SISTEMAS OPERACIONAIS

PROFESSOR: ANTÔNIO AUGUSTO M. FRÖHLICH

ALUNO: EDSON SORATO

MICROARQUITETURAS

Em engenharia da computação, microarquitetura que algumas vezes é abreviado como µarch ou uarch, é a descrição de um circuito elétrico de um computador, uma unidade central de processamento ou um processador digital de sinais, que é suficiente para descrever completamente a operação de hardware.

Desde 1950 muitos computadores utilizam multiprogramação para implementar seus controles lógicos que decodificam as instruções que executam. O termo microarquitetura foi usado para descrever as unidades que eram controladas pelo microprograma.

A microarquitetura está relacionada com a arquitetura do set de instruções porém não é a mesma coisa. Grosseiramente falando, a arquitetura do set de instruções é o mesmo que o modelo de programação do processador do ponto de vista de um programador de linguagem assembly ou de um projetista de compilador. Inclui o modelo de execução, os registradores do processador, formatos de endereços e dados, etc. A microarquitetura é basicamente a estrutura de baixo nível que comanda os detalhes que estão escondidos do modelo de programação. Ela descreve as partes constituintes do processador e como estas se interconectam e operam entre si para implementar a especificação da arquitetura.

A microarquitetura de uma máquina é usualmente representada como diagramas que descrevem as interconexões dos diversos elementos da microarquitetura da máquina. O circuito eletrônico que implementa estes elementos é chamado de implementação da microarquitetura. Elementos de microarquitetura podem ser desde gates simples, registradores, LUTs, multiplexadores, contadores, etc até ALUS e elementos maiores.

Uma descrição muito simplificada de alto nível simplesmente mostra características como largura do barramento, o número de unidades de execução e seus tipos, blocos como branch prediction, cache memories, etc. Alguns detalhes pertinentes a estrutura do pipeline (como fetch, decode, assign, execute, write-back) podem ser incluídos também.

Pontos importantes:

- Uma microarquitetura simples, especialmente se ela incluir microcódigo, pode ser usada para implementar muitos diferentes sets de instruções através da troca do controle de armazenamento. Contudo isso é um pouco complicado e desajeitado, mesmo quando simplificado pelo microcódigo e/ou tabela de estruturas em ROMs ou PLAs.
- Duas máquinas podem ter a mesma microarquitetura e, desta forma, o mesmo diagrama de blocos. Mesmo assim podem ter implementações de hardware totalmente diferentes.
- Máquinas com microarquiteturas diferentes podem ter a mesma arquitetura do set de instruções e desta forma, executar os mesmos programas. Novas microarquiteturas e/ou soluções de circuito, juntamente com os avanços nos

processos de fabricação do semicondutor, é que permitem novas gerações de processadores alcançarem performance mais alta.

Decisões de projeto a nível de sistema (por exemplo, decidir se periféricos como controladores de memória devem ou não ser incluídos), podem ser consideradas parte do processo de projeto da microarquitetura. Estas decisões envolvem níveis de performance e conectividade destes periféricos.

Diferentemente de projeto de arquitetura, onde obter um nível de performance específico é o principal objetivo, o projeto de microarquiteturas se concentra mais em outras restrições. Uma vez que as decisões de projeto de microarquitetura afetam o que vai no sistema, a atenção se volta para aspectos como:

- Área/custo do chip
- Consumo de energia
- Complexidade da lógica
- Aspectos construtivos
- Conectividade
- Facilidade para debugar
- Testabilidade

Dada a importância que as microarquiteturas têm na performance da máquina, muitos esforços são dispensados no sentido de se obter novas e mais eficientes microarquiteturas. Não muito depois que os avanços na área de manufatura do chip permitiu que mais circuito fosse embutido no die, os projetistas buscaram formas de utilizar o estes novos recursos. Um dos recursos mais comuns foi a adição de mais memória cachê no die. A memória cachê é simplesmente uma memória muito rápida e que pode ser acessada em poucos ciclos de máquina em oposição aos muitos ciclos necessários para acessar a memória principal. A CPU possui um controlador de cachê que automatiza o processo de escrita e leitura na memória cachê. Projetos com RISC introduziram a memória cachê em meados de 1980. Atualmente, memórias cachê de diversos MBytes e de múltiplos níveis são encontradas nas CPUs mais modernas. O surgimento das memórias cachê resolveu o problema de utilização do pipeline. Inicialmente não fazia muito sentido desenvolver pipeline que podiam rodar mais rápido do que o tempo de latência da memória externa ao chip. Ao incluir a memória cachê no chip foi possível fazer o pipeline rodar no tempo de latência de acesso da memória cachê. Isso permitiu que o a freqüência de operação do processador crescesse muito mais rápido do que a da memória externa.

Outra barreira para o aumento de performance eram os stalls flushes do pipeline causados por saltos. Normalmente somente no momento de se executar a instrução é que o processador descobre se um salto condicional deve ou não ser executado. Desta forma, desde o momento em que o processador de instrução encontrou uma instrução de salto condicional até o momento que o valor do registrador de decisão poder ser lido, o pipeline sofreu vários ciclos de stall. Na média um quinto das instruções executadas são de salto o

que leva a um número grande de stall no pipeline. Se o salto deve ser executado então as instruções seguintes ao salto que estavam no pipeline devem ser descartadas. Técnicas como predição de saltos e execução especulativa são utilizadas para diminuir estas penalidades de saltos. Na predição de saltos o hardware faz "previsões" ponderadas sobre se o salto vai ou não ser executado. Estas "previsões" permitem ao hardware realizar um prefetch das instruções sem esperar pelo valor do registrador. No caso da execução especulativa o código subseqüente ao salto é executado antes mesmo de saber se o salto deve ser realizado ou não.

Quando a indústria de semicondutores forneceu ainda mais recursos de hardware, os projetistas introduziram os processadores superescalares. Basicamente os processadores superescalares alcançam melhor desempenho pela execução simultânea de instruções. Isto é obtido a partir da duplicação de unidades funcionais como ALUs que só foi possível quando a área do die em um único chip aumentou.

Quando os arquitetos de sistema foram "barrados" pela estabilização da freqüência de operação das CPUs e pelos tempos de acesso das memórias DRAM, procuraram por uma forma de explorar níveis de paralelismo que existem fora do programa. Uma técnica de explorar estes níveis de paralelismo é através de sistemas de multiprocessamentos utilizando computadores com múltiplas CPUs.

Outra técnica que se tornou popular é o multithreading. Nesta abordagem, quando o processador tem que fazer uma busca em um sistema de memória lento ao invés de parar (realizar um stall) até que o dado chegue, faz um chaveamento para outro programa que está pronto para executar. Desta forma, não acelera a execução de um programa particular mas aumenta o throughput do sistema e reduz o tempo de CPU em idle. Conceitualmente, multithreading é equivalente a chaveamento de contexto no nível do sistema operacional. A diferença é que uma CPU multithread pode fazer o chaveamento em um ciclo de CPU ao invés das centenas ou milhares de ciclos de instruções necessários para realizar um chaveamento normal. Isto é conseguido pela replicação do estado do hardware (como registradores e program counter) para cada thread ativa.

TRABALHOS E ESTUDOS NA ÁREA

Ravi Bhargava, Juan Rubio, Srikanth Kannan e Lizy K. John conduziram um estudo sobre os efeitos da atividade do sistema operacional e do chaveamento de contexto nas microarquiteturas. Muitos dos estudos das avaliações de desempenho das arquiteturas de computadores dizem respeito basicamente a simulação de um fluxo dinâmico de instrução de uma aplicação particular. Os testbenchs utilizados geralmente se ocupam mais da CPU e muito pouco do sistema operacional. A atividade do sistema operacional e o chaveamento de contexto são ignorados porque os simuladores populares não comportam a complexidade adicional.
Ravi et all realizaram uma análise de aplicações rodando no Microsoft Windows NT usando um processador x86. Estruturas de microarquiteturas como caches de dados e instruções, TLB e branch predictor foram analizados em detalhes.

A conclusão que este grupo de trabalho chegou é que aplicações desktop e de base de dados têm uma atividade de sistema operacional maior do que a apresentada pelos testbench (o valiado foi o SPEC2000). Da mesma forma, aplicações de desketop e base de dados são mais sensíveis a chaveamento de contexto e mudança do espaço de endereço devido a natureza iterativa com o usuário e o sistema operacional. Aplicações que não tem carga de processamento intenso podem sofrer uma maior degradação de desempenho devido ao código do sistema operacional e código de outros espaços de endereço. Estes códigos que não são da aplicação poluem o estado da microarquitetura. Contudo os BenchMarck SPEC não são afetados por esse códigos de maneira similar a menos que sejam executados simultaneamente com outros programas.

Outra observação é que se a maioria das instruções são compostas por intervalos no espaço de endereço grandes, a atividade do sistema operacional e o chaveamento de contexto são irrelevantes. Por outro lado, se pequenos intervalos formam a maioria dos intervalos como é o caso em database e aplicações desktop, então o período de warmup dos preditores e das caches se tornam um fator limitante da performance.

Além disso, observaram que a taxa de perda da cachê de instruções das aplicações desktop e database é pior do que nas aplicações do SPEC2000. Em ambos os casos, a adição do código do sistema operacional e espaços de múltiplos endereços afetaram a taxa de perda da cachê de instruções. Contudo, foi observado que como as aplicações SPEC2000 não perdem tão frequentemente a cachê de instruções, o impacto no desempenho global não foi significante como nas aplicações desktop e database.

O preditor de saltos também pode ser influenciado negativamente pelo sistema operacional e pelas aplicações múltiplas. Esta influência é muito maior nas aplicações desktop e database do que nas aplicações SPEC2000. Saltos condicionais ocorrem mais frequentemente tanto no código do sistema operacional quanto no código da aplicação. Saltos indiretos são mais comuns no sistema operacional do que na aplicação. Não considerar o código do sistema operacional pode aumentar a performance em aplicações

desktop e database quando as estruturas do preditor de saltos usadas forem pequenas.

A conclusão final de Ravi et all é que o impacto do código do sistema operacional e do comportamento das multi-aplicações é específico para a aplicação. Desta forma o impacto na microarquitetura e a inclusão de pesquisa de simulação depende do worload alvo. Disto se conclui que, se os melhoramentos na microarquitetura forem propostos para workload similar ao de aplicações desktop e database, é imperativo que estes melhoramentos sejam avaliados com ferramentas e simuladores que considerem a atividade do sistema operacional.

Hugo Marcondes, Arliones Stevert Hoeller Junior, Lucas Francisco Wannere Antônio Augusto Fröhlich propuseram uma interface altamente portável para aplicações embarcadas. Uma das principais dificuldades encontradas em um sistema deste tipo é o fato de que a plataforma de hardware é bastante específica e voltada para aplicação em questão.
Uma das estratégias para resolver este problema é o uso de interfaces de chamadas de sistema como o POSIX, WIN32 e MOSI.
Outras duas estratégias utilizadas para conseguir portabilidade em sistemas embarcados são máquinas virtuais e abstração de hardware. As máquinas virtuais oferecem um bom nível de portabilidade mas no entanto acarretam em um grande overhead de processamento e memória o que restringe o uso em sistemas embarcados.
A abstração de hardware (ECOS, LINUX, WINDOWS) apresentam a tendência de incorporar características arquiteturais da plataforma na qual foram concebidas o que limita a portabilidade.
Hugo et all propuseram um sistema operacional baseado em componentes e orientado a aplicação que é portável para a aplicação. Entre outros recursos utilizados para obterem o desempenho desejado destaca-se a programação orientada a aspectos a meta-programação estática em um ambiente orientado a objeto. Também foram utilizados mediadores de hardware o que permitiu que o mesmo sistema executasse em plataformas distintas.
A análise de domínio utilizada pelo grupo permitiu atingir o nível de portabilidade desejado. Diversas variações e semelhanças foram identificadas no domínio de sistemas operacionais para sistemas embarcados. Exemplos são: políticas de escalonamento, sincronizadores (mutex, semáforos e variáveis de condição), temporização, mecanismo de gerenciamento de memória (paginação, segmentação, ambas ou nenhuma), tratamentos de interrupção, tratamento de rotinas de entrada e saída (mapeamento em memória, portas IO, DMA, PIO). Para garantir portabilidade é desejável que essas características sejam configuráveis no sistema operacional o que faz com que este seja adaptável ao hardware e a aaplicação.
A metodologia de Projeto de Sistemas Orientados a Aplicação (AOSD – Application Oriented System Design) foi utilizado para realizar a análise de domínio. O AOSD propõe o uso de diversas técnicas de modelagem e programação para atingir um alto nível de configurabilidade com overhead de processamento e memória mínimo.
O resultado desta análise foi um sistema operacional orientado a aplicação chamado EPOS.
Na estrutura apresentada todas as unidades de hardware dependentes da arquitetura foram abstraídas através de artefatos denominadas de mediadores de hardware. Estes mediadores de hardware exportam toda a funcionalidade necessária para um nível mais alto do sistema operacional. Essas abstrações correspondem a serviços tradicionais como gerenciamento de memória, gerenciamento de processos, comunicação entre processos, etc.

Dois estudos de caso foram feitos para comprovar o funcionamento do sistema proposto: um sistema de controle de acesso utilizando um microcontrolador AVR ATmega16 e um multiplexador MPEG-2 que utiliza um PowerPC 45 da IBM.

A utilização de uma correta metodologia de análise e técnicas de programação modernas foram essenciais para o resultado positivo da pesquisa. Essas técnicas possibilitaram um contrato de interface entre abstrações de sistemas portáveis e mediadores de hardware. O sistema proposto facilita o reuso e reduz o time-to-market.

REFERÊNCIAS:

- Wikipédia

- UNDERSTANDING THE IMPACT OF X86/NT COMPUTING ON MICROARCHITECTURE. Ravi Bhargava, Juan Rubio, Srikanth Kannan, Lizy K. John

- PORTABILIDADE DE SISTEMAS OPERACIONAIS NO DOMÍNIO DE SISTEMAS EMBARCADOS. Hugo Marcondes, Arliones Stevert Hoeller Junior,Lucas Francisco Wanner and Antônio Augusto M. Fröhlich