Universidade Federal do Rio de Janeiro
Pós-Gradução em Informática

Microarquiteturas de Alto Desempenho

Multithreading

Gabriel P. Silva

Introdução

- **Muitos dos sistemas operacionais modernos suportam o conceito de *threads*, ou seja, tarefas independentes dentro de um mesmo processo que podem ser executadas em paralelo ou de forma intercalada no tempo, dentro do esquema tradicional de *time-sharing*.**
- **Nesses sistemas, uma aplicação é descrita por um processo composto de várias *threads*.**
- **As *threads* de um mesmo processo compartilham o espaço de endereçamento de memória, arquivos abertos e outros recursos que caracterizam o contexto global de um processo como um todo.**
- **Cada *thread*, no entanto, tem seu próprio contexto específico, normalmente caracterizado pelo conjunto de registradores em uso, contador de programas e palavra de *status* do processador.**

Introdução

- **O contexto específico de uma *thread* é semelhante ao contexto de uma função e, conseqüentemente, a troca de contexto entre *threads* implica no salvamento e recuperação de contextos relativamente leves, a exemplo do que ocorre numa chamada de função dentro de um programa.**
- **Este fato é o principal atrativo em favor do uso de *threads* para se implementar um dado conjunto de tarefas em contraposição ao uso de múltiplos processos.**
- **A comunicação entre tarefas é grandemente simplificada numa implementação baseada em *threads*, uma vez que neste caso as tarefas compartilham o espaço de endereçamento de memória do processo como um todo que engloba as *threads*, eliminando a necessidade do uso de esquemas especiais, mais restritos e, usualmente, mais lentos de comunicação entre processos providos pelos sistemas operacionais.**

Introdução

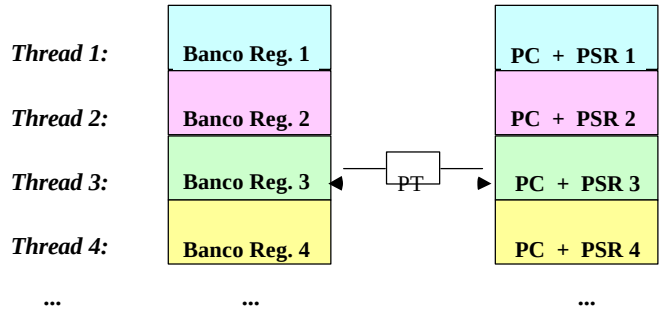
- **Tolerância da Latência**
- **As latências dos acessos de "load" no Alpha 21164 de 300 Mhz em um sistema com 4 processadores são:**
  - **7 ciclos para falha na cache primária (L1) com acerto na cache de nível 2, ambas dentro do chip;**
  - **21 ciclos para uma falha na cache de nível 2 (L2) com acerto na cache de nível 3 (L3), que está fora do chip;**
  - **80 ciclos para uma falha que é servida pela memória;**
  - **125 ciclos para uma falha que é servida pela cache de outro processador.**

Introdução

- **As arquiteturas *multithreading* procuram reduzir o *"overhead"* na troca de contexto entre *threads*, escondendo ou reduzindo o efeito negativo da latência de certas operações demoradas realizadas pelo processador.**
- **Os múltiplos contextos de *threads* nos processadores são criados através da replicação dos elementos de "hardware" que caracterizam cada contexto, tipicamente: o banco de registradores, o contador de programas e a palavra de *status* do processador.**

Multithreading

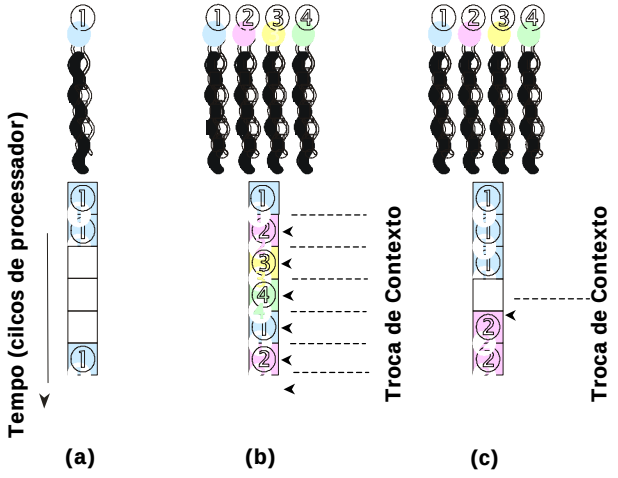
- **Rápida troca de contexto:**



Multithreading

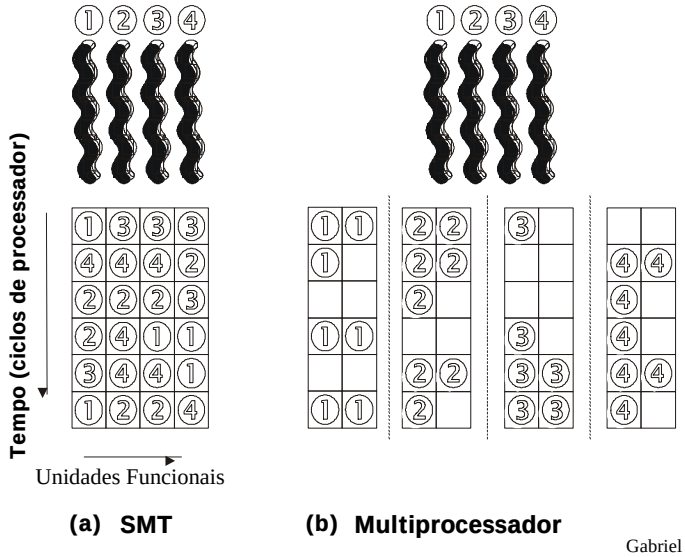
- **Granulosidade Fina**
  - **Uma instrução de cada *thread* ativa é buscada e enviada para o pipeline a cada ciclo.**
- **Granulosidade Grossa**
  - **As instruções de uma *thread* são executadas sucessivamente até que um evento de grande latência ocorra. Este evento provoca uma troca de contexto.**
- **Multithreading Simultâneo**
  - **Instruções são despachadas simultâneamente para as unidades funcionais de um processador superescalar.**
  - **Combina um processador superescalar com multithreading.**

Multithreading



(a) Processador c/ Pipeline Simples
(b) Multithreading c/ Granulosidade Fina
(c) Multithreading c/ Granulosidade Grossa

Multithreading Simultâneo (SMT)



Gabriel P. Silva

Granulosidade Fina

- O processador troca de contexto em todo ciclo
- Apenas uma instrução de cada contexto está presente no pipeline a cada instante de tempo
- Lógica de controle do pipeline é bastante simplificada pois não existem dependências de dados e de controle ➔ pipeline pode ser mais rápido
- Overhead para troca de contexto é nulo
- No mínimo, o número de contextos deve ser igual ao de estágios do pipeline. Mas, em geral, um número muito grande de contextos deve ser suportado em hardware para tornar a arquitetura efetiva
- O desempenho de código seqüencial (com uma única thread) é ruim
- Exemplo: Arquitetura MTA da Tera
  - Suporta 120 contextos
  - Pipeline com 21 estágios
  - Não possui cache de dados
  - 32 registradores e 1 PC por contexto

Gabriel P. Silva

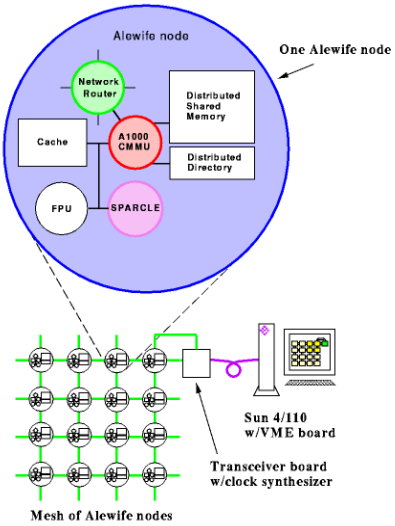
Granulosidade Grossa

- **Arquiteturas dessa classe se dividem em:**
  - **Estáticas:**
    - **Mecanismo Implícito: a troca de contexto é feita quando certas instruções (load, store, branch, etc.) são encontradas.**
    - **Mecanismo Explícito: quando instruções explícitas de troca de contexto são encontradas. O overhead de troca de contexto é 1 se a instrução é descartada no estágio de busca e zero se for executada.**
  - **Dinâmicas: o processador troca de contexto quando ocorre uma operação de longa latência (falha na cache, sincronização, etc.). O *overhead* de troca de contexto é proporcional à profundidade no pipeline do estágio que dispara a troca de contexto, já que as instruções posteriores no *pipeline* devem ser anuladas.**

Granulosidade Grossa

- **Um número não muito grande de contextos (4 a 32) é suportado**
- **Exemplo: SPARCLE (Máquina Alewife do MIT)**
  - **Baseado na arquitetura SPARC**
  - **Suporta 4 contextos cada um com 40 registradores (2 janelas da arquitetura SPARC)**
  - **Overhead de troca de contexto de 14 ciclos**
  - **Troca de contexto se dá quando há falha na cache ou em caso de falha em instruções sincronização.**

Granulosidade Grossa



Eficiência do Multithreading

- **Determinada pelos seguintes fatores:**
  - **Número de contextos suportados:**
    - **Maior número de contextos ➔ menores chances do processador parar por não ter mais nenhuma thread que possa continuar executando ➔ aumenta o custo de hardware ➔ só é efetivo se a aplicação comportar um grau de paralelismo alto**
  - **Overhead para realizar a troca de contexto x latência típica a ser ocultada**
    - **Aspecto a ser considerado em arquiteturas baseadas em multithreading de granulosidade grossa, em que a técnica só é efetiva se a troca de contexto é efetuada em um número menor de ciclos do que o gasto nas operações cuja latência deva ser ocultada**
  - **Comportamento da aplicação**
    - **Define com que freqüência operações de alta latência ocorrem. Para freqüências altas, a técnica de multithreading tem dificuldades em sobrepor a execução em um contexto com operações de longa latência de vários outros contextos.**

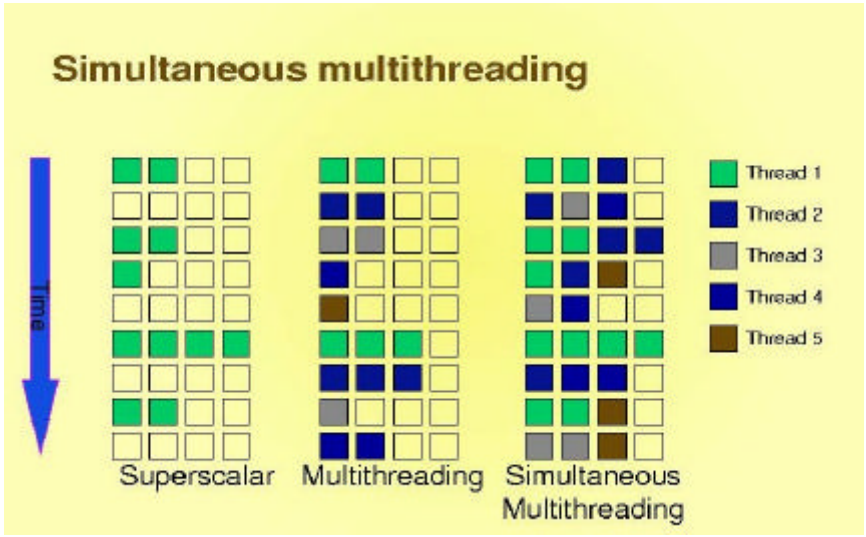
Multithreading Simultâneo

- **SMT é uma técnica que permite múltiplas *threads* despacharem múltiplas instruções a cada ciclo para unidades funcionais de um processador superescalar.**
- **SMT combina a capacidade de despacho de múltiplas instruções das arquiteturas superescalares, com a habilidade de esconder latência das arquiteturas *multithreading.***
- **A cada instante de tempo instruções de diferentes *threads* podem estar sendo executadas simultaneamente**
- **Busca reduzir o número de slots de despacho não ocupados a cada ciclo (elevado em arquiteturas *multithreading*) e o número de ciclos em que nenhuma instrução é despachada (elevado em arquiteturas superescalares)**

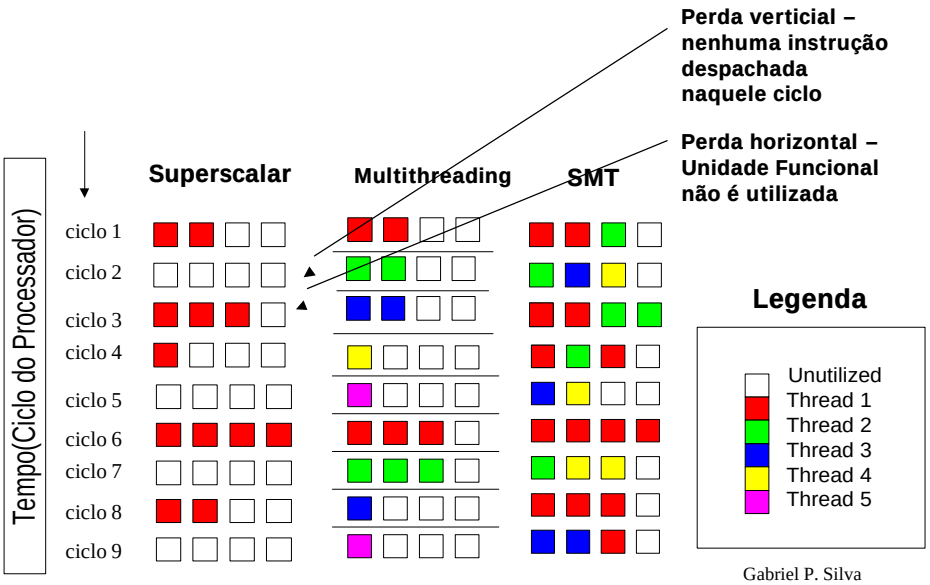
Multithreading Simultâneo

- **As arquiteturas SMT podem ser de três tipos básicos (Tullsen, ISCA95):**
  - **Full Simultaneous issue: modelo ideal, mas de difícil realização, onde não há restrição sobre o número de instruções de cada thread que pode ser despachada a cada ciclo**
  - **Limited Simultaneous Issue: apenas um número limitado (1 a 4 tipicamente) de instruções de cada thread pode ser despachado a cada ciclo**
  - **Limited Connection: restringem o número de unidades funcionais de cada tipo que podem estar conectadas a cada contexto**

Multithreading Simultâneo

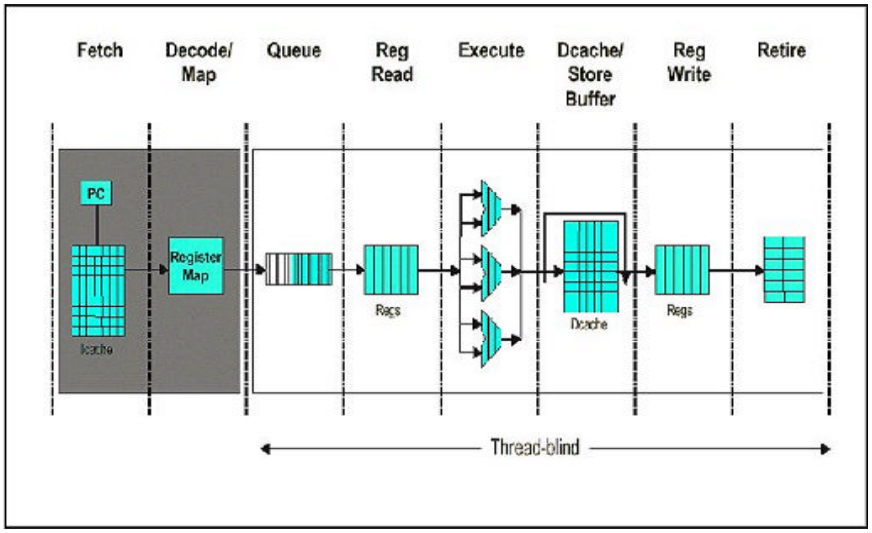


Comparação

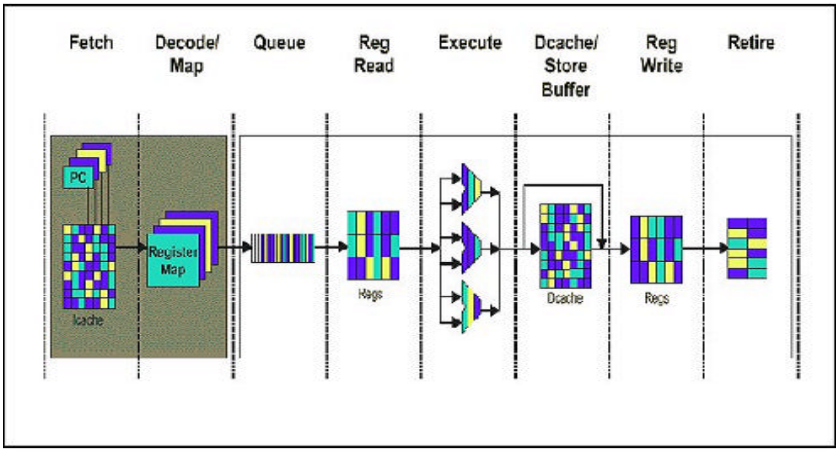


Gabriel P. Silva

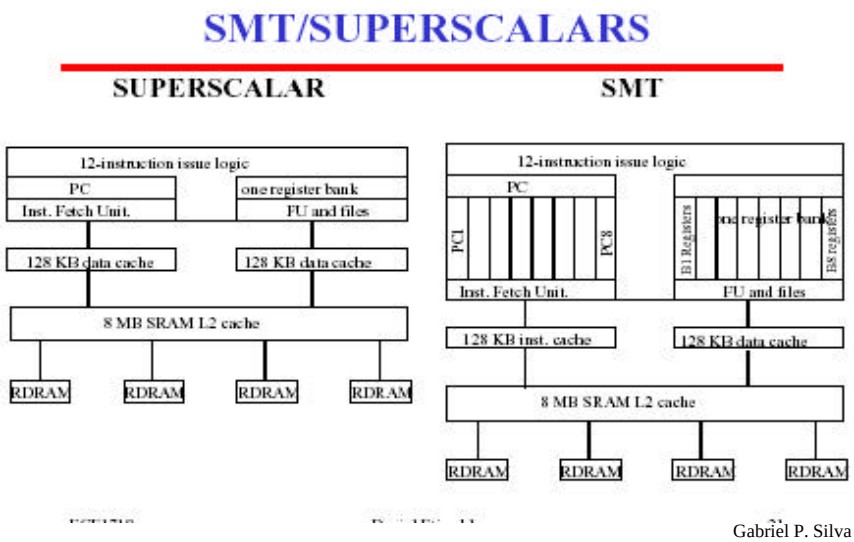
Processador Superescalar



Processador c/ Multithreading Simultâneo



Multithreading x Superescalar



Multithreading Simultâneo

- **Algumas mudanças são necessárias para o suporte a SMT:**
  - **Múltiplos PCs;**
  - **Pilhas de retornos separadas para cada *thread;***
  - **Identificação da *thread* em cada entrada do *branch target buffer (BTAC);***
  - **A predição de desvio é compartilhada;**
  - **A Memória, Cache, TLB, todo o hardware é compartilhado.**
- **Unidade de Busca:**
  - **Política de busca para selecionar a *thread* em que a próxima instrução será buscada.**

Problemas com SMT

- **Um banco de registradores grande, com registradores para as *threads* e registradores adicionais para renomeação:**
  - **Dois estágios no pipeline para acesso aos registradores**
- **Os dois estágios necessários para acesso aos registradores tem um impacto no pipeline:**
  - **Necessidade de mais um nível de *bypass;***
  - **Aumento da distância entre a busca e a execução:**
    - **Um ciclo a mais em caso de falha na predição de desvio;**
    - **Aumento da permanência das instruções de um caminho errado após a descoberta de erro na predição do desvio.**
  - **Aumento do tempo mínimo em que um registrador está ocupado.**

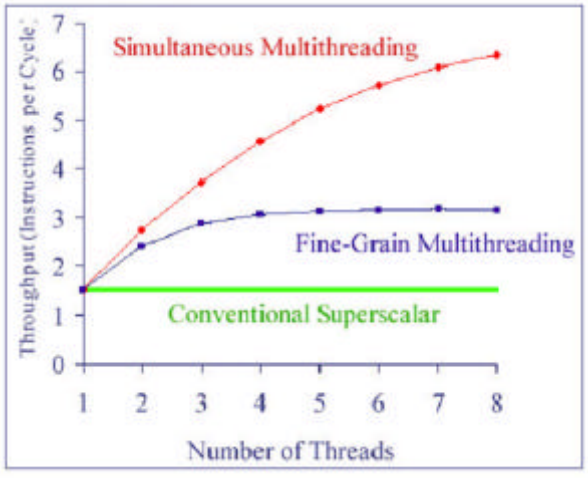
Problemas com SMT

- **Compartilhamento de recursos é um problema:**
  - **Múltiplas threads agora usam o mesmo esquema de predição de desvio:**
    - **Mais erros de predição**
    - **Entradas da BTAC compartilhadas**
  - **Múltiplas thread compartilham a cache. Mais falhas:**
    - **Diminuição da localidade com o compartilhamento;**
    - **As taxas de flahas da cache L1 podem subir até 66 %**
    - **A taxa de L2 não é tão afetada.**
  - **Aumento da taxas de falhas da TLB pelo seu compartilhamento.**

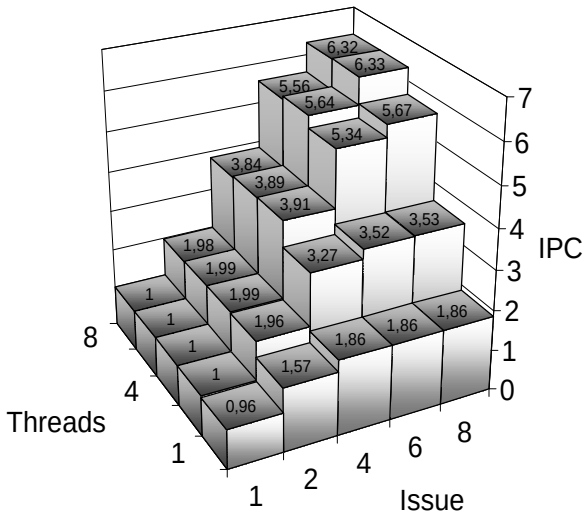
Problemas com SMT

- **Banco de registradores maior ➔ maior tempo de acesso. É adicionado 1 ciclo ao estágio de decodificação e um ciclo no estágio de escrita. São necessários então dois estágios adicionais no pipeline.**
  - **Apenas um decréscimo de 2% no desempenho, de modo que isto não é significativo.**
  - **Only decrease of 2%, so not that significant performance-wise.**
- **Todos esses problemas são mascarados pelo aumento no desempenho :**
  - **Uma arquitetura SMT com 8 threadd alcança um desempenho de 2 a 3 vezes maior que um processador superescalar.**

Potencial do SMT



Potencial do SMT



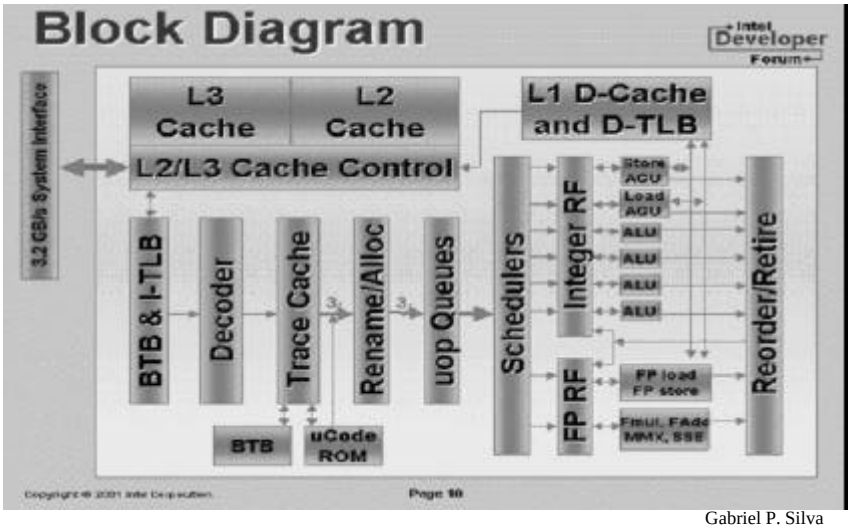
Hyperthreading Xeon

- **A Intel lançou o processador Intel Xeon, com dois processadores lógicos por processador físico.**
- **Esta implementação da tecnologia "hyperthreading" adicionou menos que 5% ao tamanho total do chip em relação ao consumo, mas pode oferecer ganhos de desempenho bem maiores que isso.**
- **De ponto de vista do software um processador aparece como sendo dois processadores.**
- **O número de transistores necessários para armazenar o estado arquitetural é uma fração extremamente pequena do total .**

Hyperthreading Xeon

- **Processadores compartilham todos os recursos no processador físico, tais como caches, unidades de execução, preditores de desvio, lógica de controle e barramentos.**
- **Cada processador lógico mantém uma cópia completa do estado arquitetural .**
- **Cada processador lógico tem seu próprio controlador de interrupção. Interrupções enviadas para um controlador lógico específico são manipuladas por cada processador lógico.**

Hyperthreading Xeon



Avaliação de Desempenho

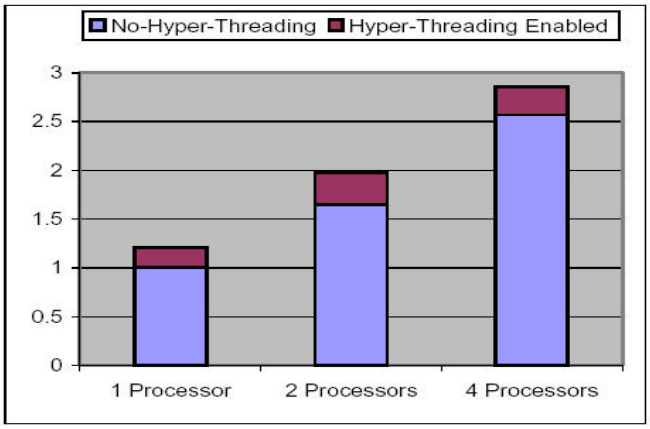


Figure 8: Performance increases from Hyper-Threading Technology on an OLTP workload

Avaliação de Desempenho

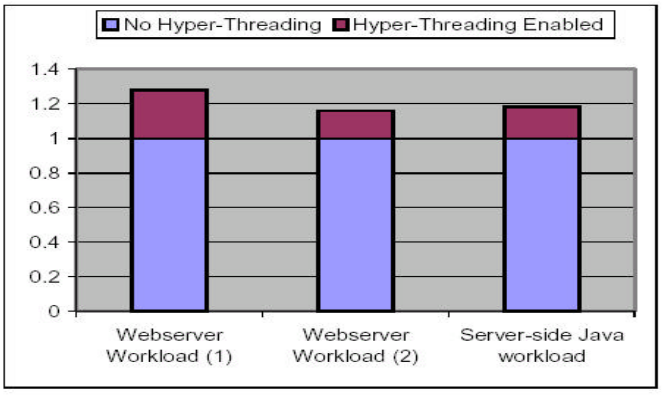


Figure 9: Web server benchmark performance

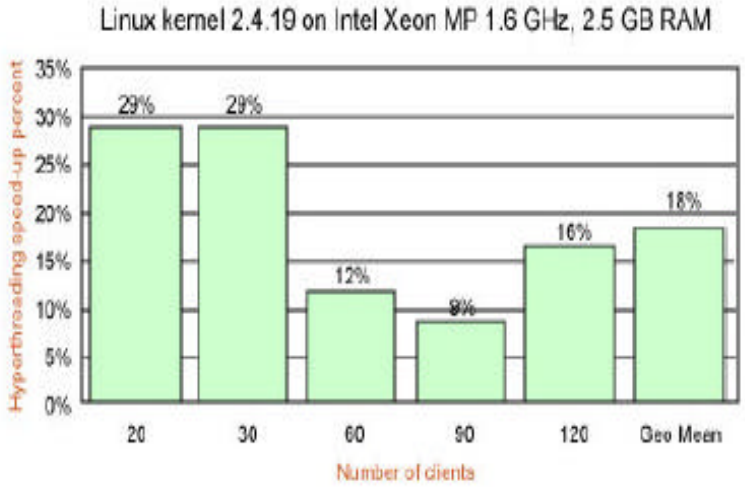
Hyperthreading no Linux

- **O kernel do linux 2.4.x faz uso do *hyperthreading* desde a versão 2.4.17.**
- **O kernel 2.4.17 sabe que existe um processador lógico, e trata-o com um segundo processador físico.**
- **Contudo, o escalanoador usado no kernel 2.4.x é ainda considerado primitivo para distinguir o problema de contenção de recursos entre dois processadores lógicos versus dois processadores físicos separados.**
- **O efeito do hyperthreading em um servidor de arquivos foi medido com o teste dbench e seu correspondente, tbench.**
- **Dbench é um programa de avaliação que permite medir o desempenho de servidores de arquivo manipulando pedidos de clientes através da rede.**

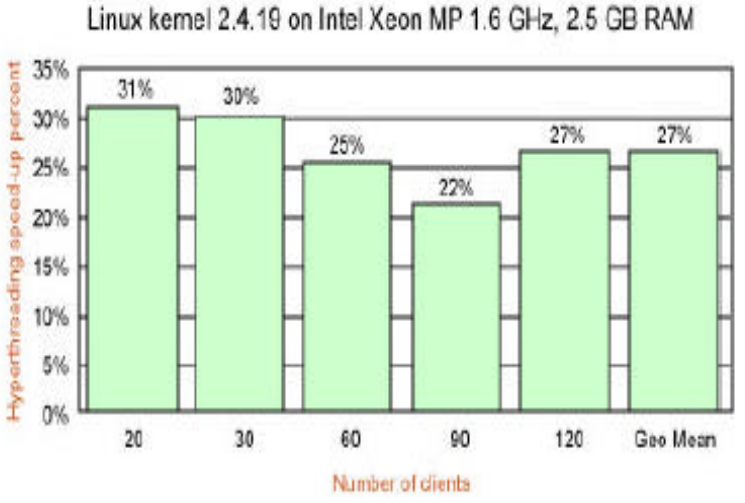
Hyperthreading no Linux

- **O programa dbench simula 90,000 operações de um cliente fazendo acesso a um arquivo de 4 MB.**
- **O conteúdo deste arquivo são diretivas como SMBopenx, SMBclose, SMBwritebraw, SMBgetatr, etc.**
- **Essas operações de E/S correspondem ao protocolo SMB que o servidor smbd do SAMBA produziria na sua execução.**
- **O tbench produz apenas a carga de processador e de TCP. O tbench faz as mesmas chamadas de socekts que o smbd faz, mas o tbench não faz chamadas ao sistema de arquivos.**
- **O protocolo SMB é utilizado pela Microsoft na família de sistemas operacionais Windows para compartilhar discos e impressoras.**

Avaliação de Desempenho (dbench)



Avaliação de Desempenho (tbench)



Multiprocessadores x Multithreading

- **A alternativa mais usual ao SMT, para aproveitamento da área em silício é o uso de multiprocessadores em um mesmo chip (CMP).**
- **Ainda não há um consenso sobre qual dessas implementações oferece os melhores resultados, ainda há muito o que pesquisar.**
- **Um dos problemas encontrados, em quaisquer dessas implementações, é na maior utilização da memória, diminuição na taxa de acerto da cache de instruções e do mecanismo de predição de desvios, devido a interferência entre as várias *threads* / processos em execução.**

SMT x CMP

- **A disputa de desempenho entre o SMT e o CMP não está decidida ainda.**
- **O CMP é mais fácil de implementar, mas apenas o SMT tem a capacidade de esconder as latências.**
- **O particionamento funcional da pastilha não é facilmente alcançado em um processador SMT devido ao despacho centralizado de instruções:**
  - **Uma separação das filas de "threads" é uma solução possível, embora não remova o despacho centralizado de instrução.**
  - **Uma combinação de multithreading simultâneo com CMP pode apresentar um desempenho melhor.**
- **Pesquisa: combinar uma organização SMT ou CMP com a habilidade de criar *threads* com suporte do compilador ou completamente dinamico além de um única thread:**
  - **Especulação ao nível de *thread*;**
  - **Parecido com o "multiscalar".**