Universidade Federal de São Carlos
Centro de Ciencias Exatas e Tecnologia
Programa de Pós-Graduação em Ciência da Computação

"Estudos de Técnicas de Virtualização de Memória em Arquiteturas Multi-Core"

Diego Pagliarini Vivencio

Orientador: Prof. Dr. Luis Carlos Trevelin

São Carlos – SP
Agosto de 2010

Universidade Federal de São Carlos
Centro de Ciencias Exatas e Tecnologia
Programa de Pós-Graduação em Ciência da Computação

"Estudos de Técnicas de Virtualização de Memória em Arquiteturas Multi-Core"

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como requisito parcial para obtenção do Título de Mestre em Ciência da Computação

Aluno:
Diego Pagliarini Vivencio

Orientador:
Prof. Dr. Luis Carlos Trevelin

São Carlos – SP
Agosto de 2010

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

V857et

Vivencio, Diego Pagliarini.
Estudos de técnicas de virtualização de memória em arquiteturas multi-core / Diego Pagliarini Vivencio. -- São Carlos : UFSCar, 2014.
80 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2010.

1. Arquitetura de computador. 2. Virtualização. 3. *Multicore*. 4. Paginação aninhada. 5. Análise de desempenho. I. Título.

CDD: 004.22 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Estudo de Técnicas de Virtualização de Memória em Arquitetura Multi-Core"

DIEGO PAGLIARINI VIVÊNCIO

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

Membros da Banca:

Prof. Dr. Luis Carlos Trevelin
(Orientador - DC/UFSCar)

Prof. Dr. Márcio Merino Fernandes
(DC/UFSCar)

Prof. Dr. Edson Toshimi Midorikawa
(POLI/USP)

São Carlos
Setembro/2010

Dedicatória

Dedico esse trabalho aos meus pais, Nelson Vivencio e Isabel Aparecida Pagliarini Vivencio, a quem devo minha formação moral, valores e princípios, e que sempre acreditaram em mim.

Agradecimentos

Agradeço primeiramente aos meus pais, Nelson e Isabel, ao meu irmão Rodrigo e sua esposa Marinês, à minha avó Cecília, à minha prima Kátia e seu esposo Edgard, que me apoiaram durante esse trabalho.

À minha namorada Livia Maria Damião, pelo apoio na fase final da dissertação.

Aos amigos Carlos Roberto Pereira Almeida Júnior e Daniel Zardi Fernandes, que me auxiliaram no desenvolvimento do trabalho.

Ao professor Luis Carlos Trevelin, pela atenção e apoio durante todo o período do mestrado.

Aos meus superiores nas empresas e órgãos públicos em que trabalhei durante o mestrado, que me ofereceram condições para que pudesse conciliar a atividade profissional com a pesquisa acadêmica.

Antes da batalha, o planejamento é tudo. Assim que começa o tiroteio, os planos são inúteis. (Dwight Eisenhower)

O único lugar onde o sucesso vem antes do trabalho é no dicionário. (Albert Einstein)

Desistir é uma solução permanente para um problema temporário. (James MacArthur)

Resumo

O uso de virtualização de computadores vem crescendo rapidamente nos últimos anos, motivando a pesquisa de melhorias em software e hardware que permitam aumento de desempenho e a redução dos gargalos inerentes à virtualização. Em meados desta década, os processadores adicionaram suporte a virtualização de CPU, simplificando o projeto dos monitores de máquinas virtuais, porém o modelo adotado apresentava restrições de desempenho quando combinado com a virtualização de memória utilizando tabelas de páginas de sombra. A inclusão de suporte a paginação aninhada em hardware foi a resposta a esse problema, oferecendo desempenho mais próximo ao nativo, isto é, sem a abstração virtual. Os processadores com múltiplos núcleos, também conhecidos como *multi-core*, foram a solução encontrada para manter o aumento de desempenho dos microprocessadores, visto que as arquiteturas monolíticas estavam próximo ao seu limite. A utilização de virtualização permite explorar o paralelismo oferecido por eles através da execução simultânea de múltiplas máquinas virtuais. O presente trabalho avalia o subsistema de virtualização de memória e sua interação com as arquiteturas multi-core, visando determinar o conjunto de características que maximizem o desempenho. Foram avaliadas as paginações de sombra e aninhada, comparando a utilização de páginas convencionais e grandes para o mapeamento da memória virtualizada. Também foi avaliada a influência decorrente da presença de um nível de memória cache compartilhado entre os núcleos de processamento. Os resultados mostraram que no cenário avaliado o melhor desempenho foi obtido através de utilização de paginação aninhada utilizando páginas grandes para o mapeamento da memória, enquanto que o nível adicional de memória cache não trouxe benefícios específicos à virtualização.

Palavras-Chave: virtualização, multi-core, paginação aninhada e análise de desempenho.

Abstract

The use of computer virtualization has grown rapidly in recent years, motivating the research for software and hardware improvements to optimize performance and reduce the bottlenecks inherent of virtualization. In the middle of this decade, the processors has added support for CPU virtualization, simplifying the design of virtual machine monitors, but the employed approach had performance limitations when combined with the virtualization of memory using shadow page tables. Adding support for nested paging hardware was the answer to this problem, providing performance closer to the native, ie, without the virtual abstraction. The multicore processors were the solution to keep the microprocessors' performance growth , as the monolithic architectures were close to their limit. The use of virtualization allows exploiting parallelism offered by them through the simultaneous execution of multiple virtual machines. This study evaluates the virtualization of memory subsystem and its interaction with the multicore architectures, to determine the set of features that maximize performance. We evaluated shadow and nested paging, comparing the use of conventional and large pages to map virtualized memory. We also analyzed the influence due to the presence of a level of cache shared among cores. The results showed that in the evaluated scenario the best performance was achieved by using nested paging using large pages to map memory, while the additional level of cache didn't bring any specific benefits to virtualization.

Keywords: virtualization, multicore, nested paging and performance analysis.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

Abrev.	Termo	Tradução
JVM	Java Virtual Machine	Máquina Virtual Java
VMM	Virtual Machine Monitor	Monitor de Máquina Virtual
VM	Virtual Machine	Máquina Virtual
KVM	Kernel Virtual Machine	Máquina Virtual do Kernel (Linux)
PCI	Peripheral Componente Interconnect	Interconexão de Componentes Periféricos
IRQ	Interrupt Request	Requisição de Interrupção
DMA	Direct Memory Access	Acesso Direto à Memória
ABI	Application Binary Interface	Interface Binária de Aplicação
CPU	Central Processing Unit	Unidade de Processamento Central
OS	Operating System	Sistema Operacional
JIT	Just In Time	
VMCS	Virtual Machine Control Structure	Estrutura de Controle de Máquina Virtual
VMCB	Virtual Machine Control Block	Bloco de Controle de Máquina Virtual
TLB	Translation Lookaside Buffer	*Buffer* de Tradução de Endereços
AE	Physical Address Extension	Extensão de Endereço Físico
PSE	Page Size Extension	Extensão de Tamanho de Página
MMU	Memory Management Unit	Unidade de Gerenciamento de Memória
NUMA	Non Uniform Memory Access	Acesso Não-Uniforme a Memória

Abrev.	Termo	Tradução
RVI	Rapid Virtualization Indexing	Indexação de Virtualização Rápida
EPT	Extended Page Tables	Tabelas de Páginas Estendidas
COW	Copy On Write	Cópia em Gravação
KSM	Kernel Samepage Merging	Mesclagem de Páginas pelo *Kernel*
SMT	Simultaneous Multi Threading	
ALU	Arithmetic Logic Unit	Unidade Lógica e Aritmética
FPU	Float Point Unit	Unidade de Ponto Flutuante
SMP	Symmetric Multi Processing	Multiprocessamento Simétrico
RAM	Random Access Memory	Memória de Acesso Aleatório
MESI	Modified/Exclusive/Shared/Invalid	Modificado/Exclusivo/ Compartilhado/Inválido
MOESI	Modified/Owned/Exclusive/ Shared/Invalid	Modificado/Possuído/Exclusivo/ Compartilhado/Inválido
CMP	Chip Multi Processing	Multiprocessamento de Chip
HT	HyperThreading	
BIU	Bus Interface Unit	Unidade de Interface de Barramento
TLP	Thread Level Parallelism	Paralelismo no Nível de *Thread*
VSMP	Virtual Symmetric Multi Processing	Multiprocessamento Simétrico Virtual
VCPU	Virtual Central Processing Unit	Unidade Central de Processamento Virtual
PWC	Page Walk Cache	*Cache* de Navegação de Páginas
GCC	GNU Compiler Collection	Coleção de Compiladores GNU

Sumário

Capítulo 1

Introdução

1.1 Considerações Iniciais

A virtualização de computadores é uma tecnologia cuja utilização remonta da década de 60, inicialmente restrita a *mainframes* e outros computadores de grande porte. Na última década houve um grande interesse nessa área, tanto pela comunidade acadêmica como pela indústria, em grande parte pela introdução de ferramentas de virtualização empregando a disseminada arquitetura x86, utilizando técnicas como a descrita em [1].

Esse interesse levou ao desenvolvimento de várias técnicas de virtualização e pesquisas voltadas a diminuir os gargalos impostos pela adição de uma camada de abstração, a exemplo das apresentadas em [2], [3] e [4]. A evolução mais recente foi a inclusão por parte dos fabricantes de microprocessadores de recursos em hardware para otimizar a execução de múltiplas máquinas virtuais, como proposto em [5].

As principais motivações para utilização de virtualização são o melhor aproveitamento do equipamento, bem como a independência da máquina virtual em relação ao hardware subjacente. Essa última característica simplifica mecanismos como balanceamento de carga, alta disponibilidade e recuperação de desastres.

As arquiteturas *multi-core* são uma evolução das arquiteturas tradicionais de multiprocessamento. Como principais vantagens, a integração de múltiplos núcleos de processamento em um único encapsulamento permite otimizar a comunicação entre núcleos através de barramentos internos de alta velocidade. A integração de memória cache compartilhada

oferece o benefício de atuar como canal de troca de dados entre núcleos, além de permitir um melhor aproveitamento espacial, de forma que núcleos com maior volume de processamento utilizem uma parcela maior da memória cache do que núcleos que estejam ociosos.

A tendência recente de crescimento do número de núcleos nos microprocessadores levou a uma maior necessidade de paralelismo por parte das aplicações, porém nem todas as aplicações são passíveis de ser paralelizadas, seja por restrições técnicas ou de custo de implementação. O emprego de virtualização permite obter pleno aproveitamento de múltiplos núcleos de processamento consolidando múltiplas máquinas virtuais em um único equipamento, sem levar a questão do paralelismo às aplicações.

Um dos principais aspectos referentes ao desempenho das máquinas virtuais consiste na virtualização do subsistema de memória. As primeiras alternativas possuíam restrições, como o baixo desempenho em algumas cargas de trabalhos ou a necessidade de alterações no sistema operacional da máquina virtual para oferecer desempenho mais próximo ao nativo. Recentemente, foi adicionado o suporte à virtualização de memória em hardware, que permite simplificar o projeto das soluções de virtualização ao mesmo tempo que oferece um melhor desempenho sem a necessidade de alterações no sistema operacional.

1.2 Objetivos

O objetivo desse trabalho é estudar as técnicas de virtualização avaliando o seu desempenho em arquiteturas *multi-core*, visando determinar quais características propiciam uma melhor interação. O projeto dará enfoque à virtualização do subsistema de memória, que é um dos aspectos críticos para o bom desempenho de uma solução de virtualização, verificando qual conjunto de configurações permite um aproveitamento máximo dos processadores que possuem múltiplos núcleos de execução. Serão avaliados os seguintes aspectos:

- Comparação dos modos de paginação utilizando tabelas de páginas de sombra a tabelas de páginas aninhadas;
- Utilização de páginas de tamanho convencional (4 KB) e páginas grandes (2 MB) para o mapeamento da memória das máquinas virtuais;

- A influência da presença de um nível de cache compartilhado entre os múltiplos núcleos de execução.

1.3 Organização do Trabalho

A sequência deste trabalho está organizada em cinco capítulos:

- Capítulo 2 – Técnicas de Virtualização: apresenta os modelos de virtualização de arquitetura e os modos de virtualização do processador;
- Capítulo 3 – Virtualização do Subsistema de Memória: descreve os modelos de virtualização do subsistema de memória e as principais técnicas de otimização;
- Capítulo 4 – Arquiteturas *Multi-Core*: apresenta os principais modelos de multiprocessamento e *multi-core*, incluindo exemplos de implementação;
- Capítulo 5 – Desenvolvimento do Projeto e Resultados: descreve o trabalho desenvolvido, compreendendo relevância, trabalhos relacionados, metodologia e os resultados obtidos;
- Capítulo 6 – Conclusões e Trabalhos Futuros: apresenta as conclusões do trabalho desenvolvido e propõe algumas linhas para trabalhos futuros.

Capítulo 2

Técnicas de Virtualização

2.1 Considerações Iniciais

A virtualização, em linhas gerais, consiste em ocultar detalhes de uma determinada arquitetura, oferecendo uma abstração em seu lugar. Um exemplo é a memória virtual, conceito criado na década de 50, que atualmente é utilizada na maioria dos sistemas operacionais e processadores modernos, e que simplificou o projeto dos aplicativos ao ocultar a existência de uma arquitetura de dois níveis no acesso à memória. Na década de 60, a IBM criou soluções utilizando o conceito de máquinas virtuais, permitindo o compartilhamento de recursos de uma única máquina entre múltiplos sistemas operacionais potencialmente distintos. Um outro exemplo é a Máquina Virtual Java (JVM), que não abstrai diretamente um sistema real, mas hoje também é considerada uma solução de virtualização.

2.2 Benefícios

Atualmente, há um grande interesse em soluções de virtualização, visando dentre vários aspectos a redução de custos, o melhor aproveitamento de equipamentos, o gerenciamento centralizado, a simplificação de soluções de *backup*, a recuperação de desastres e a maior facilidade na construção de ambientes de teste.

Dentro do ambiente *desktop*, as soluções de virtualização obtiveram penetração em várias áreas, como na execução de múltiplos sistemas operacionais em máquinas virtuais, a fim de ter à disposição softwares que não são oferecidos para o sistema operacional instalado no

equipamento. Um exemplo tradicional são os *webdesigners*, que precisam testar o aspecto das páginas em diversos navegadores e sistemas operacionais diferentes, e utilizar diversos computadores ou *multi-booting* para isso não é uma solução eficiente.

Uma solução corporativa baseada em máquinas virtuais é a virtualização de *desktops*, na qual os *desktops* são providos por máquinas virtuais hospedadas em servidores, sendo acessadas pelos usuários através de um terminal de acesso.

Nos servidores, a virtualização é focada principalmente na ideia da consolidação, em que vários servidores são substituídos por máquinas virtuais, rodando sobre um conjunto menor de servidores, com maior aproveitamento do hardware. Essa redução tem como impacto direto menor custo de propriedade, mas abre um grande número de possibilidades, como redução nos custos de energia, refrigeração e espaço físico, possibilidade de emprego de hardware com maior confiabilidade e possibilidade de replicação de serviços mais acessível, devido à redução na quantidade de equipamentos que necessitam ser duplicados.

A utilização de máquinas virtuais em ambiente servidor oferece outras possibilidades, como a migração de máquinas virtuais entre servidores físicos sem tempo de parada perceptível, como apresentado em [6] e [7]. Com isso, é possível realizar manutenções e atualizações de hardware sem exigir janela de parada, que são difíceis de obter no caso de aplicações de missão crítica. Outra possibilidade é realizar a migração de máquinas virtuais visando balanceamento de carga, migrando dinamicamente máquinas virtuais de forma a evitar que um servidor opere com 100% de ocupação de algum recurso, causando degradação na performance das máquinas virtuais, enquanto há recursos disponíveis em outros servidores. Baseado no mesmo conceito, é possível reduzir o consumo total de energia através da consolidação das máquinas virtuais num menor número de servidores nos momentos de baixo volume de processamento, permitindo desligar os servidores ociosos e reativá-los posteriormente sob demanda, conforme implementação descrita em [8] e proposta apresentada em [9].

2.3 Técnicas de virtualização

A área de virtualização é alvo de pesquisas há pouco mais de 50 anos [10]. Nesse período, foram propostas várias técnicas distintas para virtualização de computadores, algumas delas tendo se consolidado e mostrado sua viabilidade através da adoção por diversas soluções comerciais.

Algo a observar é que das várias técnicas que se mostraram viáveis, todas apresentam benefícios, da mesma forma que possuem limitações que impediram que existisse um modelo dominante. Algumas delas oferecem bom desempenho, próximo à execução nativa (direta no hardware), mas exigem para isso alterações nos sistemas operacionais ou mesmo nas aplicações, o que torna seu emprego um processo complexo e trabalhoso [3]. Outras técnicas proporcionam à máquina virtual um modelo idêntico ao hardware real, não exigindo alterações no software, mas tipicamente oferecem desempenho limitado ou exigem soluções complexas para atingir um desempenho próximo ao nativo. Alguns modelos sacrificam a segurança, relaxando o modelo de proteção das máquinas virtuais em troca de maior desempenho. Em resposta à crescente adoção de soluções de virtualização, alguns recursos foram adicionados ao hardware de forma a minimizar a diferença de desempenho entre a execução nativa e virtual.

O estudo sobre técnicas de virtualização exposto nesse capítulo analisa prioritariamente aspectos referentes à arquitetura x86. Como a utilização de virtualização é relativamente recente nessa arquitetura, ainda há inúmeros desafios a serem superados, gerando um grande volume de pesquisa e consequentemente publicações nessa área. Nas arquiteturas *mainframe*, que já utilizam virtualização para particionamento de hardware desde a década de 60, há um volume muito menor de pontos para melhoria. Apesar desse enfoque, os conceitos e vários aspectos detalhados não estão atrelados diretamente a nenhuma arquitetura em especial.

2.4 O Virtual Machine Monitor

Para permitir que a virtualização de um sistema se realize de maneira controlada é necessário um software, denominado Virtual Machine Monitor (VMM). Em 1974, Popek e Goldberg [11] definiram três características necessárias para classificar um software como um VMM:

- **Fidelidade:** programas rodando sobre o VMM executam de forma idêntica à execução em hardware, com exceção de efeitos na temporização. Esta exceção deve-se ao fato do hardware ser multiplexado entre as diversas máquinas virtuais ativas. Devido a esta característica, uma determinada máquina virtual não se mantém em execução no hardware na totalidade do tempo.
- **Desempenho:** uma quantidade significativa das instruções dos sistemas convidados devem ser executadas diretamente no hardware sem intervenção do VMM. A intervenção do VMM para a execução de instruções, por mais eficiente que seja, acrescenta um esforço extra na execução, devendo portanto limitar-se ao mínimo as instruções interceptadas por ele.
- **Segurança:** o VMM é responsável por gerenciar os recursos de hardware. Isso é necessário para garantir uma divisão justa de recursos entre as máquinas virtuais, bem como evitar brechas de segurança, como uma VM maliciosa acessar a região de memória de outra ou interceptar pacotes de rede alheios.

2.4.1 Categorias de VMM

Os VMMs podem ser classificados, baseado na forma de construção, em duas categorias [10]:

- **Tipo 1 (ou *Hypervisor*):** nessa abordagem (Figura 2.1), o VMM executa diretamente no hardware. Esse modelo é explorado por soluções como o VMware ESXi [12], Xen [13] e KVM [14].

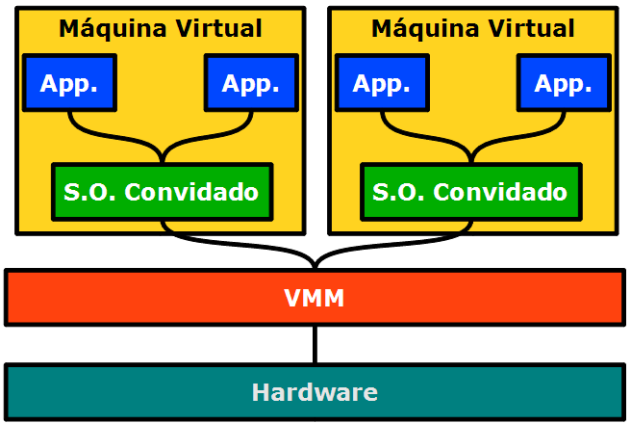

Figura 2.1: VMM Tipo 1

- **Tipo 2 (ou *Hosted*):** o VMM executa sobre um Sistema Operacional Anfitrião (*Host Operating System*), na forma de um processo (Figura 2.2). Esse modelo é utilizado no VMware Server [15] e no VirtualBox [16]. Se, por um lado, este modelo tipicamente apresenta desempenho inferior ao tipo 1, ele oferece a vantagem de permitir que o equipamento desempenhe outras funções além de executar máquinas virtuais, bem como desonerar o VMM do suporte a hardware.

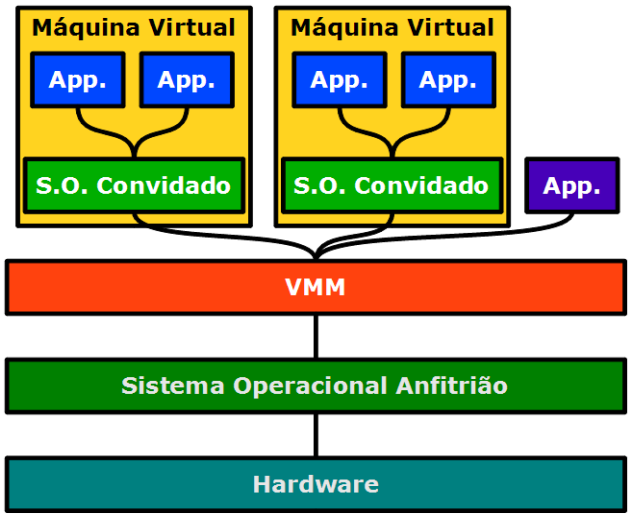

Figura 2.2: VMM Tipo 2

2.5 Formas de Virtualização de Arquitetura

Existem basicamente duas vertentes de virtualização de arquitetura: a virtualização total e a para-virtualização. Enquanto a virtualização total tenta construir um ambiente o mais próximo possível da arquitetura a ser virtualizada, no modelo para-virtualizado não existe essa rigidez, podendo haver mudanças em favor de maior desempenho ou melhorias no compartilhamento de recursos.

2.5.1 Virtualização Total

A virtualização total oferece uma abstração de hardware que permite utilizar o mesmo Sistema Operacional que seria instalado normalmente em um equipamento físico. Seu objetivo é constituir um ambiente com características que tornem impossível para o sistema determinar que está executando sobre uma máquina virtual. Por exemplo, para viabilizar o acesso à

rede é empregada uma placa de rede virtual, com slot PCI, IRQ e canal DMA, mas que não correspondem aos da placa de rede fisicamente instalada no equipamento. Nesse caso, cabe ao VMM encaminhar os pacotes enviados pela máquina virtual para a rede através da placa física.

Esse processo de abstração completa do sistema aumenta a complexidade do VMM. Várias operações que acessariam o hardware precisam ser interceptadas, adicionando uma camada que provê isolamento das máquinas virtuais, mas que interfere negativamente no desempenho.

2.5.2 Para-virtualização

Uma maneira de evitar as complexidades impostas pela virtualização total é utilizar uma abstração de máquina virtual similar, mas não idêntica, ao hardware subjacente. Essa abordagem é conhecida como para-virtualização. Apesar de possibilitar melhor desempenho, por não haver a necessidade de simular toda a estrutura do hardware, há a necessidade de modificar o sistema operacional convidado (*guest operating system*). Apesar disso, a ABI é mantida inalterada [3], não exigindo modificação nas aplicações executadas nos sistemas convidados.

2.6 Virtualização de CPU

A virtualização da CPU oferece alguns desafios ao funcionamento dos sistemas operacionais convidados. A presença do VMM viola a ideia de que o SO é o elemento com maior privilégio dentro do sistema. Como medida de segurança, os sistemas operacionais devem rodar com um nível de privilégio inferior ao do VMM.

A arquitetura x86 oferece quatro níveis de privilégios diretamente em hardware, permitindo de maneira eficiente a separação de níveis entre sistema convidado e VMM (Figura 2.3). Os níveis de privilégio são também conhecidos como *rings* (anéis), numerados de zero – que oferece o máximo privilégio de execução – a três. A implementação típica de sistemas operacionais mantém o sistema no nível zero (Figura 2.3 **(a)**), por ser o único capaz de executar instruções privilegiadas, relegando as aplicações ao nível três. Uma solução consiste em deslocar o sistema operacional convidado para o nível um, posicionando o VMM no nível

zero [16], como pode ser observado na Figura 2.3 **(b)**. Essa abordagem garante que apenas o VMM tenha acesso ao estado privilegiado da máquina, e mantém o sistema convidado isolado das aplicações, mantidas no nível três. Qualquer tentativa de executar diretamente uma instrução privilegiada será infrutífera, pois apenas o VMM dispõe de privilégios suficientes para isso.

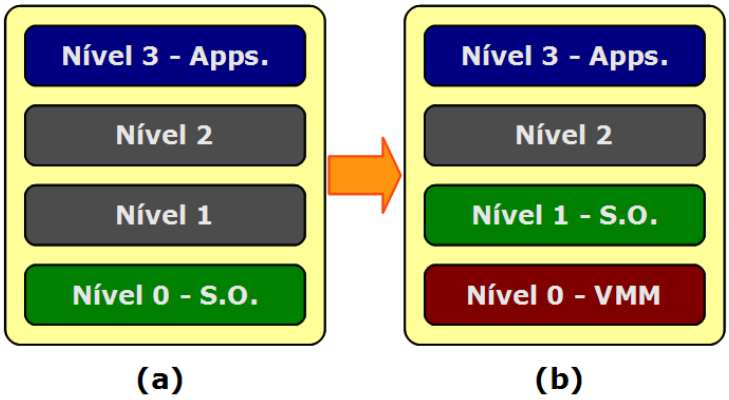

Figura 2.3: Níveis de execução em uma CPU x86

2.6.1 Virtualização Total da CPU

A arquitetura x86 possui algumas limitações que tornam a implementação de um VMM uma tarefa não trivial, sendo a principal delas o fato de que nem todas as instruções que manipulam estado privilegiado da máquina poder ser capturadas ao executar em modo de usuário.

2.6.1.1 Emulação

Uma solução para esse problema é utilizar emulação, interpretando cada instrução do sistema operacional convidado em vez de executá-la diretamente na CPU. Essa solução, se por um lado atende o critério de segurança de Popek e Goldberg [11], não é capaz de atender o critério de desempenho, pois cada instrução precisaria de várias instruções do emulador para ser executada.

Um exemplar dessa abordagem é o Bochs ([17] e [18]), emulador da arquitetura IA-32 (x86) que, apesar de oferecer uma compatibilidade total com o conjunto de instruções emulado, possui uma performance limitada.

2.6.1.2 Tradução Binária

Outra proposta é a adoção da tradução binária, que consiste em processar o código antes de sua execução, substituindo as instruções privilegiadas por um conjunto de instruções que possam executar em modo de usuário ou por um desvio para o VMM [19]. As instruções traduzidas são armazenadas em um cache de tradução (*translation cache*).

O processo de tradução ocorre sob demanda, intercalado com a execução do código. Conforme o conjunto de trabalho do sistema convidado é processado pelo tradutor binário e armazenado no cache de tradução, a tendência é haver uma redução dos ciclos de alternância entre tradução e execução. Ramos de código que raramente são executados ou rotinas de detecção de erro e tratamento de exceções tendem a ser traduzidos posteriormente, um comportamento semelhante ao desempenhado por um compilador Java JIT. Um efeito desse comportamento é que trechos de código com execução frequente possuem boa localidade dentro do cache de instruções da CPU.

Uma característica importante desse processo de virtualização é que como as instruções de modo de usuário do sistema convidado são consideradas seguras, é possível alternar entre o tradutor binário e a execução direta de instruções quando se alterna entre o *kernel* e o modo de usuário no sistema operacional convidado, e com isso garantindo que todo código em modo de usuário obtenha desempenho idêntico aos exibidos pela execução nativa. Essa característica é observada nos testes conduzidos em [19], nos quais é possível observar que os resultados para aplicativos que não exigem acesso ao *kernel*, o desempenho observado é superior a 95%.

2.6.1.3 Tradução binária adaptativa

A captura de instruções tem um custo alto nas CPUs modernas, sendo que sua substituição através de tradução binária pode oferecer melhor desempenho. A forma de tradução binária descrita em [19] atua sobre instruções privilegiadas, mas existe um outro conjunto de instruções que também precisam ser capturadas, que são as instruções não-privilegiadas que manipulam estado privilegiado, como as tabelas de páginas. A solução adotada nesse caso é chamada tradução binária adaptativa. Num primeiro momento, essas instruções são traduzidas de forma idêntica. Durante a execução, ao detectar instruções que são capturadas frequente-

mente, a tradução original é substituída por outra que evite a captura, ao custo de adicionar um desvio para bloco com a nova tradução.

2.6.2 Para-Virtualização da CPU

Enquanto na virtualização total o VMM assume a responsabilidade de ocultar as diferenças existentes ao executar o Sistema Operacional convidado em um nível de privilégios inferior, na para-virtualização da CPU o SO convidado é modificado, substituindo instruções que não são virtualizáveis e partes do sistema nas quais o desempenho é crítico por chamadas a uma API de baixo nível [4], de maneira a evitar redução de desempenho.

Além dessa substituição, é possível adotar um maior grau de comunicação entre o VMM e o SO convidado. O tratamento de exceções, como falta de páginas e interrupções por software (*software traps*), pode ser realizado através do registro de manipuladores (*handlers*) no VMM. A segurança nesse modelo pode ser garantida durante o registro [3], exigindo apenas que o manipulador não especifique execução de código no nível zero.

Apesar de tipicamente oferecer um maior desempenho do que soluções baseadas em virtualização total, como demonstrado em [3] e [20], uma limitação referente a essa abordagem reside na complexidade introduzida no *kernel* do sistema convidado. O emprego de para-virtualização exige que os desenvolvedores tenham que lidar com uma CPU virtual, que possui limitações e comportamento diferentes de uma CPU real, como exposto em [21] e [22]. A semântica da CPU virtual é definida pelo VMM e seu comportamento pode variar entre VMMs diferentes.

2.6.3 Virtualização da CPU com assistência de hardware

Em 2006 Intel e AMD adicionaram extensões aos seus processadores, tornando-os capazes de realizar virtualização utilizando a técnica de capturar e emular (*trap and emulate*). Apesar das soluções das duas empresas não serem compatíveis, suas implementações são semelhantes ([23], [24] e [25]). Essa semelhança permitiu que várias soluções oferecessem suporte a ambas, como a VMware [26], e os projetos Xen [13] e KVM [14].

2.6.3.1 Extensões de virtualização na arquitetura x86

O modelo proposto para virtualização de hardware consiste basicamente em exportar um conjunto de primitivas de forma a suportar um VMM clássico (*trap and emulate*) na arquitetura x86. Para seu funcionamento, é utilizada uma estrutura de controle armazenada em memória, denominada VMCS pela Intel e VMCB nas CPUs AMD, responsável por armazenar um subconjunto do estado da CPU virtual, bem como estado de controle. Um novo modo de execução foi adicionado à CPU, denominado modo convidado (*guest mode*) (Figura 2.4 **(b)**). Ele é menos privilegiado que o modo convencional no qual os Sistemas Operacionais executam normalmente, mas, ainda assim, permite a execução direta de código do sistema convidado, incluindo instruções privilegiadas. Uma instrução, *VMENTER* na implementação da Intel e *VMRUN* na implementação da AMD, é responsável por alternar entre o modo tradicional e o novo modo.

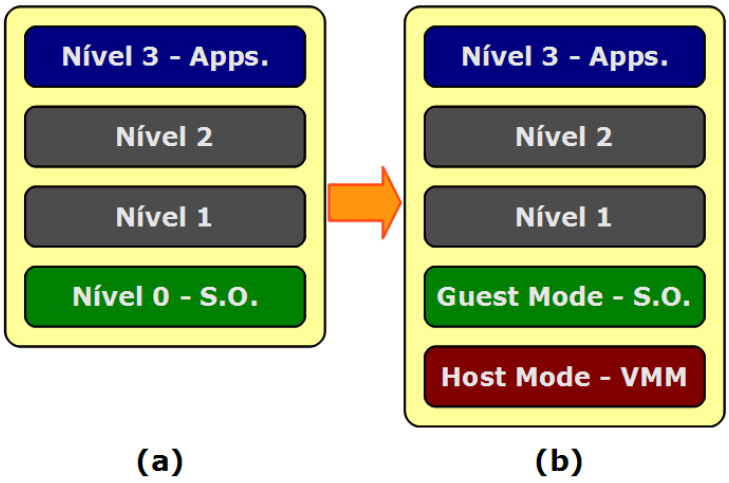

Figura 2.4: Níveis de execução em uma CPU x86 com suporte a virtualização

2.6.3.2 Forma de Operação

Quando a instrução de entrada no modo convidado é executada, o estado do sistema convidado é carregado e a CPU alterna para o modo convidado. Ele permanece nesse modo até que alguma condição pré-definida na estrutura de controle seja alcançada. Nesse ponto, os dados da CPU virtual são descarregados para a estrutura de controle e o fluxo de execução é transferido para o VMM.

O modelo utilizado para essa estrutura de controle permite ao VMM definir quais tipos de operações devem ser executadas pelo convidado e quais devem desviar o fluxo de execução para o VMM. Essa abordagem não oferece imposição sobre a forma de funcionamento do monitor, permitindo, por exemplo, oferecer acesso a dispositivos ou a manipulação de interrupções diretamente pelo sistema convidado, bem como criar um ambiente isolado.

Essa versatilidade permite a implementação de monitores com funcionamentos distintos, como os propostos em [21], que utiliza o suporte em hardware para virtualização da CPU em conjunto com os mecanismos de para-virtualização de memória e dispositivos do projeto Xen, e em [19], que adiciona suporte a execução via hardware como alternativa à tradução binária. Tal flexibilidade também abriu a possibilidade para a criação de *rootkits*, como o Blue Pill [27], que se interpõe entre o SO e o hardware através das instruções de virtualização.

O principal gargalo de desempenho na virtualização assistida por hardware refere-se ao custo e à frequência de entradas e saídas necessárias para a virtualização do sistema convidado, observado em [5] e [19]. O custo depende da implementação do suporte na CPU. A Tabela 2.1, baseada em dados obtidos em [19], apresenta a evolução no suporte a virtualização entre CPUs da geração Pentium 4 e Core 2 Duo. A redução na frequência de entradas e saídas depende da implementação do VMM e da utilização de outras tecnologias em hardware que permitam redução nessa quantidade, como a paginação aninhada de memória, discutida na seção 3.6.

CPU Operação (ciclos)	Pentium 4 3.8GHz	Core 2 Duo 2.66Ghz	Ganho Absoluto	Ganho Normalizado pelo Clock
Entrada no modo convidado	2409	937	157,10%	266,32%
Saída por falta de página	1931	1186	62,82%	131,98%
Leitura do VMCS	178	52	242,31%	387,73%
Gravação do VMCS	171	44	288,64%	453,74%

Tabela 2.1: Ciclos necessários para entrada e saída no modo Host

2.7 Considerações Finais

Nesse capítulo foram apresentadas as principais técnicas empregadas na virtualização do processador na arquitetura x86. Em um primeiro momento eram utilizadas técnicas em software, permitindo utilizar os mesmos sistemas operacionais e aplicativos utilizados em

máquinas físicas. Em um momento seguinte, já conhecendo os principais gargalos da virtualização total, foram propostas algumas modificações aos sistemas convidados a fim de contornar essas limitações. Finalmente, com a crescente adoção de virtualização em várias áreas da computação, os fabricantes de microprocessadores passaram a incluir recursos capazes de amenizar essas limitações sem a necessidade de modificar os sistemas convidados.

Existe uma variedade de tecnologias atualmente, com CPUs sem nenhum suporte a virtualização, modelos que dispõe apenas de suporte a virtualização de instruções e outras que também dispõem de otimizações para o acesso à memória. O suporte em hardware dos dois principais fabricantes, apesar de muito semelhante no formato e funcionamento, não são compatíveis. Todas essas variáveis criaram um ambiente heterogêneo, exibindo desempenho distinto de acordo com os recursos utilizados.

Um aspecto de grande relevância no desempenho de máquinas virtuais refere-se ao acesso à memória. Da mesma forma que a virtualização de CPU, existem diversas abordagens distintas para sua implementação, além de várias técnicas que visam aumentar seu desempenho ou minimizar o consumo de memória, de forma transparente ou com auxílio do sistema convidado. Considerando a importância desse tópico para o projeto desenvolvido, ele será tratado com o detalhamento adequado no capítulo seguinte.

Capítulo 3

Virtualização do subsistema de memória

3.1 Considerações Iniciais

Um fator que oferece grandes desafios na criação de um ambiente virtual é o gerenciamento de memória. Os sistemas operacionais tradicionalmente esperam que o hardware ofereça um espaço de endereçamento linear, iniciado no endereço zero. Em um ambiente virtualizado, com múltiplas máquinas virtuais e o VMM compartilhando o mesmo espaço, isso não é possível. O principal desafio da virtualização do acesso à memória é conseguir emular esse comportamento oferecendo um desempenho aceitável.

A solução mais simples seria reservar um conjunto de páginas contíguas para cada máquina virtual, carregando o *offset* referente ao início do conjunto no registrador utilizado pelo SO convidado para determinar o endereço das páginas. Apesar da simplicidade e efetividade dessa solução, ela traz como limitante o desperdício de recursos, ao reservar inicialmente toda a memória de uma máquina virtual, além de possibilitar um cenário em que existe memória disponível, porém não é possível instanciar uma nova máquina virtual, pelo fato das páginas livres não serem contíguas.

A técnica mais disseminada para o gerenciamento de memória consiste na alocação dinâmica de páginas para as máquinas virtuais, sob demanda. Essa abordagem evita os problemas encontrados na pré-alocação, otimizando o aproveitamento da memória. Com isso, porém,

retorna o problema anterior do sistema convidado esperar um espaço linear de endereçamento. A alocação sob demanda abre oportunidade para técnicas de *memory overcommitment*, em que a soma da memória definida para as VMs em execução é superior à memória disponível no equipamento.

A arquitetura x86 emprega um TLB gerenciado por hardware [3]: faltas no TLB são tratadas diretamente pelo processador, que navega nas estruturas de tabelas de páginas em hardware. Essa característica, adequada a um ambiente desprovido de virtualização, dificulta a virtualização do subsistema de memória, pois exige que o VMM se interponha em algum grau nas operações de gerenciamento de memória, a fim de garantir a consistência e segurança do ambiente como um todo.

Em face disso, existem várias técnicas empregadas na virtualização desse subsistema. Uma delas é a utilização de tabelas de página de sombra (*shadow page tables*), estruturas em memória RAM gerenciadas pelo VMM que realizam o mapeamento entre o endereço da página esperado pelo sistema convidado e o real endereço da página na memória. Outra abordagem consiste em utilizar tabelas de páginas diretas, expondo ao sistema convidado que o espaço de memória não é contíguo, evitando a necessidade de tabelas de mapeamento adicionais. Por fim, algumas CPUs recentes acrescentaram recursos que permitem manter múltiplas tabelas de tradução diretamente em hardware, possibilitando manter o mapeamento de endereços para cada máquina virtual no TLB. Cada abordagem possui qualidades e deméritos, bem como oferece a possibilidade de otimizações distintas. Esses aspectos serão tratados a seguir.

De forma a simplificar o entendimento nas seções seguintes, as páginas de memória referentes à memória virtual no sistema operacional convidado serão chamadas páginas lógicas. As páginas que o sistema convidado enxerga dentro do seu espaço de endereço serão consideradas páginas físicas e as páginas na memória RAM, gerenciadas pelo VMM, serão denominadas páginas de máquina.

3.2 Tabelas de Página e TLB

As tabelas de página atuam no processo de tradução entre endereços virtuais e endereços físicos. A forma empregada na arquitetura x86 utiliza um conjunto hierárquico de tabelas de página de 4 Kilobytes, sendo que cada nível mapeia um espaço mais restrito de memória que o superior.

A navegação nas tabelas de página é realizada em um processo iterativo. O endereço do nível mais elevado da tabela é armazenado no registrador `CR3`. Uma parte do endereço virtual é utilizada para indexar a tabela, obtendo o endereço da tabela do nível subjacente. Esse processo é repetido até alcançar o nível mais baixo, cujo valor será combinado à parte remanescente do endereço virtual a fim de determinar o endereço físico. Na Figura 3.1 é apresentado esse processo utilizando um conjunto de três níveis de tabelas de página.

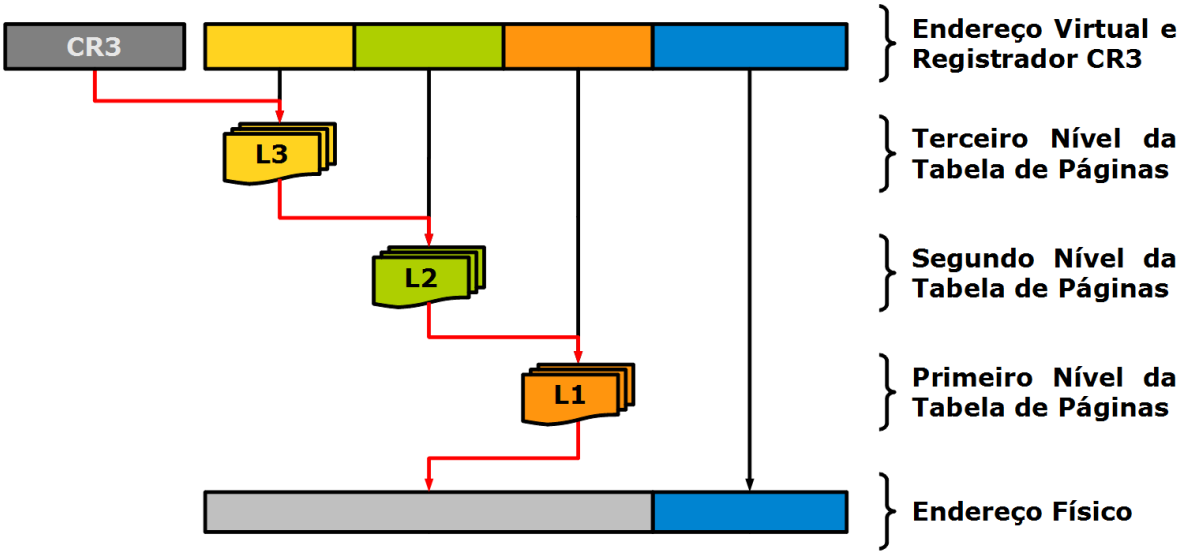

Figura 3.1: Representação do processo de navegação em tabelas de página

O TLB é um dos elementos fundamentais no bom desempenho do subsistema de memória, pois é responsável por acelerar a tradução entre endereços virtuais e físicos. Na arquitetura x86 o TLB é gerenciado pelo hardware, não sendo possível realizar otimizações, como migrar o conteúdo do TLB entre processadores ou realizar uma pré-carga de suas entradas. A única operação possível é invalidar o conteúdo do TLB, seja de maneira explícita, utilizando a instrução `INVLPG`, ou implicitamente, ao carregar um novo conjunto de tabelas de página, atribuindo ao registrador `CR3` o endereço base de outra tabela de páginas.

3.3 Tamanhos de Página

Na arquitetura x86, o tamanho padrão de páginas é 4KB, também denominadas páginas pequenas. A utilização de páginas pequenas evita desperdício de memória devido à fragmentação interna, mas com o crescente aumento no consumo de memória das aplicações, a quantidade de entradas nas tabelas de página vem aumentando rapidamente. Como exposto em [28], a quantidade de entradas no TLB não tiveram o mesmo crescimento, reduzindo de maneira significativa a cobertura do TLB.

O suporte a outros tamanhos de páginas foi incluído na arquitetura x86 através de extensões [29]. O PAE (*Physical Address Extension*) acrescentou suporte a páginas grandes com capacidade de 2MB. As extensões PSE (*Page Size Extension*) e PSE-36 empregam uma outra abordagem que permite a alocação de páginas de 4MB. O PAE vem sendo adotado amplamente pelos sistemas operacionais por questões de segurança [30], pois o suporte ao *No eXecute Bit* (NX Bit) só está disponível no modo PAE nas CPUs x86 de 32 bits.

A arquitetura x86-64 inicialmente oferecia suporte a páginas de 4KB e 2MB, sendo posteriormente adicionado o suporte a páginas que endereçam 1GB cada. Foi necessário adicionar mais um nível de tradução para permitir o aumento na capacidade de endereçamento virtual. A quantidade de níveis de paginação em cada modo pode ser observada na Tabela 3.1.

Modo de Paginação	Tamanho de Página			
	4 KB	2 MB	4 MB	1 GB
x86-PAE	3	2	N/D	N/D
x86-PSE/PSE-36	2	N/D	1	N/D
x86-64	4	3	N/D	2

Tabela 3.1: Quantidade de níveis de tradução das extensões da arquitetura x86

3.4 Tabelas de Páginas de Sombra

Esse mecanismo, descrito em [2] e [3], tem como objetivo criar uma abstração às máquinas virtuais de que elas dispõem de um espaço de endereçamento físico contíguo, através da adição de mais um nível de tradução no acesso à memória. A necessidade de uma indireção adicional a cada acesso à memória traria uma perda de desempenho acentuada, por esse motivo é utilizada uma abordagem mais elaborada.

Para o seu funcionamento, o VMM emprega uma estrutura associada a cada máquina virtual, mapeando o endereço das páginas de memória físicas com os endereços reais das páginas de máquina (Figura 3.2, [31] e [32]). Todas as operações referentes à manipulação das tabelas de páginas do sistema convidado ou do TLB são capturadas, a fim de evitar mudanças no estado da MMU. Baseado nessa estrutura, o VMM mantém em seu espaço de memória as tabelas de página de sombra, com o mapeamento direto entre a página virtual e a página de máquina (Tabela 3.2). A operação de atualização do registrador `CR3` é interceptada, e o valor atribuído passa a ser o da tabela de sombra correspondente à tabela do sistema convidado. Com essa abordagem, referências convencionais à memória não possuem nenhum custo adicional, pois o TLB realizará *cache* do mapeamento direto proveniente das tabelas de páginas de sombra.

Esse modelo possui algumas limitações, pois exige que toda manipulação de tabelas de página seja interceptada a fim de garantir a consistência das tabelas de página de sombra, bem como a propagação da atualização de *flags* de acesso e modificação realizadas pela MMU para a tabela de páginas do sistema convidado. Com isso, há um custo adicional nesse tipo de operação, que em determinadas cargas de trabalho pode causar uma redução de desempenho significativa. Além disso, também é necessário espaço adicional em memória para manter o mapeamento de páginas e as tabelas de páginas de sombra.

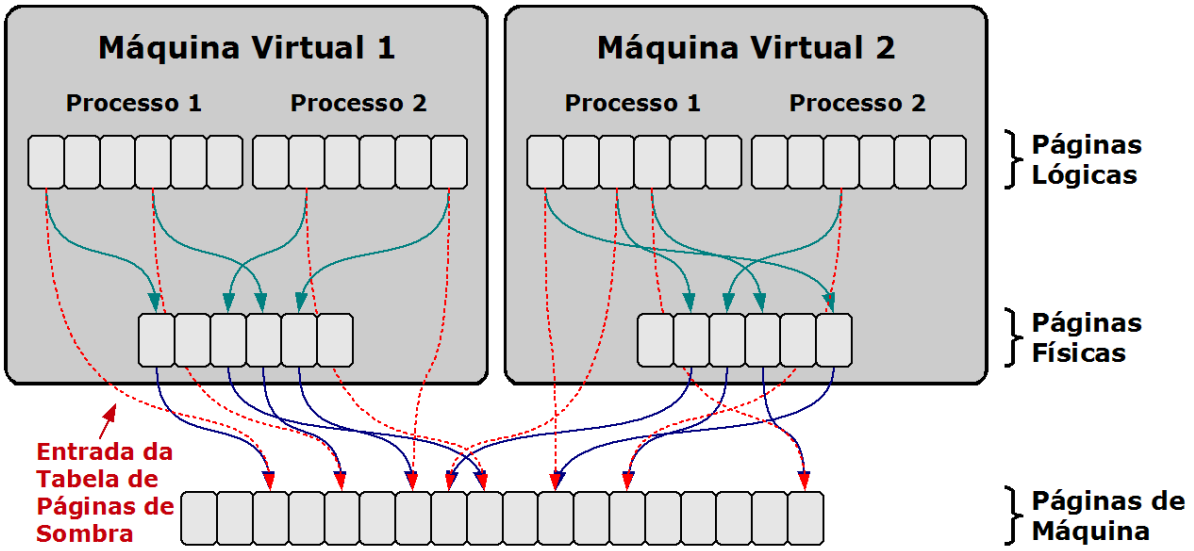

Figura 3.2: Tabela de Páginas de Sombra

Tabela de Página do S.O. Convidado	
Página Lógica	**Página Física**
0x0000	0x13B0
0x00F0	0xEF10
0x01E0	0xE100
0x02D0	0xBF40
0x03C0	0x9C90
0x04B0	0x0B40
0x05A0	0x1860
0x0690	0xCD50

Tabela de Página do VMM	
Página Física	**Página de Máquina**
0x13B0	0x9F60
0xEF10	0x0FF0
0xE100	0xD7A0
0xBF40	0x0870
0x9C90	0xAD70
0x0B40	0x4920
0x1860	0xF000
0xCD50	0x0C30

Tabela de Página de Sombra	
Página Lógica	**Página de Máquina**
0x0000	0x9F60
0x00F0	0x0FF0
0x01E0	0xD7A0
0x02D0	0x0870
0x03C0	0xAD70
0x04B0	0x4920
0x05A0	0xF000
0x0690	0x0C30

Tabela 3.2: Modelo Conceitual de Tabelas de Páginas de Sombra

Apesar desses aspectos, esse nível extra de indireção no acesso à memória oferece a capacidade de remapear páginas sem que o sistema convidado tome conhecimento, manipulando apenas as tabelas de sombra. Com esse recurso, é possível:

- Em sistemas que utilizam PAE para endereçar mais do que 4GB de memória em sistemas de 32 bits, é possível remapear as páginas utilizadas em operações DMA para os primeiros 4GB. Alguns dispositivos, por estarem restritos a um endereçamento de 32 bits, só podem endereçar os 4GB iniciais (memória baixa). A solução típica adotada ao realizar I/O envolvendo memória acima dos 4GB (memória alta) é copiar os dados temporariamente para um *buffer* na memória baixa. Essa operação pode trazer desperdício de recursos. No caso de um ambiente virtual esse problema é agravado, pois páginas que estão na memória baixa de determinada VM podem estar alocadas em páginas da máquina acima desse limite. A solução nesse caso é detectar quais páginas

são utilizadas tipicamente em I/O e remapeá-las para a memória baixa, como exposto em [2].

- Utilizar técnicas sofisticadas para gerenciamento de memória, como compartilhamento de páginas idênticas, desalocação de páginas vazias e paginação. Essas técnicas serão detalhadas na seção 3.7.
- Remapeamento de páginas em ambientes NUMA, a fim de alocar as páginas em bancos de memória que estejam mais próximos dos conjuntos de CPU na qual a máquina virtual está executando. Através desse recurso é possível manter uma boa localidade das páginas de memória, mesmo que o agendador decida migrar a execução da máquina virtual para outras CPUs. Essa operação é implementada no VMware ESX, conforme descrito em [33].

3.5 Tabelas de Páginas Diretas

Nesse modelo, o sistema convidado está ciente de que a memória física não é contígua (Figura 3.3). Por essa razão, não é necessário recorrer a mecanismos de paginação de sombra, sendo que as tabelas de páginas referenciarão diretamente páginas do hospedeiro (Tabela 3.3). Como mecanismo de proteção e atendendo ao critério de segurança de Popek e Goldberg descrito na seção 2.4, o acesso do sistema convidado às tabelas de páginas é apenas para leitura. Alterações nas tabelas de página são solicitadas ao VMM através de uma interface comum, como as *hypercalls* empregadas pelo Xen [3]. Dessa maneira, o VMM pode validar as operações solicitadas, atendendo assim ao critério de segurança.

Uma otimização possível ao empregar essa técnica é agrupar modificações a fim de aplicá-las em lote, diminuindo o *overhead* decorrente das várias chamadas necessárias de outra forma. Como exposto em [3] e [4], isso é benéfico especialmente ao criar espaços de endereçamento.

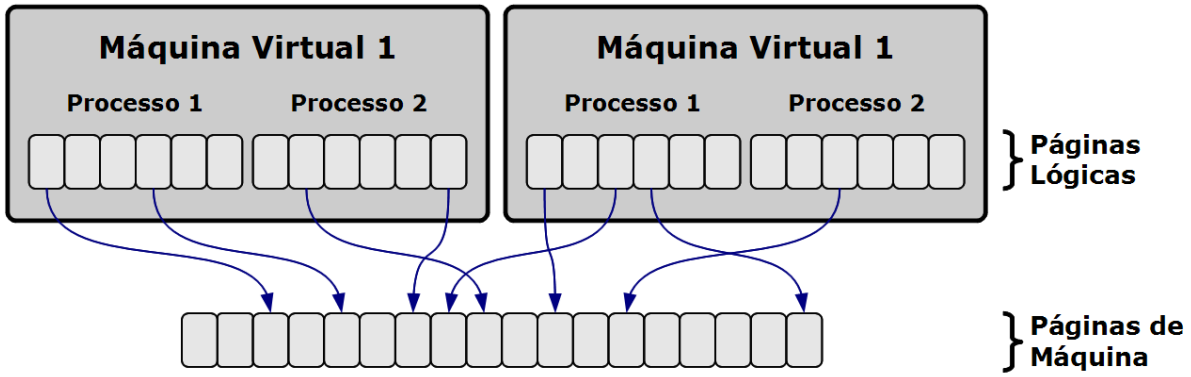

Figura 3.3: Tabela de Páginas Diretas

Tabela de Página Direta	
Página Lógica	**Página de Máquina**
0x0000	0x9F60
0x00F0	0x0FF0
0x01E0	0xD7A0
0x02D0	0x0870
0x03C0	0xAD70
0x04B0	0x4920
0x05A0	0xF000
0x0690	0x0C30

Tabela 3.3: Modelo Conceitual de Tabelas de Páginas Diretas

A grande limitação desse modelo é que para sua implementação é necessário modificar o sistema convidado, sendo considerada a parte mais complexa na implantação de um ambiente para-virtualizado, tanto em termos de mecanismos necessários no VMM quanto na modificação exigidas para portar cada sistema convidado, qustão observada em [3]. Essa característica restringe a compatibilidade àqueles que tenham sido alterados para operar sobre cada VMM em especial.

3.6 Tabelas de Páginas Aninhadas

Tanto AMD quanto Intel adicionaram suporte em hardware à virtualização do subsistema de memória nos processadores mais recentes, sob os nomes de RVI e EPT, respectivamente. Esses mecanismos oferecem operação semelhante, porém são incompatíveis, de maneira análoga ao suporte em hardware a virtualização de CPU dessas empresas (2.6.3).

No modelo tradicional de tabelas de páginas de sombra, o VMM mantém o mapeamento entre os endereços das páginas físicas e das páginas de máquina. O modelo proposto consiste em utilizar esse mapeamento gerenciado pelo VMM diretamente pela MMU na determinação do endereço da página na memória física (Figura 3.4, [31] e [32]). No caso de uma falta no TLB, a MMU percorre as páginas (*page walk)* das tabelas de páginas existentes no sistema convidado combinado às tabelas mantidas pelo VMM, a fim de determinar o mapeamento entre páginas lógicas e páginas físicas. Como visto em [5], uma vez determinado o endereço da página física, a MMU percorre as tabelas de mapeamento construídas pelo VMM, determinando assim o endereço da página de máquina correspondente à página lógica (Tabela 3.4).

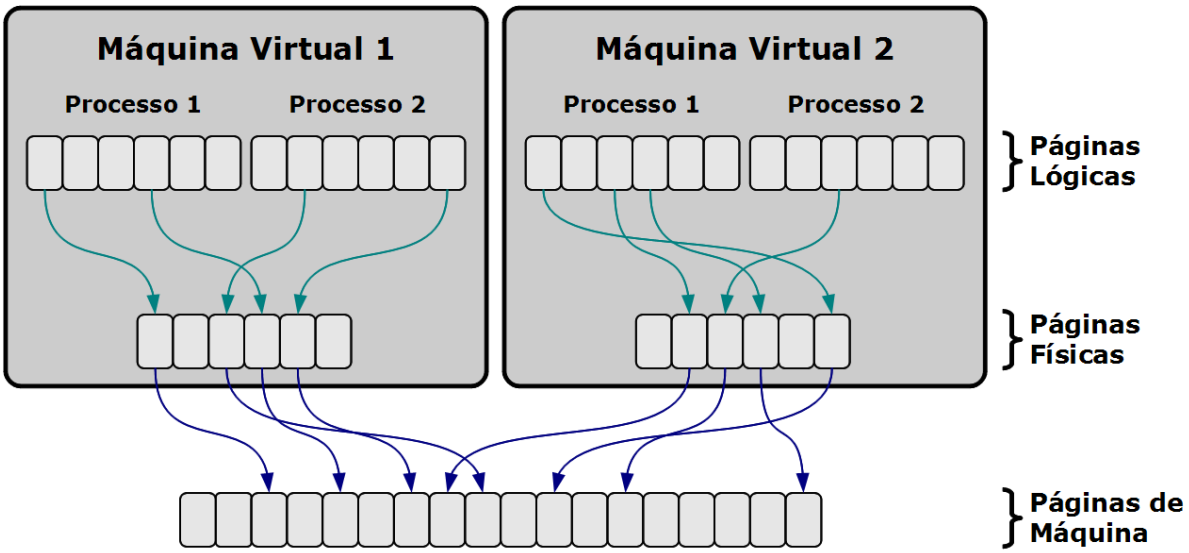

Figura 3.4: Tabela de Páginas Aninhadas

Tabela de Página do S.O. Convidado	
Página Lógica	**Página Física**
0x0000	0x13B0
0x00F0	0xEF10
0x01E0	0xE100
0x02D0	0xBF40
0x03C0	0x9C90
0x04B0	0x0B40
0x05A0	0x1860
0x0690	0xCD50

Tabela de Página do VMM	
Página Física	**Página de Máquina**
0x13B0	0x9F60
0xEF10	0x0FF0
0xE100	0xD7A0
0xBF40	0x0870
0x9C90	0xAD70
0x0B40	0x4920
0x1860	0xF000
0xCD50	0x0C30

Tabela 3.4: Modelo Conceitual de Tabelas de Páginas Aninhadas

Essa técnica oferece o benefício de não necessitar de modificações no sistema convidado para funcionar, sem a necessidade de utilizar tabelas auxiliares ou a interposição nas operações de manipulação de tabelas de página ou trocas de contexto. Como observado em [26], as tabelas aninhadas são prioritariamente estáticas e não necessitam de atualizações quando ocorrem mudanças ou a criação de novas tabelas de páginas por parte do sistema convidado. A paginação aninhada permite reduzir a quantidade de saídas do modo convidado ao ser empregada combinada à virtualização de CPU por hardware, influenciando positivamente em seu desempenho.

Uma limitação dessa técnica reside no fato que as faltas no TLB serão mais custosas, pois agora existe a necessidade de percorrer dois conjuntos de tabelas de páginas em vez de uma, sendo que as tabelas mantidas pelo VMM são percorridas para cada nível da tabela de página do sistema convidado a fim de determinar o endereço de máquina. Determinadas cargas de trabalho são afetadas sensivelmente por este custo adicional.

Uma forma de reduzir esse impacto é a utilização de páginas grandes (*large pages*), passando das típicas páginas de 4KB para páginas de 2MB ou mesmo 1GB, otimização sugerido em [34]. Os principais benefícios nesse caso são a navegação mais rápida nas tabelas de página, por possibilitar que tabelas com menor profundidade sejam capazes de endereçar a mesma quantidade de memória, assim como diminui a necessidade de percorrer as tabelas de páginas. Isso é possível pois mesmo com um menor número de entradas no TLB destinadas a páginas grandes encontrados nas CPUs modernas, como pode ser observado na Tabela 3.5 (baseada em dados obtidos em [34], [35] e [36]), é possível mapear uma quantidade maior de memória, diminuindo dessa maneira a quantidade de faltas no TLB.

Fabricante	**AMD**		**Intel**	
Processador	**Athlon 64**	**Phenom**	**Core 2 Duo**	**Core i7**
Codinome	*Hammer*	*Barcelona*	*Conroe*	*Nehalem*
Ent. L1 TLB 4 KB (Dados/Instruções)	32/32	48/32	256/128	64/(64 por *thread*)
Assoc. L1 TLB 4 KB (D/I)	total/total	total/total	4 vias/4 vias	4 vias/4 vias
Ent. L1 TLB 2 MB (D/I)	8/8	48/16	32/4	32/(7 por *thread*)
Assoc. L1 TLB 2 MB (D/I)	total/total	total/total	4 vias/4 vias	4 vias/4 vias
Ent. L1 TLB 1 GB (D/I)	0/0	48/0	0/0	0/0
Assoc. L1 TLB 1 GB (D/I)	-/-	total/-	-/-	-/-
Ent. L2 TLB 4 KB (D/I)	512 (unificado)	512/512	0/0	512 (unificado)
Assoc. L2 TLB 4 KB (D/I)	4 vias	4 vias/4 vias	-/-	4 vias
Ent. L2 TLB 2 MB (D/I)	0/0	128/0	0/0	0/0
Assoc. L2 TLB 2 MB (D/I)	-/-	2 vias/-	-/-	-/-
Ent. L2 TLB 1GB (D/I)	0/0	16/0	0/0	0/0
Assoc. L2 TLB 1GB (D/I)	-/-	8 vias/-	-/-	-/-

Tabela 3.5: Características dos TLBs de algumas CPUs modernas

Uma avaliação dos ganhos obtidos com a utilização de paginação aninhada pode ser observado em [31] e [32].

3.7 Otimizações no gerenciamento de memória

Uma das principais vantagens de virtualização é oferecer melhor aproveitamento de hardware, através da consolidação. Uma otimização é permitir a alocação de VMs cujo total de memória atribuído é superior à memória gerenciada pelo VMM, técnica conhecida como *memory overcommitment*. A fim de propiciar essa configuração, é possível adotar técnicas de otimização de memória, através de compartilhamento de regiões de memória e balanceamento de recursos gerenciado pelo VMM. Algumas técnicas estão descritas a seguir.

3.7.1 Compartilhamento Transparente de Páginas

Uma forma de reduzir a quantidade de memória alocada é a detecção de páginas idênticas, mantendo apenas uma cópia na memória física (Figura 3.5). Em um ambiente com múltiplas máquinas virtuais, as chances de compartilhamento são maiores que em uma configuração física, aumentando conforme mais VMs utilizam versões idênticas ou semelhantes do Sistema Operacional e aplicativos.

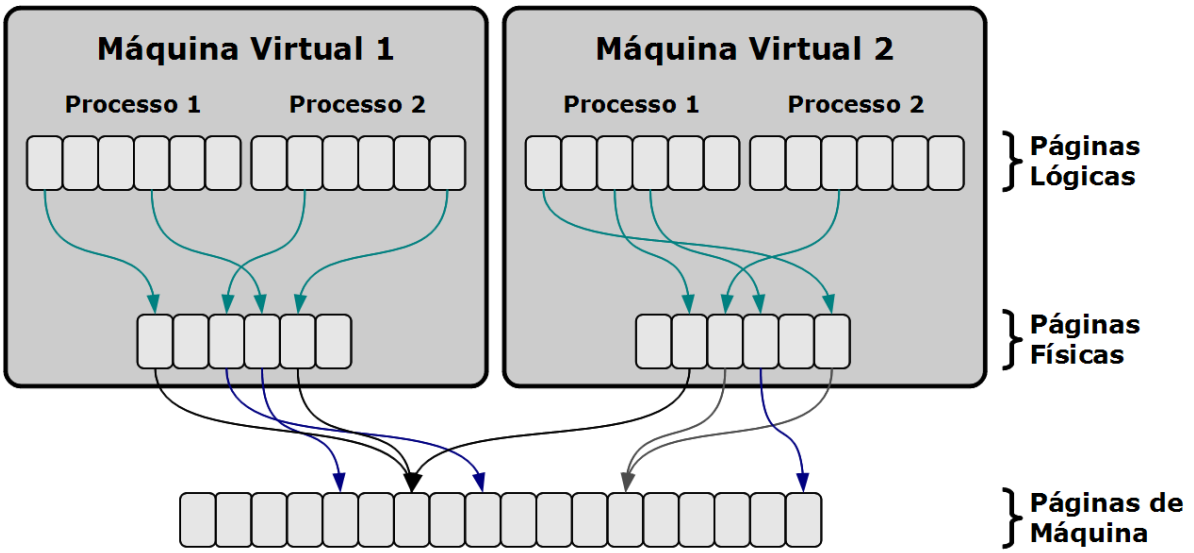

Figura 3.5: Compartilhamento Transparente de Páginas de Memória

A abordagem proposta em [2] consiste em realizar o compartilhamento de maneira transparente aos sistemas convidados. Paginas idênticas são reconhecidas através de seu conteúdo.

Uma comparação completa das páginas seria extremamente custosa em termos de poder de processamento. Uma solução para isso é calcular o *hash* das páginas e armazenar em uma tabela. Páginas que possuam o mesmo valor são fortes candidatas a serem compartilháveis, mas é preciso comparar o conteúdo das páginas de forma a confirmar essa característica. Isso é necessário devido às colisões possíveis ao extrair o *hash*, bem como da possibilidade do conteúdo de uma determinada página ter sido alterado desde que a função foi aplicada sobre a mesma.

Determinado que um conjunto de páginas são idênticas, uma das páginas é marcada como COW (*Copy On Write* ou Copiar ao Escrever), todas as referências são alteradas de forma a apontar para ela, permitindo que as páginas redundantes sejam desalocadas. Tentativas subsequentes de alterar o conteúdo dessa página serão interceptadas pelo VMM, que criará uma cópia privada destinada a quem solicitou a gravação.

Essa abordagem tem como vantagem não necessitar de modificações ou compreender o funcionamento do sistema operacional convidado. Além disso, dessa maneira todas as páginas compartilháveis poderiam ser identificadas através de análise de seu conteúdo.

Algumas políticas podem ser adotadas a fim de aumentar as chances de encontrar páginas idênticas, como priorizar páginas marcadas como somente leitura pelo convidado, páginas que contenham código ou páginas que não foram alteradas recentemente.

Como pontos fracos, pode-se citar o espaço em memória necessário para manter o valor do *hash* das páginas, assim como o consumo de recursos necessários para aplicar a função de escrutínio. Além disso, operações de escrita em páginas compartilhadas são mais custosas, pela necessidade de criar uma cópia da página para consolidar as alterações.

Os resultados obtidos em [2], analisando páginas de maneira aleatória, não apresentaram reduções significativas de desempenho, mas houve uma redução no consumo de memória oscilando entre 7,2% e 32,9%. São observados ganhos mesmo utilizando apenas uma máquina virtual.

3.7.1.1 Kernel Samepage Merging (KSM)

Na versão 2.6.32 do *kernel* Linux foi adicionado um recurso denominado KSM. Projetado para ser utilizado em conjunto com o KVM como solução de compartilhamento de páginas, ele não se restringe a esse cenário, podendo ser utilizado por qualquer aplicativo que possa ter benefícios em utilizá-lo, bastando registrar a área de memória desejada como compartilhável. Páginas compartilhadas são protegidas contra gravação, e a alteração do conteúdo criará uma cópia a quem realizou a gravação.

A implementação inicial do KSM era similar à proposta em [2], realizando o cálculo de um *hash* do conteúdo das páginas para determinar aquelas passíveis de ser combinadas, porém essa abordagem já havia sido patenteada, como pode ser observado em [38] e [39]. Foi necessário utilizar outra abordagem para a determinação de páginas idênticas, e a solução apresentada em [37] consiste na utilização de duas árvores rubro-negras (*red black tree* ou *rbtree*), denominadas estável e instável. As árvores são indexadas pelo próprio conteúdo das páginas. A determinação de páginas idênticas é feita através da comparação completa do conteúdo das páginas, utilizando a função `memcmp()`. A verificação de compartilhamento é restrita às páginas de 4 KB, devido à baixa possibilidade de encontrar páginas de 2 MB cujo conteúdo seja idêntico.

O processo de verificação é iterativo, e a cada iteração todas as páginas marcadas como compartilháveis e que não tenham sido compartilhadas são avaliadas. A árvore estável mantém todas as páginas compartilhadas, e é persistente entre as iterações. A árvore instável é descartada a cada iteração, e mantém as páginas que já foram analisadas naquela iteração.

Para cada página avaliada, inicialmente é verificado se existe uma página idêntica na árvore estável. Se não houver, seu conteúdo é comparado com as páginas armazenadas na árvore instável. Caso não seja encontrada nenhuma página idêntica a ela nesse processo, a página é adicionada à árvore instável. Após todas as páginas serem avaliadas, a árvore instável é descartada e o processo é reiniciado.

A versão disponível na versão 2.6.32 do *kernel* não permitia que as páginas compartilhadas fossem enviadas ao arquivo de paginação (*swap*), recurso que foi acrescentado na versão 2.6.33.

3.7.2 Desalocação de Páginas Vazias (*Page Trimming*)

Um comportamento comum do sistema operacional e aplicativos é alocar páginas de memória sem conteúdo, como forma de reservar memória para uso futuro. Essas páginas vazias oferecem uma grande possibilidade de reaproveitamento.

Uma abordagem consiste em desalocar as páginas de máquina vazias, removendo sua entrada das tabelas de página gerenciadas pelo VMM. Quando uma tentativa de acesso for feita àquela página, é gerada uma falta de página, que é interceptada pelo VMM. Como a página lógica está no espaço de endereçamento do sistema convidado, uma nova página é alocada, utilizando o mesmo método empregado na alocação dinâmica de memória, e o controle é devolvido ao sistema convidado.

Outra abordagem consiste em utilizar o mecanismo de compartilhamento de páginas descrito anteriormente, apontando todas as entradas para páginas vazias para uma única página na memória de máquina. Em testes realizados em [2] foi observado que de 20 a 55% dos ganhos obtidos pelo compartilhamento de páginas foi decorrente do compartilhamento de páginas vazias.

3.7.3 *Memory Ballooning*

Em condições de *memory overcommitment*, é possível que em determinado momento a quantidade de memória necessária pelas VMs supere a quantidade de memória física, mesmo empregando mecanismos de compartilhamento de páginas e remoção de páginas vazias.

Uma alternativa nesse caso é o VMM realizar paginação, enviando páginas para um arquivo de troca. O VMM não tem as melhores informações a respeito do comportamento das páginas para decidir quais devem ser paginadas e quais devem ser mantidas em memória. Isso poderia levar a problemas como paginação dupla, em que o sistema convidado decide enviar para seu arquivo de troca uma página que já está naquele gerenciado pelo VMM. Nesse caso, será gerada uma falta de página, trazendo a página para a RAM apenas para ser salva em disco pelo sistema convidado. Sob esse contexto, a melhor maneira de gerenciar memória é utilizar os mecanismos nativos dos sistemas operacionais para determinar quais páginas devem ser enviadas para disco.

A abordagem descrita em [2] consiste em instalar um *driver* no SO convidado capaz de se comunicar com o VMM. Em um primeiro momento, nenhuma página é reservada (Figura 3.6 **(a)**). Quando é necessário disponibilizar memória, o *driver* é instruído a "inflar o balão" (Figura 3.6 **(b)**), requisitando ao SO convidado, através das interfaces nativas, páginas fixas em sua memória física, cabendo a ele realizar paginação caso não haja páginas físicas disponíveis. De maneira análoga, o balão pode ser esvaziado, liberando novamente as páginas para sistema convidado (Figura 3.6 **(c)**).

O *driver* informa ao VMM quais páginas foram alocadas, de forma que ele possa reclamá-las na memória de máquina. Um problema possível é que as páginas reclamadas poderiam ser acessadas a partir da VM que as cedeu em caso de uma falha no SO, falha no *driver* de *ballooning* ou um reinício não planejado, comprometendo a segurança e isolamento das máquinas virtuais. Como forma de contornar isso, o VMM deve efetivamente remover o mapeamento para essas páginas, de tal forma que uma tentativa de acesso causaria uma falta de página e a alocação de uma nova página pelo VMM.

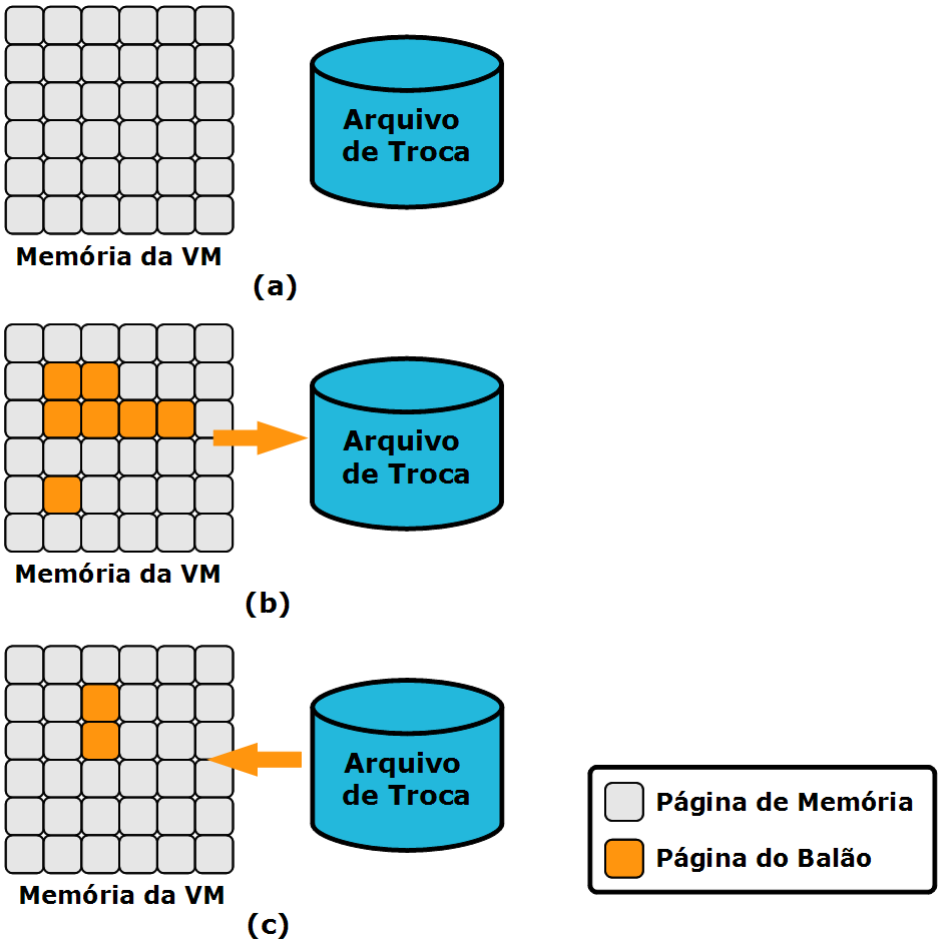

Figura 3.6: Memory Ballooning

3.7.4 Paginação

A paginação consiste no VMM enviar páginas para a memória secundária, tipicamente disco, quando não há memória RAM suficiente para acomodar todas as máquinas virtuais. Como mencionado no tópico sobre *ballooning*, o VMM não possui informações suficientes para realizar a melhor escolha de quais páginas devem ser movidas para disco, o que pode acarretar uma redução significativa de desempenho devido a decisões incorretas na escolha de páginas por parte do VMM. Seu emprego pode ser uma alternativa em sistemas convidados que não dispõem de suporte ao *memory ballooning*, evitando onerar apenas as máquinas virtuais com esse suporte quando houver necessidade de requisitar memória.

3.8 Considerações Finais

A virtualização do acesso à memória é um dos elementos cruciais na virtualização de sistemas, motivo pelo qual foi tratada em um capítulo à parte neste trabalho. Foram apresentadas os principais modelos de virtualização desse subsistema, bem como diversas técnicas de otimização, transparentes ou com a cooperação do sistema operacional convidado, visando otimizar o consumo de memória ou o desempenho.

No próximo capítulo serão discutidas as arquiteturas *multi-core*. Tais arquiteturas vêm sendo amplamente adotadas, e suas características acrescentam novos desafios à virtualização do subsistema de memória. A interação entre a virtualização do acesso à memória e as arquiteturas *multi-core* constitui o assunto do projeto desenvolvido e será discutida em detalhes no Capítulo 5.

Capítulo 4

Arquiteturas Multi-Core

4.1 Considerações Iniciais

As arquiteturas *multi-core* são consideradas uma evolução dos modelos tradicionais de multi-processamento. A melhoria contínua nos processos de produção permitiu o aumento na quantidade de componentes nos processadores, viabilizando a integração de múltiplos núcleos de processamento em uma mesma pastilha de silício.

A tendência em aumentar a quantidade de núcleos em detrimento do aumento absoluto no desempenho por núcleo trouxe novos desafios ao desenvolvimento de aplicativos, exigindo uma maior utilização de processamento paralelo para explorar em sua totalidade processadores com múltiplos núcleos. As técnicas de virtualização oferecem uma alternativa, permitindo consolidar múltiplas tarefas sequenciais em um único equipamento de maneira independente e isolada, permitindo o aproveitamento das arquiteturas *multi-core* mesmo com um conjunto de tarefas que não são paralelizáveis.

Em adição a isto, a integração existente nos processadores *multi-core* abriu a possibilidade para diversas otimizações, especialmente na intercomunicação dos núcleos, através de barramentos de alta velocidade ou no emprego de memória *cache* compartilhada como *buffer* para troca de dados. O aproveitamento desses recursos é de grande relevância no desenvolvimento dos VMMs, que passaram a levar em conta a topologia desses processadores em tarefas como alocação e migração de CPUs virtuais.

4.2 Modelos de Multi-Processamento

Nesta seção serão abordados alguns modelos que viabilizam multi-processamento. Existem modelos que permitem explorar o paralelismo interno dos processadores superescalares apresentando aos aplicativos uma única CPU física como múltiplas CPUs lógicas. Outras técnicas envolvem utilizar múltiplos processadores distintos, variando entre elas a topologia de interconexão. São classificados como *multi-core* aqueles que possuem vários processadores compartilhando a mesma interface mecânica, explorando a localidade para explorar a utilização de estruturas compartilhadas entre núcleos.

4.2.1 Processador Único

Essa é a arquitetura tradicional, em que existe apenas uma CPU ou núcleo de processamento no sistema. Por esse motivo não existe paralelismo explícito, apenas o possível através de uma arquitetura superescalar, por exemplo.

Tipicamente, o acesso à memória se dá através de um barramento de conexão com o *chipset*, onde efetivamente está o controlador de memória, como pode ser visto na Figura 4.1.

4.2.1.1 Controlador de Memória Integrado

A integração do controlador de memória no processador é uma decisão de projeto que oferece benefícios no desempenho (Figura 4.2). Através dela pode-se obter menores latências no acesso à memória, reduzindo o custo de uma falta no cache e aumentando a banda efetiva de memória.

Essa abordagem traz consigo uma limitação. Enquanto a CPU passa a ter um acesso mais eficiente à memória, outros dispositivos são acometidos de uma maior latência para acessá-la, pois passa a ser necessário atravessar mais um nível, no caso a própria CPU. Um caso típico em que essa situação pode ser observada é ao utilizar placas de vídeo de alto desempenho, que podem mover grandes blocos de dados entre sua memória privada e a memória principal do sistema.

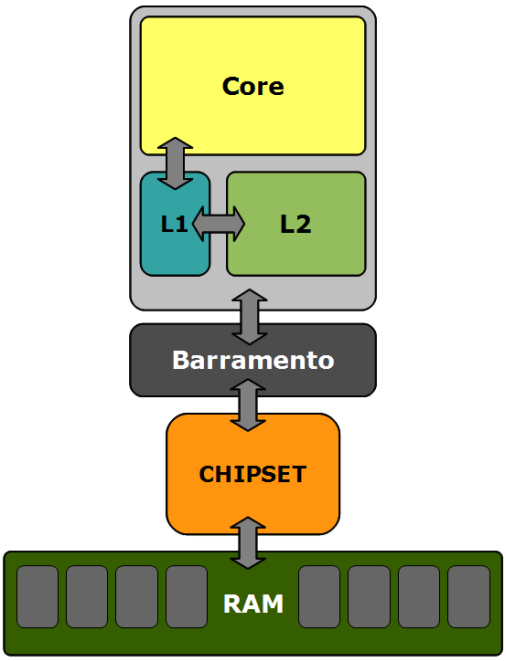

Figura 4.1: Arquitetura com processador único

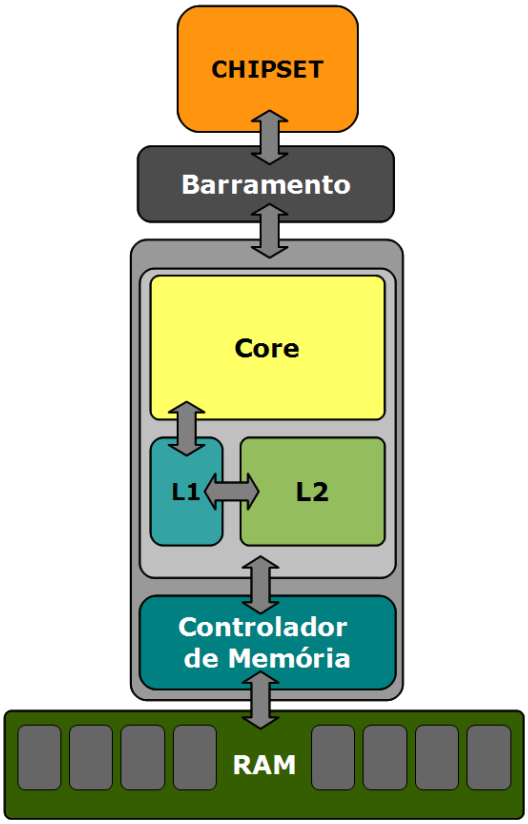

Figura 4.2: Processador único com controladora de memória integrada

4.2.2 *Simultaneous Multi Threading* (SMT)

A fim de obter um melhor aproveitamento das unidades internas de processadores com arquitetura superescalar, nesse modelo implementa-se o suporte a múltiplas *threads* em hardware. Uma única CPU física se expõe ao hardware e aplicações como se fossem duas ou mais CPUs lógicas, dando a ilusão de que se trata de um sistema multiprocessado. Estruturas como ALUs, FPUs e memórias cache são compartilhadas entre as múltiplas CPUs lógicas (Figura 4.3).

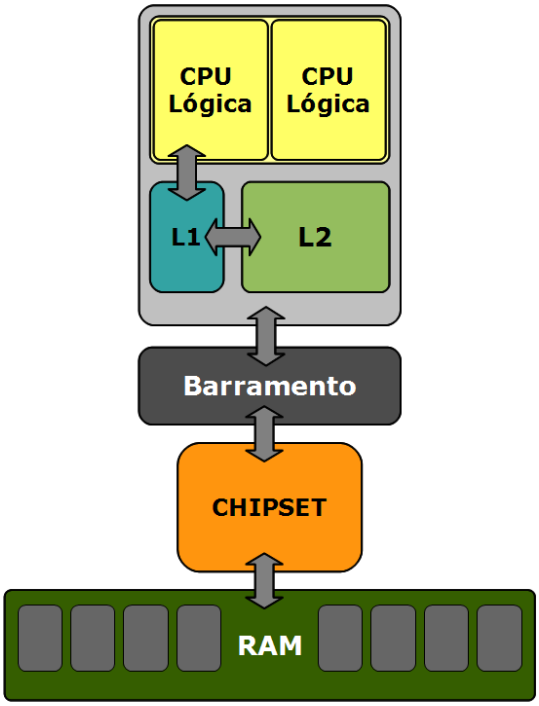

Figura 4.3: Processador com suporte a SMT

A maneira como esse arranjo permite melhor aproveitamento está exemplificada na Figura 4.4. Em **(a)** está representado um processador convencional. Nesse caso, só existe uma *thread* em processamento em uma determinada fatia de tempo do Sistema Operacional. Em **(b)** observa-se um processador que expõe 2 CPUs lógicas, de forma que o SO pode alocar 2 *threads* simultaneamente na CPU. Com isso, o processador pode aproveitar internamente os ciclos que de outra forma ficariam ociosos para realizar o processamento da outra *thread*. No exemplo dado, para realizar o mesmo processamento de 2 *threads* foram necessárias 5 fatias de tempo no caso do sistema convencional e apenas 4 devido ao *multi-threading*.

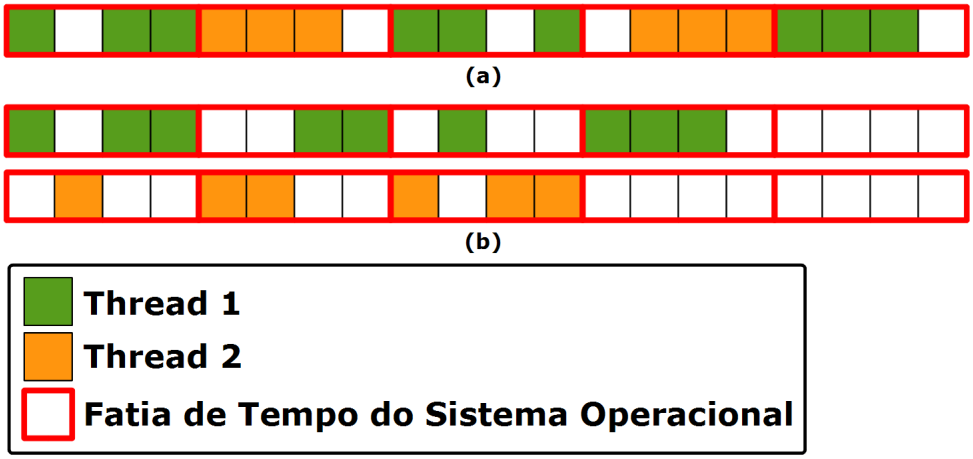

Figura 4.4: Representação do funcionamento do Simultaneous Multi Threading

4.2.3 *Symmetric Multi Processing* (SMP)

Nessa arquitetura existem múltiplos processadores, cada um com suas unidades de processamento e memórias cache. Todos os processadores estão ligados via barramento compartilhado a um chipset, ao qual está conectada a RAM (Figura 4.5).

A migração de dados entre CPUs nesse tipo uma arquitetura se dá através do barramento, geralmente com a CPU de origem salvando os dados na memória para que a CPU de destino possa carregá-los (modelo de coerência MESI). É possível também transmitir dados diretamente entre as CPUs, utilizando o modelo de coerência MOESI, descrito em [40]. Aumentando a quantidade de CPUs consequentemente haverá maior disputa pelo acesso à memória e uso do barramento, bem como tráfego de sincronização, de forma que o barramento unificado torna-se um gargalo para a escalabilidade do conjunto.

4.2.4 NUMA

Na arquitetura NUMA (*Non Uniform Memory Access*), cada CPU ou grupo de CPUs, denominado nó, possui seu próprio banco de memória, podendo acessar os bancos de memória localizados em outros nós, porém a uma velocidade mais baixa. A velocidade de acesso à memória varia de acordo com a distância entre nós, que representa o total de nós que devem ser atravessados para acessar determinada posição de memória. A Figura 4.6 exibe uma arquitetura NUMA com 2 nós, cada um com seu banco de memória e um barramento de comunicação interligando-os.

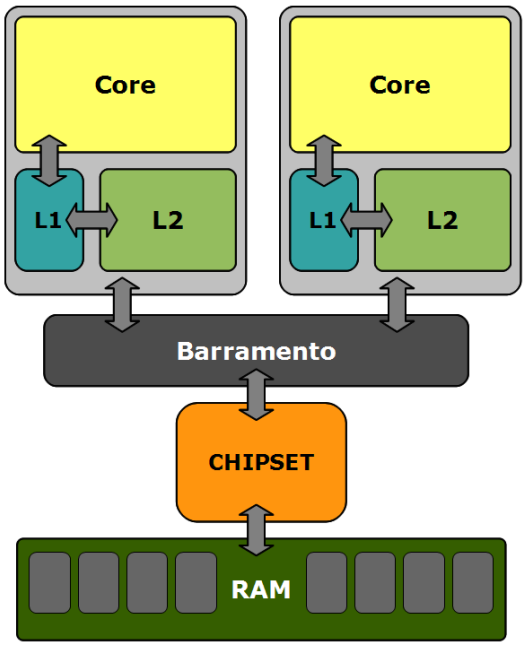

Figura 4.5: Multi-Processamento Simétrico

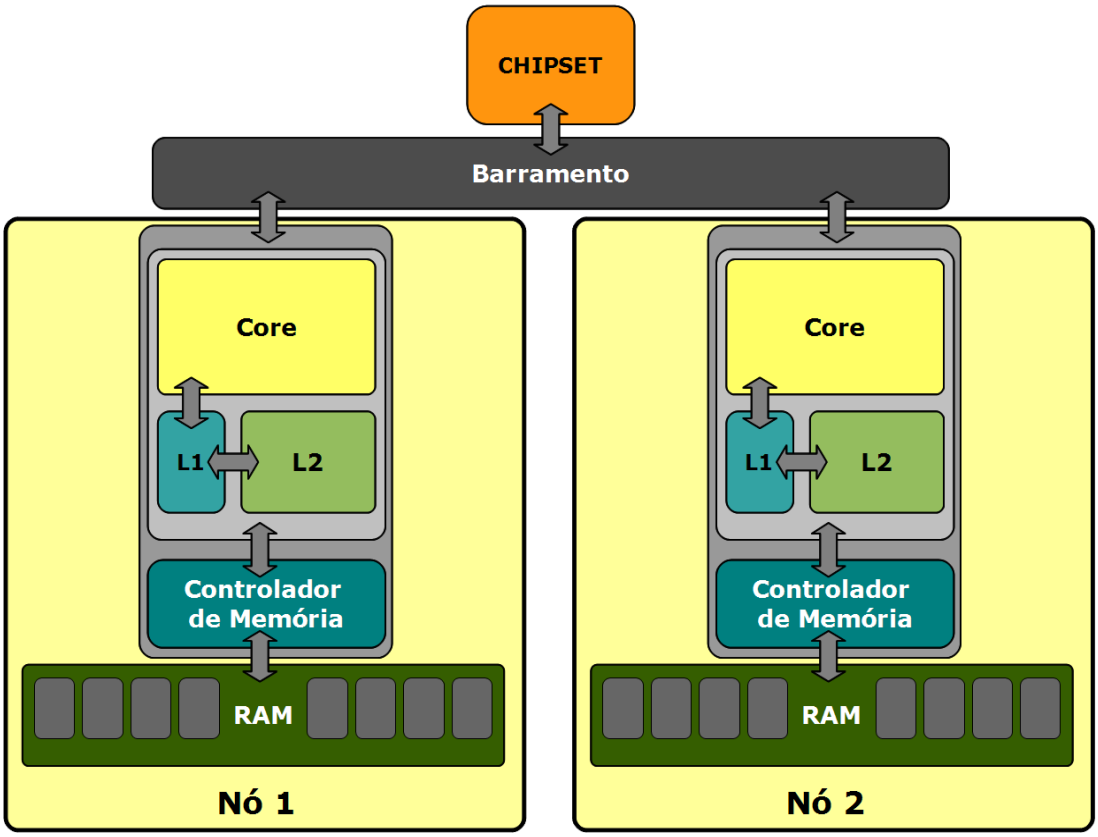

Figura 4.6: Arquitetura NUMA

4.2.5 *Chip Multi Processing* (CMP)

Diferente do SMT, em que uma CPU se apresenta como se fossem várias, no CMP, também conhecido como *multi-core*, existem efetivamente múltiplas CPUs em um único encapsulamento. Essa abordagem oferece algumas vantagens sobre o SMP:

- **Menor custo de produção:** como várias CPUs compartilham o mesmo encapsulamento, é necessário um menor número de interfaces mecânicas para um mesmo número de núcleos de processamento. Isso permite reduzir os custos, necessitando de menos soquetes e consequentemente reduzindo a complexidade da placa-mãe.
- **Barramento dedicado de inter-comunicação:** CPUs dentro do mesmo encapsulamento podem se comunicar por um barramento exclusivo. Como não existem as limitações elétricas de trafegar dados pela placa de circuito impresso, é possível alcançar uma maior velocidade de transferência. Com um barramento dedicado, é possível aliviar o barramento externo da comunicação entre os núcleos.
- **Estruturas compartilhadas entre núcleos:** A fim de otimizar a comunicação e o aproveitamento de recursos, é possível empregar estruturas comuns entre os núcleos. Um exemplo é a utilização de um cache nível dois ou três compartilhado entre os núcleos, permitindo a troca de dados entre núcleos através da memória cache, assim como permitir que um núcleo utilize uma porção maior do cache caso necessário, otimizando dessa forma seu aproveitamento.

O modelo *multi-core* foi a solução encontrada para obter um melhor aproveitamento da crescente quantidade de transistores possíveis em uma mesma área de silício, pois como apresentado em [41], as soluções para extração de paralelismo interno, como aumento de *clock*, execução fora de ordem, execução especulativa e arquitetura superescalar possuem limites a partir dos quais o aumento na complexidade não oferece ganhos significativos de desempenho. Em [42] observa-se que as técnicas empregadas para aumentar o desempenho de um processador monolítico levam a um crescimento significativo de sua complexidade, levando a uma baixa eficiência do pipeline e alto consumo de energia.

Um dos benefícios da utilização de múltiplos núcleos se refere à questão de projeto e produção. Em vez de desenvolver um único núcleo de elevada complexidade, no paradigma

multi-core pode-se desenvolver um núcleo mais simples e replicá-lo no mesmo processador. Da mesma forma, pode-se aproveitar a redução no tamanho dos transistores para aumentar a contagem de núcleos em vez de desenvolver núcleos mais complexos, diminuindo assim o custo total do projeto.

Ao empregar múltiplos núcleos, além do paralelismo interno existente nos processadores modernos, passa a haver também paralelismo explícito. Com isso, os sistemas operacionais e softwares precisam ser otimizados para aproveitar essa característica. Como nem todas as tarefas são paralelizáveis, o ganho de desempenho passa a ser limitado pelas rotinas com comportamento serial, comportamento descrito na Lei de Amdahl ([43]). Aumentar o número de tarefas distintas em execução é uma alternativa para aproveitar todos os núcleos. A utilização de virtualização permite obter melhor aproveitamento de arquiteturas *multi-core* nesse caso, pois as máquinas virtuais são independentes.

4.3 Implementações

Nessa seção serão expostos alguns processadores x86 lançados na última década e que exploram de alguma forma de multi-processamento, seja através de *multi-threading* ou *multi-core*. Também serão apresentados os processadores UltraSPARC T1 e T2, que incluem suporte tanto a *multi-threading* quanto a *multi-core*. O objetivo é analisar as decisões de projeto envolvidas e quais os benefícios que trouxeram para o desempenho geral do sistema.

4.3.1 *HyperTheading*

HyperThreading (HT) [44] é o nome da tecnologia de SMT empregada pela Intel em algumas de suas linhas de processadores. O HT é empregado em três gerações de processadores Intel, com objetivos distintos:

4.3.1.1 Intel Pentium 4 HT

Essa linha engloba alguns modelos Pentium 4 e os Xeon derivados [45]. Os Pentium 4 empregavam um pipeline longo, visando atingir altas frequências de *clock*. Houve 4 grandes revisões nessa linha, com o processo de litografia passando por 180nm, 130nm 90nm e 65nm. Nas duas primeiras gerações era empregado um *pipeline* de 20 estágios, enquanto nas demais este foi alongado para 31 estágios. Estes valores estão muito acima dos processadores

lançados nessa década, que oscilaram entre 10 e 16 estágios. Uma bolha no *pipeline* é mais custosa em uma CPU quanto maior for seu *pipeline*.

A função do HT nesse caso era permitir aproveitar a ociosidade do processador no caso de uma bolha no pipeline para processar tarefas de outra *thread*. Os processadores com esse recurso ativado eram apresentados ao Sistema Operacional como 2 CPUs lógicas, abrindo a possibilidade deste despachar duas tarefas simultaneamente à CPU, diminuindo dessa maneira a ociosidade por falta de instruções para processar.

4.3.1.2 Intel Atom

O Atom [46] é uma CPU projetada desde o princípio para oferecer um consumo mínimo, ainda assim oferecendo um desempenho aceitável para tarefas simples de escritório e compatibilidade com o conjunto de instruções das CPUs x86 existentes no mercado, como observado em [47]. Enquanto CPUs móveis tradicionais possuem consumo médio projetado entre 10 W e 35 W e desktops oscilam entre 45 W e 140 W, o consumo projetado dos Atom oscila entre 0,65 W e 4 W [48]. Em [49] é indicado que apesar de seu desempenho inferior ao da maioria das CPUs contemporâneas, sua relação *performance/watt* possibilita consumir menos energia.

Uma das decisões de projeto do Atom na busca por menor consumo elétrico foi a implementação de processamento *in order* [50], evitando dessa forma consumir energia e transistores com mecanismos de reordenação de instruções. Essa abordagem trouxe como efeito colateral uma maior propensão à ociosidade pois caso uma instrução dependa de um dado que esteja na memória não é possível processar outras instruções até que a dependência seja satisfeita. A função do HT nos Atom é oferecer duas CPUs lógicas de forma que caso uma CPU lógica esteja aguardando um dado da memória para prosseguir seu processamento, a outra CPU lógica pode aproveitar as unidades que ficariam ociosas. Em [51] é apresentado que o suporte a HT em um pipeline *in order* oferece tipicamente um ganho de 36% de desempenho para um aumento de 19% no consumo elétrico.

4.3.1.3 Intel Nehalem

O projeto de codinome Nehalem é a base para os processadores Core i7 e Xeon X5500. Ele apresenta um modelo *quad-core* nativo, e cada núcleo oferece a possibilidade de executar duas *threads* simultâneas, permitindo portanto a execução de 8 tarefas simultaneamente. No caso do Nehalem, a utilização do *HyperThreading* tem como principal objetivo ser capaz de aproveitar as melhorias em sua microarquitetura, que trouxeram maior possibilidade de paralelismo interno, como indicado em [36].

4.3.2 *Dual-Core* com núcleos independentes

Nesse modelo são empregados dois núcleos totalmente independentes (Figura 4.7). Para comunicação entre núcleos é necessário recorrer ao barramento. Por esse motivo, para fins de escalonamento e migração de tarefas, seu comportamento não difere de um arranjo SMP (Figura 4.5) com um núcleo em cada encapsulamento [52]. Essa arquitetura foi empregada pelo Pentium D, da Intel.

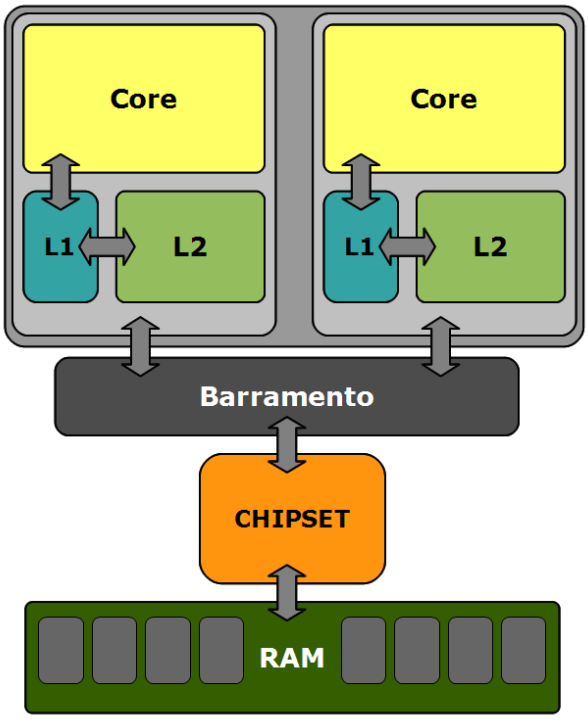

Figura 4.7: Dual-Core com núcleos independentes

4.3.2.1 Pentium D

Esses processadores são basicamente dois núcleos Pentium 4 unidos em um único encapsulamento. Foram lançados modelos fabricados em 90nm (codinome *Smithfield*) e 65nm (codinome *Presler*). A diferença entre eles é que enquanto no *Smithfield* os núcleos empregados deveriam estar adjacentes no *wafer* (Figura 4.8), não havia essa exigência nos *Presler* (Figura 4.9). Essa característica permitiu aumentar a produtividade, reduzindo a taxa de descarte por defeitos e possibilitando combinar núcleos com características semelhantes (como por exemplo a frequência de operação máxima) oriundos de regiões distintas do *wafer*.

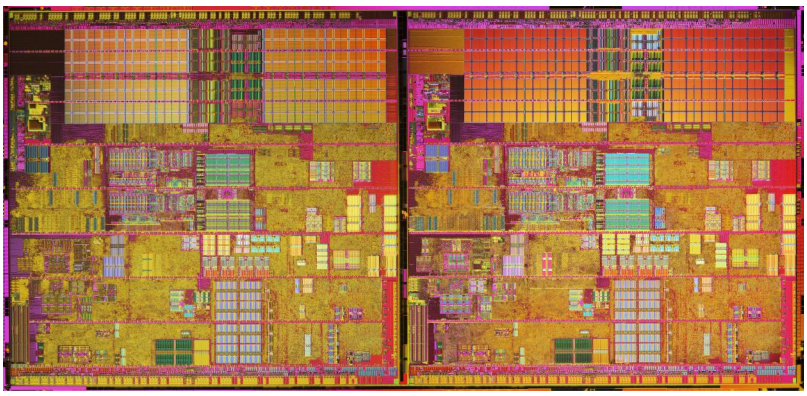

Figura 4.8: Processador Pentium D 90nm (Smithfield)

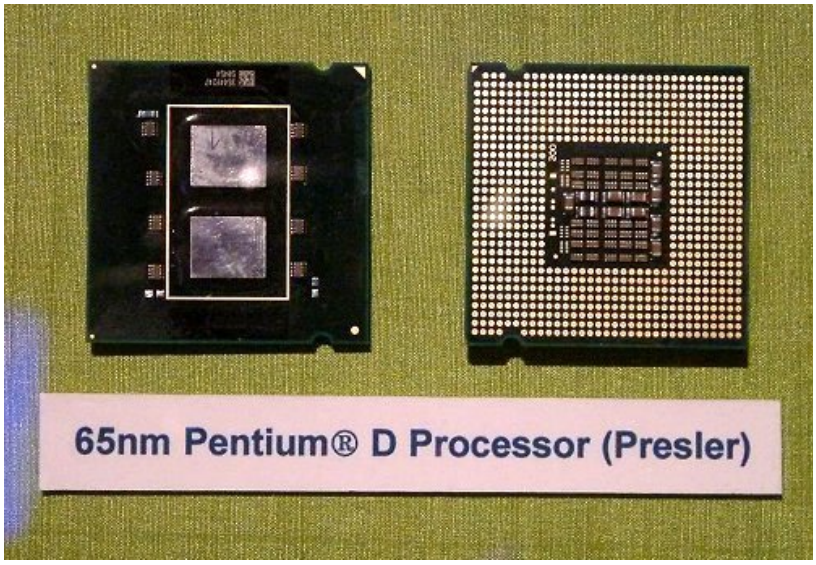

Figura 4.9: Processador Pentium D 65nm (Presler)

4.3.3 *Dual-Core* nativo

Nesse modelo é explorada a capacidade de comunicação entre núcleos. Eles estão dispostos em uma única pastilha de silício, permitindo que troca de dados possa ocorrer a uma velocidade mais alta e sem a necessidade de ocupar o barramento de comunicação principal.

4.3.3.1 Athlon X2

O Athlon X2 [53] emprega o modelo *dual-core* nativo (Figura 4.10) com controladora de memória integrada. Um barramento interno possibilita a comunicação entre núcleos, além de realizar balanceamento de carga no acesso à memória e ao barramento externo. Do ponto de vista de migração de tarefas e compartilhamento de dados entre núcleos, não há diferenças sobre um modelo SMP tradicional, exceto pela maior velocidade possível ao trocar informações diretamente através do barramento interno.

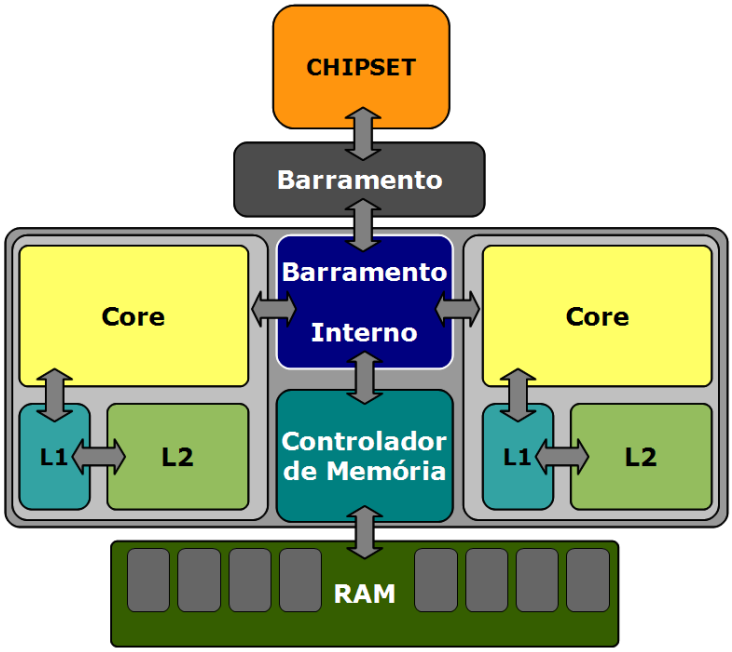

Figura 4.10: Representação do Processador AMD Athlon X2

4.3.4 *Dual-Core* com cache compartilhado

Uma maneira mais avançada de explorar a existência de dois núcleos na mesma pastilha de silício é compartilhar um nível de memória cache entre eles. Isso agiliza tarefas que possuem dados compartilhados e migração de tarefas entre núcleos, que podem aproveitar os dados

armazenados no cache. Além disso, é possível realizar balanceamento de carga, permitindo que um núcleo utilize uma parcela maior do cache caso necessário.

4.3.4.1 Intel Core 2 Duo

Os processadores Core 2 Duo e Xeon derivados são *dual-core* e possuem um cache nível dois compartilhado (Figura 4.11), variando entre 2 e 6MB conforme o modelo [54]. Além do cache, a interface com o barramento externo (BIU – *Bus Interface Unit*) é compartilhada pelos núcleos, como apresentado em [36].

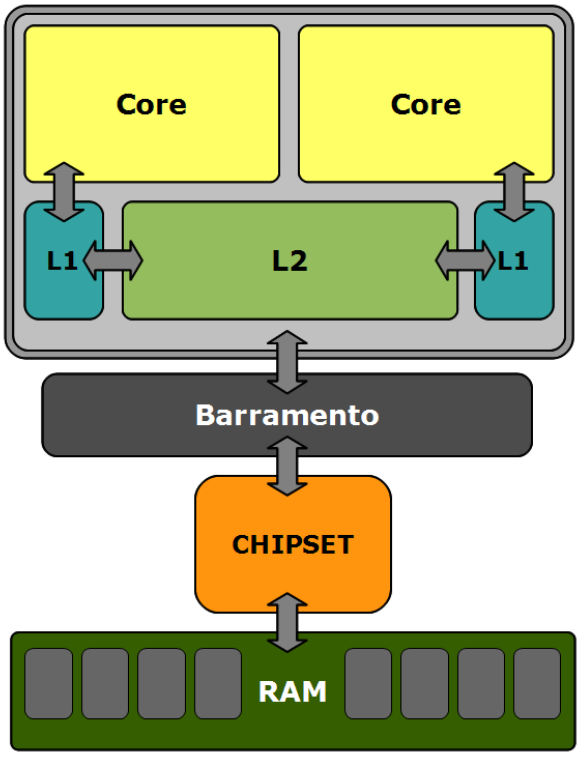

Figura 4.11: Representação do Processador Intel Core 2 Duo

4.3.5 *Quad-Core* nativo com cache compartilhado

Nas arquiteturas *quad-core* nativas baseadas em x86 pode ser observada a existência de um cache nível três compartilhado entre os núcleos, mantendo os caches nível um e dois exclusivos. Com isso, é possível manter um pequeno cache nível dois de baixa latência e alta vazão em cada núcleo em conjunto com um terceiro nível de grandes proporções, capaz de otimizar a troca de dados entre núcleos e ao mesmo tempo diminuir a dependência do barramento de

memória. Existem processadores da AMD e da Intel empregando esse modelo, que serão analisados a seguir.

4.3.5.1 AMD Phenom/Phenom II

Os processadores Phenom e Phenom II (Figura 4.12) empregam um cache dividido em três níveis, cujos tamanhos podem ser observados na Tabela 4.1 (baseada em [36], [55] e [56]), sendo o terceiro nível compartilhado por todos os núcleos. Os níveis um e dois são exclusivos (i.e. um dado pode estar em apenas um nível em determinado instante), e cada nível atua como cache de vítimas (*victim cache*) do nível anterior [34]. No modelo de cache de vítimas, quando uma linha de cache é carregada em um determinado nível da hierarquia, a linha descartada para disponibilizar espaço é enviada para o nível imediatamente abaixo.

No modelo de cache empregado, os dados trazidos da memória principal são carregados diretamente no primeiro nível do cache. Caso um dado buscado por um núcleo seja encontrado no terceiro nível, ele é enviado ao primeiro nível daquele núcleo, e pode ser mantido no terceiro, caso seja determinado que esse dado está sendo utilizado por múltiplos núcleos simultaneamente [34]. O cache nível três é adaptativo, podendo diminuir a latência quando existe uma baixa demanda por vazão, e aumentar a vazão quando necessário. Linhas de cache não compartilhadas tem preferência em caso de descarte do terceiro nível.

Fabricante	**AMD**		**Intel**
Processador	**Phenom**	**Phenom II**	**Core i7**
Codinome	*Barcelona*	*Shangai*	*Nehalem*
Total de Núcleos	4	4	4
L1 Dados	64 KB x4	64 KB x4	32 KB x4
L1 Instruções	64 KB x4	64 KB x4	32 KB x4
L2	512 KB x4	512 KB x4	256 KB x4
L3	2048 KB x1	6144 KB x1	8192 KB x1
Efetivo	4608 KB	8704 KB	8192 KB

Tabela 4.1: Cache dos processadores AMD Phenom/Phenom II e Intel Core i7

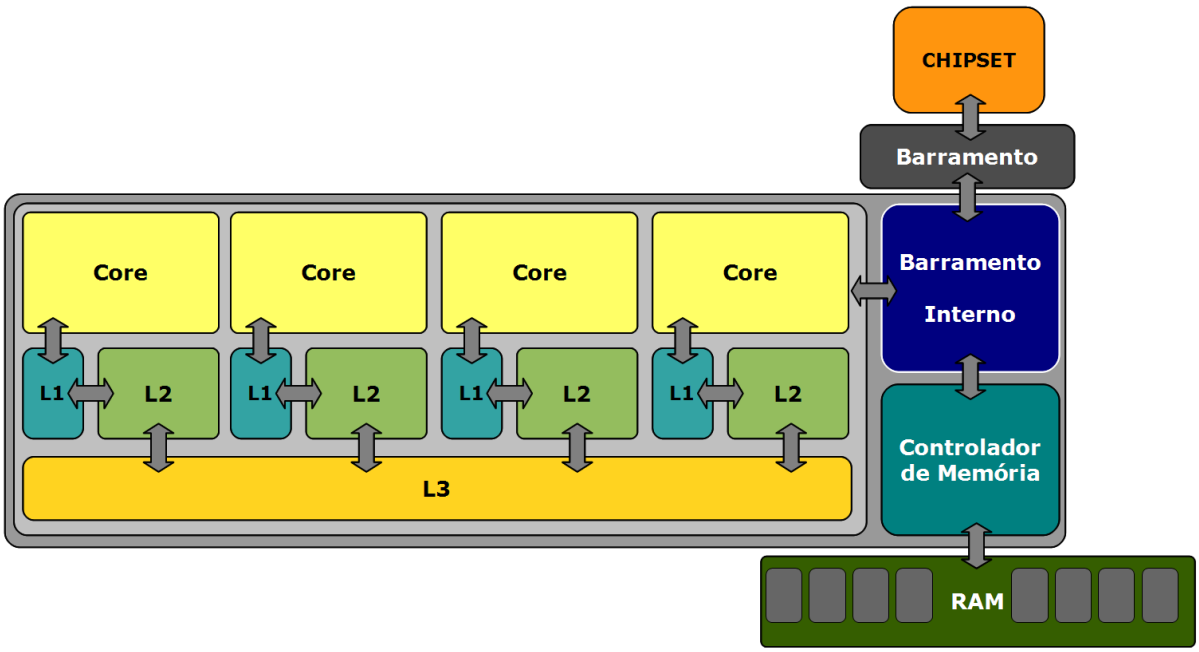

Figura 4.12: Representação dos processadores AMD Phenom/Phenom II

4.3.5.2 Intel Core i7

Os processadores Core i7 (Figura 4.13) também possuem uma organização de três níveis do subsistema de memória cache (vide Tabela 4.1). Os caches nível um e dois são exclusivos, enquanto o cache de nível três é inclusivo, mantendo uma cópia dos dados existentes nos caches privativos de todos os núcleos [36]. Com isso, caso um dado não esteja no terceiro nível de cache, não é necessário consultar os outros núcleos a fim de verificar se algum deles mantém esse dado (*snoop traffic*), reduzindo a latência e aumentando o desempenho geral [57].

Essa abordagem é viável pois a soma do *cache* privativo de todos os núcleos é inferior a 16% do tamanho do *cache* L3, e o consumo será menor quanto maior for a quantidade de dados compartilhados. Tal abordagem não seria adequada na arquitetura da AMD apresentada anteriormente, uma vez que as cópias dos *caches* locais no terceiro nível consumiriam mais do que 40% de sua capacidade.

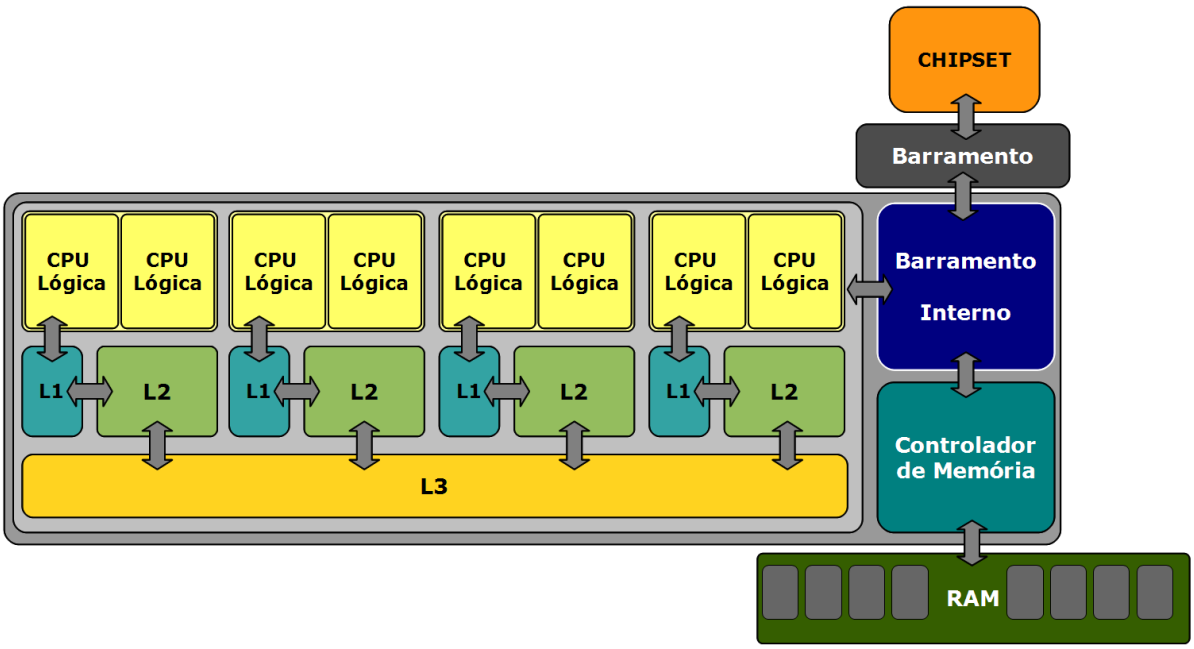

Figura 4.13: Representação do Processador Intel Core i7

4.3.6 Sun UltraSPARC T1/ UltraSPARC T2

Os processadores UltraSPARC T1 e UltraSPARC T2 utilizam a arquitetura SPARC V9, de 64 bits. A arquitetura destes processadores privilegia o paralelismo no nível de *threads* (*Thread Level Parallelism* ou TLP) [58]. Cada núcleo de processamento é capaz de processar quatro *threads* no UltraSPARC T1 e oito no caso do UltraSPARC T2 [42]. Essa grande quantidade permite minimizar a ociosidade das unidades de processamento, pois a execução paralela de múltiplas *threads* permite mascarar a latência no acesso à memória [58]. Uma comparação da quantidade de *threads* simultâneas executadas por esses processadores e outros processadores multi-core pode ser vista na Tabela 4.2.

Fabricante	**AMD**	**Intel**	**Sun**	
Processador	**Phenom II**	**Core i7**	**UltraSPARC T1**	**UltraSPARC T2**
Codinome	*Shangai*	*Nehalem*	*Niagara*	*Niagara 2*
Total de Núcleos	4	4	8	8
Threads/Núcleo	1	2	4	8
Total de Threads	4	8	32	64

Tabela 4.2: Capacidade de execução de múltiplas threads dos processadores analisados

Em ambos processadores, cada núcleo possui um primeiro cache nível 1 privativo, enquanto o segundo nível de cache é compartilhado entre todos os núcleos e dividido em múltiplos bancos (quatro bancos no T1 e oito no T2), permitindo dessa forma o acesso concorrente a regiões distintas. Em ambos os processadores, o acesso à memória principal é feito através de quatro canais de memória independentes.

4.4 *Multi-Core* e Virtualização

Desde as primeiras arquiteturas multi-processadas, os VMMs adotaram mecanismos que permitissem obter um melhor aproveitamento das mesmas. Dois aspectos são relevantes, a migração de máquinas virtuais entre CPUs e a utilização de multi-processamento pelas VMs, também conhecido por virtual SMP (vSMP). De maneira análoga à migração de processos em Sistemas Operacionais, há um custo envolvido para a migração de uma CPU virtual (vCPU) entre processadores, referente ao salvamento de dados alterados nos caches, bem como a necessidade de se recarregar dados relevantes no cache do novo processador. Vários aspectos devem ser avaliados:

- **Migração em processadores com suporte a Multi-Threading:** O VMM deve evitar máquinas virtuais com mais de um processador virtual tenha suas CPUs virtuais alocadas em CPUs lógicas de um mesmo processador. Como as CPUs lógicas compartilham as estruturas internas do processador (ALU, FPU, cache, etc) a disputa por esses recursos reduziria o desempenho. Por outro lado, a migração de uma CPU virtual entre CPUs lógicas do mesmo núcleo é mais eficiente do que migrar entre núcleos distintos, também devido ao compartilhamento de estruturas internas, notadamente memória cache e banco de registradores, eliminando a necessidade de recarregar essas estruturas.
- **Migração entre processadores em uma arquitetura NUMA:** Nessa arquitetura, alguns nós podem estar mais distantes do que outros (maior número de nós intermediários). Essas características apresentam alguns desafios. O custo variável de acesso à memória deve ser levado em conta ao alocar múltiplas vCPUs de uma determinada máquina virtual, visando maximizar a proximidade das vCPUs em relação à principais páginas de memória da VM. A migração de uma vCPU em uma VM mono-processada

deve levar em conta o custo de acessar as páginas de memória que estão alocadas em outro nó a fim de determinar os melhores candidatos. Outra opção, apresentada em [33], consiste em realizar um processo de migração de páginas, movendo as páginas para o mesmo nó que a vCPU .

- **Gerenciamento de Energia:** A migração de vCPUs deve levar em conta aspectos de balanceamento de carga e gerenciamento de energia. Em sistemas com suporte a modos de dormência profunda de CPUs que ofereçam um consumo mínimo, é interessante concentrar as vCPUs sob o conjunto mínimo de CPUs capazes de atender os requisitos de processamento. Por outro lado, em sistemas com processadores que possuem frequência e tensão de núcleo variável sob demanda, distribuir as vCPUs entre todas as CPUs permite minimizar o consumo energético.

Com a introdução de CPUs *multi-core*, novos aspectos passaram a ser considerados:

- **Estruturas compartilhadas entre múltiplos núcleos:** Na maioria das arquiteturas *multi-core* pode-se encontrar estruturas compartilhadas, como controlador de memórias, cache nível dois ou nível três. Essa característica faz com que migrações entre certas CPUs sejam mais custosas do que outras. Por exemplo, em um sistema SMP com 2 CPUs dual core com cache L2 compartilhado, a migração entre cores da mesma CPU é menos custosa, por permitir aproveitar os dados já carregados nos *caches* L2, especialmente se for empregado um modelo de cache inclusivo. Isso é passível de ocorrer mesmo em um único processador *multi-core*. O Core 2 Quad é um processador da Intel que possui quatro núcleos de processamento organizados em dois pares, e cada par possui um segundo nível de cache compartilhado.
- **Virtualização do subsistema de memória:** Uma característica que merece análise é questão da paginação aninhada, que exige o acesso a mais entradas de tabela de página para a tradução entre endereço físicos e de máquina. Nesse caso, a presença de um *cache* compartilhado é interessante tanto pela maior quantidade para manter as entradas adicionais quanto pela capacidade de otimizar a migração de vCPUs entre núcleos aproveitando as entradas mantidas em *cache*.

- **Múltiplos cores em arquitetura NUMA:** Nesse caso em particular, existe a questão do custo extra de acessar a memória localizada em outros nós. Com uma arquitetura NUMA e *multi-core*, passa a existir uma classe de candidatos cujo custo de migração é mínimo, que são os núcleos localizados no mesmo nó. Essa característica deve ser levada em conta no processo de migração. No exemplo exibido na Figura 4.14, o custo para migrar uma vCPU do Core 1 para o Core 2 é menor do que seria para migrar para os Core 3 ou 4, considerando que as páginas de memória utilizadas por essa vCPU foram alocadas prioritariamente no banco de memória do Nó 1.

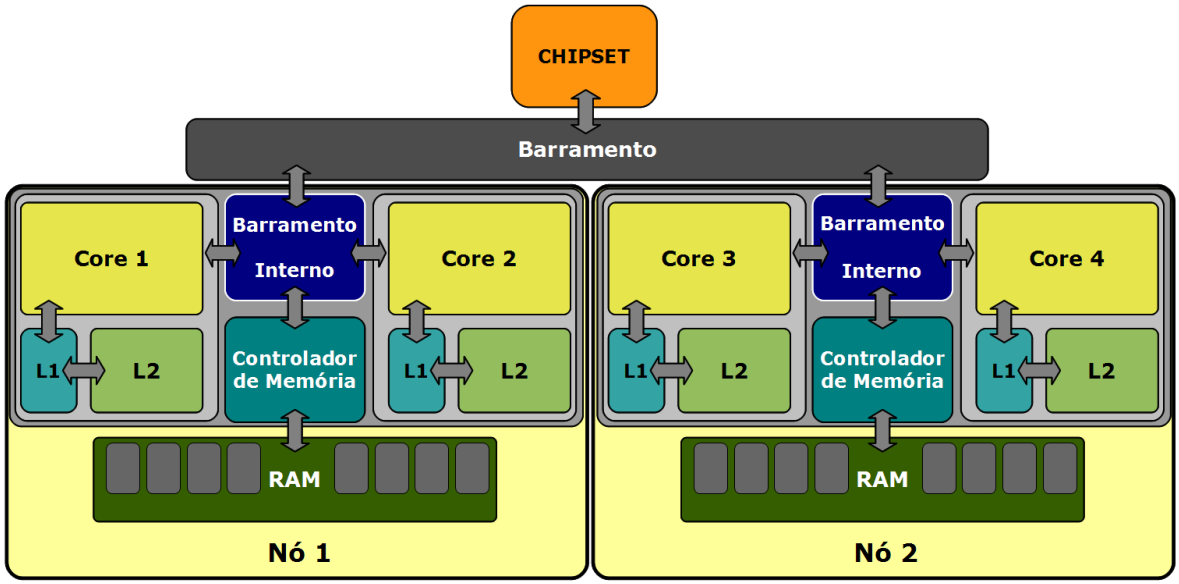

Figura 4.14: NUMA com processadores multi-core

4.5 Considerações Finais

As arquiteturas *multi-core* oferecem benefícios sobre as arquiteturas tradicionais de multi-processamento, possibilitando o emprego de barramentos de alta velocidade para comunicação entre núcleos e níveis de memória cache compartilhados, características desejáveis em tarefas com alto grau de paralelismo e na migração de tarefas entre núcleos.

As ferramentas de virtualização também foram beneficiadas pelos processadores com múltiplos núcleos, mas seu pleno aproveitamento trouxe novos desafios à arquitetura dos VMMs. A migração de máquinas virtuais deve levar em conta a disposição dos núcleos de

processamento, bem como da existência das estruturas compartilhadas, de forma a minimizar o custo dessa operação.

No próximo capítulo será analisada a virtualização do subsistema de memória considerando as características das arquiteturas *multi-core*, com o objetivo de verificar a influência no desempenho devido aos recursos oferecidos por essas arquiteturas.

Capítulo 5

Desenvolvimento do Projeto e Resultados

5.1 Considerações Iniciais

O objetivo do projeto desenvolvido foi estudar as técnicas de virtualização do subsistema de memória avaliando o desempenho nas arquiteturas *multi-core*, tendo em vista a melhor adequação a estas. Embora existam diversos trabalhos analisando esses tópicos de maneira isolada, como em [59], que avalia em profundidade as características de vários modos de paginação, não foi encontrada nenhuma publicação que avaliasse esses dois aspectos de forma conjunta. Em face dessa ausência, considerou-se relevante avaliar essa interação.

Foram comparados os modos de paginação através das tabelas de página de sombra e das tabelas de páginas aninhadas. A tradução de endereços é mais custosa ao empregar paginação aninhada, pois faz-se necessário percorrer as tabelas do sistema convidado e também as gerenciadas pelo VMM. Uma das maneiras de minimizar esse esforço adicional consiste em utilizar páginas grandes para mapear as páginas de memória do sistema convidado, reduzindo a quantidade de níveis a serem percorridos para determinar o endereço físico. A utilização de páginas grandes com esse propósito também foi objeto do projeto desenvolvido.

A avaliação foi realizada através de *benchmarks*, utilizando um VMM com suporte a paginação de sombra e aninhada, bem como utilizar páginas grandes para mapeamento da memória da máquina virtual. Foram empregados processadores *multi-core* com as extensões

necessárias para a utilização de tabelas de páginas aninhadas e mesma microarquitetura interna, diferindo apenas na presença de um terceiro nível de cache compartilhado.

5.2 Relevância do tema escolhido

A virtualização vem rapidamente se tornando uma *commodity* nos *datacenters*, com as principais empresas de diversos setores adotando em seus ambientes computacionais, como bancos comerciais, empresas farmacêuticas, setor energético, linhas áreas, indústria química, área educacional, dentre outros setores. Uma única empresa do setor de virtualização reporta em [60] estar presente em 100% das empresas listadas no Fortune 100 e no Fortune Global 100, 98% das empresas do Fortune 500 e 95% das listadas no Fortune Global 500, além de valores expressivos em outros indicadores.

Em um primeiro momento, as soluções de virtualização foram voltadas para *datacenters* de grande porte, mas com a oferta de soluções gratuitas, como VMware Server [16], VMware ESXi [12], Citrix XenServer [61], Microsoft Hyper-V Server [62], dentre outras, aliado a guias de instalação, dimensionamento e configuração, tendo como exemplo [63], [64] e [65], houve também uma maior adoção em empresas de pequeno porte. Essas empresas não teriam condições de arcar com os custos de licenciamento e contratos de suporte que de outra forma seriam necessários.

A paginação aninhada é uma técnica de virtualização de memória que depende de suporte em hardware para seu funcionamento. A oferta de processadores com suporte é relativamente recente, com as primeiras CPUs com suporte lançadas no final de 2007, pela AMD, sob o nome de RVI [66] e no final de 2008, pela Intel, sob o nome de EPT [67]. Um aspecto importante é que todas as CPUs com suporte a paginação aninhada são *multi-core*, possuindo nativamente quatro ou seis núcleos de processamento, o que pode ser observado em [66], [67], [68], [69] e [70].

5.3 Trabalhos Relacionados

Nessa seção serão apresentados os trabalhos de maior relevância e que contribuíram na definição do trabalho a ser desenvolvido. O primeiro trabalho discorre a respeito da paginação aninhada, apresentando suas características e propondo algumas abordagens para sua otimização. O trabalho a seguir analisa a questão da heterogeneidade dinâmica existente em ambientes *multi-core*, e propõe a utilização de virtualização para um melhor aproveitamento dessas arquiteturas.

5.3.1 Paginação bidimensional para sistemas virtualizados

A paginação aninhada deve atuar de forma transparente para os sistemas operacionais convidados. As entradas das tabelas de páginas mapeiam endereços virtuais para endereços físicos no espaço de endereço do sistema convidado. Para cada nível de navegação nas tabelas de página do sistema convidado é necessário realizar uma navegação nas tabelas aninhadas, a fim de determinar o endereço na memória física da entrada referente ao próximo nível. Esse comportamento se caracteriza como paginação bidimensional. Em [5] é observado que uma paginação convencional para uma tabela de m níveis exige referenciar m entradas nas tabelas de página, enquanto uma paginação aninhada com uma tabela de m níveis no sistema convidado e n níveis na tabela aninhada exige referenciar $mn + m + n$ entradas. Considerando uma tabela de três níveis no sistema convidado e outros três na tabela aninhada, são necessários quinze acessos, cinco vezes a quantidade necessária se não houvesse paginação aninhada envolvida (Figura 5.1).

A forma de amenizar esse aumento proposta em [5] consiste em utilizar um cache de navegação (*Page Walk Cache* ou PWC), que armazena na hierarquia de memória *cache* do processador as entradas das tabelas de página, reduzindo a latência média de navegação. São avaliadas três abordagens:

- **PWC Unidimensional:** são mantidas em cache apenas entradas da tabela de página do sistema convidado, ainda exigindo o acesso à memória para acessar as entradas das tabelas aninhadas.

- **PWC Bidimensional:** tanto as entradas das tabelas do sistema convidado quanto das tabelas aninhadas são mantidas em cache, minimizando o número de acessos à memória em relação ao PWC unidimensional.
- **PWC Bidimensional com traduções aninhadas:** além das entradas das duas tabelas, é utilizada uma segunda estrutura, denominada TLB aninhado (*Nested TLB*), que armazena as traduções realizadas pelas tabelas de página aninhada, permitindo com isso navegar diretamente entre entradas da tabela do sistema convidado em conjunto com as entradas do TLB aninhado. O PWC ainda mantém as entradas da tabela aninhada, empregadas quando não existe a tradução correspondente no TLB aninhado.

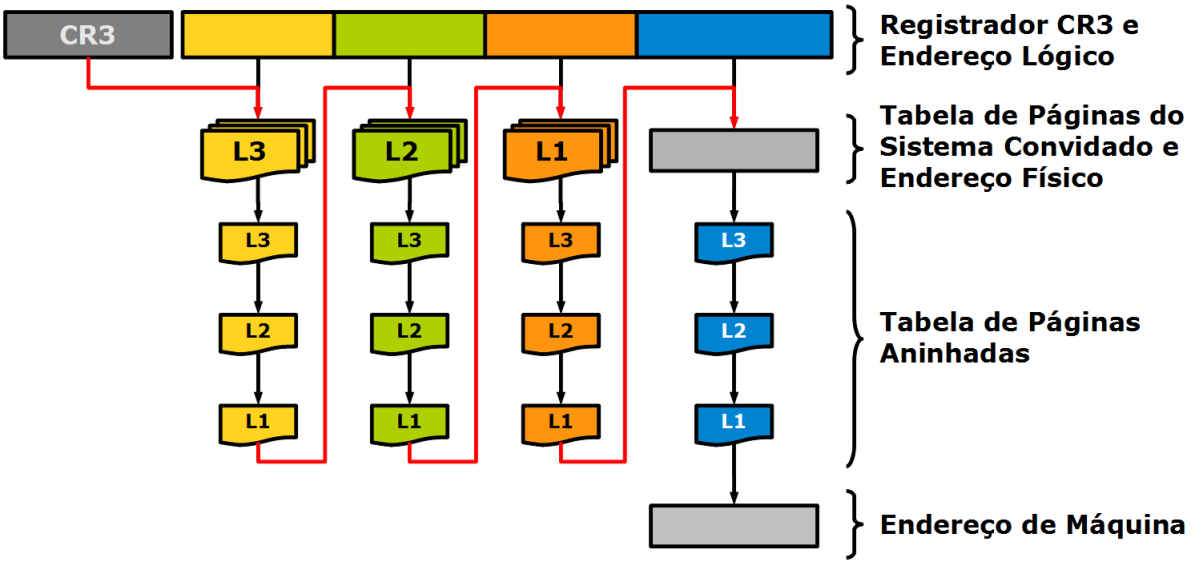

Figura 5.1: Paginação bidimensional

Os resultados obtidos em [5] indicam um ganho de até 46% empregando um TLB aninhado em conjunto com um cache de navegação bidimensional. A utilização de páginas grandes de 2MB por parte do VMM apresentou um aumento de até 22% no desempenho, decorrente da redução no número de entradas necessárias para a tradução.

5.3.2 Heterogeneidade dinâmica em ambiente *multi-core*

Os núcleos das arquiteturas *multi-core* disponíveis atualmente possuem recursos simétricos, não havendo distinção física entre eles. A existência de múltiplas cargas de trabalho em execução simultaneamente cria condições de heterogeneidade dinâmica, em que

fatores como conteúdo do cache, TLB, gerenciamento de energia, dentre outras, acabam por tornar os múltiplos núcleos distintos ao longo do tempo. Essas condições afetam o processo de agendamento de tarefas nos múltiplos núcleos de processamento, pois dificultam a tarefa de determinais quais núcleos estão disponíveis ou são mais adequados para executar uma determinada tarefa [41].

Essa situação é observada na consolidação de servidores através de virtualização, pois cada máquina virtual opera de forma independente das demais. Essa condição acaba por criar assimetria lógica de caches [71].

A proposta em [41] é abstrair os processadores físicos em CPUs virtuais (vCPUs), mas atribuindo ao hardware a responsabilidade de alocar dinamicamente as vCPUs de uma determinada máquina virtual nos núcleos mais adequados. Avanços nas arquiteturas *multi-core* são abstraídos das máquinas virtuais e do próprio VMM.

5.4 Plataforma de Hardware

Para a realização dos testes, foram utilizados os processadores Athlon II X3 440 e Phenom II X3 720 BE (*Black Edition*), que possuem três núcleos de execução independentes. Ambos possuem a mesma microarquitetura interna, a única característica que os diferencia é a presença de um terceiro nível de cache unificado, com a capacidade de 6MB, no Phenom II X3. O funcionamento do cache nível três nos processadores AMD foi detalhado na seção 4.3.5.1. As características dos caches dos processadores está exposta na Tabela 5.1:

CPU	Phenom II X3		Athlon II X3		Associativade
	Núcleo	Total	Núcleo	Total	
L1 Instruções	64 KB	192KB	64 KB	192KB	2 vias
L1 Dados	64 KB	192KB	64 KB	192KB	2 vias
L2	512 KB	1536 KB	512 KB	1536 KB	16 vias
L3	-	6144 KB	-	-	48 vias
Total	640 KB	8064 KB	640 KB	1920 KB	-

Tabela 5.1: Característica dos caches dos processadores utilizados nos experimentos

Para comparação, os dois processadores foram configurados a operar com uma frequência de 3GHz. O cache L3 existente no Phenom II está integrado ao *north bridge*

interno do processador e opera na frequência de 2 GHz. Quanto ao suporte a virtualização, há suporte ao AMD-V (CPU) e também ao RVI, que confere suporte a paginação aninhada.

Esses processadores suportam endereçamento virtual utilizando páginas padrão de 4 KB e páginas grandes de 2 MB e 1 GB. Ao utilizar paginação aninhada, o emprego de páginas de 2 MB pelo VMM com um sistema convidado que utiliza páginas padrão leva a uma redução de 24 para 19 etapas na navegação para determinar o endereço de máquina a partir do endereço virtual, uma redução de 20,84% que também afeta a quantidade de entradas da tabela que precisam ser mantidas em memória e a efetividade da memória *cache* e do TLB.

Os processadores utilizados possuem uma controladora de memória integrada que oferece suporte a dois canais de 64 bits. A controladora pode operar com memórias DDR2 ou DDR3, combinando os canais para formar um único canal de 128 bits (modo *ganged*) ou para operação de maneira independente (modo *unganged*). Para os testes foram utilizados dois pentes de 2 GB DDR2-800, configurado no modo *unganged*, que oferece melhor desempenho em ambientes com múltiplas tarefas simultâneas. A configuração completa do equipamento pode ser observada na Tabela 5.2:

Componente	Especificação
Processador	AMD Athlon II X3 440
	AMD Phenom II X3 720 BE
Placa-Mãe	Asus M3A78-EM
Memória	2 x 2 GB DDR2-800
Disco Rígido	Samsung HD250HJ (250GB, 7200rpm)

Tabela 5.2: Características do equipamento utilizado nos testes

5.5 Plataforma de Software

O ambiente para os testes utilizará uma distribuição GNU/Linux com o *hypervisor* KVM. A especificação desse ambiente será detalhada nas seções a seguir.

5.5.1 Sistema Operacional GNU/Linux

Para a realização dos testes decidiu-se por utilizar uma distribuição GNU/Linux. Os principais motivos para essa escolha foram:

- Estabilidade;
- Alto desempenho;
- Bom aproveitamento do hardware;
- Agendamento de tarefas preparado para *multi-core* e *multi-threading*;
- Grande oferta de ferramentas para extração de informações do sistema;
- Ampla documentação técnica e funcional.

Dentre as principais distribuições, algumas são voltadas a manter-se sempre com as versões mais recentes do *kernel* e outros pacotes que a compõem. Novas versões são lançadas frequentemente, em média a cada seis meses, com um período de manutenção e atualizações de cada versão relativamente baixo, tipicamente inferior a dezoito meses. Nessa categoria pode-se enquadrar Ubuntu, Fedora, OpenSUSE e Mandriva. O principal problema nessa política é que adição de novos recursos só é feita no lançamento de uma nova versão, exigindo uma atualização de versão que provavelmente trará efeitos colaterais consigo.

Outro perfil contempla as distribuições denominadas empresariais (*enterprise*). Nessa categoria o lançamento de novas versões é menos frequente, em geral a cada dois anos ou mais, e o suporte tipicamente ultrapassa os dois anos, podendo chegar a sete anos no caso do RHEL (*Red Hat Enterprise Linux*) e do CentOS (*Community Enterprise Operating System*). Nessas distribuições há uma preocupação muito maior em manter a API e a ABI estável por um longo período, garantindo que softwares homologados não deixem de funcionar ao aplicar uma atualização. Atualizações e inclusão de novos recursos são realizadas através de *back-port*s [72], em que modificações realizadas em uma versão posterior do software são incorporadas na versão oferecida pela distribuição. Dentre os principais exemplos dessa categoria pode-se citar RHEL, CentOS, Debian, Ubuntu LTS (*Long Term Support*) e SLES (*SUSE Linux Enterprise Server*).

Há também as distribuições classificadas como *rolling-release*. Nessas distribuições não existe o conceito de versão, os pacotes que as compõem são atualizados regularmente. Esse modelo garante flexibilidade ao mesmo tempo em que a manutenção torna-se mais complexa, exigindo do administrador garantir a inter-compatibilidade dos pacotes, um trabalho normalmente realizado pelos mantenedores da distribuição. Os principais exemplares dessa categoria são o Gentoo e o Arch Linux, sendo que a principal diferença entre eles é que os pacotes no

Gentoo são prioritariamente disponibilizados na forma de código-fonte para serem compilados na máquina-alvo, enquanto o Arch Linux oferece pacotes pré-compilados para as arquiteturas i686 e x86-64.

5.5.2 KVM – *Kernel Virtual Machine*

O KVM [14] é um subsistema Linux que utiliza extensões em hardware para proporcionar a capacidade de um VMM ao Linux. Desenvolvido inicialmente como um módulo externo, foi incluído ao *kernel* Linux na versão 2.6.20 [73]. Para a criação de máquinas virtuais, o KVM utiliza uma versão modificada do software QEMU [74], que é responsável pela criação das abstrações de dispositivos (unidade de disco, interface de rede, placa de vídeo, etc.) para a máquina virtual.

O KVM é estruturado como um dispositivo de caractere Linux (`/dev/kvm`), acessível no espaço de usuário através de um conjunto de `ioctls`, que são utilizadas para criar e executar as máquinas virtuais. As principais operações oferecidas são:

- Criação de uma máquina virtual;
- Alocação de memória para uma máquina virtual;
- Leitura e escrita de registradores das CPUs virtuais;
- Injeção de interrupções em uma CPU virtual;
- Execução de uma CPU virtual.

Inicialmente havia apenas suporte para as arquiteturas x86 e x86-64, estendido às arquiteturas PowerPC, IA-64 e IBM S/390, voltadas principalmente a computação de grande porte, na versão 2.6.26 do *kernel*. Devido às diferenças entre as instruções de virtualização da Intel e AMD, o KVM é dividido em 2 módulos, um módulo base (`kvm.ko`) e módulos específicos para cada arquitetura, `kvm-intel.ko` e `kvm-amd.ko`, respectivamente.

5.5.2.1 Virtualização de Memória no KVM

Para alocar a memória correspondente ao espaço de endereçamento do sistema convidado, o *kernel* aloca páginas de memória não-contíguas, tal como é a alocação de memória de aplicações convencionais.

O suporte a utilização de páginas grandes para mapear o espaço de endereços do sistema convidado foi adicionado à versão 2.6.26 do *kernel* Linux. Para realizar a alocação o KVM

emprega o sistema de arquivos *hugetlbfs*, no qual as páginas devem ser alocadas antecipadamente e não podem ser paginadas. O gerenciamento exige a utilização de uma API específica, diferente da utilizada para a memória baseada em páginas de tamanho convencional. Essas características limitam sua aplicação de maneira geral e também para virtualização, em razão da necessidade de realizar reserva antecipada e das páginas grandes não serem paginávels.

O KVM utiliza páginas grandes para mapear todo o espaço de endereçamento da máquina virtual, e não apenas no mapeamento de páginas grandes alocadas pelo sistema convidado, utilizando uma única página grande para mapear múltiplas páginas convencionais (Figura 5.2). Da mesma forma, é possível alocar páginas grandes no sistema convidado mesmo quando a memória da máquina virtual é mapeada em páginas convencionais. Nesse caso, as páginas grandes são emuladas por um conjunto de páginas de tamanho convencional (Figura 5.3), não necessariamente contíguas.

A implementação inicial da MMU virtual utilizava paginação de sombra. Seu funcionamento era rudimentar, refletindo as operações de gerenciamento de TLB do sistema convidado nas tabelas de página de sombra. A cada troca de contexto todo o TLB era invalidado, e consequentemente também eram as tabelas de página de sombra, causando impacto significativo no desempenho em ambientes com múltiplos processos [14].

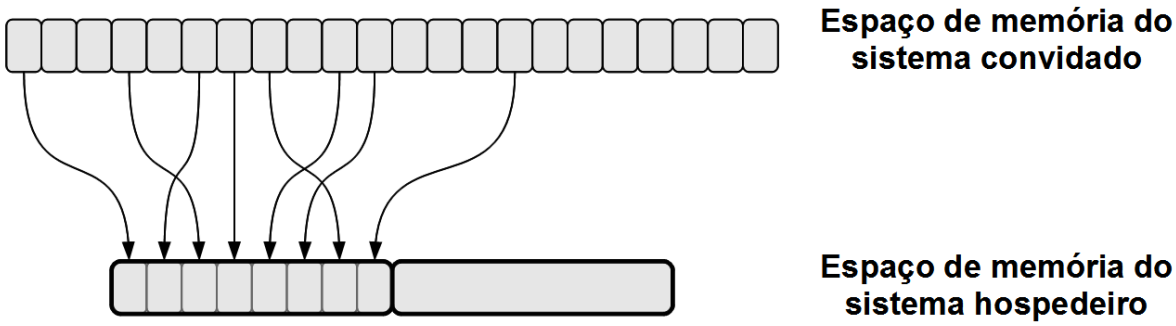

Figura 5.2: Página grande do hospedeiro mapeando múltiplas páginas convencionais do sistema convidado

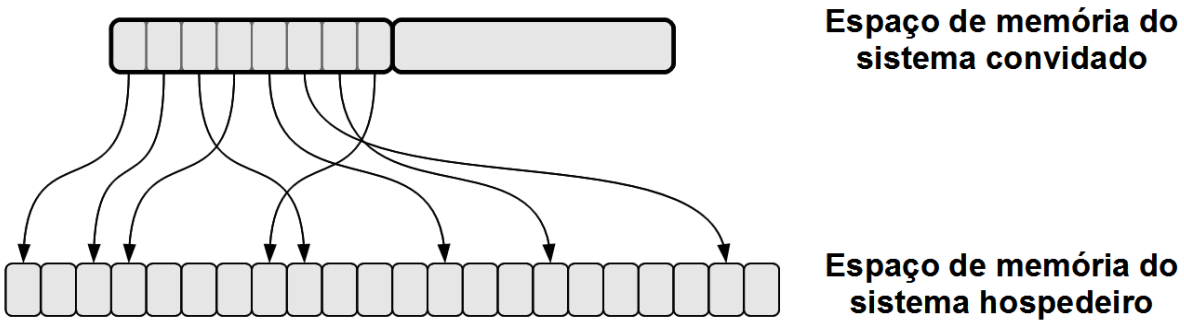

Figura 5.3: Página grande do sistema convidado mapeada por múltiplas páginas convencionais do sistema hospedeiro

Visando otimizar o desempenho da paginação de sombra, a MMU virtual foi modificada de forma a não descartar as tabelas de página de sombra durante a troca de contexto. Como forma de garantir a consistência entre as tabelas de páginas do sistema convidado e as de sombra, as páginas do hospedeiro que possuem uma equivalente na tabela de sombra são protegidas contra modificações. Dessa maneira as alterações são interceptadas pelo KVM, que pode refleti-las na página de sombra correspondente. Como observado em [59], essa mudança aumentou significativamente a complexidade da MMU virtual, em troca de oferecer um desempenho aceitável.

Com a introdução da paginação aninhada nos processadores da AMD e Intel foi adicionado o suporte ao AMD RVI e ao Intel EPT na versão 2.6.26 do *kernel*. Sua implementação trouxe benefícios em cargas de trabalho que tipicamente tem baixo rendimento ao utilizar paginação de sombra, tipicamente aquelas que impõem uma carga intensa sobre a MMU. O KVM utiliza por padrão a paginação aninhada, exigindo um parâmetro adicional (`npt=0`) na carga do módulo específico do fabricante. No caso dos processadores analisados o comando que desativa o suporte ao RVI é o `modprobe kvm-amd npt=0`.

Na versão 2.6.32 do *kernel* Linux foi acrescido o KSM, um recurso que permite o compartilhamento transparente de páginas entre múltiplas máquinas virtuais. Suas características foram descritas na seção 3.7.1.1.

5.5.2.2 Para-Virtualização de Dispositivos

O KVM oferece suporte a para-virtualização de dispositivos através do *framework* VirtIO [75], um padrão para dispositivos para-virtualizados no Linux. Atualmente há suporte para dispositivos de rede, disco e *memory ballooning*. Para a utilização desse *framework* é neces-

sário que o sistema convidado possua *drivers* específicos. O *kernel* Linux inclui suporte nativo a esses dispositivos desde a versão 2.6.25. Também estão disponíveis *drivers* para sistemas convidados Windows. Caso não haja suporte a VirtIO por parte do sistema convidado, é possível utilizar dispositivos emulados que, embora não ofereçam o mesmo desempenho, são suportados pela maioria dos sistemas operacionais.

5.5.3 Configuração de Software do Ambiente

O Sistema Operacional hospedeiro empregado nos testes foi a distribuição Linux Ubuntu Server 10.04 LTS, codinome *Lucid Lynx*, em sua versão 64 bits (amd64). Essa versão utiliza o *kernel* versão 2.6.32. A versão do KVM utilizada foi a fornecida pela distribuição (v.0.12.3).

5.5.3.1 Utilização de páginas grandes

Para habilitar a alocação de páginas de 2MB através do *hugetlbfs* foi necessário incluir um ponto de montagem para esse sistema de arquivos, para que o mesmo fosse montado durante a carga do SO foi incluída a seguinte linha no arquivo `/etc/fstab/`:

```
hugetlbfs           /hugepages        hugetlbfs   defaults    0  0
```

A definição da quantidade de páginas alocadas foi feita no arquivo `/etc/sysctl.conf`, acrescentando a seguinte definição:

```
vm.nr_hugepages = 512
```

Com a definição acima são alocadas 512 páginas de 2 MB, reservando 1 GB de memória para utilização pelo KVM.

5.5.3.2 Paginação aninhada

A ativação e desativação do suporte a paginação aninhada é feita durante a carga do módulo do KVM específico do fabricante, conforme mencionado na seção 5.5.2.1. Nos testes que envolviam a análise da paginação aninhada, antes de iniciar a máquina virtual era executado o *script* a seguir:

```
rmmod kvm-amd
rmmod kvm
modprobe kvm-amd npt=1
```

As duas primeiras linhas removem o módulo comum e o módulo para processadores AMD, enquanto a terceira linha realiza a carga do módulo específico com a paginação aninhada ativada, sendo que o módulo comum é carregado automaticamente por ser dependência do módulo específico. O parâmetro `npt=1` é opcional, uma vez que por padrão a paginação aninhada é ativada sempre que disponível no processador, porém foi mantido por questões de documentação, explicitando a configuração escolhida.

Para a realização dos testes envolvendo tabelas de página de sombra era desativado o suporte à paginação aninhada, mediante execução do seguinte *script*:

```
rmmod kvm-amd
rmmod kvm
modprobe kvm-amd npt=0
```

A única diferença em relação ao *script* anterior se refere à carga do módulo do fabricante, feita com a opção `npt=0` a fim de desativar o suporte à paginação aninhada, restando ao KVM utilizar a paginação de sombra.

5.5.3.3 Máquina Virtual

A máquina virtual foi configurada com 1 GB de memória RAM e um disco virtual de 8 GB, no qual foi instalada a mesma distribuição Linux utilizada no hospedeiro. Para sua instalação foi criada uma partição, no formato ext4, ocupando todo o espaço disponível. O disco virtual foi configurado de tal forma que o sistema hospedeiro não realizasse cache de disco do mesmo, o que poderia afetar nos resultados aferidos. Pelo mesmo motivo, a máquina virtual não possui um dispositivo de *swap* e o suporte a *memory balloning* também foi desativado. Visando reduzir o *overhead* decorrente da emulação de dispositivos, o disco e a rede virtual foram configurados para utilizar o *framework* VirtIO. Para iniciar a máquina virtual era utilizado o seguinte comando:

```
kvm -enable-kvm -m 1G [-mem-path /hugepages] [-smp N] -drive
file=/root/VMs/Lucid.img,if=virtio,cache=none,boot=on -k pt-br -name
'Ubuntu 10.04 LTS' -vga std -vnc :0 -balloon none -net
nic,model=virtio -net tap -daemonize -no-kvm-irqchip
```

O parâmetro `-m 1G` indica a quantidade de memória que será utilizada pela VM, no caso 1 GB. Nos testes em que eram utilizadas páginas de 2 MB para mapeamento da memória o

parâmetro `-mem-path /hugepages` era adicionado à linha de comando, uma vez que ele indica ao KVM o caminho onde está o montado o sistema de arquivos *hugetlbfs*. O parâmetro `-smp N` foi utilizado quando a máquina utilizava mais de um processador virtual, com N valendo 2 ou 3. Os demais parâmetros eram constantes entre as diferentes execuções da máquina virtual, e serviam para indicar a configuração de disco, teclado, interface de vídeo, modo de *balloning* e interface de rede, de acordo com as características descritas anteriormente.

5.6 Metodologia

Para a obtenção dos resultados, cada teste foi repetido dez vezes, sendo que os resultados apresentados são a média do tempo de execução total, em segundos, de cada uma dessas iterações. Antes de cada conjunto de execuções foram feitas duas execuções do teste a ser aferido, com o objetivo de realizar o aquecimento (*warm up*) do cache de disco do sistema operacional convidado, já que como descrito na seção 5.5.3.3, o sistema hospedeiro foi configurado de forma a não realizar cache do disco virtual. Os tempos das execuções de aquecimento não foram computados para a obtenção do resultado final.

Os testes foram realizados com a máquina virtual configurada para utilizar um, dois e três processadores virtuais, com a memória mapeada pelo KVM tanto por páginas de 4KB quanto por páginas de 2MB, sendo que estas foram alocadas através do *hugetlbfs*. Todos os testes foram realizados nos dois processadores já mencionados. Para efeito de comparação, os testes também foram realizados no sistema hospedeiro, e seus resultados serão referidos como nativos no restante desse capítulo.

Todas as operações foram realizadas no disco virtual em vez de empregar um *RAM Disk*, uma abordagem tradicionalmente utilizada para eliminar a influência do subsistema de disco dos resultados [59]. Como o subsistema de memória está sendo avaliado, a utilização de um disco virtual armazenado na memória principal poderia influenciar os resultados obtidos.

5.6.1 Testes Realizados

Foram realizados dois tipos de testes. O primeiro consistiu em realizar a compilação e construção do servidor web Apache `httpd`, utilizando a ferramenta `make` e o compilador GCC 4.4.3, medindo o tempo total, para o processo ser completado. As características dessa tarefa causam uma atividade intensa da MMU, pois, como observado em [32], um grande número de processos de vida curta são criados, um para cada arquivo a ser compilado. Esse teste representa a categoria de aplicações compostas por múltiplas tarefas de curta duração, que não se restringe apenas ao processo de compilação mas também a cargas de trabalho tipicamente encontradas em servidores e ouros ambientes com grande quantidade de acesso concorrente, em que cada requisição dispara a criação de um novo processo ou *thread* para atendê-la. Ao iniciar o processo de compilação pode-se definir o total de tarefas simultâneas que podem estar em execução simultaneamente. Foram utilizados os seguintes valores de acordo com a quantidade de núcleos alocados para a máquina virtual:

Núcleos	Comando
1	make -j1
2	make -j2
3	make -j3

Tabela 5.3: Comandos utilizados no teste de compilação do Apache httpd

O segundo teste realizado consistiu em aferir o tempo necessário, para comprimir o arquivo TAR do kernel Linux versão 2.6.33 (*linux-2.6.33.2.tar*), que possui 377 MB, utilizando o software Parallel Bzip2 (*pbzip2*) [76] na versão 1.0.5, configurado para utilizar blocos de 900 KB. O pbzip2 é uma implementação do compressor bzip2 com suporte a multiprocessamento, que apresenta um *speedup* quase linear em equipamentos com múltiplos processadores [77]. Cada bloco de compressão é gerenciado de maneira independente, sendo comprimido por um único processador. O algoritmo utilizado exige sincronização apenas para a leitura do arquivo de entrada e a gravação do arquivo de saída, de forma a garantir que os dados comprimidos serão salvos na ordem correta. A classe de aplicações representada por esse teste são aquelas compostas por tarefas paralelas de longa duração que possuem baixo nível de intercomunicação. Algumas das tarefas que estão nessa categoria são processos de renderização, conversão de vídeo e operações de encriptação, que podem dividir o fluxo de entrada em blocos e processá-los de maneira independente sem prejuízos ao resultado final.

5.7 Apresentação dos resultados

Nesta seção serão apresentadas as observações mais relevantes baseadas nos resultados obtidos. Uma vez que cada teste possui características distintas, serão explorados diferentes aspectos de cada um deles. A seguinte terminologia será utilizada nas tabelas e gráficos apresentados a seguir:

Código	Significado
NPT	Paginação Aninhada (*Nested Page Tables*)
SPT	Paginação de Sombra (*Shadow Page Tables*)
Nativo	Execução Nativa (sem virtualização)
4 KB	Memória mapeada em páginas de 4 KB
2 MB	Memória mapeada em páginas de 2MB

Tabela 5.4: Terminologia utilizada na apresentação dos resultados

5.7.1 Compilação do Apache

Nos testes de compilação do servidor Apache, o melhor resultado foi obtido ao utilizar paginação aninhada mapeada por páginas de 2 MB no sistema hospedeiro, como pode ser visto na Tabela 5.5. Os resultados da execução nativa estão dispostos na Tabela 5.6 para comparação:

Modo de Paginação	CPU	Páginas de 2 MB			Páginas de 4 KB		
		1 CPU	3 CPU	Speedup	1 CPU	3 CPU	Speedup
Paginação Aninhada	**Phenom II**	79,09	38,07	2,08	89,10	42,40	2,10
	Athlon II	96,21	48,12	2,00	108,85	53,90	2,02
Paginação de Sombra	**Phenom II**	123,62	67,15	1,84	127,98	69,67	1,84
	Athlon II	155,45	84,07	1,85	158,48	85,83	1,85

Tabela 5.5: Resultado obtidos dos experimentos de compilação do Apache na máquina virtual

CPU	Nativo		
	1 CPU	3 CPU	Speedup
Phenom II	76,94	36,34	2,12
Athlon II	93,92	45,36	2,07

Tabela 5.6: Resultado obtidos dos experimentos de compilação do Apache no sistema hospedeiro

A Tabela 5.7 apresenta a melhoria obtida ao utilizar paginação aninhada ao invés da paginação de sombra ultrapassa os 42% em ambos os processadores. Conforme mencionado na seção 5.6.1, uma das principais características desse teste é aplicar uma grande carga sobre a

MMU, o que no caso da paginação de sombra se traduz numa maior quantidade de saídas do modo convidado para a criação e manutenção das tabelas de página de sombra por parte do VMM.

Também podemos observar dos resultados da Tabela 5.7 que o benefício do terceiro nível de cache ao utilizar virtualização é superior ao observado na execução nativa. Embora em valores absolutos o benefício seja pequeno, é um indicativo de que um nível de cache compartilhado permite reduzir o esforço adicional na tradução de endereços em situações de intensa atividade da MMU.

CPU	Páginas de 2 MB		Ganho NPT/SPT	Nativo
	NPT	SPT		
Phenom II	38,07	67,15	43,31%	36,34
Athlon II	48,12	84,07	42,76%	45,36
Ganho Cache %	20,89%	20,13%	-	19,89%

Tabela 5.7: Comparativo de desempenho ao utilizar paginação aninhada e de sombra

A utilização de páginas de 2 MB no sistema hospedeiro apresentou um ganho notável, acima dos 10% ao utilizar paginação aninhada, decorrente da redução no número de travessias de tabelas de página, enquanto ao utilizar paginação de sombra o benefício foi inferior a 4%, como pode ser observado na Tabela 5.8. Não era esperado um ganho significativo no caso da paginação de sombra, pois uma página de 2 MB do sistema hospedeiro será dividida em 512 páginas de 4 KB para serem mapeadas pelas tabelas de página de sombra. O benefício nesse caso se deve a melhor localidade de memória, causado pela alocação de blocos de 2 MB pelo sistema hospedeiro.

CPU	Pag. Aninhada		Ganho 2 MB / 4 KB	Pag. Sombra		Ganho 2 MB / 4 KB
	2 MB	4 KB		2 MB	4 KB	
Phenom II	38,07	42,40	10,21%	67,15	69,67	3,62%
Athlon II	48,12	53,90	10,72%	84,07	85,83	2,05%

Tabela 5.8: Comparativo de desempenho ao utilizar páginas de 4 KB e 2 MB na compilação do Apache

Os gráficos a seguir apresentam a redução no tempo de execução decorrente do aumento no número de núcleos de execução utilizados pela tarefa de compilação nos processadores Phenom II (Figura 5.4) e Athlon II (Figura 5.5). É possível observar que o comportamento ao utilizar paginação aninhada com páginas de 2 MB é muito próximo da execução nativa. O

desempenho ao utilizar paginação de sombra com páginas de 2 MB e 4 KB foram similares, atingindo valores mais próximos entre si conforme a contagem de núcleos aumenta, comportamento exibido nos dois processadores utilizados.

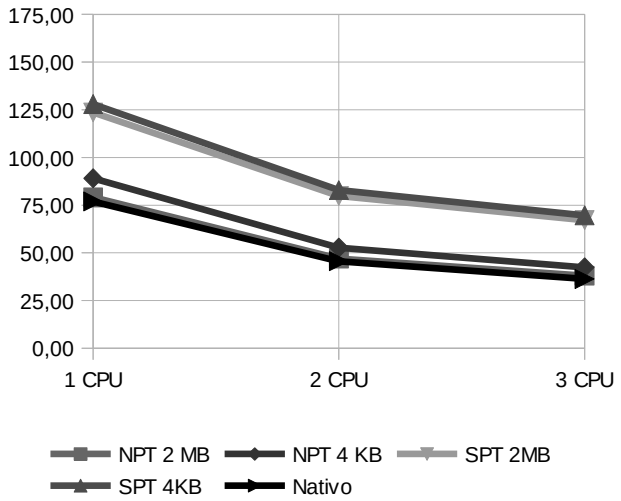

Figura 5.4: Tempo de execução x Quantidade de núcleos no processador Phenom II para compilação do Apache httpd

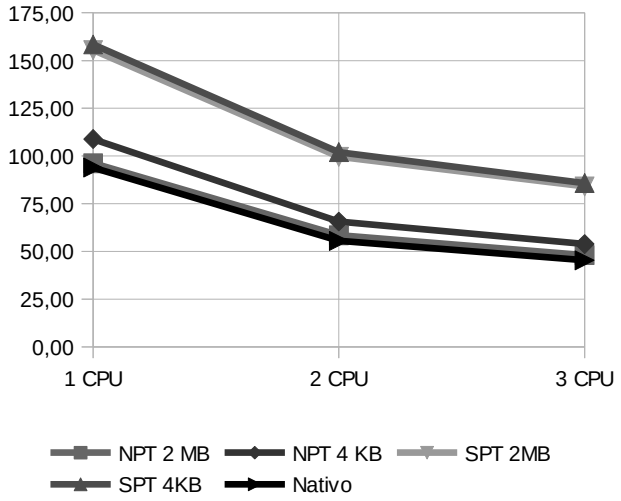

Figura 5.5: Tempo de execução x Quantidade de núcleos no processador Athlon II para compilação do Apache httpd

5.7.2 Parallel BZIP2

Nos testes conduzidos com o pbzip2, o desempenho foi maximizado ao utilizar páginas de 2 MB no sistema hospedeiro, como pode ser observado na Tabela 5.9. Esse comportamento não foi observado no processador Athlon II, que apresentou um tempo menor ao utilizar paginação de sombra com páginas de 4 KB e três núcleos ativos, porém a diferença foi de 0,02 s, inferior a 1%. Para referência, os resultados da execução nativa estão na Tabela 5.10.

Modo de Paginação	CPU	Páginas de 2 MB			Páginas de 4 KB		
		1 CPU	3 CPU	Speedup	1 CPU	3 CPU	Speedup
Páginação Aninhada	Phenom II	57,38	20,73	2,77	58,13	21,08	2,76
	Athlon II	74,95	27,59	2,72	76,30	28,19	2,71
Paginação de Sombra	Phenom II	58,51	20,67	2,83	58,70	20,78	2,82
	Athlon II	75,91	27,63	2,75	76,49	27,61	2,77

Tabela 5.9: Resultado obtidos dos experimentos de compressão com pbzip2 na máquina virtual

CPU	Nativo		
	1 CPU	3 CPU	Speedup
Phenom II	56,04	19,44	2,88
Athlon II	73,39	26,13	2,81

Tabela 5.10: Resultado obtidos dos experimentos de compressão com pbzip2 no sistema hospedeiro

A diferença entre a utilização de páginas de 4 KB e 2 MB foi pequena, inferior a 1% ao empregar tabelas de página de sombra, como pode ser visto na Tabela 5.11. Isso se deve ao fato do pbzip2 realizar alocação prévia dos blocos de memória utilizados durante o processo de compressão [76], de tal maneira que durante a execução a atividade da MMU é reduzida. O benefício é inferior ao utilizar paginação de sombra pois nesse caso uma página de 2 MB está sendo mapeada como múltiplas páginas de 4KB no sistema convidado.

CPU	Pag. Aninhada		Ganho 2 MB / 4 KB	Pag. Sombra		Ganho 2 MB / 4 KB
	2M	4K		2M	4K	
Phenom II	20,73	21,08	1,66%	20,67	20,78	0,53%
Athlon II	27,59	28,19	2,13%	27,63	27,61	-0,07%

Tabela 5.11: Comparativo de desempenho ao utilizar páginas de 4 KB e 2 MB no pbzip2

Na Figura 5.6 pode ser observado o benefício decorrente da adição do terceiro nível de cache compartilhado nos diferentes modos de paginação e tamanhos de página, com a

máquina virtual configurada para utilizar três núcleos. Os ganhos obtido em cada modo são muito próximos, e ligeiramente inferiores aos obtidos na execução nativa.

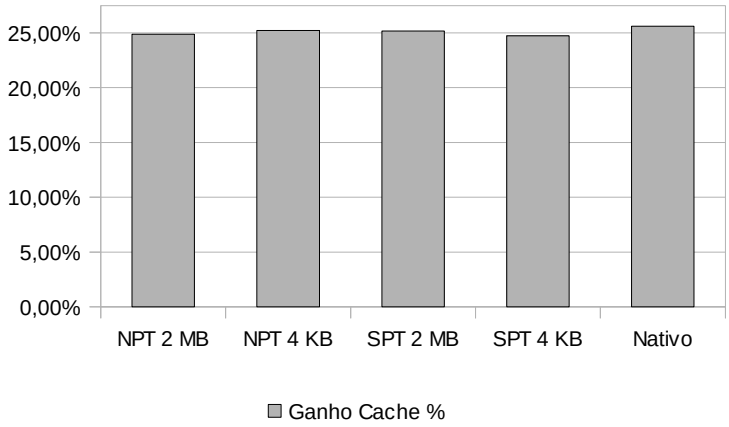

Figura 5.6: Influência do cache nos modos de paginação no pbzip2

A Figura 5.7 exibe o *speedup* ao aumentar de um para três núcleos de execução, utilizando páginas de 2 MB no sistema hospedeiro, comparada à execução nativa. Enquanto a paginação aninhada obteve um desempenho absoluto mais próximo ao nativo ao utilizar apenas um núcleo de processamento, o ganho ao aumentar a quantidade de núcleos de execução é inferior ao exibido pela paginação de sombra, indicando que em tarefas que envolvem pouca manipulação das tabelas de página a paginação de sombra pode ser mais escalável e apresentar resultados superiores à paginação aninhada.

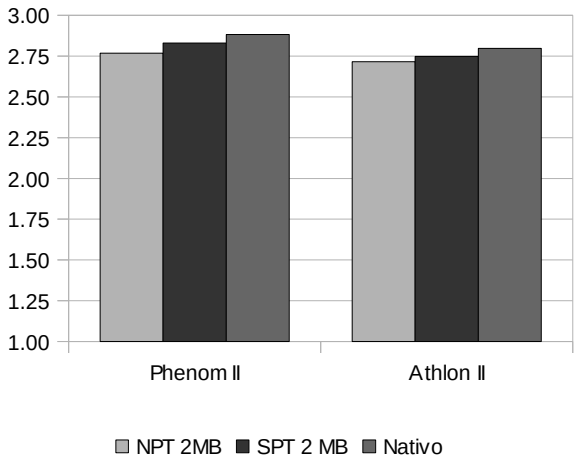

Figura 5.7: Speedup utilizando páginas de 2 MB na paginação aninhada e de sombra

Capítulo 6

Conclusões e Proposta de Trabalhos Futuros

6.1 Conclusões

Esse trabalho apresenta uma avaliação de técnicas de virtualização do subsistema de memória considerando aspectos existentes nas arquiteturas *multi-core*. Foi realizada uma análise comparativa das técnicas de virtualização do subsistema de memória, avaliando a paginação de sombra e a paginação aninhada, considerando também a utilização de páginas convencionais e grandes para mapear a memória da máquina virtual. Essas características foram analisadas levando em conta a escalabilidade das máquinas virtuais em um processador com múltiplos núcleos de processamento e também a presença de um nível adicional de memória cache compartilhada pelos núcleos de processamento.

Conforme observado nos testes realizados, a configuração mais efetiva consistiu em utilizar paginação aninhada aliada à adoção de páginas de 2 MB para o mapeamento da memória da máquina virtual pelo VMM, apresentando os melhores resultados ao aumentar a quantidade de núcleos de execução. Essa abordagem mostrou-se capaz de oferecer um desempenho notadamente mais próximo ao nativo em tarefas que exigem uma grande quantidade de manipulações das tabelas de página, situação explorada pelo teste de compilação do Apache. Nas tarefas em que atividade na MMU é reduzida, representado pelo teste de compressão paralela de dados, não há diferença significativa entre os dois modos de paginação ou o

tamanho de página quando se aumenta a quantidade de núcleos de execução, porém a utilização de páginas grandes, combinada à paginação aninhada, apresentou benefícios consistentes, ainda que reduzidos, em relação à utilização de páginas de 4 KB, ao diminuir a quantidade de etapas exigidas para a tradução de endereços lógicos para os de máquina.

Outra característica analisada foi a escalabilidade num ambiente de múltiplos núcleos de execução. Exceto pela utilização de paginação de sombra no teste de compilação do Apache, que apresentou tanto desempenho como escalabilidade sensivelmente inferiores aos resultados nativos, nas outras condições avaliadas foi observado um *speedup* similar aos das execuções dos experimentos diretamente no hardware, com uma pequena margem a favor da execução nativa. Isso demonstra que a virtualização impõe uma sobrecarga crescente conforme aumenta-se a quantidade de núcleos de execução, porém a diferença não é significativa a ponto de inviabilizar a utilização de mais núcleos de execução quando essa característica beneficiar a tarefa a ser desempenhada pela máquina virtual.

A presença do terceiro nível de *cache* trouxe benefícios nos resultados dos testes de compilação, oferecendo melhores resultados em relação à execução sem a abstração virtual. No teste de compactação paralela, que não exige manipulação intensiva das tabelas de página, foi observado o comportamento oposto, com ganhos pouco maiores na execução nativa. Em ambas as situações a diferença foi relativamente pequena, indicando que o *cache* compartilhado não oferece impacto significativo sobre a virtualização do subsistema de memória, sendo que o benefício de sua presença depende principalmente da aplicação a ser executada.

Um aspecto que merece destaque é que enquanto a implementação das tabelas de página de sombra é uma tecnologia madura, os recursos de paginação aninhada foram introduzidos recentemente nos processadores x86, e ao mesmo tempo que possuem oportunidades não exploradas de melhorias, como as propostas em [5], seu desempenho atual é similar ou superior àquele apresentado pelas tabelas de página de sombra em grande parte das aplicações, comportamento observado nos resultados dos experimentos conduzidos e também em [31] e [32]. O KVM por padrão utiliza a paginação aninhada sempre que disponível e desabilitar esse recurso não é trivial, como observado na seção 5.5.2.1. Levando em conta esses aspectos podemos considerar que a paginação de sombra ficará restrita a uma solução de contingência conforme a paginação aninhada se tornar um recurso padrão na arquitetura x86.

6.2 Trabalhos Futuros

Conforme mencionado na seção anterior, a paginação aninhada utilizando páginas de 2 MB ofereceu o melhor desempenho nos testes conduzidos. O suporte a páginas grandes no Linux é feita através de um sistema de arquivos especial denominado *hugetlbfs,* e sendo o KVM um subsistema Linux, ele também utiliza essa abstração, como mencionado na seção 5.5.2.1.

O *hugetlbfs* impõe algumas dificuldades à sua utilização, pois exige que as páginas sejam alocadas antecipadamente, e não permite sua emulação transparente utilizando páginas de tamanho convencional, além de exigir que as páginas grandes mantenham-se na memória principal, não podendo ser paginadas. Tais restrições não são adequadas à utilização para virtualização, pois em um ambiente dinâmico nem sempre é possível prever quanto de memória será utilizada pelas máquinas virtuais a fim de reservar as páginas grandes correspondentes. Além disso, como as páginas grandes são fixas na memória RAM, a consolidação de máquinas virtuais acaba prejudicada, pois não é possível enviar ao arquivo de troca páginas menos utilizadas para viabilizar a execução de um maior número de máquinas virtuais.

Atualmente existe uma proposta para incluir o suporte a páginas grandes na infraestrutura principal de memória virtual do *kernel* Linux, analisada em [78] e [79], sob a denominação de *Transparent Hugepages*. Os objetivos são permitir a alocação de páginas grandes de maneira transparente, com a utilização automática de páginas convencionais quando não houver páginas de memória contígua para alocar uma página grande. Também está previsto o suporte a paginação, porém, como a infraestrutura de paginação não oferece suporte a gravação de grandes blocos de dados, as páginas serão automaticamente fragmentadas em páginas convencionais ao serem paginadas.

Com a utilização das *Transparent Hugepages* seria possível utilizar páginas grandes tanto para mapear a memória das máquinas virtuais quanto pelo sistema operacional convidado, sem a necessidade de modificar as aplicações existentes. A combinação de páginas grandes no sistema hospedeiro e no sistema convidado oferece o melhor desempenho ao utilizar paginação aninhada, como apontam os resultados em [5] e [80]. Por esse motivo, seria interessante avaliar sua utilização a fim de verificar o desempenho observando o impacto em arquiteturas *multi-core*, seguindo a mesma abordagem do trabalho desenvolvido.

Os testes conduzidos utilizaram processadores da AMD, em virtude da sua disponibilidade à época dos testes. A AMD foi precursora na introdução do suporte à paginação aninhada, e continua realizando pesquisas com o intuito de propiciar um maior desempenho, conforme observado em [5]. A Intel é a principal fabricante de processadores x86, uma análise da sua implementação de paginação aninhada, aliada às características distintas da hierarquia de cache em relação aos modelos utilizados, poderiam constituir uma extensão a este trabalho, através de uma análise comparativa da influência das decisões de cada implementação.

Os processadores utilizados para a obtenção dos resultados experimentais possuíam três núcleos de execução. A escolha por essa configuração se deu por conta de sua disponibilidade prévia. Atualmente existem processadores *multi-core* com seis ou oito núcleos, alguns incluindo suporte a *multi-threading*, efetivamente dobrando a quantidade de núcleos lógicos. Uma proposta de trabalho futuro seria realizar os testes em processadores com mais núcleos, permitindo verificar a escalabilidade de cada um dos aspectos avaliados neste trabalho. Tal análise permitiria estabelecer se o comportamento previsto no teste da compressão paralela, em que a paginação de sombra poderia oferecer desempenho superior à paginação aninhada em tarefas que envolvem pouca manipulação das tabelas de página, conforme aumenta-se a quantidade de núcleos de execução, se concretizaria.

Também seria interessante avaliar o comportamento do KSM, mecanismo de compartilhamento de páginas utilizado pelo KVM e descrito na seção 3.7.1.1. Um aspecto a se avaliar seriam os possíveis ganhos de desempenho decorrentes do aumento da localidade ocasionado pelo compartilhamento de páginas. O compartilhamento de páginas também pode aumentar a eficiência dos níveis compartilhados de *cache*, tratando-se de um tema promissor de estudo.

Referências Bibliográficas

[1] UNITED STATES PATENT AND TRADEMARK OFFICE. DEVINE, S.W. et al. **Virtualization system including a virtual machine monitor for a computer with a segmented architecture**, U.S. Patent 6397242, 28 Mai. 2002.

[2] WALDSPURGER, C.A. Memory Resource Management in VMware ESX Server, In: 5TH SYMPOSIUM ON OPERATING SYSTEMS DESIGN AND IMPLEMENTATION, Rev. 36, SI, Dez. 2002, Boston, **Proceedings...**, p. 181-194.

[3] BARHAM P. et al. Xen and the art of virtualization, In: 19TH ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, Dez. 2003, New York, **Proceedings...**, p. 164-177.

[4] AMSDEN Z. et al. VMI: An Interface for Paravirtualization, In: LINUX SYMPOSIUM, vol. 2, Jul. 2006, Ottawa, **Proceedings...**, p. 371-386.

[5] BHARGAVA R. et al. Accelerating two-dimensional page walks for virtualized systems, In: 13TH INTERNATIONAL CONFERENCE ON ARCHITECTURAL SUPPORT FOR PROGRAMMING LANGUAGES AND OPERATING SYSTEMS, Mar 2008, New York, **Proceedings...**, p. 26-35.

[6] NELSON M. et al. Fast transparent migration for virtual machines, In: USENIX ANNUAL TECHNICAL CONFERENCE, 2005, Anaheim, **Proceedings...** Berkeley: USENIX Association, 2005, p. 391-394.

[7] PRATT I. et al. Xen 3.0 and the Art of Virtualization, In: LINUX SYMPOSIUM, vol. 2, Jul. 2005, Ottawa, **Proceedings...**, p. 65-78.

[8] VMWARE, **VMware Distributed Power Management: Concepts and Usage**, 24 Mar. 2010. Disponível em: <http://www.vmware.com/resources/techresources/1080>. Acesso em: Jun. 2010.

[9] NATHUJI R.; SCHWAN K. VirtualPower: coordinated power management in virtualized enterprise systems, 21ST ACM SIGOPS SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES – SOSP '07, Dez 2007, New York, **Proceedings...**, p. 265-278.

[10] LAUREANO M. **Máquinas Virtuais e Emuladores: Conceitos, Técnicas e Aplicações.** São Paulo: Novatec, 2006. 184 p.

[11] POPEK G.J.; GOLDBERG R.P. **Formal requirements for virtualizable third generation architectures**, Communications of the ACM, Vol. 17 Issue 7, Jul 1974, p. 412-421.

[12] VMWARE, **The Architecture of Vmware ESXi**, Disponível em: <http://www.vmware.com/files/pdf/ESXi_architecture.pdf>. Acesso em: Jun. 2010.

[13] XENSOURCE, **Xen Users' Manual v3.3**, Disponível em: <http://bits.xensource.com/Xen/docs/user.pdf>. Acesso em: Jul. 2010.

[14] KIVITY A. et. al. KVM: the Linux Virtual Machine Monitor, In: **Linux Symposium**, vol. 2, Jun. 2007, Ottawa, **Proceedings...**, p. 225-230.

[15] VMWARE, **VMware Server User's Guide**, Rev. 28 Ago. 2008. Disponível em: <http://www.vmware.com/pdf/vmserver2.pdf>. Acesso em: Set. 2009.

[16] ORACLE, **Oracle VM VirtualBox User Manual: Technical background**, Disponível em: <http://www.virtualbox.org/manual/ch10.html>. Acesso em: Set. 2009.

[17] BOCHS, **Bochs: The Cross Platform IA-32 Emulator**, Disponível em: <http://bochs.sourceforge.net/>. Acesso em: Fev. 2009.

[18] LAWTON K.P. **Bochs: A Portable PC Emulator for Unix/X**, Linux Journal, Volume 1996 Issue 29 Article No. 7, Set. 1996

[19] ADAMS K.; AGESEN O. A comparison of software and hardware techniques for x86 virtualization, In: 12TH INTERNATIONAL CONFERENCE ON ARCHITECTURAL SUPPORT FOR PROGRAMMING LANGUAGES AND OPERATING SYSTEMS, 2006, San Jose, **Proceedings...**, p. 2-13.

[20] VMWARE, **Performance of VMware VMI***,* Rev. 13 Fev. 2008. Disponível em: <http://www.vmware.com/resources/techresources/1038>. Acesso em: Mar. 2009.

[21] NAKAJIMA J.; MALLICK A.K. Hybrid-Virtualization - Enhanced Virtualization for Linux, In: LINUX SYMPOSIUM, Jun. 2007, Ottawa, **Proceedings...**, p. 87-96.

[22] VMWARE, **Paravirtualization API Version 2.5**, Rev. 23 Fev. 2006. Disponível em: <http://www.vmware.com/pdf/vmi_specs.pdf>. Acesso em: Mai. 2009.

[23] ADVANCED MICRO DEVICES, **Secure virtual machine architecture reference manual**, AMD Publication No. 33047 Rev. 3.01, Mai. 2005.

[24] INTEL, **Intel® 64 and IA-32 Architectures Software Developer's Manual – Volume 2B: Instruction Set Reference, N-Z**. Disponível em: <http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-2b-manual.pdf>. Acesso em: Jan. 2010.

[25] INTEL, **Intel® 64 and IA-32 Architectures Software Developer's Manual – Volume 3B: System Programming Guide, Part 2**. Disponível em: <http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.pdf>. Acesso em: Jan. 2010.

[26] VMWARE, **Software and Hardware Techniques for x86 Virtualization**, Rev. 25 Jun. 2009. Disponível em: <http://www.vmware.com/resources/techresources/10036>. Acesso em: Fev. 2010.

[27] RUTKOWSKA J. **Subverting Vista Kernel For Fun And Profit**, In: BLACK HAT BRIEFINGS 2006, Ago. 2006, Las Vegas.

[28] NAVARRO J. et al. Practical, transparent operating system support for superpages, In: 5TH SYMPOSIUM ON OPERATING SYSTEMS DESIGN AND IMPLEMENTATION, 2002, Boston, **Proceedings...**, p. 89-104.

[29] JACOB B.; MUDGE T. **Virtual Memory in Contemporary Microprocessors**, IEEE Micro, vol. 18, 1998, p. 60-75.

[30] RED HAT, **New security Enhancements in Red Hat Enterprise Linux v.3, update 3**. Disponível em: <http://www.redhat.com/f/pdf/rhel/WHP0006US_Execshield.pdf>. Acesso em: Jan. 2010.

[31] VMWARE, **Performance Evaluation of Intel EPT Hardware Assist**. Rev. 30 Mar. 2009. Disponível em: <http://www.vmware.com/resources/techresources/10006>. Acesso em: Fev. 2010.

[32] VMWARE, **Performance Evaluation of AMD RVI Hardware Assist**. Rev. 11 Mar. 2009. Disponível em: <http://www.vmware.com/resources/techresources/1079>. Acesso em: Fev. 2010.

[33] OGLESBY R.; HEROLD S.; LAVERICK M. **VMware Infrastructure 3: Advanced Technical Design Guide and Advanced Operations Guide.** Tech Target, 2008. 816 p**.**

[34] ADVANCED MICRO DEVICES, **Software Optimization Guide for AMD Family 10h Processors***,* AMD Publication No. 40546 Rev. 3.11, Mai. 2009.

[35] ADVANCED MICRO DEVICES, **Software Optimization Guide for AMD64 Processors***,* AMD Publication No. 25112 Rev. 3.11, Set. 2005

[36] INTEL, **Intel® 64 and IA-32 Architectures Optimization Reference Manual**, Disponível em: <http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf> . Acesso em: Fev. 2010.

[37] ARCANGELI A.; EIDUS I.; WRIGHT C. Increasing memory density by using KSM, In: LINUX SYMPOSIUM, Jul. 2009, Ottawa, **Proceedings...**, p. 19-28.

[38] EDGE J. **KSM runs into patent trouble**. Disponível em: <http://lwn.net/Articles/309155/>. Acesso em: Out. 2009.

[39] UNITED STATES PATENT AND TRADEMARK OFFICE. WALDSPURGER C.A. **Content-based, transparent sharing of memory units**, U.S. Patent 6789156, 7 Set. 2004.

[40] ADVANCED MICRO DEVICES, **AMD64 Architecture Programmer's Manual Volume 2: System Programming**. AMD Publication No. 24593 Rev. 3.15, Nov. 2009.

[41] WELLS P.M.; CHAKRABORTY K.; SOHI G.S. **Dynamic heterogeneity and the need for multicore virtualization**, In: SIGOPS OPERATING SYSTEM REVIEW, vol. 43, New York: ACM, 2009, p. 5-14.

[42] SHAH M. et al. **UltraSPARC T2: A highly-treaded, power-efficient, SPARC SOC**, In: SOLID-STATE CIRCUITS CONFERENCE, 2007. ASSCC '07. IEEE Asian, 2007, p. 22-25.

[43] AMDAHL G.M. Validity of the single processor approach to achieving large scale computing capabilities, In: APRIL 18-20, 1967, SPRING JOINT COMPUTER CONFERENCE, 1967, Atlantic City: ACM, **Proceedings...**, Atlantic City: ACM, p. 483-485.

[44] INTEL, **Intel® Hyper-Threading Technology**, Disponível em: <http://www.intel.com/technology/platform-technology/hyper-threading/>. Acesso em: Abr. 2009.

[45] INTEL, **Intel® Corporation Hyper-Threading Technology System Requirements**. Disponível em: <http://www.intel.com/products/ht/hyperthreading_more.htm>. Acesso em: Mai. 2010.

[46] INTEL, **Intel® Atom™ Processor Microarchitecture**. Disponível em: <http://www.intel.com/content/www/us/en/processors/atom/atom-processor-embedded-technology.html>. Acesso em Mai. 2010.

[47] BEAVERS B. **The Story behind the Intel Atom Processor Success**, In: DESIGN & TEST OF COMPUTERS, IEEE, vol. 26, Issue 2, 2009, p. 8-13.

[48] INTEL, **Intel Atom Processor Family**. Disponível em: <http://ark.intel.com/products/family/29035/Intel-Atom-Processor>. Acesso em: Mai. 2010.

[49] HEISS A.; APPEL D. Intel Atom, **Revista PC&Cia**, São Paulo, n. 87 , p. 26-30, Jun. 2009.

[50] INTEL, **Intel 64 and IA-32 Architectures Software Developer's Manual – Volume 1: Basic Architecture**. Disponível em: <http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-1-manual.pdf>. Acesso em: Mai. 2010.

[51] GEROSA G. et al. **A Sub-1W to 2W Low-Power IA Processor for Mobile Internet Devices and Ultra-Mobile PCs in 45nm Hi-K Metal Gate CMOS**, In: SOLID-STATE CIRCUITS CONFERENCE, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International, 2008, p. 256-611.

[52] APPEL D. Agendando Tarefas com Eficiência no Linux, **Revista PC&Cia**, São Paulo, n. 70, p. 9-13, Mai. 2007.

[53] ADVANCED MICRO DEVICES, **AMD Athlon X2 Dual-Core Processor Product Data Sheet**, AMD Publication no. 43042 rev. 3.00, Mai. 2007.

[54] INTEL, **Intel Core 2 Duo Desktop Processor Family**. Disponível em: <http://ark.intel.com/products/family/26547/Intel-Core2-Duo-Desktop-Processor>. Acesso em Mai. 2010.

[55] ADVANCED MICRO DEVICES, **Family 10h AMD Phenom Processor Product Data Sheet**, AMD Publication no. 44109 rev. 3.00, Nov. 2007.

[56] ADVANCED MICRO DEVICES, **Family 10h AMD Phenom II Processor Product Data Sheet**, AMD Publication no. 46878 rev. 3.04, Fev. 2009.

[57] INTEL, **First the Tick, Now the Tock: Next Generation Intel Microarchitecture (Nehalem)**. Disponível em: <http://www.intel.com/content/dam/doc/white-paper/intel-microarchitecture-white-paper.pdf>. Acesso em: Dez. 2009.

[58] KONGETIRA P.; AINGARAN K.; OLUKOTUN K. **Niagara: a 32-way multithreaded Sparc processor**, IEEE Micro, vol. 25, 2005, p. 21-29.

[59] BAE C.; LANGE J.R.; DINDA P.A **Comparing Approaches to Virtualized Page Translation in Modern VMMs**. Disponível em: <http://v3vee.org/papers/NWU-EECS-10-07.pdf>. Acesso em: Nov. 2009.

[60] VMWARE, **Customers Success Stories** Disponível em: <http://www.vmware.com/customers/>. Acesso em: Fev. 2010.

[61] CITRIX, **Citrix XenServer: Efficient Server Virtualization Software.** Disponível em: <http://www.citrix.com/xenserver/>. Acesso em: Mar. 2010.

[62] MICROSOFT, **Microsoft Hyper-V Server: Home Page**. Disponível em: <http://www.microsoft.com/en-us/server-cloud/hyper-v-server/default.aspx>. Acesso em: Fev. 2010

[63] APPEL D. 1 PC resolve tudo – Construa seu próprio servidor de máquinas virtuais, **Revista PC&Cia**, São Paulo, n. 72, p. 10-22, Jul. 2007.

[64] APPEL D. Servidor de máquinas virtuais 2.0, **Revista PC&Cia**, São Paulo, n. 86, p. 20-26, Abr. 2009.

[65] VIVENCIO D.P. Virtualização prática com XenServer Express 5, **Revista PC&Cia**, São Paulo, n. 86, p. 27-32, Abr. 2009.

[66] ADVANCED MICRO DEVICES, **AMD Introduces the World's Most Advanced x86 Processor, Designed for the Demanding Datacenter.** Disponível em: <http://www.amd.com/us/press-releases/Pages/Press_Release_119768.aspx>. Acesso em: Jan. 2010.

[67] INTEL, **Intel Launches Fastest Processor on the Planet**. Disponível em: <http://www.intel.com/pressroom/archive/releases/2008/20081117comp_sm.htm>. Acesso em: Set. 2009.

[68] ADVANCED MICRO DEVICES, **AMD Announces Widespread Availability and Broad Global OEM Support for New Quad-Core AMD Opteron Processor**. Disponível em: <http://www.amd.com/us/press-releases/Pages/Press_Release_129135.aspx>. Acesso em: Ago. 2009.

[69] ADVANCED MICRO DEVICES, **New Six-Core AMD Opteron Processor Delivers Up to Thirty-Four Percent More Performance-per-Watt in Exact Same Platform**. Disponível em: <http://www.amd.com/us/press-releases/Pages/new_six-core_amd_opteron_processor-2009jun01.aspx>. Acesso em: Ago. 2009.

[70] INTEL, **Internet: Meet Your New Processor – Intel Xeon Processor 5500 Series**. Disponível em: <http://www.intel.com/pressroom/archive/releases/2009/20090330corp_sm.htm>. Acesso em: Ago. 2009.

[71] APPARAO P.; IYER R.; NEWELL D. **Implications of cache asymmetry on server consolidation performance**, In: IEEE INTERNATIONAL SYMPOSIUM ON WORKLOAD CHARACTERIZATION, 2008, Seattle, p. 24-32.

[72] RED HAT, **Backporting de correções de segurança**. Disponível em: <https://access.redhat.com/security/updates/backporting/>. Acesso em: Out. 2009.

[73] KVM, **Main Page – KVM**. Disponível em: <http://www.linux-kvm.org/page/Main_Page>. Acesso em: Nov. 2009.

[74] QEMU, **Main Page – QEMU**. Disponível em: <http://wiki.qemu.org/Main_Page>. Acesso em: Nov. 2009.

[75] RUSSELL R. **virtio: towards a de-facto standard for virtual I/O devices**. In: ACM SIGOPS OPERATING SYSTEMS REVIEW, Vol. 42 Issue 5, 2008, p. 95-103.

[76] GILCHRIST, J. Parallel data compression with bzip2. In: 16TH INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED COMPUTING AND SYSTEMS, 2004, **Proceedings...**, p. 559-564.

[77] GILCHRIST, J.; CUHADAR, A. **Parallel lossless data compression using the Burrows-Wheeler Transform**. In: International Journal of Web and Grid Services, v. 4, n. 1, 2008, p. 117-135.

[78] CORBET J. **Transparent hugepages**. Disponível em: <http://lwn.net/Articles/359158/>. Acesso em: Mai. 2010.

[79] ARCANGELI A. **Transparent Hugepage Support**, KVM Forum 2010, 09 Ago. 2010. Disponível em: <http://www.linux-kvm.org/wiki/images/9/9e/2010-forum-thp.pdf> Acesso em: Ago. 2010.

[80] SEREBRIN B.; RÖDEL J. **Nested paging hardware and software**, KVM Forum 2008, 13 Jun. 2008. Disponível em: <http://www.linux-kvm.org/wiki/images/c/c8/KvmForum2008%24kdf2008_21.pdf> Acesso em: Ago. 2010.