

Sistemas Distribuidos y Paralelos

Ingeniería en Computación



Modelo de programación híbrido

Universidad Nacional de La Plata



Facultad de Informática



Agenda

2

I. Modelo de programación híbrido: Introducción

II. MPI-OpenMP

III. MPI-Pthreads

IV. Modularidad

i. MPI-OpenMP

ii. MPI-Pthreads

iii. Restricciones



Agenda

3

I. Modelo de programación híbrido: Introducción

II. MPI-OpenMP

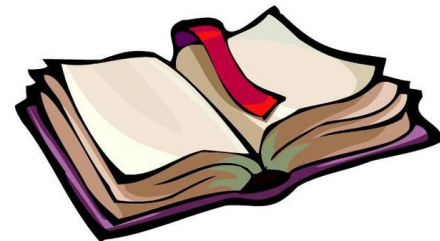
III. MPI-Pthreads

IV. Modularidad

i. MPI-OpenMP

ii. MPI-Pthreads

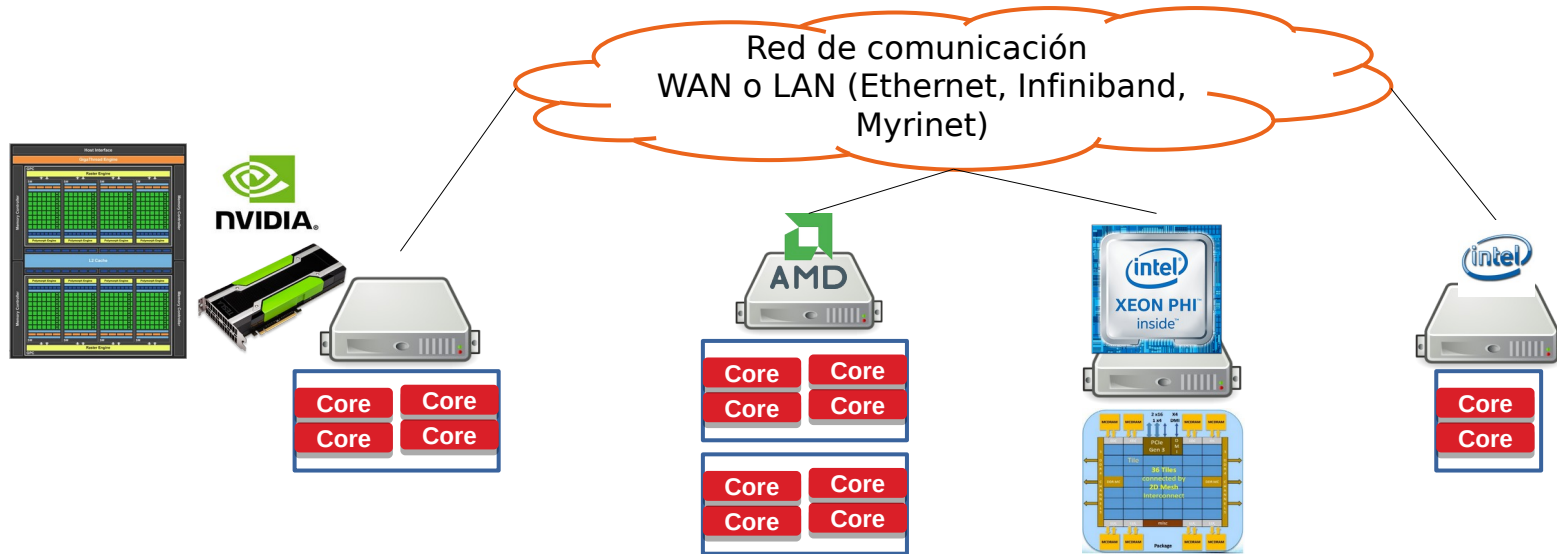
iii. Restricciones



Modelo de programación híbrido

4

- Actualmente, contamos con una gran diversidad de arquitecturas de cómputo.
- Las arquitecturas de memoria compartida y memoria distribuida pueden combinarse formando una **arquitectura híbrida** con mayor potencia de cómputo.



Modelo de programación híbrido

5

- Para explotar toda esta potencia de cómputo es necesario combinar distintas herramientas:



- MPI-OpenMP
- MPI-Pthreads
- Otros:
 - MPI-Cuda, MPI-OpenMP-Cuda, OpenMP-Cuda etc.
- Existen varias razones para elegir una u otra combinación:
 - Herramientas de desarrollo para modelo de memoria compartida (Pthreads y OpenMP) no pueden usarse sobre clusters.
 - Un aplicación MPI sobre multicores puede consumir mas memoria que Pthreads y OpenMP.
 - Uso de herramientas específicas: sobre GPU sólo podemos usar CUDA u OpenCL.
- Estas combinaciones introducen complejidad a la hora de programar y compilar.

Modelo de memoria Híbrido

Modelos de programación paralela – Modelos de arquitectura paralela

6

- El modelo de programación híbrido sólo tiene sentido cuando contamos con una arquitectura híbrida compuesta por un cluster de máquinas multicore.



		Software		
		Memoria Compartida (Ej: OpenMP)	Memoria Distribuida (Ej: MPI)	Híbrido (EJ: MPI + OpenMP)
Hardware	Memoria Compartida (Ej: multicore)	Trivial (Ej: OpenMP sobre Multicore)	Posible (Ej: MPI sobre multicore)	Posible (extraño) (Ej: MPI + OpenMP sobre multicore)
	Memoria Distribuida (Ej: cluster monocore ¹)	(NO ADECUADO) Single System Image(SSI) Overhead	Trivial (Ej: MPI sobre cluster)	Posible (muy extraño) (Ej: MPI + OpenMP sobre cluster monocore)
	Híbrido (Ej: cluster multicore)	(NO ADECUADO) Single System Image(SSI) Overhead	Posible (Ej: MPI sobre cluster multicore)	Trivial (Ej: MPI + OpenMP sobre cluster multicore)

¹Actualmente, los clusters monocore cayeron en desuso pero se mencionan para ejemplificar.

Modelo de programación híbrido

Alternativa de inicialización en MPI

7

- Cuando trabajamos con MPI e hilos, podemos inicializar el entorno MPI con la función `MPI_Init()` o bien utilizar una alternativa para ser mas específicos:

```
int MPI_Init_thread(int *argc, char ***argv, int required, int *provided)
```

- Esta función agrega dos parámetros adicionales:
 - **Required:** nivel deseado de soporte para hilos
 - `MPI_THREAD_SINGLE`: se ejecutará solo un thread.
 - `MPI_THREAD_FUNNELED`: Si el proceso es multihilo, solo el hilo que invocó a `MPI_Init_thread` podrá hacer llamados de MPI.
 - `MPI_THREAD_SERIALIZED`: Si el proceso es multihilo, solo un hilo a la vez podrá hacer llamados MPI.
 - `MPI_THREAD_MULTIPLE`: Si el proceso es multihilo, varios hilos pueden hacer llamados MPI a la vez sin restricciones.
 - **Provided:** valor de retorno. Nivel disponible de soporte para hilos.

Modelo de programación híbrido

Alternativa de inicialización en MPI

8

- Inicializar el entorno con `MPI_Init()` es equivalente a hacerlo con `MPI_Init_thread()` usando la opción `MPI_THREAD_SINGLE`.
- Sin embargo, aún cuando lo inicialicemos con `MPI_Init()` podemos crear hilos y tener una aplicación híbrida.



- La ventaja de usar `MPI_Init_thread()` es “avisarle” al entorno que vamos a crear hilos y de esta forma está preparado para una ejecución “**thread safe**”.



Agenda

9

I. Modelo de programación híbrido: Introducción

II. MPI-OpenMP

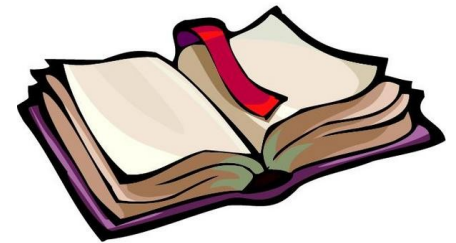
III. MPI-Pthreads

IV. Modularidad

i. MPI-OpenMP

ii. MPI-Pthreads

iii. Restricciones



MPI-OpenMP

Código ejemplo

10

mpi_omp.c

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

int main(int argc, char* argv[]){
    int rank;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);

    #pragma omp parallel
    {
        printf("Hello MPI rank %d thread %d\n",rank,omp_get_thread_num());
    }

    MPI_Finalize();
    return(0);
}
```

MPI-OpenMP

Compilación y ejecución

11

`mpi_omp.c`

Compilar: `mpicc -fopenmp -o mpi_omp mpi_omp.c`

Ejecutar: `mpirun --bind-to none -np 2 -machinefile maquinas mpi_omp`

BUG de MPI!!!

Opción para que no ejecute todos los hilos en el mismo core!!!

Archivo de máquinas

Maquina1 slots=1
Maquina2 slots=1

Posible salida de programa

```
Hello MPI rank 0 thread 0
Hello MPI rank 1 thread 2
Hello MPI rank 1 thread 0
Hello MPI rank 0 thread 1
Hello MPI rank 1 thread 3
Hello MPI rank 1 thread 1
```

Cluster

Maquina1

Rank 0

Core 0

Core 1

Maquina2

Rank 1

Core 0

Core 1

Core 2

Core 3

Agenda

12

I. Modelo de programación híbrido: Introducción

II. MPI-OpenMP

III. MPI-Pthreads

IV. Modularidad

i. MPI-OpenMP

ii. MPI-Pthreads

iii. Restricciones



MPI-Pthreads

Código ejemplo


13

mpi_pthreads.c

```
#include <mpi.h>
#include <stdio.h>
#include <pthread.h>
#include <sys/sysinfo.h>
```

```
int rank;
int T;
```

```
void* funcion(void *arg){
    int id = *(int*)arg;
    printf("Hello MPI rank %d thread %d\n",rank,id);
    pthread_exit(NULL);
}
```



```
int main(int argc, char* argv){

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);

    T=get_nprocs();
    pthread_t hilos[T];
    int hilos_ids[T];
    int id;
    for(id=0;id<T;id++){
        hilos_ids[id] = id;
        pthread_create(&hilos[id], NULL, &funcion,(void*)&hilos_ids[id]);
    }

    for(id=0;id<T;id++)
        pthread_join(hilos[id],NULL);

    MPI_Finalize();
    return(0);
}
```

MPI-Pthreads

Compilación y ejecución

14

```
mpi_pthreads.c
```

Compilar: `mpicc -lpthread -o mpi_pthreads mpi_pthreads.c`

Ejecutar: `mpirun --bind-to none -np 2 -machinefile maquinas mpi_pthreads`

BUG de MPI!!!

Opción para que no ejecute todos los hilos en el mismo core!!!



Archivo de máquinas

```
Maquina1 slots=1  
Maquina2 slots=1
```



Cluster

Maquina1



Core 0

Core 1

Maquina2



Core 0

Core 1

Core 2

Core 3

Posible salida de programa

```
Hello MPI rank 0 thread 0  
Hello MPI rank 0 thread 1  
Hello MPI rank 1 thread 0  
Hello MPI rank 1 thread 1  
Hello MPI rank 1 thread 3  
Hello MPI rank 1 thread 2
```

I. Modelo de programación híbrido: Introducción

II. MPI-OpenMP

III. MPI-Pthreads

IV. Modularidad

i. MPI-OpenMP

ii. MPI-Pthreads

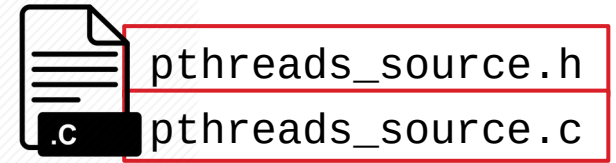
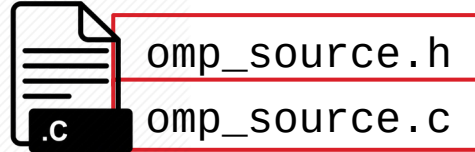
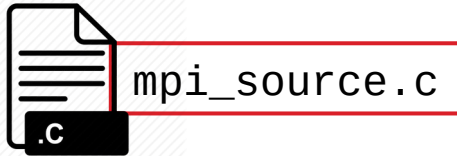
iii. Restricciones



Modularidad

16

- Una forma de organizar el código híbrido es tener el código de cada herramienta por separado (facilidad para depurar, mayor modularidad y portabilidad). 🤗



- En ocasiones no contamos con el código fuente.
- Esto nos obliga a trabajar a nivel de compilador, tenemos dos opciones:
 - 1) Utilizar un mismo compilador para compilar todos los módulos juntos
 - 2) Utilizar compiladores diferentes para cada módulo generando código objeto (.o) y luego linkear el código objeto con alguno de los compiladores.



Agenda

17

I. Modelo de programación híbrido: Introducción

II. MPI-OpenMP

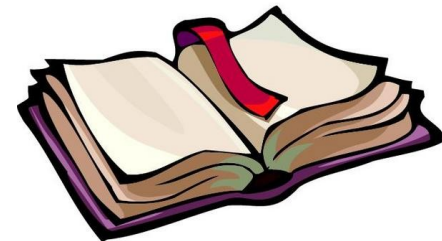
III. MPI-Pthreads

IV. Modularidad

i. MPI-OpenMP

ii. MPI-Pthreads

iii. Restricciones



Modularidad

MPI-OpenMP – Código ejemplo

18

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

static void mpi_function(int rank){
    //Comunicacion con otros procesos MPI
    omp_function(rank);
    //Comunicacion con otros procesos MPI
}

int main(int argc, char* argv[]){
    int rank;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);

    mpi_function(rank);

    MPI_Finalize();
    return(0);
}
```

mpi_source.c

```
#ifndef __OMP_FUNCTIONS__
#define __OMP_FUNCTIONS__

extern void omp_function(int rank);

#endif
```

omp_source.h

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

void omp_function(int rank){
    #pragma omp parallel
    {
        printf("Hello MPI rank %d thread %d\n",rank,omp_get_thread_num());
    }
}
```

omp_source.c

Modularidad

MPI-OpenMP – Compilación

19



- **Opción 1:** Compilación unificada con el compilador de MPI:

```
mpicc -fopenmp -o mpi_omp mpi_source.c omp_source.c
```



- **Opción 2:** Compilación por módulo en 3 pasos:

- **Compilar sólo MPI:**

```
mpicc -c mpi_source.c -o mpi_source.o
```

- **Compilar sólo OpenMP:**

```
gcc -fopenmp -c omp_source.c -o omp_source.o
```

- **Linkear con el compilador de MPI:**

```
mpicc -lgomp -o mpi_omp mpi_source.o omp_source.o
```

I. Modelo de programación híbrido: Introducción

II. MPI-OpenMP

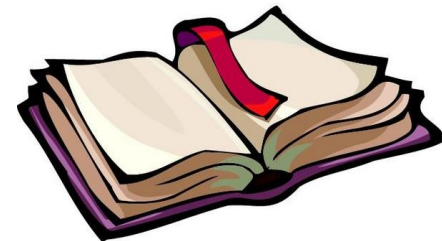
III. MPI-Pthreads

IV. Modularidad

i. MPI-OpenMP

ii. MPI-Pthreads

iii. Restricciones



Modularidad

MPI-Pthreads – Código ejemplo

21

```
#ifndef __PTHREADS_FUNCTIONS__  
#define __PTHREADS_FUNCTIONS__
```

pthread_source.h

```
extern void pthreads_function(int miRank);
```

```
#endif
```

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

mpi_source.c

```
static void mpi_function(int rank){  
    //Comunicacion con otros procesos MPI  
    pthreads_function(rank);  
    //Comunicacion con otros procesos MPI  
}
```

```
int main(int argc, char* argv[]){  
    int rank;  
    MPI_Init(&argc, &argv);  
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);  
  
    mpi_function(rank);  
  
    MPI_Finalize();  
    return(0);  
}
```

```
#include <pthread.h>
```

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

```
int rank;  
int T;
```

pthread_source.c

```
void* funcion(void *arg){  
    int id = *(int*)arg;  
    printf("Hello MPI rank %d thread %d\n",rank,id);  
    pthread_exit(NULL);  
}
```

```
void pthreads_function(int miRank){  
    rank=miRank;    T=get_nprocs(); pthread_t hilos[T];  
    int hilos_ids[T]; int id;  
  
    for(id=0;id<T;id++){  
        hilos_ids[id] = id;  
        pthread_create(&hilos[id], NULL, &funcion,(void*)&hilos_ids[id]);  
    }  
    for(id=0;id<T;id++)  
        pthread_join(hilos[id],NULL);  
}
```

Modularidad

MPI-Pthreads – Compilación

22



- **Opción 1:** Compilación unificada con el compilador de MPI:

```
mpicc -lpthread -o mpi_pthreads mpi_source.c pthreads_source.c
```



- **Opción 2:** Compilación por módulo en 3 pasos:

- **Compilar sólo MPI:**

```
mpicc -c mpi_source.c -o mpi_source.o
```

- **Compilar sólo Pthreads:**

```
gcc -lpthread -c pthreads_source.c -o pthreads_source.o
```

- **Linkear con el compilador de MPI:**

```
mpicc -o mpi_pthreads mpi_source.o pthreads_source.o
```

I. Modelo de programación híbrido: Introducción

II. MPI-OpenMP

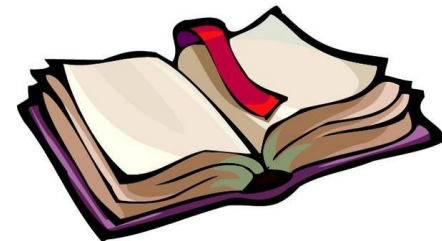
III. MPI-Pthreads

IV. Modularidad

i. MPI-OpenMP

ii. MPI-Pthreads


iii. Restricciones



Modularidad

Restricciones

24

- Siempre que se pueda, deberíamos tener el código de cada herramienta por separado.
 - Sin embargo, las características de algunos algoritmos dificultan esta tarea:
 - Algoritmos que tienen etapas donde los procesos se comunican en cada una de ellas:
- 
- MPI-Pthreads el programador puede verse obligado a tener varios llamados a `pthread_create`. (Posible solución: crear un pool de threads y pasarlo como parámetro)
 - MPI-Pthreads hilos que comunican

