APLICANDO PROCESSAMENTO PARALELO EM UM ALGORITMO DE ROTEAMENTO

REGIS ZANANDREA[1]; FELIPE S. MARQUES[1]; LEOMAR S. DA ROSA JR.[1]

[1] *Universidade Federal de Pelotas – {rzanandrea, felipem, leomarjr}@inf.ufpel.edu.br*

1. INTRODUÇÃO

Com o crescimento da tecnologia, os circuitos digitais se tornaram cada vez mais complexos. Junto com a redução do tamanho dos transístores e de outros componentes, possibilitou que circuitos digitais possuam enormes quantidades de portas lógicas. Assim, os processadores ficaram extremamente mais rápidos. O mercado também necessitou de uma tecnologia mais rápida, exigindo uma agilidade maior na concepção de circuitos integrados. A concepção manual, de circuitos digitais, era algo inviável, pois a sua complexidade era muito alta, assim necessitou de ferramentas que automatiza-se esse processo, tais ferramentas são chamadas de ferramentas de CAD.

As ferramentas de CAD, tem como o principal objetivo a automatização do processo de síntese física e lógica, sendo esses os processos mais complexos a ser feitos dentro de uma ferramenta de CAD. Entretanto, elas podem oferecer uma área para a visualização do circuito. Isso ajuda, ao usuário, a compreender melhor o circuito, na sua funcionalidade, na detecção de erros, no aperfeiçoamento e em entre outros conhecimentos. Junto com tal área, é adicionado uma ferramenta de edição, onde se pode excluir, editar e adicionar algumas células ao circuito. Para a visualização dos circuitos, as ferramentas utilizam algoritmos de posicionamento e roteamento. Tais algoritmos tem o propósito de diminuir o comprimento de fio (*wirelenght*) e tempo de atraso (*delay*).

Os algoritmos de posicionamento visam descobrir a melhor posição pra cada célula, de tal forma que minimize alguns fatores, como por exemplo o comprimento de fio. Cada célula possui um conjunto de terminais, que são entradas e saídas de sinais. Os algoritmos de roteamento procuram por tais terminais, para conectar as células, achando sempre o menor caminho. Algoritmos de posicionamento apenas posicionam e não conectam as células, deixando essa tarefa para algoritmos de roteamento.

A tarefa de roteamento de circuitos digitais está dividida em duas partes: roteamento global e roteamento detalhado. O roteamento global conecta os terminais roteando os fios através de canais. Tais canais, são espaços entre duas fileiras de células.

Existem circuitos com milhares de portas lógicas para serem posicionadas e roteadas, exigindo algoritmos rápidos para esse processo. Na literatura, há diversos algoritmos de posicionamento e roteamento, mas este trabalho irá abordar apenas algoritmos de roteamento, onde tais será aplicado o processamento paralelo. Algoritmos de posicionamento tem uma grande dependência de dados a cada iteração, e talvez, pode não ser tão útil aplicar o processamento paralelo.

Este trabalho, visa aplicar o paralelismo a algoritmos de roteamento de circuitos digitais, sendo aplicado por primeiro no algoritmo A* (HART, 1968). O algoritmo A* foi escolhido por ser um algoritmo clássico na literatura, e também por sua simplicidade de rotear circuitos. Existem além do A* diversos outros algoritmos na literatura, desde clássicos como o algoritmo *Maze Router* (LEE,

1961) até algoritmos mais recentes, como o algoritmo *ST Router* (GONÇALVEZ, 2014), que atualmente é o algoritmo estado da arte de roteamento de circuitos digitais, mas este trabalho está focado apenas no algoritmo A*. Tais algoritmos são de propósito geral, assim podem ser usados para roteamento detalhado ou roteamento global, entretanto, eles são mais adequados a roteamentos detalhados.

2. METODOLOGIA

Para a aplicação do paralelismo, foi escolhido o algoritmo A*, onde foi implementado na linguagem de programação C++. A ferramenta de programação paralela utilizada foi o OpenMP (OPENMP, 2015). O OpenMP é uma API para programação multi-processo, ou paralela, para multi-plataformas. Tendo suporte as linguagens C, C++ e Fortran. Sua característica principal é a simplicidade em seus comandos, tornando para o usuário, uma ferramenta de uso fácil e rápido.

A metodologia usada no algoritmo A* foi criar diversas *Threads*. *Thread* é uma maneira de dividir um processo em diversas tarefas para que possa ser executado em diversos núcleos do processador paralelamente.

Inicialmente o algoritmo inicia com o nó de origem. No segundo passo o algoritmo cria seus nós adjacentes do nó de origem e adiciona na fronteira. Até o momento não foi aplicado o paralelismo. A partir do segundo passo, a fronteira possui o tamanho quatro, pois o nó de origem sempre cria quatro nós adjacentes. A partir do terceiro passo, o algoritmo escolhe os melhores nós da fronteira que serão expandidos. A quantidade de nós expandidos é também o número de *threads* que executarão ao mesmo tempo, pois em cada *thread* será expandido um nó, criando seus adjacentes. Após as *threads* terminarem de executarem, todos os nós criados são armazenados, e o número máximo de nós criados a cada iteração é de dezesseis. O algoritmo volta para o terceiro passo, e continua executando até o destino ser encontrado.

Na Fig. 1, é demonstrado os quatros primeiros passos do algoritmo A* com paralelismo. Cabe ressaltar que com a aplicação do paralelismo, a quantidade de nós criados é muito maior. Pode-se notar na Fig. 1, que em relação ao algoritmo A* sem paralelismo o crescimento em direção ao destino é muito maior, pois a cada iteração com o paralelismo é criados mais nós a cada iteração do que sem o paralelismo, isso acaba gerando um aumento de memória utilizada pelo algoritmos, entretanto, tal aumento de memória em circuitos pequenos não é significante, sendo apenas considerável para circuitos maiores, que precisam de mais tempo para serem roteados.

Figura 1: Primeiros passos do A* com paralelismo

3.RESULTADOS E DISCUSSÃO

Para a geração dos resultados, o algoritmo A*, com e sem paralelismo, foi executado na mesma máquina, para que se possa ter resultados justos. Está máquina possui um processador Core i5 760 2.8 GHz, com quatro núcleos, 8 Gb DDR3, e placa de vídeo GeForce 9500 GT.

Para realizar o experimento, foi feita uma escolha aleatória de cinco caminhos, posicionados em diferentes lugares de uma grade, para serem roteados com paralelismo e sem paralelismo. Cada caminho foram roteados, com e sem paralelismo, trinta vezes, depois foi feita a média aritmética entre todos os tempos de execução dos roteamentos.

Na Fig. 2, é apresentado os resultados do experimento. O eixo y representa o tempo de execução em segundos, e o eixo x representa cada roteamento. As colunas em azul no gráfico, representam o algoritmo A* executado sem paralelismo, e as colunas em vermelho representam o algoritmo com paralelismo.

Pode-se notar, na Fig. 2, que com a aplicação do paralelismo, a execução do algoritmo A* foi mais rápida. Vale ressaltar, que para distâncias maiores, como as últimas colunas, que o tempo de execução é muito maior, o ganho em relação a distâncias menores foi maior, onde nessa última coluna, se obteve um ganho de cerca de 18%. Tal ganho é muito importante para rotear circuitos muito grandes, que precisam conectar células de uma extremidade a outra do circuito.

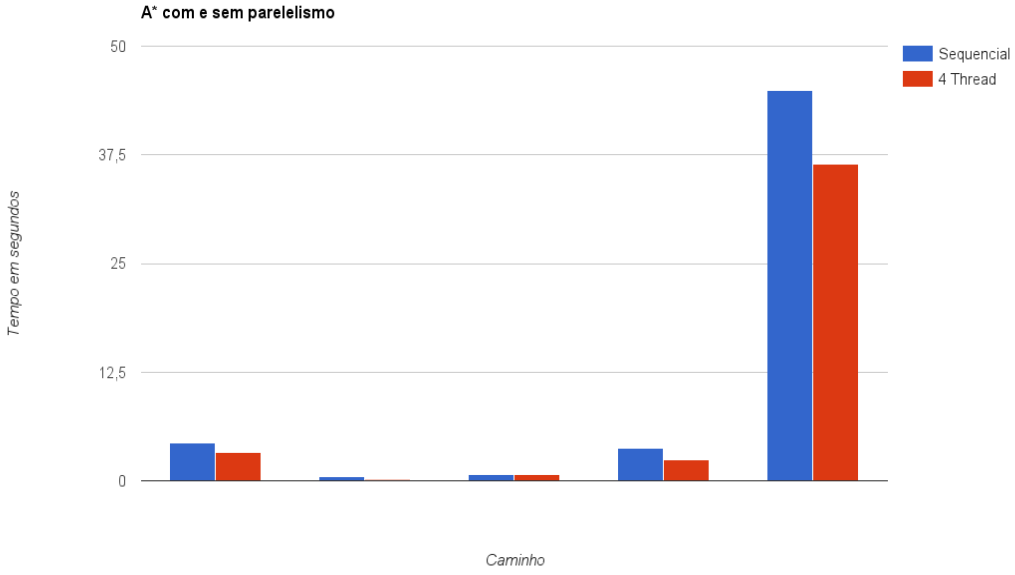

Figura 2: Resultados do algoritmo A*

4. CONCLUSÕES

Tal trabalho apresentou resultados relevantes, que podem ser aplicados em ferramentas de CAD, obtendo, aproximadamente, 18% de redução de tempo. O trabalho aplicou o paralelismo apenas no algoritmo A*, mas uma versão com paralelismo para o algoritmo ST Router está sendo desenvolvida, já que atualmente é o algoritmo estado-da-arte, e teria um grande impacto se seu tempo fosse diminuído.

Como trabalho futuro, pretende-se além de aplicar o paralelismo no algoritmo ST Router, também aplicar em diversos algoritmos da área de circuitos digitais, que demoram muito tempo para executar. Pretendendo assim, diminuir o tempo da concepção de circuito digitais. E além, utilizar tais técnicas em todo o fluxo de execução de uma ferramenta de CAD.

5. REFERÊNCIAS BIBLIOGRÁFICAS

GONÇALVES, S. M. M; DA ROSA, L. S.; DE SOUZA MARQUES, F. . **Algoritmos de Posicionamento e Roteamento para a Visualização de Circuitos Digitais**. 2014. Monografia (Graduação em Ciência da Computação) – Curso de Ciência da Computação, Universidade Federal de Pelotas.

HART, P. E.; NILSSON, N. J.; RAPHAEL, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. **IEEE Transactions on System Science and Cybernectics**, p. 100 - 107, 1968.

LEE, C. Y. . An Algorithm for Path Connections and Its Applications. **IRE Transactions on Eletronics Computers**, p. 346 - 365, 1961.

OPENMP. OpenMP API. Especiais. Acessado em 21 mai. 2015. Online. Disponível em: http://openmp.org