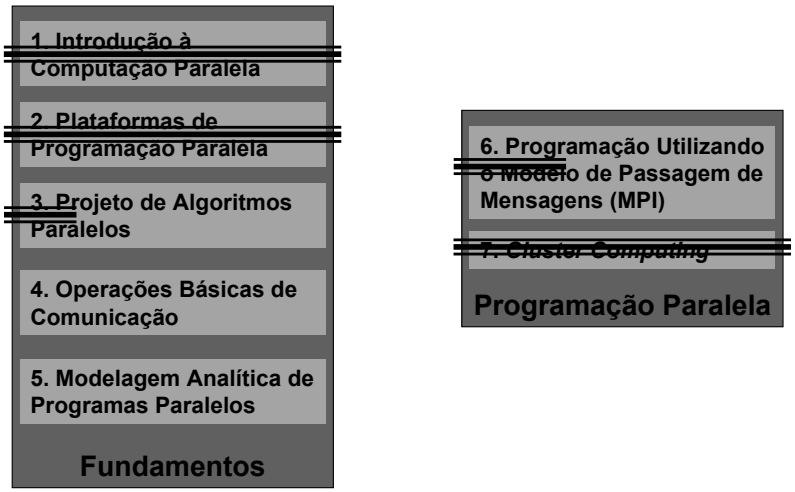
Programação MPI

Conceitos Básicos

Aula 11
Alessandro L. Koerich

Programação Paralela
Pontifícia Universidade Católica do Paraná (PUCPR)
Ciência da Computação – 6º Período

Programa do PA



Aula Anterior

- Fundamentos do Projeto de Algoritmos Paralelos
  - Decomposição
  - Tarefas
  - Granularidade
  - Dependência
  - Interação
  - Mapeamento
  - Processos
- *Cluster Computing*

Introdução

- Programação paralela usando MPI
- Objetivo: Começar a escrever programas paralelos simples.

Plano de Aula

- Passagem de Mensagens
- Introdução ao MPI
- Terminologia e Conceitos
- Estrutura de um Programa MPI
- Exemplo

Biblioteca de Rotinas para Troca de Mensagens

- Necessita–se explicitamente definir:
  - Quais processos serão executados
  - Quando passar mensagens entre processos concorrentes
  - O que passar nas mensagens
- Dois métodos primários:
  - Um para criação de processos separados para execução em diferentes processadores
  - Um para enviar e receber mensagens
- MPI

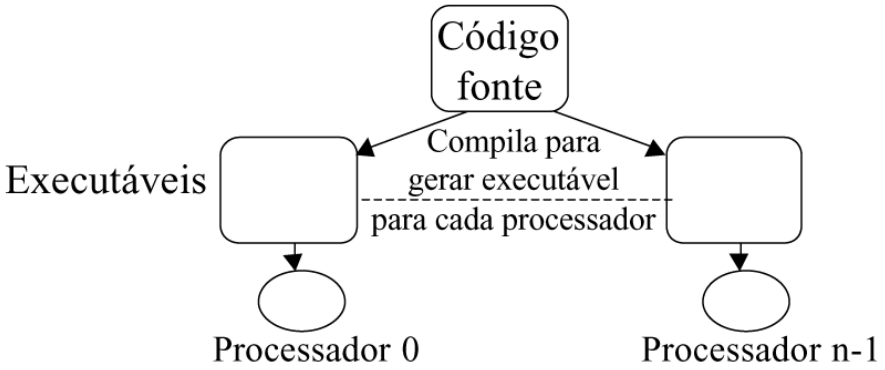
Programação por Passagem de Mensagens

- Consiste de múltiplas instâncias de um programa serial que se comunicam por chamadas da biblioteca
- Quatro classes de chamadas:
  - Inicialização, gerenciamento e conclusão
  - Comunicação entre um par de processos
  - Comunicação entre grupos de processos
  - Criação de tipos de dados arbitrários

Criação e Execução de Processos

- Não são definidos e dependem da implementação
- Criação estática de processos
  - Devem ser definidos antes da execução e inicializados juntos
- Utiliza o modelo de programação SPMD (*Single Program Multiple Data*)

Criação e Execução de Processos



Introdução ao MPI

- O que é MPI ?
- Os objetivos e escopo de MPI
- Estrutura de programas MPI
- Programação MPI
- Exemplo de implementação

O que é MPI ?

- MPI é o acrônimo para "*Message Passing Interface*".
- É uma biblioteca de funções e macros que pode ser usada em programas em C, Fortran 77 e C++.
- Foi desenvolvida em 1993–1994 por um grupo de pesquisadores da industria, governo e academia
- O primeiro padrão para programação paralela

Objetivos do MPI

- Portabilidade do código fonte
- Implementação eficiente em diversas arquiteturas
- Suporte para arquiteturas paralelas heterogêneas
- Funcionalidade: diferentes tipos de comunicação

Por que (não) utilizar MPI ?

- Você deve utilizar MPI quando necessitar:
  - Programas paralelos portáveis
  - Alta performance em programação paralela
- Entretanto, MPI não deve ser utilizado quando:
  - Outros modelos puderem oferecer uma performance e portabilidade aceitáveis
  - Bibliotecas de rotinas paralelas pré-existentes
  - Não necessitar de paralelismo

Estrutura de Programas MPI

- MPI é grande → 128 funções
  - Funcionalidade extensiva
  - Não é uma medida de complexidade
- MPI é pequena → 6 funções
  - Muitos programas paralelos necessitam somente de 6 funções básicas
- MPI é apropriada
  - Flexibilidade quando necessário
  - Não é necessário conhecer todas as partes do MPI

Um Programa MPI Genérico

- Exemplo: *greeting.c*
  - inclui arquivos cabeçalho MPI
  - declaração de variáveis
  - inicializa o ambiente MPI
  - ...faz computação e chamadas de comunicações MPI ...
  - fecha comunicações MPI

Convenções

- As constantes, *templates* e protótipos MPI estão no arquivo *header mpi.h*
- Todas entidades MPI tem o prefixo MPI_
- Funções
  - *MPI_Init, MPI_Finalize*, etc.
  - Retornam um int
- Constantes
  - *MPI_REAL, MPI_CHAR*
- *Handles* (identificadores)
  - *MPI_SUCCESS*: código de erro

Solução MPI: *Communicators*

- *Communicators*
  - Define o escopo das operações de comunicação
  - domínio de comunicação que define um grupo de processos que podem se comunicar entre si
  - utilizados em todas comunicações ponto-a-ponto e coletivas
  - Processos têm posições definidas associadas ao *communicator*
  - Processos são representados por um *rank* único que é numerado (0, 1, 2, ..., *p*-1), onde *p* é o número de processos
- *Communicator default*: MPI_COMM_WORLD
  - Significa "todos os processos na aplicação MPI"
  - Fornece todas as informações necessárias para realizar a passagem de mensagens
  - Entretanto: conjunto de rotinas MPI para formar novos *communicators*

Seis Funções MPI

- *MPI_Init*
- *MPI_Finalize*
- *MPI_Comm_rank*
- *MPI_Comm_size*
- *MPI_Send*
- *MPI_Recv*

Funções *MPI_Init* e *MPI_Finalize*

- *MPI_Init*(*int *argc, char **argv*[ ]);
- *MPI_Finalize*(*void*);
- Estas duas rotinas são responsáveis pela inicialização e limpeza
- Os argumentos da linha de comando são passados à *MPI_Init*()

Funções *MPI_Comm_rank* e *MPI_Comm_size*

- Dois números são muito úteis para a maioria das aplicações paralelas:
  - Número total de processos paralelos: MPI_Comm_size (MPI_Comm comm, int *size);
  - Identificação do processo: MPI_Comm_rank (MPI_Comm comm, int *rank);
- Esta informação é obtida do communicator MPI_COMM_WORLD usando as rotinas MPI_Comm_size() e MPI_Comm_rank().

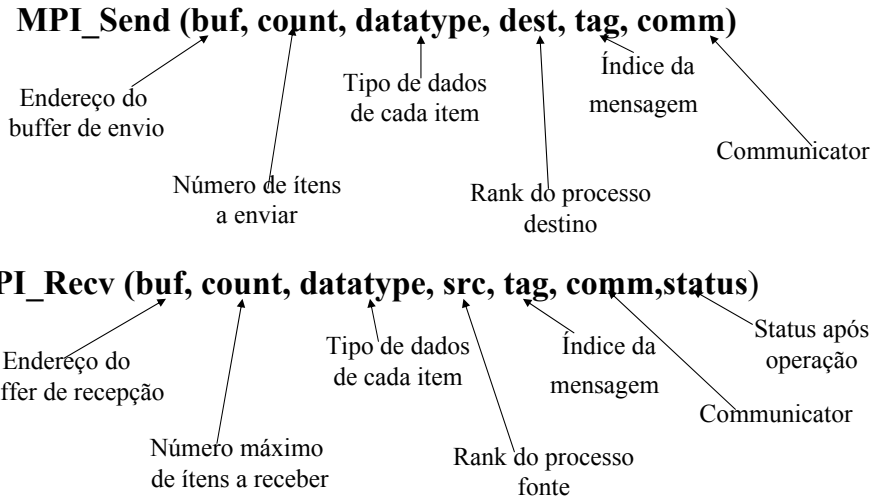
Funções MPI_Send e MPI_Recv

- MPI_Send (void *buf, int count, MPI_Datatype dtype, int dest, int tag, MPI_Comm comm);
- MPI_Recv (void *buf, int count, MPI_Datatype dtype, int source, int tag, MPI_Comm comm, MPI_Status *status);

Comunicação ponto-a-ponto

- Índices para as mensagens
  - MPI_ANY_TAG: para não especificar índice
- Para receber de qualquer fonte: MPI_ANY_SOURCE
- Tipo de dados é enviado como parâmetro na mensagem
- Rotinas com bloqueio
  - retornam quando estão completas localmente (local utilizado para guardar a mensagem pode ser utilizado novamente sem afetar envio da mensagem
- Rotinas sem bloqueio
  - retornam imediatamente mesmo que o local utilizado para guardar a mensagem não possa ser utilizado novamente sem afetar seu envio

Rotinas com bloqueio



Exemplo de rotina com bloqueio

Envio de um inteiro *x* do processo 0 para o processo 1

```
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank ==0) {
  int x;
  MPI_Send(&x, 1, MPI_INT, 1, msgtag, MPI_COMM_WORLD);
}
else  if (myrank ==1) {
  int x;
  MPI_Recv(&x, 1, MPI_INT, 0, msgtag, MPI_COMM_WORLD, status);
}
```

Tipos de Dados

- Declaradas como em “C”
- Ex:

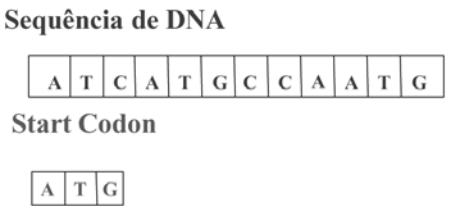
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_FLOAT	float
etc...	

Exemplo Utilizando Seis Funções MPI

```
#include <stdio.h>
#include "mpi.h"

main(int argc, char** argv) {
    int my_rank;        /* Rank of process */
    int p;              /* Number of processes */
    int source;         /* Rank of sender */
    int dest;           /* Rank of receiver */
    int tag = 50;       /* Tag for messages */
    char message[100];  /* Storage for the message */
    MPI_Status status;  /* Return status for receive */

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &p);

    if (my_rank != 0) {
        sprintf(message, "Greetings from process %d!",
            my_rank);
        dest = 0;
        /* Use strlen(message)+1 to include '\0' */
        MPI_Send(message, strlen(message)+1, MPI_CHAR, dest,
            tag, MPI_COMM_WORLD);
    } else { /* my_rank == 0 */
        for (source = 1; source < p; source++) {
            MPI_Recv(message, 100, MPI_CHAR, source, tag,
                MPI_COMM_WORLD, &status);
            printf("%s\n", message);
        }
    }

    MPI_Finalize();
} /* main */
```

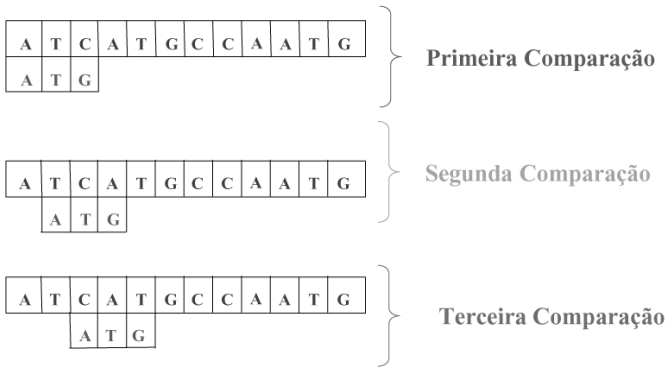

Exemplo de Implementação

- Problema Real: Encontrar o start codon ATG em seqüências de DNA



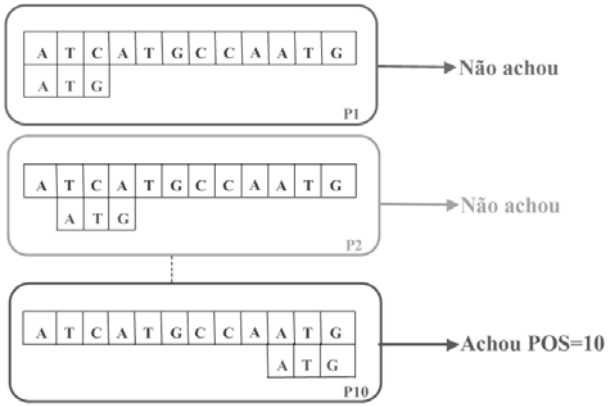
Exemplo de Implementação

- Encontrar o start codon ATG em seqüências de DNA



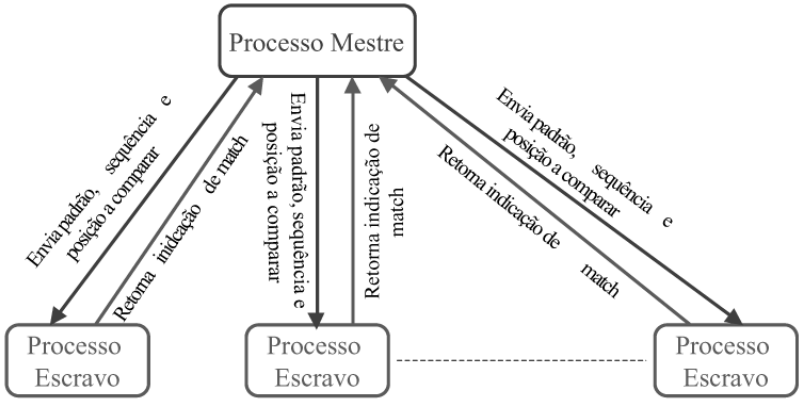
Exemplo de Implementação

- Encontrar o start codon ATG em seqüências de DNA



Exemplo de Implementação

- Encontrar o start codon ATG em seqüências de DNA



Exemplo de Implementação: Processo Mestre

- Envia padrão P[1...N] aos escravos
- Para cada seqüência Si[1...M] faça
  - envia Si para cada processador
  - verifica quantas comparações devem ser feitas por cada processador (M–N+1)/NP
  - envia o intervalo de comparações para cada processador
  - espera resultado de cada processador
- Envia indicação de fim de processo para cada processador

Exemplo de Implementação: Processo Escravo

- Recebe padrão P[1...N] do mestre
- Enquanto não recebe indicação de fim de processo do mestre, faça
  - recebe Si
  - recebe as comparações que devem ser feitas
  - executa as comparações
  - envia resultado para mestre

Sumário de MPI

- Portabilidade
- Alta performance
- Implementação eficiente em diversas arquiteturas
- Suporte para arquiteturas paralelas heterogêneas
- Funcionalidade: diferente tipos de comunicação