Programação em Computação Paralela e Distribuída

Laboratório, ERBASE 2009

MPI

Domingo Giménez

Universidad de Murcia, Spain

Grupo de Computación Paralela

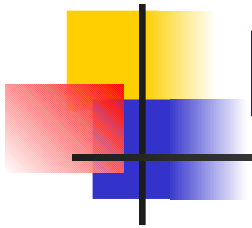
http://dis.um.es/~domingo

Murilo Boratto

UFBa

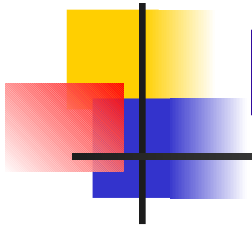
Leandro Coelho

UNEB



Noções básicas: história

- MPI: Message Passing Interface
  - Uma especificación para passo de mensagens
  - A primeira livraria de passagem de mensagens estándar e portable
  - Por consenso MPI Forum. Participantes de umas 40 organizações
  - Acabado e publicado em maio 1994. Actualizado em junho 1995
- Previamente PVM: Parallel Virtual Machine
- Outras versões: MPI2, HMPI



Noções básicas: conteúdo

- Estandarización
- Portabilidade: multiprocesadores, multicomputadores, redes, heterogéneos...
- Boas prestações, se está disponível para o sistema
- Ampla funcionalidad. Umas 140 funções. Programas em C ou Fortran e chamadas as funções da livraria MPI
- Implementaciones gratuitas (mpich, lam, ...)

Exemplo

Todos os processos executam o mesmo código

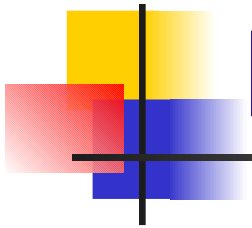
As variables estão replicadas nas memórias de todos os processos

Trabalham com MPI entre Init e Finalize

codigo3-5.c

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

main(int argc, char*argv[]) {
   int myrank, size;
   MPI_Init(&argc,&argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
   MPI_Comm_size(MPI_COMM_WORLD, &size);
   printf("Hola desde el proceso %d de %d\n", myrank, size);
   MPI_Finalize();
}
```



Exemplo

compilação:

mpicc codigo3-5.c -o codigo3-5 -O3

execução:

$> mpirun -np 4 codigo3-5

executa 4 processos iguais nos processadores estabelecidos por defeito

$> mpirun -np 4 -machinefile maquinas codigo3-5

executa-os nos processadores onde se indica no arquivo de máquinas

$> mpirun n0,1,2,3 codigo3-5

os executa nos nodos 0, 1, 2 e 3

Exemplo

compilação:

mpicc codigo3-5.c -o codigo3-5 -O3

execução:

$> mpirun -np 4 codigo3-5

executa 4 processos iguais nos processadores estab

$> mpirun -np 4 -machinefile maquinas codigo3-5

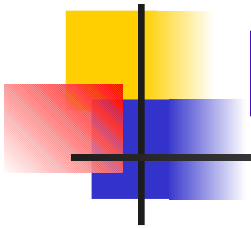
executa-os nos processadores onde se indica no arc

$> mpirun n0,1,2,3 codigo3-5

os executa nos nodos 0, 1, 2 e 3

```
$> mpirun -np 4 codigo3-5
Hola desde el proceso 0 de 4
Hola desde el proceso 2 de 4
Hola desde el proceso 1 de 4
Hola desde el proceso 3 de 4

$> mpirun -np 4 codigo3-5
Hola desde el proceso 0 de 4
Hola desde el proceso 3 de 4
Hola desde el proceso 1 de 4
Hola desde el proceso 2 de 4
```



Funções

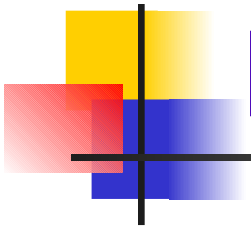
- Incluir livraria: #include <mpi.h>
- Formato das funções: erro=MPI_nome(parámetros ...)

int MPI_Init (int *argc , char ***argv)

- Inizializa MPI

int MPI_Finalize ( )

- Finaliza MPI



Funções

MPI_Comm_rank (MPI_Comm comm , int *rank)

- Obtém o identificador de processo que chama à função
- É um valor entre 0 e *p*-1

MPI_Comm_size (MPI_Comm comm , int *size)

- Obtém o número total de processos, *p*

- Comunicador: Conjunto de processos em que se fazem comunicações

MPI_COMM_WORD: o mundo dos processos MPI

Comunicações

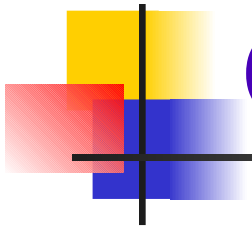
int MPI_Send (void *buffer , int contador , MPI_Datatype tipo , int destino , int tag , MPI_Comm comunicador)

int MPI_Recv (void *buffer , int contador , MPI_Datatype tipo , int origen , int tag , MPI_Comm comunicador , MPI_Status *estado)

- pode-se indicar qualquer tipo ou origem com as constantes

  MPI_ANY_TAG

  MPI_ANY_SOURCE

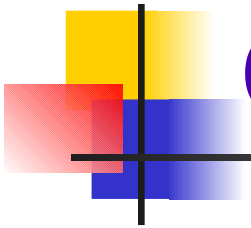


Comunicações

usam-se tipos MPI:

Básicos:

MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHOT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

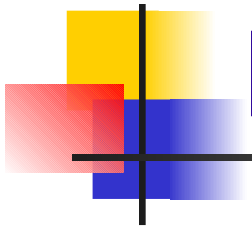


Comunicacões síncronas

- Nas comunicações síncronas o processo que envia continua sua execução, e o que recebe espera a que chegue a mensagem
- O momento em que o que envia continua depende da função de envio:
  - **MPI_Ssend**: acaba quando começa a recepção
  - **MPI_Bsend**: envio com buffer
  - **MPI_Send**: síncrono com buffer
  - **MPI_Rsend**: acaba independentemente da recepção

Comunicacões não síncronas

- **MPI_Isen**d(buf, count, datatype, dest, tag, comm, request)
- **MPI_Irecv**(buf, count, datatype, source, tag, comm, request)
- **MPI_Wait**(request,status): volta se a operação completou-se, espera até que se completa
- **MPI_Test**(request,flag,status): devolve um **flag** dizendo se a operação completou-se



Exemplos

- **codigo3-6.c** mostra um exemplo de "Hello world" com funções de comunicação
- **codigo3-7.c** e **integral.c** são exemplos de aproximação de uma integral definida
- Pode-se praticar com estes exemplos mudando o tipo de dados e de comunicações, e utlizando comunicações asíncronas
- Em **integral.c**

Custo de envio:

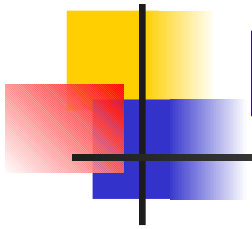
$3(p-1)(t_s+t_w)$

Comunicacões colectivas

- **MPI_Barrier(**) bloqueia os processos até que a chamam todos
- **MPI_Bcast(**) broadcast do processo raiz a todos os demais
- **MPI_Gather(**) recebe valores de um grupo de processos
- **MPI_Scatter(**) distribui um buffer em partes a um grupo de processos
- **MPI_Alltoall(**) envia dados de todos os processos a todos
- **MPI_Reduce(**) combina valores de todos os processos
- **MPI_Reduce_scatter(**) combina valores de todos os processos e distribui
- **MPI_Scan(**) redução prefixa (dados 0,...,i-1 em i)

Comunicacões colectivas

MPI Operator	Operation
MPI_MAX	maximum
MPI_MIN	minimum
MPI_SUM	sum
MPI_PROD	product
MPI_LAND	logical and
MPI_BAND	bitwise and
MPI_LOR	logical or
MPI_BOR	bitwise or
MPI_LXOR	logical exclusive or
MPI_BXOR	bitwise exclusive or
MPI_MAXLOC	max value and location
MPI_MINLOC	min value and location



Exemplo

– **integralCC.c** calcula a integral utilizando comunicações colectivas

int MPI_Bcast (void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)

todos os processadores no comunicador executam a função, e todos indicam qual é o processo raiz

Custo de envio:

$3(t_{sb}+t_{wb})$

mas t_{sb} e t_{wb} são maiores que t_s e t_w

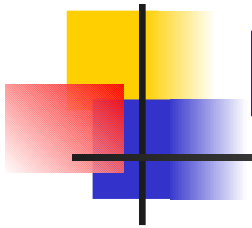
Agrupamento de dados

- Para reduzir o custo das comunicações podem-se agrupar os dados e realizar uma comunicação com os três dados juntos

Custo de envio:

$(p-1)(t_s+3t_w)$

  - Agrupando os dados num array (exemplo **integralAR.c**)
  - Com empacotamento
  - Com tipos derivados



Empacotamento

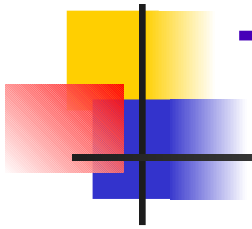
exemplo **integralPack.c**

int MPI_Pack(void *pack_data, int in_count, MPI_Datatype datatype, void *buffer, int size, int *position_ptr, MPI_Comm comm)

int MPI_Unpack(void *buffer, int size, int *position_ptr, void *unpack_data, int count, MPI_Datatype datatype, MPI_Comm comm)

agrupam-se os dados dantes de enviá-los, e depois de recebê-los se desempacotam no mesmo ordem em que se agruparam

- entender o funcionamento do exemplo



Tipos derivados

exemplo **integralTD.c**

- Criam-se em tempo de execução
- Podem-se construir tipos de maneira recursiva
- A criação de tipos requer trabalho adicional
- Especifica-se a disposição dos dados no tipo:

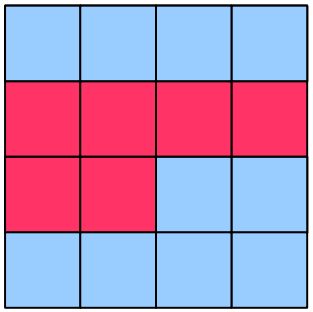
int MPI_Type_Struct (int count, int *array_of_block_lengths, MPI_Aint *array_of_displacements, MPI_Datatype *array_of_types, MPI_Datatype *newtype)

- Chamar à rutina MPI_Type_commit dantes de usar o novo tipo MPI
- Não se calculam as direcções como em C, usa-se a rutina MPI_Address

Tipos derivados

- Se os dados que constituem o novo tipo são um subconjunto de entradas há mecanismos especiais:
- int MPI_Type_contiguous(int count, MPI_Datatype oldtype, MPI_Datatype *newtype)

cria um tipo derivado formado por **count** elementos do tipo **oldtype** contiguos em memória.

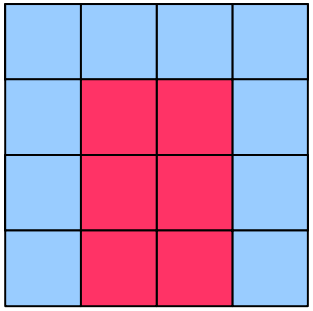


Tipos derivados

- int MPI_Type_vector(int count, int block_lenght, int stride, MPI_Datatype element_type, MPI_Datatype *newtype);

  formado por **count** elementos, a cada um com **block_lenght** elementos de tipo **element_type**. **stride** é o número de elementos de tipo **element_type** entre elementos sucessivos de **new_type**.

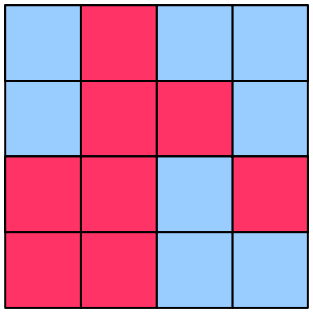
  Os elementos podem ser entradas igualmente espaciadas num array

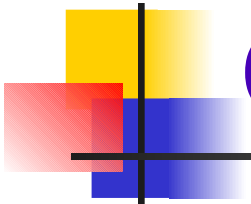


Tipos derivados

- int MPI_Type_indexed(int count, int *array_of_block_lengths, int *array_of_displacements, MPI_Datatype element_type, MPI_Datatype *newtype);

  cria um tipo derivado com **count** elementos; na cada elemento há **array_of_block_lengths**[i] dados de tipo **element_type**, e o deslocamento é **array_of_displacements**[i] unidades do tipo **element_type** desde o começo de **newtype**





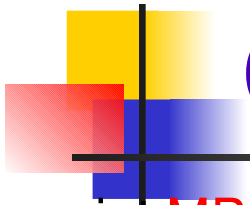
Comunicadores

- **MPI_COMM_WORLD** inclui a todos os processos
- Pode-se definir comunicadores com um número menor de processos: para comunicar dados na cada bicha ou coluna de processos na malha...
- As comunicações em diferentes comunicadores podem ser ao mesmo tempo
- Consta de:
  - um grupo: colecção ordenada de processos numerados 0,...,*p*-1
  - um contexto: um identificador que associa o sistema ao grupo
  - e a um comunicador pode-se-lhe associar uma topología virtual

Comunicadores

- Criação de um comunicador cujos processos são os da primeira bicha de nossa malha virtual. MPI_COMM_WORLD consta de $p=q^2$ processos agrupados em q bichas e colunas. O processo número x tem as coordenadas (x div q,x mod q)

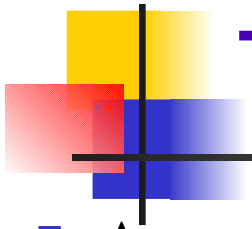
```
MPI_Group  MPI_GROUP_WORLD;
MPI_Group  first_row_group;
MPI_Comm  first_row_comm;
int  row_size;
int  *process_ranks;
process_ranks=(int *) malloc(q*sizeof(int));
for(proc=0;proc<q;proc++)
   process_ranks[proc]=proc;
MPI_Comm_group(MPI_COMM_WORLD,&MPI_GROUP_WORLD);
MPI_Group_incl(MPI_GROUP_WORLD,q,process_ranks,&first_row_group);
MPI_Comm_create(MPI_COMM_WORLD,first_row_group,&first_row_comm);
```



Comunicadores

int MPI_Comm_split(MPI_Comm old_comm, int split_key, int rank_key, MPI_Comm *new_comm)

- Cria um novo comunicador para a cada valor de **split_key**
- Os processos com o mesmo valor de **split_key** formam um grupo
- Se dois processos *a* e *b* têm o mesmo valor de **split_key** e o **rank_key** de *a* é menor que o de *b*, no novo grupo *a* tem identificador menor que *b*
- Todos os processos no comunicador devem a chamar
- Os processos que não se quer incluir em nenhum novo comunicador podem utilizar o valor MPI_UNDEFINDED em **rank_key**, com o que o valor de volta de **new_comm** é MPI_COMM_NULL



Topologías

- A um grupo pode-se-lhe associar uma topología virtual para melhorar as prestações
- As topologías podem ser:
  - de grafo em general
  - de malha ou cartesiana
- Há funções de:
  - criar topologías
  - obter identificadores de processo na topología
  - particionar malhas....