

# Infraestrutura de Hardware

## Processamento Paralelo – Multicores

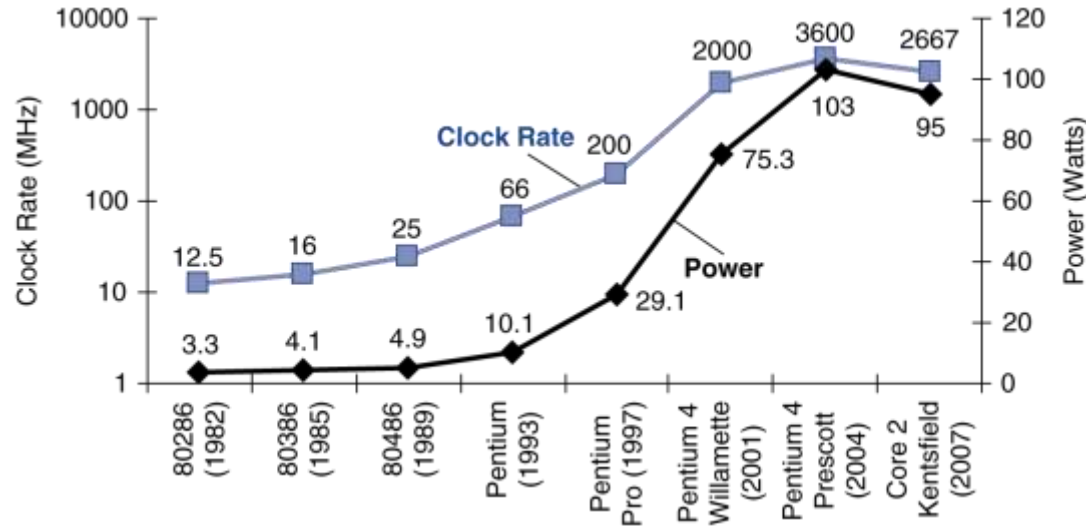


UNIVERSIDADE  
FEDERAL  
DE PERNAMBUCO

# 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 e Energia

- Quanto maior a frequência, maior a potência dissipada  
Sistema de esfriamento do processador é impraticável para frequências muito altas
- Devem ser mais eficientes em termos de energia  
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

# Interesse em Multiprocessadores

- Uma crescente utilização de servidores
- Um crescimento em aplicações data-intensive
- Um melhor entendimento de como usar multiprocessadores para explorar thread-level paralelismo
- Investimento em replicação é mais atrativo que investimento em um projeto exclusivo.

# Melhorando Desempenho com Multiprocessamento

- Alguns problemas que requerem alto desempenho podem ser solucionados com o uso de clusters

Servidores independentes conectados por rede que funcionam como um grande e único multiprocessador

  - Search engines, servidores web, servidores de email, banco de dados, etc
- Desafio é desenvolver programas paralelos que tenham alto desempenho em multiprocessadores a medida que a quantidade de processadores aumenta

# Multicores

## ■ Microprocessadores multicores

Mais de um processador por chip  
Cada processador é chamado de core

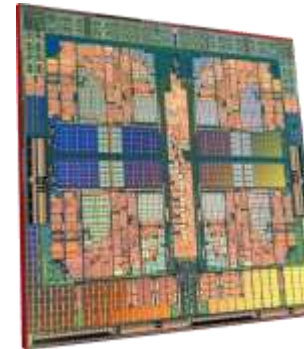
Capacidade de integração de transistor permitiu maior quantidade de cores por chip

Menor consumo de energia por core

## ■ Cada core pode rodar com frequências menores

Menor potência

Desempenho alcançado com aumento do throughput





# Exemplo: Intel Core 2 Duo

- **Cores homogêneas**

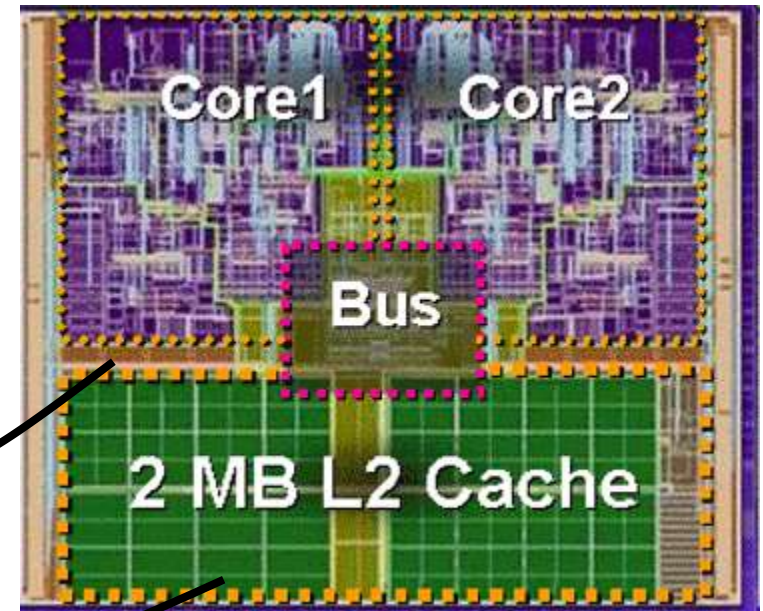
Superscalares

- (escalonamento dinâmico, especulação, superescalar)

- **Interconexão baseada em barramento**

- **Cada “core” tem cache local (L1)**

- **Memória compartilhada**  
(cache L2) no chip



Source: Intel Corp.

# Classificação de Arquiteturas Segundo A Taxonomia de Flynn

M.J. Flynn, "Very High-Speed Computers",  
*Proc. of the IEEE*, V 54, 1900-1909, Dec. 1966.

		Data Streams	
		Single	Multiple
Instruction Streams	Single	<b>SISD:</b> Intel Pentium 4	<b>SIMD:</b> instruções SSE do x86
	Multiple	<b>MISD:</b> Nenhum exemplo	<b>MIMD:</b> Clusters, Intel i7

- Flynn classificou de acordo com streams de dado e controle em 1966
- SIMD  $\Rightarrow$  Paralelismo ao nível de dados
- MIMD  $\Rightarrow$  Paralelismo ao nível de threads
- MIMD mais popular
  - Flexibilidade: N programas multithreaded

# 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
  - Programar visando desempenho de modo que seja mais transparente para o programador
  - Particionamento de tarefas (Balanceamento de carga)
  - Coordenação
  - Overhead de comunicação

# Analizando Desempenho com Multicores

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

**T<sub>m</sub>** = Tempo de Execução com melhoria

**T<sub>a</sub>** = Tempo de execução afetado pela melhoria

**Q<sub>c</sub>** = quantidade de cores

**T<sub>sm</sub>** = Tempo de Execução não afetado pela melhoria

$$T_m = T_a/Q_c + T_{sm}$$

# 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  $10 \times 10$ ) +  $10t$  (soma de 10 escalares) =  $110t$

Soma de 10 escalares deve ser sequencial, portanto leva  $10t$  ou seja  $T_{sm} = 10t$

Soma de 2 matrizes pode ser feita em paralelo.

Para 10 cores:

$$T_m = T_a/Q_c + T_{sm} = 100t/10 + 10t = 20t$$

$$\text{Ganho} = 110t/20t = 5,5$$

# 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  $10 \times 10$ ) +  $10t$  (soma de 10 escalares) =  $110t$

Soma de 10 escalares deve ser sequencial, portanto leva  $10t$  ou seja  $T_{sm} = 10t$

Soma de 2 matrizes pode ser feita em paralelo.

Para 100 cores:

$$T_m = T_a/Q_c + T_{sm} = 100t/100 + 10t = 11t$$

$$\text{Ganho} = 110t/11t = 10$$

# Analizando 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  $100 \times 100$ ) +  $10t$  (soma de 10 escalares) =  **$10010t$**

Soma de 10 escalares deve ser sequencial, portanto leva  $10t$  ou seja  $T_{sm} = 10t$

Soma de 2 matrizes pode ser feita em paralelo.

Para 10 cores:

$$T_m = T_a/Q_c + T_{sm} = 10000t/10 + 10t = \mathbf{1010t}$$

$$\text{Ganho} = 10010t/1010t = \mathbf{9,9}$$

## 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  $100 \times 1000$ ) +  $10t$  (soma de 10 escalares) =  **$10010t$**

Soma de 10 escalares deve ser sequencial, portanto leva  $10t$  ou seja  $T_{sm} = 10t$

Soma de 2 matrizes pode ser feita em paralelo.

Para 100 cores:

$$T_m = T_a/Q_c + T_{sm} = 10000t/100 + 10t = \mathbf{110t}$$

$$\text{Ganho} = 10010t/110t = \mathbf{91}$$

# Analizando 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!**

# Escalabilidade

- Aumentar o speedup em multiprocessadores mantendo o tamanho do problema fixo é difícil

**Escalabilidade forte** – quando speedup pode ser alcançado sem aumentar o tamanho do problema

**Escalabilidade fraca** – quando speedup pode ser alcançado apenas aumentando o tamanho do problema proporcionalmente ao aumento da quantidade de processadores

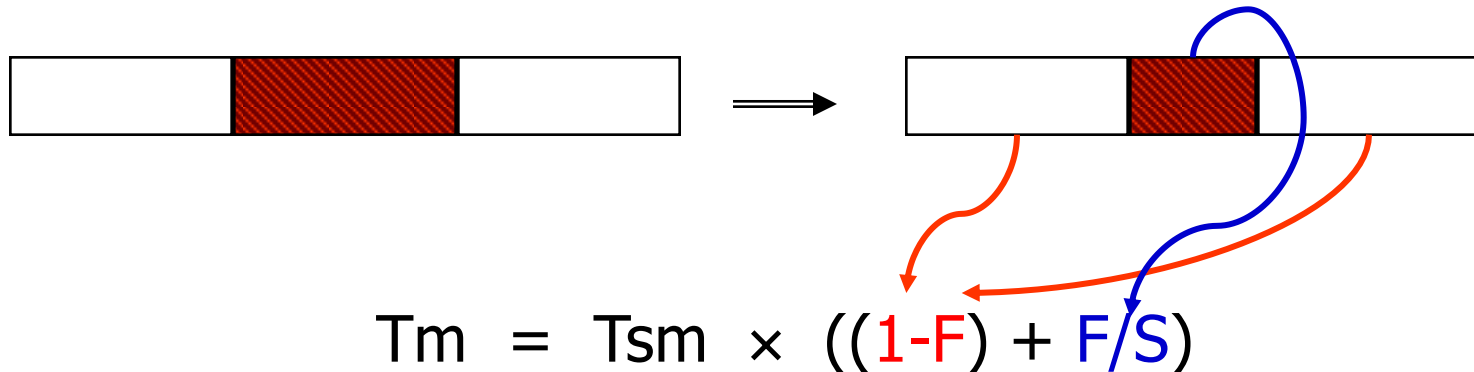
- Balanceamento de carga é um importante componente  
Um único processador com o dobro da carga em relação aos outros processadores diminui em quase metade o speedup

# Calculando Speedup com Lei de Amdahl

## ■ Speedup

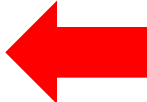
$$\text{Speedup} = \frac{T_{sm}}{T_m}$$

- Suponha que a paralelização acelere uma fração  $F$  ( $F < 1$ ) da tarefa por um fator  $S$  ( $S > 1$ ) e o resto da tarefa permanece inalterado



$$\text{Speedup} = 1 / ((1-F) + F/S)$$

# Desafio do Processamento Paralelo

- Suponha speedup de 80X speedup para 100 processadores. Qual a fração do programa que deve ser sequencial?
  - a. 10%
  - b. 5%
  - c. 1%
  - d. <1%  **0,25%**

## Resposta Usando Lei de Amdahl

$$\text{Speedup}_{\text{overall}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{parallel}}}{\text{Speedup}_{\text{parallel}}}}$$

$$80 = \frac{1}{(1 - \text{Fraction}_{\text{parallel}}) + \frac{\text{Fraction}_{\text{parallel}}}{100}}$$

$$80 \times \left( (1 - \text{Fraction}_{\text{parallel}}) + \frac{\text{Fraction}_{\text{parallel}}}{100} \right) = 1$$

$$79 = 80 \times \text{Fraction}_{\text{parallel}} - 0.8 \times \text{Fraction}_{\text{parallel}}$$

$$\text{Fraction}_{\text{parallel}} = 79 / 79.2 = 99.75\%$$

# Classificação de Multiprocessadores

- **Arquitetura Paralela = Arquitetura do Computador + Arquitetura da Comunicação**
- **Classificação por memória:**

## Processador de Memória Centralizada (Symmetric)

- Típico para sistemas pequenos → demanda de largura de banda de memória e rede de comunicação.

## Multiprocessador de Memória Fisicamente Distribuída

- Escala melhor → demanda de largura de banda para rede de comunicação

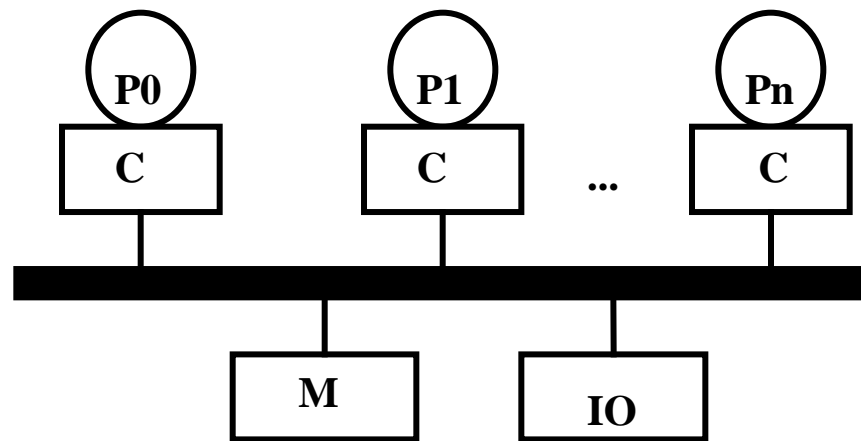


# Classificação por Memória

## ■ Multiprocessadores de Memória Centralizada

Poucos processadores ( poucas dezenas chips ou cores) em 2006

Memória única e centralizada



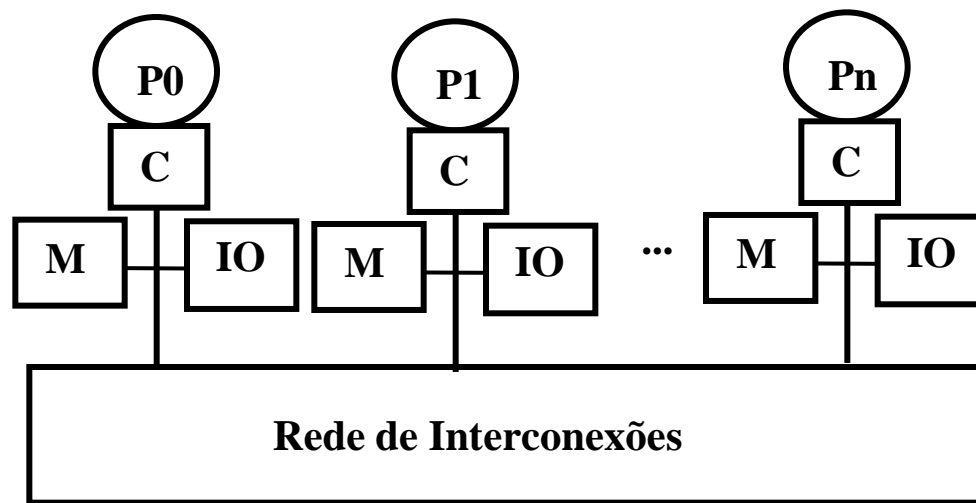
**Memória Centralizada**

# Classificação por Memória

- **Multiprocessadores de Memória Fisicamente Distribuída**

Maior número de processadores (centenas de chips ou cores)

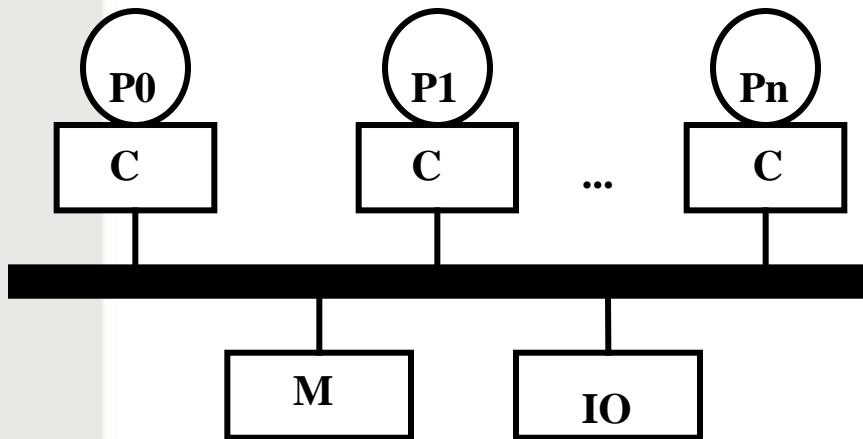
Memória distribuída entre processadores



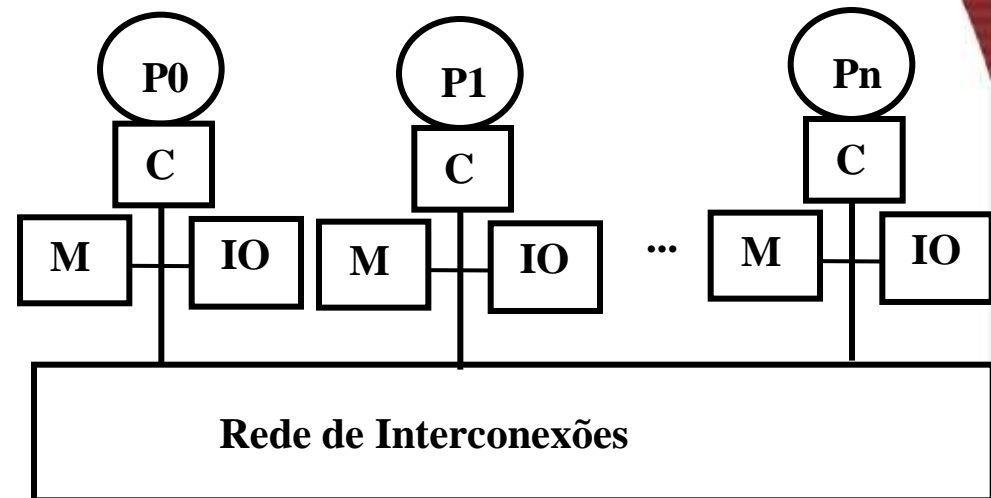
**Memória Distribuída**

# Centralizada vs. Distribuída

Escalabilidade maior



**Memória Centralizada**



**Memória Distribuída**

# Multiprocessadores – Perguntas Importantes

**Pergunta 1: Como os diferentes processadores compartilham dados (do ponto de vista de HW)?**

**Pergunta 2: Como é feita a comunicação entre os diferentes processadores (do ponto de vista de SW) ?**

# Classificação de Multiprocessadores

- **Arquitetura Paralela = Arquitetura do Computador + Arquitetura da Comunicação**
- Classificando por comunicação:

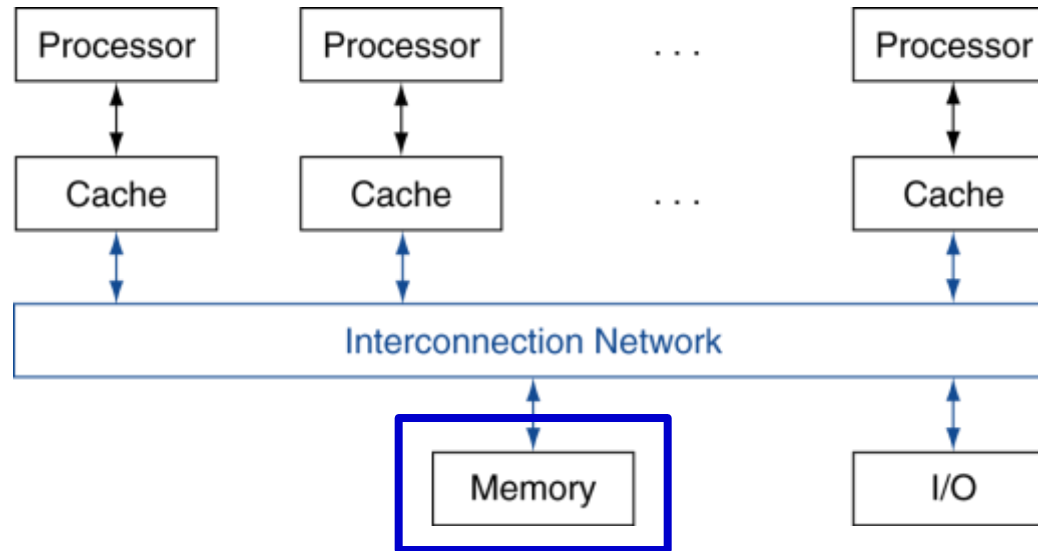
**Memória Compartilhada:** processadores se comunicam através de espaço de endereçamento comum.

- Memória Centralizada: , **UMA** (Uniform Memory Access time)
- Memória Distribuída:, **NUMA** (Non Uniform Memory Access time)

**Message-Passing** : processadores enviam mensagens

# 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**

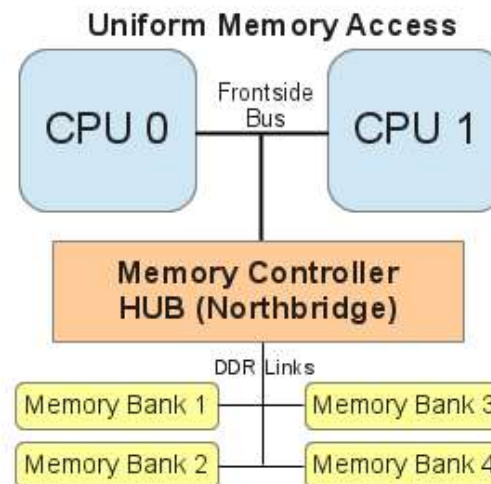
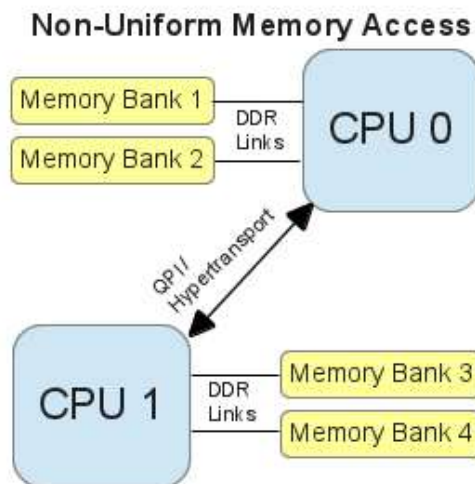
# NUMA x UMA

## ■ NUMA

Memória fisicamente distribuída (transparente ao programador)  
Processador pode acessar memória remota, mas leva mais tempo do que acessar memória local

## ■ UMA

Memória centralizada  
Tempo de acesso a memória uniforme independente do processador que faz o acesso



# Exemplo Usando Multiprocessador com Memória Compartilhada: Redução de Somas

- Soma de 100.000 números em multiprocessador UMA com 100 processadores
  - Cada processador tem um ID:  $0 \leq P_n \leq 99$
  - Partição: 1000 números por processador
  - Soma inicial de cada processador

```
sum[Pn] = 0;  
for (i = 1000*Pn; i < 1000*(Pn+1); i = i + 1)  
    sum[Pn] = sum[Pn] + A[i];
```



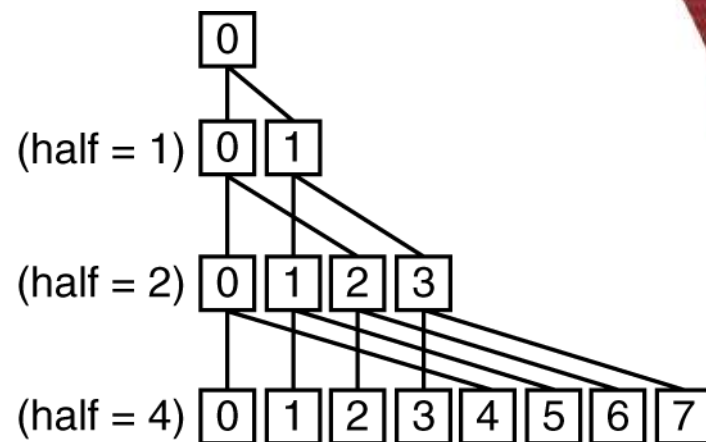
# Exemplo Usando Multiprocessador com Memória Compartilhada: Etapa de Redução de Somas

- Depois adiciona-se-se somas parciais

Redução: dividir para conquistar

Metade dos processadores somam pares, e depois um quarto dos processadores...

Necessidade de sincronização entre etapas de redução



```
half = 100; ← Variável privada
```

```
do {
```

```
  synch(); ← Sincronização
```

```
  if (half%2 != 0 && Pn == 0)
```

```
    sum[0] = sum[0] + sum[half-1];
```

```
    /* Necessário quando half é ímpar;
```

```
    Processor0 soma o elemento que sobrou*/
```

```
  half = half/2;
```

```
  if (Pn < half) sum[Pn] = sum[Pn] + sum[Pn+half];
```

Variável compartilhada

```
}while (half != 1);
```

# 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)

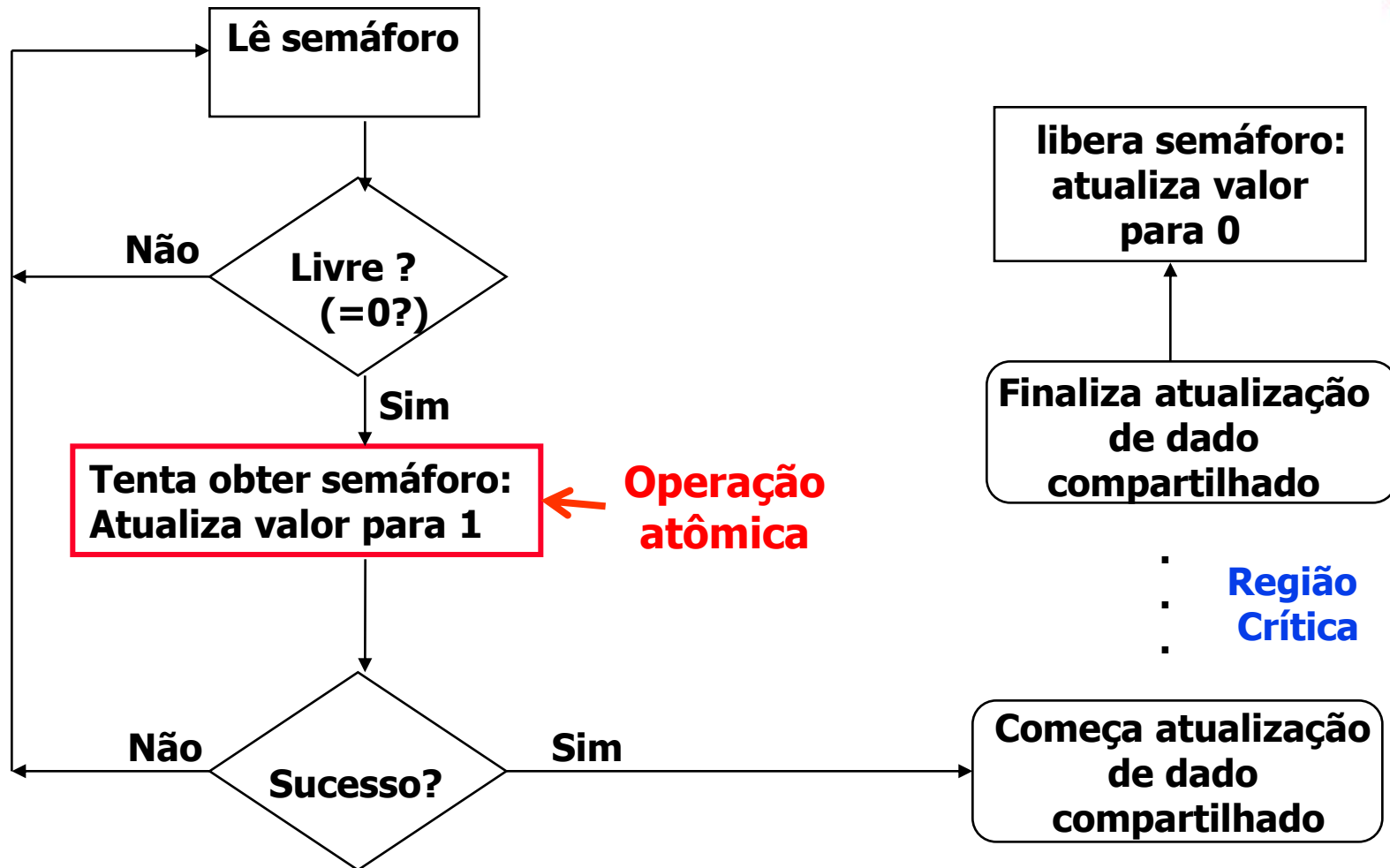
Uso de **locks (semáforos)**

Apenas um processador pode adquirir o lock em um determinado instante de tempo

# Suporte do MIPS para Sincronização

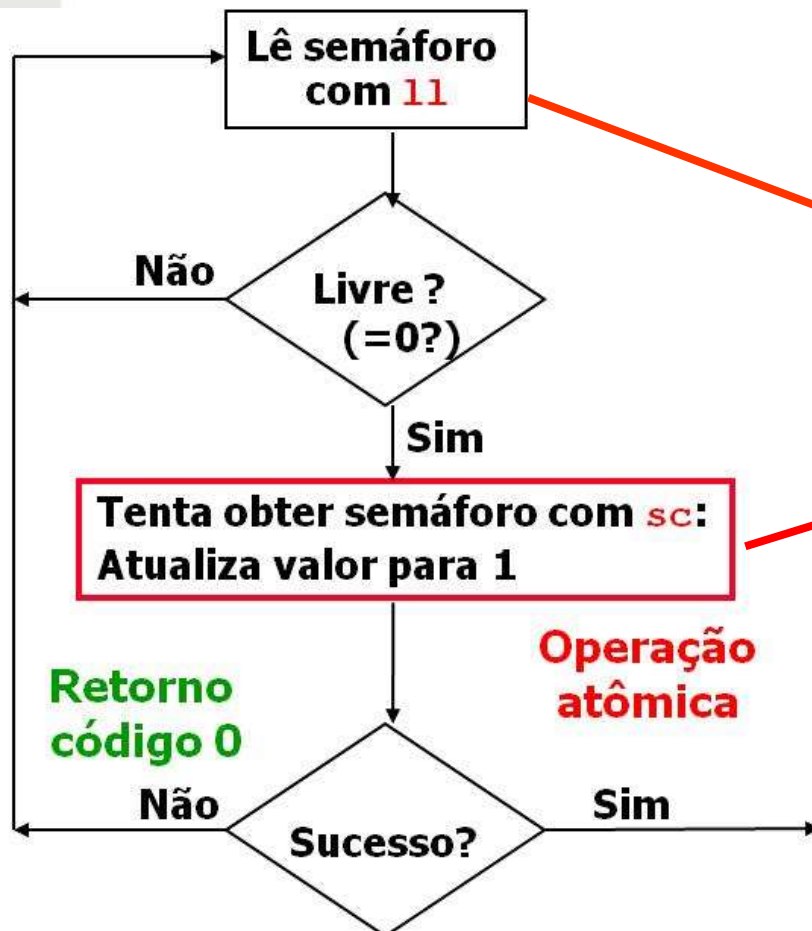
- ISA do MIPS possui duas instruções que dão suporte ao acesso sincronizado de uma posição de memória
  - ll** (**l**oad **l**inked)
  - sc** (**s**to**r**e **c**onditional)
- O par de instruções deve ser utilizado para garantir leitura e escrita atômica da memória
  - Garantia de obtenção do semáforo 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**

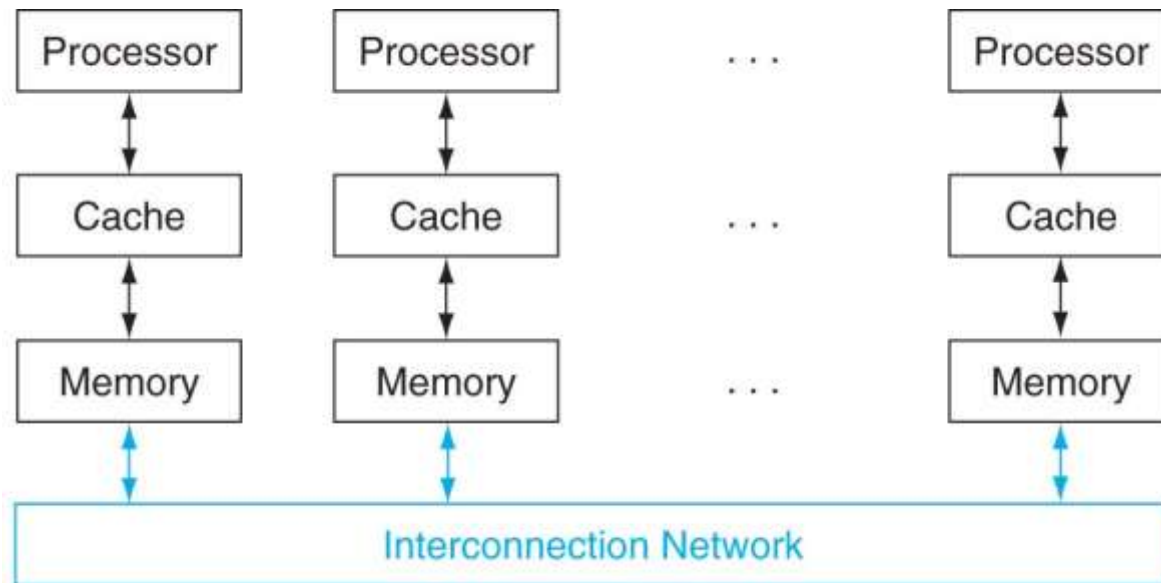


```
try: addi $t0,$zero,1
      ll $t1, 0($s1)
      bne $t1,$zero,try
      sc $t0,0($s1)
      beq $t0,$zero,try
      ... #região crítica
      ...
```

**Operação  
atômica**

# 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*)**

# Exemplo Usando Multiprocessador com Passagem de Mensagens: Redução de Somas

- Soma de 100.000 números em multiprocessador com 100 processadores

Um processador envia subconjuntos de 1000 números para cada processador/memória local

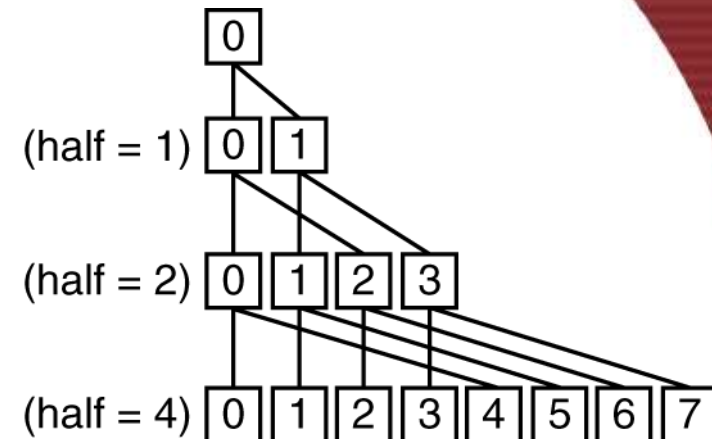
Cada processador lê os 1000 números de sua memória local

```
sum = 0;  
for (i = 0; i < 1000; i = i + 1)  
    sum = sum + AN[i]; /* array na memória  
                        local*/
```

# Exemplo Usando Multiprocessador com Passagem de Mensagens: Etapa de Redução de Somas

- Metade dos processadores enviam somas parciais, a outra metade recebe e soma, depois um quarto dos processadores...

Sincronização entre processadores é provida pelas primitivas send e receive



```
limit = 100; half = 100;
do {
    half = (half+1)/2; /* send vs. receive
                        linha divisória */
    if (Pn >= half && Pn < limit)
        send(Pn - half, sum);
    if (Pn < (limit/2))
        sum = sum + receive();
    limit = half; /* limit superior dos senders */
}while (half != 1);
```

Envia soma

Recebe soma



# 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%)

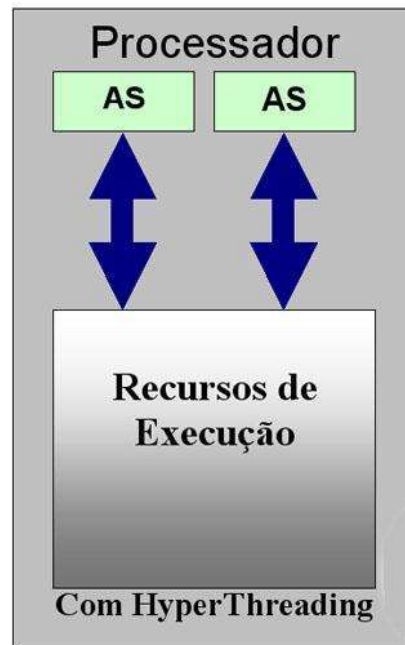
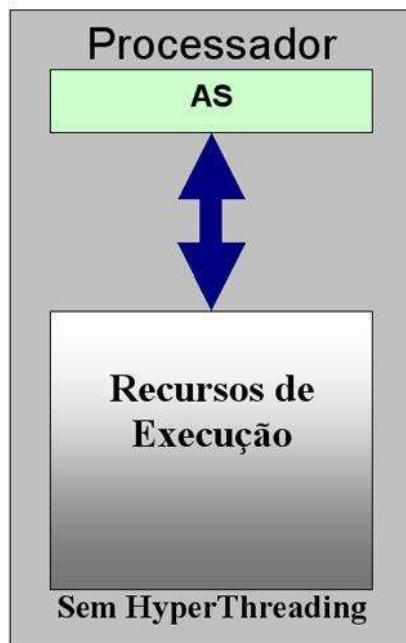
Baixo consumo de área (< 5% da área do chip)

# 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

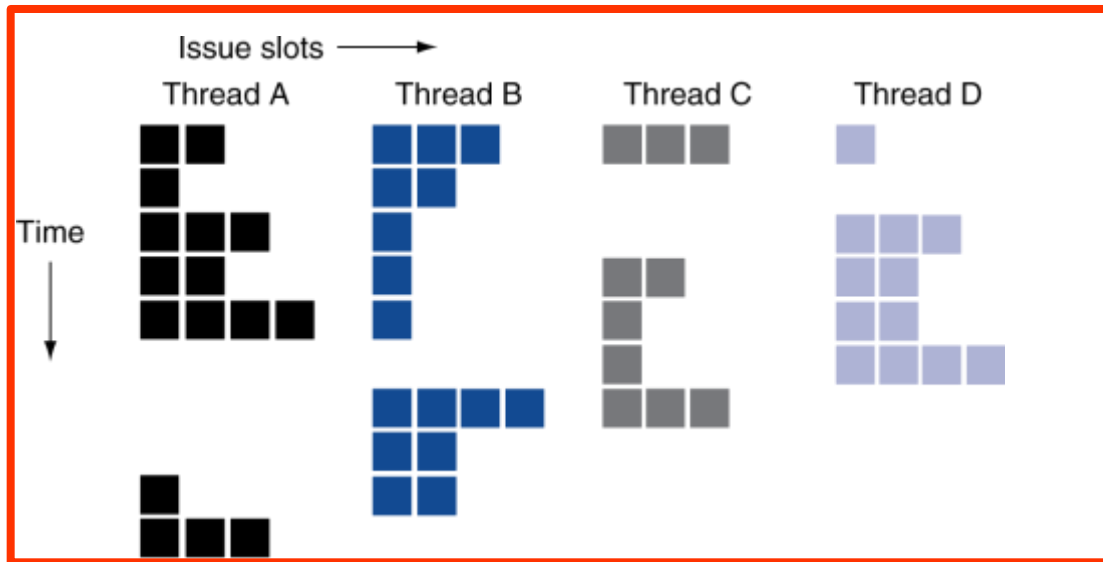
Instruções de threads diferentes podem executar paralelamente desde que haja unidades funcionais livres

Explora paralelismo ao nível de instrução (Instruction **L**evel **P**arallelism - **ILP**)

Explora paralelismo ao nível de threads ( **T**hread **L**evel **P**arallelism – **TLP**)

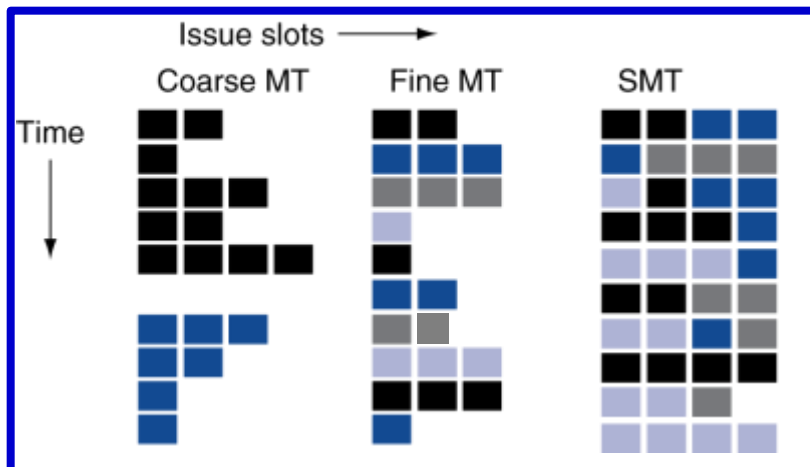
- Hyper-Threading é um caso de de SMT proposto pela Intel  
Disponível em processadores Xeon, Pentium 4- HT, Atom, Intel i7

# Exemplo de Multi-Threading



■ instrução

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



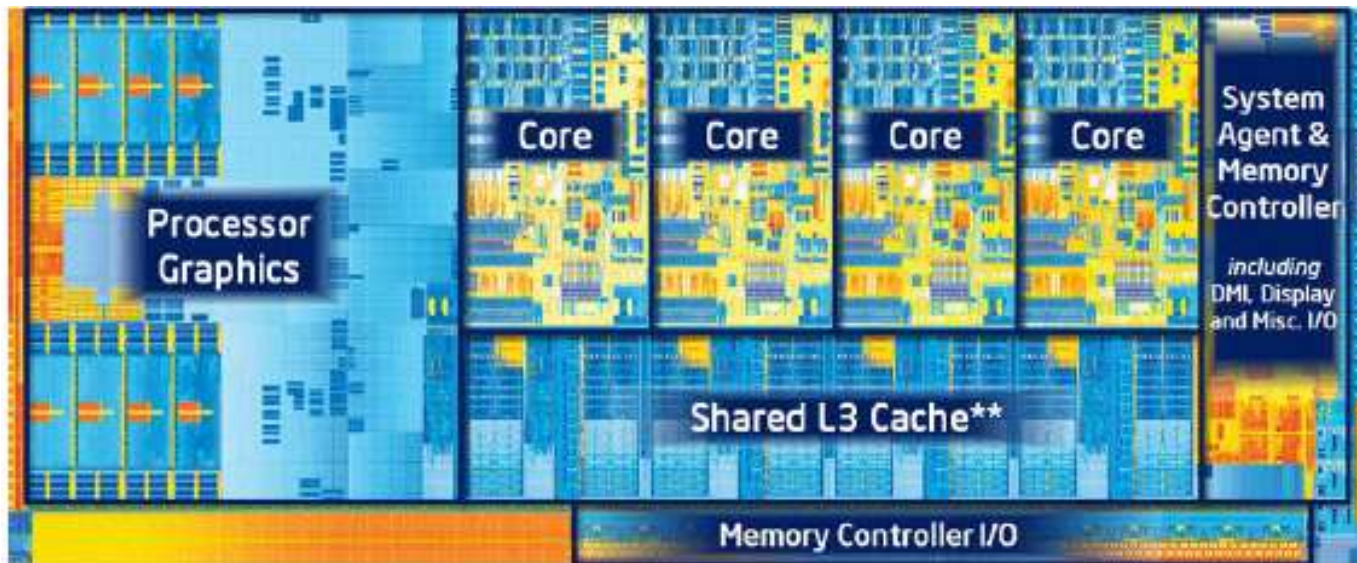
4 threads executando conjuntamente em um processador superescalar de grau 4 com multi-threading utilizando diferentes estratégias de multi-threading



# Exemplo de Multicore: Intel I7

## ■ Características principais:

- 4 a 6 cores
- 3 níveis de cache (cache L3 compartilhada)
- L3 com 4-15 MB, L1 com 32KB e L2 com 256KB
- GPU integrada
- Arquitetura Ivy Bridge

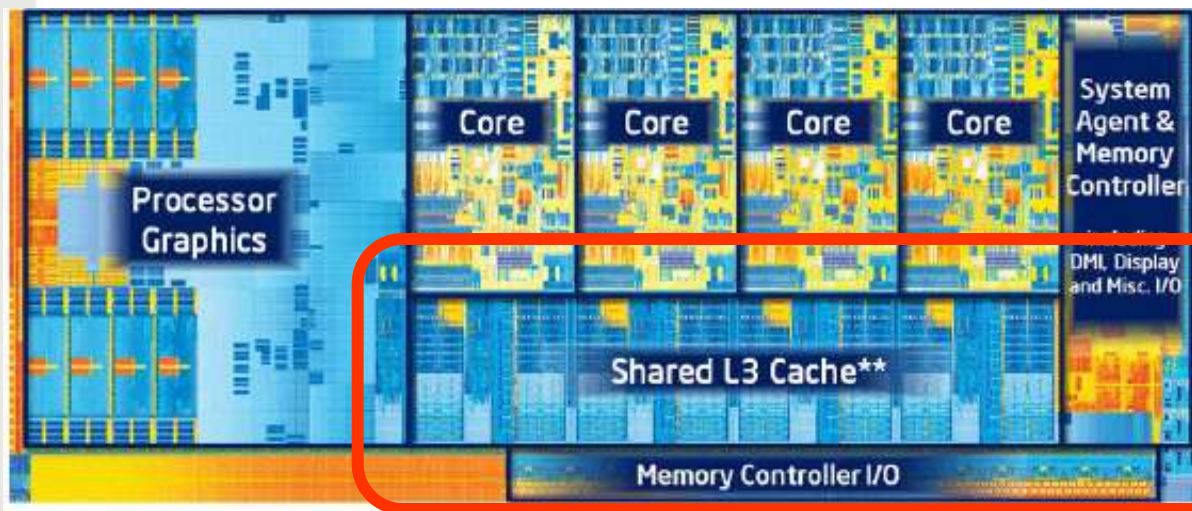


# Intel I7: Pipeline

- Cada core tem um pipeline de 14 estágios
- Frequências que variam entre 1,8 e 3,7 GHz
  - Versões para mobile e desktop
- Superescalar
  - 6 instruções (micro-ops) podem ser executadas
  - Execução fora de ordem
- Execução especulativa nos branches
  - Combina 3 técnicas diferentes baseados no histórico

# Intel I7: Arquitetura de Memória e Comunicação

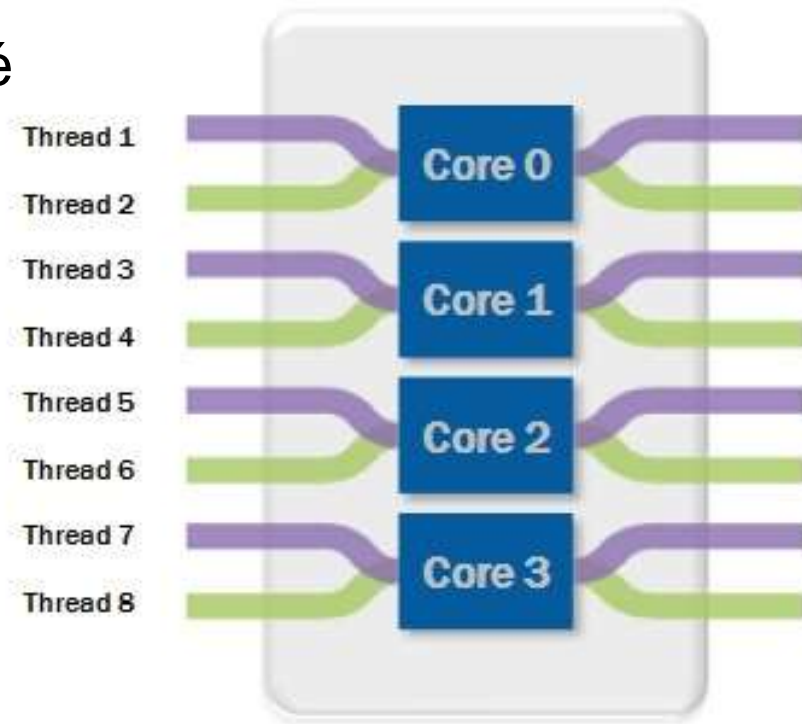
- Memória centralizada
  - Memória única vista por todos os cores (cache L3 e memória)
- Shared Memory Processor
  - Cada core tem dois níveis de cache privados
  - Cores podem compartilhar dados na L3



**Memória centralizada e compartilhada**

# Intel I7: Multithreading

- Utiliza tecnologia Hyper-Threading
- Cada core consegue executar até 2 threads simultaneamente
  - Se 4 cores, até 8 threads são executadas
- Se houver apenas uma thread executando, todos os recursos (para executar 2 threads) são alocados para a thread em execução





# 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

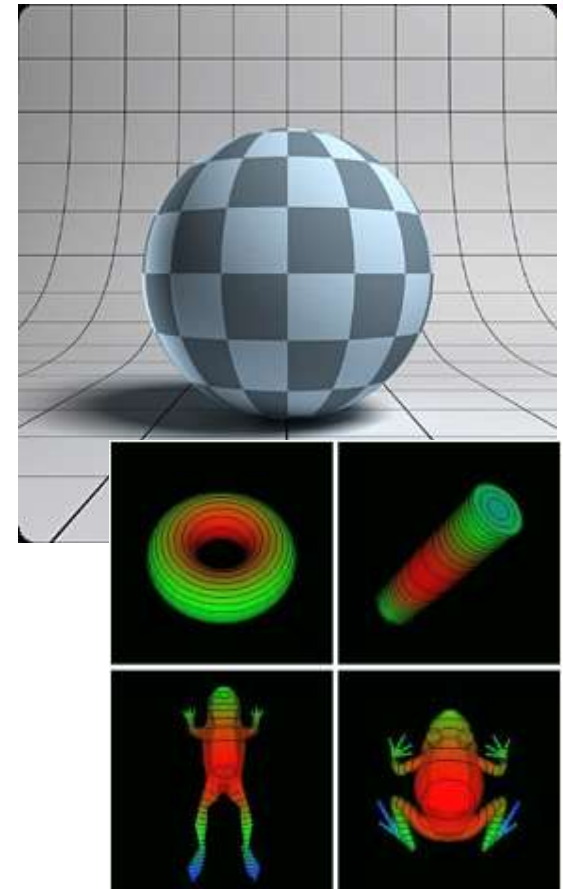
Combinação CPU-GPU –  
multiprocessamento **heterogêneo**



# 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

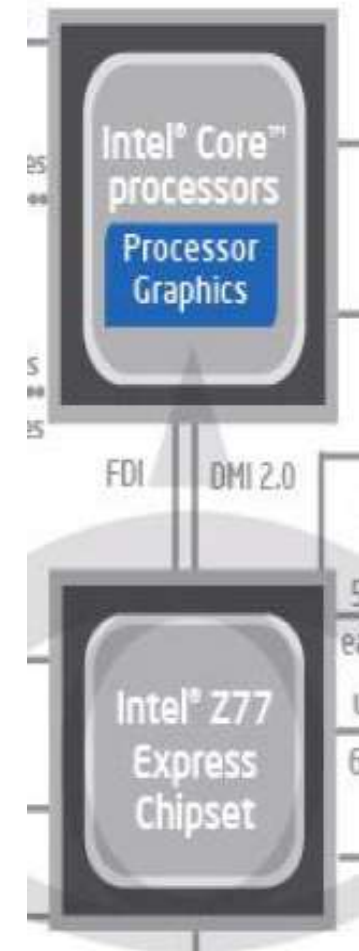
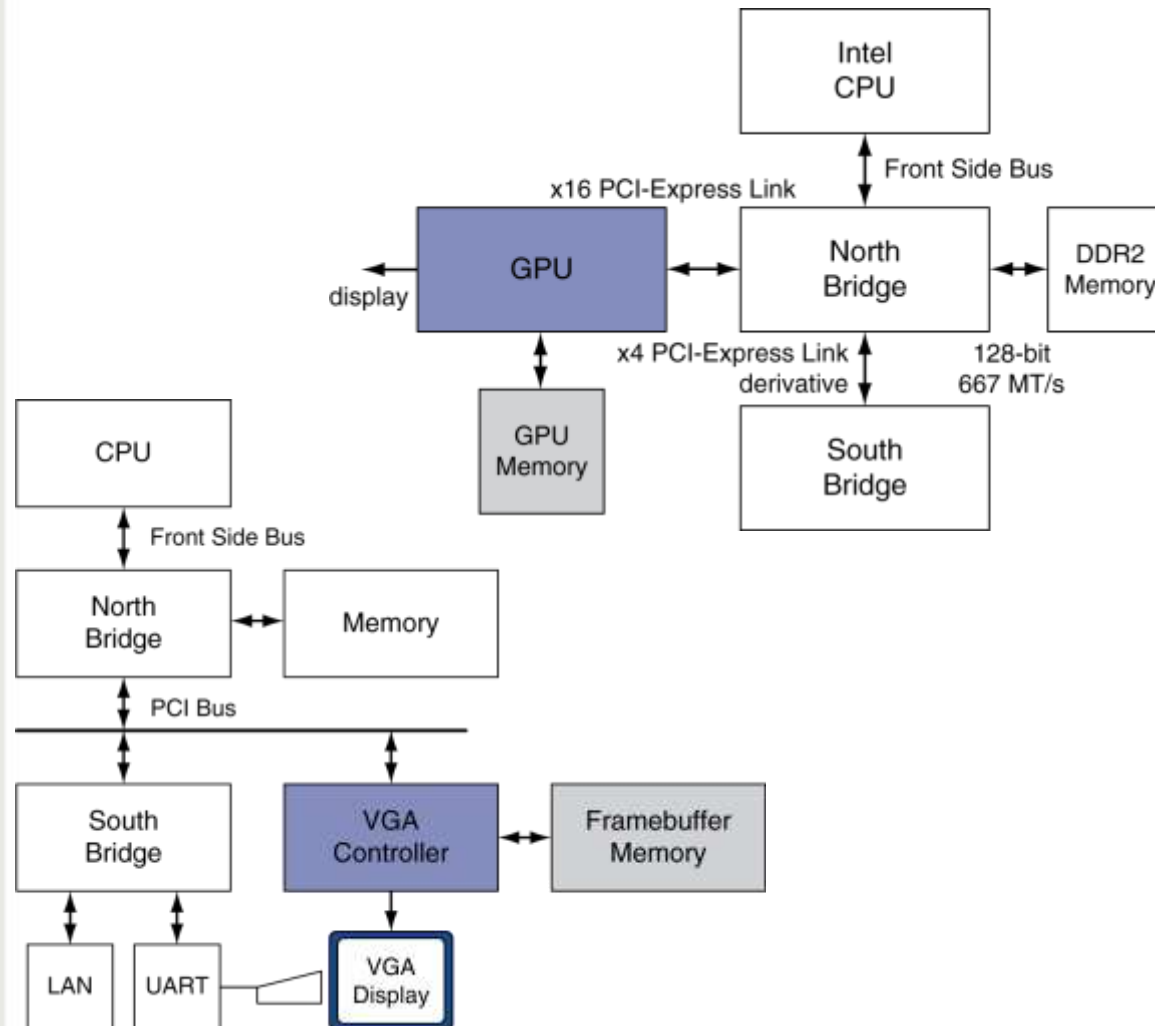
- Orientado ao processamento paralelo de dados  
Composto por vários processadores(cores) paralelos
- GPUs utilizam multi-threading intensivamente
- Compensa o acesso lento a memória com troca intensa de threads  
Menos dependência em caches multi-níveis
- Memória utilizada é mais larga e possui largura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



# Processamento Gráfico ao Longo do Tempo...



# Exemplo: Multicore Xbox360 – XCGPU

## ■ Plataforma heterogênea

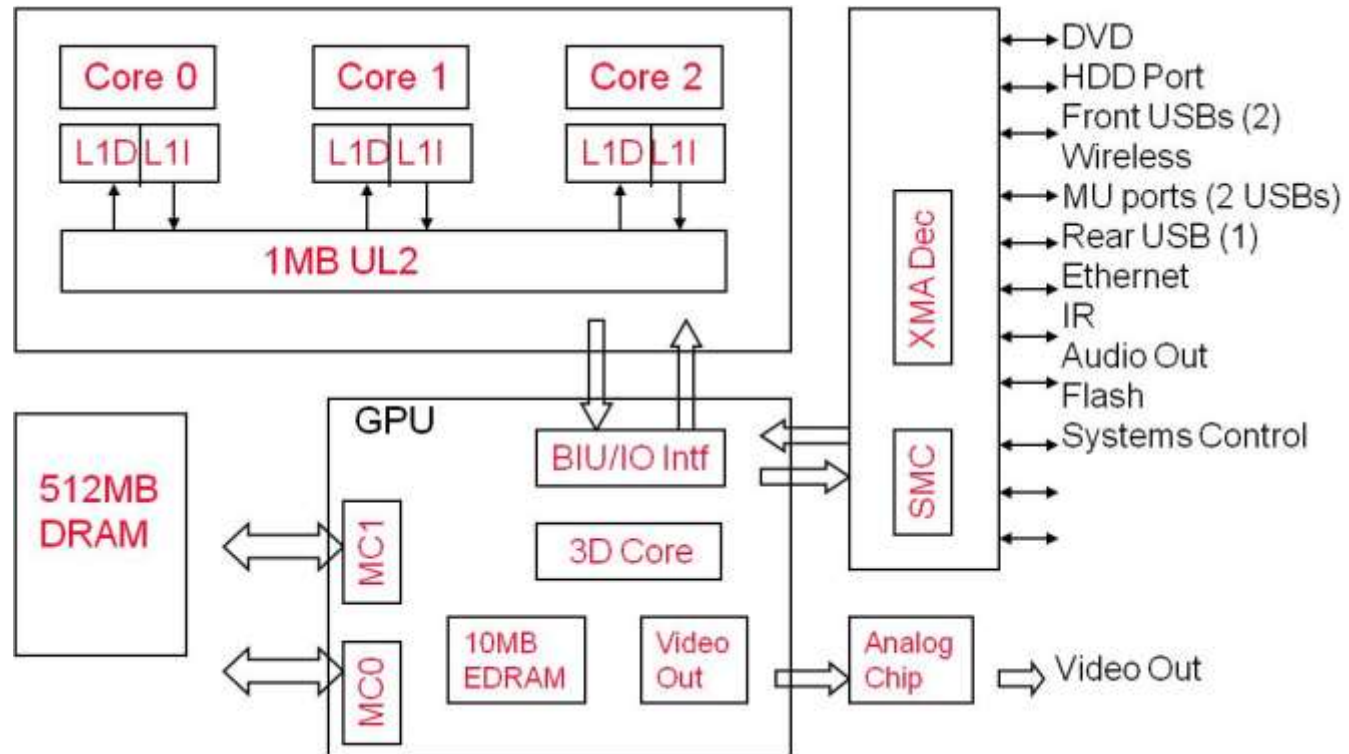
### **Xenon CPU - 3 cores SMT**

- 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

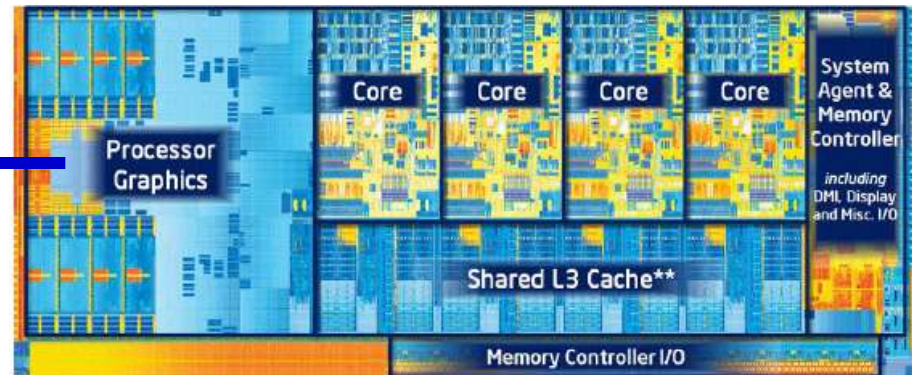
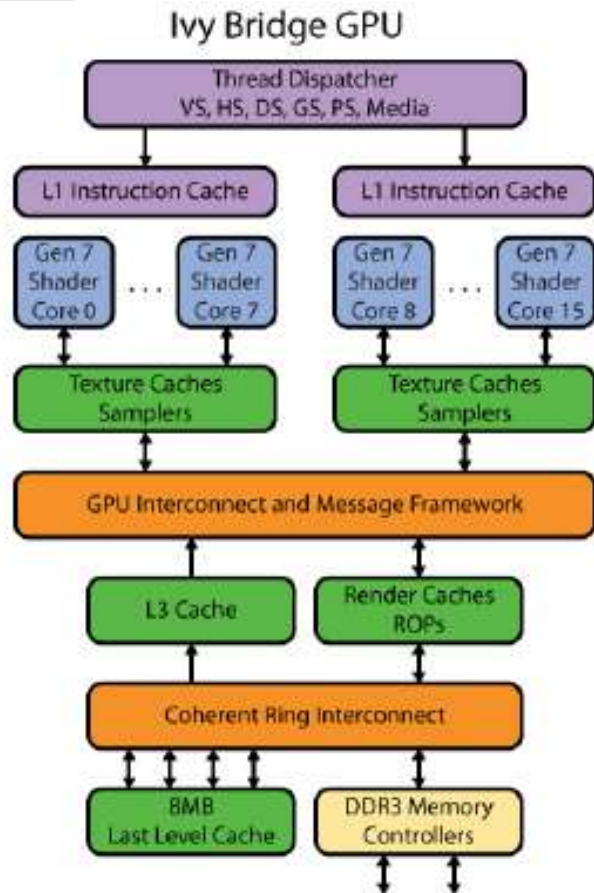
### **Xenos GPU ATI**

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

# Diagrama de Bloco do Xenon



# Outro Exemplo: Multicore Intel I7



## ■ Plataforma heterogênea

### CPU – 4 cores

- 3 níveis de cache
- Cache L3 compartilhada entre cores e GPU integrada

### Intel HD Graphics 4000 GPU

- 350 – 1350 MHz
- 16 shader cores

# Exemplo de GPU: NVIDIA Tesla

