

Sistemas Distribuidos y Paralelos

Ingeniería en Computación



MultiGPUs y modelos híbridos

Universidad Nacional de La Plata



Facultad de Informática



Agenda

2

I. Introducción a los modelos híbridos

II. Múltiples GPUs

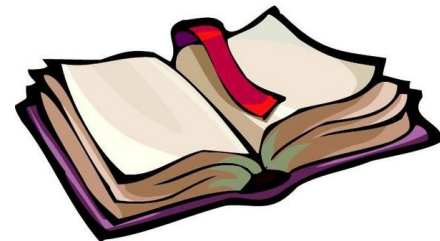
III. Modelos híbridos

- i. CUDA + OpenMP
- ii. CUDA + Pthreads
- iii. CUDA + MPI
- iv. CUDA + OpenMP + MPI / CUDA + Pthreads + MPI

IV. Modularidad

V. Distribución de carga

VI. Mas allá de los Sistemas Paralelos



Agenda

3

I. Introducción a los modelos híbridos

II. Múltiples GPUs

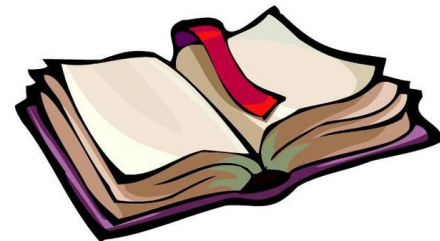
III. Modelos híbridos

- i. CUDA + OpenMP
- ii. CUDA + Pthreads
- iii. CUDA + MPI
- iv. CUDA + OpenMP + MPI / CUDA + Pthreads + MPI

IV. Modularidad

V. Distribución de carga

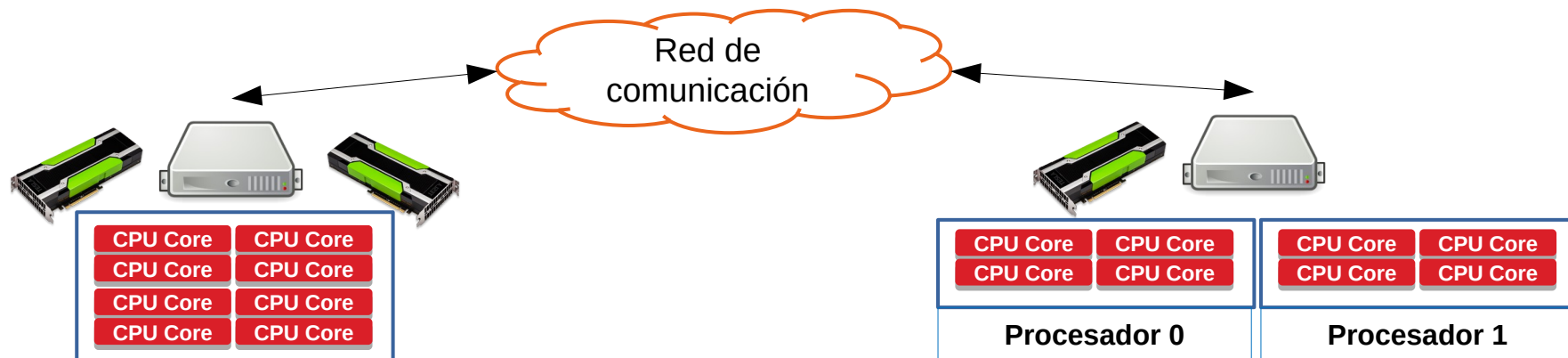
VI. Mas allá de los Sistemas Paralelos



Introducción a los modelos híbridos

4

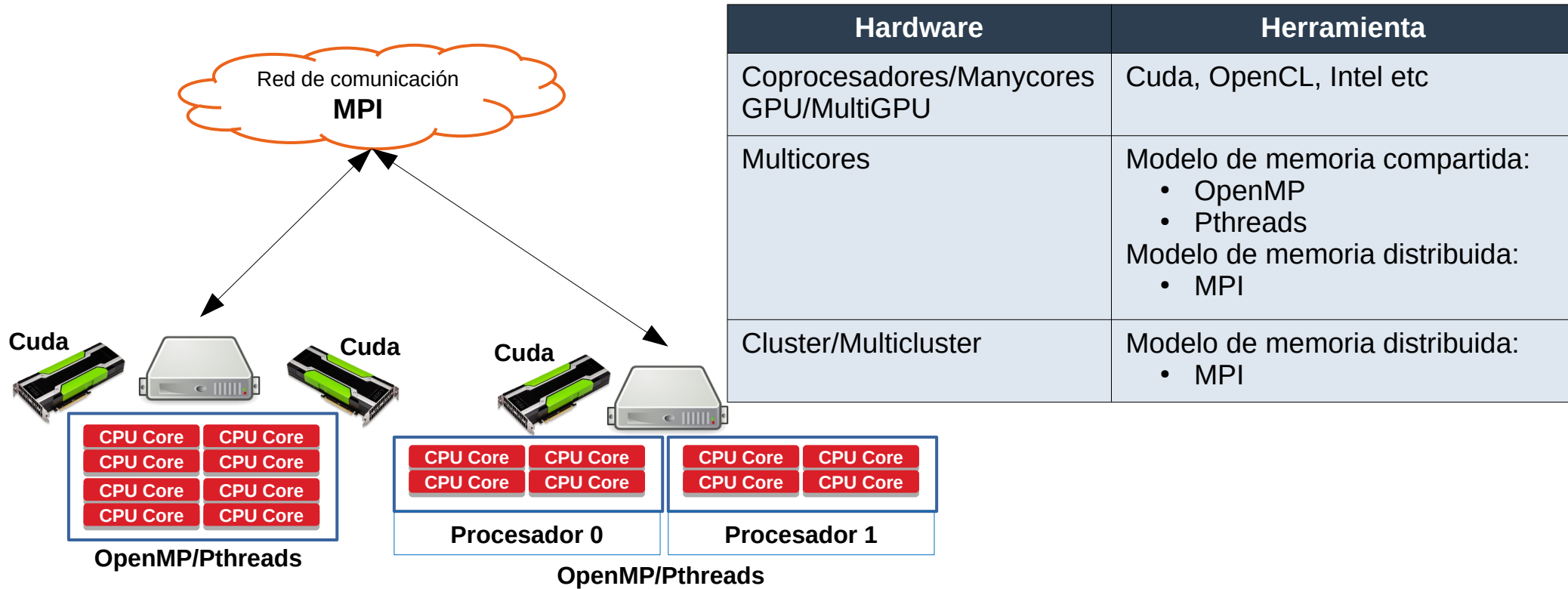
- En la actualidad existe una diversa cantidad de hardware muy potente:
 - Procesadores tradicionales (Intel, AMD, ARM etc) con múltiples cores
 - Manycores/GPUs
 - Servidores de alto rendimiento pueden incluir hasta 4 GPUs
- Las redes de comunicación permiten unir toda esta potencia de cómputo en Cluster/Multicluster formando un modelo híbrido.



Introducción a los modelos híbridos

5

- Podemos considerar 3 niveles de hardware. Cada nivel requiere una herramienta específica para programar la arquitectura subyacente.



Introducción a los modelos híbridos

6

- Programar sobre un modelo híbrido conlleva complejidad que se presenta al integrar el hardware y el software.
- Existen algunas herramientas que nos dan mayor nivel de abstracción y facilitan la programación:
 - OpenACC
 - Intel OpenAPI
- Pero si queremos tener control absoluto debemos programar sin abstraernos.
- Para un correcto desarrollo se deben tener en cuenta distintos aspectos:
 - Compilación sobre arquitecturas diferentes
 - Compilación en módulos separados para cada arquitectura
 - Enlazar los módulos de cada arquitectura
 - Integrar diversas opciones de compilación

Agenda

7

I. Introducción a los modelos híbridos

II. Múltiples GPUs

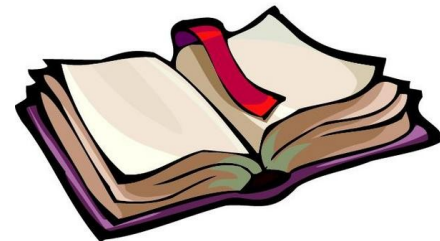
III. Modelos híbridos

- i. CUDA + OpenMP
- ii. CUDA + Pthreads
- iii. CUDA + MPI
- iv. CUDA + OpenMP + MPI / CUDA + Pthreads + MPI

IV. Modularidad

V. Distribución de carga

VI. Mas allá de los Sistemas Paralelos



Manycores - Múltiples GPUs

8

- En una máquina con más de una GPU debemos especificar sobre que GPU se están realizando las acciones. Para esto se utiliza la sentencia:

```
cudaSetDevice(int id_device);
```

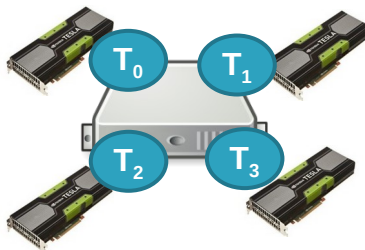
- `id_device`: número entero entre 0 y el número de GPUs - 1 instaladas.

```
...  
cudaSetDevice(0); //Todo lo que se haga de acá en más: sobre GPU0  
cudaMalloc(...);  
cudaMemcpy(...);  
Kernel_0<<<..., ..., ...>>>();  
cudaSetDevice(1); //Todo lo que se haga de acá en más: sobre GPU1  
cudaMalloc(...);  
cudaMemcpy(...);  
Kernel_1<<<..., ..., ...>>>();  
...
```


Manycores - Múltiples GPUs

9

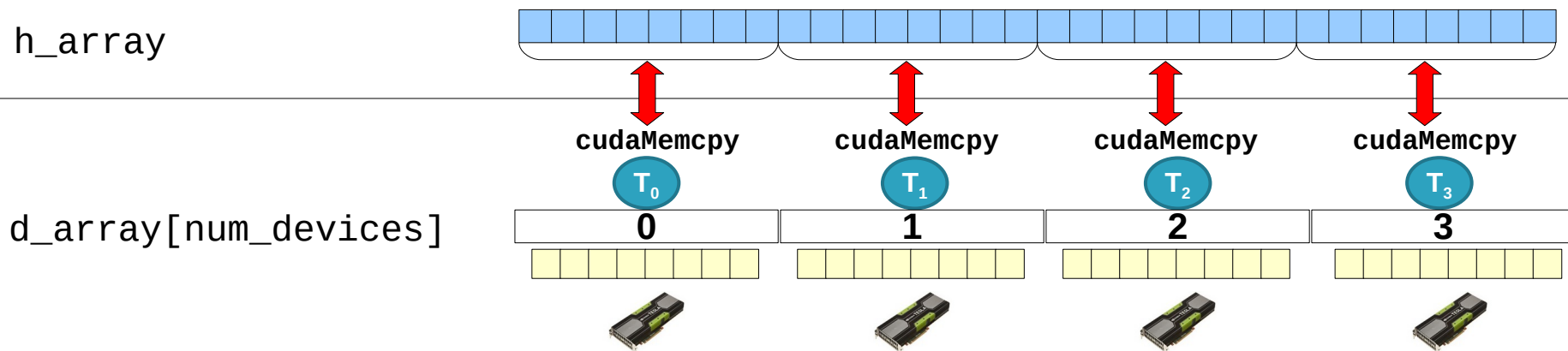
- Para realizar el cómputo correcto de un conjunto de datos, utilizando varias GPUs, se suele crear un hilo por cada GPU disponible.
- El identificador de cada hilo debería coincidir con el `id_device`.
- Cada hilo se encarga de gestionar todo lo relacionado con su propia GPU.



Manycores - Múltiples GPUs

10

- Ejemplo: Se quiere realizar cómputo sobre N elementos almacenados en un array.
 - El cómputo de una porción del array es independiente del resto.
 - Suponemos `num_devices` GPUs iguales, los datos se distribuyen proporcionalmente entre ellas (porción = $N/\text{num_devices}$)
 - Las porciones del array en GPU conviene gestionarlas mediante un array indexado por `id_device` (o identificador del hilo).
 - Cada hilo debe calcular la posición del array a copiar en su porción en GPU.



Agenda

11

I. Introducción a los modelos híbridos

II. Múltiples GPUs

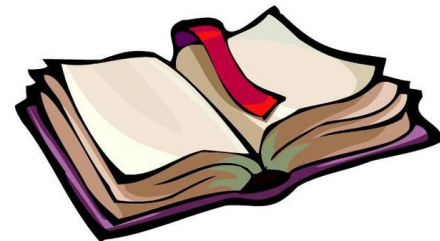
III. Modelos híbridos

- i. CUDA + OpenMP
- ii. CUDA + Pthreads
- iii. CUDA + MPI
- iv. CUDA + OpenMP + MPI / CUDA + Pthreads + MPI

IV. Modularidad

V. Distribución de carga

VI. Mas allá de los Sistemas Paralelos



Agenda

12

I. Introducción a los modelos híbridos

II. Múltiples GPUs

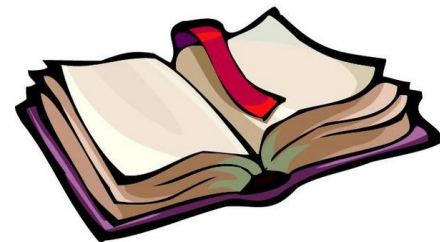
III. Modelos híbridos

- i. CUDA + OpenMP
- ii. CUDA + Pthreads
- iii. CUDA + MPI
- iv. CUDA + OpenMP + MPI / CUDA + Pthreads + MPI

IV. Modularidad

V. Distribución de carga

VI. Mas allá de los Sistemas Paralelos



- Las GPUs pueden combinarse con otras arquitecturas. Las combinaciones de herramientas en cada caso son las siguientes:
 - MultiGPU + Multicores:
 - CUDA + OpenMP
 - CUDA + Pthreads
 - MultiGPU + Cluster:
 - CUDA + MPI
 - MultiGPU + Multicores + Cluster:
 - CUDA + OpenMP + MPI
 - CUDA + Pthreads + MPI

Agenda

14

I. Introducción a los modelos híbridos

II. Múltiples GPUs

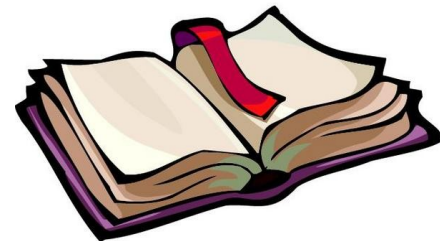
III. Modelos híbridos

- i. CUDA + OpenMP
- ii. CUDA + Pthreads
- iii. CUDA + MPI
- iv. CUDA + OpenMP + MPI / CUDA + Pthreads + MPI

IV. Modularidad

V. Distribución de carga

VI. Mas allá de los Sistemas Paralelos



CUDA + OpenMP

15

- Se suele asignar un hilo a cada GPU para su gestión y el resto de los hilos a cómputo:



- El compilador CUDA está basado en GCC. Por lo tanto, pueden utilizarse las opciones de GCC. Un programa CUDA + OpenMP se compila:

```
nvcc -Xcompiler -fopenmp programa.cu
```

I. Introducción a los modelos híbridos

II. Múltiples GPUs

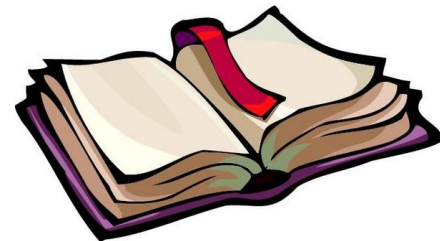
III. Modelos híbridos

- i. CUDA + OpenMP
- ii. CUDA + Pthreads
- iii. CUDA + MPI
- iv. CUDA + OpenMP + MPI / CUDA + Pthreads + MPI

IV. Modularidad

V. Distribución de carga

VI. Mas allá de los Sistemas Paralelos



CUDA + Pthreads

17

- Con Pthreads se sigue la misma estrategia que OpenMP.



- Un programa CUDA-Pthreads se compila:

```
nvcc -pthread programa.cu
```

Agenda

18

I. Introducción a los modelos híbridos

II. Múltiples GPUs

III. Modelos híbridos

- i. CUDA + OpenMP
- ii. CUDA + Pthreads
- iii. CUDA + MPI
- iv. CUDA + OpenMP + MPI / CUDA + Pthreads + MPI

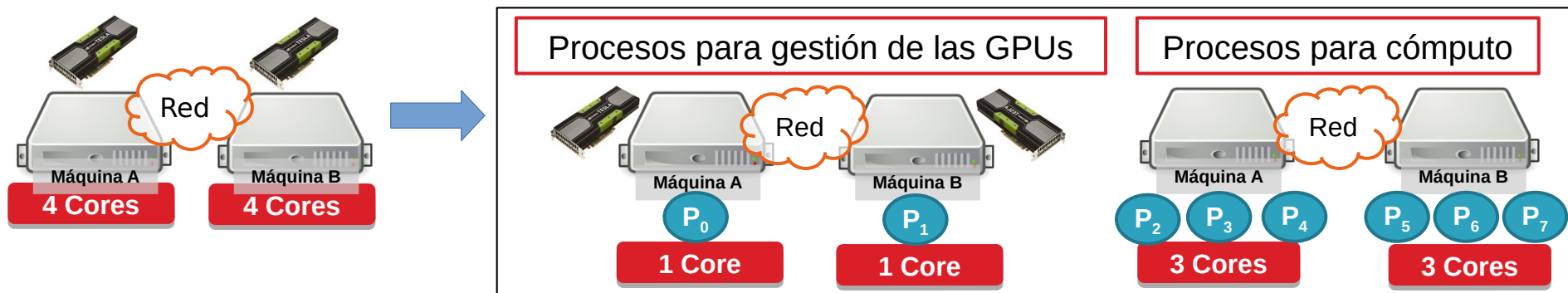
IV. Modularidad

V. Distribución de carga

VI. Mas allá de los Sistemas Paralelos



- Siguiendo la idea de memoria compartida, se asigna un proceso de gestión por GPU y el resto a cómputo:



- Puede compilarse con `nvcc` o `mpicc`. La forma más simple es hacerlo con `nvcc`. Un programa CUDA-MPI se compila:

```
nvcc -I$MPI_PATH/include -L$MPI_PATH/lib -lmpi programa.cu
```

- Luego, se ejecuta con `mpirun`

I. Introducción a los modelos híbridos

II. Múltiples GPUs

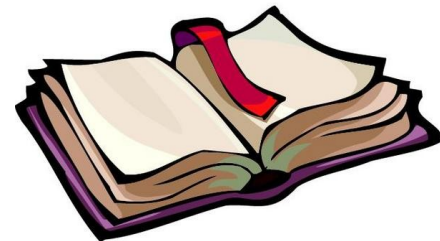
III. Modelos híbridos

- i. CUDA + OpenMP
- ii. CUDA + Pthreads
- iii. CUDA + MPI
- iv. CUDA + OpenMP + MPI / CUDA + Pthreads + MPI

IV. Modularidad

V. Distribución de carga

VI. Mas allá de los Sistemas Paralelos

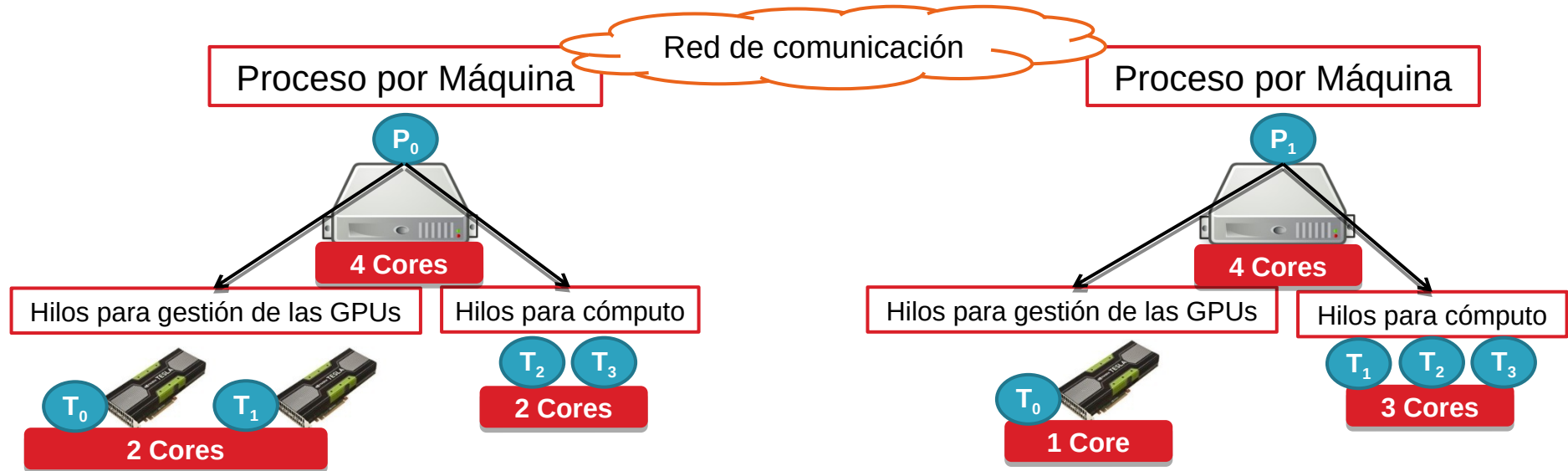


CUDA + OpenMP + MPI

CUDA + Pthreads + MPI

21

- Se asigna un proceso por máquina del cluster y este proceso crea los hilos.
- Luego se sigue la estrategia de memoria compartida (se crea y asigna un hilo de gestión por cada GPU y el resto a cómputo):



CUDA + OpenMP + MPI

CUDA + Pthreads + MPI

22

- La forma más sencilla de compilar es utilizando **nvcc**:

- Para compilar Cuda-OpenMP-MPI:

```
nvcc -Xcompiler -fopenmp -I$MPI_PATH/include -L$MPI_PATH/lib -lmpi programa.cu
```

- Para compilar Cuda-Pthreads-MPI:

```
nvcc -I$MPI_PATH/include -L$MPI_PATH/lib -lmpi programa.cu
```

- Luego, se ejecuta con **mpirun**

I. Introducción a los modelos híbridos

II. Múltiples GPUs

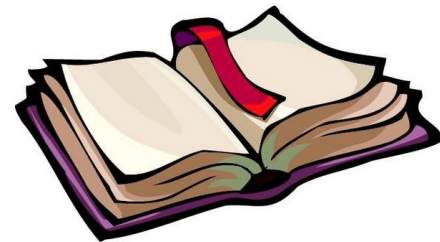
III. Modelos híbridos

- i. CUDA + OpenMP
- ii. CUDA + Pthreads
- iii. CUDA + MPI
- iv. CUDA + OpenMP + MPI / CUDA + Pthreads + MPI

IV. Modularidad

V. Distribución de carga

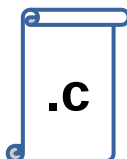
VI. Mas allá de los Sistemas Paralelos



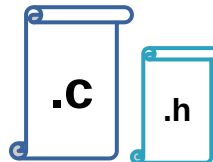
Modularidad

24

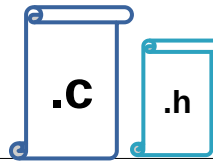
- Lo anterior asume un único archivo **.cu** que integra todas las herramientas.
- Para lograr mayor **modularidad** es conveniente ubicar el código de cada herramienta en archivos separados.



.c
mpi_source.c



.c **.h**
omp_source.c
omp_source.h
pthread_source.c
pthread_source.h



.c **.h**
cuda_source.cu
cuda_source.h

Modularidad

CUDA + OpenMP

25

- Compilación con un **único compilador**:

```
nvcc -Xcompiler -fopenmp omp_source.c cuda_source.cu
```

- **Compilación separada**:

- **Compilar** cada herramienta con su compilador:

- **OpenMP**: `gcc -fopenmp -c omp_source.c -o omp_source.o`

- **CUDA**: `nvcc -c cuda_source.cu -o cuda_source.o`

- **Linkear** los archivos de salida (.o) con un compilador:

```
gcc cuda_source.o omp_source.o -L$CUDA_PATH/lib64 -lgomp -lcudart
```

Modularidad

CUDA + Pthreads

26

- Compilación con un **único compilador**:

```
nvcc pthreads_source.c cuda_source.cu
```

- **Compilación separada**:

- **Compilar** cada herramienta con su compilador:

- **Pthreads**: `gcc -pthread -c pthreads_source.c -o pthreads_source.o`

- **CUDA**: `nvcc -c cuda_source.cu -o cuda_source.o`

- **Linkear** los archivos de salida (.o) con un compilador:

```
gcc cuda_source.o pthreads_source.o -L$CUDA_PATH/lib64 -pthread -lcudart
```

Modularidad

CUDA + OpenMP + MPI

27

- Compilación con un **único compilador**:

```
nvcc -Xcompiler -fopenmp -I$MPI_PATH/include -L$MPI_PATH/lib -lmpi mpi_source.c omp_source.c  
cuda_source.cu
```

- **Compilación separada**:

- **Compilar** cada herramienta con su compilador:

- **MPI:** `mpicc -c mpi_source.c -o mpi_source.o`
- **OpenMP:** `gcc -fopenmp -c omp_source.c -o omp_source.o`
- **CUDA:** `nvcc -c cuda_source.cu -o cuda_source.o`

- **Linkear** los archivos de salida (.o) con un compilador:

```
mpicc mpi_source.o omp_source.o cuda_source.o -L$CUDA_PATH/lib64 -lcudart -lgomp
```

Modularidad

CUDA + Pthreads + MPI

28

- Compilación con un **único compilador**:

```
nvcc -I$MPI_PATH/include -L$MPI_PATH/lib -lmpi mpi_source.c pthreads_source.c  
cuda_source.cu
```

- **Compilación separada**:

- **Compilar** cada herramienta con su compilador:

- **MPI:** `mpicc -c mpi_source.c -o mpi_source.o`
- **Pthreads:** `gcc -pthread -c pthreads_source.c -o pthreads_source.o`
- **CUDA:** `nvcc -c cuda_source.cu -o cuda_source.o`

- **Linkar** los archivos de salida (.o) con un compilador:

```
mpicc mpi_source.o pthreads_source.o cuda_source.o -L$CUDA_PATH/lib64 -lcudart
```

I. Introducción a los modelos híbridos

II. Múltiples GPUs

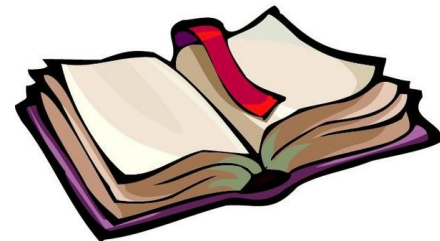
III. Modelos híbridos

- i. CUDA + OpenMP
- ii. CUDA + Pthreads
- iii. CUDA + MPI
- iv. CUDA + OpenMP + MPI / CUDA + Pthreads + MPI

IV. Modularidad

V. Distribución de carga

VI. Mas allá de los Sistemas Paralelos



Distribución de carga

30

- Una arquitectura híbrida es una arquitectura heterogénea: Las unidades de procesamiento son diferentes, con distintas potencias de cómputo.
- Cuando tenemos un problema con determinada carga de trabajo es necesario que cada unidad de procesamiento reciba parte de la carga de trabajo proporcional a su potencia de cómputo.
- Se debería balancear la carga de trabajo de manera que todas las unidades de procesamiento terminen el trabajo “casi al mismo tiempo” y ese tiempo debería ser el mínimo (el que maximice el rendimiento).
- Se debe tener en cuenta que el problema de distribución de carga balanceada es **NP-Completo**.
- Una solución a este problema no puede automatizarse. Sin embargo, existen varias estrategias para permitir un balance de carga aceptable.

Agenda

31

I. Introducción a los modelos híbridos

II. Múltiples GPUs

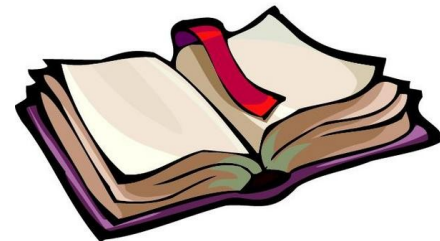
III. Modelos híbridos

- i. CUDA + OpenMP
- ii. CUDA + Pthreads
- iii. CUDA + MPI
- iv. CUDA + OpenMP + MPI / CUDA + Pthreads + MPI

IV. Modularidad

V. Distribución de carga

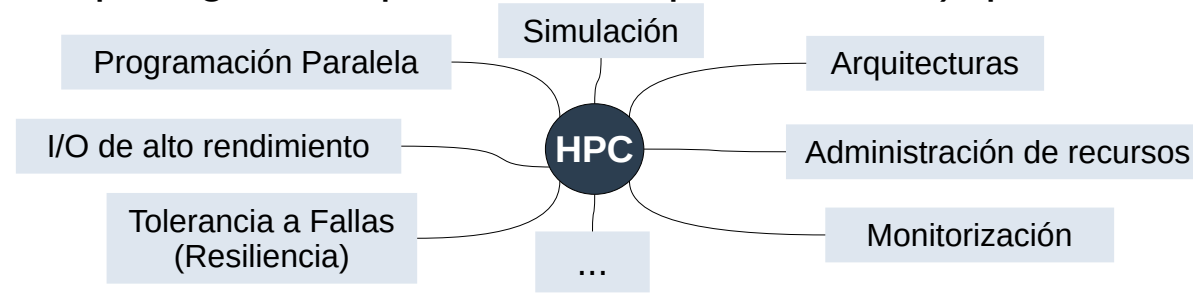
VI. Mas allá de los Sistemas Paralelos



Mas allá de los sistemas paralelos

32

- Los sistemas paralelos son parte de un concepto mas amplio conocido como HPC (High Performance Computing – Cómputo de altas prestaciones) que abarca distintas áreas de estudio:



- La tendencia es incorporar nuevos tipos de unidades de procesamiento especializadas, NPUs, TensorCores, QPU etc. No todas son aptas para resolver todo tipo de problema, pero puede aprovecharse la especialización y ventajas de cada una para resolver partes específicas de forma eficiente.



- Lo importante es la elección adecuada del conjunto de arquitecturas que resuelvan de forma mas eficiente posible un problema específico.

