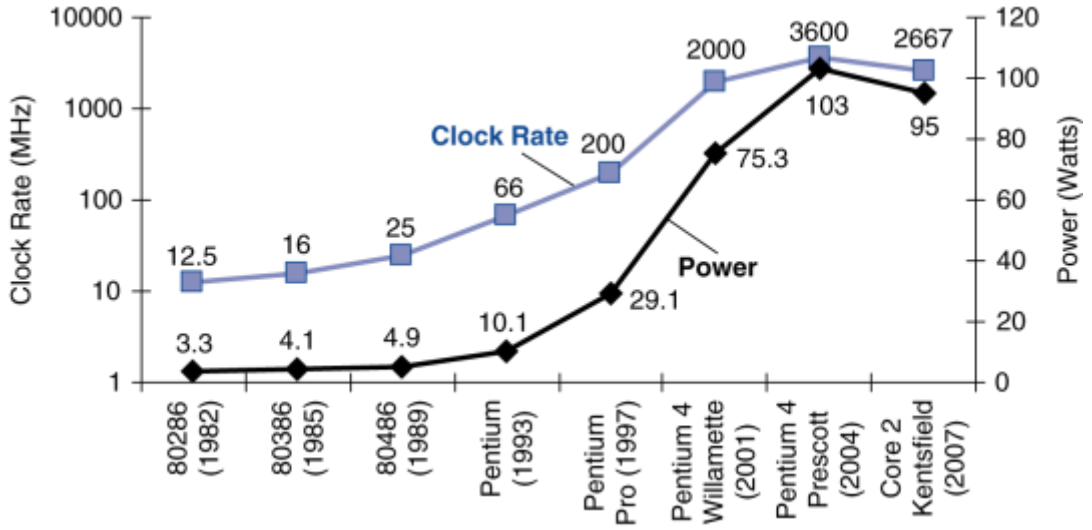
Infraestrutura de Hardware

Processamento Paralelo – Multicores, Multi-Threading e GPUs

Perguntas que Devem ser Respondidas ao Final do Curso

- Como um programa escrito em uma linguagem de alto nível é entendido e executado pelo HW?
- Qual é a interface entre SW e HW e como o SW instrui o HW a executar o que foi planejado?
- O que determina o desempenho de um programa e como ele pode ser melhorado?
- **Que técnicas um projetista de HW pode utilizar para melhorar o desempenho?**

Obstáculo para Desempenho: Potência



- Tecnologia CMOS de circuitos integrados

$$\text{Potencia} = \text{Capacitancia} \times \text{Voltagem}^2 \times \text{Frequencia}$$

× 30

Últimos 25 anos

5V → 1V

× 1000

Reduzindo Potência

- Quanto maior a frequência, maior a potência dissipada
  - Sistema de esfriamento do processador é impraticável para frequências muito altas
  - Ruim para dispositivos móveis que dependem de bateria
- **Power wall**
  - Não se consegue reduzir ainda mais a voltagem
  - Sistema de esfriamento não consegue eliminar tanto calor

Como aumentar então desempenho de uma CPU?

Multiprocessadores

- **Múltiplos processadores menores, mas eficientes trabalhando em conjunto**

- **Melhoria no desempenho:**

  Paralelismo ao nível de processo

  - Processos independentes rodando simultaneamente

  Paralelismo ao nível de processamento de programa

  - Único programa rodando em múltiplos processadores

- **Outros Benefícios:**

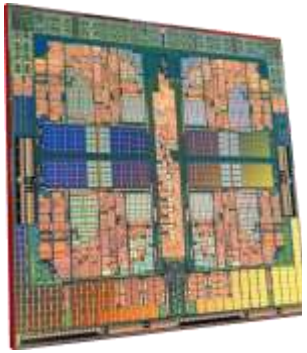
  Escalabilidade, Disponibilidade, Eficiência no Consumo de Energia

Multiprocessadores

- **Microprocessadores multicores**
  - Mais de um processador por chip
- **Requer programação paralela**
  - Hardware executa múltiplas instruções paralelamente
    - Transparente para o programador
  - Difícil de
    - Programar visando desempenho
    - Balancear carga de trabalho

Multicores

- **Microprocessadores multicores**
  - Mais de um processador por chip
  - Cada processador é chamado de core
  - Capacidade de integração de transistor permitiu maior quantidade de cores por chip
- **Cada core pode rodar com frequências menores**
  - Desempenho alcançado com aumento do throughput
  - Menor consumo de energia por core



Processamento Paralelo - Hardware e Software

		Software	
		Sequencial	Concorrente
Hardware	Serial	Multiplicação de matrizes em Java no Intel Pentium 4	Windows Vista no Intel Pentium 4
	Paralelo	Multiplicação de matrizes em Java no AMD Athlon Phenom II	Windows Vista no AMD Athlon Phenom II

- **Software sequencial/concorrente pode executar em hardware serial/paralelo**

  Desafio: fazer uso efetivo de hardware paralelo

Programação Paralela

- Desenvolver software para executar em HW paralelo
- Necessidade de melhoria significativa de desempenho
  - Senão é melhor utilizar processador com único core rápido, pois é mais fácil de escrever o código
- Dificuldades
  - Particionamento de tarefas (Balanceamento de carga)
  - Coordenação
  - Overhead de comunicação

Analisando Desempenho com Multicores

- Podemos analisar a melhoria provocado pelo aumento de cores
- Lei de Amdahl

  Tm = Tempo de Execução com melhoria

  Ta = Tempo de execução afetado pela melhoria

  Qc = quantidade de cores

  Tsm = Tempo de Execução não afetado pela melhoria

$$Tm = Ta/Qc + Tsm$$

Exemplo: Ganho de Desempenho com Multicores

Problema:

Suponha que queiramos fazer dois tipos de soma: somar 10 variáveis escalares e somar duas matrizes 10 X 10. Qual o ganho com 10 cores e com 100 cores? Considere que uma soma leva um tempo t.

Solução:

Se só houvesse um core o tempo total seria 100t (soma de duas matrizes 10 x 10) + 10t (soma de 10 escalares) = 110t

Soma de 10 escalares deve ser sequencial, portanto leva 10t ou seja Tsm = 10t

Soma de 2 matrizes pode ser feita em paralelo.

Para 10 cores:

Tm = Ta/Qc + Tsm = 100t/10 + 10t = 20t

Ganho = 110t/20t = 5,5

Exemplo: Ganho de Desempenho com Multicores

Problema:

Suponha que queiramos fazer dois tipos de soma: somar 10 variáveis escalares e somar duas matrizes 10 X 10. Qual o ganho com 10 cores e com 100 cores? Considere que uma soma leva um tempo t.

Solução:

Se só houvesse um core o tempo total seria 100t (soma de duas matrizes 10 x 10) + 10t (soma de 10 escalares) = 110t

Soma de 10 escalares deve ser sequencial, portanto leva 10t ou seja Tsm = 10t

Soma de 2 matrizes pode ser feita em paralelo.

Para 100 cores:

Tm = Ta/Qc + Tsm = 100t/100 + 10t = 11t

Ganho = 110t/11t = 10

Analisando Resultados

- Com 10 cores, esperava-se ter uma execução 10 vezes mais rápida, ou seja 110t/10 = 11t, porém só foi possível 20t

  Apenas 55% (11/20) do ganho esperado foi atingido

- Com 100 cores, esperava-se execução 100 vezes mais rápida ou seja 110t/100 = 1,1t, porém só foi possível 11t

  Apenas 10% (1,1/11) do ganho esperado foi atingido

Exemplo: Aumentando o Problema

Problema:

Suponha que queiramos fazer dois tipos de soma: somar 10 variáveis escalares e somar duas matrizes 100 X 100. Qual o ganho com 10 cores e com 100 cores? Considere que uma soma leva um tempo t.

Solução:

Se só houvesse um core o tempo total seria 10000t (soma de duas matrizes 100 x 100) + 10t (soma de 10 escalares) = 10010t

Soma de 10 escalares deve ser sequencial, portanto leva 10t ou seja Tsm = 10t

Soma de 2 matrizes pode ser feita em paralelo.

Para 10 cores:

Tm = Ta/Qc + Tsm = 10000t/10 + 10t = 1010t

Ganho = 10010t/1010t = 9,9

Exemplo: Aumentando o Problema

Problema:

Suponha que queiramos fazer dois tipos de soma: somar 10 variáveis escalares e somar duas matrizes 100 X 100. Qual o ganho com 10 cores e com 100 cores? Considere que uma soma leva um tempo t.

Solução:

Se só houvesse um core o tempo total seria 10000t (soma de duas matrizes 100 x 1000) + 10t (soma de 10 escalares) = 10010t

Soma de 10 escalares deve ser sequencial, portanto leva 10t ou seja Tsm = 10t

Soma de 2 matrizes pode ser feita em paralelo.

Para 100 cores:

Tm = Ta/Qc + Tsm = 10000t/100 + 10t = 110t

Ganho = 10010t/110t = 91

Analisando Resultados com o Aumento do Problema

- Com 10 cores, esperava-se ter uma execução 10 vezes mais rápida, ou seja 10010t/10 = 1001t, foi possível 1010t

  99% (1001/1010) do ganho esperado foi atingido

- Com 100 cores, esperava-se execução 100 vezes mais rápida ou seja 10010t/100 = 100,1t, foi possível 110t

  91% (100,1/110) do ganho esperado foi atingido

- Aumento do tamanho do problema resultou em um ganho de desempenho quase proporcional à quantidade de cores

  Balanceamento de carga entre cores também é importante

Difícil é melhorar desempenho com tamanho de problema fixo!

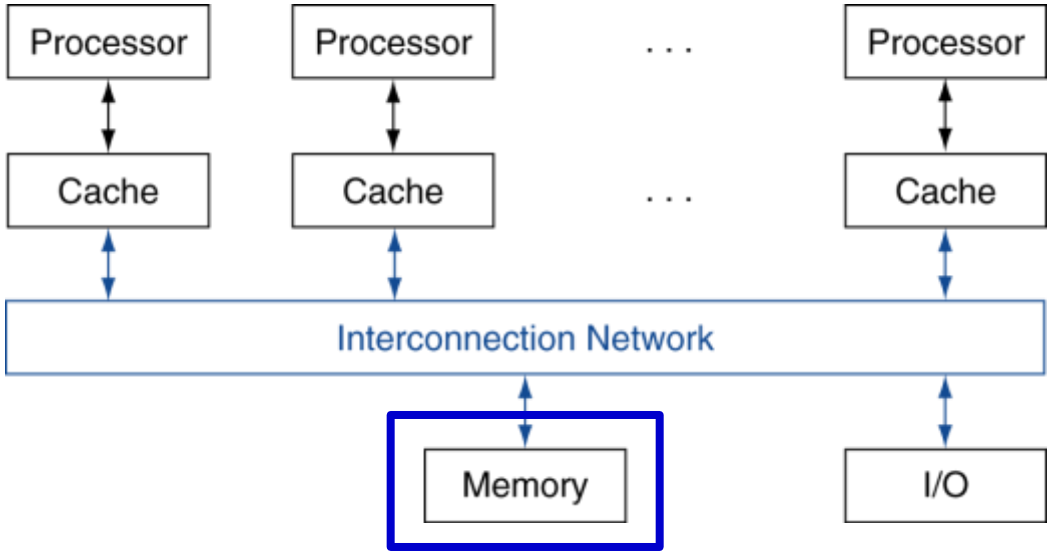
Multicores – Perguntas Importantes

Pergunta 1: Como os diferentes processadores (cores) compartilham dados ?

Pergunta 2: Como é feita a comunicação entre os diferentes processadores ?

Memória Compartilhada

- **Shared memory multiprocessor**

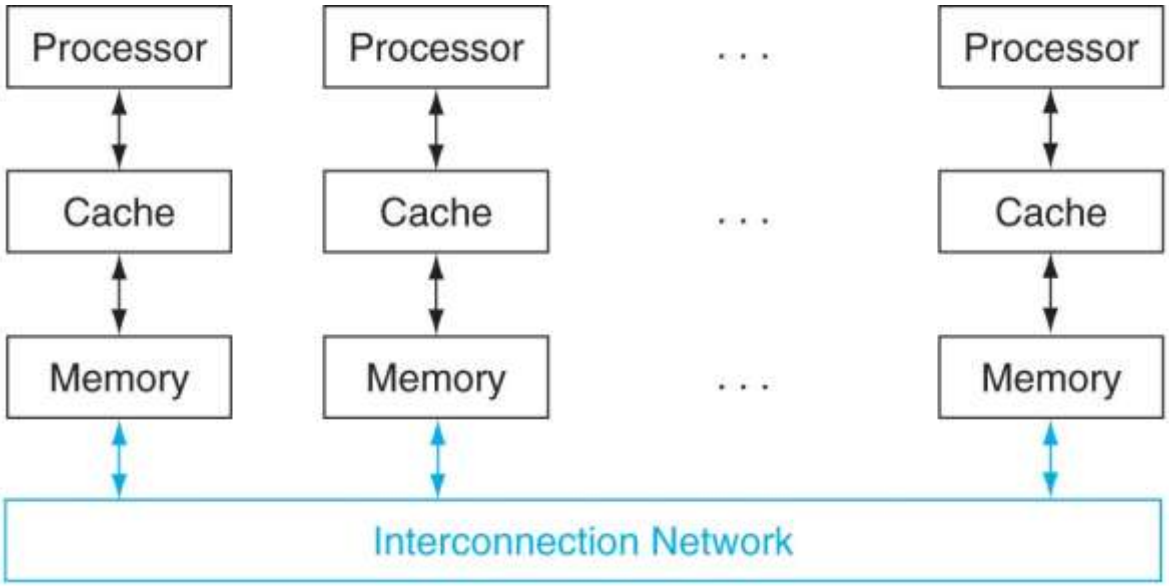


Resposta 1: Mesmo espaço de memória é compartilhado pelos diferentes processadores

Resposta 2: Comunicação é feita através de variáveis compartilhadas

Passagem de Mensagens

- **Message Passing**



Resposta 1: Processadores compartilham dados enviando explicitamente os dados (mensagem)

Resposta 2: Comunicação é feita através de primitivas de comunicação (*send* e *receive*)

Comunicação com Memória Compartilhada

- Variáveis compartilhadas contêm os dados que são comunicados entre um processador e outro
- Processadores acessam variáveis via loads/stores
- Acesso a estas variáveis deve ser controlado (sincronizado)
  - Uso de locks (semáforos)
  - Apenas um processador pode adquirir o lock em um determinado instante de tempo

Suporte do MIPS para Sincronização

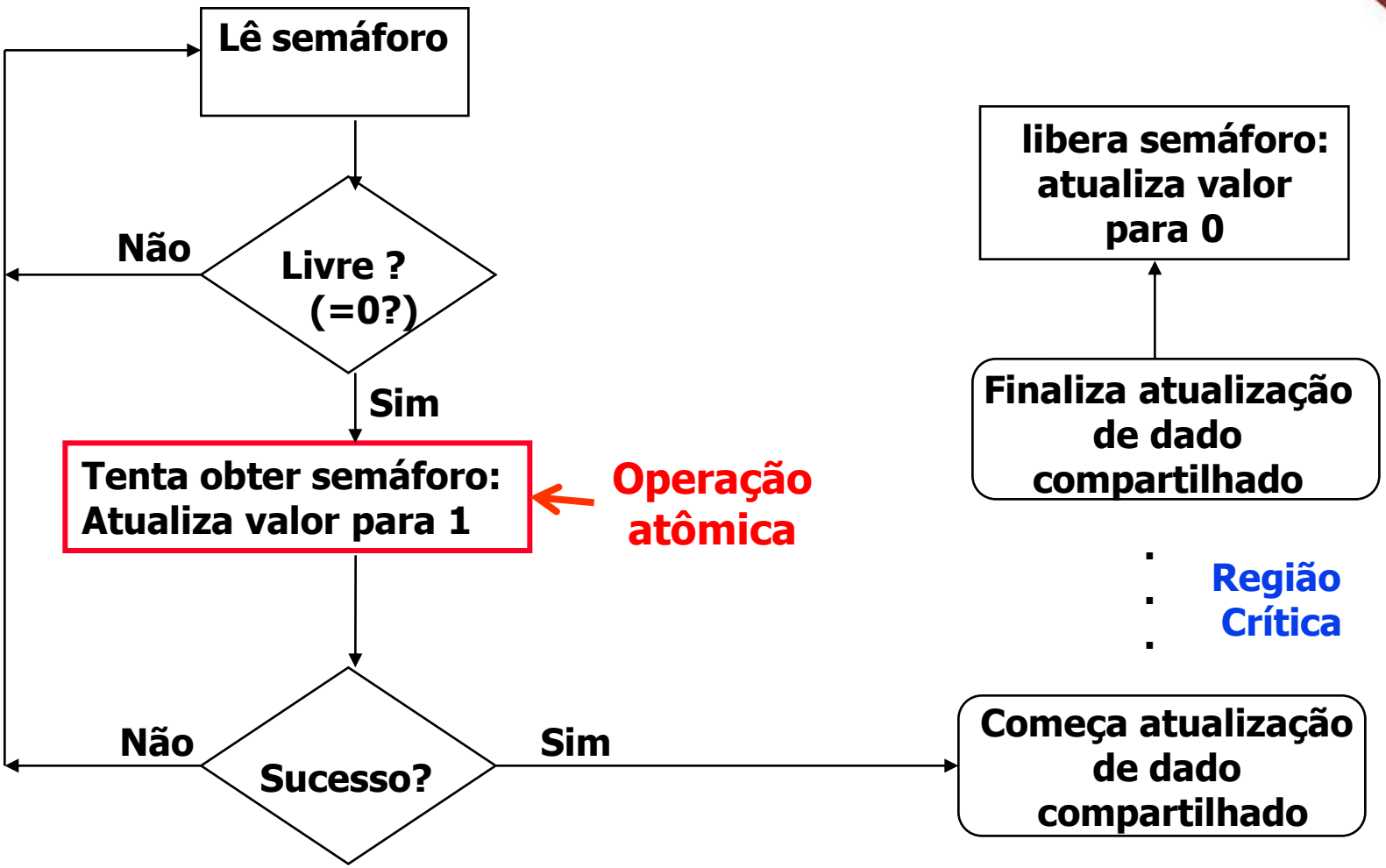
- ISA do MIPS possui duas instruções que dão suporte ao acesso sincronizado de uma posição de memória

  ```
  ll (load linked)
  sc (store conditional)
  ```

- O par de instruções deve ser utilizado para garantir leitura e escrita atômica da memória

  Garantia de obtenção do semáforo para uma variável compartilhada

Sincronizando Acesso com Semáforos



Uso do `ll` e `sc`

- `ll` lê uma posição de memória, `sc` escreve nesta mesma posição de memória se ela não tiver sido modificada depois do `ll`


```
try: addi $t0,$zero,1
     ll $t1, 0($s1)
     bne $t1,$zero,try
     sc $t0,0($s1)
     beq $t0,$zero,try
       ...  #região crítica
       ...
```

Comunicação com Message Passing

- Cada processador tem seu espaço de endereçamento privado
- Processadores mandam mensagens utilizando esquema parecido ao de acessar um dispositivo de Entrada/Saída *(veremos depois)*
  - No MIPS, via loads/stores em endereços “especiais”
  - Processador que aguarda mensagem é interrompido quando mensagem chega
- Sincronização de acesso aos dados é feita pelo próprio programa
  - Programador é responsável para estabelecer os pontos de comunicação com as primitivas *send* e *receive*

Hardware Multi-Threading

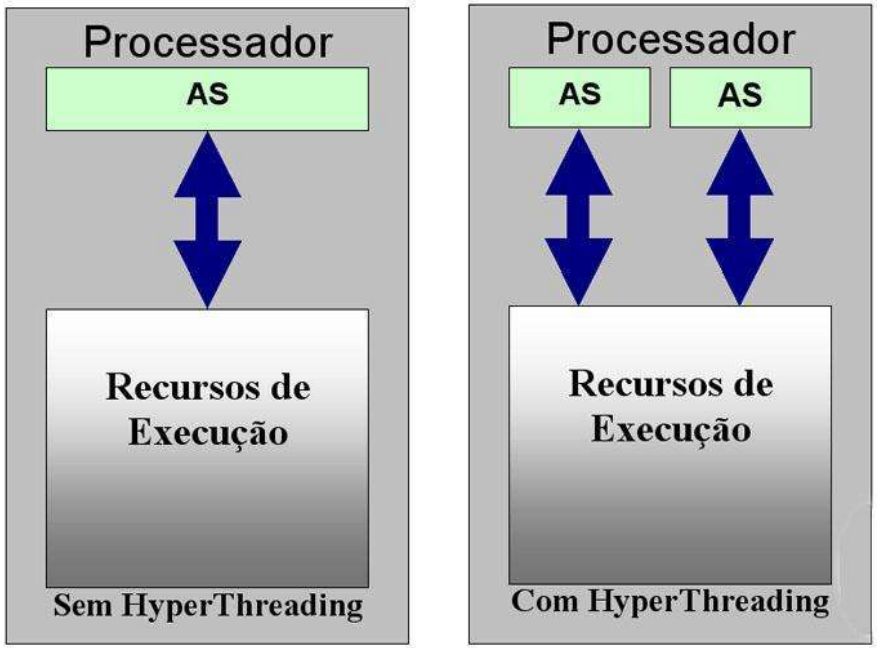
- Abordagem multi-thread
  - Aumentar utilização de recursos de hardware permitindo que múltiplas threads possam executar virtualmente de forma simultânea em único processador
  - Compartilhamento de unidades funcionais
- Objetivos
  - Melhor utilização de recursos (cache e unidades de execução são compartilhadas entre threads)
  - Ganho de desempenho (em média 20%)
  - Baixo consumo de área (< 5% da área do chip)

Hardware Multi-Threading

- **Componentes adicionais:**

  Lógica de controle e duplicação de unidades referente ao contexto do thread em execução (pilha, banco de registradores, buffers de instruções e de dados etc.)

  - Permitir concorrência na execução dos threads
  - Suporte a troca de contexto eficiente



AS – Architectural State

Tipos de Multi-Threading

- **Fine-grain (granularidade fina)**

  Processador troca de thread a cada instrução

  Usa escalonamento Round-robin

  - Pula threads paradas (stalled)

  Processador deve trocar threads a cada ciclo de clock

- **Coarse-grain (granularidade grossa)**

  Processador troca de threads somente em paradas (stalls) longas

  - Exemplo: quando dados não estão na cache

Análise sobre Fine-grain

- **Vantagem**

  Pode esconder perdas de throughput que são ocasionados por stalls longos e curtos

- **Desvantagem**

  Retarda execução de uma thread que não tem stalls e está pronta, pois só pode ser executada uma instrução da thread antes de trocar de thread

Análise sobre Coarse-grain

- **Vantagem**

  Evita trocas de threads desnecessárias, o que aumenta a velocidade de processamento de uma thread

- **Desvantagens**

  Perdas de throughput em stalls curtos

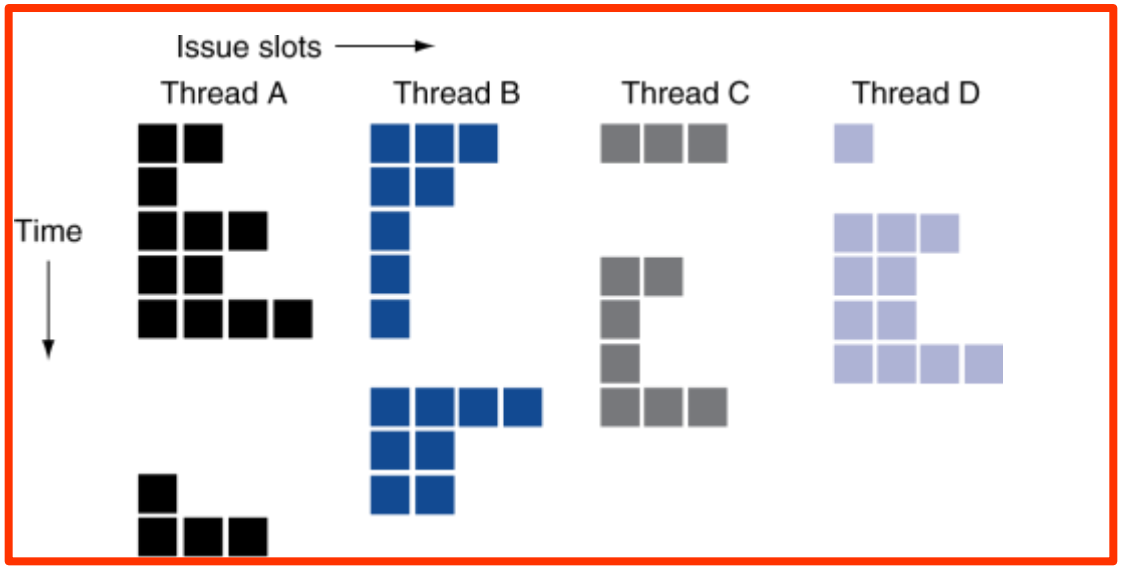
  Pipeline deve ser anulado e preenchido na troca de threads devido a um stall longo

Simultaneous Multi-Threading (SMT)

- SMT é uma variação de multi-threading para processadores superescalares com escalonamento dinâmico de threads e instruções
  - Escalonamento de instruções de threads diferentes
  - Instruções de threads diferentes podem executar paralelamente desde que haja unidades funcionais livres
  - Explora paralelismo ao nível de instrução (Instruction Level Parallelism - ILP)
  - Explora paralelismo ao nível de threads (Thread Level Parallelism – TLP)

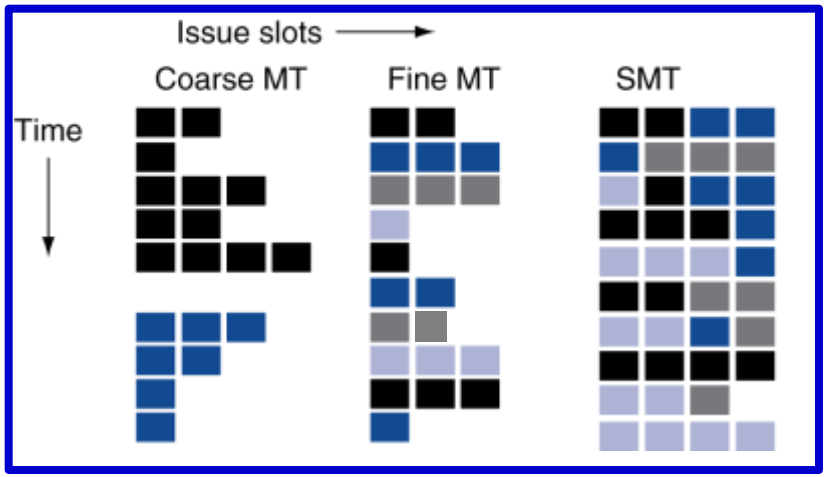
- Hyper-Threading é um caso de de SMT proposto pela Intel
  - Disponível em processadores Xeon, Pentium 4- HT, Atom, Intel i7

Exemplo de Multi-Threading



instrução

4 threads executando isoladamente em um processador superescalar de grau 4 sem multi-threading



4 threads executando conjuntamente em um processador superescalar de grau 4 com multi-threading utilizando diferentes estratégias de multi-threading

Centro de Informática

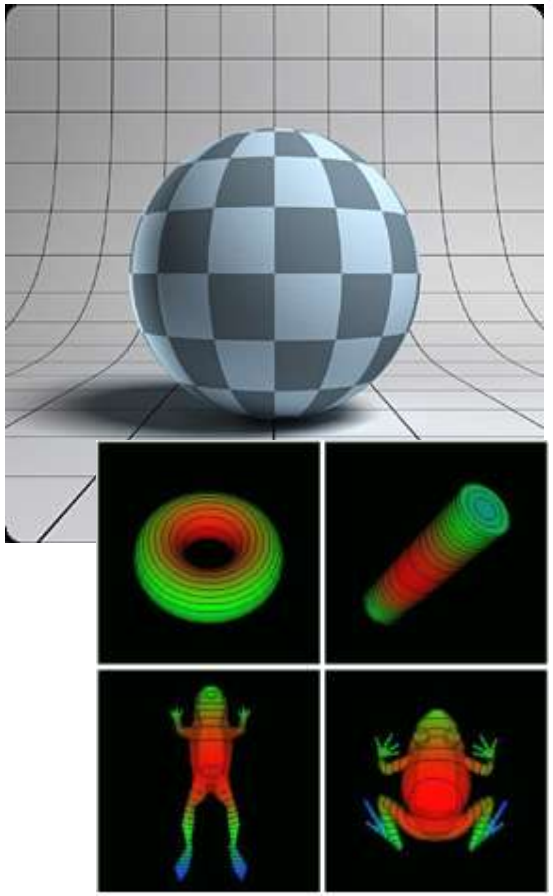
Graphics Processing Units (GPUs)

- Indústria de jogos eletrônicos impulsionou o desenvolvimento de dispositivos de alto desempenho para processamento gráfico

- GPUs são aceleradores especializados em processamento gráfico que trabalham em conjunto com a CPU
  - Livra a CPU de processamento custoso proveniente de aplicações gráficas
  - GPU dedica todos os seus recursos ao processamento gráfico
  - Combinação CPU-GPU – multiprocessamento heterogêneo

Processamento Gráfico

- Envolve desenhar formas geométricas em 2D ou 3D, renderizar conjunto de pixels entre outras coisas
- Dados (ex: conjunto de pixels) podem ser tratados independentemente
- Processamento de elementos gráficos podem ser feito de forma paralela
  - Paralelismo ao nível de processamento de dados



Características da Arquitetura de GPUs

- Orientado ao processamento paralelo de dados
  - Composto por vários processadores(cores) paralelos
- GPUs utilizam multi-threading intensivamente
- Compensa o acesso lento a memória com troca intensa de threads
  - Menos dependência em caches multi-níveis
- Memória utilizada é mais larga e possui largura de banda maior,
  - Porém são menores que memória para CPU

Mais sobre GPUs

- Existência de linguagens de programação e APIs de alto nível de abstração
  - DirectX, OpenGL
  - C for Graphics (Cg), High Level Shader Language (HLSL)
  - Compute Unified Device Architecture (CUDA)
- Tendência forte de aproveitar o paralelismo de GPUs para escrever programas de propósito geral
  - Utilização de sistemas heterogeneos CPU/GPU
    - CPU para código sequencial, GPU para código paralelo
- Líderes de mercado (em 2008)
  - NVIDIA GeForce 8800 GTX (16 processadores, cada um com unidades de processamento que comportam 8 threads simultâneas)
  - AMD's ATI Radeon and ATI FireGL

Exemplo: Multicore Xbox360 – XCGPU

- **Plataforma heterogênea**

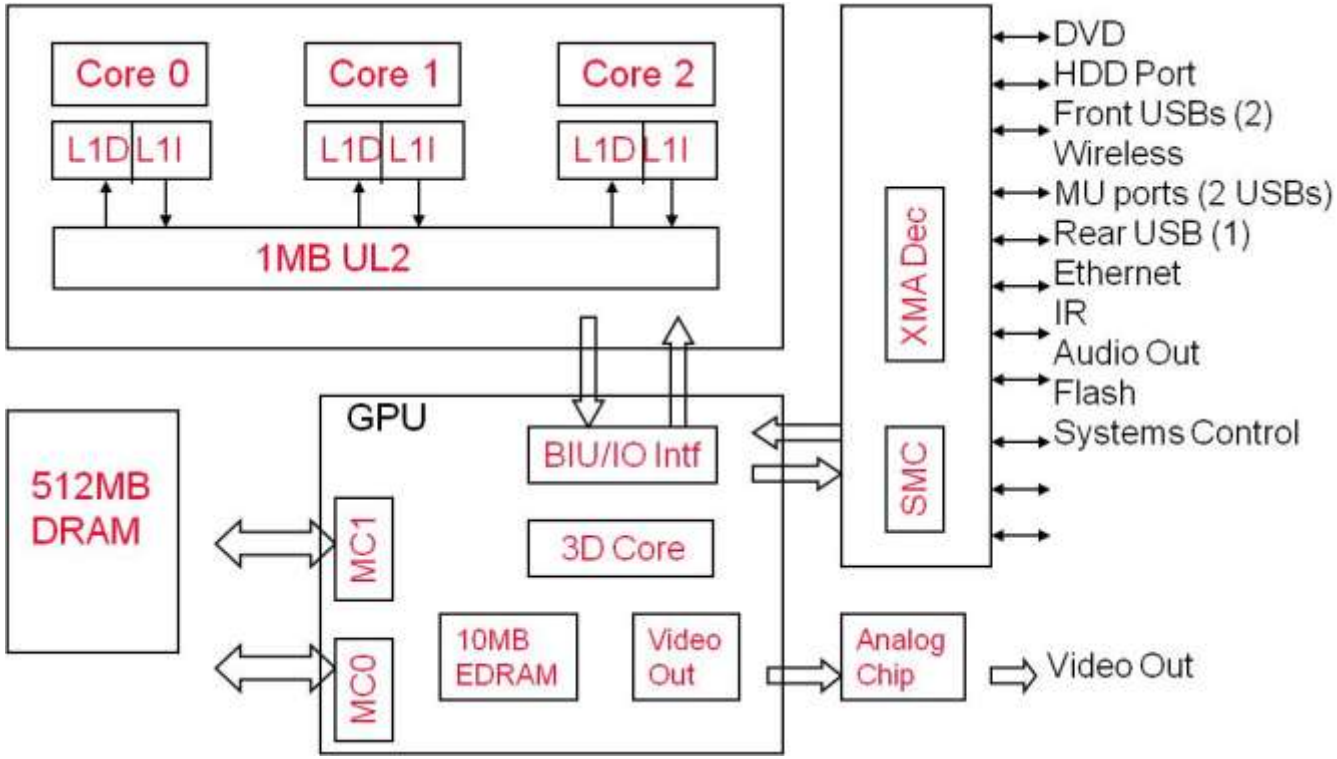
  Xenon CPU - Três cores SMT

  - 32KB L1 D$ & I$, 1MB UL2 cache
  - 3.2 Ghz
  - Superescalar grau 2, com pipeline de 21 estágios pipeline, com 128 registradores de 128 bits

  Xenos GPU ATI

  - 500MHz
  - 512MB de DDR3 DRAM
  - 48 pixel shader cores, cada um com 4 ALUs

Diagrama de Bloco do Xenon



Outro Exemplo: NVIDIA Tesla