Um Sistema Distribuído para Posicionamento de Células em Circuitos VLSI

Lucas Brusamarello[1], Renato Hentschke[1], Carlos Morelli[2], Ricardo A. L. Reis[1]

[1]Universidade Federal do Rio Grande do Sul, [2]Universidade Luterana do Brasil
lucas, renato, reis{@inf.ufrgs.br}, morelli@gravatai.ulbra.tche.br

Introdução

O fluxo de projeto de circuitos digitais exige ferramentas de CAD capazes de operar sobre sistemas integrados cada vez mais complexos. Uma das etapas de síntese de um circuito integrado é a síntese física, que é responsável pela geração do leiaute do circuito de forma que ele esteja pronto para fabricação. Como entrada, a síntese física recebe uma descrição do circuito em nível lógico. Nesta descrição, os transistores que compõem o circuito são organizados em células. A síntese física utiliza, em geral, uma biblioteca de células, onde cada tipo lógico de célula é mapeado para um desenho de leiaute correspondente. Porém, as células devem ser posicionadas no leiaute e depois roteadas. O posicionamento define coordenadas X e Y para cada célula. O roteamento conecta as células por fios de metal, respeitando restrições de desenho para não haver cruzamento de fios.

A etapa de posicionamento tem um papel fundamental na síntese de circuitos, pois ela está estritamente relacionada com o tamanho das conexões e com a roteabilidade dos circuitos. O tamanho das conexões é um dos gargalos de performance dos circuitos modernos, sendo importante que o algoritmo de posicionamento encontre o mínimo comprimento total (e médio) de conexões possível. No entanto, encontrar o posicionamento ótimo (para qualquer métrica de qualidade, por exemplo, tamanho total das conexões) é um problema NP-difícil.

Este trabalho trata da implementação de um algoritmo para posicionamento de células em circuitos VLSI [HEN 2002]. Os algoritmos encontrados na literatura podem ser divididos em dois grupos: heurísticos ou meta-heurísticos. Os heurísticos são desenvolvidos especificamente para a resolução do problema de posicionamento. Exemplos de algoritmos heurísticos: Force Directed Placement, Recursive Bisection, Quadractic Placement e outras técnicas. Porém, observa-se que o problema de posicionamento se adapta bem as meta-heurísticas, que a principio podem resolver qualquer problema de otimização. Exemplos de meta-heurísticas utilizadas com sucesso para posicionamento: Simulated Annealing, Algoritmos Genéticos, Busca Tabu.

O algoritmo utilizado neste trabalho é baseado na meta-heurística GRASP (Greedy Randomized Adaptive Search Procedure)[MOR 2000]. No conhecimento dos autores deste artigo, não há nenhuma implementação publicada de GRASP para posicionamento (somente para roteamento) de células para circuitos VLSI. Assim, este trabalho trata-se de uma primeira implementação de tal meta-heurística para esse problema. Resultados de outras aplicações mostram que o GRASP consegue vantagem sobre as demais meta-

heuristicas em diversos aspectos. O foco deste artigo é demonstrar uma característica fundamental do GRASP, que é permitir que ele possa ser executado de forma massivamente paralela. Este trabalho busca explorar isto, numa arquitetura GRID, com a aplicação de posicionamento de células.

O artigo está organizado da seguinte maneira. Primeiramente apresenta-se o GRASP em linhas gerais, destacando-se as vantagens sobre outras meta-heurísticas e suas possibilidades de implementação paralela. Depois mostraremos a implementação do sistema distribuído, que foi baseado na arquitetura de software ISAM [YAM 2003]. Finalmente, serão mostradas as simulações realizadas com a ferramenta distribuída de posicionamento. Por fim, são apresentadas as conclusões dos experimentos realizados, a fim de discutir programação distribuída aplicada a posicionamento de circuitos.

Greedy Randomized Adaptative Search Procedures

Um procedimento *GRASP* é uma heurística híbrida e iterativa de construção seguida de melhoramento. Seu processo iterativo divide-se em duas fases ou etapas distintas: fase de **construção**, onde uma solução inicial é construída aleatoriamente, baseando-se em critérios parciais de otimização semelhantes a métodos construtivos greedy; fase de **busca local**, onde a solução resultante da fase de construção é refinada, através de perturbações na estrutura, até que algum **critério de parada** seja satisfeito.

Os parâmetros do GRASP são basicamente o número de iterações desejado e um parâmetro de controle para a fase de construção, o qual define o quão aleatório ou o quão greedy é o passo construtivo. A solução resultante do GRASP será a melhor solução dentre todas as iterações executadas.O critério de parada é um parâmetro que dita quantas iterações sem melhora no resultado o algoritmo aceita. Assim, uma vez realizadas *maxGraspIterations* iterações sem melhorar o resultado atual, o algoritmo pára.

A semente aleatória utilizada deve garantir que serão geradas $m = max - min$ soluções iniciais diferentes, resultantes da fase de construção. Isso significa que, a partir do netlist do circuito, ao gerar um posicionamento inicial, as células serão posicionadas de acordo com tal semente.

Algoritmos de busca local modificam uma solução corrente em outra melhor, repetidamente, até que se alcance uma condição predeterminada. Em problemas onde a estrutura da solução e o mecanismo de perturbação permitem um número exponencial de movimentos, como é o caso do problema de posicionamento de células VLSI, a semente aleatória também será utilizada na fase de busca local para definir quais perturbações serão feitas, ou seja, qual modificação será tentada para a melhoria.

Dentre as características do GRASP, estas podem ser consideradas como vantagens sobre as demais metaheurísticas: poucos parâmetros de configuração próprios da meta-heurística, uso reduzido de memória, facilidade de paralelização e facilidade de implementação em problemas com busca local já modelada.

GRASP paralelo

A principal vantagem que o GRASP adquire ao trabalhar de forma totalmente aleatória, somada ao fato de que cada iteração é totalmente independente das anteri-

ores, é a fácil paralelização. A maneira intuitiva para paralelizar o algoritmo é atribuir uma iteração diferente para cada CPU. Não há comunicação e nem uma espécie de dependência entre uma iteração e outra, sendo que o dado que diferencia uma iteração das demais é somente a semente aleatória.

Como já foi mencionado, o mecanismo de busca do GRASP consiste em diversas buscas partindo de diferentes soluções do problema em questão. Assim, basta mudar a semente aleatória para que o posicionamento inicial (fase construtiva) seja diferente e assim todo o processo será distinto. No final, a iteração que atingir o melhor resultado é a vencedora.

Desta forma, a paralelização deste algoritmo deve considerar inicialmente uma distribuição de tarefas para executar em CPUs distintas. Depois disto, ao final do processo, deve haver um nodo responsável pelo recolhimento dos resultados e escolha do melhor resultado para reportar ao usuário.

Sistema distribuído para posicionamento

O sistema distribuído foi modelado como uma aplicação mestre/escravo, fazendo uso da arquitetura de software ISAM[1]. Nessa arquitetura, há duas entidades: um mestre (*master*), encarregado de gerenciar os trabalhos (*jobs*) a serem executados, e m trabalhadores (*workers*). Os trabalhadores se conectam ao mestre, e seu ciclo de vida consiste em (1)requisitar trabalho, (2)executar o trabalho e (3)informar ao mestre o resultado da computação. No contexto do posicionamento, chamamos de unidade de trabalho (*work unit*) um circuito o qual devemos realizar o posicionamento e uma semente aleatória (que corresponde exatamente a uma iteração do GRASP).

O mestre controla os escravos, informando a cada requisição de unidade de trabalho qual a semente aleatória a ser utilizada. A partir das execuções em paralelo dos trabalhadores, são gerados diferentes posicionamentos, com diferentes custos. O mestre deve manter o circuito posicionado com o menor custo.

Resultados

Os resultados mostrados na figura 1 foram obtidos pela ferramenta de posicionamento distribuída proposta neste artigo. Para a simulação, foram experimentados dois circuitos combinacionais, um deles contendo apenas **100 células**, e o outro contendo **1024 células**. O ambiente de simulação foi o seguinte: o **mestre** rodando numa estação *Sun Blade 2000*, as simulações de 1 a 7 máquinas utilizaram somente *Sun Blade 150* e uma estação *Sun Blade 100* foi utilizada nas simulações com 8 máquinas.

Todas as simulações processaram 51 sementes aleatórias ($[0..50]$), ou seja, havia 51 trabalhos a serem distribuídos a m máquinas. No circuito de 100 células o parâmetro *maxGraspIterations* foi setado para 10, enquanto no circuito com 1024 células esse parâmetro foi setado em 1 (o que nos dá um resultado ruim). Mantendo-se inalterados os valores dos parâmetros do algoritmo para um mesmo circuito, os resultados gerados foram os mesmos em todas as simulações, dependendo apenas do circuito (100 ou 1024 células).

[1] http://www.inf.ufrgs.br/~isam

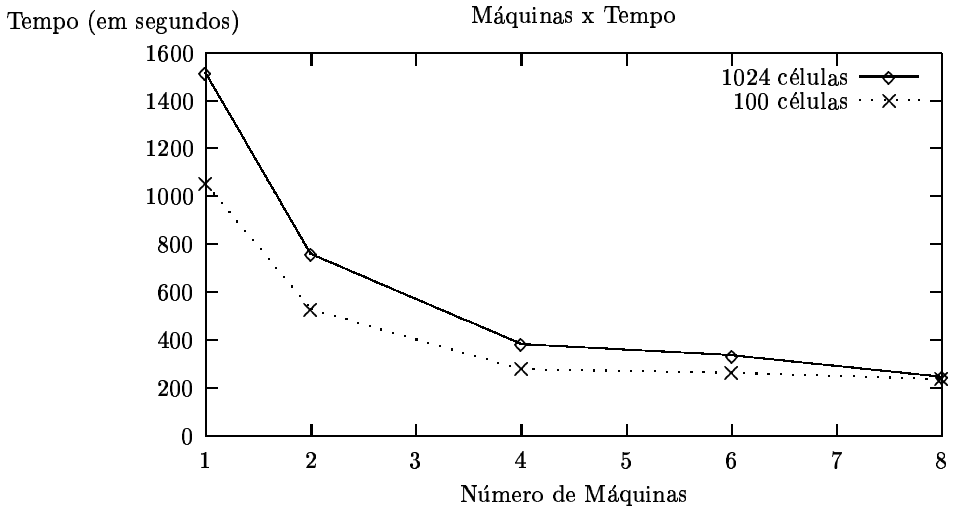

Figura 1: Posicionamento de circuitos com 100 e 1024 células

Considerações finais

Diante do posicionamento de circuitos complexos, dado o alto tempo de processamento em cada computador e o fato de haver comunicação entre os nodos apenas no início e no fim da computação, temos uma aplicação de grossa granularidade. Além disso, o sistema mostrou ser altamente escalável: o acréscimo no número de máquinas conectadas ao mestre tende sempre aumentar a eficiência do sistema, pois não há comunicação entre nodos trabalhadores (nodos trabalhadores não têm conhecimento uns dos outros) e a comunicação entre mestre e trabalhador é pequena. Assim, temos uma diminuição quase linear do tempo de execução com o aumento do número de clientes.

O sistema mostrou-se eficiente em ambiente *grid*, dadas as seguintes características: grossa granularidade de dados (elevado tempo de processamento por trabalho), alta escalabilidade e independência de plataforma (o sistema foi desenvolvido em Java).

Referências

[HEN 2002] HENTCHKE, R. **Algoritmos para o Posicionamento de Células em Circuitos VLSI**. 2002. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[MOR 2000] MORELLI, C. **Estratégias de Tendenciosidade no GRASP**. 2000. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[YAM 2003] YAMIN, A. C. et al. Towards merging context-aware, mobile and grid computing. **The International Journal of High Performance Computing Applications**, v.17, n.2, p.191–203, 2003.