# Programação Paralela e Distribuída

Aula 1

Profa.: Liria M. Sato

# Objetivo da disciplina

Mostrar como explorar em programas os recursos oferecidos pelo processamento paralelo em sistemas multiprocessadores e multicomputadores

# Conteúdo

- arquiteturas paralelas
- modelos de programação e processamento
- linguagens e ferramentas
- mecanismos de processamento: processos e threads
- comunicação e sincronização
- programação em sistemas com memória compartilhada
- programação em sistemas distribuídos, com passagem de mensagem

# Conteúdo da Aula

- Introdução
- Arquiteturas paralelas
- Como explorar o paralelismo
- Paradigmas de programação
- Threads e Processos

# Introdução

Crescimento da demanda de processamento

- desenvolvimento de sistemas de alto desempenho

Implementações possíveis de aplicações envolvendo grande capacidade de processamento:

- simulações de crash
- simulação de fenômenos físicos
- visualização científica

# Como obter alto desempenho?

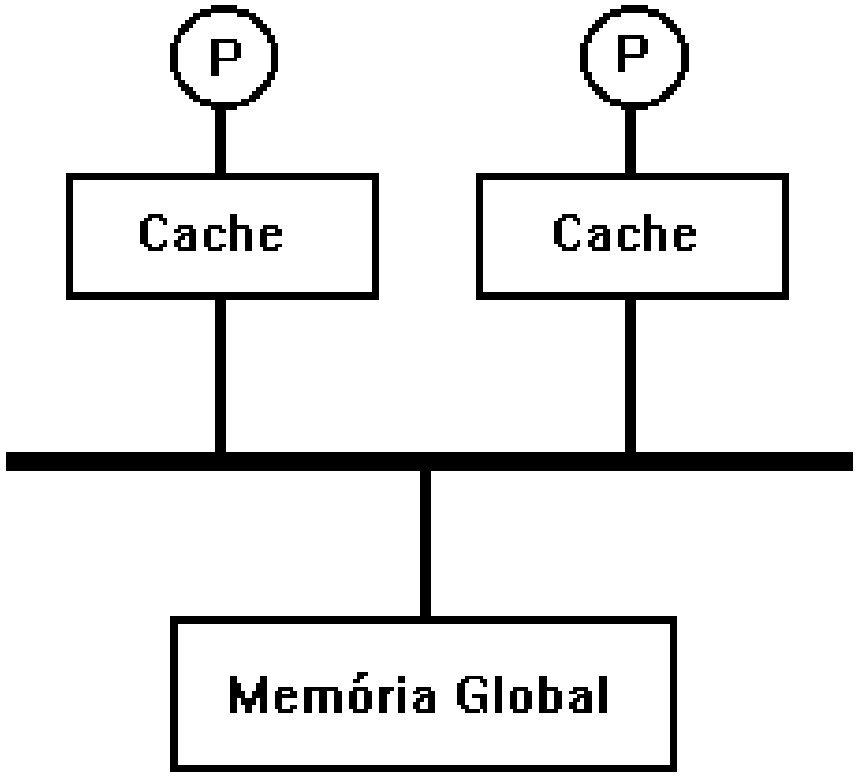
aumento do desempenho do processador

- aumento do clock $\Rightarrow$ aumento de temperatura
- melhorias na arquitetura $\Rightarrow$ processadores RISC, vetoriais, superescalares
- melhoria no acesso à memória: importância da hierarquia da memória
- (registradores, cache, memória principal)

utilização de múltiplos processadores

- paralelização da execução de operações

# Multiprocessador com memória compartilhada



Problemas

- dividir a tarefa entre os processadores
- sincronização
- comunicação

Necessidade

linguagens, compiladores e bibliotecas especiais

$\Downarrow$

utilização adequada dos recursos de processamento

# ARQUITETURAS PARALELAS

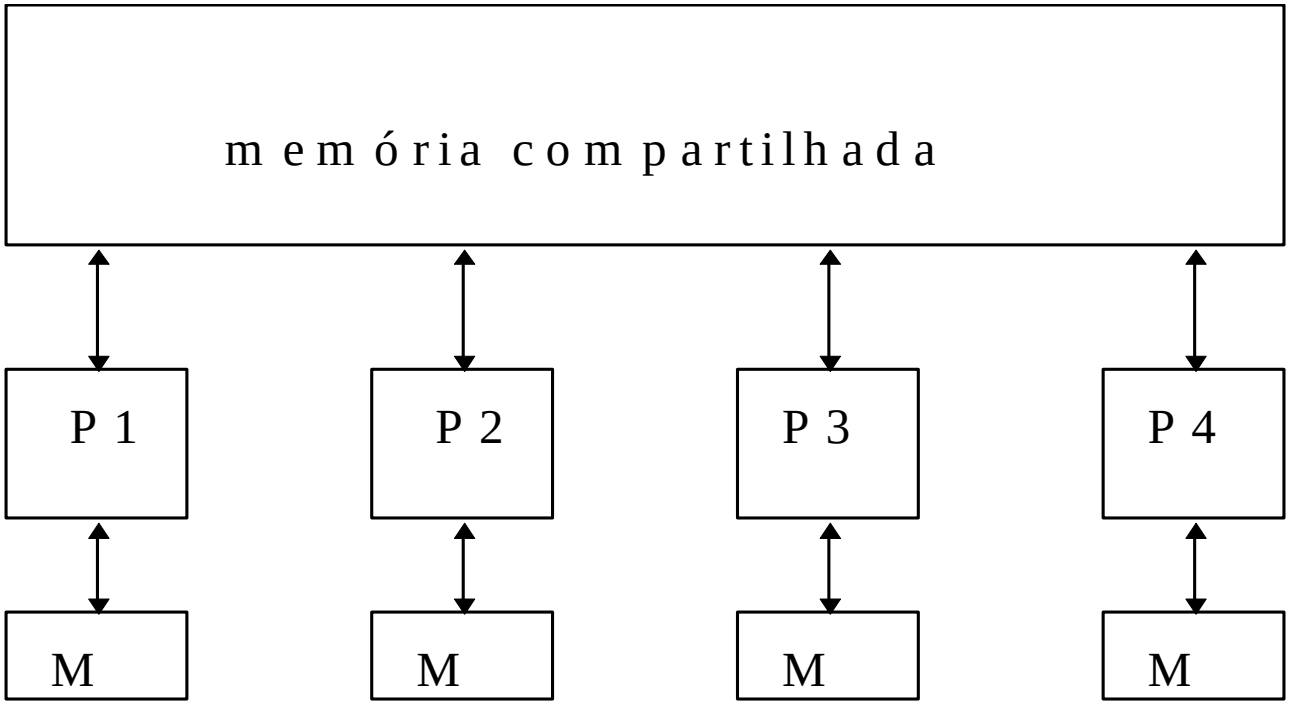
Taxonomia apresentada por Gordon Bell

- fluxo de instruções único
  - fluxo de dados único: CISC, RISC, Superescalares, VLIW RISC,...
  - fluxo de dados múltiplos: vetoriais, SIMD
- fluxo de múltiplas instruções (MIMD)
  - multiprocessador: memória compartilhada
    - acesso à memória não uniforme (NUMA)
    - acesso à memória uniforme (UMA)

# Arquiteturas paralelas

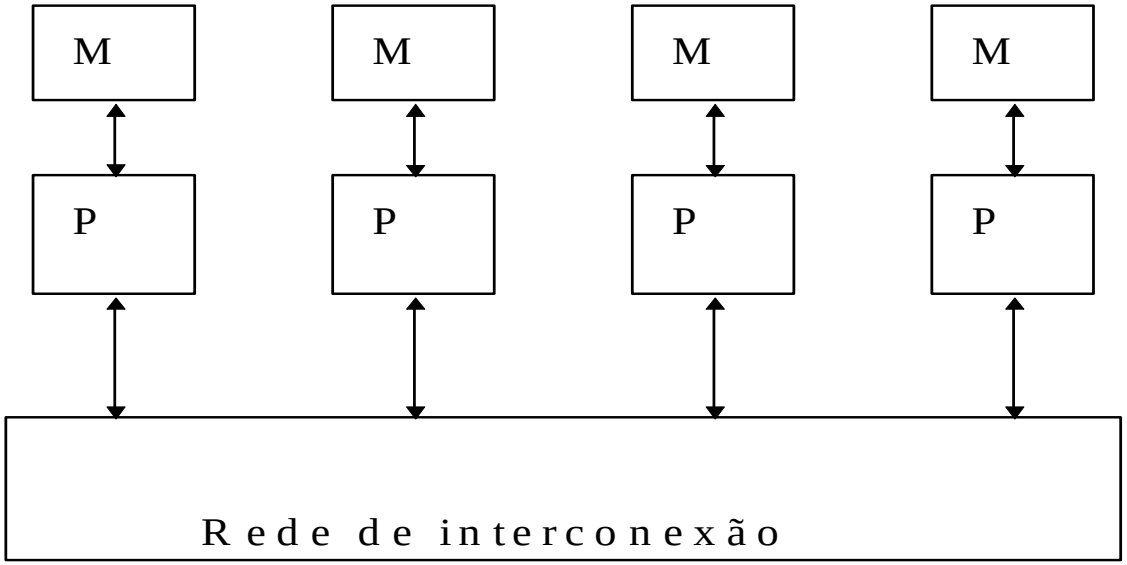
- multicomputadores
  - chaveamento :
    - computadores com interconexão de alta velocidade
  - rede:
    - aglomerados (“clusters”)
    - rede local
    - área extensa

# Multiprocessadores

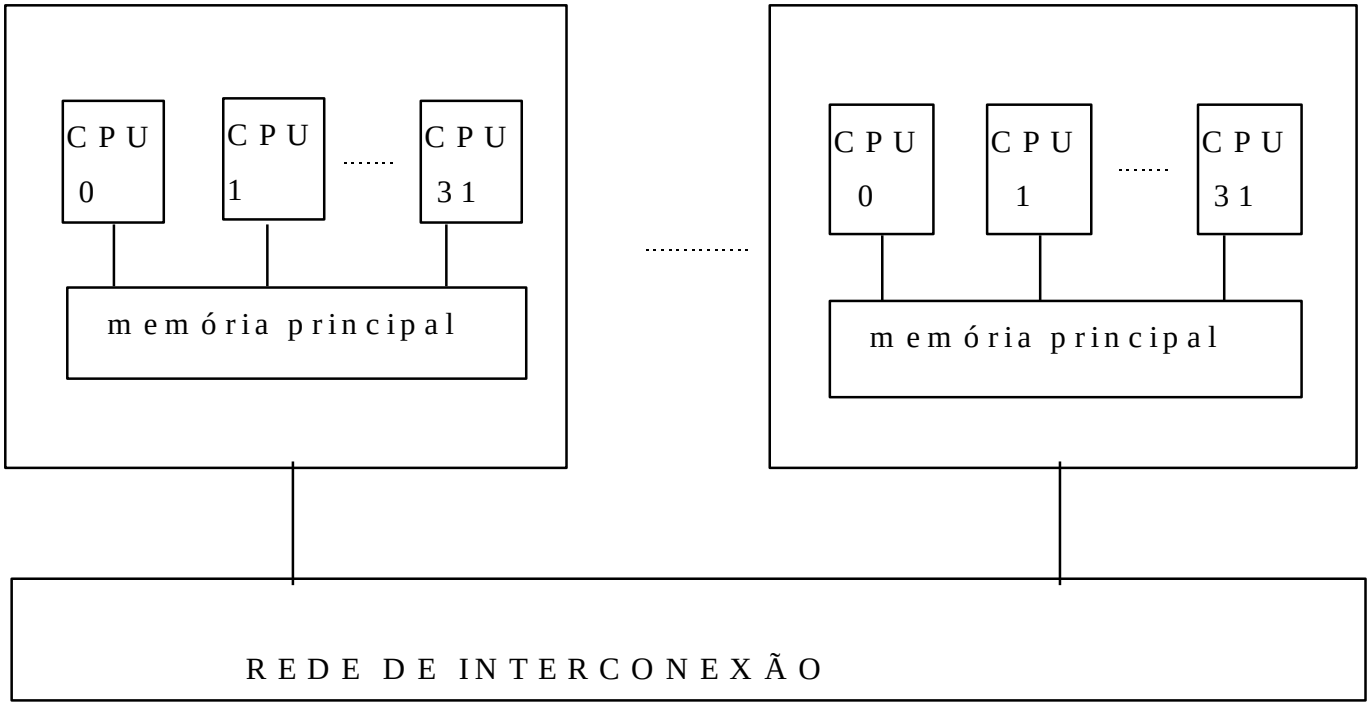


# Multicomputadores: passagem de mensagem

redes distribuídas: clusters, local, área extensa



# Arquitetura com aglomerados de processadores




Compartilhamento de memória: por hardware

# Processador Multicore

- Processadores Multicore
  - Processador:  múltiplos núcleos de processamento
  - Memória Compartilhada
- Co-processadores
  - Graphics Processing Unit (GPU)
  - Xeon-Phi

# COMO EXPLORAR O PARALELISMO

* explicitar o paralelismo no programa (programação paralela)

* utilizar compiladores paralelizantes (detecção automática de paralelismo implícito)

# PROGRAMAÇÃO PARALELA x PARALELIZAÇÃO AUTOMÁTICA

*Programação Paralela*:

* alta complexidade de programação
* sincronização entre tarefas
* análise de dependência de dados

```
(1)     A=B+1;
(2)     D=A+B*2;
```

(2) não paralelo (1): devido a dependência em (2) de (1)

* cabe ao usuário analisar e detectar fontes de paralelismo
* baixa portabilidade
* alta exploração de paralelismo
* possibilidade do usuário particionar o programa explicitamente

# Paralelização Automática

* usuário não necessita de conhecimentos de programação paralela

* utilização de programas já existentes

* não exploração de algumas fontes de paralelismo no programa

# PROGRAMAÇÃO PARALELA EXPLICITA

Formas de Programação Paralela Explícita:

- chamadas de rotinas de uma biblioteca de paralelismo
- diretivas do compilador
- construções paralelas de linguagens de programação paralela

```
/* execucao do calculo parcial */
forall i = 0 to MAXRET    {
  x = ((i-0.5) * largura);    /* calcula x */
  local_pi = local_pi + (4.0 / ( 1.0 + x * x));
  local_pi = local_pi * largura;
  /* atualização da variavel global */
  lock (&semaforo);
  total_pi = total_pi + local_pi;
  unlock (&semaforo);
  }
printf (”valor de pi = %d\n”, total_pi);
}
```

# DIRETIVAS DO COMPILADOR

Paralelismo explicitado: diretivas do compilador

Exemplo:OPENMP

- pragmas

```
#pragma omp diretiva [(modificador(lista..))]
```

Exemplos:

```
#pragma omp parallel
#pragma omp pfor iterate(i=0;maxits;1)
```

```
#define MAXRET  1000000
double total_pi = 0.0;
void main()
  {
  int i;
  double x, largura,
   local_pi;
  largura = 1/MAXRET;
#pragma omp parallel
   shared(total_pi,largura)
   private(i,x,local_pi)
{
#pragma omp pfor iterate (i=0;
   MAXRET; 1)
 for (i = 0; i < MAXRET; i++)
    {
    x = ((i-0.5) * largura);
              /* calcula x */
    local_pi = local_pi + (4.0 /
   ( 1.0 + x * x));
    }
  local_pi = local_pi * largura;
#pragma omp critical
  {
  total_pi = total_pi +
   local_pi;
  }
}
  printf (”valor de pi = %d\n”,
   total_pi);
}
```

# BIBLIOTECA DE PARALELISMO

* especificação do paralelismo:
  - chamadas de rotinas de uma biblioteca de paralelismo

Exemplo: chamadas do S.O (processos e threads)

Bibliotecas oferecem:

* alocação de memória para dados compartilhados;
* criação de processos ou threads;
* identificação de processos ou threads;
* exclusão mútua;
* sincronização de processos ou threads.

```
void *PIworker(void *arg)
{
   int i,myid;
   double sum,mypi,x;
   myid=(int *)arg;
   sum=0.0;
   for
   (i=myid+1;i<=n;i+=num_thread
   s)  {
       x=w*((double)i-0.5);
       sum+=f(x);
   }
   mypi=w*sum;
/* reduce value */
pthread_mutex_lock(&red_mutex);
   pi+=mypi;
   printf("pi-local%f\n",pi);

   pthread_mutex_unlock(&reduct
   ion_mutex);
   return(0);
}
```

```
void main
{
   int i;
   int numthreads=4;
   w=1.0/(double)n;
   pi=0.0;
   tid=(pthread_t *) calloc(num_thr
   eads,sizeof(pthread_t));
   pthread_mutex_init(&red_mutex,NU
   LL)
/* create threads */
   for (i=0;i<num_threads;i++) {
       if (pthread_create(&tid[i],N
   ULL,PIworker,(void *) i)){
           fprintf(stderr,"Cannot c
   reate thread %d\n",i);
           exit(1);
          }
   }
/* join threads */
   for (i=0;i<num_threads;i++) {
        pthread_join(tid[i],NULL);
   }
 printf("PI=%16f\n",pi);
}
```

# MODELOS E LINGUAGENS DE PROGRAMAÇÃO PARALELA

**LINGUAGENS**

especificação de paralelismo:

construções sintáticas

**Linguagem Ideal:**

construções sintáticas para expressar paralelismo em todos níveis de granularidade

* Concurrent C [Gehani]: "multitasking"
* Outras: "loops" e/ou blocos paralelos
* CPAR: vários níveis

# Paradigmas de programação

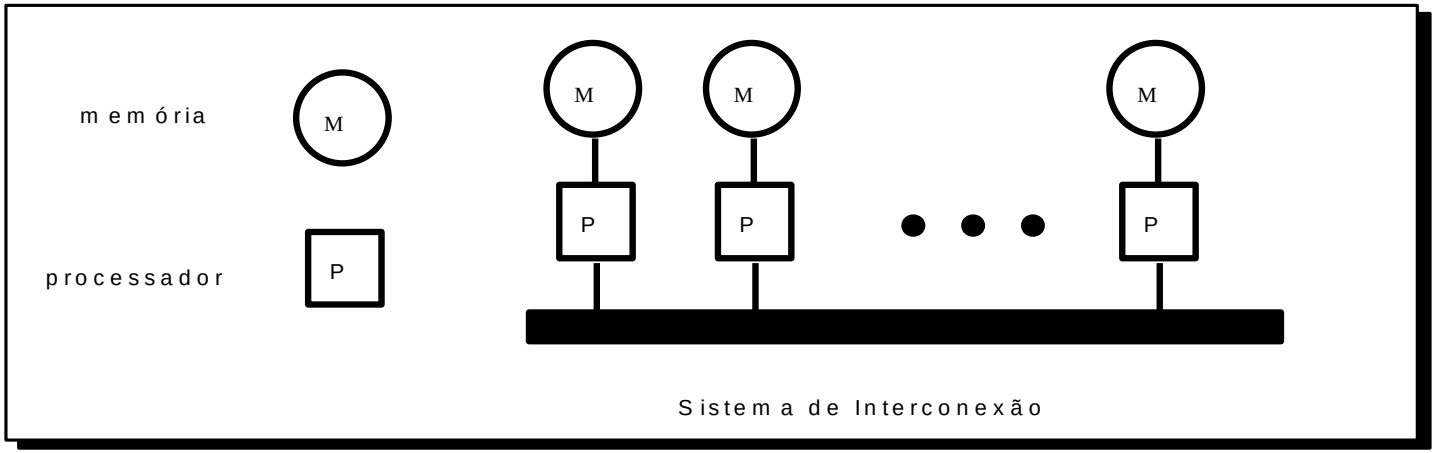
* funcional
* procedimental

  variáveis compartilhadas

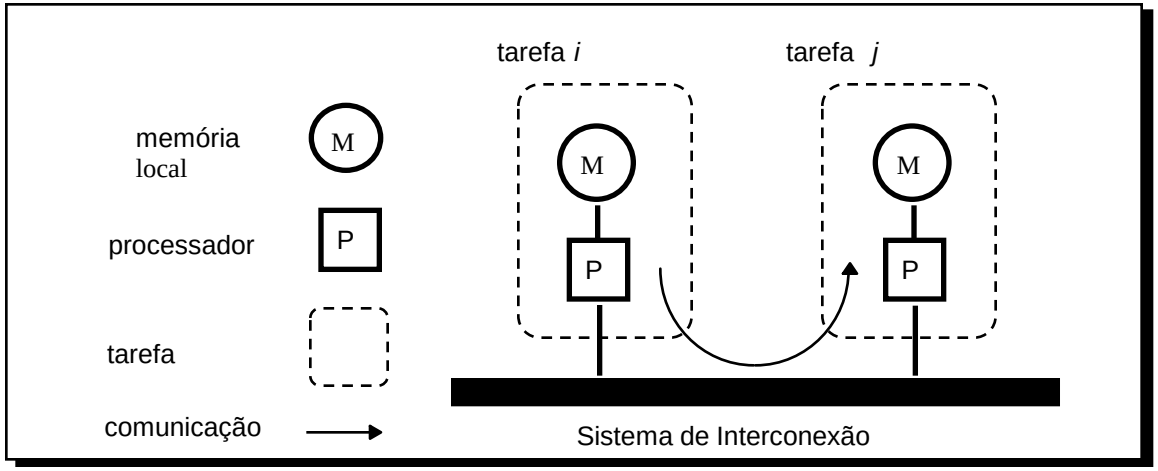
  (sist.distribuídos: DSM)

* passagem de mensagem
* programação lógica
* programação orientada a objetos

# Programação Procedimental por Passagem de Mensagem

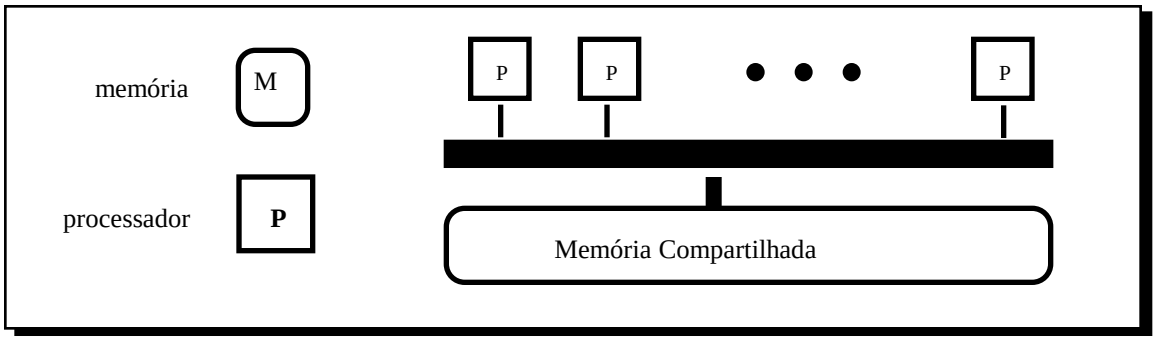


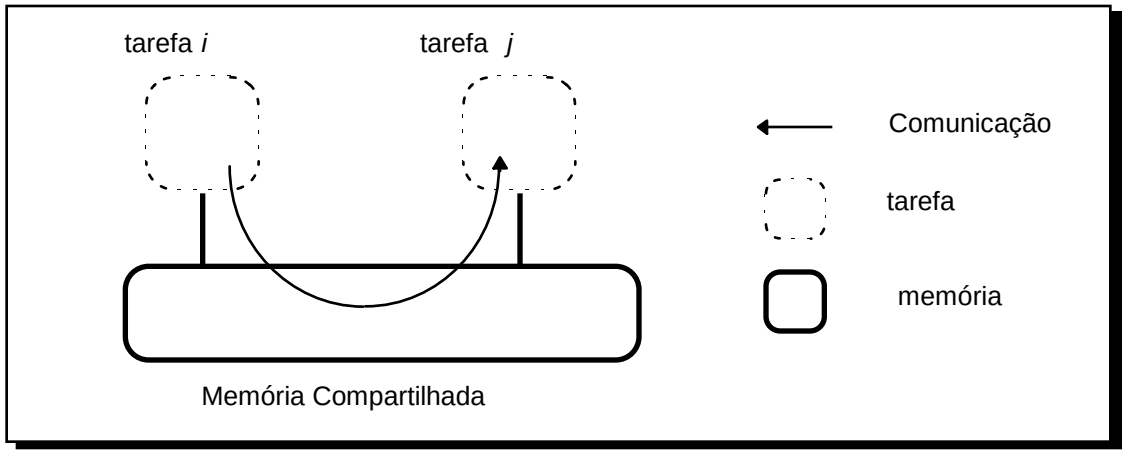

## Comunicação entre tarefas



# Programação Procedimental com memória compartilhada

## Arquitetura



# PROGRAMAÇÃO PROCEDIMENTAL BASEADA EM VARIÁVEIS COMPARTILHADAS

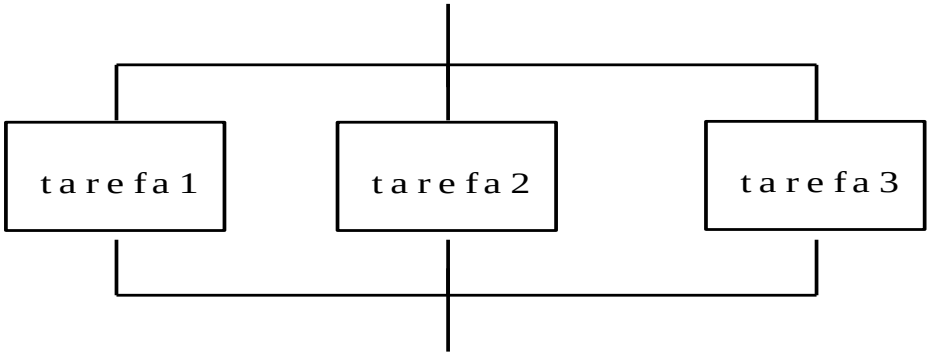
Fontes de Paralelismo:

- Subrotinas
- Laços
- Blocos Básicos
- Comandos ou operações
- Instruções

# Macrotarefas e Microtarefas

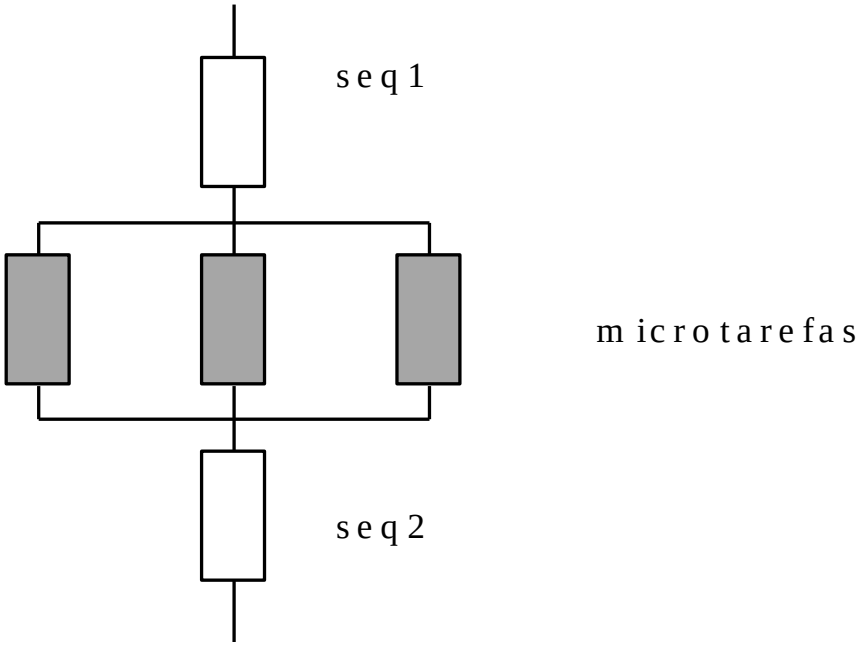
"Multitasking" (ou "macrotasking")

- Particiona o programa em múltiplas tarefas (macrotarefas) executados paralelamente pelos múltiplos processadores .



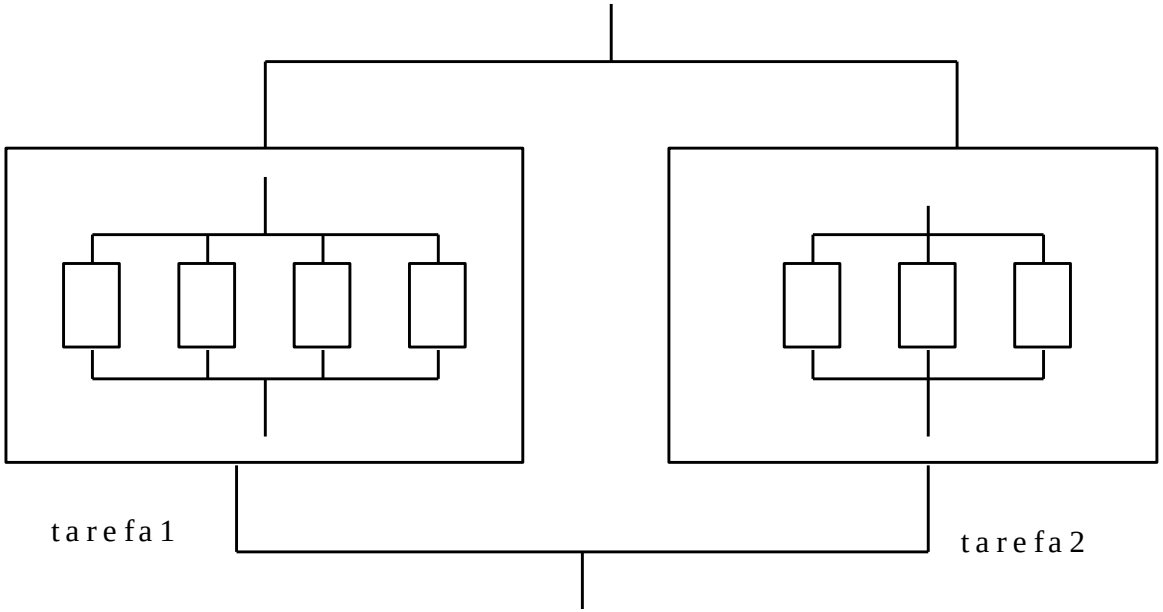
“Microtasking”

- laços paralelos: iterações são executadas paralelamente pelos múltiplos processadores
- blocos de comandos paralelos



“Macrotasking e Microtasking”

*programa*: múltiplas macrotarefas, sendo cada uma particionada em microtarefas



# Threads e Processos

- thread: é um fluxo de controle seqüencial em um programa.
- programação multi-threaded: uma forma de programação paralela onde várias threads são executadas concorrentemente em um programa. Todas threads executam em um mesmo espaço de memória, podendo trabalhar concorrentemente sobre dados compartilhados.

# Threads e Processos

- multi-threaded programming: todas as threads compartilham o mesmo espaço de memória e alguns outros recursos, como os descritores de arquivos.
- multi-processing (UNIX): processos executam sobre seu próprio espaço de memória. O compartilhamento de dados se obtém através de funções que fazem com que endereços lógicos apontem  para o mesmo endereço físico.

# Pthreads e Processos

- Processos:

  exemplos: teste-proc.c,  proc-compartilha.c

  erro: proc-compartilha-erro.c

- POSIX :  pthreads

  exemplos: teste-pthread.c,

  pthread-compartilha.c

  pi-pthreads.c

- Compilação:

gcc –o teste-pthread teste-pthread.c -lpthread