UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Tiago Augusto Fontana

GERAÇÃO DE ÁRVORE DE RELÓGIO PARA AVALIAÇÃO DE POSICIONAMENTO DE CIRCUITOS INTEGRADOS

Florianópolis

2015

Tiago Augusto Fontana

GERAÇÃO DE ÁRVORE DE RELÓGIO PARA AVALIAÇÃO DE POSICIONAMENTO DE CIRCUITOS INTEGRADOS

Trabalho de Conclusão de Curso submetido ao Curso de Bacharelado em Ciências da Computação para a obtenção do Grau de Bacharel em Ciências da Computação.

Orientador: Prof. Dr. José Luís Almada Güntzel
Universidade Federal de Santa Catarina
Coorientador: Me. Vinícius dos Santos Livramento
Universidade Federal de Santa Catarina

Florianópolis

2015

Tiago Augusto Fontana

GERAÇÃO DE ÁRVORE DE RELÓGIO PARA AVALIAÇÃO DE POSICIONAMENTO DE CIRCUITOS INTEGRADOS

Este Trabalho de Conclusão de Curso foi julgado aprovado para a obtenção do Título de "Bacharel em Ciências da Computação", e aprovado em sua forma final pelo Curso de Bacharelado em Ciências da Computação.

Florianópolis, 08 de Dezembro 2015.

Prof. Dr. Mário Antônio Ribeiro Dantas
Coordenador
Universidade Federal de Santa Catarina

Banca Examinadora:

Prof. Dr. José Luís Almada Güntzel
Universidade Federal de Santa Catarina
Orientador

Me. Vinícius dos Santos Livramento
Universidade Federal de Santa Catarina
Coorientador

Prof. Dr. Laércio Lima Pilla
Universidade Federal de Santa Catarina

LISTA DE FIGURAS

LISTA DE SIGLAS E ABREVIATURAS

SUMÁRIO

1 INTRODUÇÃO

Atualmente, arquiteturas de computadores são construídas de forma hierárquica quanto à memória. Essa hierarquia diz respeito à passagem de dados entre os diferentes níveis, ou seja, quais caminhos existem para que dados sejam comunicados entre pontos dessa hierarquia. Níveis de memória mais altos possuem maior capacidade de armazenamento, porém seu tempo de acesso é maior. Parte dessa hierarquia é composta por um ou mais níveis de cache, memórias com capacidade reduzida, mas maior velocidade, permitindo que, em um dado momento, um conjunto de dados qualquer possa ser acessado mais rapidamente. No nível mais baixo dessas hierarquias estão as unidades de processamento, acessando e operando sobre os dados em memória. Quanto mais próximo for o nível de memória em que esse dados estiverem, menor o tempo de acesso. Quando essas unidades precisam se comunicar entre si, elas fazem uso da hierarquia de memória.

A hierarquia pode ser organizada de várias formas, podendo se tornar complexa e de grande profundidade. Uma possibilidade na organização é o compartilhamento de alguns níveis de cache, ou seja, duas unidades de processamento ou mais estarão debaixo de uma mesma cache na hierarquia. Isso permite, por exemplo, realizar comunicações com eficiência, pois o tempo entre algum dado ser atualizado e o novo valor ser visto é determinado pelo tempo de acesso à cache compartilhada. Uma grande variedade de organizações pode ser encontrada ao se considerar arquiteturas como multicore, em que várias unidades de processamento, que recebem o nome de núcleos (cores), estão físicamente agrupadas de algum modo, podendo haver compartilhamento de cache, ou NUMA (Non-Uniform Memory Access), em que mais níveis são acrescentados à hierarquia, os quais compõe a rede de interconexão, a qual, por si só, também pode ser organizada de várias formas distintas. Essa organização compreendendo hierarquia de memória e unidades de processamento, na sua totalidade, define uma topologia de máquina.

A necessidade de plataformas para rodar aplicações de alto desempenho tem dado origem às diversas arquiteturas paralelas modernas existentes. Suas topologias são as mais variadas, visando atender às necessidades de várias classes de aplicações com características e comportamentos distintos. Diante da crescente complexidade das topologias dessas máquinas, a sua organização e as demais características dos elementos que compõe a hierarquia de memória são aspectos de muita relevância para o desempenho de aplicações.

Certas combinações de fatores da aplicação e da arquitetura podem resultar na melhoria ou na degradação do desempenho. Tais fatores podem ser, por exemplo, a quantidade de dados manipulados e o tamanho das caches, que podem comportar ou não todos os dados simultaneamente, ou os padrões de troca de mensagens entre tarefas e a localização delas, além dos meios existentes para realizar essas comunicações, que podem resultar em maior ou menor eficiência.

Ainda, em arquiteturas NUMA, nas quais diferentes regiões da memória possuem diferentes tempos de acesso, é importante que haja proximidade entre os dados acessados por uma thread e o núcleo em que ela reside. Portanto, é essencial o conhecimento da topologia da máquina, que possibilita o devido ajuste das aplicações a ela, de modo a aproveitar ao máximo os recursos disponíveis.

Disso vem a necessidade de haver alguma representação da topologia para fornecer as informações necessárias sobre ela, seja diretamente às aplicações ou a outras partes do sistema, que usarão tais informações para realizar otimizações estática ou dinamicamente. Como exemplo de uso estático, pode-se citar compilação de algoritmos com conhecimento da hierarquia [Sequoia], ou posicionamento de processos MPI [hwloc-2010]; e, quanto ao uso dinâmico, posicionamento de threads e dados OpenMP [FGOMP].

[hwloc-2010] http://www.open-mpi.de/papers/pdp-2010/hwloc-pdp-2010.pdf [FGOMP] https://hal.inria.fr/inria-00496295/document

No entanto, a disponibilização de tais informações gera custos adicionais, além de ter outras implicações relacionadas ao tamanho das estruturas de dados que podem afetar o desempenho. Assim, é necessário que haja um compromisso entre o tempo de acesso e o espaço ocupado pela representação utilizada. Tempos de acesso muito grandes podem acabar anulando os ganhos das otimizações. Já se as estruturas de dados forem muito espaçosas, pode ser que não possuam boa localidade espacial, dependendo dos padrões de referência aos dados em acessos consecutivos. Isso pode resultar em perda de desempenho ocasionada por faltas de cache, tanto no acesso às informações da topologia, que estarão espalhadas em diversas posições da cache, podendo ser substituídas com maior probabilidade, quanto no acesso pelas aplicações aos seus próprios dados. Entretanto, é possível que a adição de algumas informações facilitem certas consultas sobre a topologia sem causar tais prejuízos, que é o desejado.

1.1 MOTIVAÇÃO

Os exemplos de usos estáticos e dinâmicos dados acima, além de vários outros existentes, com o uso de benchmarks, servem como justificativa para a realização de esforços para desenvolver reprentações com as características citadas, isto é, bom tempo de acesso e uso eficiente da memória.

Visto que, como dito anteriormente, os níveis de cache inferiores são mais rápidos, o ideal é que os dados estejam sempre nos níveis os mais próximos possíveis, de modo que seu uso nas computações seja mais eficiente. Diante disso, compilação com conhecimento da hierarquia [citar de novo?] se vale do fato de que frequentemente problemas podem ser divididos em problemas menores de tamanho variável, e ajustar esses tamanhos à capacidade das caches torna o uso delas mais efetivo, pois todos os dados usados nessas partes menores da computação caberão nelas. Ainda, quando é possível haver vários níveis de subdivisão do problema, formando também uma espécie de hierarquia de subdivisões, os tamanhos das partes em diferentes níveis podem ser ajustados aos níveis de cache consecutivos. Isso pode ser visualizado com facilidade no exemplo de multiplicação de grandes matrizes presente no artigo referenciado [ou aqui].

A velocidade de níveis de cache mais próximos também beneficia a comunicação. Portanto, em conjunto com dados sobre os padrões de comunicação entre processos, as informações sobre compartilhamento de caches podem ser usadas para definir um posicionamento de processos MPI que favoreça as comunicações [hwloc-2010]. Outra otimização possível é o uso de metódos específicos do hardware para realizar comunicações dentro de um nodo.

No contexto de arquiteturas NUMA, para diminuir o tempo de acesso a memórias remotas, há a possibilidade de mover os dados para outro nodo ou as threads para outros núcleos. Seguindo o princípio de realizar um combinação dessas opções com base nos níveis da topologia, o posicionamento dinâmico de threads e dados desenvolvido no ForestGOMP [hwloc-2014], uma extensão de uma implementação de OpenMP, se mostrou efetivo. Um cenário apresentado é a existência de vários conjuntos de threads e dados com grande afinidade, em que a migração de uma thread para outro núcleo só ocorreria se houvesse um nível de cache compartilhado, de modo a manter a thread próxima dos seus dados, enquanto em outros casos poderia haver a migração de todas as threads e dados relacionados.

Esses exemplos ilustram como informações sobre a hierarquia po-

dem efetivamente ser usadas para melhorar o desempenho de aplicações que seguem modelos ou estratégias em uso real, ou seja, os benchmarks utilizados possuem características encontradas na solução de problemas reais. Isso diz respeito a, por exemplo, padrões de comunicação ou distribuição de carga, que podem apresentar irregularidades e outras características presentes em aplicações científicas de diversas áreas.

1.2 OBJETIVOS

Este trabalho tem como objetivo o desenvolvimento de uma representação de topologias de máquina, compreendendo as estruturas de dados utilizadas e os métodos de acesso, que mantenha o compromisso necessário, conforme apresentado anteriormente, entre tempo de acesso e espaço ocupado na memória pelas estruturas de dados.

1.2.1 Objetivos específicos

Os objetivos específicos são:

1. Analisar fatores relevantes para a eficiência na representação de topologias
2. Desenvolver representações (estruturas de dados e métodos de acesso)
3. Testar as representações desenvolvidas, por meio de experimentos em diferentes máquinas, observando o uso da memória e o tempo de execução
4. Disponibilizar uma nova ferramenta para a representação de topologias de máquina

2 CONCEITOS FUNDAMENTAIS

O objetivo deste capítulo é explanar alguns conceitos teóricos necessários para a compreensão da presente monografia.

2.1 RESTRIÇÕES DE *SETUP* E *HOLD*

Restrição de *Setup*: especifica a quantidade de tempo que um sinal de entrada de dados deve ser estável antes da transição de relógio para cada elemento de armazenamento (por exemplo, *flip-flop* ou *latch*).

$$\mathrm{D}_{i,j} + \mathrm{t}_{dCQ} \leq \mathrm{T}_{cyc} - \mathrm{t}_{skew_{i,j}} - 2\mathrm{t}_{jit} - \mathrm{t}_s \tag{2.1}$$

Restrição de *Hold*: especifica a quantidade de tempo que um sinal de entrada de dados deve ser estável após a borda relógio em cada elemento de armazenamento.

$$\mathrm{d}_{i,j} + \mathrm{t}_{cCQ} \geq \mathrm{t}_h + \mathrm{t}_{skew_{i,j}} - 2\mathrm{t}_{jit} - \mathrm{t}_s \tag{2.2}$$

A equação 2.1 delimita o período de relógio. Nestas equações, $d_{i,j}(D_{i,j})$ são o mínimo (máximo) *delay* entre os elementos sequenciais *i,j*. T_{cyc} é o período de relógio, $t_s(t_h)$ são os tempos de *setup (hold)*, t_{dCQ} é o atraso de saída do relógio (*delay* contaminado) dos elementos sequenciais, $t_{skew_{i,j}}$ é o *skew* e t_{jit} é o *jitter*. O *skew* local é $t_{skew_{i,j}} = t_i - t_j$ dos elementos sequenciais i e j onde t_i e t_j são os delays do sinal de relógio para os pinos de relógio dos elementos sequenciais, que também são chamados de *clock sink*. O *jitter* é a variação máxima da borda do relógio ao tempo de chegada do relógio a qualquer *sink*.

2.2 CLOCK SKEW

Clock Skew é a diferença máxima de chegada entre dois *sinks*. Este parâmetro do relógio é de severa importância, uma vez que o sinal de relógio deve chegar a todos os consumidores num mesmo instante de tempo. Se $t(u, v)$ representa o *delay* entre dois sinais u e v, o *skew* da árvore de relógio T é denotado pela equação 2.3 (KAHNG et al., 2011).

$$skew(\mathrm{T}) = \max_{S_i, S_j \in S} |t(S_0, S_i) - t(S_0, S_j)| \tag{2.3}$$

3 TÉCNICAS CLÁSSICAS PARA O ROTEAMENTO DO SINAL DE RELÓGIO

3.1 ALGORITMO METHOD OF MEANS AND MEDIANS

Este algoritmo foi proposto inicialmente por Jackson, Srinivasan e Kuh (1990). Sua ideia geral consiste em:

(a) Encontra o centro de massa segundo a equação:

$$\mathrm{x}_c(S) = \frac{\sum_{i=1}^{n} x_i}{n} \qquad \text{e} \qquad \mathrm{y}_c(S) = \frac{\sum_{i=1}^{n} y_i}{n} \tag{3.1}$$

(b) Divide o intervalo na mediana

(c) Encontra o centro de massa de cada sub-intervalo.

(d) Efetua o roteamento entre os centros de massa encontrados.

(e) Efetua a chamada recursiva do algoritmo, até que o número de *sinks* seje igual a um.

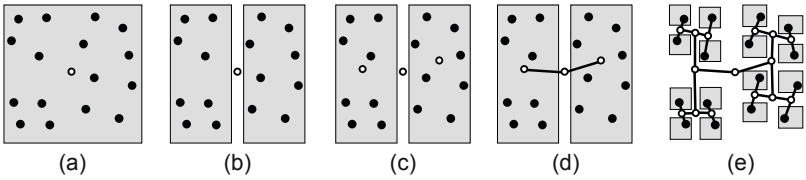

Figura 1: Passos da execução do algoritmo *Method of Means and Medians*.
Fonte: (KAHNG et al., 2011)
Esta imagem será substituida por uma nova, criada por min.

3.2 ALGORITMO RECURSIVE GEOMETRIC MATCHING

3.3 ALGORITMO DEFERRED-MERGE EMBEDDING

Algoritmo 1: Basic_Method_of_Means_and_Medians

Input : $Conjuntodesinksrestantes S, Centrodemassa C_1, orient$
Output: Roteamento da árvore local de relógio

1 $C_2 = localiza\_centro\_de\_massa(S)$;
2 $roteamento(C_1, C_2)$;
3 **if then**
4 | $return$;
5 **end**
6 $ordenamento(S, orient)$;
7 $basic\_MMM(S_{begin,mid}, C_2, !orient)$;
8 $basic\_MMM(S_{mid+1,end}, C_2, !orient)$;

4 ANDAMENTO DO TRABALHO

Este é um relatório de andamento da monografia. Até o presente momento da entrega deste, ja foi implementado o primeiro algoritmo clássico (*Method of Means and Medians* (MMM)), e o segundo (*Recursive Geometric Matching* (RGM)) encontra-se em desenvolvimento. Já foi realizada uma breve revisão bibliográfica, bem como uma breve integração na infraestrutura da International Conference On Computer Aided Design (ICCAD) (KIM et al., 2015).

Pretende-se terminar a implementação dos dois algoritmos clássicos restantes (*Recursive Geometric Matching* (RGM) e *Deferred-Merge Embedding* (DME)) e executar as avaliações experimentais até Março de 2016, reservando assim o tempo do semestre letivo para a escrita restante desta monografia. Pretende-se também realizar a defesa deste trabalho no primeiro semestre de 2015.

JACKSON, M.; SRINIVASAN, A.; KUH, E. Clock routing for high-performance ics. In: **Design Automation Conference, 1990. Proceedings., 27th ACM/IEEE**. [S.l.: s.n.], 1990. p. 573–579. ISSN 0738-100X.

KAHNG, A. B. et al. **VLSI Physical Design: From Graph Partitioning to Timing Closure**. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2011. ISBN 9789048195909.

KIM, M. et al. Iccad-2015 cad contest in incremental timing-driven placement and benchmark suite. In: **ICCAD**. [S.l.: s.n.], 2015. p. 921–926.