O AVANÇO TECNOLÓGICO DOS PROCESSADORES E SUA UTILIZAÇÃO PELO SOFTWARE

EDSON PEDRO FERLIN
Professor e Coordenador - Curso de Engenharia da Computação / UnicenP
Professor - PUCPR
Professor - SPEI
ferlin@unicenp.br

RESUMO

Neste artigo abordaremos algumas questões envolvendo o avanço tecnológico dos processadores e a sua utilização pelos softwares e em especial pelo sistema operacional. Os processadores têm tido um crescimento muito significativo nas últimas décadas, em virtude dos recursos tecnológicos que foram incorporados nas suas arquiteturas. Contudo, para aproveitar este potencial é preciso que o software também acompanhe este crescimento, mas o que se percebe é que isto não está acontecendo, pelo menos não com a velocidade que seria necessária para se obter um desempenho próximo do máximo permitido para a arquitetura. Como discutiremos neste artigo, poderemos observar que muitos dos limitadores são influenciados pelos fatores mercadológicos e que implicam em uma limitação no desempenho dos computadores, em virtude da utilização de softwares obsoletos que não utilizam adequadamente os recursos computacionais dos processadores, gerando uma baixa eficiência.

Palavras-chave: Arquitetura, Desempenho, Processador, Software, Tecnologia.

ABSTRACT

This paper addresses some questions regarding the technological advance of processors and their utilization by software, specially operating systems. The processors have gained importance in the last decades, because of technological resources that were incorporated in their architecture. However, in order to take advantage of this potential, it is necessary that software also follow suit, what is not happening, at least not as quickly as necessary to obtain a performance close to the maximum allowed by the architecture. It was observed that many limiting factors are imposed by the market, due to the utilization of obsolete software, which do not adequately use the computational resources of the processors, resulting low efficiency in the systems.

Key words: Architecture, Performance, Processor, Software, Technology.

O AVANÇO TECNOLÓGICO DOS PROCESSADORES E SUA UTILIZAÇÃO PELO SOFTWARE

EDSON PEDRO FERLIN

1. INTRODUÇÃO

Atualmente, muito se comenta a respeito do avanço tecnológico dos processadores, como por exemplo o aumento da freqüência de clock, o acréscimo de memória cachê, maior número de unidades funcionais, execução especulativa, hyper-threading e outros. Contudo, pouco se discute a respeito do software, pois apenas sabe-se que executa e a partir daí é uma incógnita para a maioria dos usuários e também para uma parcela dos desenvolvedores. Muitos programadores se preocupam somente em digitar linhas de código sem compreender a arquitetura que está por de traz, e que efetivamente executará o software. Eles não sabem também, por exemplo, qual o software é o mais apropriado para uma dada arquitetura ou se está conseguindo utilizar a potencialidade do processador.

Os computadores possuem em sua arquitetura como componente principal o processador, que é o cérebro do sistema e que realiza o processamento das informações. O processador não é o único componente, mas é com base nele que o computador ganha a maioria das características, dentre outras a freqüência de operação (clock), a largura do barramento de dados, e a capacidade máxima de memória.

O processador é em disparado o produto de maior avanço tecnológico construído pelo homem, sendo muitos outros produtos desenvolvidos pela utilização dele nos computadores, direta ou indiretamente. Os aviões, os equipamentos médicos e as simulações meteorológicas são exemplos da utilização deste componente.

2. OS PROCESSADORES

Os processadores atuais, como Pentium, PowerPC, UltraSparc, e outros, incorporam diversos recursos computacionais, como memória cachê, pipeline, unidade de ponto flutuante, unidade de branch, que não existiam nas primeiras gerações de processadores. Esta evolução teve inicio em 1971 com o lançamento do primeiro microprocessador, o 4004 da Intel e que era um modesto processador de 4 bits, mas que inaugurou a nova tecnologia dos integrados VLSI (*Very Large Scale Integration*), devido ao avanço de uma nova área da eletrônica que iniciava, denominada de Microeletrônica (ENDERLEIN, 1994).

Contudo, foi com o desenvolvimento do conceito de computador pessoal (PC – *Personal Computer*), na década de 80, que a computação começou a tomar forma como a conhecemos hoje, pois o mercado dos computadores para uso doméstico fez com que as diversas empresas de hardware tivessem que conquistar mercado com novos produtos. Um dos principais produtos foi o processador com o objetivo de atender a este novo mercado mais exigente em termos de velocidade de processamento.

Com a redução dos espaços ociosos dentro do encapsulamento do chip (Circuito Integrado) do processador, devido ao avanço da microeletrônica, possibilitou que mais recursos,

que antes estavam fora do processador, fossem, agora, encapsulados na mesma unidade. Um dos primeiros recursos incorporados aos microprocessadores foi co-processador aritmético, que realizava todo o processamento das operações de ponto flutuante, e que agora são realizadas pela unidade de ponto flutuante, que é uma das unidades internas dos processadores.

Os primeiros processadores tinham uma série de limitações que os tornavam de uso restrito e o software, no passado, estava quase sempre em concordância com o hardware, ou seja, o software tinha o código que era próprio para a arquitetura do processador e por conseqüência do computador. Entretanto, com o avanço da microeletrônica os processadores continuaram evoluindo a passos bem mais audaciosos, criando com isso microprocessadores avançados, longe do que os softwares conseguiam utilizar, resultando em uma lacuna entre o hardware e o software, a qual continua nos dias de hoje.

Esta distância mencionada está aumentando, pois a evolução dos processadores é muito rápida, tendo como evidência o grande número de novos que foram lançados nos últimos anos, sem contar com os que ainda estão nas pranchetas dos engenheiros e projetistas e que aguardam a finalização dos testes para serem disponibilizados no mercado.

Todavia, por motivos econômicos as empresas de software não estão conseguindo acompanhar esta evolução. Para cada novo processador devemos ter também novos softwares e é neste ponto que os problemas começam, pois desta forma há diversas versões para um mesmo software, onerando-o e restringindo ainda este mercado.

O software é composto por instruções que são próprias de cada processador ou a uma família de processadores como é o caso do x86 da Intel. É por meio das instruções que se consegue acessar os diversos recursos dos processadores. Estas instruções estão definidas no conjunto de instruções no nível ISA (*Instruction Set Architecture*) do processador (TANENBAUM, 2001). Quando se compila um programa, o compilador transforma cada comando da linguagem em uma seqüência de instruções apropriada para a arquitetura. Esta seqüência de instruções é montada pelo desenvolvedor do compilador e é também elaborada com base na lógica do programador. Contudo nem sempre o compilador consegue aproveitar as melhores instruções do conjunto de instruções disponíveis para aquele determinado processador, deixando o software muito pobre em termos de utilização dos recursos. Este comportamento ocorre porque nem sempre o desenvolvedor conhece tão bem a arquitetura, os recursos do processador e a melhor maneira de acessá-los por meio da programação.

Para compreendermos um pouco melhor estes fatos, verificaremos os recursos atualmente disponíveis da arquitetura interna dos processadores, com base na arquitetura do Pentium 4 da Intel (HINTON, 2001), não por ser o mais avançado, mas pelo simples fato de ter grande utilização no mercado.

2.1. Arquitetura Interna do Processador

Na Figura 1 vemos a arquitetura interna de um processador Pentium 4, o qual possui sete unidades funcionais independentes: Store, Load, 2xALU(*Aritmetic Logic Unit*) Inteiro, ALU Complexo, MMX/SSE (*MultiMídia eXtension*)/(*Streaming SIMD Extensions*) e FP (*Floating Point*). Além destes recursos, temos também as memórias cachês nível 1 e nível 2, execução fora-de-ordem e pipeline com 20 estágios. Contudo, se a seqüência das instruções, e a dependência entre elas, não for a mais apropriada, então a maioria destas unidades fica-

rão ociosas, resultando em um desempenho inferior ao possível, dando uma falsa sensação de baixa eficiência.

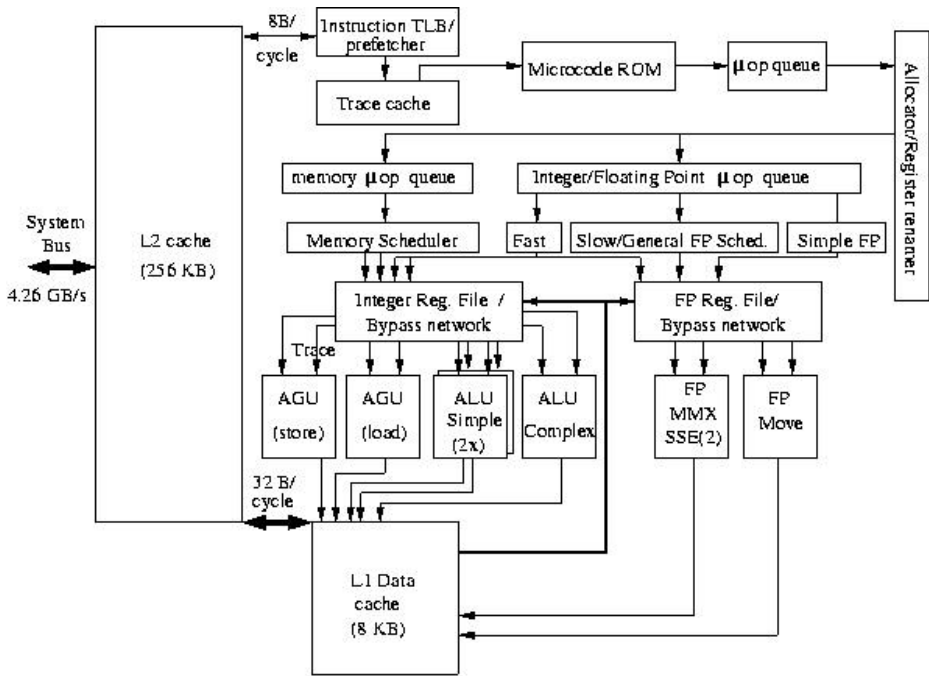

Figura 1. Arquitetura Interna do Processador Pentium 4 (Fonte: Intel)

A seguir estão descritos os principais recursos tecnológicos incorporados nos processadores. Ressaltamos que nem todos os recursos descritos se encontram presentes em todos os processadores, pois dependem da arquitetura e da topologia interna de cada um:

2.1.1. Unidade de Inteiro

A unidade de inteiro é a unidade funcional mais utilizada no processador. Ela é a responsável pelas operações aritméticas unárias e binárias que operam sobre os números inteiros e também operações booleanas. É denominada na maioria das vezes como ULA (Unidade Lógica Aritmética), ou em inglês como ALU (*Aritmetic Logic Unit*), e sua largura (em bits) é determinada pelo barramento de dados do processador. Esta é uma unidade presente desde o primeiro processador e é considerada como a primeira unidade funcional, devido a estar presente no modelo de máquina de Von Neumann (STALLINGS, 2002).

Apesar do fato de ser uma unidade básica de todos os processadores, ela possui diferenças funcionais consideráveis, por exemplo, se não tiver a função de multiplicação então se deve re-escrever o código utilizando o recurso de sucessivas somas, e logicamente

isto tem impacto direto sobre o software. A utilização de diferentes seqüências de instruções implica em tempo de execução e tamanho em bytes também diferentes, dependendo da arquitetura do processador.

2.1.2. Unidade de Ponto Flutuante

Esta unidade foi incorporada, a partir dos 80486DX (HYDE, 2003), para possibilitar a execução de operações de ponto flutuante no próprio processador. Anteriormente era necessário à utilização de um *chip* adicional que desempenhava esta função e era chamado de co-processador aritmético. Contudo com a crescente necessidade deste tipo de operação, por parte do software, os projetistas resolveram incorporar dentro do processador este recurso, não mais como um opcional mas como um recurso de grande valia para o processamento numérico. Esta unidade é normalmente denominada de FP (*Floating Point*).

Este recurso proporcionou um ganho no desempenho, pois, as operações que envolvem ponto flutuante, tinham que ser emuladas via software utilizando a ULA. Com este recurso passaram a ser executadas diretamente pelo hardware desta unidade especializada resultando em uma economia de tempo, de processamento e de recurso computacional.

2.1.3. Pipeline

A idéia de um *Pipeline* é a de dividir o ciclo da instrução em estágios consecutivos (STALLINGS, 2002), como uma linha de produção. Desta forma, consegue-se acelerar a execução das instruções, resultando em uma redução do tempo de execução e por conseqência o aumento do desempenho. Teoricamente, quanto maior o número de estágios do *pipeline*, maior o *speedup* (PATTERSON & HENNESSY, 2000). Entretanto, na prática, o ganho potencial com a introdução de estágios adicionais no *pipeline* é reduzido pelo aumento do custo, por atrasos entre estágios e pelo fato de que as instruções de desvio requerem que o *pipeline* seja esvaziado.

Um dos maiores problemas do projeto de *pipeline* é assegurar um fluxo constante de instruções nos estágios iniciais, pois desta maneira ele não se esvaziará. O principal impedimento a isso é a existência das instruções de desvio condicional, que faz com que se tenha que retirar as instruções do *pipeline* para colocar outras que correspondam ao ramo do desvio a ser executado. Antes que a instrução seja realmente executada, é impossível determinar se o desvio será executado ou não. Para minimizar este impacto é que os processadores têm a unidade de *branch*, que procura fazer uma antecipação sobre o ramo a ser executado (FOG, 2003).

O *pipeline* é determinante na execução dos programas. Análises de comportamento de programas mostram que desvios condicionais são tomados em mais de 30% das vezes (LILJA, 1988), e isto, obviamente, tem um impacto direto sobre o desempenho do *pipeline* e por conseqüência também sobre o desempenho geral do sistema. O *pipeline* proporciona um paralelismo temporal, em função de que há diversas instruções sendo parcialmente executadas, durante o mesmo intervalo de tempo, em cada estágio do *pipeline*. Este recurso foi incorporado nas arquiteturas RISC e VLIW em virtude de uma característica marcante na qual as instruções consomem o mesmo tempo para serem executadas. A duração de tempo igual para as instruções implica também, mas não necessariamente, em um tempo igual nos

estágios. Esta característica é crucial para as arquiteturas com *pipeline*, garantindo que nenhuma instrução paralise o fluxo, o que resultaria no esvaziamento do *pipeline*.

2.1.4. Unidade de *Branch*

Os desvios condicionais são os grandes causadores da degradação do desempenho do computador, principalmente nos computadores com processadores que tem *pipeline* em sua arquitetura interna. Segundo Lilja (LILJA, 1988) em análises de comportamento de programas mostram que os desvios condicionais são tomados em mais de 30% das vezes, e isto obviamente tem um impacto direto sobre o *pipeline*.

Os projetistas adotam nas arquiteturas com *pipeline* o recurso da unidade de *branch*, que realiza uma estatística em tempo real da probabilidade da ocorrência de determinado ramo de execução (INTEL, 1997). Com isso o processador reduz o número de esvaziamentos do *pipeline*, pois antevê o ramo mais provável de ocorrência, resultando em um ganho na quantidade de instruções executadas por ciclo de máquina.

Normalmente, as arquiteturas que possuem este recurso necessitam que os programas sejam re-ordenados de forma a ter uma instrução válida, ou seja, que pertença ao ramo correto de execução, imediatamente após a instrução de salto (desvio). Cabe ressaltar que este processo ocorre de forma transparente para o usuário e é gerenciado pelo próprio processador.

2.1.5. Unidades de *Load* e *Store*

As arquiteturas atuais possuem duas unidades especializadas para as operações de leitura e escrita na memória principal, denominadas de *Load* e *Store*, respectivamente. Em algumas arquiteturas estas duas unidades são agrupadas em uma única unidade denominada de *Load/Store*.

A unidade de *Load* é a responsável pela busca das instruções e dados da memória, enquanto o barramento estiver ocioso, para os registradores internos ou então para a memória cachê. Uma das vantagens é que aproveita a ociosidade do barramento para antecipar esta busca e outra é que possibilita a manutenção do *pipeline* sempre cheio.

A unidade *Store* é quem realiza a escrita dos dados processados novamente na memória principal, também aproveitando o tempo em que o barramento está ocioso.

O trabalho destas duas unidades é favorecido pela largura do barramento de dados, atualmente de 64bits, que possibilita uma boa taxa de transferência. Estas duas unidades funcionais isentam os demais recursos do processador das operações de leitura e escrita na memória principal. Estes foram recursos provenientes das arquiteturas RISC (*Reduced Instruction Set Computer*), e que foram incorporados também nas arquiteturas VLIW (*Very Large Instruction Word*), garantindo um melhor desempenho destes processadores.

Uma implicação para o software nas arquiteturas com este tipo de recurso é que a operação de *Load* é sempre atrasada, necessitando colocá-la algumas instruções antes no código para que consiga buscar as informações no tempo hábil, ou seja, dentro do ciclo de máquina.

2.1.6. Unidade de Gerenciamento da Memória

Os processadores modernos, como o caso do PowerPC 750 (IBM, 1999), podem gerenciar uma grande quantidade de memória, tanto física 32bits (4Gbytes) quanto virtual 52bits (4Pbytes). Desta forma é necessário que o processador tenha uma unidade que faça esse processo de translação, dos endereços virtuais em físicos, de maneira rápida e sem dispêndios de ciclos de máquina adicionais por parte das unidades de execução. Com isso, o processador ganha tempo de execução, sem que seja necessário o dispêndio de tempo para este processo. Este processo é realizado pela unidade de gerenciamento da memória, denominada em inglês de MMU (*Manager Memory Unit*), e possibilita aos computadores, por meio do Sistema Operacional, a utilização do conceito de memória virtual (TANENBAUM, 2001).

Esta unidade é importantíssima nos processadores, pois é a responsável pelo gerenciamento das memórias física e virtual, controlando o acesso à memória cachê e também mantendo a coerência entre os dados presentes na memória principal com a cachê. Ela gerencia a troca de processos na memória (*swapping*), a partição e paginação da memória e a memória virtual (STALLINGS, 2002).

O gerenciamento eficiente da memória é vital para computadores com multiprogramação, garante que diferentes processos estejam na memória ao mesmo tempo e preservando a isonomia entre eles.

2.1.7. Unidade *Dispach*

A unidade *Dispach* é um dos pontos-chave da arquitetura VLIW (PHILIPS, 2003). Ela é a responsável pelo envio das instruções para as respectivas unidades funcionais, com base no gabarito de instruções. Este gabarito contém as instruções que podem ser executadas simultaneamente pelo processador em suas unidades internas, e é montado pelo compilador, durante a compilação do programa.

Esta unidade é posicionada no centro lógico da arquitetura e deve possuir uma inteligência suficiente para a tomada das decisões sobre o escalonamento das instruções nas respectivas unidades funcionais, na medida em que estas unidades ficarem ociosas. Contudo, nem sempre todas as unidades estarão em operação ao mesmo tempo, pois dependente do algoritmo, então o gabarito não será preenchido por completo e na posição vaga será colocada instrução NOP (*No-OPeration*). Esta instrução não interfere no processamento, apenas consome ciclos de máquina. Como conseqüência, o compilador para este tipo de processador deve possuir uma inteligência e um conhecimento a respeito da arquitetura suficiente para efetuar o preenchimento do gabarito, aproveitando ao máximo os recursos da arquitetura VLIW. Em caso contrário o código não fornecerá as condições para se alcançar um bom desempenho, como é o caso do paralelismo interno disponível neste tipo de arquitetura.

A unidade de *dispach* só escalona as instruções que estiverem explicitadas no gabarito. Desta forma, os softwares que não estiverem preparados com esta configuração, não apresentarão o melhor desempenho.

2.1.8. Superescalar

Um processador superescalar é aquele no qual são usadas várias instruções independentes (IBM, 1999). Cada *pipeline* tem diversos estágios, podendo manipular várias instruções a cada instante. O uso de vários *pipelines* introduz um novo nível de paralelismo, possibilitando executar diversos ramos de instrução de cada vez. Um processador superescalar explora o que é conhecido como paralelismo no nível de instruções, que diz respeito o nível em que instruções de um programa podem ser executadas em paralelo internamente nas unidades funcionais.

Um processador superescalar busca tipicamente várias instruções de cada vez, e tenta encontrar instruções que sejam independentes umas das outras e que possam, portanto, serem executadas em paralelo.

O paralelismo no nível de instruções existe quando as instruções de uma seqüência são independentes e podem, portanto, ser executadas em paralelo, por sobreposição. O grau de paralelismo no nível de instruções é determinado pela freqüência com que ocorrem as dependências de dados verdadeiras e de desvio (STALLINGS, 2002). Esses fatores, por sua vez, são dependentes da aplicação e da arquitetura do conjunto de instruções (ISA – *Instruction Set Architecture*) do processador.

O paralelismo de máquina é uma medida da capacidade do processador em aproveitar o paralelismo no nível de instruções. O paralelismo de máquina é determinado pelo número de instruções que podem ser buscadas e executadas ao mesmo tempo (número de *pipelines* paralelos) e pela velocidade e eficácia dos mecanismos usados pelo processador para identificar instruções independentes. Tanto o paralelismo no nível de instruções quanto o paralelismo de máquina são fatores importantes para melhoria do desempenho. Um programa pode não ter paralelismo no nível de instruções em grau suficiente para tirar total proveito do paralelismo de máquina. Contudo, o uso de uma arquitetura com um conjunto de instruções de tamanho fixo, como em computadores com arquiteturas RISC, se observa o aumento do paralelismo no nível de instruções. Por outro lado, um paralelismo de máquina limitado restringe o desempenho, qualquer que seja o programa.

A abordagem superescalar depende da habilidade de executar várias instruções em paralelo, por isso diz-se que o paralelismo é do nível de instruções. Este modelo é amplamente utilizado nas arquiteturas RISC, VLIW e EPIC (*Explicity Parallel Instruction Computing*), pois aproveita a característica das instruções de tamanho fixo e garante um desempenho superior às arquiteturas CISC (*Complex Instruction Set Computer*).

2.1.9. Memória Cachê

A memória cachê é utilizada pelos processadores como um repositório rápido das informações durante a execução dos programas, tanto para as instruções quanto para os dados (STALLINGS, 2002), reduzindo significativamente o número de acessos à memória principal.

A memória cachê é mais rápida do que a memória principal. Esta diferença entre a memória cachê e a memória principal está no tipo de *chips* utilizados, pois a memória principal utiliza células de memória dinâmica (maior quantidade e mais lenta) enquanto que a memória cachê utiliza células estáticas (menor quantidade e mais rápida). Isto está atrelado

ao custo, pois a memória dinâmica é mais barata que a memória estática. Além da célula de memória ocupar um espaço físico menor, permite uma maior integração resultando em uma redução do espaço físico ocupado pela memória na placa-mãe contribuindo na redução do tamanho dos computadores.

A memória cachê é um recurso amplamente utilizado nos processadores, pois o ganho no desempenho é significativo e têm feito com que os projetistas de processadores incorporassem diversos níveis de cachê: nível 1 – *on-chip* - interno ao processador (instruções e dados), nível 2 - *on-board* na placa-mãe (dados) e nível 3 – *slot* do processador (dados).

A forma como os programas são elaborados também afeta o funcionamento da memória cachê. Quanto maior o número de acessos à memória principal, para buscar os dados que não estão presentes na memória cachê, maior será o dispêndio de tempo, e conseqüentemente menor será o desempenho. Uma solução é a adoção de um número pequeno de variáveis e que implicará em uma menor quantidade de posições no cachê utilizadas para este armazenamento.

2.1.10. Execução Fora de Ordem

O processador deve também ser capaz de identificar paralelismo ao nível de instruções e coordenar a busca, decodificação e execução de instruções, em paralelo. Desta forma, uma das possibilidades de se explorar este paralelismo é iniciar e terminar a execução das instruções fora de ordem (em inglês *Out-of-Order*) (HINTON, 2001). Este recurso permite que o processador execute instruções sem seguir a ordem original do programa, desde que não ocorra dependência e conflito entre as instruções.

A execução fora-de-ordem possibilita um incremento no desempenho, já que a execução das instruções necessita de um número menor de ciclos de máquina. Entretanto, o software tem que ter um paralelismo que permita que as unidades estejam ao máximo ocupadas e isso depende da aplicação e também do compilador.

Este tipo de execução é de extrema importância para as máquinas superescalares, por possibilitar um melhor aproveitamento deste recurso e por conseqüência em um melhor desempenho.

Este recurso tenta na medida do possível obter o maior desempenho em uma arquitetura superescalar, em virtude de executar o maior número de instruções por ciclo de máquina, contornando as dependências entre instruções do programa. Isto implica que, independentemente do software, o processador maximiza o desempenho com a execução das instruções fora-de-ordem, antecipando as instruções independentes e executando-as simultaneamente nas unidades funcionais.

2.1.11. *Hyper-Threading*

Os estudos preliminares da Intel têm demonstrado que computadores que utilizam a tecnologia *Hyper-Threading*, que foi incorporada no Pentium 4, obtêm um acréscimo de 30% no desempenho quando comparado com um computador que não habilita esta tecnologia (BARON, 2002). A tecnologia *hyper-threading* faz com que o processador consiga trabalhar simultaneamente com mais informações, aumentando a velocidade de

execução dos sistemas operacionais e aplicativos que foram escritos para funcionar com ela. Esta tecnologia permite que um único processador físico consiga executar concorrentemente duas ou mais seqüências de códigos diferentes (*threads*) (INTEL, 2003), como mostrados na Figura 2.

A Figura 3 mostra dois desenhos nos quais podemos visualizar um sistema mono processado com a tecnologia *hyper-threading* (a) e um sistema dual tradicional com dois processadores sem *hyper-threading* (b). Na Figura 3 percebe-se, claramente, que no primeiro desenho temos um processador com *hyper-threading* tem duas máquinas lógicas IA32 (INTEL, 2003), denominadas de AS(*IA32 – Architectural State*) executando sobre um único processador físico. Enquanto, que no segundo desenho, o sistema tem dois processadores físicos cada um com uma única máquina IA32 lógica.

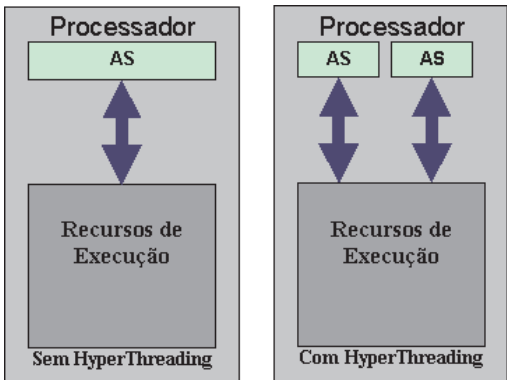

Figura 2. Comparação entre dois processadores, um sem e outro com

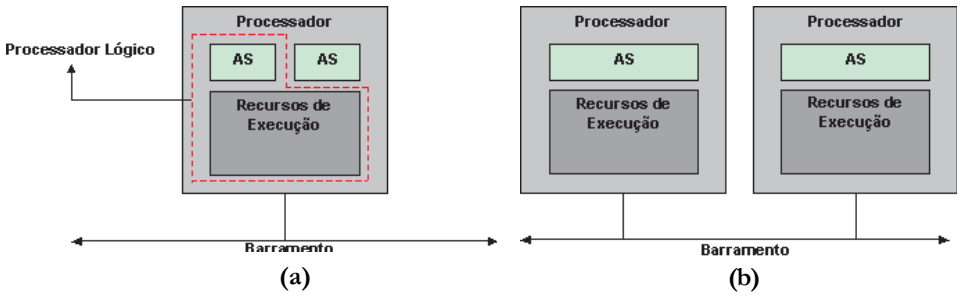

Figura 3. Comparação de um processador com tecnologia *Hyper-Threading* (a) e um Sistema dual tradicional (b) (Fonte: Intel)

A própria Intel confirma que os softwares (sistema operacional e aplicativos) que forem escritos para funcionar com a tecnologia *hyper-threading* conseguem um aumento no desempenho. Conforme informações da Intel os novos *chips* conseguiriam um aumento de desempenho de aproximadamente 25% com o Windows XP, pois ele já possibilita a utilização da tecnologia *hyper-threading* (MARTELL, 2002).

As configurações *hyper-threading* necessitam que o *chipset* e a BIOS (*Basic Input/Output System*) utilize esta tecnologia, e também um sistema operacional que inclua otimizações para a tecnologia *Hyper-Threading* (Intel, 2003) no seu código, permitindo aproveitar este recurso apropriadamente.

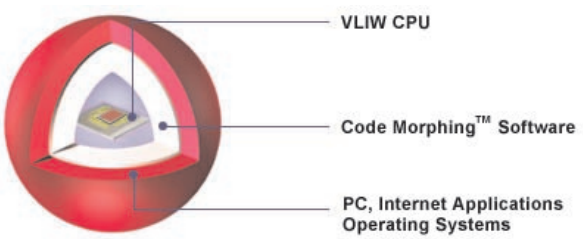

Figura 4. Ilustração do *Code-Morphing* (Fonte: Transmeta)

2.1.12. *Code-Morphing*

O *Code-Morphing* é uma camada de software que está entre o processador físico e a camada de software, e faz a translação do código x86 para um conjunto de instruções para a máquina VLIW, que é o centro do processador Crusoe da Transmeta (KLAIBER, 2000). Esta camada de software fica na memória do microprocessador e a translação é feita em tempo de execução.

Este recurso permite que uma arquitetura VLIW, do processador Crusoe da Transmeta, consiga executar os programas que estão no código x86, e isto garante a compatibilidade em termos de software, mas há uma perda de desempenho, mesmo que esta perda seja pequena. Contudo, este recurso evita o processo de re-compilação dos códigos para a nova arquitetura, reduzindo os custos, o tempo deste processo e a independência do programador.

3. A MEMÓRIA PRINCIPAL

Analisando a memória principal de um computador tradicional, vemos que atualmente a placa-mãe, por exemplo para o Pentium III, suporta fisicamente até 4 pentes DIMM (*Dual In-Line Module Memory*). Se verificarmos no mercado a maior capacidade dos pentes de memória DIMM, atualmente, é de 512Mbytes, logo com os 4 pentes temos a capacidade máxima de 2Gbytes de memória RAM (*Random Access Memory*) física na placa-mãe. Contudo o processador Pentium III, por exemplo, suporta 32 bits de endereços físicos, alcançando, desta maneira, uma capacidade de 4Gbytes de memória física, que é muito além do máximo disponível na placa-mãe.

Se olharmos para a questão software, por exemplo, o sistema operacional que gerencia a memória, a situação fica ainda pior, pois o Windows 98/ME, por exemplo, gerencia no máximo 128Mbytes de memória física (LOPES, 2001). Neste caso, a placa-mãe suporta 2 vezes menos memória, e o sistema operacional gerencia 16 vezes menos, do que o processador acessa fisicamente. Este fato representa uma sub-utilização da potencialidade do processador no que se refere à memória.

Isto demonstra que não se está aproveitando a potencialidade do processador. Neste caso particular da memória isto não é crucial, dependendo da quantidade de memória que o software requer. Na maioria das vezes o usuário tem uma quantidade de memória superior à sua necessidade computacional. Contudo, nas situações onde a exigência computacional é grande, isto deve ser observado.

4. O COMPILADOR

O compilador é um componente de suma importância, tanto que a própria Intel, um dos maiores fabricantes de processadores, inclui um capítulo específico (*Chapter 6 – Suggestions for chossing a Compiler*) (INTEL, 1997), na documentação técnica dos processadores, sobre a escolha do compilador, para que se possibilite um melhor aproveitamento da arquitetura do processador. Entretanto, a maioria dos programadores desenvolve seus programas sem observar estes fatores, por utilizarem o compilador da linguagem que dispõem e que não necessariamente é o mais apropriado para aquele computador, para o qual está se desenvolvendo o programa.

Os programas gerados por compiladores diferentes produzem códigos, normalmente, diferentes e não somente em quantidade de bytes e da seqüência de instruções, mas também em desempenho, isto em decorrência da lógica intrínseca aos compiladores.

Para que um compilador possa gerar um código executável com um bom desempenho, é necessário que o desenvolvedor do compilador domine a arquitetura do processador, para poder utilizar as instruções e os recursos de maneira apropriada.

Muitos programadores crêem que basta digitar o código fonte em uma linguagem, mandar compilar e tudo acontecerá como o previsto, ou seja, o programa estará pronto para ser executado no computador e tudo funcionando corretamente. Em parte isto é verdadeiro, pois o programa executará, mas não necessariamente aproveitando adequadamente os recursos do processador e do computador. Os processadores podem ser muito diferentes, apesar de possuírem um núcleo comum em termos da arquitetura, como é o caso Pentium 4 e o 80486 que possuem o código x86. Isto ocorre com muita freqüência, pois o compilador é desenvolvido para uma arquitetura ISA (*Instruction Set Architecture*) (TANENBAUM, 2001) que compõe uma família de processadores, como é o caso da x86 da Intel, que integra todos os processadores até o Pentium 4. Contudo, a cada novo processador, temos algumas dezenas de novas instruções para possibilitar o acesso aos recursos presentes nos novos processadores.

O termo "*for Windows*" é aplicado erroneamente. O correto seria dizer que o software é "*for x86 Intel*", em virtude de que o sistema operacional Windows também é compilado para o nível ISA da família de processadores x86 da Intel. Se analisarmos os recursos dos processadores atuais da Intel, como exemplo o Pentium 4, comparativamente com o que havia disponível no início da arquitetura x86, percebe-se que os softwares, compilados para o x86 básico, aproveitam muito pouco dos recursos atuais, como é o caso das unidades MMX (*MultiMídia eXtensions*) (INTEL, 1997), SSE (*Streaming SIMD Extensions*) (INTEL, 2003) e outras. Desta forma, não estamos utilizando a potencialidade deste processador, em termos de aumento no processamento, por não utilizar as instruções especificas da arquitetura para esta finalidade.

6. AS MÁQUINAS PARALELAS

A redução do custo das máquinas paralelas, tanto as multiprocessadas quanto as multicomputadoras (clusters), possibilitou a sua utilização em aplicações que até pouco tempo eram pouco prováveis, como exemplo nas pequenas empresas, onde o volume de informações aumentou muito, exigindo desta forma um maior poder, em termos de capacidade, de processamento.

Uma realidade atual é utilização das máquinas duais (com dois processadores), e outra possibilidade é a utilização dos clusters baseados em computadores pessoais. Atualmente elas não implicam em um custo tão elevado de hardware, já que os preços dos computadores reduziram bastante nos últimos anos.

Entretanto, o software para estas máquinas não acompanhou esta tendência, pois não estão sendo desenvolvidos softwares paralelos, com exceção de alguns específicos como é o caso dos softwares científicos.

Estes softwares têm um grande interesse por parte da comunidade científica em decorrência da maior necessidade computacional que os algoritmos científicos exigem. Este fato deve mudar nos próximos anos, pois o mercado das máquinas paralelas está em crescimento e o software deverá atender esta nova tendência. Além disso, cada vez mais

necessitamos de poder de processamento que um único computador não consegue mais fornecer, tornando esta migração para o modelo paralelo um caminho certo.

6. O DESEMPENHO

A grande maioria dos usuários substitui o computador antes mesmo de ter conseguido aproveitar a potencialidade máxima. Isto ocorre, basicamente, por uma questão de marketing, pois se cria uma falsa expectativa de que o novo computador terá um desempenho melhor e isto não necessariamente se confirma na prática. Entretanto, isto é verdadeiro na maioria dos casos, mas pelos motivos errados, pois os softwares continuam os mesmos e o computador tem um melhor desempenho devido o aumento da freqüência de operação do processador, além de outros fatores como o aumento da memória (principal e cachê) e etc. Desta forma, isto acontece sem um aproveitamento adequado dos recursos disponíveis, e se o software fosse realmente adequado à máquina poderia resultar em um desempenho muito melhor, sem acarretar em alterações no hardware.

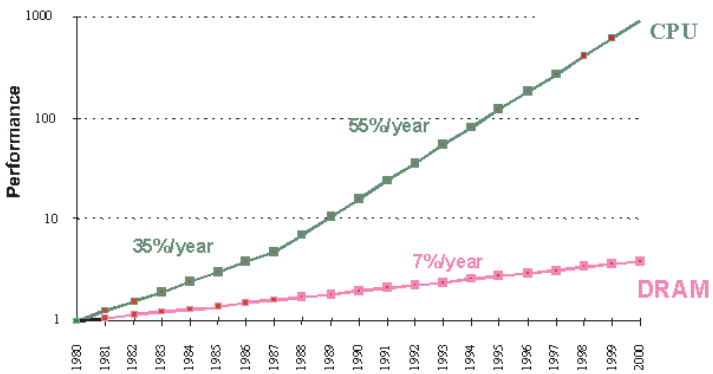

Figura 5. Evolução da performance nas últimas duas décadas. (Fonte: YOON, 2002)

Na Figura 5, obtida em (YOON, 2002), percebe-se claramente que a performance dos processadores (CPU-*Central Processing Unit*) cresceu a uma taxa de 55% ao ano, a partir de 1987 até 2000, enquanto que a memória DRAM (*Dynamic Random Access Memory*) atingiu um crescimento de no máximo 7%. Esta diferença atribui-se aos recursos tecnológicos incorporados nos processadores em decorrência da tecnologia de fabricação VLSI (microeletrônica), fazendo com que se reduzisse significativamente o tamanho dos componentes.

Com base no gráfico da Figura 5, contata-se que a taxa de crescimento do desempenho dos processadores até o final da década de 80 foi de 35% ao ano, sendo basicamente em decorrência da tecnologia de fabricação, com equipamentos de maior precisão, novos materiais semicondutores e etc. A partir da década de 90 a taxa se elevou para 55%, pois além da tecnologia de fabricação, houve também um ganho ocasionado pelos novos recursos computacionais com a alteração da arquitetura dos processadores tanto internas (unidades funcionais) quanto externas (memórias e barramentos).

Entretanto, os processadores atuais conseguirão obter desempenho ainda melhor, se o software explorar o paralelismo interno do processador, pois eles são desenvolvidos para que se explore este recurso computacional, podendo chegar à execução de uma dezena de instruções por ciclo de máquina.

Com os modernos computadores faz-se necessário uma melhor interface hardware/software, para que se consiga alcançar ao máximo o desempenho da arquitetura. Neste

ponto surge um fator determinante que é o software, e que ainda é a principal influência na redução do desempenho dos sistemas computacionais. Na maioria das vezes ele não está em concordância com o hardware, mais especificamente com o processador, não aproveitando os recursos deles, o que poderia representar um desempenho superior ao que se está obtendo atualmente.

7. CONCLUSÃO

Algumas perguntas surgem com base neste artigo e tentaremos respondê-las na medida do possível. Contudo, este trabalho não tem por objetivo ser um roteiro de procedimentos que esclareçam estas dúvidas por completo, mas tem o propósito de fomentador de novas idéias e conceitos para podermos tratar com estas questões.

A primeira questão é como fazer para que o software se aproxime mais do hardware? Sabemos que o software está atrelado à lógica de quem o implementa, e por este motivo temos softwares que apesar de fazerem as mesmas tarefas, são implementados por meio de seqüências de instruções diferentes, pois as pessoas raciocinam com diferentes conhecimentos na solução de problemas implicando em uma heterogeneidade em termos do software. Uma das formas para se reduzir a lacuna entre o hardware e o software é ter o código binário específico para a arquitetura, e isto faz com que ele consiga utilizar eficientemente os recursos do computador. Contudo, para que isto aconteça, dois itens são fundamentais: o código aberto e o compilador próprio para o computador. O primeiro item pode ser superado com a utilização de programas no estilo *open-source*; já o segundo é um pouco mais trabalhoso, por não dispormos de tantos compiladores quanto o número de processadores, e, isto já implica em não conseguirmos extrair o melhor proveito do computador. Este item é possível para desenvolvedores (empresas e programadores) de soluções computacionais, mas não é viável para os que não desenvolvem softwares, que não dispõem de ferramentas e pessoal para esta tarefa de re-escrever e recompilar o código.

Outra questão é como fazer com que usuários, que utilizam produtos prontos e empacotados, consigam aproveitar da melhor maneira possível o computador, obtendo o melhor custo/benefício, já que eles não conseguem compilar para este computador? Neste caso, a melhor solução é a utilização de softwares baixados a partir da base de programas do fornecedor, o que possibilita baixar uma versão do software que seja a mais adequada para o computador. Esta sugestão está baseada na idéia de um servidor de software, ou até mesmo que o software seja atualizado periodicamente, obtendo a versão mais condizente com a arquitetura do computador. Contudo, não há nenhuma garantia que o desempenho será ou não melhor, mas com toda a certeza, os problemas serão minimizados e o poder computacional será melhorado.

Deste fato surge uma outra pergunta: Como saber qual o computador? Para esta pergunta a resposta é simples, pois há diversas maneiras de se obter as informações sobre o processador, e uma é, por exemplo, utilizando a instrução CPUID (INTEL, 2004) e (AMD, 2002), presente nos computadores IA-32 da Intel e da AMD.

Uma última questão é como obter um melhor desempenho? Uma das maneiras simples de se alcançar um melhor desempenho, é recompilar os programas para as novas arquiteturas dos processadores, utilizando-se para isto os compiladores próprios para a arquitetura. Apesar do mecanismo de compilação ser simples ele não é trivial, por não dispormos do programa

fonte e nem sempre temos o compilador apropriado para o processador do computador de destino. Desta maneira, estamos à *mercê* dos grandes desenvolvedores de software, ou em função do código dos programas, ou mesmo em decorrência das versões do software, tanto os aplicativos quanto o próprio sistema operacional.

Todos os softwares são influenciados pelo hardware, mais precisamente pelo processador, e por sua vez influenciam o desempenho dos computadores. Quando se aborda o termo software é preciso ter em mente que abordamos os programas aplicativos, sistema operacional e o *firmware* (BIOS). Os softwares são influenciados na medida em que devem ser escritos para uma arquitetura, com base nas instruções (nível ISA) do processador, acessando os recursos tecnológicos disponíveis. Por outro lado, o software influencia o desempenho na medida em que pode limitá-lo, devido a não utilização de forma eficiente dos recursos das arquiteturas dos processadores. Com isso, o melhor desempenho está atrelado a uma ótima configuração do hardware e o software compatível com esta arquitetura, mais precisamente com o processador. Salienta-se que tanto o hardware quanto o software dependem fortemente da aplicação e da quantidade de recursos necessários para a execução da tarefa, e está ligado com a maneira como se desenvolve a programação do computador por parte do programador.

AGRADECIMENTOS

Os mais sinceros agradecimentos ao Prof. Marcos Augusto Hochuli Shmeil pela revisão e valiosa contribuição neste trabalho.

REFERÊNCIAS BIBLIOGRÁFICAS

AMD Corporation. **Processor recognition**. 20734. November 2002.

BARON, Max. **Technology 2001 –** On a Clear Day You Can See Forever. **Microprocessor Report.** Microdesign Resourses. Feb. 25, 2002.

ENDERLEIN, Rolf. **Microeletrônica**. São Paulo: EDUSP, 1994.

FOG, Agner. Branch prediction in the pentium family. **Dr. Dobb´s Journal.** 2003.

HINTON, Glenn et al. The microarchitecture of the pentium 4 processor. **Intel Technology Journal,** Q1, 2001.

HYDE, Randall. **The art of assembly language**. No Starch Press. 2003.

IBM Corporation. **PowerPC 740/750: RISC microprocessador user´s manual**. 1999.

INTEL Corporation. **Intel architecture optimization manual**. Order 2422816-003. 1997.

INTEL Corporation. **IA-32 Intel architecture softwares developer´s manual.** Volume 1: Basic Architecture. 2003.

INTEL Corporation. **Intel processor identification and the CPUID instruction.** 241618-025. 2004.

KLAIBER, Alexander. **The tecnology behind crusoe processor.** Transmeta Corporation. January 2000.

LILJA, DAVID. Reducing the branch penality in pipelined processors. **Computer,** Jul. 1988.

LOPES, Airton et al. 65 macetes para ganhar velociadade. **Info Exame.** Edição 189, dezembro 2001.

MARTELL, Duncan. **Intel antecipa novas tecnologias para chips**. Disponível em: <http://cgi2.uol.com.br/cgi-bin/info/print.cgi > acesso em: 10/09/2002.

PATTERSON, David A. & Hennessy, John L. **Organização e projeto de computadores: a interface hardware/software**. 2. ed. Rio de Janeiro: LTC, 2000.

PHILIPS Semiconductors. **Introduction to VLIW computer architecture.** 9397-750-01759. Disponível em: <www.semiconductors.philips.com/acrobat/other/vliw-wp.pdf> Acesso em: 2003.

STALLINGS, Willian. **Arquitetura e organização de computadores.** 5. ed. São Paulo: Prentice Hall, 2002.

TANENBAUM, Andrew S. **Organização estruturada de computadores**. 4. ed. Rio de Janeiro: LTC, 2001.

YOON, Hyunsoo. **CS510: computer architectures**. Lecture. Mensagem recebida por: <hyoon@cs.kaist.ac.kr> em: 2002.