Exemplos de programação paralela

1

Tarefas

- Vimos o modelo de programação multitarefa
- A interface Posix para gerir tarefas
- Vamos ver alguns exemplos de programação

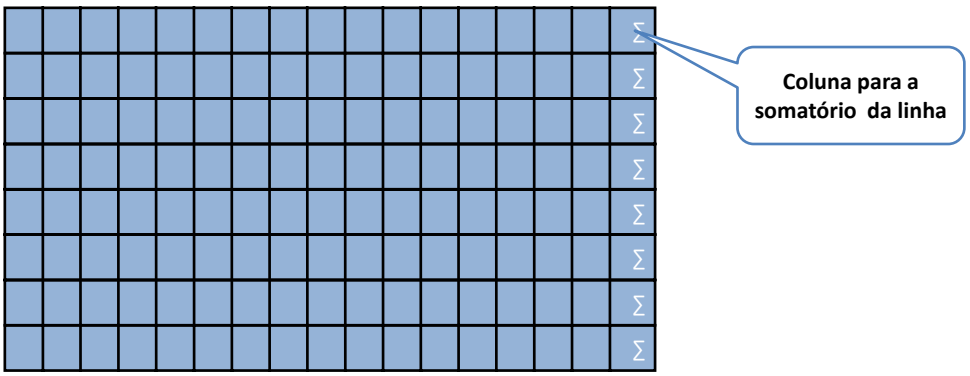
2

Interesse didáctico do ambiente multitarefa

- Podemos experimentar algoritmos paralelos num ambiente contido de um processo
- Muito mais fácil criar algoritmos paralelos e verificar as suas consequências do que utilizando directamente processos

4

Exemplo: somar linhas de matriz



5

Solução sequencial

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#define N 5
#define TAMANHO 10

int buffer [N] [TAMANHO];

void *soma_linha (int *linha) {
  int c, soma=0;
  int *b = linha;
  for (c = 0; c < TAMANHO - 1; c++) {
    soma += b[c];
}

  b[c]=soma; /* soma->ult.col.*/
  return NULL;
}
```

```
int main (void) {
  int i,j;

  inicializaMatriz(buffer(N, TAMANHO);

  for (i=0; i< N; i++)
   soma_linha(buffer[i]);


  imprimeResultados(buffer);

  exit(0);
}
```

6

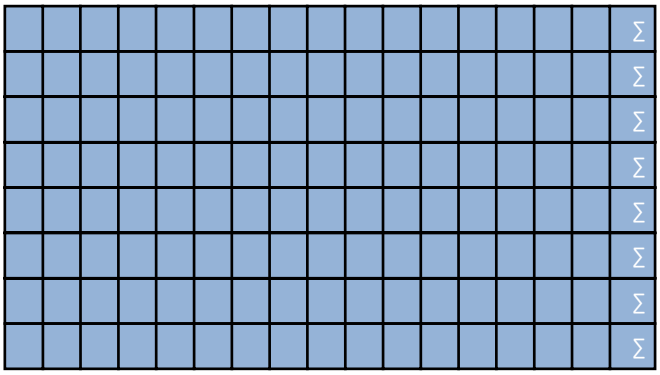
Solução sequencial

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#define N 5
#define TAMANHO 10

int buffer [N] [TAMANHO];

void *soma_linha (int *linha) {
  int c, soma=0;
  int *b = linha;
  for (c = 0; c < TAMANHO - 1; c++) {
    soma += b[c];
}

  b[c]=soma; /* soma->ult.col.*/
  return NULL;
}
```

```
int main (void) {
  int i,j;

  inicializaMatriz(buffer(N, TAMANHO);

  for (i=0; i< N; i++)
   soma_linha(buffer[i]);


  imprimeResultados(buffer);

  exit(0);
}
```

7

Execução sequencial



8

Solução com paralelismo

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#define N 5
#define TAMANHO 10

int buffer [N] [TAMANHO];

void *soma_linha (int *linha) {
  int c, soma=0;
  int *b = linha;
  for (c = 0; c < TAMANHO - 1; c++) {
    soma += b[c];
  }

  b[c]=soma; /* soma->ult.col.*/
  return NULL;
}
```

9

Solução com paralelismo

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#define N 5
#define TAMANHO 10

int buffer [N] [TAMANHO];

void *soma_linha (void *linha) {
  int c, soma=0;
  int *b = (int*) linha;

  for (c=0;c<TAMANHO-1;c++) {
    soma += b[c];
  }

  b[c]=soma; /* soma->ult.col.*/
  return NULL;
}
```

```
int main (void) {
  int i,j;
  pthread_t tid[N];

  inicializaMatriz(buffer, N, TAMANHO);

  for (i=0; i< N; i++){
    if(pthread_create (&tid[i], 0, soma_linha,
                       buffer[i])== 0) {
      printf ("Criada a tarefa %d\n", tid[i]);
    }
    else {
      printf("Erro na criação da tarefa\n");
      exit(1);
    }
  }

  for (i=0; i<N; i++){
    pthread_join (tid[i], NULL);
  }
  printf ("Terminaram todas as threads\n");

  imprimeResultados(buffer);

  exit(0);
}
```

10

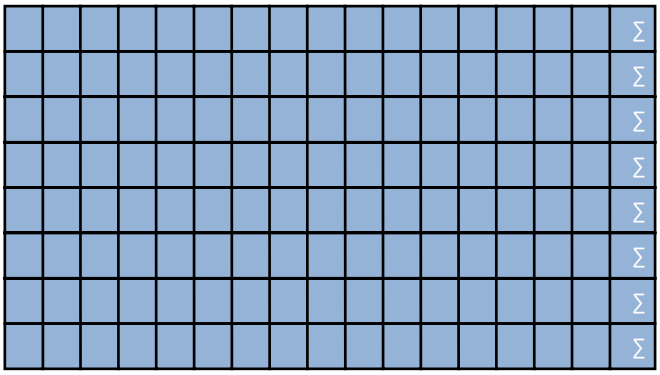
Solução com paralelismo

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#define N 5
#define TAMANHO 10

int buffer [N] [TAMANHO];

void *soma_linha (void *linha) {
  int c, soma=0;
  int *b = (int*) linha;

  for (c=0;c<TAMANHO-1;c++) {
    soma += b[c];
  }

  b[c]=soma; /* soma->ult.col.*/
  return NULL;
}
```

```
int main (void) {
  int i,j;
  pthread_t tid[N];

  inicializaMatriz(buffer, N, TAMANHO);

  for (i=0; i< N; i++){
    if(pthread_create (&tid[i], 0, soma_linha,
                       buffer[i])== 0) {
      printf ("Criada a tarefa %d\n", tid[i]);
    }
    else {
      printf("Erro na criação da tarefa\n");
      exit(1);
    }
  }

  for (i=0; i<N; i++){
    pthread_join (tid[i], NULL);
  }
  printf ("Terminaram todas as threads\n");

  imprimeResultados(buffer);

  exit(0);
}
```

12

Execução em N+1 tarefas paralelas



Algoritmo interessante se tivermos paralelismo real se existir apenas um CPU pode até ser mais lento

13

Programa paralelo que incrementa uma variável global

- Duas tarefas em paralelo incrementam uma variável global (`glob`)
- O programa tem como parâmetro de entrada o número de vezes que a tarefa deve executar o ciclo de incrementação da variável
- O resultado do programa deveria ser
  - ciclo = *x*
  - `glob` = 2*x*

14

Programa paralelo que incrementa uma variável global

```
static volatile int glob = 0

static void * threadFunc(void *arg)
{
    int loops = *((int *) arg);
    int loc, j;

    for (j = 0; j < loops; j++) {
        loc = glob;
        loc++;
        glob = loc;
    }

    return NULL;
}

}
```

15

Programa paralelo que incrementa uma variável global

```
int
main(int argc, char *argv[])
{
    pthread_t t1, t2;
    int loops, s;

    loops = (argc > 1) ? getInt(argv[1], GN_GT_0, "num-loops") : 10000000;

    s = pthread_create(&t1, NULL, threadFunc, &loops);
    if (s != 0) errExitEN(s, "pthread_create");

    s = pthread_create(&t2, NULL, threadFunc, &loops);
    if (s != 0) errExitEN(s, "pthread_create");
```

16

Programa paralelo que incrementa uma variável global

```
    s = pthread_join(t1, NULL);
    if (s != 0) errExitEN(s, "pthread_join");

    s = pthread_join(t2, NULL);
    if (s != 0)errExitEN(s, "pthread_join");

    printf("glob = %d\n", glob);
    exit(EXIT_SUCCESS);
}
```

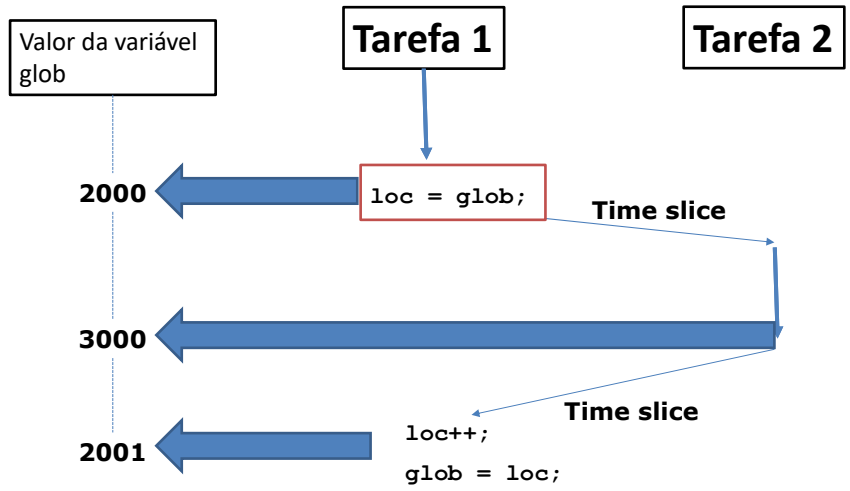
17

Razão do problema

- As tarefas estão a executar-se em paralelo, portanto o despacho está a funcionar comutando-as ao final de um *time slice* uma vez que não se bloqueiam em outras operações porque estão continuamente a incrementar a variável
- Se o ciclo for pequeno, provavelmente, a tarefa termina antes de expirar o *time slice*, mas se for longo tal pode suceder e então podemos ter um comportamento errado do programa

18

Explicação temporal do erro



19

- Em concorrência a comutação das tarefas ou processos ocorre de forma não deterministica
- O erro ocorre porque uma variável partilhada entre duas tarefas não pode ser acedida sem garantir que o seu estado é consistente

20

Conclusões

- São fluxos de execução independentes
- Partilham o mesmo espaço de endereçamento pelo que a execução de acessos a memória é rápida
- São mais leves que os processos pelo que a sua criação é muito mais rápida
- Permitem explorar o paralelismo de múltiplos algoritmos
- Tem de existir formas de evitar incoerências no acesso a variáveis partilhadas devido à concorrência

21