RouterServer: Agente Roteador Paralelo de Células VLSI

EVANDRO DE ARAÚJO JARDINI[1]
DILVAN DE ABREU MOREIRA[1]

ICMC – Instituto de Ciências Matemáticas e Computação
eajardini@yahoo.com.br
dilvan@icmc.sc.usp.br

Abstract. The *Agents 2* system was developed to automate the cell generation for integrated circuits (standard cells). Many server agents compose the system: the *Placer* (that places the components of a circuit) and many *Routers* (that wires the circuits' components). The servers work distributed over a computer network and system scalability increases as new computers are added to the network. However, the system is not capable of using the resources of computers with more than one CPU.

To solve this problem a new router server was developed, called *RouterServer*. It was developed using the Java language and a multithreaded design. It allows the *Agents 2* system to use the parallelism that exists in multiprocessors machines with shared memory. At the same time it retains its scalability in distributed networked systems.

Keywords: Programação Paralela, Linguagem Java, Agentes de Software e células VLSI.

1 Introdução

O projeto de circuitos integrados é normalmente baseado numa estrutura hierárquica de especificações. Os módulos em níveis superiores são compostos por submódulos e estes são formados por outros submódulos e assim sucessivamente. No fim dessa estrutura estão módulos formados apenas por transistores. Estes módulos são denominados de células-padrão (*standard cells*) [01].

O modo tradicional de criar o *layout* de circuitos ASIC (*Application Specific Integrated Circuits*) requer um projetista humano para interagir com um programa do tipo CAD (*Computer Aided Design*). Ele utiliza uma metodologia de *layout* baseada em bibliotecas de células padrão. Essa metodologia é boa para os projetos ASIC porque os processos realizados para geração de *layouts* (principalmente posicionamento e roteamento dos componentes) podem ser automatizados, diminuindo o tempo gasto no projeto e aumentando a sua confiabilidade [1].

Porém, essa metodologia possui inconvenientes em relação à manutenção da biblioteca de células no que se refere à atualização do processo de manufatura e adicionalmente, o número total e a variedade de células existentes, pois, algumas células requeridas em um projeto podem não existir na biblioteca forçando uma adaptação do projeto. Em conseqüência disso, o desempenho do circuito pode ser sacrificado [01].

A solução para os problemas apresentados é a geração de células-padrão específicas à medida que surgirem necessidades em projetos. Utilizando-se ferramentas para geração automática de *layout* de células-padrão obtém-se grandes vantagens. Elas são capazes de gerar *layouts* para uma grande quantidade de circuitos SSI (*Small Scale Integration*) para diferentes processos de manufatura. Como essas ferramentas produziriam células que se encaixariam exatamente nas necessidades do projeto para diferentes tecnologias utilizadas para fabricação, elas solucionariam as duas principais inconveniências da metodologia baseada em células.

O sistema *Agents 2* tem justamente o objetivo de gerar automaticamente *layouts* de células-padrão para circuitos integrados. O sistema foi desenvolvido baseado na tecnologia de agentes de software e era, originalmente, escrito em C++ e Lisp. O sistema original foi portado para Java e é agora formado por dois programas servidores. Os programas e suas respectivas funções são:

- Servidor **Placer**: Posiciona os componentes pertencentes ao circuito gerando diversos layouts com posicionamentos diferentes. Cada layout

gerado é enviado a um servidor *Router* para ser roteado.

- Servidor **Router**: Este servidor tem o objetivo de rotear as trilhas que interconectam os componentes posicionados em um circuito eletrônico.

As informações relativas às regras de projeto que o layout tem de obedecer (que variam de acordo com o processo de fabricação) são lidas de um arquivo pelo *Placer* e transmitido para todos os *Router*s. Ambos os servidores, *Placer* e *Router* foram desenvolvidos para trabalharem de forma distribuída em uma rede de computadores. Isso permite um ganho de escalabilidade à medida que o número de servidores *Router*s aumenta.

Apesar do sistema *Agents 2* explorar o paralelismo inerente aos sistemas distribuídos em rede, espalhando os servidores *Router* por vários computadores através da rede, ele não consegue aproveitar recursos de computadores que possuam mais de um processador.

Para obter este proveito, o servidor *Router*, agora denominado *RouterServer*, teve de ser reescrito. Este artigo descreve a maneira como isso foi realizado e os resultados obtidos.

2 Projeto de Circuitos Integrados

Apesar do foco desse trabalho não ser a geração de *layout* de circuitos integrados, para melhor entendê-lo é necessário algum conhecimento sobre a tecnologia básica da fabricação de circuitos integrados. Um circuito integrado é formado por componentes eletrônicos (transistores, diodos, capacitores, resistores, etc) colocados numa pastilha de silício, denominada *chip*, através de um processo de fabricação [02].

Diferente do processo de fabricação de automóveis, onde as peças são adicionadas separadamente, no processo de fabricação de um circuito integrado não há a adição de componentes de forma separada, o processo é feito como um todo. Todos os componentes são fabricados ao mesmo tempo em camadas. Essa tecnologia de fabricação é denominada tecnologia de produção integrada [02]. Todo o circuito é encapsulado em um único invólucro e compartilha os mesmos pinos de contatos. Essa técnica possibilita reduzir os custos de fabricação dos circuitos.

O processo de fabricação de um circuito integrado é divido em três etapas básicas [02]:

- Etapa 1: Projeto do circuito e fabricação das máscaras.
- Etapa 2: Obtenção de camada de Silício, Fotolitografia, Corrosão, Epitaxia, Metalização e Teste dos chips.
- Etapa 3: Corte das pastilhas, Soldagem, Encapsulamento e Teste.

Para esse trabalho a primeira etapa, o projeto do circuito, é a mais importante. Antes de iniciar o processo de fabricação, é elaborado um projeto para a confecção do circuito. O projeto é a especificação de todo circuito que será desenvolvido. Fazendo uma analogia, o projeto do circuito pode ser comparado a uma maquete de uma construção, na qual se tem um esboço de toda a obra. O projeto é composto de várias máscaras. Essas máscaras são *layouts* específicos para cada camada que será sobreposta para formar o chip. É na automação dos *layouts* de mascaras que o sistema *Agents2* é utilizado/

O *layout* de cada máscara está relacionado ao material presente na camada do circuito que ela representa. Resulta da função que o circuito deve desempenhar.

3 O Sistema *Agents2*

O sistema Agents é um conjunto de programas criados para gerar automaticamente *layouts* de células-padrão nas tecnologias CMOS, BICMOS e circuitos mistos digitais/analógicos. Ele foi desenvolvido originalmente [01] nas linguagens C++ e Lisp e foi posteriormente portado para a linguagem Java (*Agents 2*). O *Agents 2* é formado por dois módulos: os servidores *Placer* e *Router*. Esses módulos podem rodar distribuídos em rede usando o modelo cliente/servidor, com vários servidores *Router*s para cada servidor *Placer*.

O *layout* gerado pelo *Agents 2* não usa um *grid* virtual. Ele é um *layout* de máscara pronto para ser usado na confecção do circuito integrado. O programa não usa *grid* virtual nem durante o posicionamento dos componentes, nem durante o roteamento destes, todas as operações com *layou*t são executadas ao nível de *layout* de máscara.

3.1 Posicionamento de Componentes

O servidor *Placer* é quem recebe dos clientes a descrição dos circuitos a serem gerados [01]. Ele trabalha em três fases:

- Geração de colunas de transistores MOS que tem suas portas conectadas entre si ou de outros

componentes que estejam interconectados (transistores bipolares, por exemplo).

- União das colunas de FETs para formar grupos. Para se unir, colunas de fets tem de partilhar um certo numero de conexões via dreno ou fonte. A idéia é unir fets que possam ser difundidos numa mesma linha de difusão.

- Posicionamento de grupos de componentes usando o algoritmo genético e envio do circuito resultante para um servidor *Router* para roteamento. Vários circuitos são gerados e enviados para roteamento, até que um deles seja roteado satisfatoriamente.

O servidor *Placer* usa diversos servidores *Router* ao mesmo tempo, desse modo ele pode gerar vários circuitos, solicitar aos roteadores que os conectem e coletar os resultados à medida que os circuitos são roteados. Essa técnica explora o processamento distribuído, repartindo a tarefa de rotear circuitos pelos vários computadores ligados a uma rede.

3.2 Roteamento de Circuitos

O *Router* foi desenvolvido tentando imitar o modo pelo qual os projetistas humanos utilizam um sistema de CAD (*Computer Aided Design*) para rotear circuitos. O projetista toma todas decisões importantes sobre o projeto, como por exemplo, onde os fios serão traçados, a qualidade do roteamento, a necessidade de reroteamento dos fios, etc. O sistema CAD fornece ao projetista ferramentas para representar e manipular o projeto.

O *Router* possui os componentes responsáveis por imitar o papel do projetista e o papel do CAD. O papel do projetista é feito pelos agentes de software *RouterExpert* e *Connect* e o papel do CAD é feito pelo objeto *Design*.

O objeto Design guarda os dados do projeto assim como os métodos para analisá-lo ou mudá-lo. Como ele faz o papel do CAD, ele prove os outros dois agentes com os meios para coletar informações sobre o projeto (se dois fios se chocam, a área ocupada por uma subnet, etc.) e os métodos para implementar mudanças no projeto (inserir fios, destruir fios, etc.).

3.2.1 Os agentes RouterExpert e Connect

O *Router* foi desenvolvido utilizando a tecnologia de agentes de softwares. Agentes são componentes de software que se comunicam com seus pares através da troca de mensagens usando uma linguagem de comunicação [03]. Essa característica permite aos agentes cooperarem entre si. Os agentes podem ser tão simples quanto uma sub-rotina ou ser entidades maiores como um programa com algum grau de autonomia [01].

A tecnologia de agentes de software é empregada em dois níveis: macro-agente e micro-agente.

O macro-agente *RouterExpert* é o responsável por criar e eliminar agentes *connects*, verificar a qualidade dos fios, determinar o custo de uma rota, etc.

Os agentes *Connects* são micro-agentes responsáveis por encontrarem as rotas de interconexão dos componentes de um circuito. A idéia é se ter vários agentes *connects* trabalhando em paralelo na procura das rotas.

Juntos, os agentes *RouterExpert* e *Connect* implementam o roteamento usando uma variação do algoritmo *maze routing* [04]. Esse algoritmo baseasse num *grid* retangular que representa o circuito. Ele mapeia este *grid* determinando quais células estão livres ou bloqueadas e para cada célula é atribuído um custo (figura 1). O algoritmo analisa e expande os vértices da árvore de busca gerada até encontrar o objetivo.

Uma variação desse algoritmo introduz o conceito de pontos interessantes [05]. Ao invés de expandir os vértices em todas as direções, os agentes *Connect* expandem diretamente para um ponto interessante. Um ponto é considerado interessante (figura 2) quando está alinhado com o ponto objetivo (figura 2A) ou com as extremidades de um obstáculo (figuras 2B e 2C). Na verdade à extremidade de um obstáculo é somada uma margem de segurança proporcional à distância mínima de separação do obstáculo e do fio em roteamento mais a metade da largura desse fio.

3.2.2 Funcionamento dos Agentes RouterExpert e Connect

Quando o servidor *Router* recebe um circuito do servidor *Placer*, ele o recebe como um objeto *Design* serializado. Ele recompõe o objeto e o passa para o agente *RouterExpert* dar início ao processo de roteamento.

De posse do objeto *Design*, o *RouterExpert* realizará, inicialmente, roteamentos simples. Ele conecta pontos que já estejam alinhados em polissilício ou difusão.

bloqueado	5	6	7	8	9
3	4	5	6	7	8
2®	3®	4®	5®	6®	7 (Objetivo)
1-	2	3	bloqueado	7	8
0- (Origem)	bloqueado	bloqueado	bloqueado	bloqueado	7
1	2	3	4	5	6

Figura 1: Circuito mapeado em um *grid* para utilização do algoritmo *maze routing*.

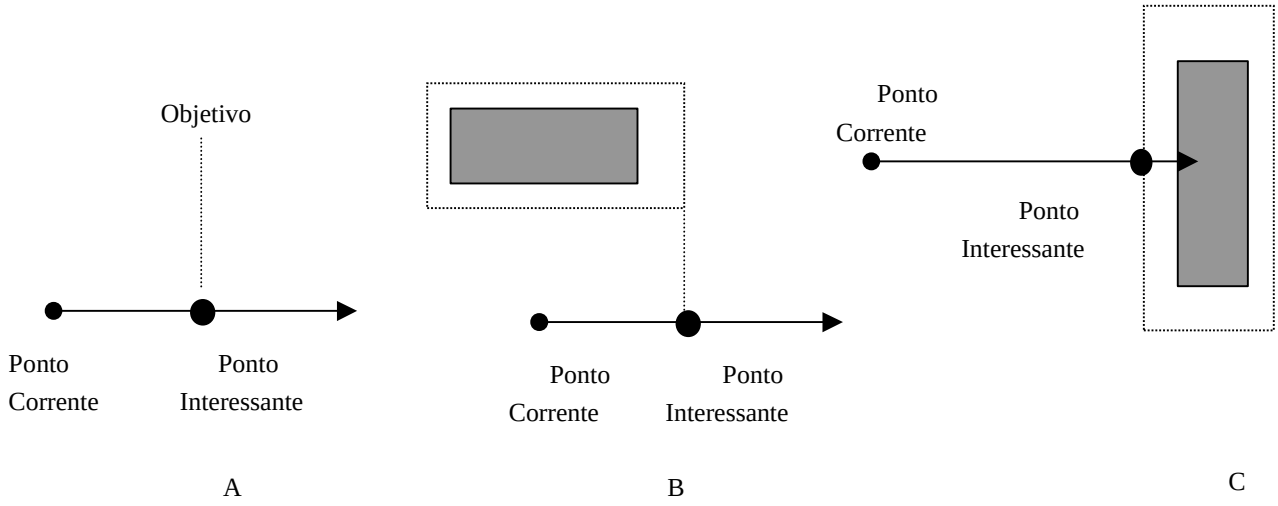


Figura 2: Pontos Interessantes.

Para conectar os demais componentes restantes, os nós desconectados são colocados em uma lista, ordenados por importância e tamanho. Os menores serão os primeiros da lista. Deste estágio em diante, serão utilizadas apenas as camadas de *polissilício*, *metal1* e *metal2* para interconexão.

No objeto *Design*, cada nó contém uma lista de *subnets* que conectam parcialmente os componentes do nó (lista *routingNets*). Uma *subnet* é o conjunto de fios que interligam parte dos componentes que pertencem ao mesmo nó do circuito. Já uma *net* conecta todos os componentes do mesmo nó. O método *connectNode* do agente *RouterExpert* utiliza-se desta lista para conectar as *subnets* pertencentes a um nó. Ele faz isto utilizando o método *connectSubnet*. Se o método não conectar uma *subnet*, ela será colocada no final da lista e a seguinte será conectada. O método retorna uma condição de erro, se alguma *subnet* não for totalmente conectada.

A conexão das *subnets*, por fios¸ está baseada nos pontos interessantes e é realizada pelos agentes *Connect*. Eles analisam os pontos interessantes que estão próximos ao seu ponto de origem. Um agente C*onnect* é criado para cada ponto interessante. A partir desta análise, várias operações podem ser realizadas pelos *Connects*: eles

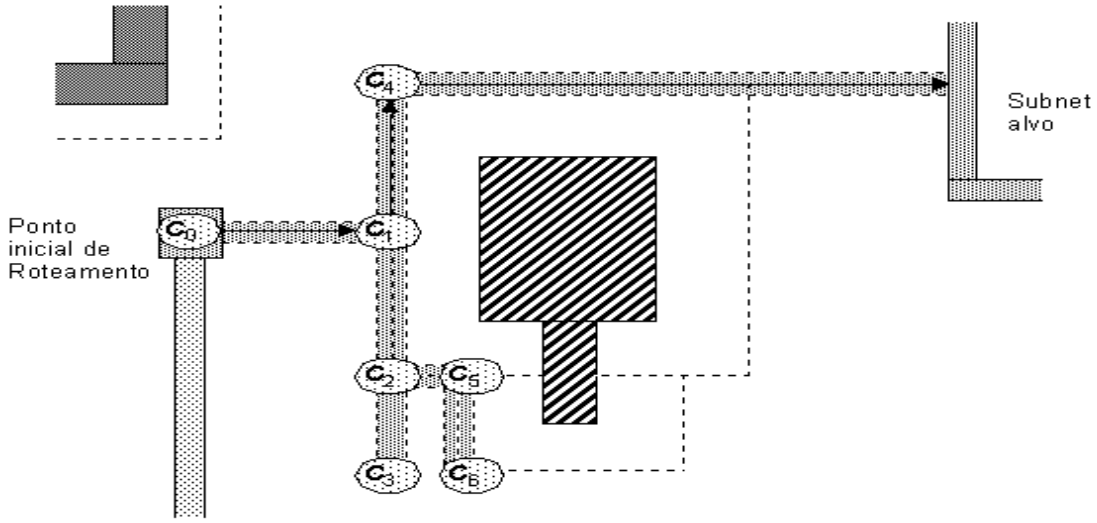


Figura 3 - Agentes *Connects* procurando por pontos

podem mudar de camada, conectar o fio com o destino, criar um novo pedaço de fio, se reproduzirem ao encontrar um obstáculo no caminho, etc.

O objetivo dos agentes *Connect* é alcançar um ponto na *subnet* destino. O funcionamento básico dos agentes *Connect*s inicia-se com a criação de um agente *Connect* em um ponto de origem dentro do *layout* (ponto C_0 na figura 3). Este agente analisará os pontos interessantes a sua volta. Para cada ponto interessante a ser explorado é criado um novo agente (pontos C_1, C_2, .., C_n) . Os agentes irão se reproduzir até encontrarem seu objetivo: a *subnet* destino.

3.2.3 Paralelismo Simulado

O agente *RouterExpert* controla uma população de microagentes *Connect* e a maneira como eles executam o roteamento. A idéia é ter uma população de agentes simples (daí o nome de microagente para o *Connnect)* cooperando entre si de forma a encontrar uma solução. Se um agente encontra um novo ponto interessante, ele se reproduz. Se ele gastou todas as suas opções de procura, ele morre. Se completar um fio, ele envia este fio para o agente *RouterExpert*. O *RouterExpert* então, se encarrega de "matar" todos os agentes *Connects* cujo custo do fio seja maior ou igual ao encontrado.

Para implementar essa "fazenda" de agentes *Connect*, controlada pelo agente *RouterExpert*, foi criada uma espécie de máquina virtual capaz de simular um sistema paralelo usando filas (figura 4). Um agente *Connect* por vez é rodado. À medida que ele roda, ele pode sugerir *Best Wires* ao *RouterExpert* ou "procriar" e gerar novos agentes *Connect* para pesquisar outros pontos interessantes do projeto. Os novos agentes *Connect* criados são postos em duas filas:

- *Agents List* - Uma fila com agentes *Connect* comuns.
- *Agents Queue* - Uma fila para agentes *Connect* que iriam implementar a ação de desfazer conexões de um nó que bloqueava o seu caminho.

Essas duas listas são criadas em separado porque a operação de desfazer um nó é muito custosa computacionalmente (já que o nó desfeito terá de ser posteriormente refeito). Assim primeiramente são executados os agentes na *Agents List* e quando essa estiver vazia, é executado pelo menos um agente da *Agent Queue*.

Essa máquina virtual foi criada em C++, ao invés de se usar *thread*s e rodar os agentes *Connect* realmente em paralelo, porque na época não se dispunha de um computador com sistema operacional que permitisse o uso de múltiplos *thread*s. Ela foi mantida intacta no porte do

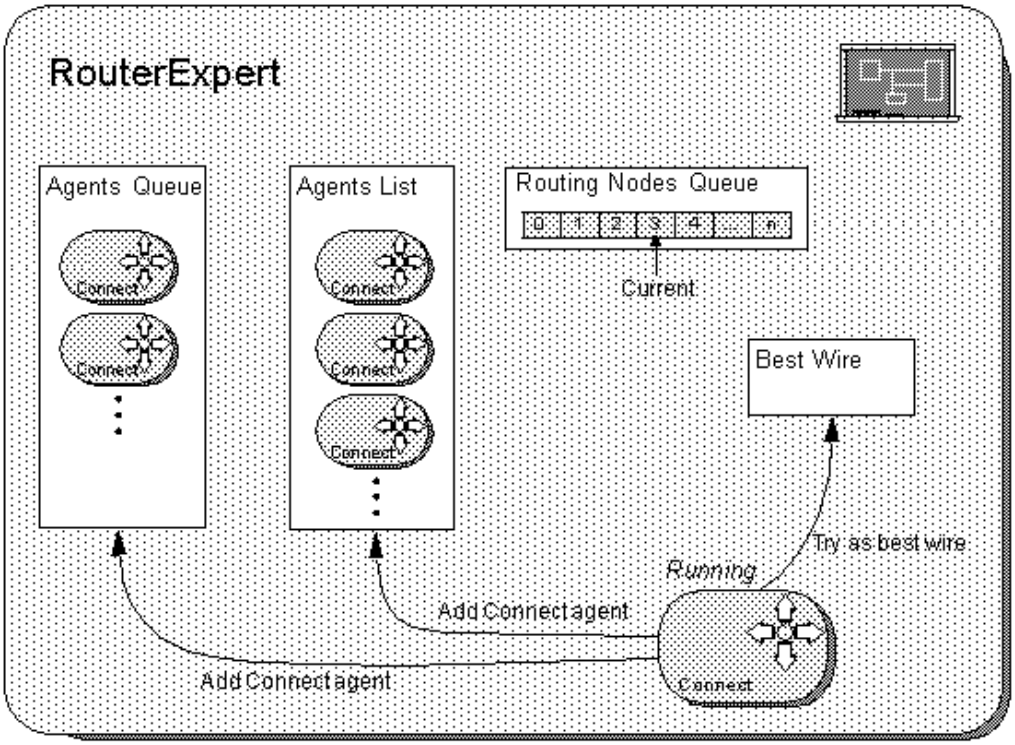


Figura 4: Máquina Paralela Virtual.

servidor *Router* para a linguagem Java, gerando uma versão serial incapaz de usar os recurso de múltiplos *thread*s de execução em máquinas com muitos processadores.

3.2.4 Paralelização da Máquina Virtual

A paralelização do algoritmo foi realizada na máquina virtual (figura 4) no momento em que houvesse a necessidade de executar agentes *Connect*s para procurar pontos interessantes, permitindo a execução simultânea desses agentes.

Quando um agente *Connect* é executado ele começa a analisar os pontos interessantes a sua volta. Para cada ponto interessante a sua volta, ele cria outros *Connect* agentes. Ao invés desses agentes serem colocados na lista *AgentsList*, que não existe mais, o agente *Connect* pai chama um método no agente *RouterExpert* que os põe numa lista. O *thread* de execução do *RouterExpert* tem um *loop* que cria novos *threads* para os agentes *Connect* dessa lista e os executa imediatamente (é nesse *loop* onde se pode controlar o número de agentes *Connect* que poderão rodar em paralelo, de um único agente até sem limites).

O resto de seu código teve de ser alterado para resolver problemas de sincronismo entre os vários agentes *Connect* rodando e o *RouterExpert*. A lista *AgentsQueue* não foi removida, pois ela guarda os agentes *Connect* que irão implementar a ação de desfazer conexões de um nó (que bloqueia o seu caminho) e desfazer um nó é uma operação muito custosa. A lista *AgentsQueue* permite guardar esses agentes retardando a sua execução dando chance que um caminho menos custoso seja encontrado antes.

Na versão seqüencial do roteador não ocorriam problemas de sincronismo, uma vez que só havia um agente *Connect* sendo executado por vez. Entretanto na

nova versão, o problema de sincronismo foi introduzido, sendo um dos principais obstáculos a conclusão do projeto.

A solução foi analisar o código do agente *RouterExpert* e verificar quais métodos eram acessados pelos agentes *Connects*. Essa análise identificou que os seguintes métodos deveriam ser sincronizados:

- **Método *AddAgent***: Este método é chamado pelos agentes *Connects* toda vez em que há a necessidade de criar novos agentes *Connect* para pesquisa de novos pontos interessantes.
- **Método *AddAgentInQueue***: Este método é chamado pelos agentes *Connects* toda vez em que há a necessidade de criar novos agentes *Connect* que irão desfazer conexões em algum nó que os bloqueia.
- **Método *TryAsBestWire***: Quando um agente *Connect* alcança uma *subnet* com seu fio, ele chama esse método para propor esse fio ao *RouterExpert* como o melhor fio. Esse método determina se o fio encontrado é melhor que o último proposto (cada fio tem um custo proporcional a seu material e comprimento).

Outro problema de sincronismo acontece quando um fio considerado aceitável pelo *RouterExpert* é mandado por um agente *Connect*. Esse fio vai ser então usado para conectar as duas *subnet*s. Mas vários agents *Connect* podem ainda estar rodando, o problema é como pará-los sem criar nenhuma inconsistência de variáveis ou *interlock* entre *thread*s. Os próprios agentes *Connect* resolvem esse problema:

- Eles constantemente monitoram se seus fios são piores que o melhor fio que o *RouterExpert* tem, caso isso aconteça eles morrem.
- Eles monitoram se o *thread* deles foi interrompido pelo *RouterExpert* (chamando o método `interrupted()` no *thread* do agente *Connect*), caso isso aconteça eles morrem.

Finalmente algumas adaptações tiveram de ser feitas em algumas estruturas de dados internas que não possuíam suporte a *multithreads*.

4 Resultados

Uma das motivações para realização deste trabalho foi o interesse em pesquisas envolvendo a exploração dos recursos fornecidos por computadores multiprocessados. A idéia é que a medida em que há um aumento da utilização desses recursos o tempo final gasto no processo de roteamento dos circuitos deve diminuir. O objetivo dos testes aqui reportados foi testar se as modificações realizadas no servidor *RouterServer* permitiram maior escalabilidade à medida que o número de processadores disponíveis em um computador aumentava. Além disso, o sistema deve manter a escalabilidade ambientes distribuídos.

Foram feitos dois conjuntos de testes: O primeiro, testou o desempenho do servidor *RouterServer* em relação à sua utilização com diferentes números de *threads* e processadores. O servidor *RouterServer* foi executado em apenas uma máquina de cada vez. Esses testes analisaram sua escalabilidade e também tentaram determinar o número ótimo de *threads* simultâneos para um melhor desempenho em cada tipo de configuração. O segundo conjunto de testes analisou o desempenho do programa como parte integrante do sistema *Agents 2* sendo executado de forma distribuída numa rede de computadores heterogêneos.

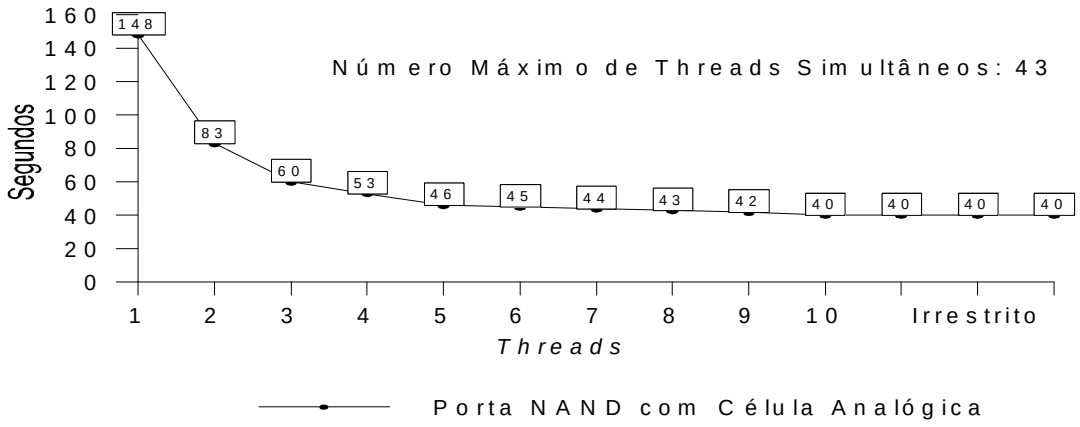
4.1 Testes em Máquinas Individuais

Para a realização dos testes foram utilizadas as seguintes máquinas:

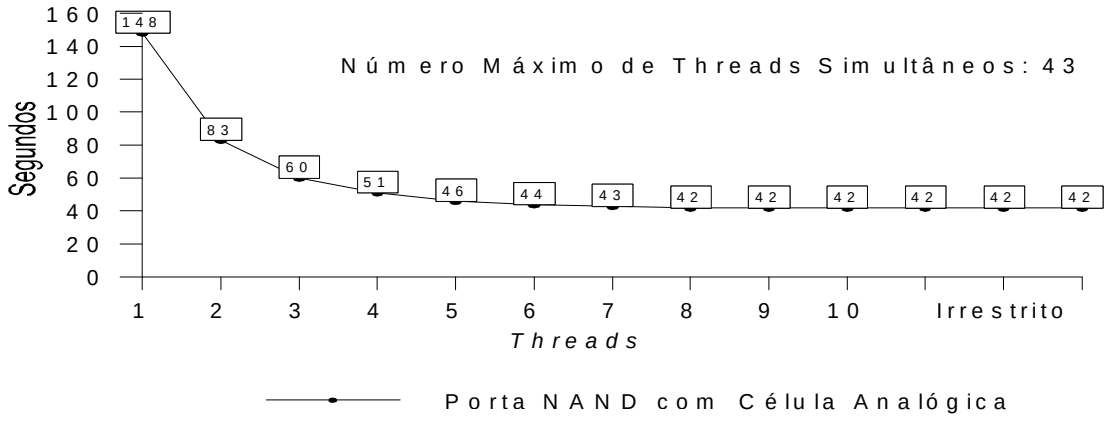
- Computador 1 – Computador Intergraph com 4 processadores Pentium Pro de 200 Mhz utilizando Windows NT 4.0.
- Computador 2 – Computador com 2 processadores Pentium Pro de 200 Mhz utilizando Windows NT 4.0.
- Computador 3 – Notebook com 1 processador K6 II 400mhz utilizando Windows 98

O conjunto de testes em máquinas individuais foi realizado utilizando circuitos com componentes já posicionados permitindo verificar o desempenho do *RouterServer em* computadores diferentes.

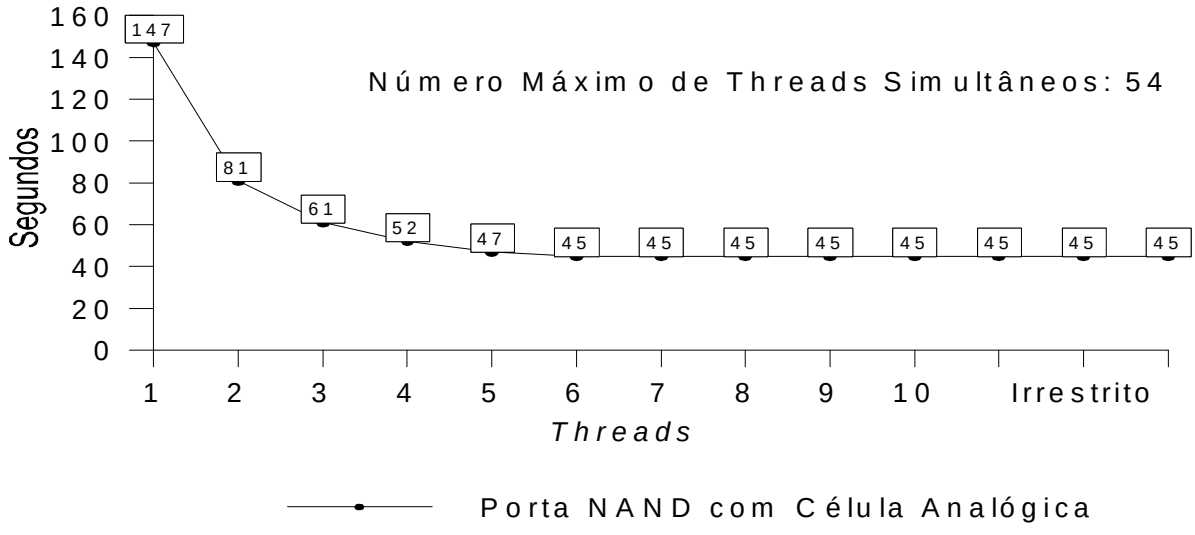
O gráfico da figura 5 demonstra o tempo levado para rotear uma porta NAND com célula analógica (figura 6) roteada nos computadores 1, 2 e 3 respectivamente.



A



B



C

Figura 5: Resultados dos tempos gastos para rotear uma porta NAND com Célula Analógica nos computadores A: Computador 1 B: Computador 2 C: Computador 3

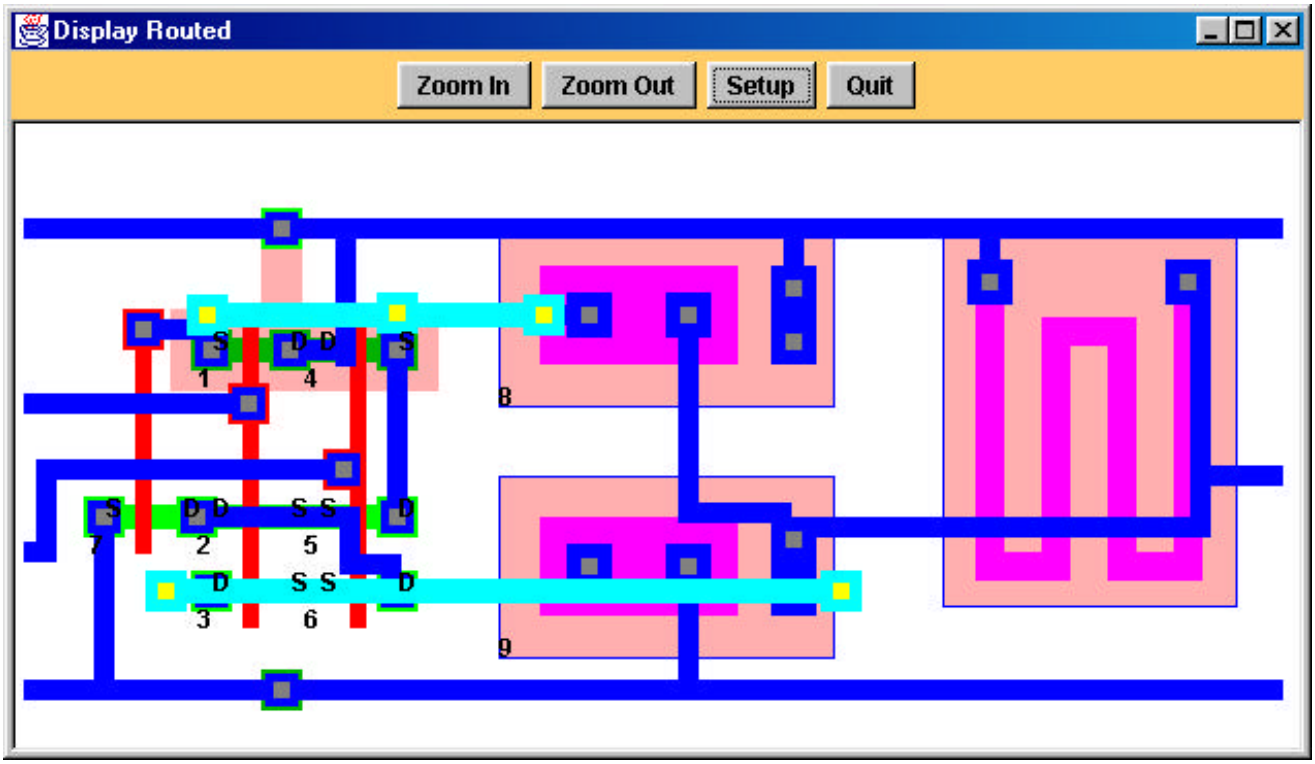


Figura 6: Porta NAND com Célula Analógica

Os gráficos apresentados na figura 5 comprovam a escalabilidade do *RouterServer* em relação ao número de *threads* disponíveis. Primeiramente, como já foi discutido, se for feita uma análise dos gráficos separadamente, fica comprovado que à medida que o número de *threads* disponíveis aumenta, o tempo de roteamento dos circuitos diminui até um ponto limite. Agora, se os gráficos dos computadores 1, 2 e 3 forem comparados uns com os outros, comprova-se a escalabilidade do *RouterServer* em relação ao número de processadores disponíveis. À medida que o número de processadores disponíveis vai aumentando, os tempos gastos nos roteamentos dos circuitos tendem a diminuir.

Vale ressaltar que a diferença entre os tempos de execução entre as máquinas não foi maior pelo motivo de que apenas um só *Router* gerando um só circuito não necessita de muito poder de computação.

Outro problema que deve ser relembrado é que as máquinas envolvidas em todos os testes não diferem apenas em números de processadores, mas também em velocidade, memória, *motherboards*, etc. Assim os valores obtidos não podem ser considerados valores precisos e sim aproximações razoáveis dos verdadeiros resultados.

4.2 Testes em Computadores Distribuídos

Este trabalho também teve o comprometimento de manter a escalabilidade do sistema em ambientes distribuídos obtida na versão original do sistema *Agents*. O objetivo dos testes é de mostrar a diminuição dos tempos de roteamentos à medida que novas estações são acrescentadas.

Para a realização dos testes foram utilizados os seguintes computadores:

- Computador 1 – Microcomputador Intergraph com 4 processadores Pentium Pro de 200 Mhz utilizando Windows NT 4.0.

- Computador 2 – Microcomputador com 1 processador Pentium II de 266 Mhz utilizando Windows NT 4.0.
- Computador 3 – Microcomputador com 2 processadores Pentium II de 350 Mhz utilizando Windows NT 4.0.

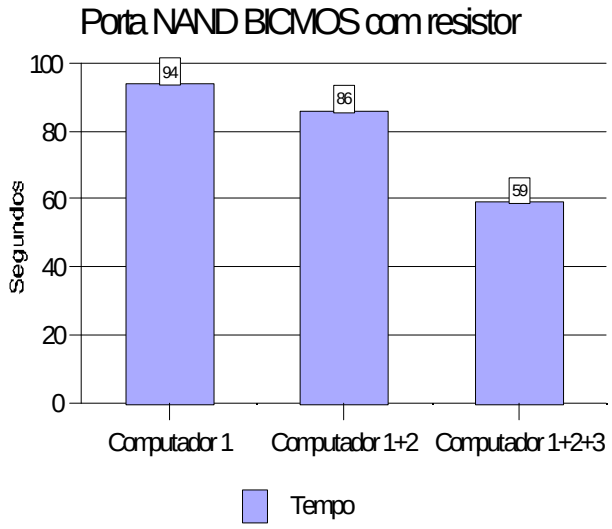


Figura 7: Teste distribuído utilizando circuito de Porta nand

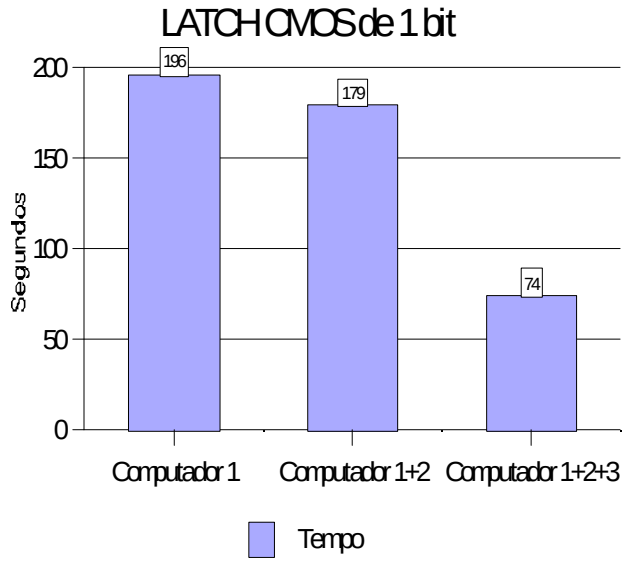


Figura 8: Teste distribuído usando o circuito latch CMOS.

Os gráficos das figuras 7 e 8 seguintes demonstram os resultados obtidos.

Os gráficos apresentados demonstram que, além de obter escalabilidade em computadores com memória compartilhada, o sistema *Agents* 2 (com o novo roteador) ainda mantém a escalabilidade obtida no sistema original rodando distribuído. Os resultados dos testes realizados podem variar um pouco a cada execução dos programas devido a fatores ligados as diferenças de configuração dos computadores e uso da rede.

5 Conclusão

Este artigo apresentou o sistema *Agents2*. O sistema é formado por dois servidores: O servidor Placer e o Servidor Router. Na versão original, os servidores não eram capazes de explorar computadores multiprocessados. No trabalho realizado foram feitas modificações no servido *Router*, denominado *RouterServer*, para que aproveitasse recurso de computadores multiprocessados. O objetivo deste artigo foi o de demonstrar como isso foi realizado e os resultados obtidos com as modificações. Pelos gráficos apresentados, conclui-se que os objetivos desejados foram

6 Referên

[01] – Moreira,
for leaf Cell Ge
Canterbury, 199

[02] – Enderlei
Universidade de

[03] Geneseret
Communication
48-53, 14

[04] Moore, E.
of the Compu
Havard Univ. Press. Cambridge Mass., vol.30, 1959, pp. 285-292.

[05] Arnold, M.H., Scott, W.S. An Interactive Maze *Router* with Hints. Proc. Of the 25th ACM/IEEE Design Automation Conference, 1988 IEEE, p 72-67