MC723 - Roteiro do Projeto 3

Murilo Frônio Bássora (RA 094236), Ruan Ramos Monteiro Silva (RA 104034), Daniel Vatanabe Pazinato (RA 116545) e Rafael Faria Castelão (RA 118439)

Neste experimento, vamos implementar um processador *multicore* no simulador do MIPS (8 cores) e testá-lo com um programa que possua uma carga de execução pesada (a decidir ainda). Utilizaremos um software de algum benchmark para sistemas multicores na realização dos testes. Além disso, criaremos um módulo de hardware para acelerar o desempenho do programa de teste. Este módulo de hardware será uma nova instrução do simulador. Como preparação, respondemos a três perguntas teóricas de base:

1. Como um processador multicore é iniciado?

Como base para explicação, vamos usar o ARM Linux SMP. Quando a máquina é iniciada ou é dado um reset, o código do Boot Monitor é executado de um endereço pré definido na memória flash. O Boot Monitor inicializa os periféricos e depois lança o real bootloader U-Boot. Esse inicializa a memória principal e copia a imagem do kernel do Linux comprimido, que fica localizado ou na memória flash, MMC, CompactFlash ou em um host PC, para a memória principal para ser executado. Depois a imagem do kernel é descomprimida, começa a iniciar as estruturas de dados, cria alguns processos de usuário, inicia todos os cores a executa o command shell no user-space. Como o sistema é multicore o Boot Monitor é executado no core principal enquanto as outras CPUs executam a instrução WFI (Wait For Interruption). Quando o kernel é iniciado, a CPU principal envia um sinal para as secundárias para iniciarem suas memórias, caches, MMU e estrutura de dados referente ao kernel.

Segundo a especificação da Intel para os processadores da família P6, todos os processadores no bus system (incluindo um processador único dentro do sistema uniprocessor) executam o protocolo de iniciação de processador único (MP) através do bus APIC. O processador que é selecionado através deste protocolo como o processador iniciador então imediatamente começa a execução do código de inicialização do atual seguimento de código começando no deslocamento do registrador EIP.

2. Como é feito (implementado) o controle de concorrência em hardware?

Os diferentes dispositivos de controles de concorrência utilizados em software (mutex, semáforos) necessitam de uma implementação em hardware para que o computador reconheça qual core tem o lock da região crítica. Esta implementação é possível através de duas instruções garantidas atômicas, o **load-link** and **store-conditional**. O **load-link** copia para um registrador o valor do mutex. Se ele tiver livre (valor no registrador igual a zero) a instrução **store-conditional** muda o valor do mutex para 1 conseguindo o lock. O **store-conditional** só irá funcionar se o valor copiado por **load-link** é o mesmo do mutex, ou seja, nenhuma outra thread conseguiu o lock. Após utilizar o recurso a thread irá liberar a lock colocando zero no mutex.

3. Como escolher o melhor trecho de código para implementar em hardware?

Teoricamente, métodos que usam muitas instruções e que são chamados múltiplas vezes no codigo são candidatos à implementação em hardware. Um exemplo desta técnica é visto em aplicações que usam operações de criptografia, como em criptografia de curvas elípticas. Como o custo de execução das operações com curvas elípticas é critico para o bom funcionamento do software e é normalmente alto, a implementação delas em hardware traz muitos beneficios para o desempenho da aplicação. Saberemos melhor qual trecho de código implementaremos dependendo da aplicação que escolhermos para rodar os testes.

Link do github: https://github.com/danielvatanabe/mc723-lab3