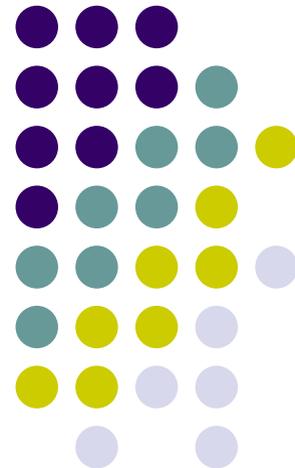


Computação Paralela

Passagem de Mensagens

João Luís Ferreira Sobral
Departamento de Informática
Universidade do Minho

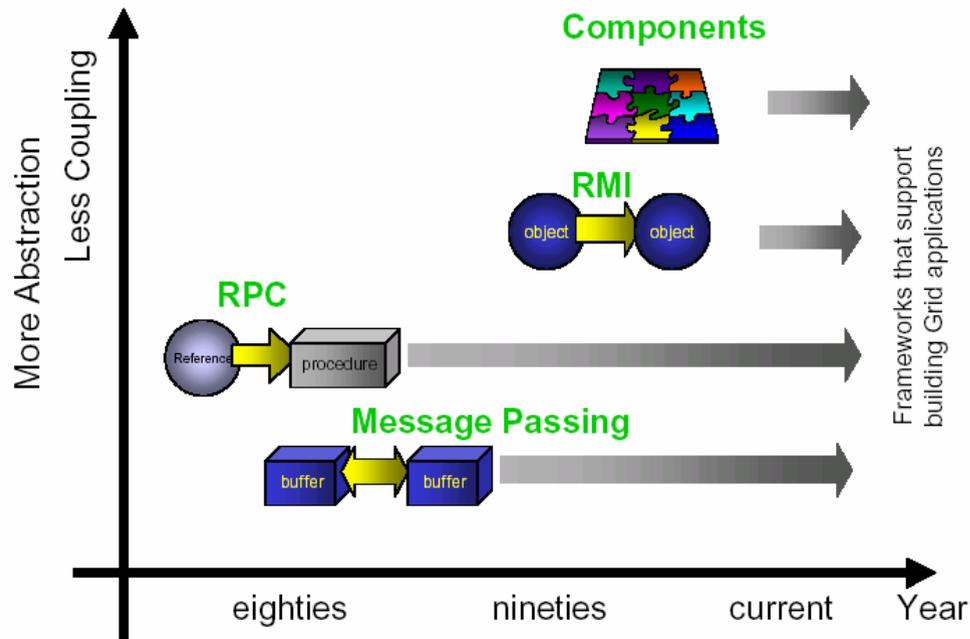
Novembro 2005





Passagem de Mensagens

Paradigmas existentes para aplicações distribuídas



- .A passagem de mensagens (ainda?) é mais eficiente que RMI



Passagem de Mensagens

Conceitos base

- Especificação do paralelismo através de processos com um espaço de endereçamento próprio
- Processos podem ser idênticos (SPMD) ou não (MIMD)
- Identificação de entidades através de portas
- Envio e recepção explícito de mensagem (de/para uma porta específica)
- Empacotamento/desempacotamento explícito da informação em mensagens
- Primitivas mais elaboradas para comunicação (difusão/redução/barreiras)





Passagem de Mensagens

Passagem de mensagens VS Invocação remota de métodos (RMI)

	Passagem de Mensagens	Invocação Remota de Métodos
Dados a transmitir	Empacotamento de dados	Lista de parâmetros
Envio de pedidos/ informação	Envio explícito de mensagens etiquetadas	Invocação de um método específico
Recepção de pedidos/ informação	Recepção explícita de mensagens	Execução implícita do método invocado
Reacção do receptor	A acção a executar é determinada pela etiqueta (tag) da mensagem	A acção a executar é determinada pelo método invocado
Identificação do receptor	Canal, nome ou anónima	Apontador para objecto remoto (proxy)



Passagem de Mensagens

MPI (Message Passing Interface) <http://www.mpi-forum.org>

- Standard para a passagem de mensagens que surgiu do esforço de várias entidades para tornar as aplicações paralelas mais portáteis
- Baseado num modelo SPMD (o mesmo processo é executado em várias máquinas)
- Implementado através de uma biblioteca de funções
- Implementações mais frequentes (de uso livre): MPICH e LamMPI
- Suporta vários modos de passagem de mensagens, grupos de comunicação e uma grande variedade de operações colectivas



Passagem de Mensagens

MPI (Funcionalidades)

- Iniciação e terminação
 - **MPI::Init MPI::Finalize**
- Identificação do ambiente de execução
 - **Número de processos total: MPI::Comm.Get_size**
 - **Número do processo actual: MPI::Comm.Get_rank**

- Exemplo 1 (C++)

```
#include "mpi.h"
#include <iostream>
int main( int argc, char *argv[] ) {
    int rank, size;
    MPI::Init(argc, argv);
    rank = MPI::COMM_WORLD.Get_rank();
    size = MPI::COMM_WORLD.Get_size();
    std::cout << "I am " << rank << " of " << size << "\n";
    MPI::Finalize();
    return 0;
}
```



Passagem de Mensagens

MPI (Funcionalidades – cont.)

- Compilação e execução dos programas (MPICH)
 - **compilar:** mpicxx (em /usr/local/mpich-1.6.1/bin)
 - **executar:** mpirun -np <nºprocessos> a.out
- Comunicação entre processos
 - cada processo é identificado pela sua ordem dentro de um grupo (por defeito existe um grupo que inclui todos os processos: COMM_WORLD)
 - no envio e recepção de mensagens é necessário especificar o tipo de dados da mensagem (MPI::INT, MPI::DOUBLE, etc)
 - cada mensagem inclui uma etiqueta que permite distinguir as mensagens de uma mesma fonte
 - Envio: MPI::comm.Send(buf, count, datatype, dest, tag)
Retorna assim que a mensagem foi colocada no *buffer* do receptor
 - Recepção: MPI::comm.Recv(buf, count, datatype, src, tag)
ANY_SOURCE e ANY_TAG permitem a recepção de várias fontes e etiquetas Iniciação e terminação



Passagem de Mensagens

MPI (Funcionalidades – cont.)

- **Exemplo 2 (C++)**

```
#include "mpi.h"
#include <iostream>
int main( int argc, char *argv[] ) {
    int rank, buf;
    MPI::Init(argv, argc);
    rank = MPI::COMM_WORLD.Get_rank();
    // Process 0 sends and Process 1 receives
    if (rank == 0) {
        buf = 123456;
        MPI::COMM_WORLD.Send( &buf, 1, MPI::INT, 1, 0 );
    }
    else if (rank == 1) {
        MPI::COMM_WORLD.Recv( &buf, 1, MPI::INT, 0, 0 );
        std::cout << "Received " << buf << "\n";
    }
    MPI::Finalize();
    return 0;
}
```



Passagem de Mensagens

MPI (Funcionalidades – cont.)

- **Exemplo 2 (C)**

```
#include "mpi.h"
#include <stdio.h>
int main( int argc, char *argv[] ) {
    int rank, buf;
    MPI_Status status;
    MPI_Init(&argv, &argc);
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    /* Process 0 sends and Process 1 receives */
    if (rank == 0) {
        buf = 123456;
        MPI_Send( &buf, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
    }
    else if (rank == 1) {
        MPI_Recv( &buf, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status );
        printf( "Received %d\n", buf );
    }
    MPI_Finalize();
    return 0;
}
```



Passagem de Mensagens

MPI – Outras funções

- BCast – distribui os dados a vários processos
- Reduce – recebe dados de vários processos
- Barrier – Espera que todos os processos terminem
- Envio e recepção assíncrona de mensagens:

```
MPI::Request request;  
MPI::Status status;  
request = MPI::comm.Isend(start, count, datatype, dest, tag);  
request = MPI::comm.Irecv(start, count, datatype, dest, tag);  
request.Wait(status);  
  
flag = request.Test( status );
```



Passagem de Mensagens

Implementação em Java

- mpiJava – distribuição mais conhecida, “binding” para MPI (requer a instalação de MPI)
- MPJE – sucessor de mpiJava com suporte transparente a MPI e “nio”; versão preliminar
- CCJ, JMPI – implementações em java.io; fraco desempenho
- MPP – baseada em “nio” de Java 4; versão mais eficiente, não segue o standard MPI

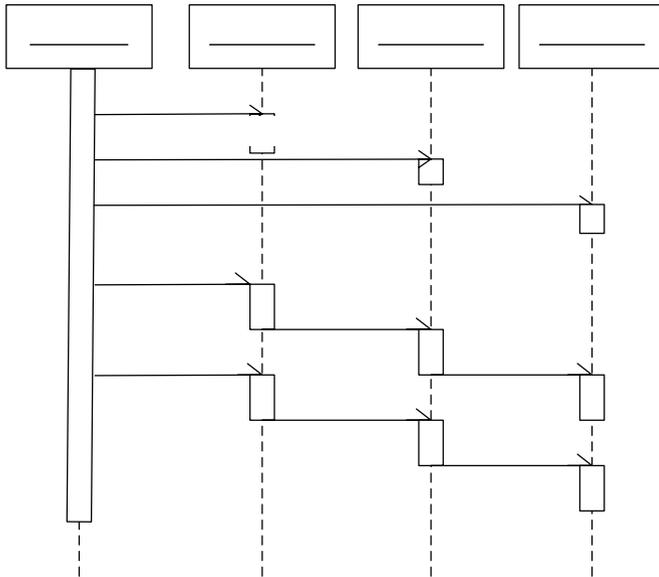
```
Communicator comm=new BlockCommunicator(); // package mpp.*
int nproc = comm.size();
int myrank = comm.rank();
int [] buf = new int[1];
// Process 0 sends and Process 1 receives
if (rank == 0) {
    buf[1] = 123456;
    comm.send( buf, 0, 1, 1 ); // send(int[] data, int offset, int length, int peer)
} else if (rank == 1) {
    comm.recv( buf, 0, 1, 0 ); // recv(int[] data, int offset, int length, int peer)
    System.out.println("Recebi" + buf[0]);
}
}
```



Passagem de Mensagens

Exemplo: Cálculo de números primos (RMI vs MPI)

- **JavaRMI** – O gerador invoca o método *init* em cada filtro. O método *filter* é invocando entre filtros
- **MPI** – Os parâmetros de *init* são passados na linha de comandos. Os pacotes de números dever ser recebidos explicitamente e enviados ao filtro seguinte após o processamento.



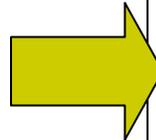


Passagem de Mensagens

Exemplo: Cálculo de números primos (RMI vs MPI), cont.

- Cadeia de três objectos/processos para calcular os números primos:

```
public static void main(String[] args) {  
  
    Generator2 g = new Generator2();  
    int MaxP = 10000000;  
    int SMaxP = (int) Math.sqrt(MaxP);  
  
    IPrimeFilter [] filtros = new IPrimeFilter[3];  
    for (int i = 0; i<3; i++) filtros[i] = new PrimeFilter();  
  
    filtros[0].init(3, SMaxP/3, SMaxP, filtros[1], 10);  
    filtros[1].init(SMaxP/3+1, 2*SMaxP/3, SMaxP, filtros[2], 10);  
    filtros[2].init(2*SMaxP/3+1, SMaxP, SMaxP, null, 10);  
  
    int pack = MaxP/10;  
    for(i=0;i<10;i++) g.generate(i*pack, (i+1)*pack, filtros[0]);  
  
}
```



```
int main(int argc, char *argv[]) {  
    ...  
    myrank = MPI::COMM_WORLD.Get_rank();  
    ...  
    if (myrank==0) {  
        ... // criar e iniciar filtro local  
        ... // gerar pacotes de números  
        ... // processar  
        MPI::COMM_WORLD.Send(...);  
    } else if(myrank==1) {  
        ... // criar e iniciar filtro local  
        MPI::COMM_WORLD.Recv(...);  
        ...// processar  
        MPI::COMM_WORLD.Send(...);  
    }  
    else {  
        ... // criar e iniciar filtro local  
        MPI::COMM_WORLD.Recv(...);  
        ...// processar  
    }  
}
```



Passagem de Mensagens

Exercícios

- Alterar o código para implementar um *farming* com a biblioteca MPP.
- Alterar o código anterior para implementar uma *Pipeline*