CAP-387(2016) – Tópicos Especiais em Computação Aplicada: Construção de Aplicações Massivamente Paralelas

Aula 27: Sistemas de Memória Distribuída

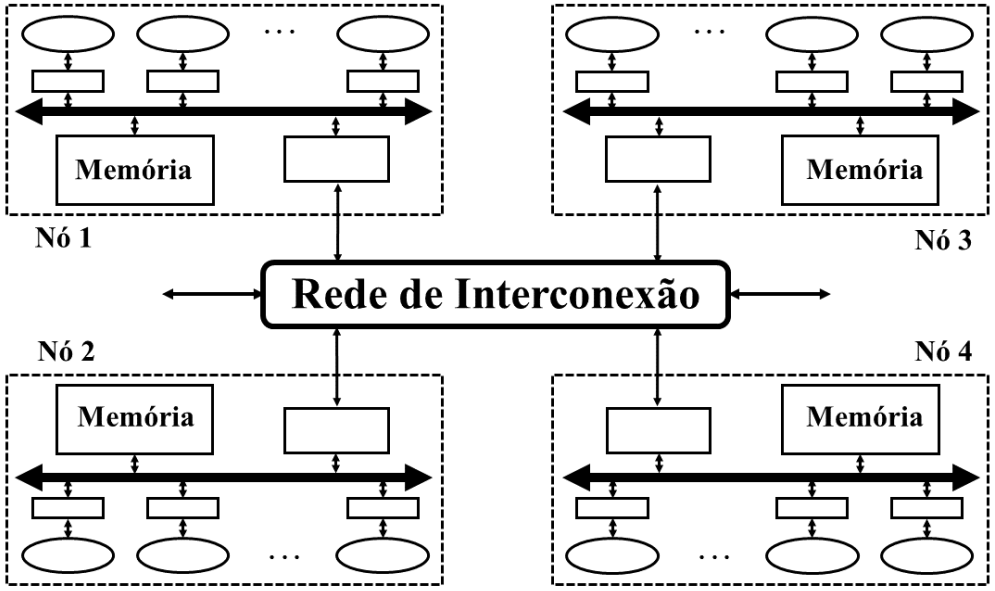
Celso L. Mendes, Stephan Stephany

LAC / INPE

Emails: celso.mendes@inpe.br, stephan.stephany@inpe.br

Prog. em Memória Distribuída

- Objetivo: explorar todas as CPUs, com overhead de comunicação entre elas tolerável
- Paradigma preferencial: troca de mensagens entre CPUs
  - Um ou mais processos por nó
  - Mensagens entre processos
  - Participação necessária do Sistema Operacional



Troca de Mensagens: Histórico

- Ambiente de clusters de estações de trabalho:
  - PVM: Parallel Virtual Machine *http://www.csm.ornl.gov/pvm/*
    - Biblioteca com funções para criar processos e trocar dados
  - Desenvolvido por usuários de labs nacionais nos EUA
- Ambiente de máquinas paralelas:
  - Cada fabricante fornecia sua biblioteca de troca de mensagens
    - Ex: IBM (SP2), Intel (iPSC, Paragon), etc
  - Mudança de plataforma sempre exigia ajustes no código

Troca de Mensagens (cont.)

- Padronização: Message-Passing Interface (MPI)
  - Fórum MPI criado na década de 90: indústria, labs, universidades
  - Padronização da interface (biblioteca) para mensagens
  - Especificação para Fortran, C e C++ (C++ não é suportada hoje)
  - Implementação modelo: MPICH (Argonne National Lab - EUA)
    - Código totalmente aberto, com documentação, etc (www.mpich.org)
    - Licenciada por diversos fabricantes
  - Forte adesão de toda a comunidade

Evolução de MPI

- Versões oficiais
  - MPI-1: 1994, MPI-2: 1997, MPI-3: 2012, MPI-4: ?
  - Documentação: MPI Forum *http://mpi-forum.org/*
- MPI-1:
  - Início das atividades do fórum em 1992, no SC
  - Participantes dos EUA, Europa, Japão
  - Funções ponto-a-ponto (com ou sem bloqueio), coletivas, datatypes, topologias de processadores
  - Interfaces de C e Fortran77
  - Lançamento official: verão de 1994

Evolução de MPI (cont.)

- MPI-2:
  - Início das atividades no fórum em 1995
  - Várias inovações
    - I/O paralelo, comunicação unilateral (put/get), gerenciamento dinâmico de processos, suporte a múltiplos threads, etc.
  - Interfaces adicionais para Fortran90 e C++, suporte para uso de múltiplas linguagens num programa
  - Lançamento official: 1997
    - Largamente utilizado por cerca de uma década

Evolução de MPI (cont.)

- Limitações de MPI-2, após uma década:
  - Grande aumento do paralelismo no hardware
  - Amplo surgimento de processadores multi-core (após 2004)
    - Programação com múltiplos threads ganha importância
  - Suporte para RDMA (Remote Direct Memory Access) em redes, mais amplo que o previsto em MPI-2/RMA
  - Evolução das linguagens: C, C++, Fortran
  - Experimentos/testes de sucesso com coletivas sem bloqueio
  - Interesse crescente em tolerância da falhas

Evolução de MPI (cont.)

- Revisões de MPI-2
  - Início das discussões no forum: 2008
  - Versões oficiais: MPI-2.1(2008), MPI-2.2 (2009)
- MPI-3 (incluído neste curso)
  - Exploração de ambientes de memória compartilhada
  - Operações coletivas sem bloqueio (*nonblocking*)
  - Interface específica para ferramentas
  - Revisão/melhoria de operações unilaterais
  - Versões oficiais: MPI-3.0 (2012), MPI-3.1 (2015)

MPI Básico

- Cada processo tem um identificador (*rank*: 0,1,2,…)
- Implementação de MPI: biblioteca libmpi.a
- Funções ponto-a-ponto: 2 elementos envolvidos
  - Exemplos: MPI_Send(), MPI_Recv()
- Funções coletivas: Todos os elementos são envolvidos
  - Exemplos: MPI_Bcast(), MPI_Reduce(), MPI_Barrier()
- Diferentes tipos de funções de envio/recebimento
  - síncrono × assíncrono
  - {com × sem} bloqueio

MPI Básico (cont.)

- Seis funções *canônicas*:
  - MPI_Init(): inicialização da biblioteca
  - MPI_Finalize(): encerramento da biblioteca
  - MPI_Comm_size(): retorna número de processos
  - MPI_Comm_rank(): retorna identificador (rank)
  - MPI_Send(): envia mensagem
  - MPI_Recv(): recebe mensagem

  É possível escrever ~99% dos programas MPI com isto!
  (porém com pouca eficiência em alguns casos)

MPI - Exemplo

```
#include “mpi.h”
int main(int argc,char *argv[])
{
   int rank, size, i, provided;
   float A[10];
   MPI_Init_thread(&argc, &argv, MPI_THREAD_SINGLE, &provided);
   MPI_Comm_size(MPI_COMM_WORLD, &size);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   for (i=0; i<10; i++)  A[i] = I * rank;
   printf(“Rank %d entre %d\n”, rank, size);
   printf(“Valores de A:\n”);
   for (i=0; i<10; i++) printf(“%f “, A[i]);
   printf(“\n”);
   MPI_Finalize();
}
```

Introduzido em MPI-2
(necessário para uso de OpenMP)

Comunicador Universal
(todos os ranks)

Ordenação da saída dos processos indefinida!

Comunicadores em MPI

- Conceito de comunicador (communicator):
  - Sub-grupo de processos
  - Há funções específicas de MPI para criar sub-grupos
- Cada processo tem um rank no sub-grupo
- Se um processo está em mais de um sub-grupo, ele terá um rank em cada sub-grupo
- Toda comunicação ocorre dentro de um sub-grupo
  - Mensagens ponto-a-ponto vão de um rank de origem para um rank de destino, ambos no mesmo sub-grupo
  - Operações coletivas envolvem *todos* os ranks do sub-grupo
- Comunicador especial: MPI_COMM_WORLD (todos os ranks)

Tags em MPI

- Tag em mensagens
  - Número inteiro que acompanha toda mensagem
  - Definido pelo programador
- Objetivo: auxiliar o processo que recebe a mensagem
  - Chamada a MPI_Recv() pode especificar um certo tag
    - Apenas mensagens com aquele tag são consideradas
  - Chamada pode especificar também MPI_ANY_TAG
    - Mensagens com qualquer tag são aceitas
    - Tag da mensagem de fato recebida pode ser obtido em *status*
      - *MPI_Recv(buf,count,datatype,source,tag,communic,status)*

Chamadas com Bloqueio

- Envio de mensagens
  - *MPI_Send(buffer,count,datatypdest,tag,comm)*
  - Retorno ocorre quando o buffer pode ser re-utilizado (nenhuma garantia quanto à recepção da mensagem)
- Recebimento de mensagens
  - *MPI_Recv(buffer,count,datatype,source,tag,comm,status)*
  - Retorno ocorre quando uma mensagem de *source/tag* foi receida e o buffer já contém dados válidos
  - É válido que a quantidade de dados recebidos seja menor que {*count* × *datatype*}
  - tag = status.MPI_TAG; source = status.MPI_SOURCE; MPI_get_count(&status,datatype,&recvd_count)

Bufferização e Deadlocks

- Necessidade de buffers:

  Exemplo: mensagem grande a ser enviada

  Processo 0: Processo 1:

  MPI_Send(…,1,…) *MPI_Send(…,0,…)*

  MPI_Recv(…) *MPI_Recv(…)*

  - Caso não haja buffers suficientes no sistema, nenhuma das duas mensagens pode ir adiante
  - Programa causa deadlock!
  - Tal programa é dito *não-seguro*: seu funcionamento depende da existência de buffers no sistema

Alternativas para Bufferização

- Possíveis soluções para evitar deadlock:

a) Ordenar programa mais cuidadosamente

Processo 0:	Processo 1:
MPI_Send(…,1,…)	*MPI_Recv(…)*
MPI_Recv(…)	*MPI_Send(…,0,…)*

b) Utilizar *MPI_Sendrecv(…)*
   - buffers de envio e de recebimento são passados

c) Utilizar send com buffer explícito – *MPI_Bsend(…)*

d) Utilizar funções MPI sem bloqueio

Funções MPI sem Bloqueio

- Funções de envio/recebimento:

  MPI_Request request;

  MPI_Isend(…,comm,&request), MPI_Irecv(…,comm,&request)

  - Retorno das chamadas é imediato

- Funções de apoio:

  MPI_Wait(&request) : bloqueio até o término da operação

  MPI_Test(&request, &flag, &status) : teste do término (*flag*)

  - Em caso de término, *status* recupera os valores da operação

  MPI_Waitall(count,request[],status[])

  MPI_Waitany(count,request[],&index,&status)

  MPI_Waitsome(incount,request[],&outcount,index[],status[])

Funções MPI sem Bloqueio (cont.)

- Funções de apoio (cont.):
  - Há versões de *MPI_Test* correspondentes a *Waitall, Waitany, Waitsome:*
    - *MPI_Testall:* testa a chegada de todos os *requests* indicados
    - *MPI_Testany:* testa a chegada de algum dos *requests*
    - M*PI_Testsome*: testa a chegada de alguns dos *requests*
- Teste de chegada de mensagem, com/sem bloqueio

  *MPI_Iprobe(source, tag, comm, *flag, *status)*

  Retorna imediatamente, com flag=1 se há mensagem com *source/tag/comm* que satisfaçam os valores indicados

Modos de Comunicação em MPI

- Modo Síncrono: *MPI_Ssend(…)*
  - Chamada não retorna até que um *recv* tenha sido iniciado
- Modo com Buffer: *MPI_Bsend(…)*
  - Programador provê um buffer pré-alocado, a ser usado pelo sistema
- Modo "Pronto" (Ready): *MPI_Rsend(…)*
  - Programador garante que um *recv* já foi chamado
  - Comportamento indefinido se o *recv* não houver sido chamado
- OBS: *MPI_Recv(…)* pode receber mensagem com qualquer modo

Temporização em MPI

- Leitura do relógio (wall time)

  double MPI_Wtime()

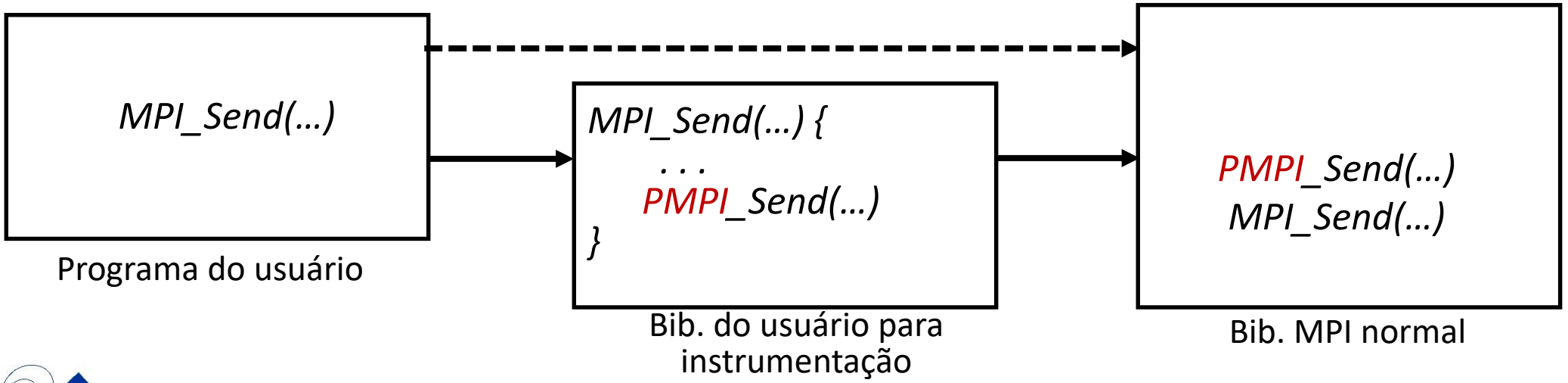
- Resolução do relógio

  double MPI_Wtick()

- Leituras do relógio, em geral, são *locais*
- Em raros casos, pode haver um relógio global
  - MPI_WTIME_IS_GLOBAL tem valor 1

MPI - Interface de Instrumentação

- Objetivo: permitir o uso de ferramentas de *profiling*
- Implementação:
  - Para cada função *MPI_Xyz(…)*, há uma cópia *PMPI_Xyz(…)*
  - Passar ao linker primeiro a nova biblioteca de instrumantação



Protocolos de Envio em MPI

- Formas de envio dos dados da mensagem
  - Modo “eager”: origem envia os dados imediatamente, assumindo que existe espaço de buffer no destino
  - Modo “rendezvous”: (a) origem envia requisição; (b) destino devolve autorização; (c) origem envia dados
- Principais características:
  - Modo *eager* é mais rápido, mas assume que há buffer no destino
  - Modo *rendezvous* é mais lento porém mais seguro
  - Afeta apenas a implementação do envio; usuário não precisa fazer nada, porém o desempenho será afetado
  - Em geral, bibliotecas MPI implementam um limiar: mensagens menores que o limiar usam modo *eager*, maiores usam *rendezvous*
    - Tipicamente, é possível mudar o limiar através de uma variável ambiental

Modelo de Desempenho

- Tempo de Comunicação
  - $T(n) = a + b \cdot n$, onde:
    - a: custo fixo por mensagem (latência)
      - Envolve custos de hardware e de software
      - Sistemas atuais: 1~10 µs
    - b: custo de envio por byte transmitido
    - $1/b$: largura de banda do canal de transmissão
      - Sistemas atuais: 0,1~10 GB/s
    - n: tamanho da mensagem, em bytes
- Forma típica de medição experimental
  - Envios bi-direcionais (ping-pong)
  - Medição apenas pelo lado originário, $T_{msg}=T_{total} \div 2$