostrada a seguir.

```
aceita(ΔC, T) {
/* retorna 1 se a configuração é aceita */
/* random() retorna um número no intervalo [0,1] */
        se (ΔC ≤ 0)     retorna(1);
        y = exp(-ΔC /T);
        r = random();
        se (r < y)      retorna 1;
        retorna 0;
}
```

Observe que novos estados caracterizados por $\Delta C \leq 0$ sempre satisfazem o critério de aceitação. No entanto, para estados caracterizados por $\Delta C > 0$ o parâmetro T tem um papel fundamental. Se T é muito grande, então r provavelmente será menor do que y e o novo estado é quase sempre aceito. Se T é pequeno, próximo de 0, então somente novos estados que são caracterizados por variações muito pequenas de $\Delta C > 0$ têm alguma chance de serem aceitos.

Nos testes, os melhores resultados foram obtidos inicializando-se o parâmetro T com um valor bastante alto, onde praticamente todos os estados propostos são aceitos. Além disso, permite-se que o sistema atinja o "equilíbrio" em cada temperatura. Isto é, um número suficiente de iterações é realizado no loop interno. O critério de parada é satisfeito quando o valor da função custo permanece o mesmo após várias iterações do processo de annealing.

3.3 - Implementação do algoritmo SA

O algoritmo SA serial usado neste trabalho, base de nossas tentativas de paralelização, é inspirado no algoritmo TimberWolf [8]. TimberWolf tem as seguintes características principais:

O espaço de estados: As células são dispostas em uma grade uniforme. É permitido às células ocuparem a mesma posição na grade, fato este refletido por um aumento na função custo.

Geração de novos estados: O algoritmo começa com um placement aleatório. Novos estados são obtidos trocando-se duas células de lugar (interchange) ou movendo-se uma célula para outra localização (displacement). O uso de ambos os métodos é necessário para atingir os melhores resultados. O problema de duas células tenderem a ocupar o mesmo lugar no espaço é resolvido introduzindo-se uma função penalidade que aumenta a função custo quando houver sobreposição. A proporção entre o número de displacements e interchanges tem um efeito pronunciado na qualidade final do placement. Os testes mostraram que uma proporção de 4 para 1 fornece os melhores resultados. Isto é implementado gerando-se dois números aleatórios. O primeiro entre 1 e o número de células e o segundo entre 1 e o número de células multiplicado por 5. Se o segundo número gerado for menor do que o número de células, procede-se um interchange para produzir o novo estado. Se o segundo número não corresponde ao número de uma célula procede-se a um displacement da célula correspondente ao primeiro número.

O displacement de uma célula é controlado por uma janela delimitadora a qual limita o campo de deslocamento de uma célula. Isto se deve ao fato de que nos estágios finais do algoritmo o displacement de uma célula tem pouca chance de ser aceito a menos que ele seja muito pequeno. Limitando-se o campo de deslocamento nas últimas fases do algoritmo, as células passam a efetuar muitos deslocamentos pequenos buscando reduzir o número de sobreposições e o custo da fiação. A janela limitadora é definida como uma região retangular centrada na célula a ser movida. No início do algoritmo, quando T é muito grande, essa janela tem dimensões iguais a duas vezes a dimensão da grade. As dimensões da janela são reduzidas proporcionalmente ao logaritmo de T. A nova localização é escolhida randomicamente no interior da janela delimitadora. Os interchanges de células são também controlados pelas janelas limitadoras. Duas células são trocadas de posição somente se a janela puder ser posicionada de forma a conter as duas células.

A função custo: A função custo para o problema de placement é baseada no comprimento estimado da fiação. O comprimento de um fio ligando vários pinos - uma rede - é estimado pelo semi-perímetro do retângulo que envolve todos os pinos. Para uma rede de 2 células esta medida coincide com a distancia Manhattan. A função *overlapping()*, a qual penaliza sobreposições de células, também influencia o valor da função custo. Uma característica adicional do sistema é a de que é possível atribuir pesos diferentes à altura e à largura do retângulo usado para calcular o custo de uma rede. Com isso consegue-se privilegiar ligações horizontais ou verticais, conforme seja o caso. De modo a não ser necessário recalcular totalmente a função custo a cada novo movimento, armazena-se, para cada módulo, a lista das redes às quais ele pertence. Dessa forma somente essas redes precisam ser recalculadas. Para reduzir ainda mais o processamento do cálculo da função custo, observa-se que uma rede só precisa ser totalmente recalculada em alguns casos, quando o módulo a ser movido estabelece uma das fronteiras do retângulo.

Estratégia de annealing: É adotado um esquema simples $T_{k+1} = \alpha_k * T_k$, $\alpha_k < 1$, com uma temperatura inicial arbitrariamente alta, determinada empiricamente. Os melhores

resultados foram obtidos quando α_k é maior (aproximadamente 0,95) durante as fases do algoritmo em que a função custo decresce mais rapidamente. Além disso α_k tem valores menores (aproximadamente 0,80) nas fases inicial e final do algoritmo. Um número constante de movimentos é efetuado em cada temperatura para atingir o equilíbrio. Este número é especificado como um múltiplo do número de células a serem alocadas. O critério de parada é implementado guardando-se o valor da função custo ao final de cada estágio do processo de annealing. O critério de parada é satisfeito quando o valor da função custo não se alterar por 5 iterações consecutivas.

Estruturas de dados: Os dados do problema são armazenados na forma de listas encadeadas. Uma lista para cada módulo contendo as redes às quais o módulo pertence. Estas listas são usadas para se ter acesso apenas às redes que precisam ser recalculadas em função do deslocamento de um módulo. Existem ainda listas por rede contendo os módulos que pertencem a uma dada rede. Estas são usadas quando é necessário recalcular completamente o custo de uma rede.

4 - Técnicas de Paralelização

Em linhas gerais, pode-se identificar duas aproximações para acelerar o algoritmo SA empregando paralelismo. A primeira consiste em reduzir o tempo de processamento por movimento, dividindo-se a tarefa de propor e avaliar uma nova configuração entre vários processadores. Como os processadores precisam se sincronizar ao menos uma vez por movimento, esta é uma solução de granularidade fina uma vez que envolve um grande número de mensagens trocadas entre os processadores. A segunda alternativa é processar vários movimentos completos em paralelo. Devido ao alto custo de comunicação imposto pela máquina paralela usada (cluster de workstations rodando PVM e se comunicando através de TCP/IP sobre barramentos Ethernet) não é possível explorar paralelismo de granularidade fina. Assim, as técnicas aqui propostas se concentram em algoritmos em que os processadores propõem e avaliam novos estados independentemente.

Uma restrição que se impõe é a de que os movimentos aceitos em paralelo não sejam contraditórios (por exemplo, mover a mesma célula para duas posições diferentes). Além disso, decisões erradas de aceitação/rejeição são possíveis quando os movimentos são gerados em paralelo. Durante a avaliação de cada movimento os processadores, individualmente, não conhecem os movimentos de todas as células e assumem que somente as células envolvidas no movimento corrente vão mudar de posição. Algumas vezes a decisão local de aceitar ou rejeitar um movimento pode levar a uma decisão global incorreta. Essa decisão incorreta pode levar não apenas a um comportamento oscilatório, como também afetar a convergência do método de annealing.

Existem duas alternativas para o problema das decisões erradas. A primeira é permitir movimentos paralelos contraditórios tendo em mente que eles podem afetar a convergência. Neste caso deve-se considerar a quantidade de paralelismo permissível, isto é, a fração dos módulos que podem ser movidos concorrentemente sem afetar as propriedades de convergência do algoritmo. A segunda alternativa é proibir a aceitação em paralelo de movimentos contraditórios. Para conseguir isso nós podemos polarizar a seleção de movimentos - por exemplo, a grade poderia ser dividida em regiões e cada região ser alocada a um processador - ou nós podemos filtrar os efeitos da aceitação de

movimentos que colidem depois que eles tenham sido calculados. Essa última estratégia foi empregada nesse trabalho.

4.1 - O conjunto mínimo

A fim de garantir que não hajam movimentos contraditórios, todos os processadores tentam, em paralelo, um movimento. Todos os movimentos descartados são incluídos na contagem de movimentos (o algoritmo de annealing precisa de um certo número de movimentos por temperatura para atingir o equilíbrio) e apenas um dos movimentos aceitos é incluído e efetuado. É claro que, como apenas um movimento aceito é efetuado a cada passo, não há movimentos contraditórios. Esse conjunto de movimentos considerados (todos os recusados e um aceito), chamado de conjunto mínimo, não é, obviamente, ótimo. No caso extremo poderia-se tentar determinar, a cada passo, o maior subconjunto dos movimentos calculados que não provocasse colisão. No entanto, essa tarefa poderia ser tão complicada que acabaria por anular os benefícios advindos da exploração do paralelismo. Esta estratégia de paralelização foi primeiro apresentada em [6].

Um modelo simples ilustra o desempenho do conjunto mínimo. Seja N o número de processadores e M o número de movimentos aceitos a cada passo. Na temperatura T, a probabilidade de aceitação de um movimento individual é $a(T)$. Uma vez que cada processador aceita um movimento com probabilidade $a(T)$ ou o rejeita com probabilidade $(1 - a(T))$, M tem uma distribuição de probabilidade binomial. $Pr(M = m)$. A probabilidade de que m movimentos sejam aceitos é dada então por:

$$Pr(M=m) = \binom{N}{m} * a(T)^m * [1-a(T)]^{N-m}$$

O tamanho esperado para o conjunto mínimo S é dado por:

$$E[S] = \Pr(M=0) * N + \sum_{m=1}^{N} \Pr(M=m) * (N-m+1)$$

que simplifica para:

$$E[S] = N * (1-a(T)) + 1 - (1-a(T))^N$$

Da equação acima percebe-se que, em altas temperaturas, onde a maioria dos movimentos é aceita, não é muito efetivo alocar-se muitos processadores ao cálculo dos candidatos a movimento. Isso dá margem a um esquema adaptativo de alocação dos processadores. A estratégia do conjunto mínimo parece ser adequada para baixas temperaturas mas ainda nos falta uma estratégia adequada ao regime de temperaturas mais elevadas. Esta estratégia é apresentada na seção 4.2. A seguir apresentamos detalhes de implementação do algoritmo do conjunto mínimo.

Implementação - Foi introduzida uma relação de mestre-escravo entre os processadores. Um processador mestre cria os processos escravos e participa também da parte serial do processamento. Para atingir o equilíbrio térmico um grande número de movimentos deve ser executado em cada temperatura. Para aumentar a velocidade os estados são avaliados assíncronamente. Os processadores continuamente geram novos movimentos, calculam o custo da configuração resultante e decidem sobre a aceitação ou não desta configuração. Se um escravo detectar uma configuração aceitável ele informa

ao processador mestre. Se o mestre decidir que aquela configuração gera um conjunto mínimo ele inicia uma atualização global da base de dados a fim de que todos os processadores recebam a nova configuração. Observe que a nova configuração é obtida incrementalmente pelos demais processadores. O processador que gerou a configuração aceita envia para os demais apenas o movimento que gerou esta configuração. Os estados rejeitados em cada processador contribuem para a contagem de movimentos necessários a se atingir o equilíbrio numa dada temperatura. Para diminuir a comunicação entre os processadores, os escravos acumulam um certo número de estados rejeitados antes de enviar a contagem ao mestre. Assim, o mestre é responsável por atualizar a temperatura, manter a integridade da base de dados e monitorar o critério de parada do algoritmo. Idealmente, a cada instante, existe apenas uma configuração válida e essa é conhecida por todos os processadores. Observe que a maior parte das mensagens trocadas se deve à aceitação de novos estados e, quando este evento é raro, o custo de comunicação é reduzido. Por esse motivo, esse algoritmo parece ser mais adequado ao regime de temperaturas baixas. Esta estratégia geral foi melhorada com a utilização de alguns conceitos novos descritos a seguir.

Validação - Devido aos atrasos impostos pela rede, alguns dos processadores podem, momentaneamente, estar trabalhando sobre configurações desatualizadas. O mestre tem de ser capaz de distinguir um pedido de validação gerado a partir de uma modificação na configuração corrente de um outro proveniente de uma configuração desatualizada. Para resolver este problema adotou-se um mecanismo de senhas. O mestre atribui uma senha para a próxima configuração válida. O primeiro estado que chegar ao mestre com a senha válida é aceito, os demais são ignorados. Quando o mestre faz a atualização global da base de dados ele envia a senha válida para o próximo estado. Um processador que tenha, nesse meio tempo, pedido validação de configuração ao mestre, percebe que o seu estado não foi aceito pela chegada de uma mensagem de nova configuração.

Recuperação de um movimento - A fim de reduzir a perda de boas configurações em temperaturas mais altas foi introduzido um mecanismo de recuperação de um movimento. Um escravo que tenha aceito uma configuração descobre se esta configuração foi usada ou não para a atualização global. Se ela não foi usada, ele tenta o mesmo movimento novamente. Os testes mostraram que esta estratégia leva a um comportamento ligeiramente melhor da convergência do algoritmo.

4.2 - Cadeias Paralelas

A ineficiência do algoritmo anterior em temperaturas elevadas se deve às altas taxas de aceitação e ao excessivo número de sincronizações que se tornam necessárias. Para superar esta dificuldade foi abandonada a premissa de que a base de dados é única para todos os processadores. Cada processador realiza o algoritmo SA seqüencial em sua totalidade na sua cópia local do problema. Depois que todas as cadeias são computadas para uma dada temperatura é realizada uma sincronização global. A cadeia de mais baixo custo é escolhida como ponto de partida para a próxima temperatura. O ganho em velocidade é obtido devido ao fato de que as cadeias podem ser mais curtas do que no SA seqüencial. Em temperaturas mais baixas, onde o decaimento da função custo é muito pequeno de uma temperatura para outra, existiria um certo desperdício computacional em manter os processadores trabalhando em paralelo. Uma vez que em baixas temperaturas o comportamento do algoritmo do conjunto mínimo é muito melhor, parece natural combinar os dois métodos

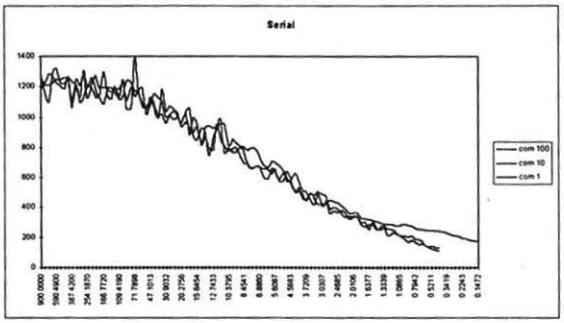
5 - Resultados

Resultados de algoritmos paralelos são normalmente apresentados como o ganho em velocidade de processamento em relação a uma implementação serial. Optamos aqui por mostrar alternativas de paralelização e os problemas encontrados em cada uma delas. Observe-se que para obter medidas de tempos significativas em um cluster de workstations seria necessário o uso das máquinas em modo exclusivo, isto é, sem o uso concorrente de outros usuários. Os resultados são mostrados para um placement de 100 células. Isto é grande o suficiente para avaliar o comportamento do algoritmo e, ao mesmo tempo, pequeno o bastante para permitir um grande número de execuções.

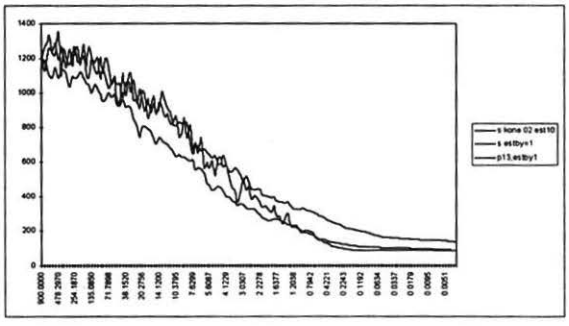
A fim de aplicar a estratégia adaptativa proposta torna-se necessário determinar quando passar de uma estratégia a outra. Nos exemplos o ponto de *crossover* é determinado em função do número de aceitações $a(T)$ ou, mais especificamente, quando $a(T) < 0.2$. Os resultados posteriores mostraram que este é um valor alto para o esquema de cadeia única ser efetivo, mas o seu uso foi interessante porque tornou aparente muitos problemas decorrentes do fato do algoritmo ser assíncrono.

- **Cadeias Paralelas**

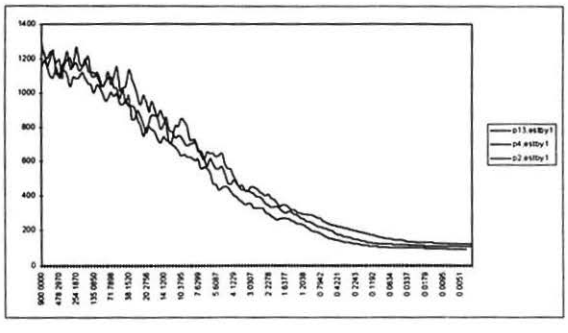
Um dos problemas do esquema de cadeias paralelas é o fato de que todos os processos se sincronizam ao término do cálculo de cada temperatura, ou seja, o tempo de processamento é amarrado pelo mais lento dos processadores da máquina virtual. Uma solução que cedo se vislumbrou foi escolher a cadeia de menor custo com os n primeiros processos que estabilizassem em cada temperatura. A questão que surge é qual o tamanho do conjunto n? Para responder a esta pergunta buscou-se descobrir qual era o número de estados necessários para atingir a estabilidade em uma dada temperatura. Para isto construiu-se uma versão serial do algoritmo e variamos o número de estados calculados por temperatura. Os resultados obtidos são mostrados na figura abaixo.



O gráfico mostra o decaimento da função custo com a temperatura para uma família de curvas. Cada uma das curvas representa um número de estados calculados por temperatura igual a k * *número de células* para k igual a 1, 10 e 100. A curva de pior desempenho corresponde a $k = 1$. Observa-se, no entanto, que não existe diferença significativa entre as curvas para $k = 10$ ou $k = 100$. Adotou-se então a curva $k = 10$ como padrão contra o qual o algoritmo paralelo deve ser comparado. Como vimos anteriormente, o ganho decorrente do esquema de cadeias paralelas é devido ao fato de que as cadeias podem ser mais curtas. A próxima questão a ser respondida é então qual o tamanho dessas cadeias paralelas.



A curva de pior desempenho corresponde ao algoritmo serial com $k = 1$. A curva intermediária corresponde ao algoritmo serial com $k = 10$, nossa referência de bom desempenho. A curva de melhor desempenho corresponde à versão paralela, com 13 processadores na máquina virtual e $k = 1$. Cabe ressaltar que, para temperaturas mais altas, a versão paralela tem sempre desempenho melhor do que o correspondente serial. A pergunta seguinte a ser respondida é quantos processadores eu devo manter na máquina virtual para que o esquema de cadeias curtas seja efetivo. O gráfico seguinte dá a resposta para esta questão.

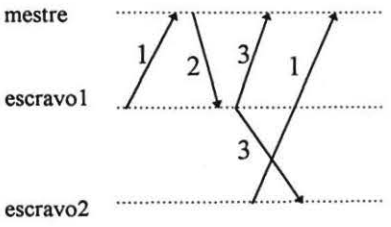


A curva de melhor desempenho corresponde ao esquema com 13 processadores na máquina virtual, a curva intermediária a 4 processadores e a curva de pior desempenho a apenas 2 processadores. Verificou-se que, mantidos ao menos 5 processadores na máquina virtual, não havia diferença significativa em relação à referência de 13 processadores. Conclui-se portanto que 5 ou 6 processadores parece ser um bom número. De posse deste dado pode-se montar um esquema de barreira em que os 6 primeiros processadores que chegarem à barreira determinam a configuração inicial para a próxima temperatura. Os demais interrompem o cálculo e recomeçam na nova temperatura. Até a presente data o mecanismo de barreira ainda não foi implementado, mas a simples utilização das cadeias curtas proporcionou uma significativa redução no tempo de processamento quando comparado à versão serial.

- **Conjunto mínimo**

A análise do esquema do conjunto mínimo será realizada através de diagramas no tempo das trocas de mensagens. Existem basicamente quatro tipos de mensagens circulando na máquina virtual: a) Pedido de validação de um estado (escravo → mestre) e a provável aceitação deste estado (mestre → escravo); b) Mensagem de atualização do contador de estados rejeitados (escravo → mestre); c) Envio de nova configuração (qualquer processador para todos os demais); d) Mensagem de atualização de temperatura (mestre → escravos). Essas situações são vistas nos diagramas a seguir:

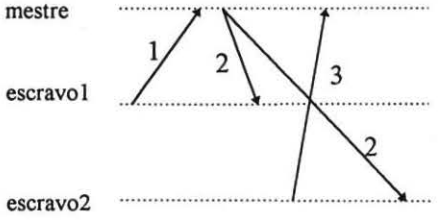
a) Escravo aceita um estado



1) escravo pede validação
2) acknowledge
3) broadcast da nova configuração

Observe que o escravo2 percebe que o seu pedido de validação não foi considerado através da chegada de uma mensagem de nova configuração.

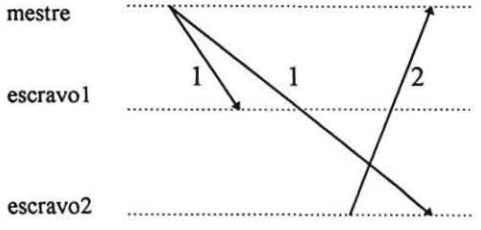
b) Mensagem de atualização do contador de estados rejeitados



1) atualização de estados rejeitados
2) nova temperatura
3) pedido de validação de estado

Observe que o escravo2 sabe que o seu pedido de validação não foi considerado pela chegada de uma mensagem de atualização de temperatura.

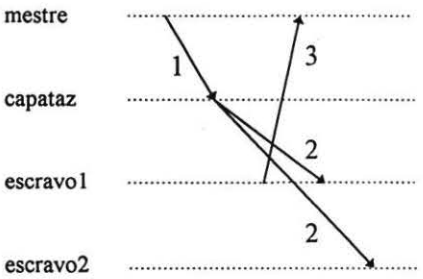
c) Mestre aceita nova configuração



1) broadcast de nova configuração
2) pedido de validação

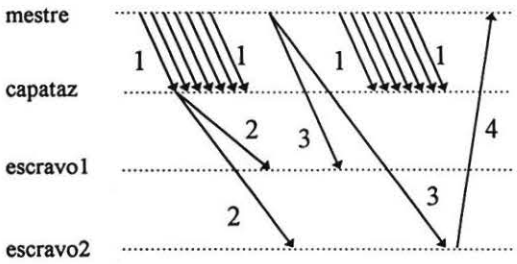
Observe que o escravo2 sabe que o seu pedido de validação não foi considerado pela chegada de uma mensagem de nova configuração.

Através de resultados experimentais verificou-se uma efetiva contribuição dos escravos no cálculo das cadeias. Cerca de 60% ou 70% dos estados intermediários são calculados pelos escravos. No entanto, o tempo de processamento paralelo mostrou-se muito maior do que o obtido na versão serial. Isto indica que, provavelmente, a temperatura ainda estava muito alta quando da troca de estratégias resultando em um número excessivo de estados aceitos e, conseqüentemente, em uma troca intensiva de mensagens entre os processadores. No entanto, buscando identificar as fontes de erro devido ao fato dos processos serem assíncronos, mantivemos o ponto de *crossover* e tentamos reduzir o tempo de processamento modificando o esquema de comunicação entre os processadores. Primeiramente, observamos que 63% do tempo total de processamento era gasto com a mensagem de broadcasting de nova configuração do mestre para os escravos. Este mau desempenho talvez não deva ser creditado ao PVM, mas sim à configuração usada (cluster de workstations ligadas por barramento Ethernet). De modo a não bloquear o mestre durante a operação de broadcasting pensamos então em criar um processo intermediário, que chamaremos aqui de capataz. O capataz recebe a mensagem de nova configuração do mestre (uma troca de mensagens rápida em PVM) e faz o broadcasting da nova configuração aos escravos. Esperava-se com isto que o nível de contribuição dos escravos caísse mas que, ao menos, eles não atrasassem o mestre na tarefa de computar uma cadeia. O novo esquema é visto na figura a seguir.



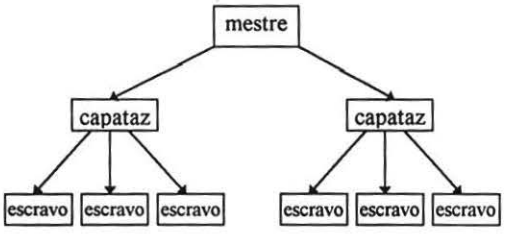
1) nova configuração
2) broadcast de nova configuração
3) pedido de validação de estado

Um problema que logo se evidenciou com essa solução foi o acúmulo de mensagens no capataz. A figura a seguir ilustra este fato.



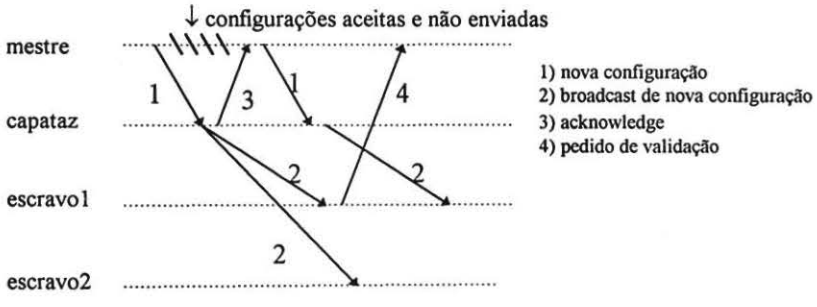
1) nova configuração
2) broadcast de nova configuração
3) mensagem de nova temperatura
4) pedido de validação

Observe que as mensagens de nova configuração se acumulam no capataz para serem enviadas aos escravos. Antes que todas sejam recebidas, o mestre completa o cálculo de uma temperatura e envia a nova configuração, temperatura e senha em broadcast para os escravos. Todas as mensagens acumuladas no capataz devem agora ser recebidas e descartadas pelos escravos, uma vez que elas se referem a uma configuração já ultrapassada. Observou-se então que os escravos ficavam tão ocupados recebendo e descartando mensagens que o nível de contribuição ao cálculo das cadeias caiu a níveis próximos de zero. Isto é, as cadeias eram quase que inteiramente calculadas pelo processo mestre. Uma possível solução para o problema, ainda não implementada, é mostrada na figura a seguir.



Aqui, o acúmulo de mensagens em cada um dos capatazes tende a diminuir, uma vez que é menor o número de escravos ligados a cada um deles. Além disso, os processos capatazes poderiam também participar do processamento ao invés de, simplesmente, rotear mensagens para os escravos.

Uma outra facilidade implementada é a de prover um nível de acknowledge entre o mestre e o capataz. O objetivo é o de não sobrecarregar o capataz e, em conseqüência, os escravos com mensagens que serão descartadas. Um dos problemas desta solução é o de que, uma vez que nem todos os movimentos aceitos chegam aos escravos, a base de dados não mais pode ser obtida incrementalmente. É necessário agora transmitir toda a configuração ao invés de apenas o movimento aceito.



Efetivamente, o nível de contribuição dos escravos para o cálculo da cadeia aumentou. No entanto a convergência do algoritmo piorou. Pode-se compreender o motivo desta degradação pela figura acima. Observe que quando o escravo1 recebe a nova configuração, vinda do capataz, o mestre já aceitou muitas outras configurações. O escravo1 aceita um movimento baseado na configuração desatualizada e pede validação

ao mestre. Este movimento é obviamente rejeitado, o que o escravo percebe pela chegada de outra mensagem de nova configuração. Percebe-se daí que os escravos conseguem contribuir apenas para a contagem de movimentos rejeitados. Além disso os movimentos rejeitados pelos escravos são, em sua maioria, obtidos sobre configurações já desatualizadas.

6 - Conclusões

O artigo apresenta estratégias para implementar o algoritmo de simulated annealing (SA) em um cluster de workstations. Ainda que um problema específico - o problema de placement - tenha sido a base de nossa implementação paralela, acreditamos que as técnicas desenvolvidas sejam aplicáveis a uma grande variedade de problemas para os quais o método SA é uma técnica efetiva de busca de soluções.

Foram apresentadas duas estratégias de paralelização. Uma - cadeias paralelas - mais apropriada ao regime de altas temperaturas e a outra - conjunto mínimo - com melhores resultados próximo ao congelamento.

O esquema de cadeias paralelas mostrou-se efetivo, com uma significativa redução no tempo de processamento quando comparado à versão serial. Isto se deve à utilização de cadeias curtas sem prejuízo da convergência do algoritmo.

O esquema do conjunto mínimo apresentou alguns problemas devido, principalmente, à comunicação assíncrona entre os processos. Algumas soluções estão sendo estudadas, sendo a mais óbvia a troca de estratégias em temperaturas mais baixas. Outras alternativas são uma mudança na hierarquia dos processos ou uma transição mais suave de uma estratégia para outra. Nesse caso, nós não passaríamos de um esquema de cadeia paralelas para o do conjunto mínimo. Ao contrário, o número de processadores envolvidos em uma tarefa ou outra seria progressivamente alterado.

7 - Bibliografia

[1] Casotto A., Romeo F., Vincentelli A.S., *"A Parallel Simulated Annealing Algorithm for the Placement of Macro-Cells"*, IEEE Transactions on Computer Aided Design, Vol. CAD-6, No. 5, Sep 1987.

[2] Diekmann R., Lüling R., Simon J., *"Problem Independent Distributed Simulated Annealing and its Applications"*, Proc. of the 4th IEEE Symposium on Parallel and Distributed Processing (SPDP '92).

[3] Geist A. et al., *"PVM3 Users's Guide and Reference Manual"*, Oak Ridge National Laboratory, 1994.

[4] Huang M.D., Romeo F., Vincentelli A.S., *"An Efficient General Cooling Schedule for Simulated Annealing"*, IEEE International Conference on Computer Aided Design, 1986.

[5] Kirkpatrick S., Gelatt C.D., Vecchi M.P., *"Optimization by Simulated Annealing"*, SCIENCE, Volume 220, Number 4598, May 1983.

[6] Kravitz S.A., Rutenbar R.A., *"Placement by Simulated Annealing on a Multiprocessor"*, IEEE Transactions on Computer-Aided Design, Vol. CAD-6, No. 4, July 1987.

[7] Laarhoven P.J.M., Aarts E., *"Simulated Annealing. Theory and Applications"*, D.Reidel Publishing Company, 1987.

[8] Sechen C., Vincentelli A.S., *"The Timberwolf Placement and Routing Package"*, IEEE Journal of Solid-State Circuits, Vol SC-20, No. 2, April 1985.