PARALELIZAÇÃO DE ALGORITMOS GENÉTICOS APLICADOS AO PROBLEMA DE PLACEMENT EM CLUSTERS DE ESTAÇÕES DE TRABALHO

Jonas Knopman*
Júlio S. Aude**

NCE/UFRJ
Caixa Postal 2324
Rio de Janeiro - RJ - 20001-970
e-mail: jonas@nce.ufrj.br

* NCE/UFRJ e Aluno de Doutorado da COPPE/UFRJ

** NCE/UFRJ e Instituto de Matemática da UFRJ

Resumo

Este artigo analisa alternativas eficientes de paralelização de algoritmos genéticos aplicados ao problema de placement em circuitos VLSI ou em placas de circuito impresso, considerando o uso de PVM em uma plataforma formada por uma rede de estações de trabalho. Com o objetivo de obter-se inicialmente um algoritmo seqüencial capaz de produzir resultados próximos do ótimo, foi verificado, através de uma análise experimental extensa e detalhada, ser necessário ajustar corretamente os parâmetros de controle do algoritmo. Com estes parâmetros ajustados, o algoritmo mostrou-se bastante adequado à exploração de paralelismo. O artigo apresenta, então, uma proposta de implementação paralela do algoritmo que alcança uma redução significativa da necessidade de comunicação entre os processadores. Esta implementação produziu resultados de mesma qualidade que a versão seqüencial, porém com um speed-up aproximadamente igual a 6 em uma rede composta de 8 estações de trabalho.

Abstract

This paper analyses efficient alternatives for the parallelization of genetic algorithms when they are applied to the placement problem in VLSI circuits or in printed circuit boards, considering the use of PVM on a cluster of workstations. Through a detailed experimental analysis, it was observed that the correct tuning of the control parameters is essential for the sequential algorithm to produce nearly optimal results. Once the correct tuning of the control parameters was obtained, the algorithm proved to be very suitable for parallelization. The paper presents a proposal for the parallel implementation of the algorithm which achieves a considerable reduction of the need for communication among processors. This implementation has produced results with similar quality to the sequential version and exhibited a speed-up around 6 in a cluster consisting of 8 workstations.

1. Introdução

O ambiente típico de projeto de circuitos integrados ou de sistemas eletrônicos em geral é normalmente constituído de um sistema de CAD disponível em estações de trabalho interconectadas em rede, permitindo, de preferência, o acesso a uma base de dados comum à equipe de projetistas. Apesar do elevado custo computacional envolvido na busca de soluções adequadas para alguns dos problemas enfrentados nas diferentes etapas de um projeto complexo, raramente o poder computacional do ambiente paralelo representado pela rede de estações de trabalho é utilizado. Este artigo propõe técnicas de paralelização, neste tipo de ambiente, de **algoritmos genéticos** [Srin94] aplicados ao problema de posicionamento (*placement*) de módulos em circuitos impressos ou em circuitos integrados.

Os algoritmos genéticos baseiam-se nos conceitos de genética e evolução das espécies. O interesse em métodos heurísticos baseados em fenômenos físicos ou naturais teve início na década de 70 quando Holland [Holl75] enunciou os princípios dos algoritmos genéticos que buscam reproduzir o efeito conseguido na natureza pelo processo de seleção natural, que leva à sobrevivência dos indivíduos mais aptos. Os algoritmos genéticos conseguem, com alta probabilidade, localizar o mínimo global em um cenário composto por vários mínimos locais, porém a um custo computacional elevado.

Este artigo apresenta um extrato dos principais resultados e conclusões obtidos no estudo desenvolvido com o objetivo de propor técnicas eficazes de paralelização dos algoritmos genéticos para o ambiente PVM [Geis94] em uma plataforma composta de estações de trabalho interligadas por uma rede Ethernet. A Seção 2 descreve de forma sucinta o problema de *placement*. A Seção 3 apresenta os princípios básicos de funcionamento dos algoritmos genéticos. Na Seção 4 são descritos os principais resultados alcançados com a tentativa de sintonizar os parâmetros de controle do algoritmo genético seqüencial de forma a produzir-se resultados de boa qualidade para o problema de *placement*. Na Seção 5 são analisadas alternativas de implementação de um algoritmo paralelo eficiente e é proposta uma técnica de paralelização que obteve speed-up quase linear com o número de processadores em uma plataforma constituída por uma rede de 8 estações de trabalho interligadas por uma rede Ethernet. Finalmente, a Seção 6 sumariza os principais resultados e conclusões derivados deste trabalho de pesquisa e comenta sobre as perspectivas de desenvolvimento futuro.

2. O Problema de *Placement*

Uma das etapas críticas do projeto físico de um circuito é o posicionamento do conjunto de módulos que compõem o circuito em um substrato qualquer, que pode ser, na prática, uma placa de circuito impresso ou uma área de silício. Os módulos são interligados entre si através de seus "terminais" de entrada e/ou saída. O conjunto de terminais pertencente a uma mesma conexão elétrica forma uma *rede*. Em geral, um mesmo módulo possui terminais em mais de uma rede. Formalmente, o problema de *placement* consiste em encontrar a localização ótima dos módulos de modo a minimizar uma função custo que, idealmente, deve refletir diversos requisitos a serem minimizados: comprimento médio das interconexões, ruído, dissipação de calor, área, atraso na propagação de sinais críticos, congestionamento de interconexões, etc.

Por serem tais objetivos conflitantes, é impossível incorporá-los todos em uma única função custo. Na prática, portanto, costuma-se adotar o comprimento total das

interconexões como a função custo a ser minimizada, uma vez que, dessa forma, uma aproximação razoável de alguns outros objetivos do projeto físico é, em geral, alcançada. De fato, circuitos com ligações curtas entre terminais têm, em geral, baixa dissipação de calor, pequena área, maior velocidade e baixo nível de ruído, embora, possam apresentar regiões de congestionamento que dificultem o roteamento automático das interconexões.

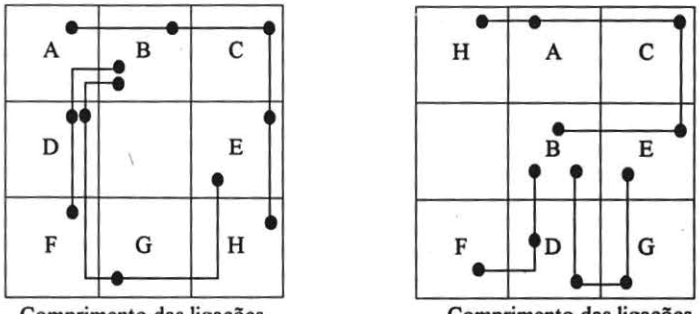
A distância normalmente usada para medir o custo de um *placement* é a distância retilínea ou distância *Manhattan*, isto é, se (x_1, y_1) e (x_2, y_2) são dois pontos no plano a distância retilínea d é dada por:

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

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 1 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

Figura 1: Diferença na função custo resultante de duas alternativas de *placement.*

Encontrar uma boa solução para o problema de *placement* é uma etapa crucial no projeto físico de circuitos VLSI, pois tem um impacto definitivo na velocidade final e no custo do *chip*. Este problema tem sido estudado há vários anos e as melhores soluções têm sido obtidas por algoritmos iterativos tais como *Simulated Annealing* [Sech88] e algoritmos evolucionários [Klin87], os quais incorporam algum mecanismo para escapar de mínimos locais. Ainda que estes algoritmos produzam boas soluções eles são, tipicamente, caracterizados por tempos de execução excessivamente longos o que tem motivado a pesquisa pelo uso de técnicas de paralelização para a solução do problema de *placement.*

[1]Em problemas reais o número dos pinos deveria aparecer ao lado da identificação dos módulos, i.e., A_1, B_3 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.

3. Fundamentos dos Algoritmos Genéticos

O conjunto de características de um indivíduo, que unicamente o distingue dos demais, é um fator importante na determinação de seu grau de aptidão natural para sobreviver em um meio qualquer. Estas características, por sua vez, são determinadas pelo material genético do indivíduo. Cada uma das características do indivíduo é controlada por uma unidade básica chamada de *gene*. Os conjuntos de genes formam os *cromossomas*. Ainda que a evolução se manifeste sob a forma de uma sucessão de modificações nas características dos indivíduos, são as modificações no material genético das espécies que constituem a essência do processo de evolução. Assim, a evolução das espécies se dá através da ação conjunta da seleção natural e da recombinação de material genético que ocorre durante a reprodução.

O processo de evolução inicia-se na recombinação de material genético proveniente dos pais durante a reprodução. Novas combinações de genes são geradas a partir das já existentes. A troca de material genético entre cromossomas é chamada de *crossover*. Pedaços dos cromossomas dos pais são trocados durante o *crossover* criando a possibilidade do surgimento de uma combinação mais adequada de genes e, conseqüentemente, de indivíduos mais aptos. A repetição do processo de seleção e *crossover* ao longo das gerações permite a evolução contínua do material genético da espécie e a geração de indivíduos que sobrevivem melhor em um dado ambiente.

Holland [Holl75] propôs os algoritmos genéticos como programas de computador que reproduzem o processo de evolução encontrado na natureza. Os algoritmos genéticos manipulam uma população de potenciais soluções para um problema de busca ou otimização. Eles operam sobre a representação codificada das soluções, equivalente ao material genético dos indivíduos na natureza. O algoritmo genético de Holland codifica as soluções em cadeias de *bits*. Cada solução é associada a uma medida de qualidade ou *fitness* que reflete a capacitação de um indivíduo em relação aos demais indivíduos da população. Assim como na natureza, um mecanismo de *seleção* força a contínua evolução da qualidade das gerações. Quanto maior o *fitness* de um indivíduo, maiores suas chances de reproduzir-se e de sobreviver. A recombinação de material genético é simulada através de um mecanismo de *crossover* que troca pedaços entre as cadeias de bits dos pais. Outra operação, chamada de *mutação*, provoca alterações esporádicas e randômicas nas cadeias de bits. A operação de mutação tem também uma analogia direta com o fenômeno encontrado na natureza e tem o papel de reintroduzir na população material genético perdido.

A estrutura de um algoritmo genético simples pode ser vista na Figura 2. Durante a geração t o algoritmo mantém uma população de soluções potenciais (cromossomas) $P(t) = \{x_1^t, \ldots, x_n^t\}$. Cada solução x_i^t é avaliada para fornecer a medida da sua qualidade ou *fitness*. Uma nova população $P(t+1)$ é então formada selecionando-se, probabilisticamente, os indivíduos mais *fitted* de $P(t)$. Alguns dos membros desta nova população sofrem transformações por meio de operações de *crossover* e mutação. O operador de *crossover* combina características dos pais pela troca de segmentos dos seus cromossomas. Por exemplo, se os pais são representados por vetores de dimensão 5, $(a_1, b_1, c_1, d_1, e_1)$ e $(a_2, b_2, c_2, d_2, e_2)$ a troca de segmentos a partir do segundo gene resultaria nos novos indivíduos $(a_1, b_1, c_2, d_2, e_2)$ e $(a_2, b_2, c_1, d_1, e_1)$. O operador de mutação arbitrariamente altera um ou mais genes de um cromossoma selecionado com uma probabilidade igual à probabilidade de mutação.

Assim, o projeto de um algoritmo genético para uma dada aplicação deve prover os seguintes componentes:

- Uma representação na forma de cromossomas das soluções potenciais do problema.
- Uma forma de criar uma população inicial de soluções válidas.
- O projeto de uma função de avaliação que classifica os indivíduos em função do seu *fitness*.
- Operadores genéticos que alterem a composição dos filhos.
- A atribuição de valores aos diversos parâmetros usados pelos algoritmos genéticos (tamanho da população, probabilidade de aplicação dos operadores genéticos, critério de parada, etc.)

```
/* t        ⇒   contador de gerações                                   */
/* P(t)     ⇒   População na geração t                                 */

t = 0;
Inicializa P(t);
Avalia P(t);              /* Calcula a função fitness para cada indivíduo na população.  */
Enquanto não atingiu o critério de terminação                {
        t = t + 1;
        Seleciona P(t) a partir de P(t-1);
        Altera P(t);      /* aplica os operadores de crossover e mutação.               */
        Avalia P(t);
}
Imprime resultados;
```

Figura 2: Estrutura de um algoritmo genético simples

4. Algoritmo Genético Seqüencial

Esta seção apresenta os principais resultados obtidos como fruto de uma análise experimental extensa e detalhada realizada visando a implementação de um algoritmo genético seqüencial adequado para aplicação e avaliação de técnicas de paralelização do algoritmo. Por adequado entende-se um algoritmo robusto, isto é, que produz resultados em sucessivas experiências com um baixo desvio padrão em torno de uma média próxima ao valor ótimo da solução do problema obtido anteriormente com o uso de *Simulated Annealing*.

4.1. Representação do Problema

Ao contrário da abordagem tradicional de algoritmos genéticos, que emprega a representação em strings de bits, foi utilizada a codificação direta das variáveis do problema nos genes do cromossoma. Assim, cada gene constitui-se em um par ordenado de números inteiros (x, y) correspondente à posição no plano do módulo associado ao gene. O principal objetivo desta implementação é aproximar o algoritmo genético do espaço de solução do problema, permitindo-se que os operadores incorporem algumas características do problema.

Um indivíduo consiste de um vetor (chamado de cromossoma) de pares ordenados (chamados de genes). Cada um dos genes corresponde a um módulo a ser alocado. A inicialização consiste então em se atribuir pares ordenados válidos aos genes de um cromossoma, não sendo permitidas sobreposições (*overlappings*) de módulos.

Dado um cromossoma em particular, a função custo retorna um valor numérico proporcional à "utilidade" ou "habilidade" do indivíduo representado pelo cromossoma. O custo associado a um cromossoma é uma característica individual, ou seja, ele não é influenciado pelo custo dos demais indivíduos da população. A função custo utilizada está associada ao comprimento total da fiação. O comprimento de um fio ligando vários terminais em uma rede é estimado pelo semiperímetro do retângulo que envolve todos os terminais.

A população de soluções potenciais é armazenada na forma de um vetor de estruturas, contendo a função custo, o *fitness* do indivíduo e um ponteiro para a codificação da solução (cromossoma). As redes são armazenadas na forma de listas encadeadas onde os elementos das listas são os módulos que compõem a rede.

4.2. Operadores Genéticos e Parâmetros de Controle

A escolha apropriada da função *fitness* foi importante para o bom desempenho do algoritmo genético. Esta função fornece uma medida da qualidade de um indivíduo em relação ao restante da população. A medida de *fitness* é usada durante o processo de reprodução para fazer com que os indivíduos mais *fitted* participem de um número maior de cruzamentos e tenham também maior possibilidade de sobrevivência. Uma função bi-linear mostrou-se a mais adequada para gerar o valor de *fitness*, propiciando um maior controle sobre a pressão seletiva. Com esta função consegue-se controlar o grau de polarização, no processo de reprodução, em favor dos indivíduos com *fitness* maior que a média em relação aqueles com *fitness* abaixo da média.

O método de seleção utilizado deve privilegiar os indivíduos mais *fitted* no processo de reprodução. Foi utilizado o chamado método da roleta, que sorteia um indivíduo a partir da escolha de um ponto em um círculo dividido em setores com largura proporcional ao *fitness* dos indivíduos candidatos a participarem do processo de reprodução.

Após experimentações sem sucesso com os operadores de inversão e o *crossover* cíclico propostos por Mazumder et al. [Mazu93], foi utilizado o crossover de 2 pontos [Sysw89] [Spea91]. No entanto, para que se obtivesse bons resultados com esta abordagem foi essencial proceder-se a uma alocação inicial dos módulos no cromossoma. O objetivo desta alocação é reduzir o número de redes seccionadas pela partição do cromossoma na operação de *crossover*. A ordenação dos módulos no cromossoma nada tem a ver com a posição desses módulos no plano de alocação. Ela constitui-se, tão somente, em uma tentativa de manter juntas, após o *crossover*, redes já bem posicionadas presentes em um dos pais.

Dado o custo computacional elevado de avaliação da alocação ótima dos módulos em um cromossoma usando um algoritmo do tipo *simulated annealing*, foi empregada uma heurística de processamento rápido e que produziu resultados aceitáveis. Esta heurística é baseada em um algoritmo guloso, onde os módulos são alocados no cromossoma em posições contíguas, da esquerda para a direita de forma que a cada passo seja alocado o módulo que tem mais ligações com os módulos já alocados.

O operador de mutação empregado realiza operações de troca e de deslocamento de genes em um indivíduo da população. A probabilidade de mutação utilizada foi de 1%. Caso a posição no plano escolhida aleatoriamente esteja ocupada por algum outro gene daquele indivíduo, uma operação de troca é realizada, senão é feita uma simples operação de deslocamento.

4.3 O Algoritmo Seqüencial

O algoritmo seqüencial desenvolvido baseou-se na proposta formulada por Baluja [Balu92] de dividir a população total em sub-populações distintas processadas independentemente, havendo porém trocas eventuais de indivíduos entre as sub-populações. Esta idéia foi fundamental para obter-se resultados com boa qualidade no uso de algoritmos genéticos aplicados ao problema de *placement*.

A implementação da idéia de sub-populações foi realizada com o conceito de "deme", um grupo de 10 indivíduos formado a partir de uma sub-matriz 3x3 da matriz completa que representa o plano de alocação. A cada posição da matriz completa são sempre associados dois indivíduos. A posição central da matriz 3x3 correspondente ao deme a ser tratado a cada passo é escolhida aleatoriamente. Os 10 indivíduos do deme são formados pelos dois cromossomas no centro do deme e por 1 cromossoma escolhido aleatoriamente de cada um dos 8 vizinhos do centro.

Abaixo é mostrado o pseudocódigo do algoritmo empregado, onde, a cada geração, 2 filhos são produzidos por crossover a partir de 2 pais selecionados dentre os 5 indivíduos mais *fitted* do "deme". Pela estratégia de *tournament* adotada, estes dois filhos sempre substituem os dois indivíduos no centro do deme após sofrerem eventualmente uma operação de mutação.

```
/* Melhor_k   ⇒  Indivíduo mais fitted na geração atual                    */
/* Melhor_k-1 ⇒  Indivíduo mais fitted na geração anterior                 */
/* Pior_k     ⇒  Indivíduo menos fitted na geração atual                   */

Genese ( );                          /* Produz uma população inicial de POPSIZE indivíduos. */
                                     /* As células nos indivíduos não se sobrepõem          */
Repete por T gerações          {
    EscolheDeme( );                  /* Aleatoriamente escolhe um deme na população         */
    Pior_k = Melhor_k-1;             /* Esquema elitista. Substitui o pior indivíduo desta  */
                                     /* geração pelo melhor da geração anterior             */
    Evaluation( );                   /* Avalia o fitness dos indivíduos no deme.            */
                                     /* Transformação bilinear                              */
    Pai1 = Seleção( );               /* Escolhe o primeiro pai. Método da roleta.           */
    Pai2 = Seleção( );               /* Escolhe o segundo pai                               */
    Gera 2 filhos segundo o esquema: {
        Crossover(Pai1, Pai2);       /* Gera um filho.                                      */
        EliminaOverlapping( )        /* Elimina sobreposições.                              */
        Mutação( )                   /* Probabilisticamente aplica mutação ao               */
    }                                /* indivíduo recém gerado.                             */
    Tournament( );                   /* Os 2 indivíduos recém gerados substituem os         */
}                                    /* indivíduos no centro do deme.                       */
Imprime resultados.
```

A fim de possibilitar uma maior compreensão do efeito do tamanho da população sobre a qualidade do *placement*, o algoritmo foi testado, executando-o no mínimo 15 vezes, para diversos valores deste parâmetro. Em cada um dos testes permitiu-se que o algoritmo executasse por um número arbitrariamente grande de iterações de modo que ele atingisse a convergência. Os resultados são apresentados na Figura 3 para um problema composto por 80 módulos e 30 redes equipotenciais (CIRCUITO_1). Os pontos na curva referem-se à função custo (C_x) e aos valores de ($C_x + \sigma$) e ($C_x - \sigma$), onde σ é o desvio padrão da distribuição. A reta horizontal no gráfico corresponde ao resultado obtido pelo algoritmo *Simulated Annealing*.

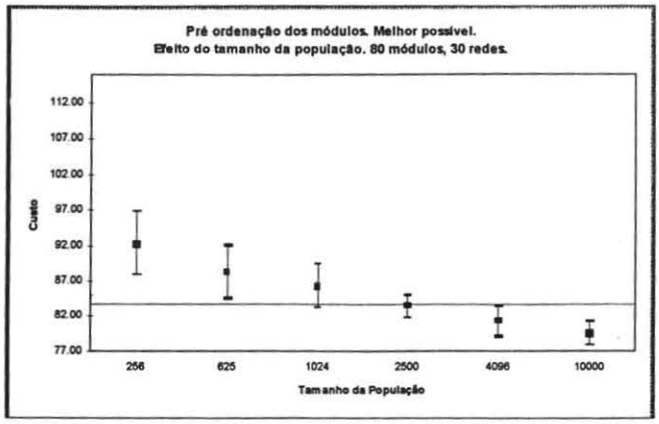


Figura 3 - Efeito da ordenação dos módulos no cromossoma. Melhor ordenação possível. CIRCUITO_1.

A Figura 4 mostra os resultados obtidos para um circuito composto por 100 módulos e 300 redes equipotenciais (CIRCUITO_2). A curva tracejada corresponde à regressão logarítmica dos pontos medidos.

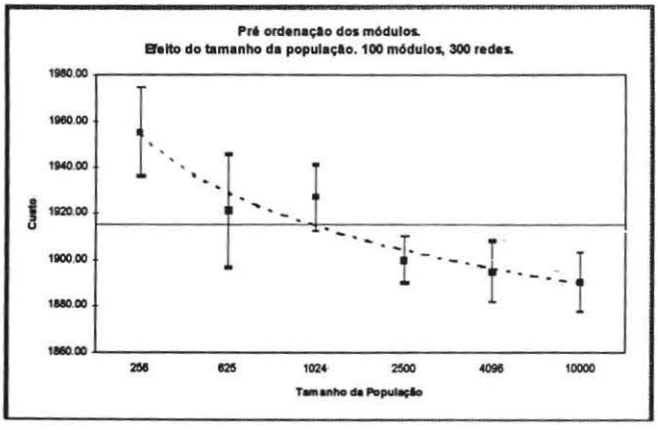


Figura 4 - Pré alocação dos módulos. Efeito do tamanho da população. CIRCUITO_2.

A partir das Figuras, percebe-se claramente o efeito pronunciado do tamanho da população sobre o custo final obtido. Como era de se esperar, devido ao tamanho reduzido das sub-populações (10 indivíduos), os melhores resultados foram obtidos com um número de indivíduos muito superior àqueles usados por algoritmos genéticos tradicionais. No caso dos circuitos de teste este valor situou-se entre 1024 e 2500 indivíduos mas, é de se supor, que estes valores variem com o tamanho e a complexidade do circuito, tendo de ser ajustados caso a caso.

Deve-se ressaltar que, uma vez que as subpopulações são sempre formadas por 10 indivíduos, o tempo de processamento de uma única geração não é afetado pelo tamanho da população. Este parâmetro tem impacto apenas sobre os requisitos de memória e sobre o número de iterações necessárias a atingir-se a convergência. Estes fatos, aliados à independência do processamento das sub-populações, apontam para a conveniência de uma solução paralela discutida na Seção 5.

5. Paralelização do Algoritmo Genético

A adoção do esquema de sub-populações precisa, para ser efetivo, de populações de tamanho consideravelmente maior do que os algoritmos genéticos tradicionais. Isto se deve ao fato de que populações maiores levam a soluções melhores devido à maior diversidade existente na população. Populações grandes devem ser avaliadas por muitas gerações (deve-se recordar que geração, no nosso contexto, refere-se à criação de dois novos indivíduos em um *deme*) o que torna o algoritmo genético lento quando comparado à alternativa do *Simulated Annealing*. Uma alternativa imediata para este problema é a paralelização do algoritmo buscando encontrar soluções de melhor qualidade ou em um tempo menor de processamento.

Os algoritmos genéticos são procedimentos altamente paralelos o que, em princípio, facilita sua implementação em qualquer tipo de arquitetura paralela. Uma possível estratégia de paralelização a ser adotada teria como base um modelo mestre-escravo, onde os dados são centralizados no mestre e os procedimentos são distribuídos pelos processadores escravos. Neste modelo, o mestre executa a fase de seleção e manda pares de indivíduos para os processadores escravos. Os escravos recebem os pares, transformam-nos usando os operadores genéticos, computam a função custo dos indivíduos gerados e, finalmente, retornam os novos indivíduos para o processador mestre. Esta estratégia básica não é efetiva em máquinas de memória distribuída pois: o mestre representa um gargalo no sistema; uma fração da computação é seqüencial (a fase de seleção); e a granularidade do processo é muito fina gerando comunicação excessiva entre os processadores. O custo de comunicação é um dado ainda mais crítico no ambiente de programação usado neste trabalho (um *cluster* de estações de trabalho comunicando-se por troca de mensagens usando TCP/IP sobre uma rede Ethernet) e, na prática, limita as alternativas de paralelização que podem ser empregadas. Devido a este fato, a estratégia de paralelização mais efetiva tende a ser baseada no uso de populações distribuídas, onde os processadores manipulam grupos menores de indivíduos e computam por um certo número de gerações entre trocas de mensagens.

Nesta seção são apresentadas duas alternativas de paralelização usando esta filosofia: um modelo centralizado onde, de tempos em tempos, toda a população é centralizada em um processador mestre e um modelo distribuído, possivelmente assíncrono, onde os indivíduos, periodicamente, migram de um processador para outro.

5.1 O Modelo Centralizado

Nesta implementação os processadores são organizados na forma mestre-escravos. A população é distribuída entre os processadores que computam, no seu pedaço da população, todos os passos do algoritmo genético seqüencial consistindo de seleção, reprodução e *tournament*. Periodicamente a população é reunida no processador mestre. O mestre embaralha a população e a redistribui entre os escravos.

O número de operações de comunicação pode ser feito tão pequeno quanto se queira, aumentando-se o número de gerações entre sincronizações. Uma vez que o tamanho das sub-populações em cada um dos processadores é, em geral, muito pequeno, o número de sincronizações representa um compromisso entre velocidade de processamento e a qualidade da solução gerada.

O pseudo código do procedimento executado pelo processador mestre é mostrado a seguir:

```
/* nhost    ⇒    número de processadores na máquina virtual                    */

Dispara os processos escravos;

Genese ( );          /* Gera população inicial de POPSIZE/nhost indivíduos.    */

Repete por T gerações        {
        Repete até sincronização  {
              Reprodução( );
        }
/*      SINCRONIZAÇÃO:                                                         */
        Recebe as sub populações dos escravos;
        Embaralha a população resultante;
        Redistribui a população embaralhada para os escravos;
}
Imprime resultados.
```

O procedimento *Reprodução( )* é o mesmo apresentado na Seção 4 para o algoritmo seqüencial.

Cada uma das posições da matriz completa de indivíduos tem a chance de ser selecionada K vezes entre sincronizações. Assim, o número de gerações entre sincronizações (n_g) é dado por: $n_g = K * POPSIZE / nhost$. Nos nossos experimentos variamos K de 1 até 10.

O embaralhamento da população se dá no processador mestre. Um vetor *pool*[ ] é ordenado segundo um campo *idx* atribuído randomicamente no intervalo [0, 10**POPSIZE*]. A operação de ordenação é otimizada pelo fato de que o vetor *pool*[ ] não contém os cromossomas propriamente ditos, mas apenas ponteiros para estes. Assim, as pesadas estruturas de dados representando os cromossomas não precisam ser movidas durante a ordenação. O código da operação de embaralhamento é visto a seguir.

```
/*
Funções:
        rnd(n)    ⇒    Devolve um número inteiro no intervalo [0, n)
        qsort( )  ⇒    Quick Sort
*/

for(Paux=pool, i=0; i<POPSIZE; i++)        {
        Paux->idx=rnd(10*POPSIZE);
        Paux++;
}
qsort((void *)pool, POPSIZE, sizeof(struct Nicho), (pint)sort_pool);
```

O algoritmo não apresentou bons resultados para qualquer valor do período de sincronização K. Para valores pequenos de K a execução do algoritmo é muito lenta, já que um número excessivo de bytes deve ser transmitido pela rede a cada sincronização e estas são muito freqüentes. Para valores maiores de K observa-se uma redução no tempo de processamento ao custo porém de uma perda de qualidade na solução obtida. Pode-se entender este fato imaginando-se a situação em que os processadores não se comunicam durante o procedimento de cálculo. Cada um deles estaria trabalhando então sobre um pedaço por demais reduzido da população, o que acarretaria na convergência para um mínimo local. A solução parece então apontar para o modelo distribuído, que será visto na próxima seção.

5.2 O Modelo Distribuído

A centralização da população implica em uma grande quantidade de dados circulando pela rede o que ocasiona um gargalo no processamento. Na implementação distribuída elimina-se a figura do mestre. Cada processador tem um pedaço da população total e executa o algoritmo genético sobre o seu conjunto de indivíduos, do início ao fim do processamento. A comunicação acontece periodicamente quando os processadores enviam alguns dos seus indivíduos para os vizinhos. Os indivíduos recebidos são imediatamente incluídos pelo processador hospedeiro na população local. A esperança aqui é a de que o conjunto das sub-populações possa beneficiar-se pela migração de indivíduos potencialmente *fitted*.

Cada um dos processadores executa um algoritmo genético baseado no modelo descrito na Seção 4. Teoricamente, em uma máquina maciçamente paralela, seria possível atribuir um processador a cada posição da matriz, o centro de um "deme", e substituir toda a população em um único passo, efetuado de forma síncrona por todos os processadores. Esta observação deu margem à estratégia de paralelização proposta neste artigo, que se baseia na distribuição da população entre os processadores segundo o esquema ilustrado na Figura 5, onde as regiões de fronteira são duplicadas. É importante observar que, na Figura 5, o processador ao centro nunca processa "demes" centrados nas posições hachuradas, que, portanto, nunca são escritas por este processador. A reduzida necessidade de comunicação representa uma característica muito interessante deste algoritmo: apenas os indivíduos na fronteira das áreas atribuídas aos processadores precisam ser transferidos. Além disso, uma vez que o instante de recebimento de uma fronteira não é crítico no desempenho do algoritmo, a comunicação entre os processos pode ser assíncrona. Deve-se ressaltar, no entanto, que a migração de indivíduos produz melhores resultados quando os processadores encontram-se em estágios semelhantes do

processo de otimização a fim de que um indivíduo exportado não domine a população hospedeira nem por esta seja dominado. Portanto, a estratégia assíncrona deve ser empregada, preferencialmente, em arquiteturas compostas por máquinas homogêneas e dedicadas. Quando este não for o caso pode-se forçar a sincronização através do mecanismo de barreiras.

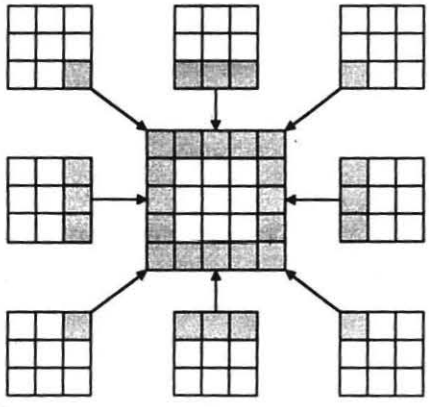


Figura 5: Estratégia de migração de indivíduos no algoritmo distribuído.

De forma análoga à estratégia centralizada, o número de operações de comunicação pode ser feito tão pequeno quanto se queira, representando um compromisso entre a qualidade da solução e o tempo de processamento. Em nosso testes cada uma das posições da matriz tem a chance de ser selecionada uma vez entre sincronizações. Assim, o número de gerações entre sincronizações (n_g) é dado por:

$$n_g = POPSIZE \,/\, nhost$$

O pseudocódigo do procedimento executado por todos os processadores é o seguinte:

```
/* nhost  ⇒   número de processadores na máquina virtual                    */

Genese ( );      /* Gera população inicial de POPSIZE/nhost indivíduos.     */

Repete por T gerações      {
        Repete até envio de dados      {
              Recepção( );
              Reprodução( );
        }
/*      ENVIO DE DADOS:                                                     */
        EnviaFronteira(NW);        Envia Fronteira(N);
        EnviaFronteira(NE);        EnviaFronteira(E);
        EnviaFronteira(SE);        EnviaFronteira(S);
        EnviaFronteira(SW);        EnviaFronteira(W);
}
```

O procedimento *Reprodução()* é idêntico àquele apresentado na Seção 4. O procedimento *Recepção()* verifica a existência de dados a serem recebidos e sua procedência a fim de determinar a posição onde eles devem ser inseridos.

Nos testes experimentais foi utilizado um *cluster* de estações IBM Power PC 25T dedicadas, onde nenhum outro processo encontrava-se em execução nas máquinas. Oito processadores estavam disponíveis para os testes. O algoritmo foi testado para um problema pequeno (CIRCUITO_1) e um problema médio (CIRCUITO_2). Os resultados podem ser vistos nas tabelas a seguir. Os resultados do *annealing* serial foram incluídos para fins de comparação. N é o número de processadores na máquina paralela virtual.

CIRCUITO_1. POPSIZE = 1024 indivíduos.				
	Genético Serial	Paralelo. $N = 4$	Paralelo. $N = 8$	*Annealing* Serial
Custo Médio	86,27	83,20	82,10	83,67
Desvio Padrão	3,14	2,91	1,98	3,34
tempo(s)	170,49	68,13	59,57	--
Speedup	--	2,50	2,86	--

Tabela 1 - Resultados para o CIRCUITO_1. População = 1024 indivíduos.

CIRCUITO_1. POPSIZE = 2500 indivíduos.				
	Genético Serial	Paralelo. $N = 4$	Paralelo. $N = 8$	*Annealing* Serial
Custo Médio	83,36	83,70	83,50	83,67
Desvio Padrão	1,67	2,45	2,15	3,34
tempo(s)	416,14	144,67	104,04	--
Speedup	--	2,88	4,00	--

Tabela 2 - Resultados para o CIRCUITO_1. População = 2500 indivíduos.

CIRCUITO_2. POPSIZE = 1024 indivíduos.				
	Genético Serial	Paralelo. $N = 4$	Paralelo. $N = 8$	*Annealing* Serial
Custo Médio	1926,80	1931,27	1916,82	1915,00
Desvio Padrão	14,68	19,94	13,68	28,00
tempo(s)	960,43	277,92	163,77	--
Speedup	--	3,46	5,86	--

Tabela 3 - Resultados para o CIRCUITO_2. População = 1024 indivíduos.

CIRCUITO_2. POPSIZE = 2500 indivíduos.				
	Genético Serial	Paralelo. $N = 4$	Paralelo. $N = 8$	*Annealing* Serial
Custo Médio	1900,00	1903,10	1921,90	1915,00
Desvio Padrão	9,75	15,40	11,70	28,00
tempo(s)	2347,28	662,44	388,05	--
Speedup	--	3,54	6,05	--

Tabela 4 - Resultados para o CIRCUITO_2. População = 2500 indivíduos.

Observe-se que a solução paralela mantém a qualidade da versão serial proporcionando ainda um efetivo *speedup* no tempo de processamento. Tal *speed-up* foi resultado de dois fatores importantes: a busca em paralelo da solução ótima por vários processadores e a utilização de um tamanho de população em cada processador significativamente menor do que o tamanho de população mínimo que poderia ser utilizado para obter-se uma solução de boa qualidade com o algoritmo genético seqüencial.

6. Conclusões

Este trabalho compreendeu um estudo experimental detalhado dos algoritmos Genéticos tendo em vista a aplicação ao problema de *placement*. Tendo em vista a capacidade do algoritmo de encontrar a solução ótima global do problema a um custo computacional elevado, foram analisadas também diferentes técnicas de paralelização do algoritmo, considerando-se uma plataforma formada por um *cluster* de estações de trabalho e o uso do ambiente de programação paralela PVM. As principais conclusões deste trabalho foram as seguintes:

- Após os devidos ajustes nos parâmetros de controle, o algoritmo genético produziu resultados bastante satisfatórios para o problema de *placement*;
- Para um bom desempenho do algoritmo genético seqüencial, foi fundamental trabalhar-se com um tamanho de população significativamente grande, o processamento de sub-populações pequenas de forma independente e adotar-se uma estratégia de alocação inicial dos genes no cromossoma;
- O algoritmo genético foi facilmente paralelizável e produziu um speed-up aproximadamente linear com o número de estações de trabalho incorporadas ao cluster.
- O *speed-up* obtido com a implementação paralela proposta foi devido a três fatores fundamentais: o reduzido custo de comunicação; a busca em paralelo da solução ótima; e a possibilidade de associar-se a cada processador uma população de tamanho relativamente pequeno.

Como evolução futura deste trabalho está prevista a implementação e análise de desempenho dos algoritmos genéticos nos ambientes MPVM, PVM Padrão e MULPLIX nativo a serem disponibilizados em breve no multiprocessador MULTIPLUS [Aude95] de memória compartilhada distribuída em desenvolvimento no NCE.

O ambiente MPVM é uma implementação do ambiente PVM no multiprocessador MULTIPLUS que mapeia "tasks" concorrentes do PVM em "threads" do sistema operacional MULPLIX e utiliza o compartilhamento de memória entre threads para implementar as funções de troca de mensagens. No ambiente MULPLIX nativo, o usuário trabalha diretamente com o paradigma de memória compartilhada e possui maiores facilidades para explorar de forma conveniente a arquitetura do tipo NUMA do MULTIPLUS.

Espera-se que, deste exercício de paralelização em três ambientes distintos, resultem, em decorrência da busca de maior desempenho, não apenas novas propostas de implementação dos algoritmos, como também que alguns gargalos da plataforma MULTIPLUS/MULPLIX sejam detectados e eventualmente removidos.

7. Agradecimentos

Os autores agradecem à FINEP, CNPq e RHAE o apoio dado ao desenvolvimento deste trabalho.

8. Referências

[Aude95] Aude, J.S. et al., *Implementation of the Multiplus/Mulplix Parallel Programming Environment*, Anais do VII SBAC-PAD, Canela, RS, Agosto 1995.

[Balu92] Baluja, S., *A Massively Distributed Parallel Genetic Algorithm (mdpGA)*, Technical Report, School of Computer Science, Carnegie Mellon University, 1992.

[Geis94] Geist A. et al., *PVM3 Users's Guide and Reference Manual*, Oak Ridge National Laboratory, 1994.

[Holl75] Holland, J.H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[Klin87] Kling, R.M. and Banerjee, P., *ESP: A new standard cell placement package using simulated evolution*, in Proc. 24th ACM/IEEE Design Automation Conference, 1987.

[Mazu93] Mazumder P. and Mohan S., *Wolverines: Standard Cell Placement on a Network of Workstations*, IEEE Transactions on Computer-Aided Design, Vol 12, N. 9, September 1993.

[Sech88] Sechen, C., *VLSI Placement and Global Routing Using Simulated Annealing*, Kluwer, 1988.

[Spea91] Spears W.M. and De Jong, K.A., *On the Virtues of Parametrized Uniform Crossover*, Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, Los Altos, CA, 1991.

[Srin94] Srinivas, M. and Patnaik, L.M., *Genetic Algorithms: A Survey*, IEEE Computer, p. 17-26, June 1994.

[Sysw89] Syswerda, G., *Uniform Crossover in Genetic Algorithms*, in Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, Los Altos, CA, 1989.