Computação Paralela e Distribuída

Ano lectivo 2023-24

Rascunho - Será reorganizado e actualizado

Tema #4: Programação em Sistemas de Memória Distribuída

Título: Introdução ao MPI

João José da Costa
joao.costa@isptec.co.ao

Coordenação de Engenharia Informática
Departamento de Engenharias e Tecnologias
Instituto Superior Politécnico de Tecnologias e Ciências

MPI

Message Passing Interface

Tópicos

- Contexto
- Conceitos básicos
- Comunicadores
- Mensagens
  - Ponto-a-ponto
  - Muito-para-muito
- Exemplo de aplicação

Passagem de Mensagem

Fundamentos para o MPI

- Passagem de mensagem: modelo de programação para computadores paralelo de memória distribuída
  - Todos os processadores executam um processo independente
  - Espaço de endereçamento disjuntos, sem partilha de dados pela memória
  - Toda a comunicação entre processos é feita cooperativamente, através de chamada de subrotinas.

- Este modelo teve sucesso pelas seguintes razões:
  - Mapeia para uma grande variedade de hardware
  - O paralelismo e a comunicação são explícitas
    - Força o programador a modelar o paralelização logo no princípio
  - MPI torna o programa portável

O que é o MPI?

Message Passing Interface (MPI)

- MPI: é um padrão para programação científica em computadores paralelos de memória distribuída
  - Biblioteca de rotinas que permitem aplicações de passagem de mensagem
  - Especificação de interface, não uma implementação concreta

- Antes do MPI:
  - Diferentes bibliotecas para cada tipo de computador
    - CMMD (Thinking Machines CM5)
    - NX (Intel iPSC/860, Paragon)
    - MPL (SP2) e muito mais...
  - PVM (Parallel Virtual Machine): tenta se tornar um padrão, mas sem alto-desempenho e sem boa especificação.

História do MPI

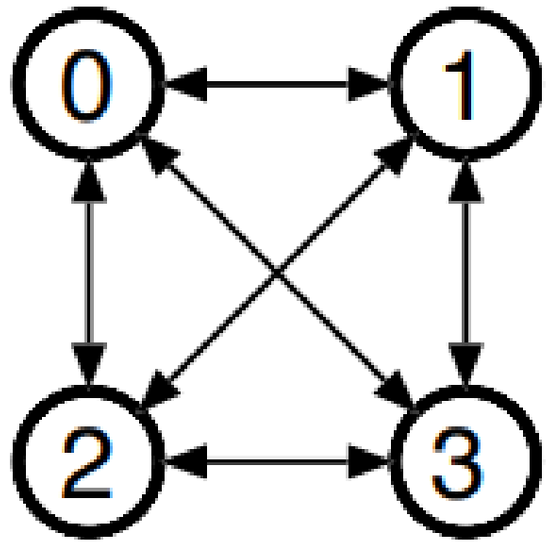
Fundamentos para o MPI

- O MPI foi desenvolvido por um forum MPI, uma organização voluntária em representação de indústria e labs/academias governamentais
  - MPI – 1 (1994): codificado com existência prática
    - "Quem sou?", "Quanto processos existem?"
    - Comunicação de send/recv
    - Comunicação colectiva, p.e. Broadcast, reduction, all-to-all
    - Muitos outros aspectos
  - MPI – 2 (1997)
    - I/O paralelo
    - C++/Fortran 90
    - Comunicação: get/put
    - ...
  - MPI – 3 (2012)
    - Criação de processo dinâmico
    - Tolerância a falha
    - Extensão de rotina existente

Aplicação MPI

Exemplo “Hello, world” : Fundamentos

- Os elementos da aplicação são:
  - 4 tarefas, numeradas de 0 a 3
  - Comunicação entre as mesmas



- O conjunto de tarefas mais o canal de comunicação é chamado de “**`MPI_COMM_WORLD`**”

Aplicação MPI

Exemplo “Hello, world” : Código-fonte

```
#include <mpi.h>
#include <stdio.h>
main(int argc, char *argv[])
{
 int id, p;
 MPI_Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD, &p);
 MPI_Comm_rank(MPI_COMM_WORLD, &id);

 printf("Oi do host %d de %d\n", id, p);
 MPI_Finalize();
}
```

Aplicação MPI

Exemplo “Hello, world” : Compilação e execução

- Diferente em cada máquina, depende da actual implementação do MPI
- OpenMPI é uma implementação de alta qualidade e open-seource do MPI
- Combina com as implementações MPI bem-conhecidas
  - FT-MPI, da Universidade de Tennessee
  - LA-MPI, do Laboratório Nacional de Los Alamos
  - LAM/MPI, da Universidade Indiana
  - PACX-MPI, da Universidade de Stuttgart
- Compilar: **`$mpicc -o hello hello.c`**
- Iniciar quatro processos: **`$mpirun -np 4 ./hello`**

Aplicação MPI

Exemplo “Hello, world” : Saída de amostra

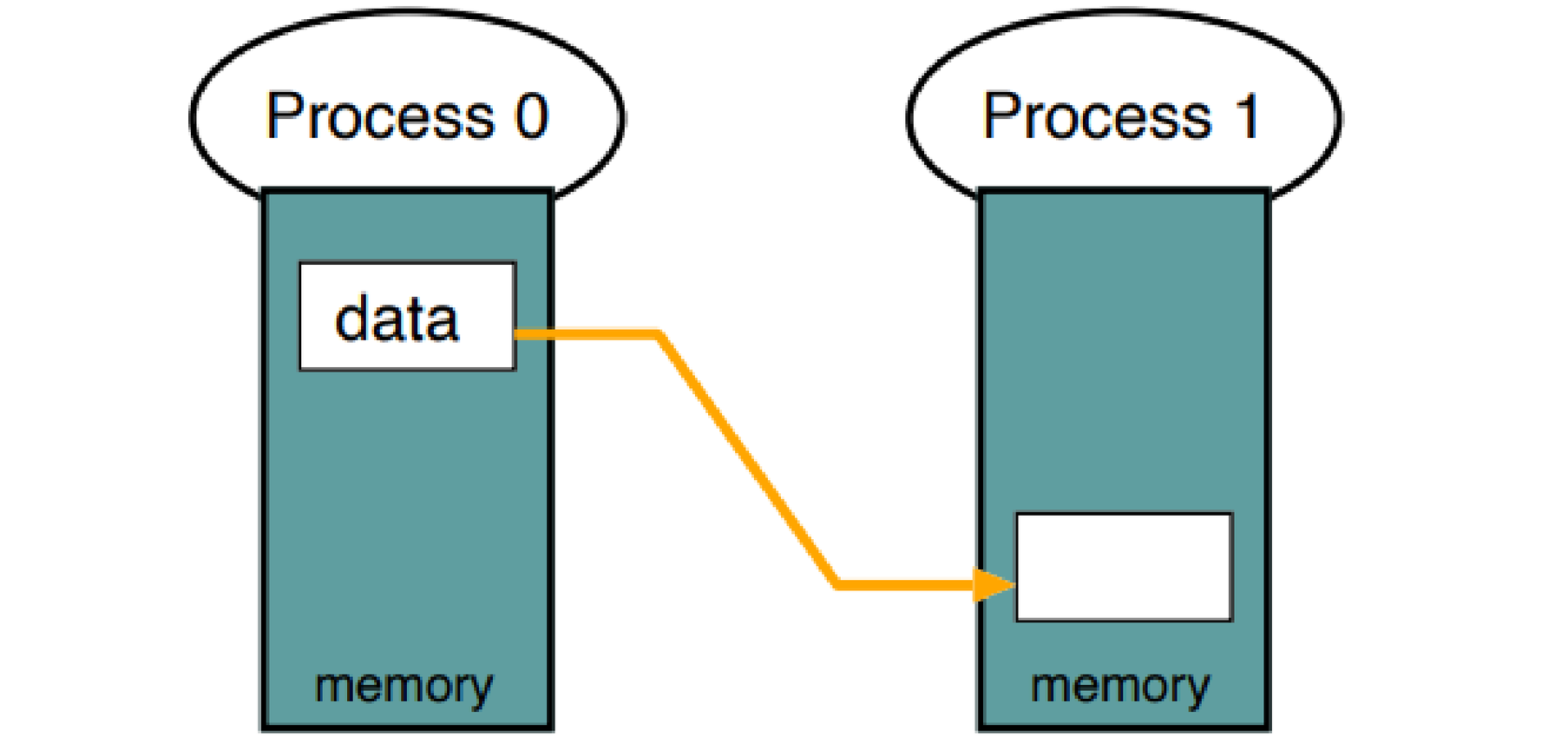
- Executa com 4 processos

```
Oi do host 2 de 4
Oi do host 1 de 4
Oi do host 0 de 4
Oi do host 3 de 4
```

- Note que:
  - A ordem de saída pode alterar de execução para execução
  - Habilidade para utilizar o **`stdout`** não é garantido pelo MPI

Comunicação

Comunicação ponto-a-ponto




```
MPI_Send(data,...)
```

```
MPI_Recv(data,...)
```

Comunicação

Comunicação ponto-a-ponto: send/recv

- A tarefa **0** envia um vector **A** para tarefa **1**, que recebe como **B**:

Tarefa 0

```
#define TAG 252
double A[10];
MPI_Send(A, 10, MPI_DOUBLE, 1, TAG, MPI_COMM_WORLD);
```

Tarefa 1

```
#define TAG 252
double B[10];
MPI_Recv(B, 10, MPI_DOUBLE, 0, TAG, MPI_COMM_WORLD, &estado);
```

OU

```
MPI_Recv(B, 10, MPI_DOUBLE, MPI_ANY_SOURCE, MPI_ANY_TAG,
MPI_COMM_WORLD, &estado);
```

Comunicação

Comunicação ponto-a-ponto: send/recv

- Envio (MPI_Send)
  - Precisa passar o ID de destino
  - ID deve ser válido (0, ..., P - 1) no comunicador
- Recepção (MPI_Recv)
  - Precisa passar o ID de origem (source)
  - A curinga MPI_ANY_SOURCE combina com qualquer source.
- Etiqueta (TAG) de mensagem
  - No lado do emissor, especifica um rótulo para a mensagem
  - No lado do receptor, precisa usar o mesmo rótulo da mensagem a receber
  - No lado do receptor, a curinga MPI_ANY_TAG combina com qualquer rótulo de mensagem.

Comunicação

Comunicação ponto-a-ponto: send/recv

- A menos que haja um bom motivo para fazer isso, não utilize constante curinga!
- No entanto, podem haver boas razões para utilizar as constantes curinga:
  - Receber mensagens de diversas fontes no mesmo buffer (utilize MPI_ANY_SOURCE)
  - Receber várias mensagens da mesma fonte no mesmo buffer e não se importar com a ordem (utilize MPI_ANY_TAG)

Comunicação

Comunicação ponto-a-ponto: send/recv

- Tipos de dado pré-definidos para o C
  - MPI_INT
  - MPI_FLOAT
  - MPI_DOUBLE
  - MPI_CHAR
  - MPI_LONG
  - MPI_UNSIGNED
- Independente de linguagem
  - MPI_BYTE
- MPI permite a definição de outros tipos/estruturas de dados

Comunicação

Comunicação ponto-a-ponto: send/recv

- MPI_Status é uma estrutura
  - status.MPI_TAG é um rótulo da mensagem recebida (útil se MPI_ANY_TAG foi especificado)
  - status.MPI_SOURCE é o identificador do remetente da mensagem recebida (útil se MPI_ANY_SOUCE foi especificado)
  - Quantos elementos de determinado tipo de dado foram recebidos

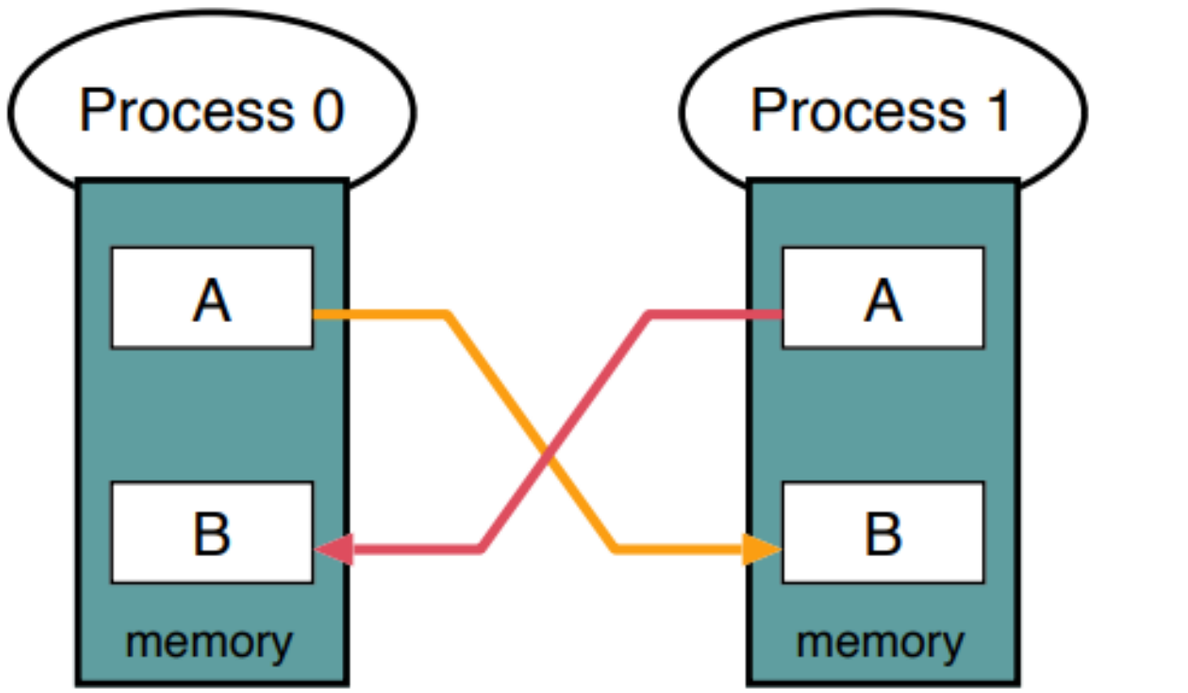
```
MPI_Get_count(MPI_Status *status,
              MPI_Datatype datatype,
                  int *count)
```

Comunicação

Comunicação ponto-a-ponto: send/recv

- Caso de estudo: troca de dado entre tarefas

- **Requer espaço em buffer para evitar deadlock. Porquê???**




```
MPI_Send(A,...)
MPI_Recv(B,...)
```

```
MPI_Send(A,...)
MPI_Recv(B,...)
```

Um programa correcto não deve depender de buffer!!!!!

Comunicação

Comunicação ponto-a-ponto: send/recv

Operações sem-bloqueio

- Sem deadlock;
- Os dados podem ser transferidos concorrentemente.

```
#define TAG 252
#define WORLD MPI_COMM_WORLD
MPI_Request request;
MPI_Status status;
```

Tarefa 0

```
MPI_Irecv(B, 100, MPI_DOUBLE, 1, TAG, WORLD, &request);
MPI_Send(A, 100, MPI_DOUBLE, 1, TAG, WORLD);
MPI_Wait(&request, &status);
```

Tarefa 1

```
MPI_Irecv(B, 100, MPI_DOUBLE, 0, TAG, WORLD, &request);
MPI_Send(A, 100, MPI_DOUBLE, 0, TAG, WORLD);
MPI_Wait(&request, &status);
```

Comunicação

Comunicação ponto-a-ponto: send/recv

Em alguns computadores pode ser possível realizar trabalhos úteis enquanto os dados estão a ser transferidos.

Redução da latência

```
#define TAG 252
#define WORLD MPI_COMM_WORLD
MPI_Request requests[2];
MPI_Status statuses[2];
                    Tarefa
MPI_Irecv(B, 100, MPI_DOUBLE, p, TAG, WORLD, &request[1]);
MPI_Isend(A, 100, MPI_DOUBLE, p, TAG, WORLD, &request[0]);
  .... Realiza aqui algum trabalho útil ....
MPI_Waitall(2, requests, statuses);
```

- **Irecv/Isend** inicia a comunicação
- A comunicação prossegue enquanto o processador está a realizar trabalhos úteis
- Suporte de hardware necessário para sobreposição real

Comunicação

Comunicação ponto-a-ponto: send/recv

Operações em MPI_Request.

- **`MPI_Wait(INOUT request, OUT status)`**
  - Espera completar a operação
  - Retorna a informação (se aplicável) em estado
  - Liberta o objecto do pedido (e define como **`MPI_REQUEST_NULL`**)

- **`MPI_Test(INOUT request, OUT flag, OUT status)`**
  - Testa para determinar se a operação foi concluida
  - Retorna informação de estado se concluída
  - Liberta o objecto do pedido se concluída

- MPI_Cancel(IN request)
  - Cancela e conclui um pedido

- MPI_Request_free(INOUT request)
  - Liberta o objecto de pedido, mas não espera a conclusão da operação.
- **`MPI_Waitall(..., INOUT array of requests, ...)`**
- **`MPI_Testall(..., INOUT array of requests, ...)`**
- **`MPI_Waitany/MPI_Testany/MPI_Waitsome/MPI_Testsome`**

Comunicação

Comunicação ponto-a-ponto: send/recv

Problemas de comunicação sem bloqueio

- Preocupações óbvias:
  - Não pode modificar o buffer entre **`Isend()`** e o **`Wait()`** correspondente – os resultados são indefinidos
  - Não pode visualizar ou modificar o buffer entre **`Irecv()`** e o **`Wait()`** correspondente – os resultados são indefinidos
  - Não pode haver dois **`Irecv()`** pendentes para o mesmo buffer
- Menos óbvio:
  - Pode não olhar para o buffer entre Isend() e o Wait() correspondente
  - Não pode haver dois Isend() pendentes para o mesmo buffer

Comunicação

Comunicadores

- Um comunicador é um objecto que representa:
  - Um conjunto de tarefas
  - Canais de comunicação privadas entre essas tarefas
    - **`MPI_COMM_WORLD`** é um comunicador que inclui todas as tarefas e está disponível na inicialização.
- Comunicadores permitem definir o escopo das operações colectivas.

Comunicação

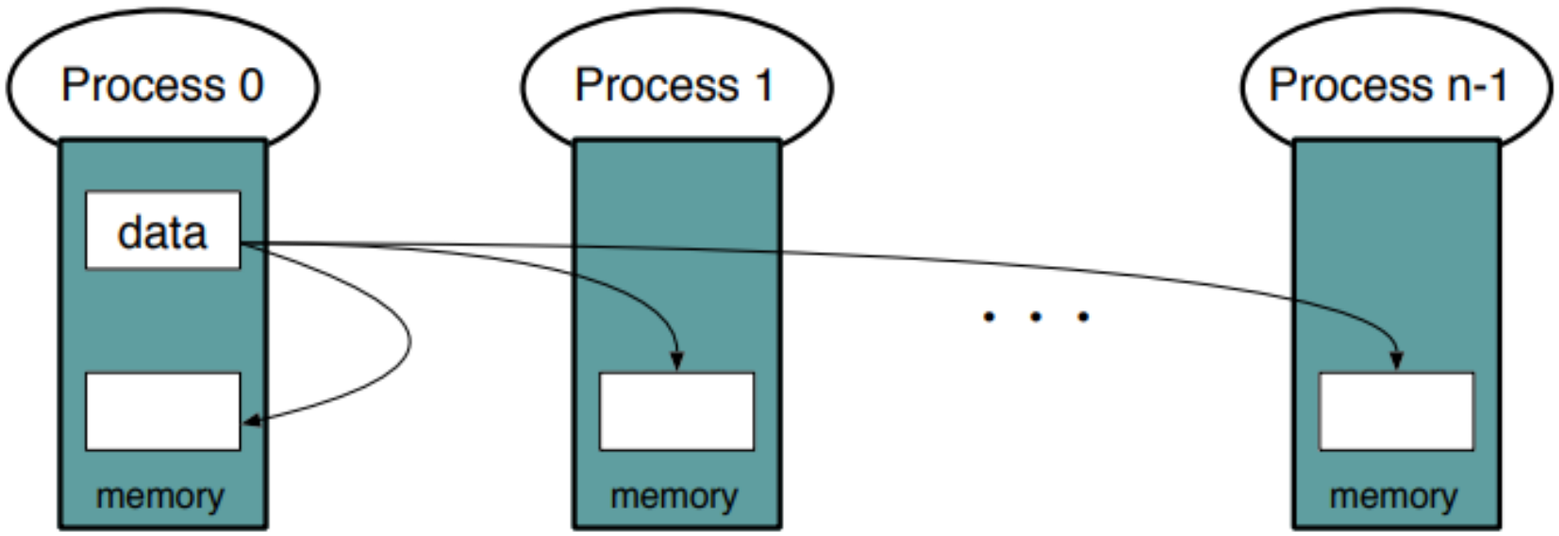
Operações colectivas

- A comunicação colectiva é a comunicação entre um grupo de tarefas:
  - Difusão (broadcast)
  - Dispersão/União (Scatter/gather)
  - Operações globais (redução)
  - Prefixo paralelo (*scan*)
  - Sincronização (barreira)

Comunicação

Operações colectivas – Difusão (broadcast)

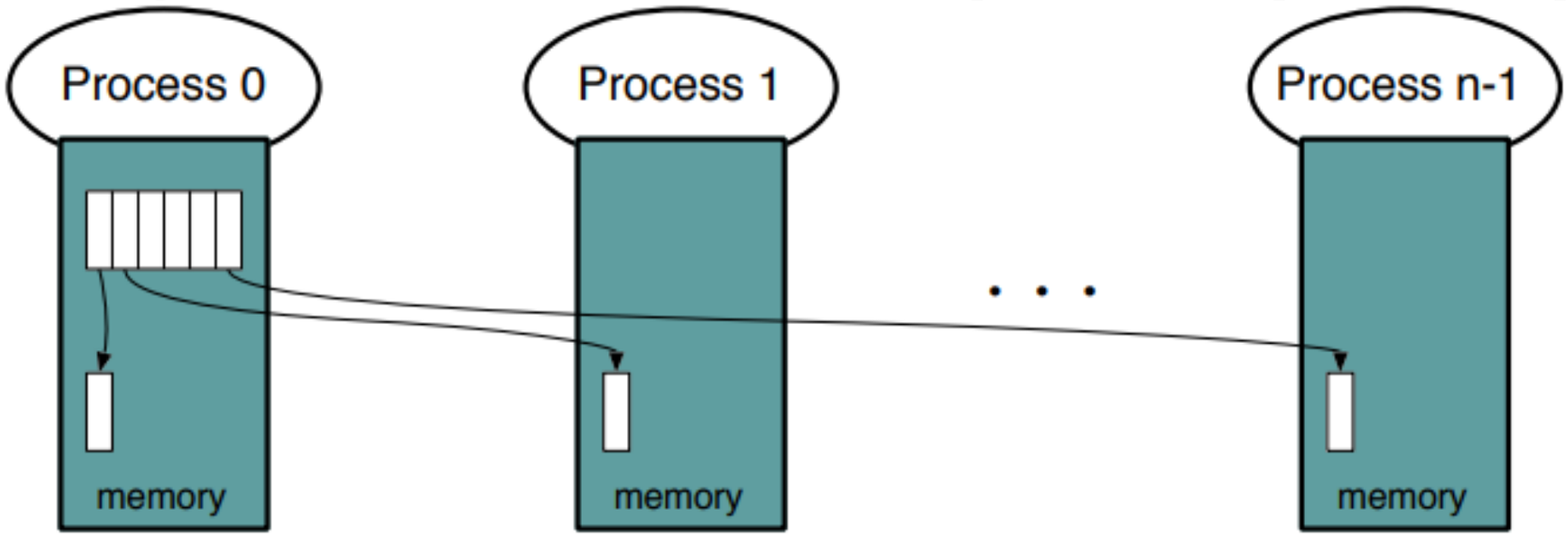
Implementa-a! Qual é a complexidade da comunicação? Se optimizado usando árvore, qual é a complexidade de comunicação?



- **`MPI_Bcast(buf, len, type, root, comm)`**
  - A tarefa com o id igual ao **`root`** é a fonte de dado (em buf)
  - As outras tarefas recebem o dado.

Comunicação

Operações colectivas – Dispersão (scatter)



Propõe uma implementação trivial? Qual é sua complexidade de comunicação?

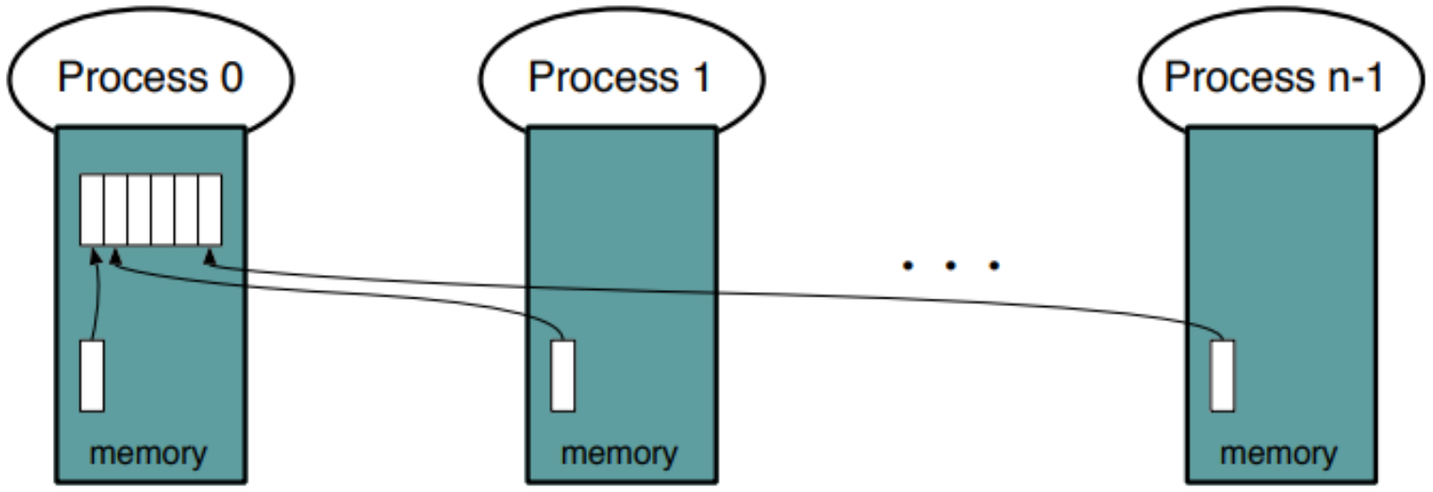
- **`MPI_Scatter()` / `MPI_Scatterv()`**
  - Subpartes de um único grande vector são distribuídas para tarefas

```
int MPI_Scatter(const void* sbuf, int scount, MPI_Datatype
stype, void* rbuff, int rcount, MPI_Datatype rtype, int
root, MPI_Comm comm)

int MPI_Scatterv(const void* sbuf, const int scounts[],
const int displacements[], MPI_Datatype stype, void* rbuf,
int rcount, MPI_Datatype rtype, int root, MPI_Comm comm)
```

Comunicação

Operações colectivas – União (gather)



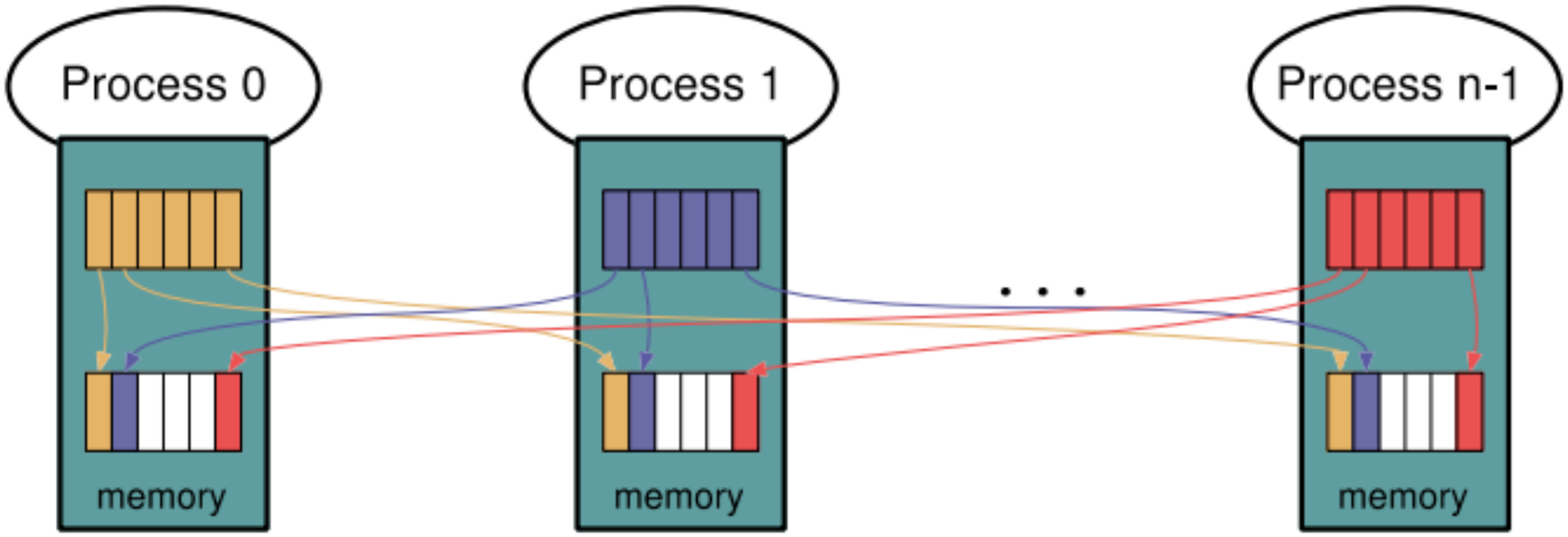
Propõe uma implementação trivial? Qual é sua complexidade de comunicação?

- **`MPI_Gather() / MPI_Gatherv()`**
- **`MPI_Allgather() / MPI_Allgatherv()`**
  - Cada tarefa contribui com dados locais que são reunidos em uma matriz maior.

```
int MPI_Gather(const void* sbuf, int scount, MPI_Datatype
stype, void* rbuff, int rcount, MPI_Datatype rtype, int
root, MPI_Comm comm)
```

Comunicação

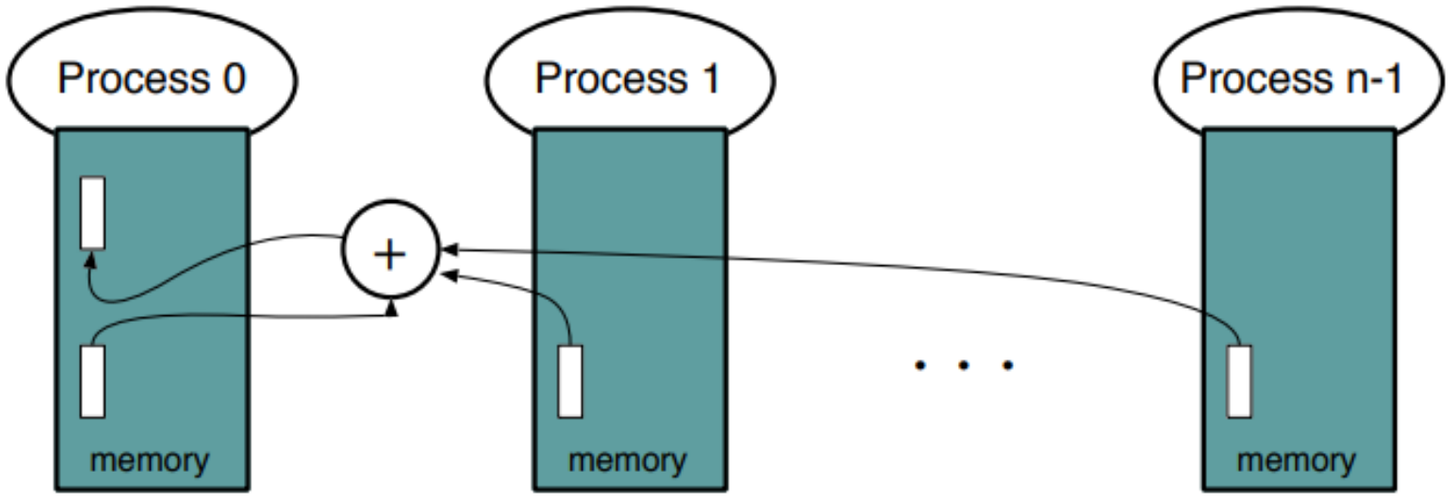
Movimentos de dados de todos para todos



- **`MPI_Alltoall(sbuf, scount, stype, rbuf, rcount, rtype, comm)`**
  - Todas as tarefas enviam e recebem dados de todas as outras tarefas
  - Para um comunicador com N tarefas
    - **`sbuf`** contém N blocos de elementos **`scount`** cada
    - **`rbuf`** recebe N blocos de elementos **`rcount`** cada
    - Cada tarefa envia o bloco i de **`sbuf`** para a tarefa i
    - Cada tarefa recebe o bloco j de **`rbuf`** da tarefa j

Comunicação

Redução



- **`MPI_Reduce(indata, outdata, count, type, op, root, comm)`**
- **`MPI_Allreduce(indata, outdata, count, type, op, comm)`**
  - Combina os elementos no buffer de entrada de cada tarefa, colocando o resultado no buffer de saída.
  - Reduce: a saída aparece apenas no buffer do **`root`**.
  - Allreduce: a saída aparece em todos os processos
  - Alguns tipos de operação: MPI_SUM, MPI_PROD, MPI_MAX, MPI_MIN,...
  - Operações arbitrárias definidas pelo utilizador em tipos de dados arbitrários definidos pelo utilizador são possíveis.

Comunicação

Redução – Exemplo de produto escalar

```
/* distribui dois vectores entre todos os processos, tal que
 processador 0 tem os elementos 0...99
 processador 1 tem os elementos 100...199
 processador 2 tem elementos 200...299 etc. */
double produto(double a[100], double b[100])
{
    double gresultado = lresultado = 0.0;
    int i; /* computa o produto escalar local */
    for (i = 0; i < 100; i++)
       lresultado += a[i] * b[i];
    MPI_Allreduce(&lresultado, &gresultado, 1, MPI_DOUBLE,
                          MPI_SUM, MPI_COMM_WORLD);
    return gresultado;
}
```

Comunicação

Sincronização

- **`MPI_Barrier(comunicador)`**
  - Nenhum processo passa a barreira até que todos os processos tenham entrado na mesma.

Exemplo de Aplicação

#1 – Satisfiabilidade de circuito

Exemplo #1

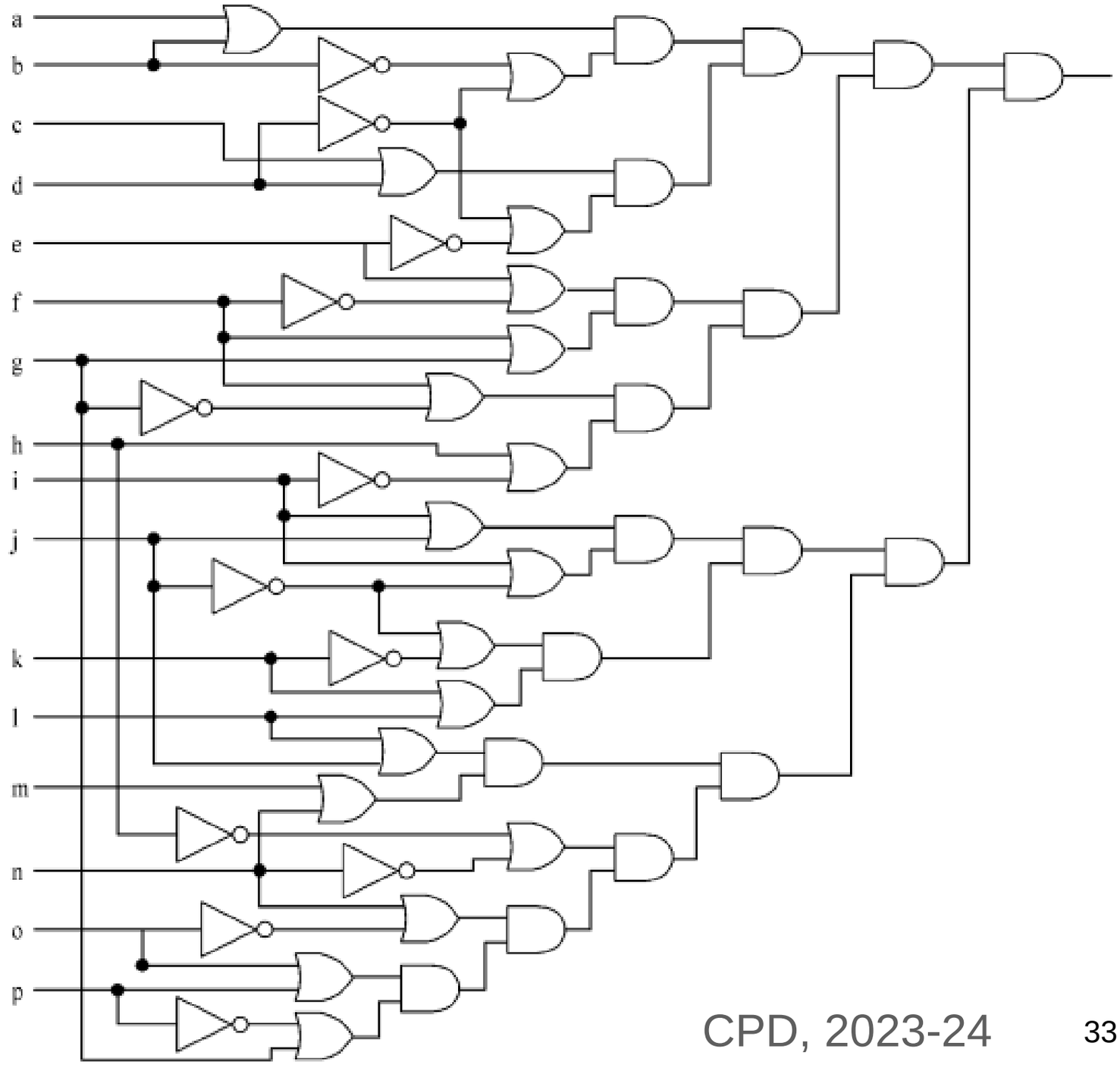
Problema de MPI: Satisfiabilidade de circuito

- **`Satisfiabilidade de circuito (CSAT)`**
  - Determinar uma combinação de valores de entrada que afirmem o nó de saída de um circuito lógico ou prove que tal combinação não é possível.
  - Problema básico em testes de circuitos VLSI
  - Problema NP-Completo

Exemplo #1

Problema de MPI: Satisfiabilidade de circuito

- **Instância CSAT**



Exemplo #1

Problema de MPI: Satisfiabilidade de circuito

Problema CSAT

- Determinar todas as combinações de entrada que afirmam (assert) a saída do circuito
  - Assert = faça com que assuma o valor lógico 1 (verdadeiro)

- Resolva este problema através de uma pesquisa exaustiva
  - Teste todas as combinações de entrada

Exemplo #1

Problema de MPI: Satisfiabilidade de circuito

Metodologia de Foster para o CSAT

- Particionamento
  - Faça com que cada combinação de entrada teste uma tarefa primitiva

- Comunicação
  - Sem canais – sem interações entre as tarefas

- Agregação e mapeamento
  - Distribuir as tarefas uniformemente entre CPUs disponíveis.

Exemplo #1

Problema de MPI: Satisfiabilidade de circuito

Implementação do CSAT em C com MPI

```
#include <mpi.h>
#include <stdio.h>
int main (int argc, char *argv[])
{
 int i, id, p;
 MPI_Init (&argc, &argv);
 MPI_Comm_rank (MPI_COMM_WORLD, &id);
 MPI_Comm_size (MPI_COMM_WORLD, &p);
 for (i = id; i < 65536; i += p)
    verifica_circuito (id, i);
 printf ("Processo %d concluido.\n", id);
 fflush (stdout);
 MPI_Finalize ();
 return 0;
}
```

Exemplo #1

Problema de MPI: Satisfiabilidade de circuito

Implementação do CSAT em C com MPI

```
/* Returna 1 se 'i'° bit de 'n' é 1; 0 caso contrário */
#define EXTRAI_BIT(n,i) ((n&(1<<i))?1:0)
void verifica_circuito (int id, int z) {
 int i, v[16]; /* cada elemento é um bit de z */
 for (i = 0; i < 16; i++) v[i] = EXTRAI_BIT(z,i);

  if (    (v[0] || v[1]) && (!v[1] || !v[3]) && (v[2] || v[3])
          && (!v[3] || !v[4]) && (v[4] || !v[5]) && (v[5] || !v[6])
          && (v[5] || v[6]) && (v[6] || !v[15]) && (v[7] || !v[8])
          && (!v[7] || !v[13]) && (v[8] || v[9]) && (v[8] || !v[9])
          && (!v[9] || !v[10]) && (v[9] || v[11]) && (v[10] || v[11])
          && (v[12] || v[13]) && (v[13] || !v[14]) && (v[14] || v[15]))
 {
      printf ("%d) %d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d\n", id,
          v[0],v[1],v[2],v[3],v[4],v[5],v[6],v[7],v[8],v[9],
          v[10],v[11],v[12],v[13],v[14],v[15]);
      fflush (stdout);
  }
}
```

Exemplo #1

Problema de MPI: Satisfiabilidade de circuito

Compilação e execução do CSAT

- Comando para compilar

```
$ mpicc <flags> -o <executável> <código-fonte>.c
```

  - Os mesmos flags como gcc, e mais
  - Links com a biblioteca MPI

- No nosso caso:

```
$ mpicc -O -o csat1 csat1.c
```

  - Consulte a documentação para conhecer mais flags,...

- Execução

```
$ mpirun -np <p> <executável> <args>
```

  - <args>: argumentos de linha de comando do programa.

Exemplo #1

Problema de MPI: Satisfiabilidade de circuito

Amostra de saída do CSAT

```
$ mpirun -np 1 csat1
0) 101011111O011001
0) 011011111O011001
0) 111011111O011001
0) 101011111O011001
0) 011011111O011001
0) 111011111O011001
0) 101011111O111001
0) 011011111O111001
0) 111011111O111001
Processo 0 concluido.
```

```
$ mpirun -np 2 csat1
0) 011011111O011001
0) 011011111O011001
0) 011011111O111001
1) 101011111O011001
1) 111011111O011001
1) 101011111O011001
1) 111011111O011001
1) 101011111O111001
1) 111011111O111001
Processo 0 concluido.
Processo 1 concluido.
```

```
$ mpirun -np 3 csat1
0) 011011111O011001
0) 111011111O011001
2) 101011111O011001
1) 111011111O011001
1) 101011111O011001
1) 011011111O111001
0) 101011111O111001
2) 011011111O011001
2) 111011111O111001
Processo 1 concluido.
Processo 2 concluido.
Processo 0 concluido.
```

Exemplo #1

Problema de MPI: Satisfiabilidade de circuito

Decifrando as saídas do CSAT

- A ordem de saída reflete apenas parcialmente a ordem dos eventos de saída dentro do computador paralelo
  - Se o processo A imprimir duas mensagens, a primeira mensagem aparecerá antes da segunda
  - Se o processo A chamar printf antes do processo B, não há garantia de que a mensagem do processo A aparecerá antes da mensagem do processo B.

Exemplo #1

Problema de MPI: Satisfiabilidade de circuito

Comunicação colectiva

- Aprimore o programa: escreva uma nova versão do programa CSAT para que retorne o número total de soluções.
- Modifique a função **`verifica_circuito`**
  - Retorna 1 se o circuito for satisfatório com a combinação de entrada
  - Retorna 0 caso contrário
- Cada tarefa mantém uma contagem local de combinações de entrada satisfatórias que encontrou.
- Incorpora redução de soma no programa.

Exemplo #1

Problema de MPI: Satisfiabilidade de circuito

Novo código e declarações

```
int conta; /* soma local*/
int global_conta; /* soma global*/
int verifica_circuito(int, int);

...

conta = 0;
for (i = id; i < 65536; i += p)
 conta += verifica_circuito(id, i)
```

Exemplo #1

Problema de MPI: Satisfiabilidade de circuito

Protótipo da MPI_Reduce

```
int MPI_Reduce (
 void *operand, /* endereço do 1° elemento de redução */
 void *result, /* endereço do 1° resultado de redução */
 int count, /* tamanho */
 MPI_Datatype type, /* tipo dos elementos */
 MPI_Op operator, /* operador de redução */
 int root, /* processo que recebe o resultado */
 MPI_Comm comm /* comunicador */
)
```

```
MPI_Reduce (&conta, &global_conta, 1, MPI_INT, MPI_SUM,
0, MPI_COMM_WORLD);
```

Exemplo #1

Problema de MPI: Satisfiabilidade de circuito

Nova versão do CSAT

```
#include <mpi.h>
#include <stdio.h>
int main (int argc, char *argv[]){
     int i, id, p;
     int conta, global_conta;
     MPI_Init (&argc, &argv);
     MPI_Comm_rank (MPI_COMM_WORLD, &id);
     MPI_Comm_size (MPI_COMM_WORLD, &p);
         conta = 0;
         for (i = id; i < 65536; i += p)
             conta += verifica_circuito (id, i);
     MPI_Reduce (&conta, &global_conta, 1, MPI_INT, MPI_SUM, 0,
        MPI_COMM_WORLD);
         printf ("Processo %d concluido.\n", id);
         fflush (stdout);
     MPI_Finalize ();
     if (!id) printf("Há %d diferente(s) soluções.\n", global_conta);
     return 0;
    }
```

Exemplo #1

Problema de MPI: Satisfiabilidade de circuito

Amostra de saída do CSAT

```
$ mpirun -np 3 csat2
0) 0110111110011001
0) 1110111111011001
1) 1110111110011001
1) 1010111111011001
2) 1010111110011001
2) 0110111111011001
2) 1110111110111001
1) 0110111110111001
0) 1010111110111001
Processo 1 concluido.
Processo 2 concluido.
Processo 0 concluido.
Há 9 diferente(s) soluções.
```

Exemplo #1

Problema de MPI: Satisfiabilidade de circuito

Código de Benchmarking

- Métrica de interesse: tempo real.

  `double MPI_Wtime()`

  – Tempo em segundos desde um momento arbitrário no passado

  `double MPI_Wtick()`

  – Resolução do temporizador

- Como eliminar tempos de inicialização?

  `int MPI_Barrier(MPI_Comm comm)`

  – Barreira de sincronização

Exemplo #1

Problema de MPI: Satisfiabilidade de circuito

Código de Benchmarking

```
double tempo;

...

MPI_Init (&argc, &argv);
MPI_Barrier (MPI_COMM_WORLD);
tempo = - MPI_Wtime();

...

MPI_Reduce (...);
tempo += MPI_Wtime();
```

Exemplo #1

Problema de MPI: Satisfiabilidade de circuito

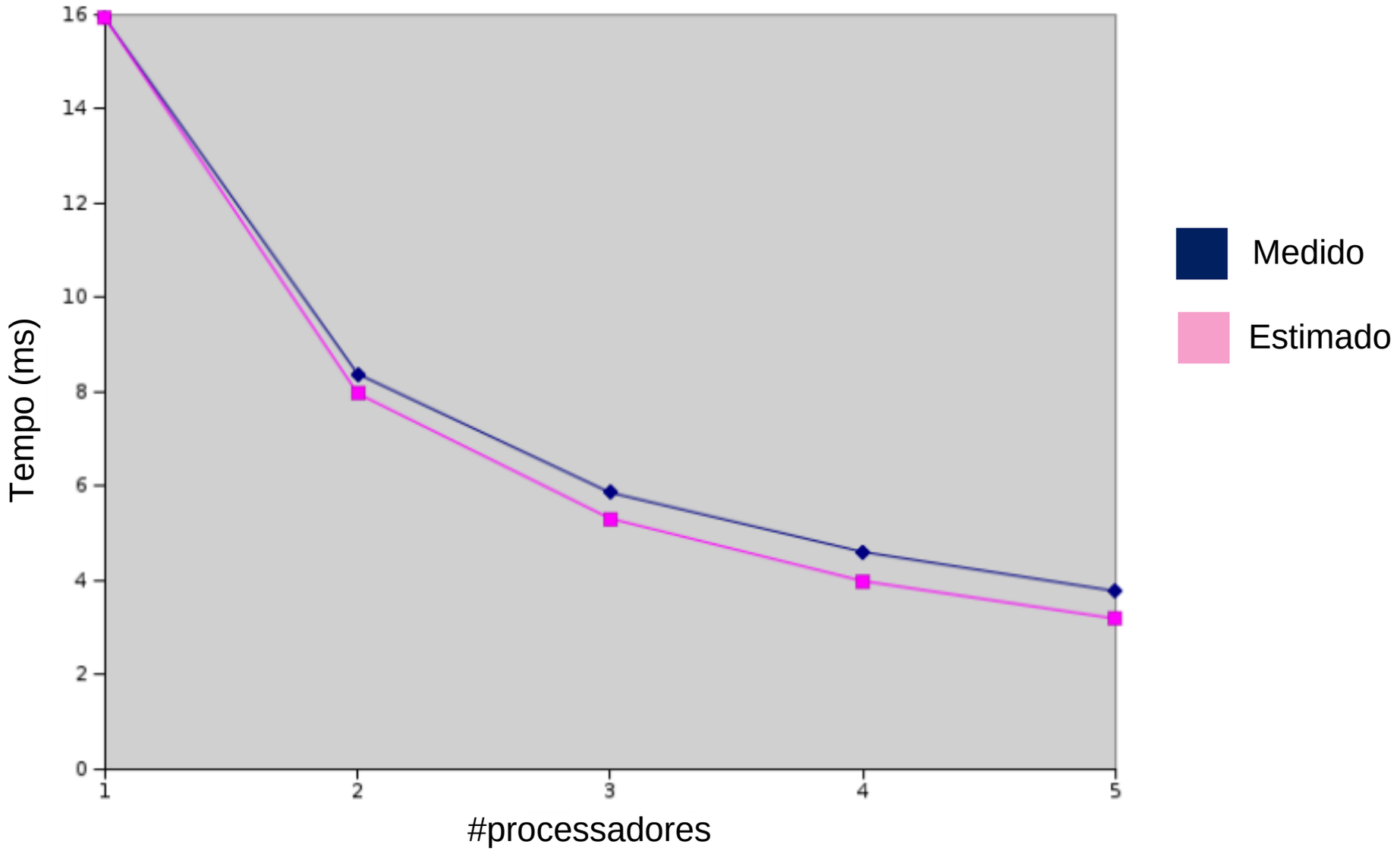
Código de Benchmarking

Processadores	Tempo (ms)
1	15.93
2	8.38
3	5.86
4	4.60
5	3.77

Exemplo #1

Problema de MPI: Satisfiabilidade de circuito

Resultado do Benchmarking



Revisão

- Contexto
- Conceitos básicos
- Comunicadores
- Mensagens
  - Ponto-a-ponto
  - Muito-para-muito
- Exemplos de aplicação

Referências

- Consultar a pasta “References” dentro da pasta do tema.

Bom trabalho!!!!