

Programação Concorrente e Paralela

Comunicação por Troca de Mensagens

Noemi Rodriguez

2016



- forma básica de comunicação em ambientes de memória distribuída
 - outras camadas podem ser construídas sobre trocas de mensagens básicas
- **mas também utilizada em ambientes de memória compartilhada**
 - grande guerra filosófica...

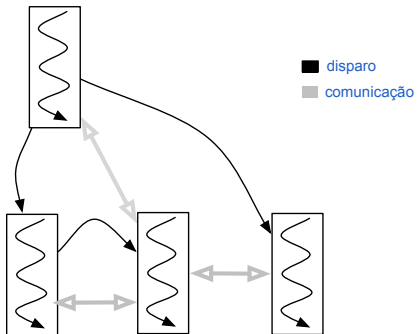
TM em concorrência e paralelismo

- troca de mensagens em “aplicações fechadas”, entre fluxos de controle fortemente acoplados

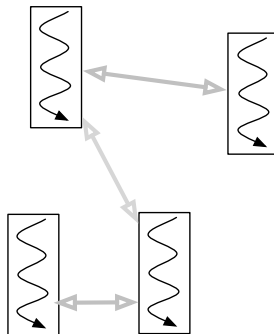


Troca de Mensagens

aplicações fechadas: comunicação entre fluxos com "antepassados" em comum



aplicações abertas: comunicação entre fluxos arbitrários



- identificação de pares
- semântica de envio e recebimento
- formatos e tipos de dados
- operações coletivas?



Identificação de Pares

- problema grande em sistemas abertos/distribuídos
- em sistemas fechados (aplicações): atribuição de identificadores
 - tipicamente inteiros de 0 a $n - 1$

```
myrank = getrank();  
if (myRank == 0)  
    send (1, ...);  
else if (myrank ==1) {  
    receive(0,...);  
    printf ("recebi msg de 0!\n");  
}
```



canais

- *canais* ou *mailboxes* podem ajudar a organizar a comunicação
 - envio para “qualquer um de um grupo”
 - trocas de mensagens de diferentes tipos

```
send ("trabalhadores", ...);  
...  
receive("trabalhadores",...);
```



```
send (destino, blablabla);  
/* o que podemos afirmar nesse ponto de execução?!? */
```

alternativas

- envio síncrono (ou bloqueante): controle retorna depois que mensagem é entregue ao destino
- envio assíncrono (ou não bloqueante): controle retorna imediatamente
- recebimento síncrono (ou bloqueante): controle retorna apenas quando mensagem solicitada já foi recebida
- recebimento assíncrono: ?!? diferentes interpretações

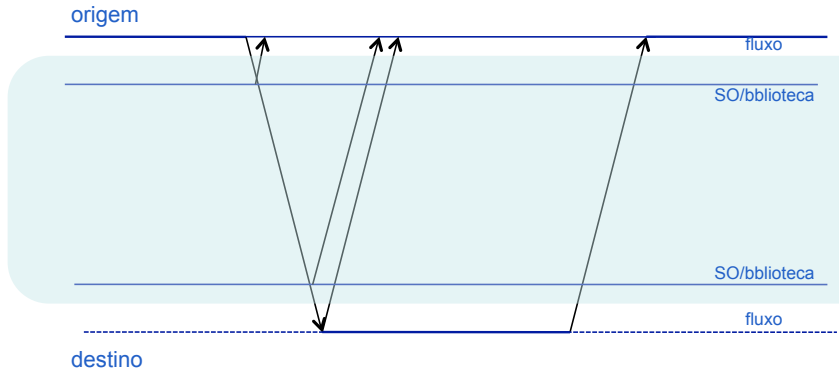


relação com bufferização, determinismo e oportunidades de concorrência, ...

- envio síncrono (ou bloqueante): controle retorna depois que mensagem é entregue ao destino
- envio assíncrono (ou não bloqueante): controle retorna imediatamente
- recebimento síncrono (ou bloqueante): controle retorna apenas quando mensagem solicitada já foi recebida
- recebimento assíncrono: ?!? diferentes interpretações



envio e recebimento: alternativas



- valores tipados
 - que tipos têm suporte?
- ... sequências de bytes
 - interfaces de sistemas operacionais

problemas de representação

- arquiteturas heterogêneas podem demandar conversões entre formatos



- modelo SPMD frequentemente requer operações que envolvem grupos de participantes
 - já vimos operações de *redução* no openMP

operações coletivas comuns

- redução
- barreira
- distribuição de dados
 - broadcast
 - scatter
 - gather



Grupos

- operações coletivas ocorrem dentro de um grupo de fluxos
- outra visão de *comunicação em grupo*
 - conceito mais ligado a sistemas distribuídos

- como definir grupo?
- ordenação de mensagens



MPI

- exemplos de *An Introduction to Parallel Programming*, de Peter Pacheco.



Exemplo: Aproximação por Trapézios



Exemplo: Aproximação por Trapézios

```
int main(void) {
    int n = 1024;
    double a = 0.0, b = 3.0, h, local_a, local_b;
    ...
    MPI_Init(NULL, NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
    h = (b-a)/n; local_n = n/comm_sz;
    local_a = a + my_rank*local_n*h; local_b = local_a + local_n*h;
    local_int = Trap(local_a, local_b, local_n, h);
    if (my_rank != 0) {
        MPI_Send(&local_int, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
    } else {
        total_int = local_int;
        for (source = 1; source < comm_sz; source++) {
            MPI_Recv(&local_int, 1, MPI_DOUBLE, source, 0,
                MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            total_int += local_int;
        }
    }
}
if (my_rank==0) /* mostra resultados */
```



Exemplo: Aproximação por Trapézios (segunda versão)

com operações coletivas!

```
MPI_Init(NULL, NULL);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
Get_input(my_rank, comm_sz, &a, &b, &n);
h = (b-a)/n;          /* h is the same for all processes */
local_n = n/comm_sz; /* So is the number of trapezoids */
local_a = a + my_rank*local_n*h;
local_b = local_a + local_n*h;
local_int = Trap(local_a, local_b, local_n, h);
MPI_Reduce(&local_int, &total_int, 1, MPI_DOUBLE, MPI_SUM, 0,
          MPI_COMM_WORLD);
```



Exemplo: Aproximação por Trapézios (segunda versão)

```
void Get_input(int my_rank, int comm_sz, double* a_p, double* b_p,
               int* n_p) {

    if (my_rank == 0) {
        printf("Enter a, b, and n\n");
        scanf("%lf %lf %d", a_p, b_p, n_p);
    }
    MPI_Bcast(a_p, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(b_p, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(n_p, 1, MPI_INT, 0, MPI_COMM_WORLD);
}
```



- barreiras implícitas
- grupos implantados por *comunicadores*



Visão estática do programa

- tipicamente um ou dois processos por “nó”
 - como tratar máquinas com vários núcleos?
- dificuldades com ambientes dinâmicos



Linguagem D

- proposta para o “nicho” C/C++
- ... com mais garantias na compilação
- compartilhamento de memória com restrições



exemplo: produtor/consumidor em D

```
import std.algorithm, std.concurrency, std.stdio;

void main() {
    enum bufferSize = 1024 * 100;
    auto tid = spawn(&fileWriter);
    // Read loop
    foreach (immutable(ubyte)[] buffer; stdin.byChunk(bufferSize)) {
        send(tid, buffer);
    }
}

void fileWriter() {
    // Write loop
    for (;;) {
        auto buffer = receiveOnly!(immutable(ubyte)[])( );
        tgt.write(buffer);
    }
}
```

