Programação Paralela e Distribuída

AULA 2

Profa.: Liria M. Sato

Conteúdo

- O Sistema de Programação e Processamento CPAR
- Memória compartilhada
- Paralelismo homogêneo e heterogêneo

O SISTEMA DE PROGRAMAÇÃO E PROCESSAMENTO CPAR

OBJETIVOS:

* expressar com clareza e facilidade o máximo de paralelismo

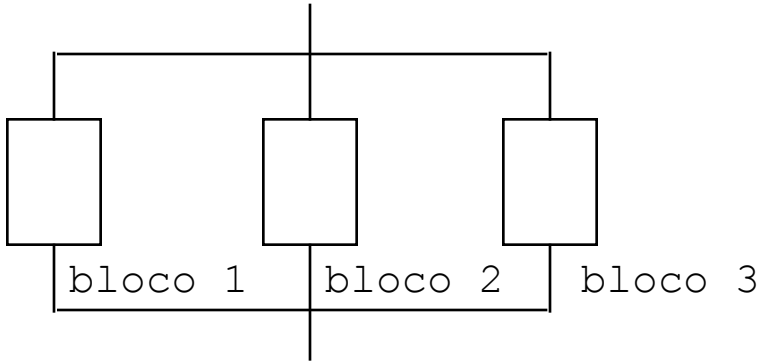
linguagem CPAR

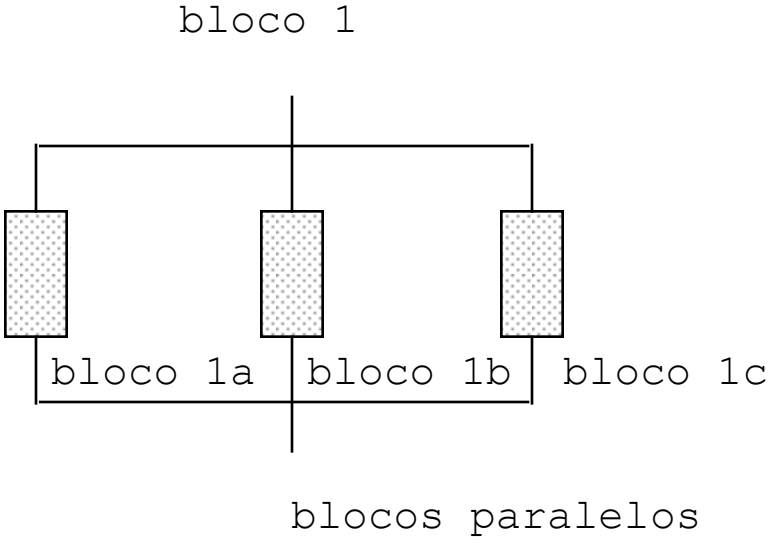
* equilíbrio entre facilidade de programação e eficiência

O MODELO DE PROGRAMAÇÃO

1. paralelização da função principal: blocos paralelos
   conteúdo do bloco:

* tarefas
* elementos sequenciais





2. "multitasking"("macrotasking") e "microtasking"

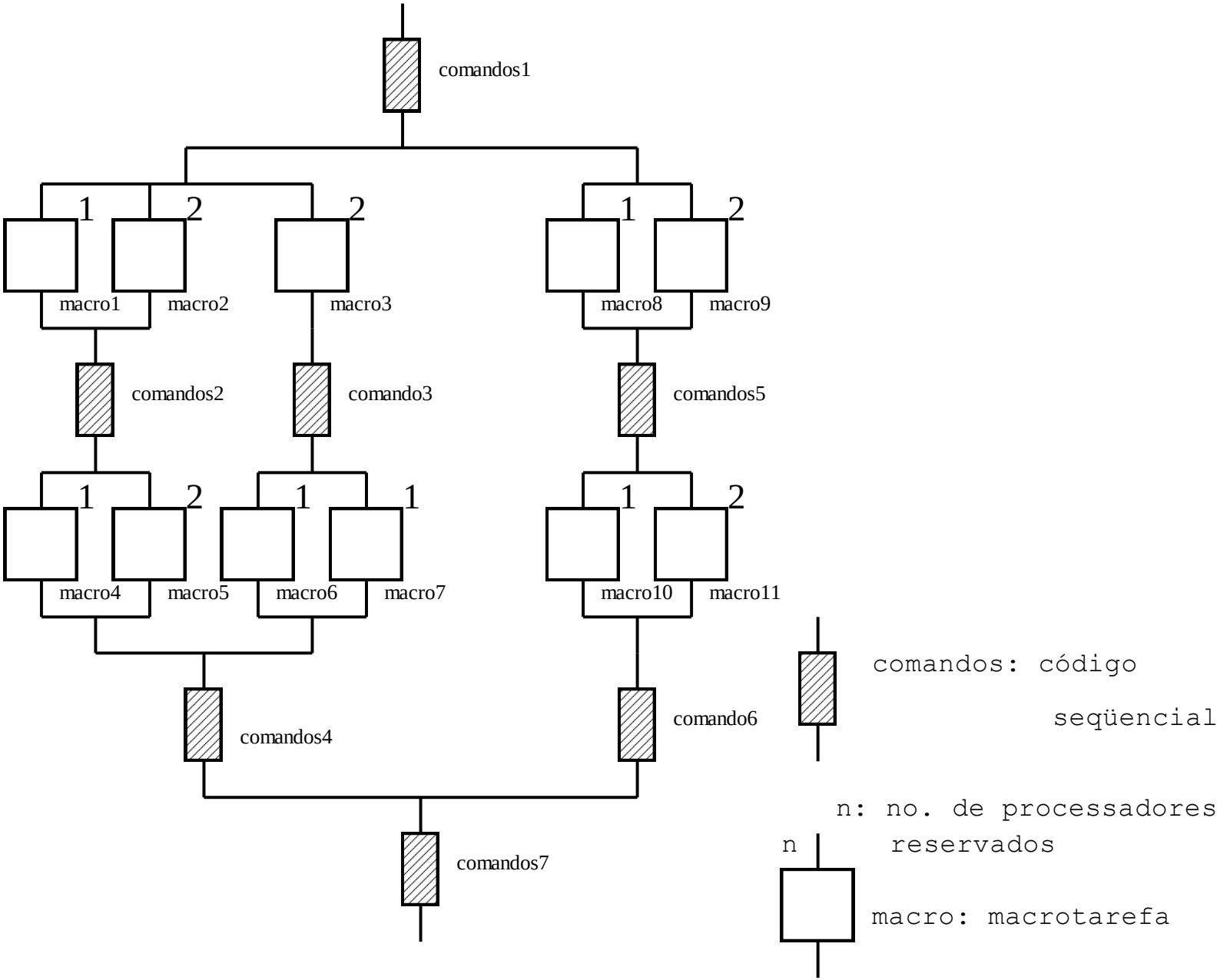
"microtasking":

* Loops paralelos:

    iterações executadas por threads já em execução

CPAR: threads responsáveis pela execuçao da tarefa (macrotarefa)

* colocadas em execução na sua criação.
* estas threads: executam as microtarefas


comandos1
1
2
2
1
2
macro1
macro2
macro3
macro8
macro9
comandos2
comando3
comandos5
1
2
1
1
1
2
macro4
macro5
macro6
macro7
macro10
macro11
comandos4
comando6
comandos7
comandos: código seqüencial
n: no. de processadores reservados
n
macro: macrotarefa

A LINGUAGEM CPAR

* uma extensão da linguagem C
* possui primitivas para explicitar blocos paralelos;
* possui primitivas para a declaração e a execução de macrotarefas;
* possui primitivas para explicitar microtarefas;
* permite a declaração de variáveis locais e de variáveis compartilhadas globais e locais à macrotarefa.
* fornece um mecanismo de exclusão mútua (monitor): utilização segura da memória compartilhada, denominado monitor;
* a comunicação entre macrotarefas: memória compartilhada;
* a sincronização entre macrotarefas: rotinas de biblioteca.

MACROTAREFAS

1. declaração:

task spec nome_macrotarefa(parametros)

nome_macrotarefa: nome da macrotarefa

parametros: é opcional

declarações das entradas: é opcional. É utilizada na passagem de mensagem entre macrotarefas**.**

task body nome_macrotarefa(parametros)

declaração de parametros

{ declarações de variáveis

comandos

}

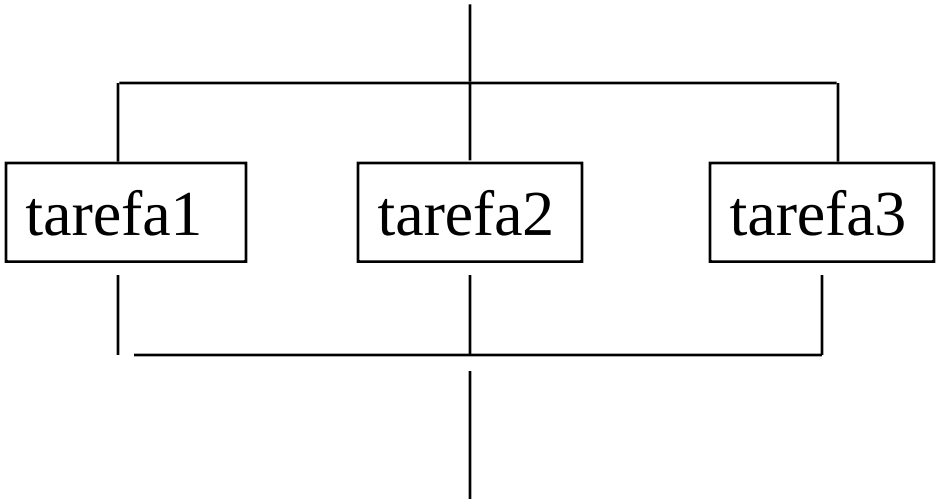
2. criação e ativação de macrotarefa:

```
create n,nome_macrotarefa(argumentos)
```

total de processadores para executar a macrotarefa.

argumentos: parâmetros passados para a macrotarefa.

MACROTASKING



MACROTASKING

```
#include <stdio
task spec tarefa1();
task spec tarefa2();
task spec tarefa3();
task body tarefa1()
{ int i;
   for (i=0;i<199;i++)
      { printf("%c",'a');
        fflush(stdout);
      }
}
```

```
task body tarefa2()
{ int i;
   for (i=0;i<199;i++)
      { printf("%c",'b');
        fflush(stdout);}
}
task body tarefa3()
{ int i;
   for (i=0;i<199;i++)
      {printf("%c",'c');
        fflush(stdout);}
}
```

MACROTASKING

```
main()
{ printf(“INICIO TESTE\n”);
   alloc_proc(4);
   create 1,tarefa1();
   create  1,tarefa2();
   create  1,tarefa3();
 }
```

Programas em CPAR

- Sistema CPAR
- Compilador e biblioteca
- versão atual: utiliza processos
- outros sistemas: processos ou threads
- mostrar exemplo.cpar e exemplo.c

sincronização (macrotasking)

- espera pelo término de uma tarefa

  wait_proc(nome_tarefa);

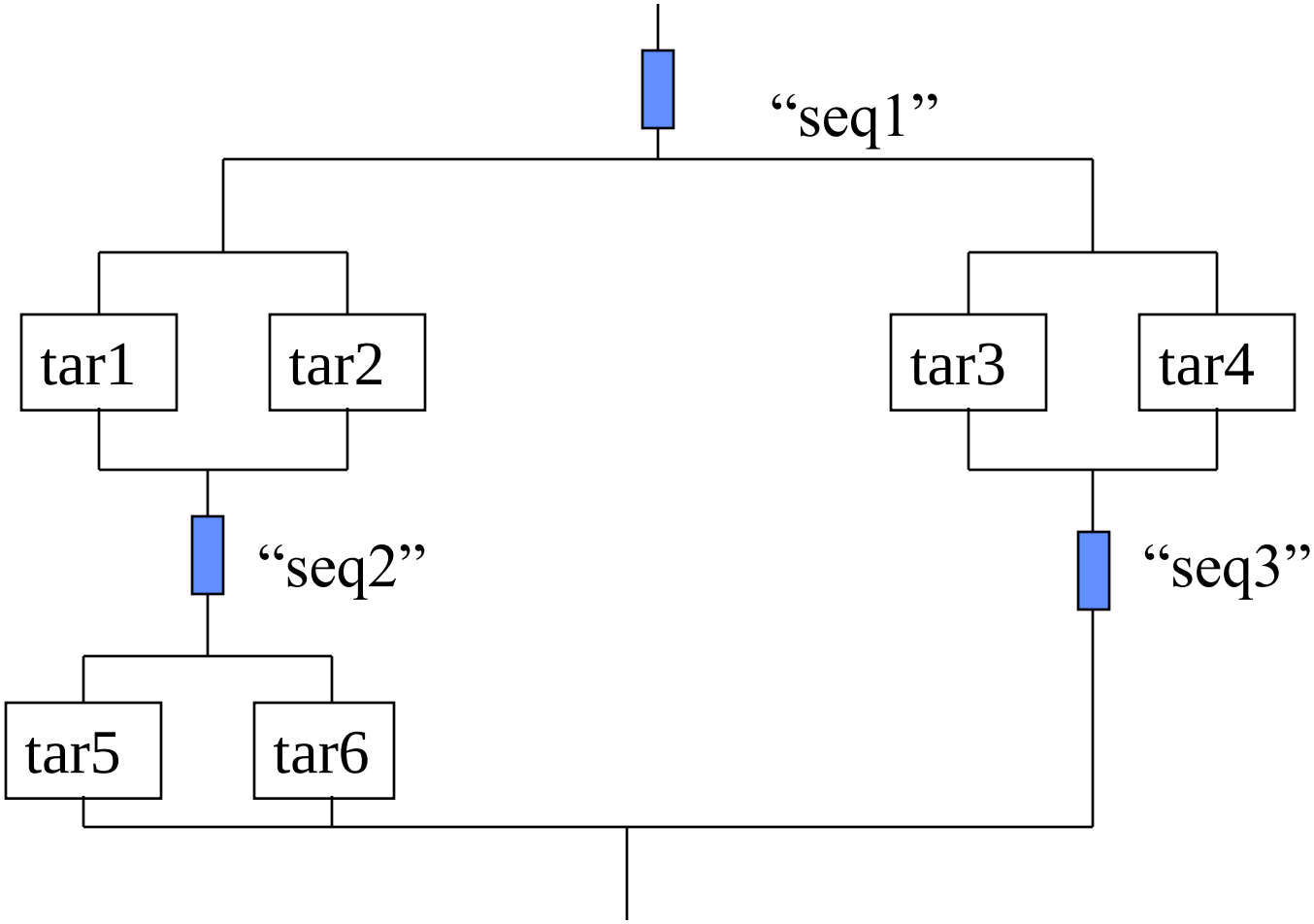
  exemplo: wait_proc(tarefa1);

- espera pelo término de todas tarefas em execução

  wait_all()

exemplo: teste2.cpar

BLOCOS PARALELOS



BLOCOS PARALELOS

- comando cobegin ... also ... coend
- restrição: presente apenas na função main()

```
main()
  {  ...
     cobegin
        .....
        also
        ......
        also
        .......
     coend
      ......
   }
```

exemplo: teste3.cpar

blocos paralelos: expansão

- uso de processos: cada bloco é um processo
- exemplo: teste3.cpar e teste3.c

exercício

- Implementar o programa ilustrado na figura de blocos paralelos

memória compartilhada

O sistema CPAR permite o uso de:

· variáveis privadas (locais)

· variáveis compartilhadas: que podem ser:

- compartilhadas entre macrotarefas (globais)
- compartilhadas entre microtarefas de uma macrotarefa

variáveis compartilhadas

variaveis privadas: declaração de variáveis privadas ou locais é oferecida pela linguagem C.

variáveis compartilhadas: estas devem ser declaradas explicitamente como sendo compartilhada

CPAR: é utilizada a palavra chave shared.

shared declaracao_variavel;

Ex.: shared char A;

variáveis compartilhadas globais

declaradas no processo principal e acessíveis ao processo principal e a todas as macrotarefas

```
shared char A;
task spec tarefa1();
task spec tarefa2();
task body tarefa1()
{char carac;
___
___
carac = A;
___
___

}
```

```
task body tarefa2()
{
char carac;
___
___
carac = A;
___
___
}
main()
{
___
 A='b';
___
}
```

variáveis compartilhadas entre microtarefas

São declaradas na macrotarefa e são acessíveis somente às microtarefas daquela macrotarefa.

```
task spec tarefa1();
task body tarefa1()
{
shared char A[100]; /*valida somente na tarefa1*/
char carac;
___
___

}
```

exercício

- Sejam as matrizes 100x100 (double) A,B,C ,D
- iniciar A,B,C,D : A[i][j]=i+j ; B[i][j]=i+2*j
  C[i][j]=2*i+3*j; D[i][j]=2*i+j
- RESULTADO=A*B + C*D
- multiplicação entre matrizes (sequencial)

```
for (i=0;i<99;i++)
   for (j=0;j<99;j++)
        for (k=0;k<99;k++)
             X[i][j]=X[i][j]+A[i][k]*B[k][j];
```

MACROTASKING e MICROTASKING

No sistema CPAR, que combina “macrotasking”com “microtasking” as threads responsáveis pela execução das microtarefas de uma macrotarefa são criadas na criação da macrotarefa

macrotarefa:

* paralelismo homogêneo
* paralelismo heterogêneo

Início da macrotarefa

· as threads são criadas e colocadas em execução.

Término da macrotarefa

·o processo principal aguarda o término de todas threads para dar como finalizada a macrotarefa.

Paralelismo homogêneo

microtarefas executam mesmo código (loop)

```
forall i=1 to max{
  a[i]=b[i]+1;
  c[i]=c[i]+a[i];
}
```

Paralelismo heterogêneo

microtarefas executam códigos distintos

```
parbegin
  a=a+sqr(b);
  x=x*2;
also
 c=c+sqr(c);
 y=y+1;
parend
```

paralelismo heterogêneo

```
parbegin
  lista_comandos 1
also
  lista_comandos 2
also
  lista_comandos 3
parend
```

```
forall  i$=1 to 3
{if  (i$==1)
     { lista_comandos 1 }
   else
      if (i$==2)
           { lista_comandos 2 }
        else
           { lista_comandos 3 };
   }
```

Laço Paralelo: exemplo

iniciação de uma matriz

```
#include <stdio.h>
shared float a[1000][1000];
task spec inic_mat();
task body inic_mat()
{int i,j;
  forall i=0 to 999
    for(j=0;j<1000;j++)
        a[i][j]=i+2*j;
 }
```

```
main()
{ alloc_proc(4);
   printf(“inicia matriz\n”):
   create   4,inic_mat();
   wait_proc(inic_mat);
   printf(“iniciacao efetuada\n”);
 }
```

Laço Paralelo

- CPAR: iterações são particionadas uniformemente (pré-scheduling)
- pré-compilador: gera código para tratar o particionamento
- em outros sistemas: self-scheduling (distribuiçao dinâmica de blocos de tamanho definido), guided self-scheduling (similar ao self-scheduling, com calculo dinâmico do tamanho dos blocos)
- exemplo: teste_forall.cpar e teste_forall.c
- exemplo: multiplicação de matrizes

Exercício

- iniciar uma matriz a[1000][1000]
- efetuar a busca de um dado elemento (entrada do elemento pelo teclado)
- utilizar 4 processadores