A Utilização de Modelagem Analítica no Projeto de Arquiteturas de Processadores[1]

Rafael L. Sagula[2]
Ronaldo A. L. Gonçalves[3]
Tiarajú A. Diverio[4]
Philippe O. A. Navaux [4]
Instituto de Informática e PPGC da UFRGS
Cx. P. 15064 - 91501-970 Porto Alegre, RS, Brasil
{sagula, ronaldog, diverio, navaux}@inf.ufrgs.br

Abstract

This work presents the utilization of analytical modeling in the design of processor architectures. To validate the facilities of that methodology, a case study was done and the results were analyzed. Analytical modeling was applied on the design of an instruction fetch mechanism for a simultaneous multithreaded (SMT) architecture and showed that some interesting results can be achieved with minimum design effort. The analytical modeling allowed, in a very fast way, the analysis of such architecture using many architectural parameters such as number of threads, fetch width, i-cache miss rate, and instruction latency. The obtained results show that the utilization of analytical modeling, mainly during the initial phases of design, can provide anticipated estimations about both behavior and performance of the proposed architecture and reduce the design time.

Keywords: Analytical Modeling, Petri Net, SMT.

1. Introdução.

De mais a mais, as aplicações tornam-se mais complexas e maiores, necessitando de mais poder de processamento. Nesse sentido, o projeto da arquitetura de processadores de alto desempenho sempre tem sido um tópico de intensa pesquisa, cujo objetivo principal é desenvolver arquiteturas que executem as tarefas para as quais foram construídas de forma mais rápida e segura possível. Esta relação entre a computação realizada e o tempo gasto pode ser chamada de desempenho.

Historicamente, além do avanço tecnológico na compactação de dispositivos e do aumento da freqüência de relógio dos mesmos, várias foram as técnicas desenvolvidas para melhorar o desempenho das arquiteturas, quase sempre através da reorganização dos seus componentes internos, permitindo, assim, que um maior número de instruções sejam executadas em menor espaço de tempo. Dentre essas técnicas estão ([JOH91], [HEN94], [SMI95], PAT90]): *pipelining*, arquiteturas superescalares, VLIW, *super- pipelining*, *multipath execution* e *simultaneous multithreading* (SMT) [TUL95].

Antes da implementação de novas arquiteturas, é fundamental a utilização de técnicas capazes de estimar o desempenho das mesmas, principalmente para evitar o desenvolvimento de arquiteturas que não satisfaçam as condições mínimas esperadas. Como a construção de um protótipo físico de um processador é uma tarefa muito cara, existem duas outras metodologias bastante comuns e menos onerosas [JAI91]: simulação e modelagem analítica.

[1] Trabalho realizado com apoio do CNPq e CAPES
[2] Aluno de Mestrado em Ciência da Computação junto ao PPGC/UFRGS
[3] Aluno de Doutorado em Ciência da Computação junto ao PPGC/UFRGS
[4] Professores Orientadores junto ao PPGC/UFRGS

Normalmente, tanto a simulação quanto a modelagem analítica tentam representar e executar as funções características de funcionamento do processador, permitindo estimar o desempenho da arquitetura proposta, sob diferentes configurações. Infelizmente, os parâmetros considerados em ambas as abordagens são incapazes de expressar com fidelidade uma situação que só pode ser obtida em um ambiente de computação real, mas ainda assim, estas são técnicas necessárias quando se deseja implementar soluções com melhores chances de serem bem sucedidas.

As simulações apresentam três principais problemas que justificam a utilização de modelagem analítica como opção mais atrativa: o desenvolvimento de simuladores pode levar muito tempo de programação e, conforme a complexidade da arquitetura, o mesmo poderá ser escrito e rescrito centenas de vezes; as simulações são extremamente lentas, as máquinas utilizadas ficam estressadas e os resultados podem levar dias para serem obtidos; e os simuladores, em geral, não possuem flexibilidade suficiente para permitir mudanças estruturais rápidas na arquitetura.

Neste trabalho, são descritas os principais métodos formais de modelagem analítica e, para comprovar a facilidade e rapidez de sua utilização, é apresentado um estudo de caso no projeto de uma arquitetura SMT onde os resultados são analisados.

2. Modelagem Analítica.

Várias pesquisas têm sido feitas na área de modelagem analítica e muitas arquiteturas tem sido modeladas ([NEM91],[MAR84],[SAA90],[JAC96],[KAN97]). Segundo Lindemann ([LIN98]), se a questão de desempenho for considerada nas fases iniciais do projeto do sistema, então se deve usar modelagem analítica para isso, pois o sistema não está ainda operacional a ponto de fornecer dados sobre seu desempenho. Para Sahmer, Trivedi e Puliafito ([SAH96]) a modelagem analítica pode ser executada juntamente com simulação. Neil J. Gunther [GUN98] afirma que o ciclo de desenvolvimento de um produto deve ser o mais rápido possível, em função das metodologias modernas de gerenciamento e da competitividade do mercado, e propõe uma metodologia baseada em modelos analíticos, demostrando seu uso através de vários estudos de caso.

Por fim é importante salientar que os três trabalhos concordam num ponto: a exclusão da modelagem e da avaliação de desempenho dos projetos atuais abre espaço para técnicas empíricas calcadas na experiência do analista, surgindo assim a necessidade da utilização de modelos ou técnicas que não atrapalhem o projeto, mas consigam fornecer dados importantes à realização do mesmo. Modelos analíticos são construídos utilizando teoria de filas, cadeias de Markov e redes de Petri, explicados na próxima seção.

2.1. Métodos Formais para Modelagem Analítica.

Geralmente, os métodos formais fornecem uma solução fechada para o problema de desempenho da arquitetura, podendo apresentar-se na forma de um sistema de equações lineares, por exemplo, onde os resultados podem ser obtidos de maneira muito mais rápida que na simulação. Embora permitam que alterações arquiteturais sejam incorporadas rapidamente no modelo, a exatidão desses resultados está vinculada diretamente à questão do detalhamento do mesmo, que, sendo grande, implicará no aumento da complexidade.

Nesse sentido, o modelo analítico pode produzir resultados não muito precisos, mas que são justificados pela rapidez na sua execução. Além disso, o emprego de modelos nas fases iniciais do

projeto de arquiteturas é ainda mais justificado uma vez que a quantidade de modificações na arquitetura é bastante grande e não há muita preocupação com a exatidão dos resultados de desempenho.

A modelagem de sistemas pode ser subdividida em três áreas: "dependabilidade" (confiabilidade ou disponibilidade do sistema para o usuário); desempenho do sistema; ou a combinação dos dois primeiros. Os métodos formais quase sempre são baseados nos estados que um sistema pode assumir e na ocorrência de eventos que modifiquem seu estado. Nos próximos parágrafos são descritos os principais métodos formais.

- Modelos markovianos [SIL92]: destinam-se a análise de "dependabilidade" e desempenho de sistemas. Sua característica principal é que o comportamento futuro de um processo independe de seu estado atual ou de seu histórico passado. Um modelo de Markov com tempo discreto é também chamado de cadeia de Markov e é formada por vértices, representando os estados do sistema, e transições entre esses estados. Cada transição possui um peso ou probabilidade de ocorrência e são denominados de estocásticos, ou probabilísticos. A Figura 1 mostra um exemplo de cadeia de Markov com três estados.

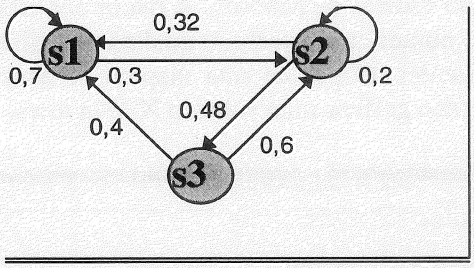

Figura 1 - Exemplo de cadeia de Markov.

O uso de cadeias de Markov para modelagem de sistemas onde o número de estados é muito grande pode tornar-se impraticável sem a ajuda uma ferramenta automática, que construa a cadeia a partir de um modelo mais abstrato. Em [NEM91], é apresentada uma arquitetura *multistream* modelada com cadeias de Markov, onde os resultados obtidos com o modelo foram bastante próximo dos resultados obtidos em uma simulação posterior, com erro da ordem de apenas 5%.

- Rede de Petri [PET62]: composta por elementos chamados de: lugares, transições e arcos. Os arcos de entrada conectam lugares a transições, enquanto os arcos de saída ligam uma transição a uma saída. Existem ainda outros tipos de arcos, tais como os arcos inibidores e os arcos sensores [GUS95]. Os lugares podem conter marcações (*tokens*), configurando o estado do sistema pelos número e tipo de marcações em cada lugar. As transições são elementos ativos no modelo e podem ser disparadas, mudando o estado do sistema.

Uma extensão deste método, chamada Rede de Petri Estocástica [MAR84] incorpora o fator "atraso de ativação" junto as transições, para especificar o tempo em que a transição deve ficar habilitada até que ela realmente ocorra. Três diferentes tipos de atraso podem ser utilizados: transições imediatas, que não possuem atraso; transições exponenciais, que possuem um atraso segundo uma distribuição exponencial; e transições determinísticas, que possuem um atraso fixo.

• Teoria de filas [ROB94]: é um dos formalismos mais usados para modelar sistemas computacionais, pois a maioria dos elementos encontrados nesses sistema são baseados no modelo de filas e servidores, tais como: fila de processos a espera de CPU; fila de requisições de acesso à disco e fila de mensagens em um nó de rede.

Redes de filas são um subconjunto da teoria de filas, que é especializada para avaliação de desempenho ([LIN98]). Estas redes são muito intuitivas e possuem métodos eficientes e conhecidos de solução, mas não podem representar eficientemente sistemas onde há concorrência dentro de uma tarefa, sincronização e posse simultânea de recursos ([SAH96] e [LIN98]).

Algumas ferramentas auxiliam o usuário a modelar seu sistema, com facilidades de edição gráfica, execução e análise sobre os resultados obtidos. Neste contexto, podem ser citadas as ferramentas UltraSAN ([COU91],[DEA95],[SAN95]), SHARPE ([SAH96]), TamgramII ([CAR97]), PDQ ([GUN98]), XSimNet ([GUS95], [GUS95a]) e DSPNExpress ([LIN98]), que foi utilizada nesse trabalho.

O DSPNExpress foi criado em 1992 pelo Prof. Dr. Christoph Lindemann ([LIN98]). Segundo o autor, o desenvolvimento dessa ferramenta foi motivado pela falta de pacotes disponíveis para uma análise numérica eficiente de DSPNs, ou seja, trabalha com modelagem através de redes de Petri e utiliza como método de resolução cadeias de Markov. A maior contribuição científica do DSPNExpress recai sobre o módulo de solução numérica eficiente, o qual resolve sistemas maiores e em menos tempo do que os demais pacotes existentes. O pacote está disponível para SunOS4.1.4 ou Solaris2.5, HP-UX10.10 e IRIX6.2, e sua interface gráfica roda sobre o X Windows. A Figura 2 mostra a interface da ferramenta DSPNExpress.

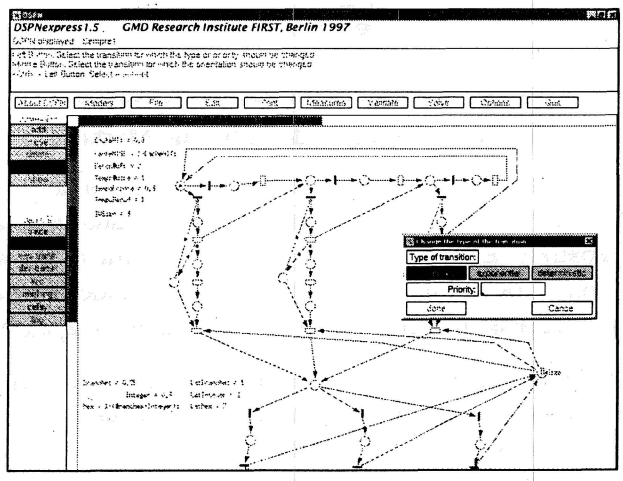

Figura 2 –Interface do DSPNExpress

3. Estudo de Caso: uma Arquitetura SMT.

As arquiteturas SMT (*Simultaneous Multithreaded*) têm sido alvo de pesquisas recentes com o objetivo de suportar computação paralela e de alto desempenho ([TUL95], [AKK98], [GOO98]). Estas arquiteturas exploram paralelismo através do escalonamento de instruções provenientes de diferentes *threads*, e possuem a habilidade de esconder as latências das instruções. Recentemente, uma nova

abordagem para arquiteturas SMT foi proposta em ([GON98], [GON98b]), denominada *SuperEscalar com Múltiplos PRocessos em Execução* - SEMPRE, que visa a execução de processos ao invés de *threads*, além de incorporar ao nível arquitetural algumas funções normalmente executadas pelo sistema operacional.

Como toda arquitetura SMT, um dos principais desafios deste projeto é o desenvolvimento da unidade de busca, que deve ser capaz de buscar, de forma eficiente, instruções provenientes de diferentes fluxos para manter o *pipeline* o mais ocupado possível.

Com o objetivo de antecipar estimativas de desempenho, sob diferentes configurações arquiteturais, e prover importantes informações que auxiliem no processo de tomada de decisão durante o desenvolvimento do simulador, o presente artigo apresenta uma modelagem analítica para a unidade de busca desta arquitetura, analisando seu funcionamento e mostrando sua importância. Na próxima seção, uma breve descrição da arquitetura SEMPRE é apresentada.

3.1. A Arquitetura SEMPRE

A arquitetura SEMPRE (SuperEscalar com Múltiplos PRocessos em Execução) é vista na figura 3, que mostra seus principais componentes funcionais e estruturais. Seu *pipeline* superescalar contém 5 estágios funcionais (busca, decodificação, execução, término e conclusão), providos de técnicas usuais de previsão de desvios, busca especulativa, renomeação simplificada de registradores e execução fora-de-ordem de instruções prontas.

A arquitetura contém vários *slots* para armazenar as instruções buscadas de diferentes processos. Estes processos são escalonados da fila de processos prontos (FP), que contém os descritores de todos os processos criados no sistema. Associado a cada *slot* existem um registrador RDP, que contém o contador de programa e o *time-slice* do processo que está sendo buscado e uma fila de instruções FI. Também existe um banco de registradores (*frame*) para armazenar o contexto de cada processo da FP. As unidades funcionais são compartilhadas pelas instruções que são despachadas simultaneamente dos *slots*.

Durante a busca de instruções na *cache*, é buscado um bloco de instruções por vez, para cada *slot* da arquitetura, no estilo *round-robin*. Uma vez selecionado o *slot*, a busca é feita a partir do endereço contido no contador de programa do RDP e as instruções buscadas são inseridas na respectiva FI. Sempre que for necessária uma troca de contexto em um determinado *slot*, o estágio de busca escalona um processo da FP e substitui o RDP do *slot* com o contador de programa e o *time-slice* do processo escalonado, permitindo que durante a próxima busca as instruções do novo processo sejam inseridas na respectiva FI.

O descritor do processo anterior, já atualizado, ainda permanece na arquitetura, passando pelas fila de ativos (FA) e fila de processos em trânsito (FT), até que sua última instrução seja concluída, quando então ele volta para a FP. De uma forma geral, a troca de contexto pode ocorrer durante a ocorrência de i-*cache miss*, durante a ocorrência de instruções privilegiadas e com o término do *time-slice* do processo, que deve ser decrementado automaticamente pelo *hardware*. As instruções privilegiadas, que podem ser executadas pela unidade de busca, para auxiliar no gerenciamento de processos são: *create*, *kill*, *suspend* e *resume*.

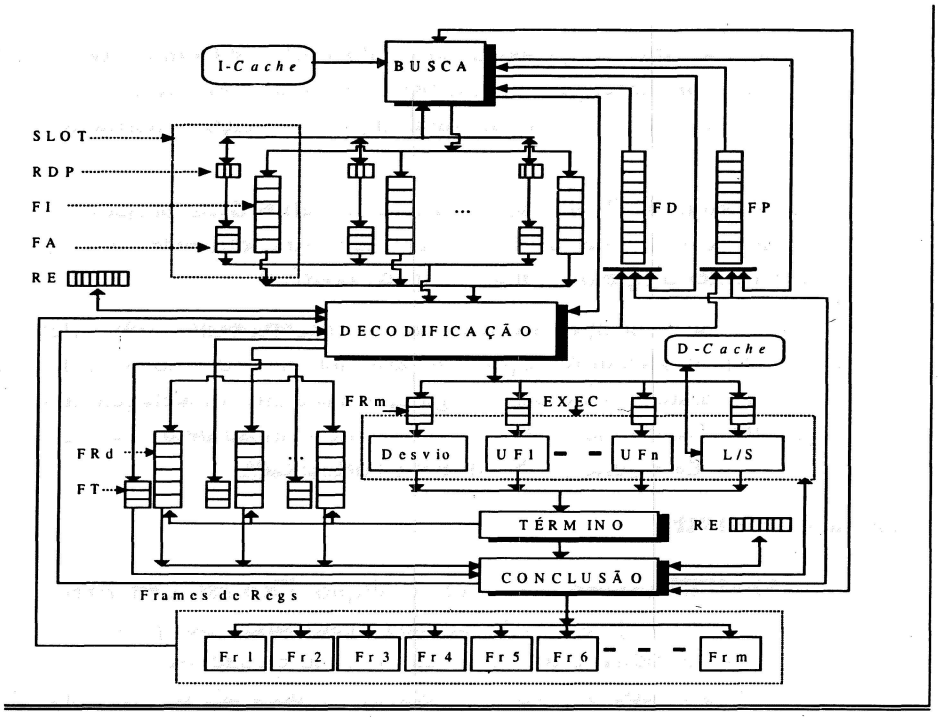

Figura 3 - Arquitetura *SuperEscalar com Múltiplos PRocessos em Execução*

O estágio de decodificação executa três principais atividades em conjunto: escalonamento das instruções dos *slots*, decodificação (incluindo renomeação simplificada) e despacho para as filas de remessa (FRm) das unidades funcionais. As instruções contidas nas primeiras entradas das FIs dos *slots* são decodificadas e somente aquelas que estão prontas são despachadas, já que execução fora-de-ordem em arquiteturas SMT não permite substancial melhoria no desempenho [HIL98] e evita a poluição das filas de remessa com instruções não-prontas.

Durante o estágio de execução, cada unidade funcional executa as instruções de sua respectiva FRm, retirando-as em ordem através de deslocamento. O resultado da instrução é enviado para o estágio de término, que atualiza as entradas das filas de reordenação (FRd). Para cada FRd, o estágio de conclusão verifica em-ordem quais são as instruções "terminadas" e para cada uma, o resultado é gravado no *frame* de registradores correspondente. Quando termina a execução de um contexto o processo é retirado da fila FT e é reinserido na fila FP, caso a execução tenha sido normal. Ainda no estágio de conclusão são controladas a execução especulativa, ocorrência de exceções e mortes de processos. Maiores detalhes sobre esta arquitetura são encontrados em ([GON98], [GON98b]).

3.2. Aplicação das Técnicas de Modelagem.

A unidade de busca da arquitetura SEMPRE foi modelada utilizando a ferramenta DSPNExpress, que permitiu o desenvolvimento de um modelo analítico estruturado como uma rede de Petri. Primeiramente, foi desenvolvido um modelo estático para 4 *slots* conforme mostra a Figura 4, que posteriormente foi transformado no modelo otimizado da Figura 5. Este segundo modelo permite,

em tempo de resolução, definir a configuração dos seguintes parâmetros arquiteturais: taxa de *i-cache hit* na *cache* nível 1, latência da *cache* nível 1, largura de busca, latência de execução das instruções e, principalmente, número de *slots* (número de *threads* que são buscados).

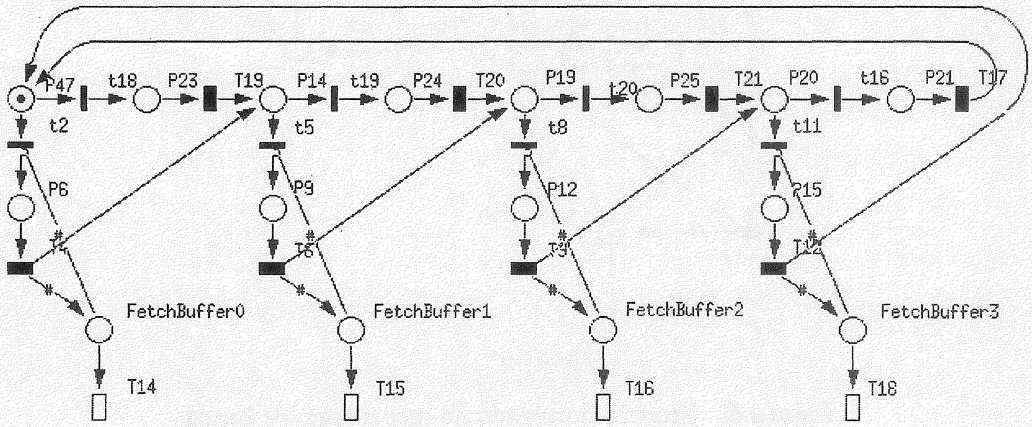

Figura 4 – Modelo para 4 *slots* do mecanismo de busca

Na modelagem aqui apresentada foram feitas as seguintes considerações:

- O estágio de execução foi simplificado, onde foi considerado haver um número suficiente de unidades funcionais para executar qualquer freqüência de *ILP* simultaneamente de cada *slot* contendo instruções prontas. Com isso, visando manter a ocupação máxima das unidades funcionais, o desempenho da arquitetura pode ser medido pelo número médio de instruções disponíveis em todas as filas de instruções. Este índice de desempenho é aqui chamado de TOFI (Taxa de Ocupação das Filas de Instruções).
- Também se considerou que a unidade de busca possui um mecanismo de pre-fetch eficiente que associado ao escalonamento de processos antecipa a busca das instruções antes dos processos serem escalonados. Isto é feito porque a arquitetura utiliza a fila de prontos FP para fornecer as informações necessárias ao pre-fetch. Assim, um i-cache miss nunca ocorre durante a busca de instruções mas durante a previsão de um desvio para um caminho tomado que não se encontra na i-cache. Neste caso, o contexto é trocado e quando o mesmo processo for novamente escalonado suas instruções, para o caminho tomado, já estarão na i-cache devido a antecipação pelo pre-fetch.
- A latência da cache de instruções foi considerada de 1 ciclo para cache hit, mas como um i-cache miss tem sua latência mascarada pela troca de contexto, tanto na ocorrência de i-cache hit quanto de i-cache miss o próximo slot será buscado no tempo máximo de 1 ciclo.

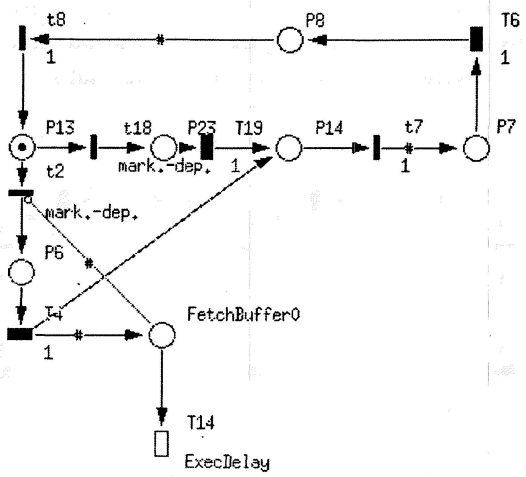

Figura 5 – Modelo otimizado do mecanismo de busca

Na primeira configuração da modelagem, foi variado o número de *slots* (de 2 até 16) para diferentes larguras de busca (4, 6, 8 e 10), onde o TOFI obtido pode ser visto na Tabela 1. No gráfico da figura 6, pode-se observar que os valores convergem para um limite máximo quando o número de *slots* é aumentado. Esse limite pode ser calculado através da expressão $(L+1)(L/2)(H)$, onde L é a largura de busca e H a taxa de *i-cache hit*. Para este experimento, considerou-se uma taxa de *i-cache hit* de 80% com latência de instruções nas FIs de 1 ciclo.

Tabela 1: TOFI obtido variando o número de *slots* e a largura de busca

Número de *slots*	Largura de Busca			
	4	**6**	**8**	**10**
2	4,871567	8,60887	12,52798	16,50313
3	6,273544	11,50576	17,18173	23,04421
4	7,090579	13,48623	20,68061	28,26982
5	7,536037	14,78429	23,24309	32,36248
6	7,769324	15,60911	25,08772	35,53814
7	7,887886	16,1169	26,38691	37,97561
8	7,946594	16,41913	27,27599	39,81361
9	7,97501	16,59328	27,86485	41,1658
10	7,988463	16,69041	28,24186	42,13206
11	7,994814	16,74332	28,47547	42,80083
12	7,997645	16,77113	28,6158	43,24952
13	7,998981	16,78571	28,69768	43,54153
14	7,999565	16,79299	28,74442	43,72587
15	7,999783	16,79661	28,77021	43,83975
16	7,999927	16,79829	28,78446	43,9081

Observa-se, nesses resultados, que o aumento no número de *slots* não causa modificações no TOFI a partir de um certo valor, pois a largura de busca passa a ser o limitante nesse caso. Além disso, algumas combinações diferentes podem gerar resultados aproximados, como é o caso onde a largura de

busca é 6 e o número de *slots* é maior que 8. Os valores obtidos nesse caso são próximos daqueles com largura de busca 8 e o número de *slots* 4, ou então com largura de busca 10 e número de *slots* 2.

Nesse caso, outros fatores limitantes de desempenho deverão ser levados em consideração para que a melhor alternativa seja escolhida. Mas, também, é possível que a decisão seja baseada em fatores não relacionados à arquitetura, mas sim em restrições de projeto, tais como, por exemplo, custo final da arquitetura ou disponibilidade de componentes no mercado. Sabe-se que, quanto maior a largura de busca utilizada, mais caros são os componentes relacionados (*cache* e barramento), entretanto, o aumento no número de *slots* exige que uma área maior no *chip* seja utilizada somente para os *buffers* e mecanismos de controle.

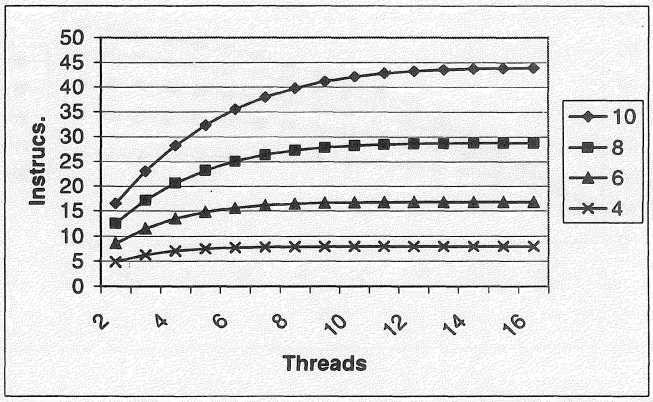

Figura 6: TOFI x Número de *Slots* x Largura de Busca

A figura 7 mostra a segunda configuração da modelagem, onde foram variadas a taxa de *i-cache hit* e o número de *slots*. Para tanto, manteve-se a largura de busca em 8 instruções e a latência das instruções nas FIs em 1 ciclo. Observa-se que o TOFI é diretamente proporcional a taxa de *i-cache hit*. Isso ocorre principalmente porque outros fatores que influenciariam no desempenho da *cache* não foram modelados, tal como o número de *slots*, que diminuiria a taxa de *i-cache hit*. Entretanto, a interpretação do gráfico indica que a ocorrência de poucos acertos na *cache* sugere que mais *slots* sejam utilizados para manter o *pipeline* com maior número de instruções.

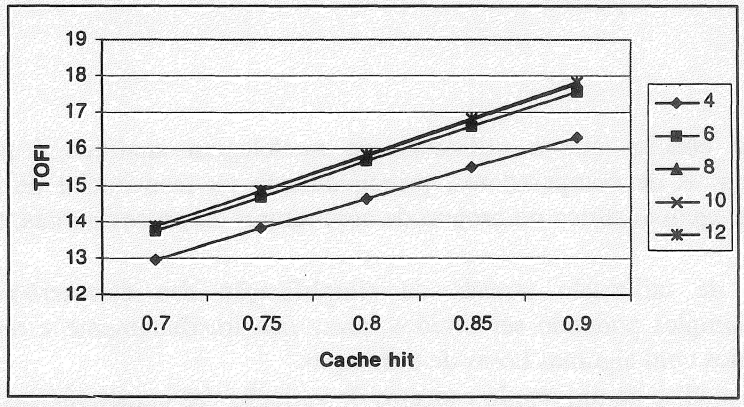

Figura 7: TOFI x Número de *slots* x Taxa de *i-cache hit*

Na Figura 8, foi variado a latência das instruções nas FIs de 0,4 ciclo até 4 ciclos em média por instrução, o que significa que o grau de *ILP* varia de 0,26 IPC (no caso de 3.8) até 2,5 IPC (no caso de 0,4). Manteve-se a largura de busca em 8 instruções e o número de *slots* foi também variado. Observa-se que, para aplicações com alto grau de *ILP*, a variação no número de *slots* não influencia muito no TOFI. Da mesma forma, essa variação cresce a medida que o paralelismo diminui.

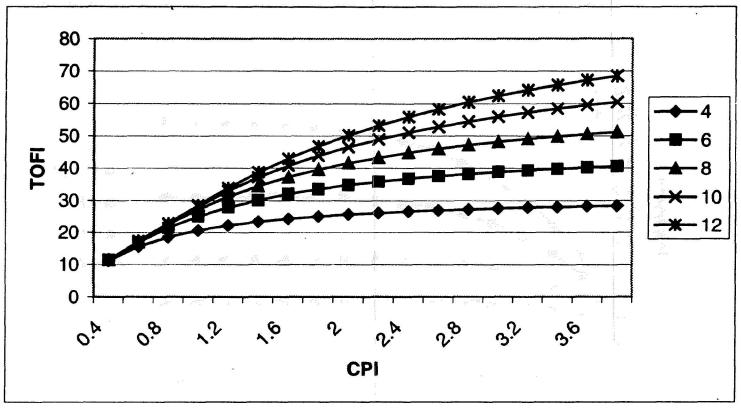

Figura 8: TOFI x Latência de Instruções x Número de *slots*

Em uma primeira análise, pode-se concluir que, com uma latência de 0,5 ciclos para as instruções nas FIs, o aumento na quantidade de *slots* não proporciona nenhuma ganho com relação ao TOFI, justificando assim a utilização de um número menor de *slots*. Mas com uma análise mais profunda do processo de modelagem pode-se concluir que este fato não necessariamente implica em maior desempenho no nível de *ILP*, pois, apesar do TOFI ser constante em ambas as configurações, provavelmente a configuração com maior número de *slots* poderá estar executando um maior número de instruções no nível global da arquitetura.

4. Considerações Finais

O presente trabalho conclui que a utilização de modelagem analítica é importante no desenvolvimento de arquiteturas de computadores, principalmente na fase inicial de projeto. Com relação a utilização de modelagem analítica no projeto de arquiteturas de processadores, três principais conclusões são obtidas:

- Simplicidade de utilização através da simplificação das características a serem simuladas, com resultados simples podendo ser obtidos quase que imediatamente e resultados mais elaborados podendo ser obtidos com algumas horas de utilização.
- Facilidade de análise de um amplo espectro de variação das características arquiteturais, permitindo obter informações a respeito de mínimos e máximos desempenhos. Estas informações se consideradas em conjunto com a disponibilidade e custo do *hardware*, podem favorecer uma especificação mais vantajosa do projeto em termos de custo/benefício.

• Facilidade para a tomada de decisão dinâmica durante o desenvolvimento do projeto, provendo informações para que o simulador seja desenvolvido de forma definitiva, evitando manutenções futura no código.

Em continuidade ao trabalho aqui apresentado, a próxima etapa é o desenvolvimento de uma modelagem para a unidade de execução da arquitetura SEMPRE, para estimar a taxa de IPC. Outro fator que também pode ser considerado nas próximas modelagens é a variação da taxa de *i-cache miss* em função do aumento do número de *threads* buscados, o que pode influenciar o desempenho do sistema. Novos parâmetros também deverão ser considerados, tais como tipos de instruções e tipos e quantidades de unidades funcionais e, além disso, a variação de diferentes parâmetros simultaneamente pode trazer resultados mais significativos.

Por fim, observa-se que a utilização de modelagem analítica no projeto de arquiteturas de processadores é uma metodologia promissora, pois permite a verificação de hipóteses da especificação do projeto, a um baixo custo e em um tempo razoável.

5. Referências Bibliográficas.

[AKK98] Akkary, H., Driscoll, M. A., A Dynamic Multithreading Processor, Proceedings of the MICRO-31: ACM/IEEE International Symposium on Microarchitecture, Dallas, Texas, December, 1998.

[CAR97] Carvalho, L.: Uma Ferramenta para Modelagem de Sistemas de Comunicação, Computação e Confiabilidade. Trabalho de Mestrado. COPPE/UFRJ, 1997.

[COU91] Couvillion, J.; Freire, R.; Johnson, R.; Obal II, W.D.; Qureshi, M.A.; Rai, M.; Sanders, W.H.; Tvedt, J.E.: Performability Modeling with UltraSAN. IEEE Software, vol. 8, no. 5, Sept. 1991, pp. 69-80.

[DEA95] Deavours, D.D.; Obal II, W.D.; Qureshi, M.A.; Sanders, W.H.; Van Moorsel, A.P.A.: UltraSAN Version 3 Overview. Proceedings of the Sixth International Workshop on Petri Nets and Performance Models, Durham, NC, October 3-6, 1995, pp. 216-217.

[GON98] Gonçalves, R. A. L., Navaux, P. O. A., SEMPRE: Uma Arquitetura SuperEscalar com Múltiplos PRocessos em Execução, X SBAC-PAD, Búzios, Brazil, Setembro, 1998.

[GON98b] Gonçalves, R. A. L., Navaux, P. O. A., Proposta de uma Arquitetura *Multi-Threading* Voltada para Sistemas Multi-Processos, IV Congresso Argentino de Ciência da Computação - CACIC'98, Neuquén, Argentina, Outubro de 1998.

[GOO98] Goosens, B. Tipi: The Threads Processor, Proceedings of the MTEAC'98 - Workshop on Multithreaded Execution, Architecture and Compilation: held in conjuction with HPCA-4, Las Vegas, Nevada, February, 1998.

[GUS95] Gustavson, A.; Törn, A.: OO Tokens and Sensor Arcs in the Tool XSIMNET. IMACS European Simulation Meeting, at the Szechenyi Istvan College in Györ, Hungary, August 28-30, 1995.

[GUN98] Gunther, Neil J.: The Practical Performance Analyst: Performance-by-Design Techniques for Distributed Systems. McGraw-Hill, 1998.

[HEN94] Hennessy, J., Patterson, D. A., Computer Architecture: A Quantitative Approach., San Mateo, CA: Morgan Kaufmann, 1994.

[HIL98] Hily, S., Seznec, A., Out-of-Order Execution May Not Be Cost-Effective on Processors Featuring Simultaneous Multithreading, Publication Interne 1179, Institute de Recherche en Informatique et Systèmes Aléatoires (IRISA), Beaulieu, France, 1998.

[JAC96] Jacob, Bruce L.; Chen, Peter M.; Silverman, Seth R.; Mudge, Trevor N.: An Analytical Model for Designing Memory Hierarchies. IEEE Transactions on Computer, Col. 45, No. 10, Outubro de 1996.

[JAI91] Jain, R.: The Art of Computer Systems Performance Analysis. John Wiley and Sons, New York, 1991.

[JOH91] Johnson, M., Superscalar Microprocessor Design, Prentice Hall Series in Innovative Technology, PTR Prentice Hall, Englewood Cliffs, New Jersey, 288p., 1991.

[KAN97] Kant, L.; Sanders, W.H.: Analysis of the Distribution of Consecutive Cell Losses in an ATM Switch Using Stochastic Activity Networks. Special Issue of International Journal of Computer Systems Science & Engineering on ATM Switching, vol. 12, no. 2, March 1997, pp. 117-129.

[LIN98] Lindemann, Cristoph: Performance Modelling with Deterministic and Stochastic Petri Nets. John Wiley and Sons, 1998.

[MAR84] Marsan, M.; Baldo, G.; Conte, G.: A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. ACM Transactions on Computer Systems, May 1984.

[MOR98] Moreno, Edward D.; Kofuji, Sergio T.: Um Modelo RPDE para Busca Antecipada de Dados num Multiprocessador Baseado em um Simples Nó SMP. X Simpósio Brasileiro de Arquitetura de Computadores. Anais. Búzios, RJ - 28 a 30 de Setembro de 1998.

[PAT90] Patterson, David A.; Hennessy, John L.: Computer Architecture: A Quantitative Approach. Morgan Kaufmann Publishers, 1990.

[PET62] Petri, C. A.; Kommunikation mit Automaten, Tese de Doutorado, Universität Bonn, Germany, 1962.

[ROB94] Robertazzi, T.: Computer Networks and Systems: Queuing Theory and Performance Evaluation. Springer-Verlag, 1994.

[SAA90] Saavedra-Barrera, Rafael H.; Culler, David E.; von Eicken, Thorsten: Analysis of Multithreaded Architectures for Parallel Computing. 2nd Annual ACM Symposium on Parallel Algorithms and Architecture; Crete, Greece; July, 1990, pp. 169-178.

[SAH96] Sahner, Robin A.; Trivedi, Kishor S.; Puliafito, A.: Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package. Kluwer Academic Publishers, 1996.

[SAN95] Sanders, W.H.; Obal II, W.D.; Qureshi, M.A.; Widjanarko, F.K.; UltraSAN Version 3: Architecture, Features, and Implementation. Proceedings of the AIAA Computing in Aerospace 10 Conference, San Antonio, TX, March 28-30, 1995, pp. 327-338.

[SIL92] Silva, E.; Muntz, R.: Métodos Computacionais de Solução de Cadeias de Markov: Aplicações a Sistemas de Computação e Comunicação. VIII Escola de Computação, Gramado-RS, 1992.

[SMI95] Smith, J.E, Sohi, G.S., The Microarchitecture of SuperScalar Processors, Proceedings of the IEEE, 83(12), pp.1609-1624, December, 1995.

[TUL95] Tullsen, D. M., et all, Simultaneous Multithreading: Maximizing On-Chip Parallelism, Proceedings of the ISCA'95, Santa Margherita Ligure, Italy, Computer Architetcure News, n.2, v.23, 1995.