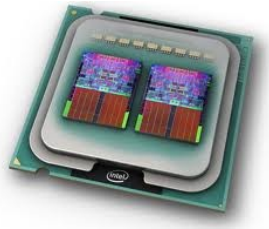
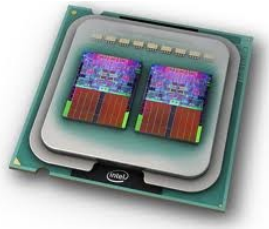


Programação de Sistemas

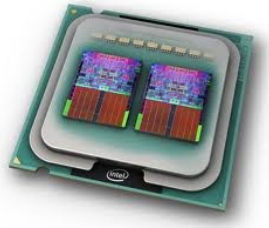
Multi-núcleos

Introdução (1)



- **[Def] núcleo** (“core”**)** é uma unidade de processamento. O **multicore** (Dual, Quad,...) possui 2 ou mais núcleos que
  - residem em diferentes pastilha de silício (“chip”) encapsuladas no mesmo circuito e ligadas por um barramento local
  - residem na mesma pastilha de silício (“chip”).
    Esta é a forma mais usada, porque os barramentos são mais lentos.
- Teoricamente um Dual core tem eficiência dupla de um núcleo singular. Na realidade, os ganhos só chegam aos 50%

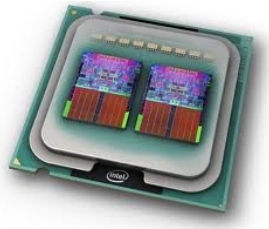
Introdução (2)



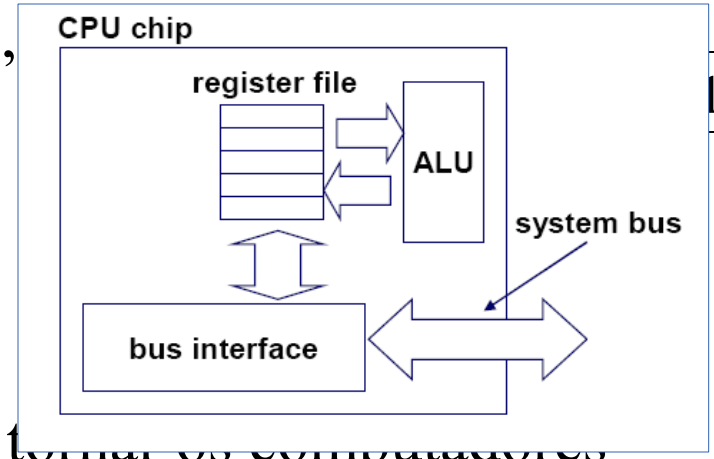
- Calendarização
  - Primeiro processador com dois núcleos, Power 4, criado pela IBM em 2000.
  - Primeiro Dual core da Intel distribuído em Jan 2006.
  - O MIT desenvolveu o Tile64 com 64 núcleos e comercializado pela empresa Tilera.

- Processadores multi-núcleo são MIMD: executam threads distintas (Multiple Instructions), operando em diferentes partes de memória (Multiple Data).

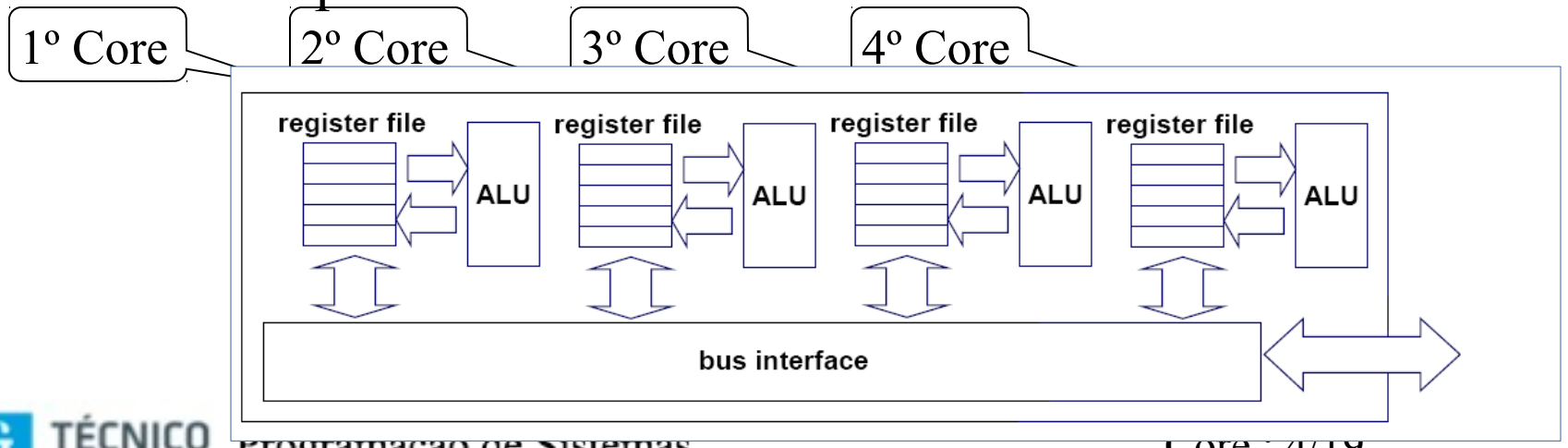
Arquitectura multi-núcleo (1)



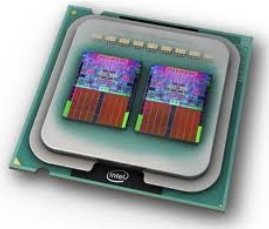
1. Os computadores “clássicos” [...] le core são de núcleo singular.



2. Múltiplos núcleos permitem tornar os computadores mais potentes.



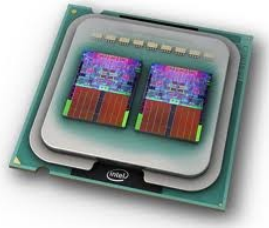
Arquitectura multi-núcleo (2)

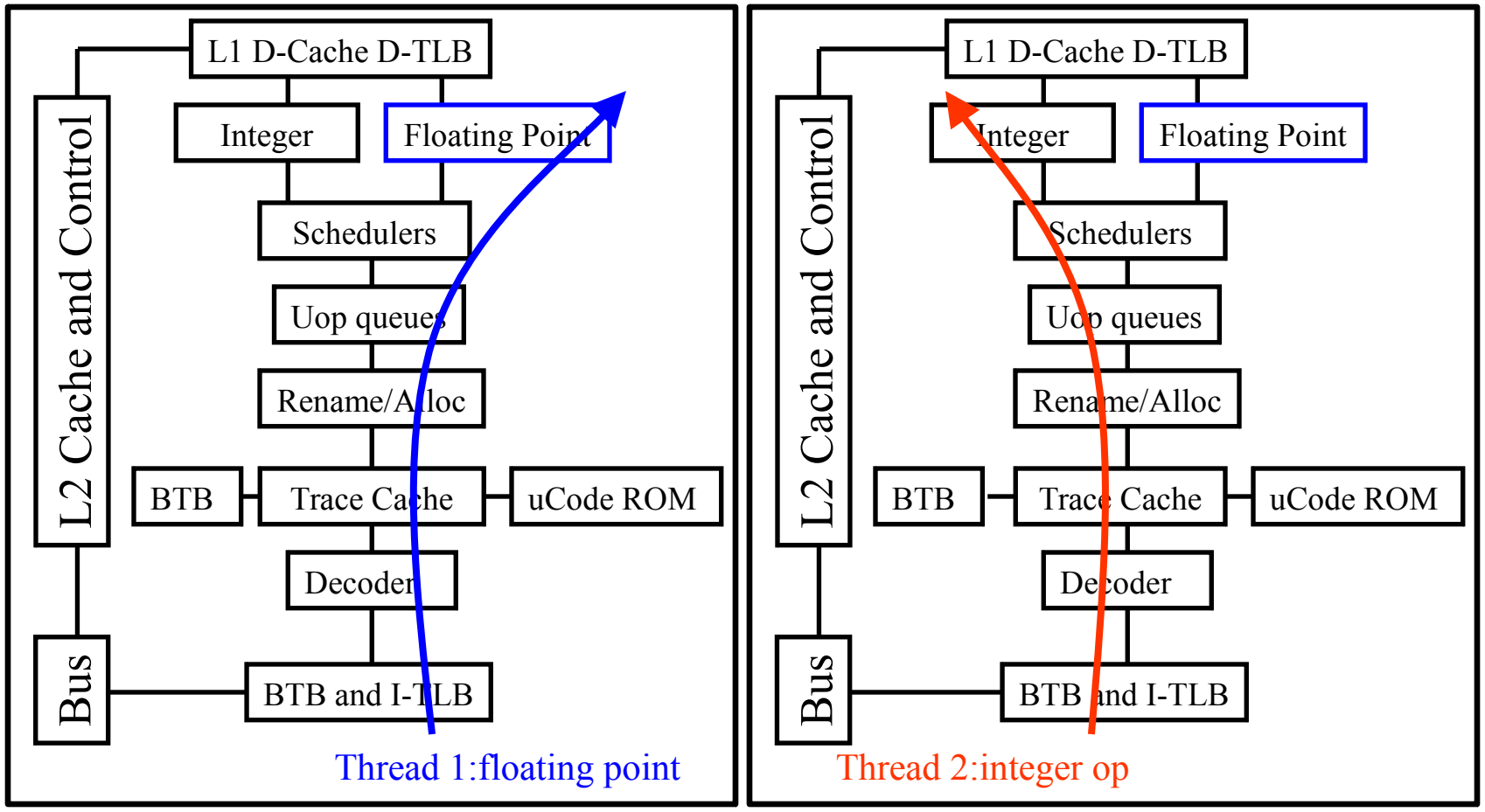


- Os núcleos partilham a memória central. As memórias cache são
  - L1 – privadas
  - L2 – privadas nalguns núcleos, partilhadas noutros.
- O suporte de várias núcleos é designado SMP-Symmetric Multiprocessing
  - Disponibilizado no kernel 2.4.17
  - Usado por omissão no Ubuntu 7.10 e Fedora Core 6.

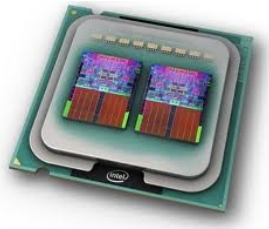
- As *threads* correm de forma independente em cada núcleo.

Arquitectura multi-núcleo (3)





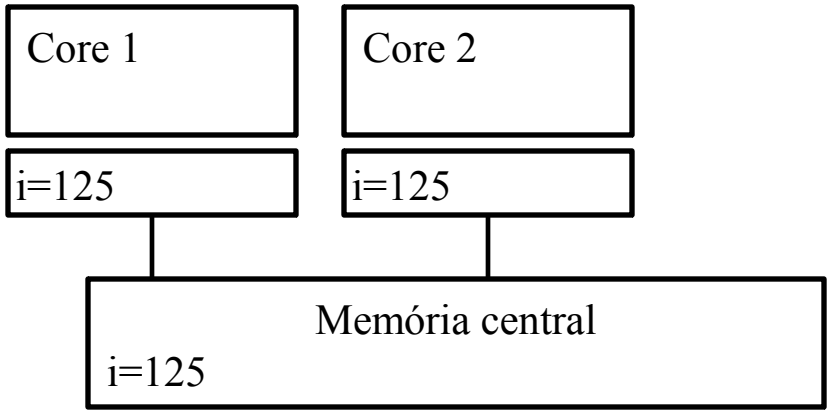
Coerência de cache (1)



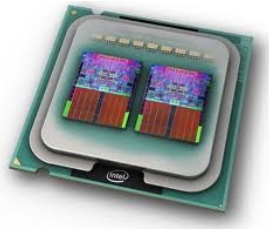
- Possuindo cada núcleo memória cache privada, coloca-se o problema de coerência.

Exemplo: seja a variável i, com valor 125.

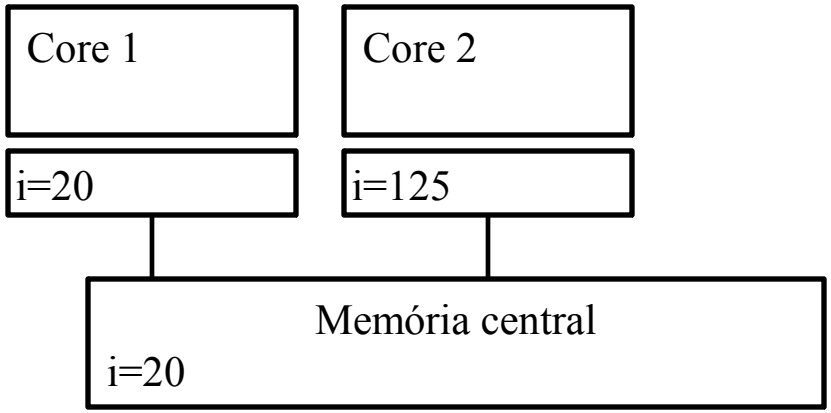
1) Núcleos 1 e 2 lêm conteúdo de i.



Coerência de cache (2)



2) Núcleo 1 actualiza i para 20.

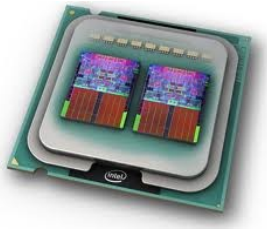


3) Núcleo 2 lê valor de i da sua cache, **que não é coerente**!

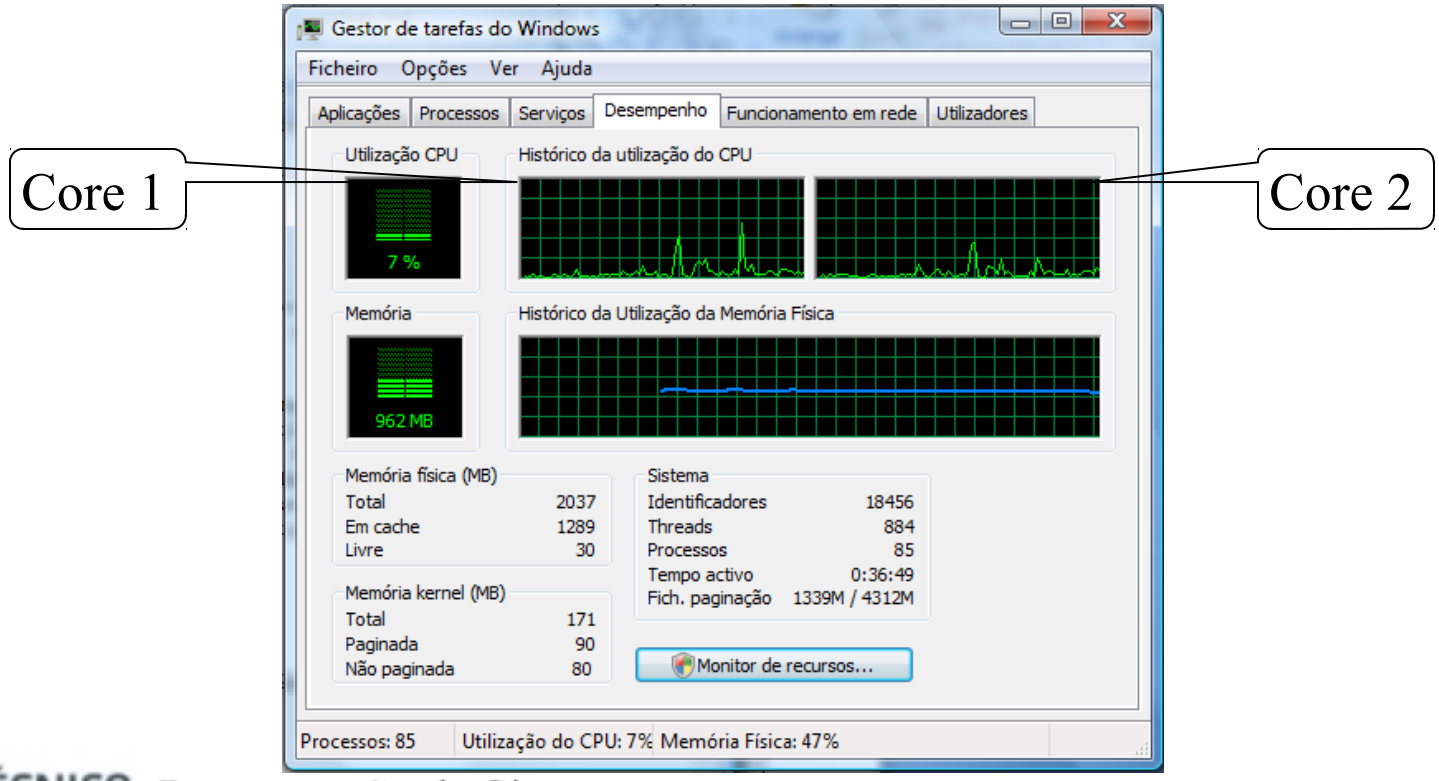
Soluções: ao alterar uma variável,

- o núcleo envia a todos os outros núcleos uma indicação de não validade (menos tráfego nos casos de múltiplas alterações).
- o núcleo envia a todos os outros núcleos o valor actualizado.

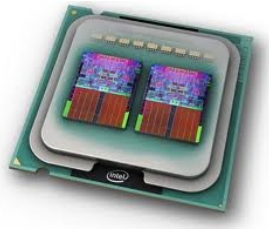
Estado de núcleos



- O leitor pode aceder no Windows, através do gestor de tarefas, ao estado dos dois núcleos.

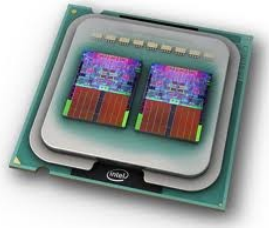


Vantagens/Inconvenientes do SMP



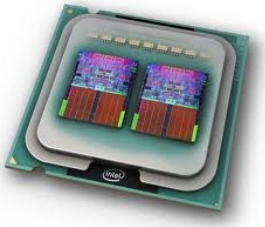
- As aplicações que beneficiam do SMP:
  - Servidores de bases de dados.
  - Servidores Web.
  - Compiladores.
  - Aplicações científicas.
  - Compressão de áudio e vídeo.
- Há aplicações para as quais o SMP pode ser mais lento:
  - Processadores de texto.
  - Jogos.

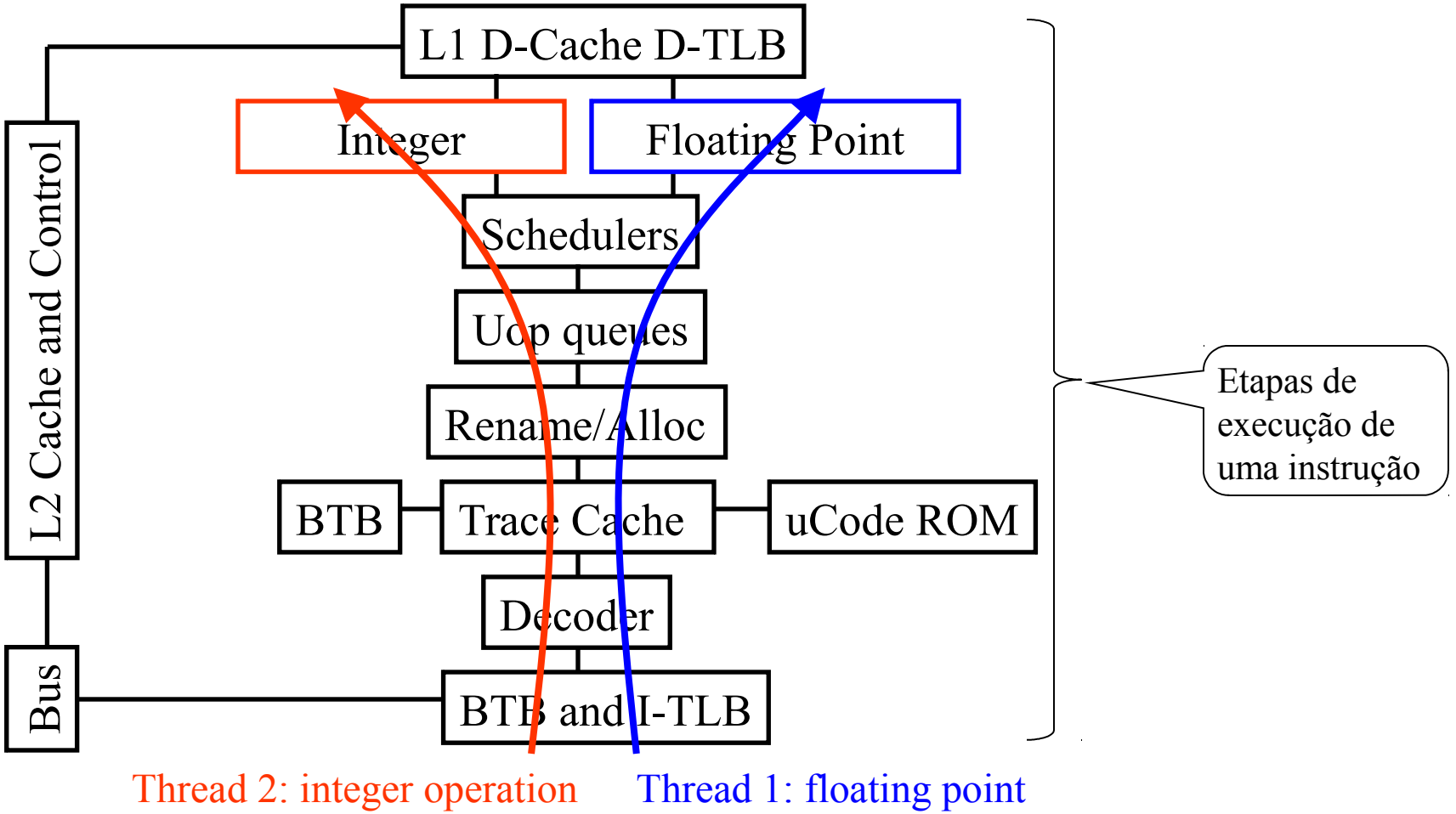
Multithreading (1)



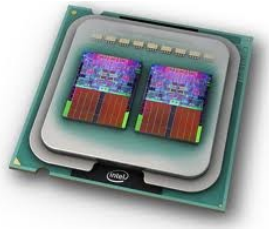
- O *pipeline* do núcleo pode ficar parado ("stalled") devido a:
  - Espera do resultado da unidade aritmética (ex: multiplicação em vírgula flutuante é muito demorada).
  - Espera de chegada de um dado vindo da memória.
- Na arquitectura **SMT**-Simultaneous Multithreading, várias threads operam simultaneamente no mesmo núcleo.
  - Uma thread opera inteiro, outra opera vírgula flutuante. Atenção! Duas threads não podem efectuar, no mesmo núcleo, o cálculo de inteiros ao mesmo tempo!
  - Tempos mortos das unidades do pipeline de um fio de execução podem ser ocupados por outro fio de execução.

Multithreading (2)





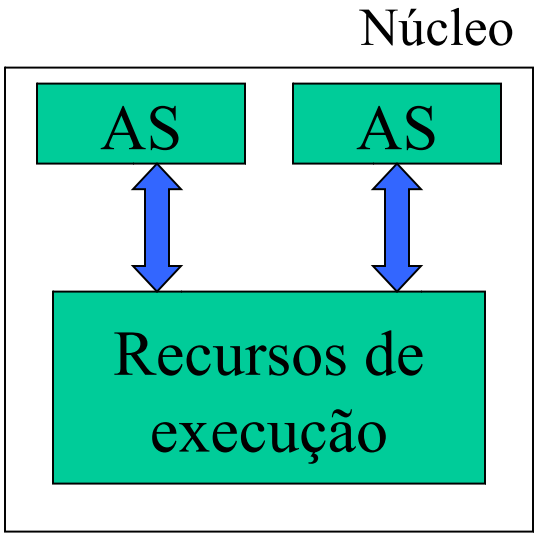
Multithreading (3)



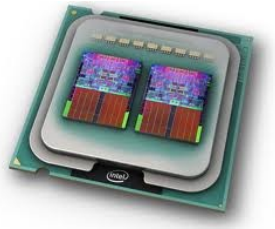
- A Intel implementou SMT (primeiro no Xeon em 2002, depois no Pentium 4), designada por *Hyper-Threading*.

Nota: Segundo a Intel, implementação do HT melhorou desempenho 15%-30% para aumento na superfície de apenas 5%.

- Cada núcleo possui
  - Processadores lógicos AS-Architecture State, cada um com registos de uso geral, cache L1 de 16KB, controlador de interrupções APIC.
  - Um processador físico comum, com barramento de sistema, cache L2 de 1MB, ALU e FPU.

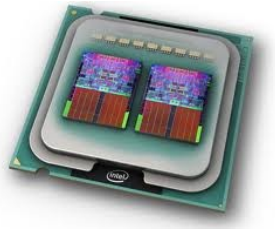


Programação paralela (1)



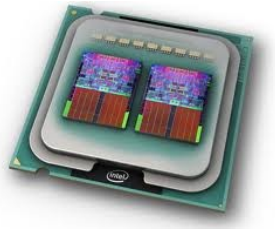
- Neste capítulo, a concorrência (processos e fios de execução) foi programada por funções do sistema operativo.
  Esta abordagem revela alguns inconvenientes:
  - Exige conhecimentos especializados de programação, fora do âmbito de trabalho de alguns utilizadores (físicos, meteorologistas, projectistas de automóveis,...).
  - Dirigidos para multiprocessamento, pouco praticáveis para sistemas paralelos e distribuídos.
- Uma alternativa é mascarar, o mais possível, o paralelismo no compilador ou na linguagem de programação.

Programação paralela (2)

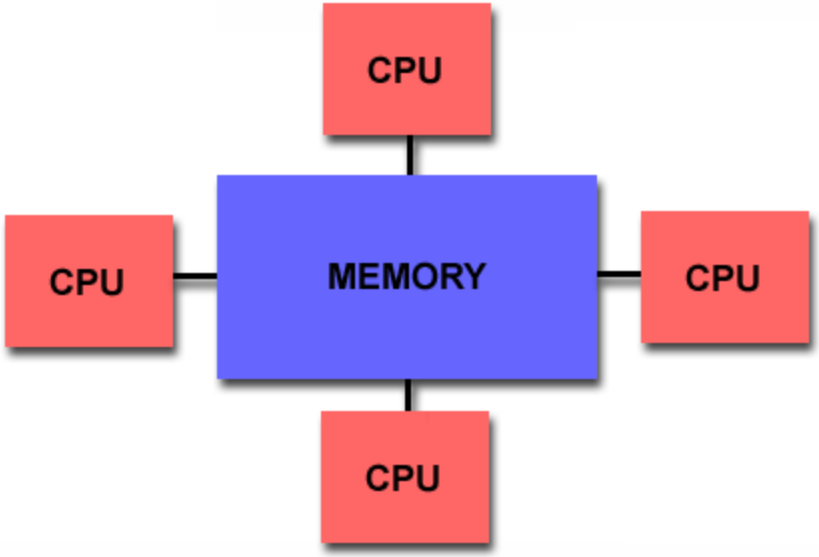


1. Compiladores estendidos detectam paralelismo em programas sequenciais e produzam programas executáveis em paralelo.
   Existem vários tipos de paralelismo
   - Dados: mesma operação aplicada a dados distintos
     ```
     for(i=0;i<10;i++) a[i]=b[i]+c[i];
     ```
   - Funcional: operações distintas exercidas sobre dados distintos
     ```
     a=2; b=3;
     m=(a+b)/2; s=(a^2+b^2)/2;
     ```
2. Linguagem de programação (ex: FORTRAN e C) são estendidas com APIs dedicadas a determinadas arquitecturas.

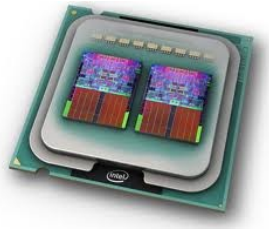
Programação paralela (3)



A. OpenMP (http://www.openmp.org)
   - API orientada a aplicações científicas (“number crunching”).
   - Particularmente adaptada a arquitecturas de memória partilhada multi-plataforma (Unix e Windows).



Programação paralela (4)

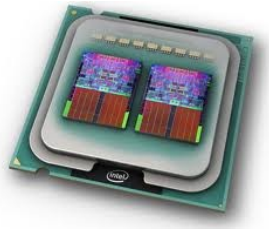


- OpenMP estende compiladores muito divulgados (gcc,Visual Studio) através de directivas pragma (C) ou comentários especiais (FORTRAN).
  - Directivas de paralelização (região, for)
  - Directivas de ambiente de dados (partilhado, privado, *threadprivate*, *reduction*, ...)
  - Directivas de sincronização (barreiras, *critical*, ...)

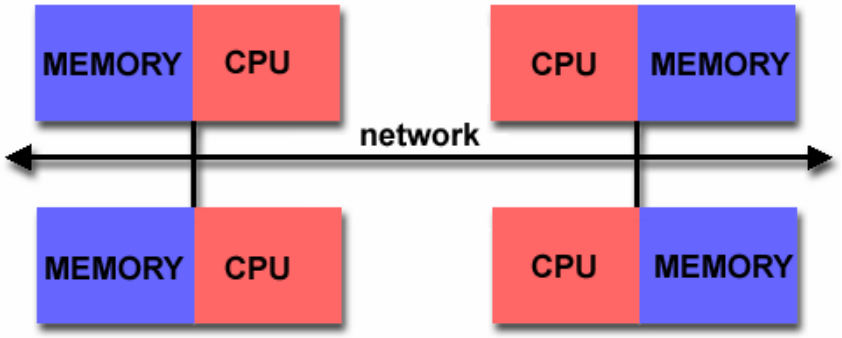
Exemplo: lançadas *threads*, cada uma executando um subconjunto de iterações. Todas as *threads* esperam no final do ciclo *for*.

```
#include <omp.h>
int main() {
/* … */
#pragma omp parallel
    #pragma omp for
    for( … ) { … }
```

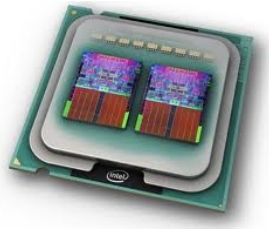
Programação paralela (5)



B. MPI-Message Passing Interface (http://http://www-unix.mcs.anl.gov/mpi/)
   - Baseado no envio de mensagens
   - Adaptada a arquitecturas de memória distribuída, pode também ser usado em arquitecturas de memória partilhada.
   - O MPI tem de ser inicializado, por forma a serem abertas todas as conexões TCP entre os processadores.



Programação paralela (6)



O programa possui o seguinte formato:

```
#include <mpi.h>
     int main(int argc,char **argv) {
     /* … */
     MPI_Init(&argc,&argv);          /* inicializa MPI */
     MPI_Comm_size(MPI_COMM_WORLD,&size); /* indica número processos*/

     /* … */
     /* envia mensagem */
     MPI_Send(msg,strlen(msg)+1,MPI_CHAR,dest,tag,MPI_COMM_WORLD);
     /* recebe mensagem */
     MPI_Recv(message,100,MPI_CHAR,source,tag,MPI_COMM_WORLD,&status);

     MPI_Finalize();              /* finaliza MPI */
     /* … */
}
```