Entornos Masivamente Paralelos

M. Graciela Molina

m.graciela.molina@gmail.com

WTPC
workshop en técnicas de programación científica

Qué es HPC: High Performance Computing

- HPC = Computación de Alto Desempeño = Eficiencia
- HPC: Me importa qué tan rápido obtenga una respuesta
- HPC: Alta productividad
- HPC: Software viejo + Hardware nuevo

WTPC workshop en técnicas de programación científica

Qué es HPC: High Performance Computing

¿Dónde?

- Smartphone
- Desktop/laptop
- Clúster
- Supercomputadora
- En la nube

WTPC workshop en técnicas de programación científica

Qué es HPC: High Performance Computing

¿Cuándo?

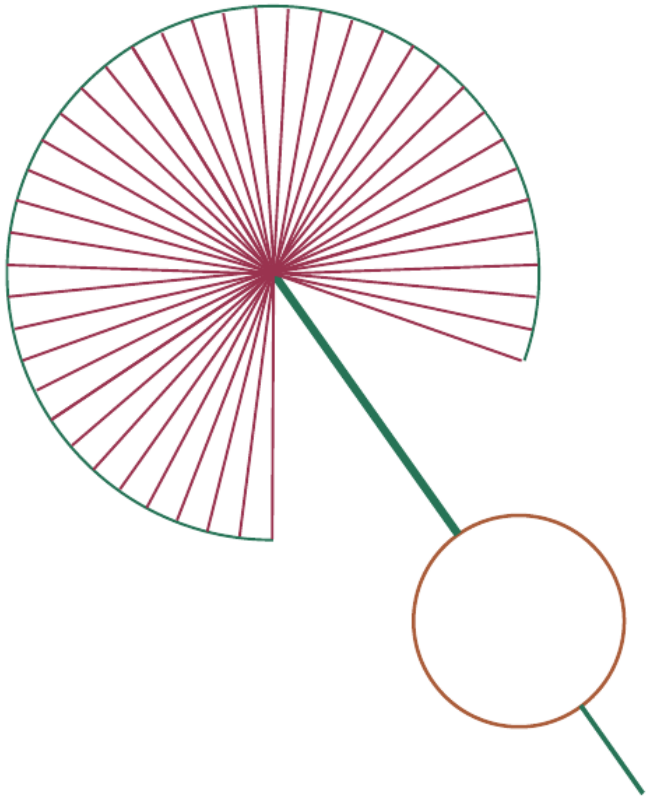
- Aprovechar el hardware que tenemos
- Decidir qué hardware comprar
- Obtener resultados de simulaciones extremas

¿Qué resignamos?

Interfaces amigables, software reutilizable y portable ...

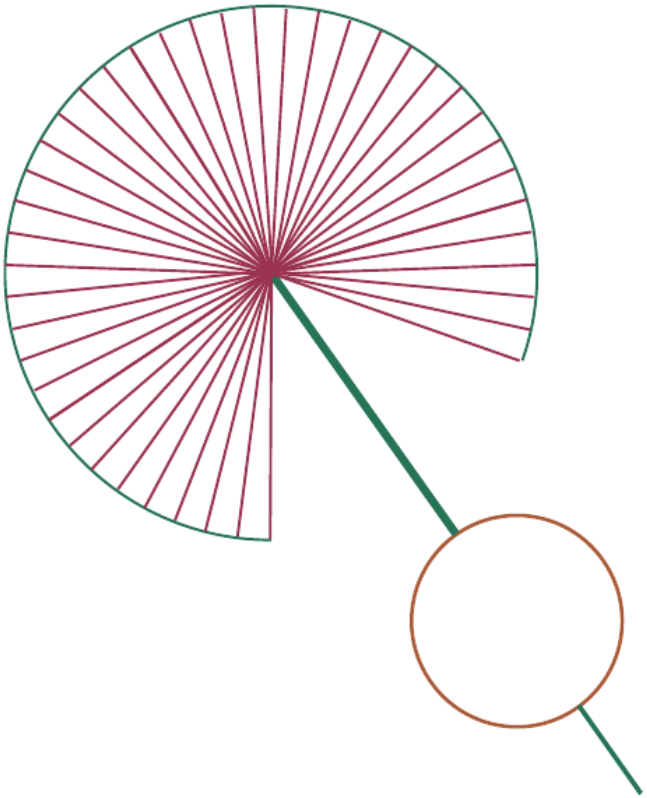
WTPC
workshop en técnicas de programación científica

HPC en una PC de escritorio



Procesadores (sin tiempo que perder)
Conexiones (clave)
SO elige cómo se conecta

HPC en un Clúster

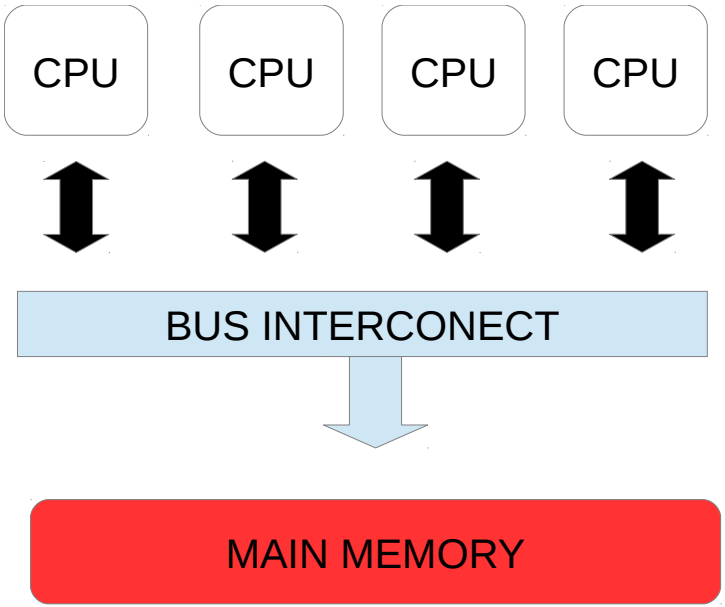


Computadores (sin tiempo que perder)
Conexiones (clave)
Nodo maestro elige cómo se conecta

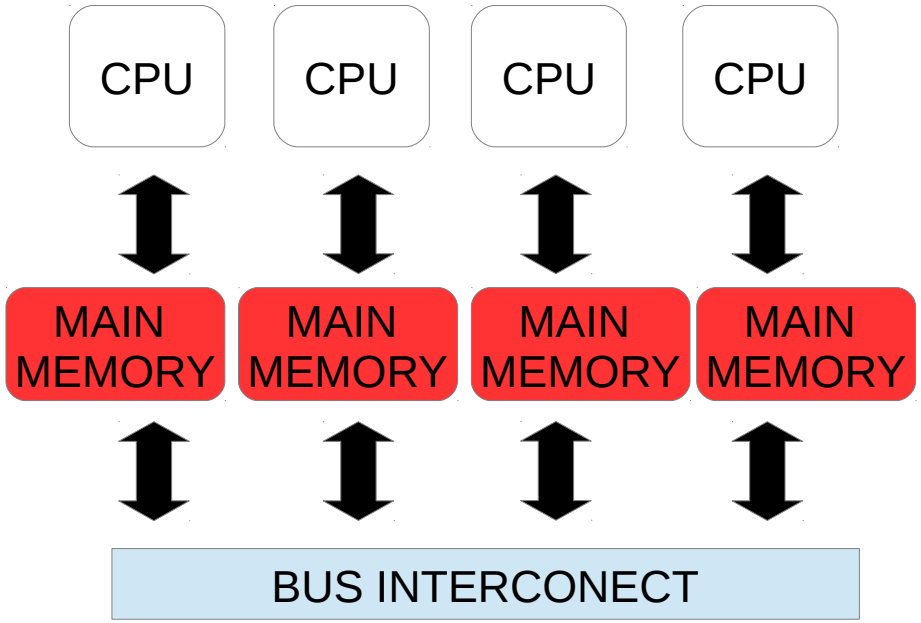
WTPC workshop en técnicas de programación científica

Arquitectura de un clúster

Symmetric MultProcessors



NonUniform Memory Access



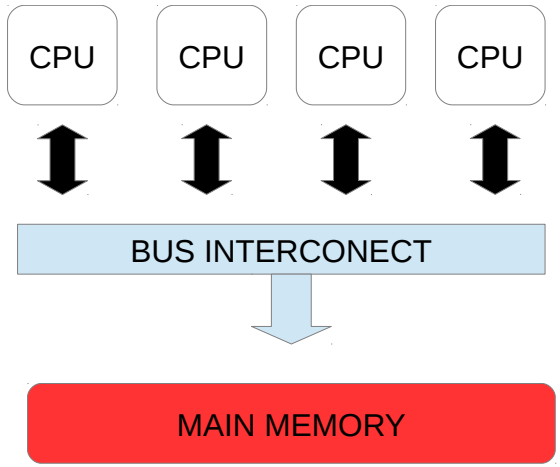
Memoria compartida

Ventajas

- Fácil para el programador
- Compartir datos es más rápido y directo

Desventajas

- Escalabilidad pobre
- Sincronización a cargo del programador
- Más difícil y costoso diseñar y producir máquinas con memoria compartida a medida que se aumenta la cantidad de procesadores

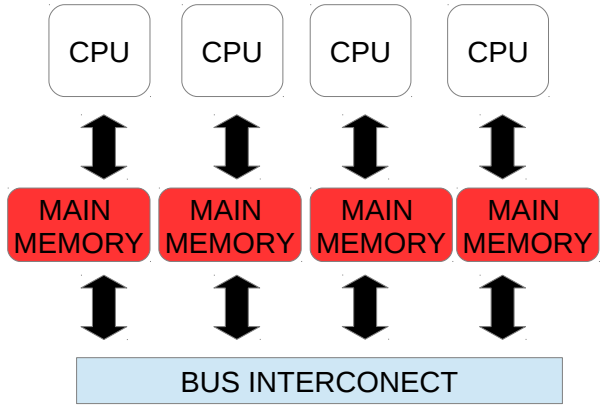


WTPC workshop en técnicas de programación científica

Memoria distribuida

Ventajas

- Memoria escala con el numero de procesadores
- Cada procesador accede rápidamente a su propia memoria local sin interferencias y sin overhead
- Obtener hardware off-the-shelf con una *performance* muy razonable



Desventajas

- Comunicación de datos entre procesos a cargo del programador
- Complicado adaptar código existente
- Tiempo de acceso a los datos no es uniforme (y varía mucho!)

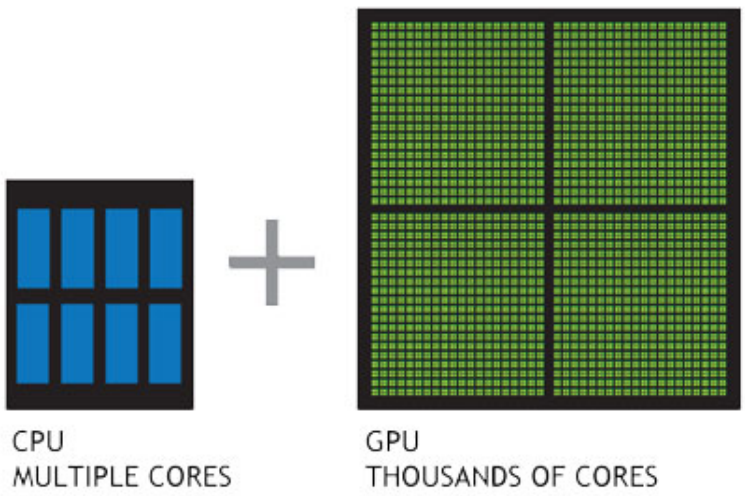
Arquitectura de un clúster

¿Cómo impacta en el diseño del software?

- Paralelismo masivo
- Complejidad creciente
- Menos eficiencia para software viejo
- Poca previsibilidad

¡Hay que pensar en el hardware al momento de codificar!

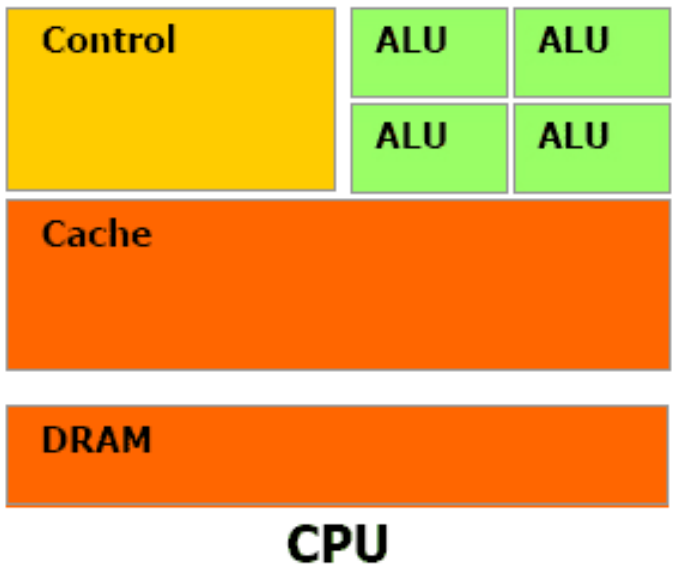
Aceleradores

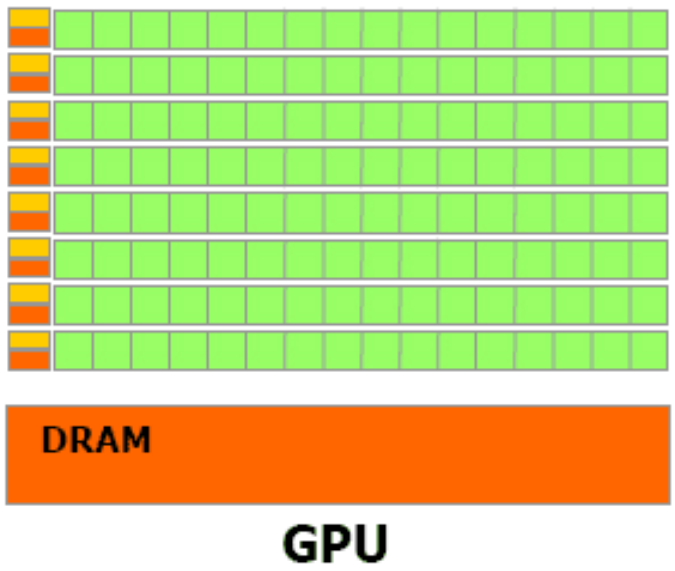


GPU (*graphics processing unit*)

Aceleradores

Esquematicámente:





Aceleradores

La idea general es:

menos ctrl **menos caché** **más ALUs**

↓

Paralelismo masivo

(para alimentar tantas ALUs)

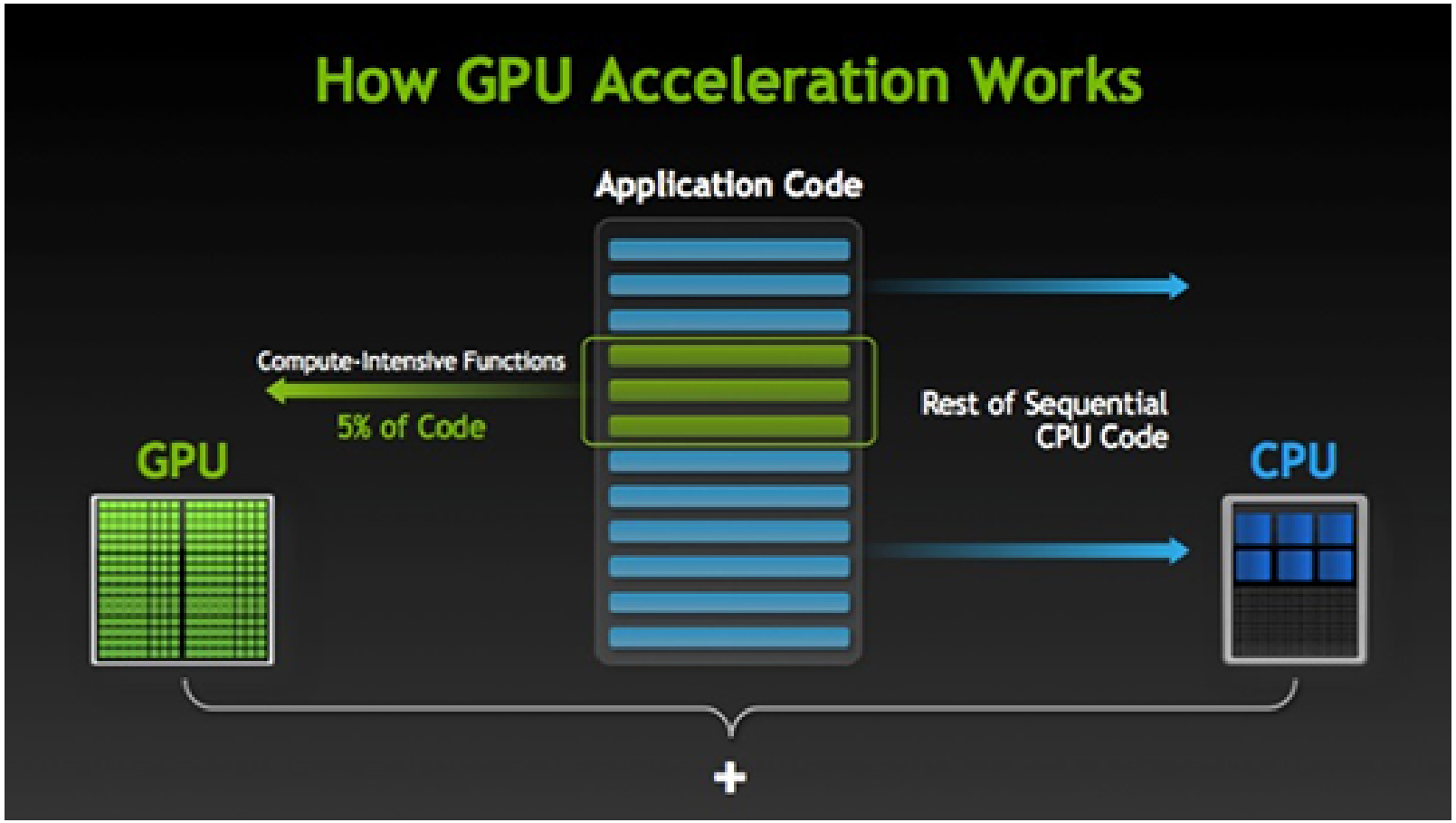
↓

Paralelismo de datos

(suficiente como para ocultar la latencia)

Aceleradores

Esquematicámente:



Aceleradores

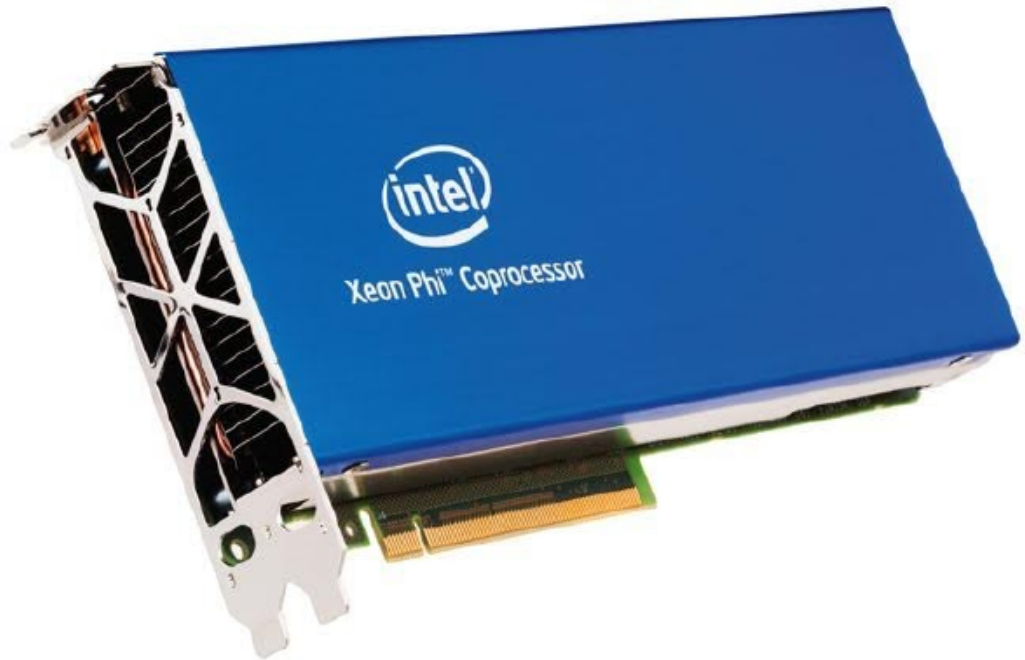
¿Por qué acelera?

- Diseño muy escalable
- Mucho ancho de banda
- Muchos procesadores de baja frecuencia
- Ideal para el procesado masivo de datos

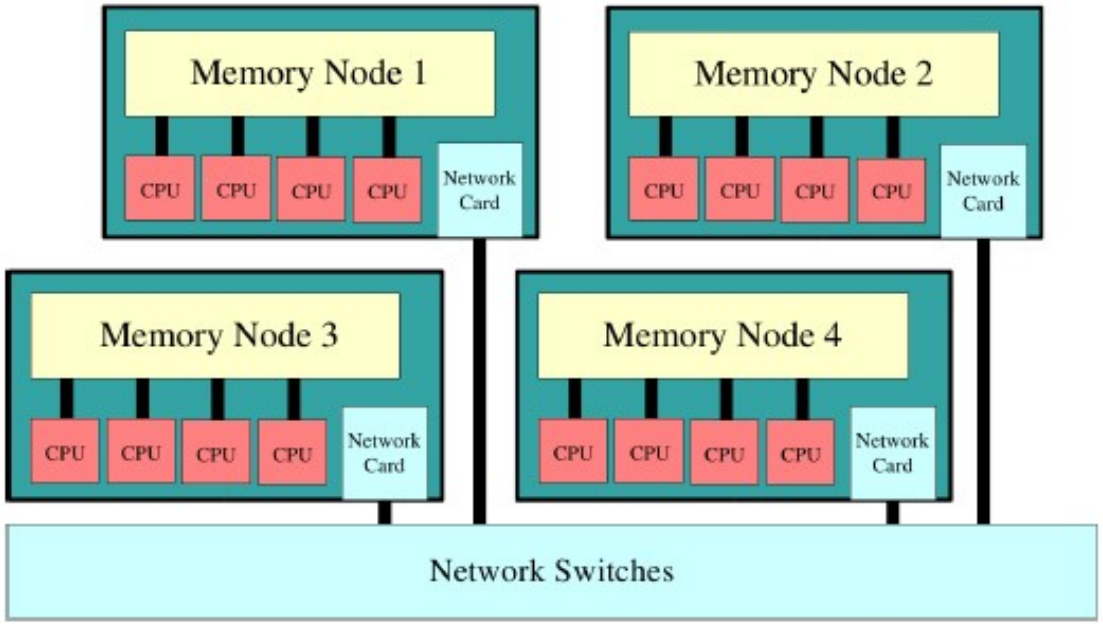
No siempre acelera

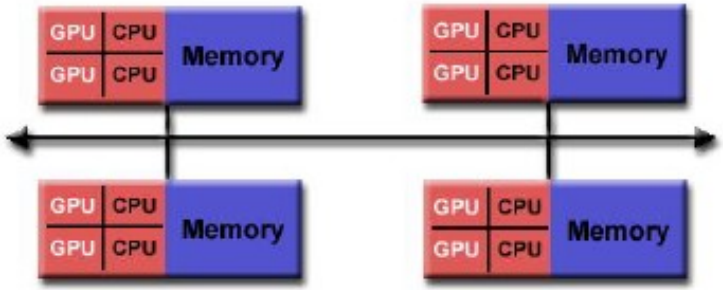
- Hay que pasarle la información a la placa
- Difícil sincronizar los procesadores
- Ejecución en serie MUY lenta

Aceleradores



Arquitecturas híbridas

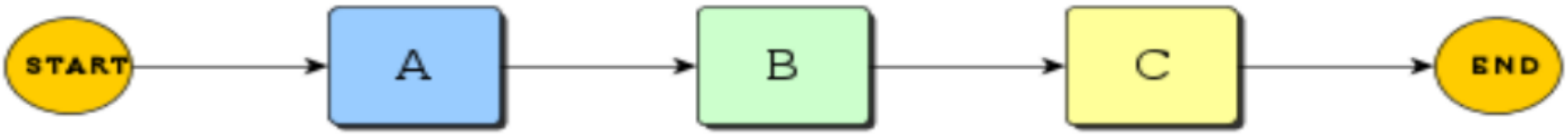




HPC en una Supercomputadora

¿Cómo programamos para estas arquitecturas?

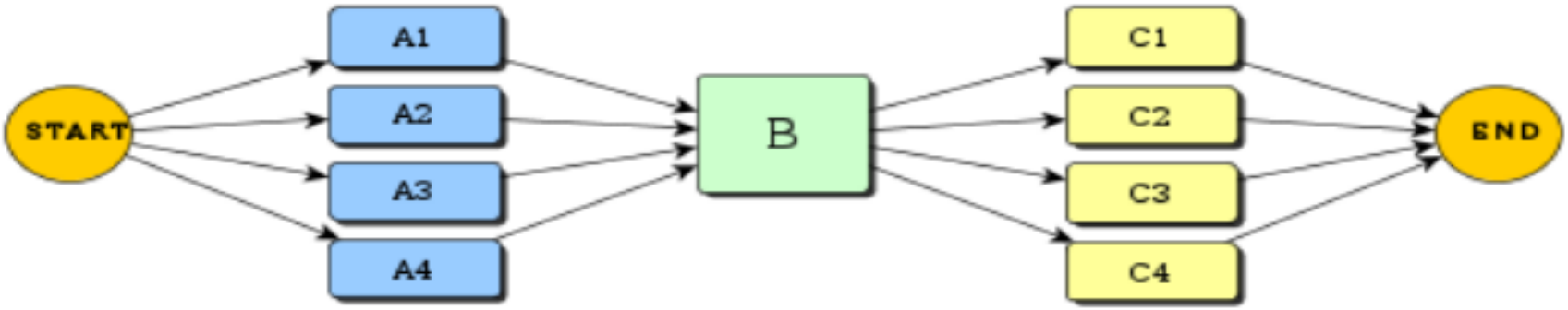
Procesamiento secuencial



Ya utilicé técnicas de optimización y aún necesito mejorar la *performance* de mi código.
Y si agrego un core... ¿cuánto mejora?

¿Cómo programamos para estas arquitecturas?

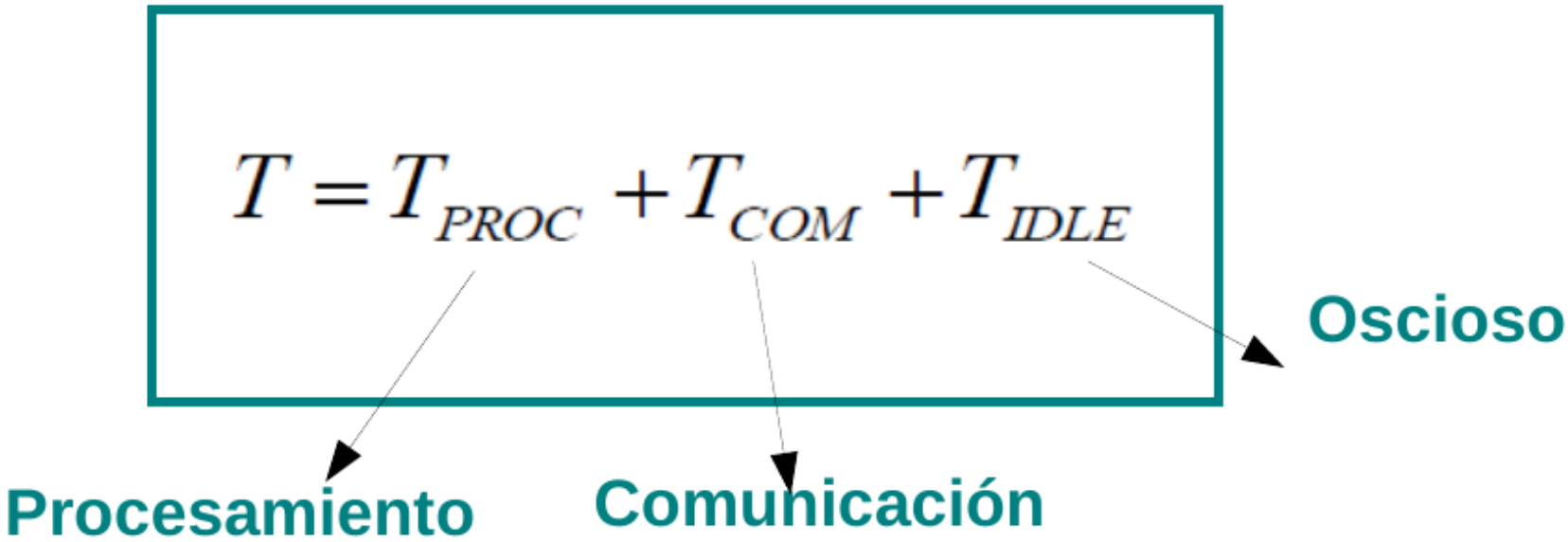
Procesamiento paralelo



Mi problema se puede subdividir en problemas independientes o es necesario ejecutar un gran número de veces una misma simulación

Rendimiento de aplicaciones paralelas

De que depende el tiempo de ejecución de un programa paralelo?



Rendimiento de aplicaciones paralelas

$$T_{PROC}$$

Depende de:

- Complejidad y dimensión del problema
- Número de tareas utilizadas
- Características de los elementos de procesamiento (hardware, heterogeneidad, no dedicación)

Rendimiento de aplicaciones paralelas

$$T_{COM}$$

Depende de la localidad de procesos y datos (comunicación inter e intra-procesador, canal de comunicación)

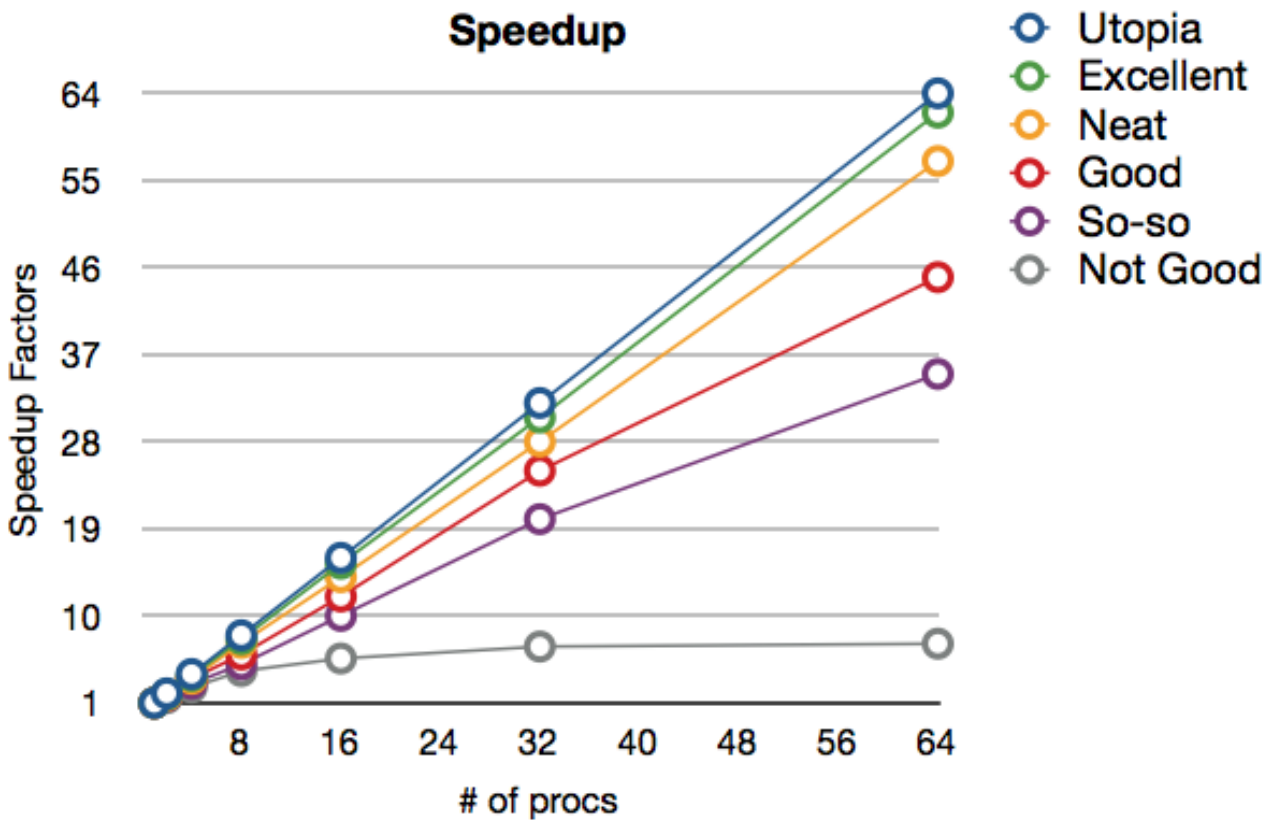
$$T_{IDLE}$$

Debido al no determinismo en la ejecución, minimizarlo es un objetivo de diseño.

WTPC workshop en técnicas de programación científica

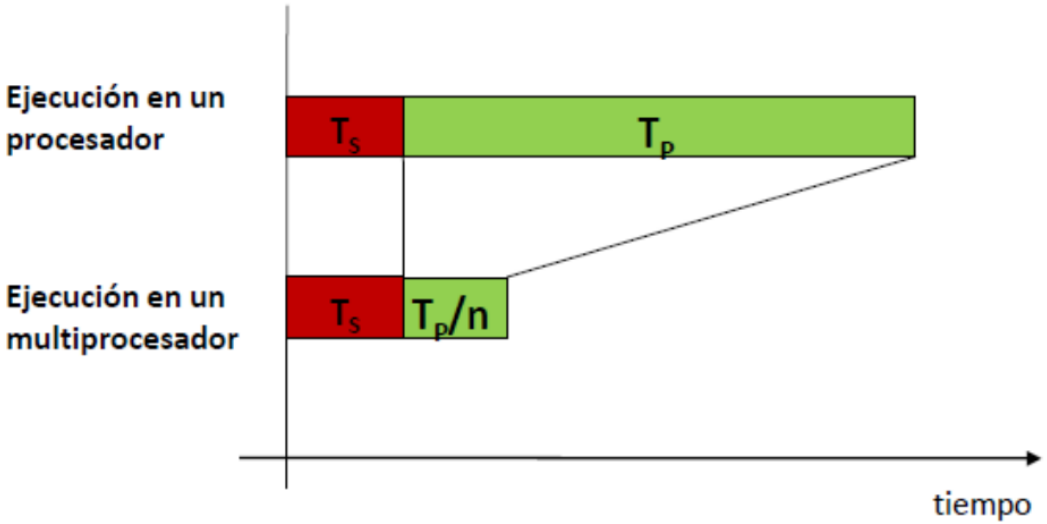
Speed Up

$$S_N = T_1 / T_N$$

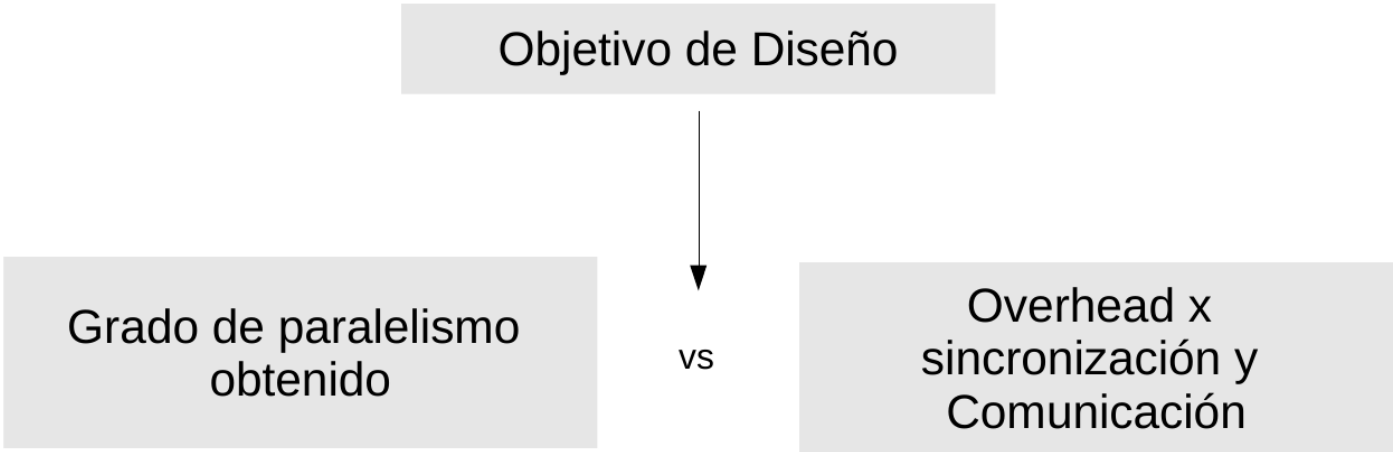


Rendimiento de aplicaciones paralelas

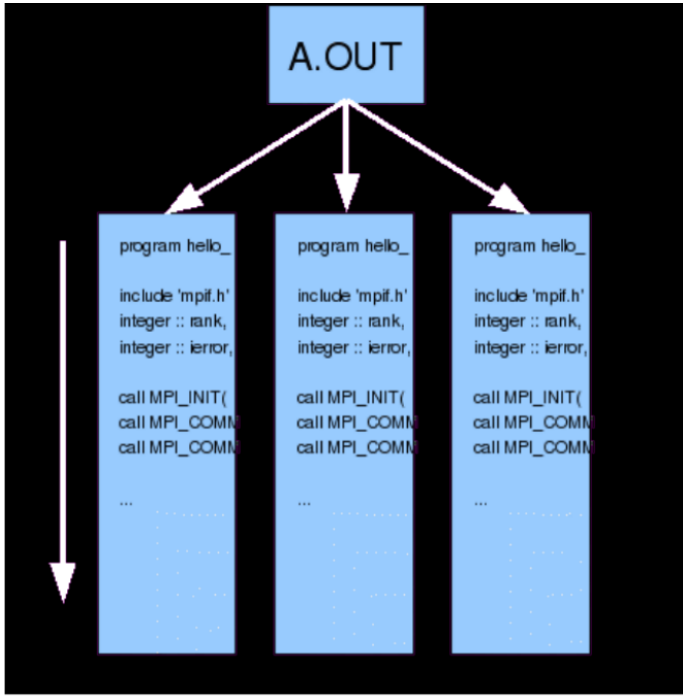
Ley de Amdhal



Rendimiento de aplicaciones paralelas



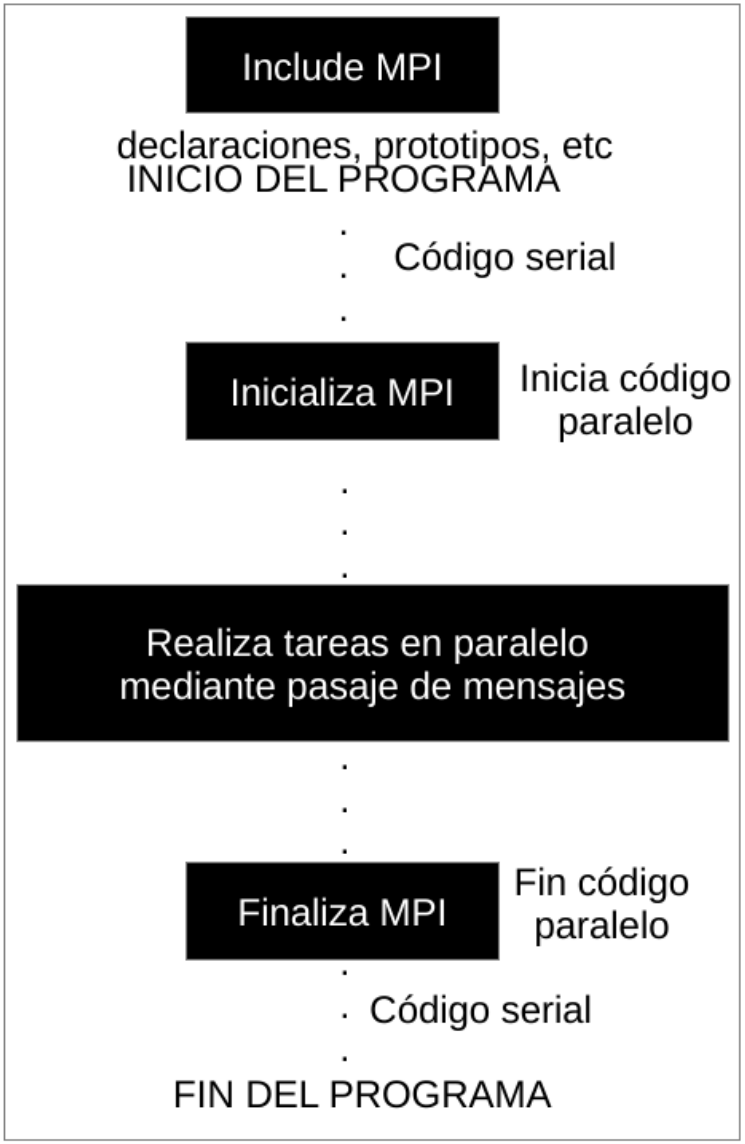
MPI: Message Passing Interface



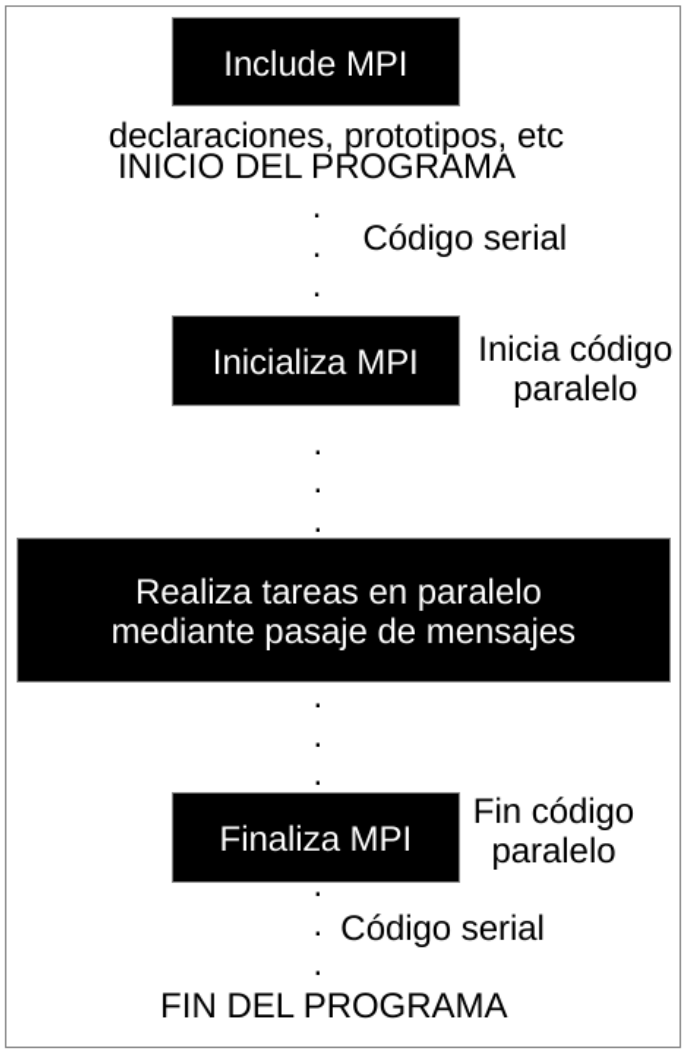
- Estándar portable
- Plataforma objetivo: memoria distribuida
- Paralelismo explícito
- Número de tareas definido en tiempo de ejecución
- Comunicación = Pasaje de mensaje entre los procesadores

WTPC Workshop en técnicas de programación científica

MPI: Message Passing Interface



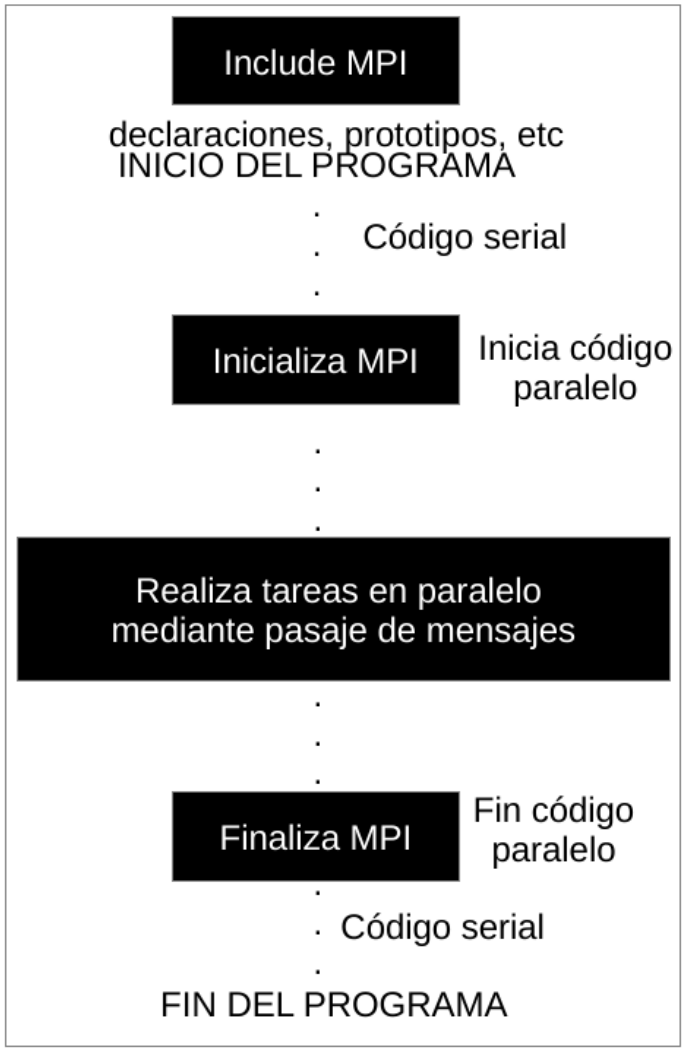
MPI: Message Passing Interface



C:
#include <mpi.h>

Fortran:
include 'mpif.h'

MPI: Message Passing Interface



Formato de funciones en MPI

C:
error = MPI_Xxxxx(parameter, ...);

MPI_Xxxxx(parameter, ...);

Fortran:
CALL MPI_XXXXX(parameter, ..., IERROR)

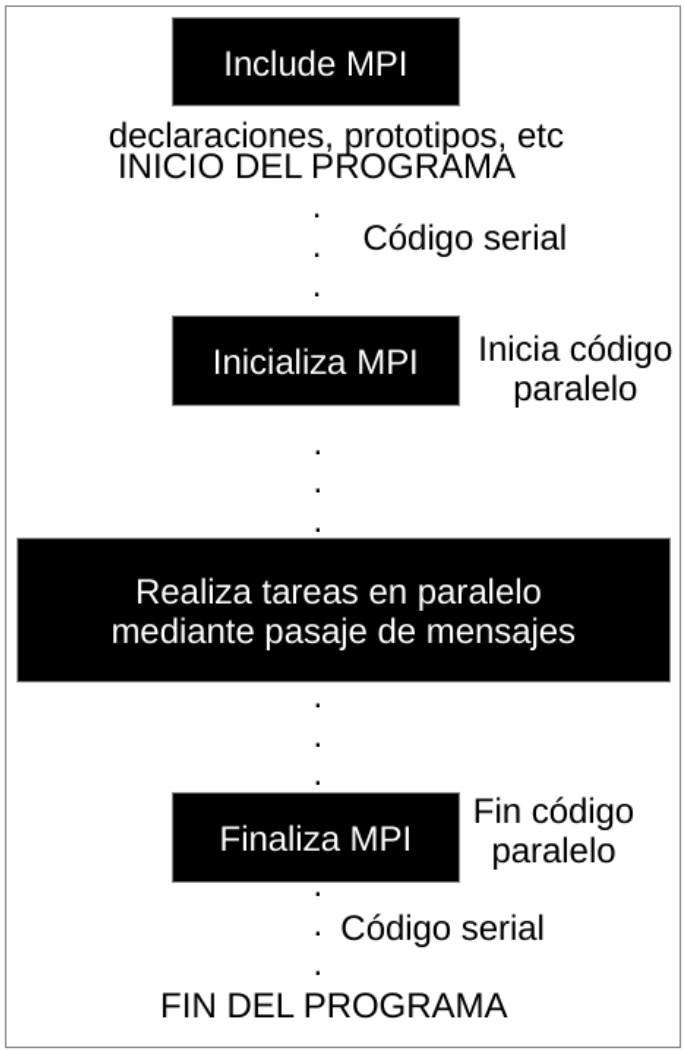
MPI: Message Passing Interface

MPI Datatype	C datatype
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_SHORT	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	

MPI Datatype	Fortran Datatype
MPI_INTEGER	INTEGER
MPI_REAL	REAL
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER(1)
MPI_BYTE	
MPI_PACKED	

WTPC workshop en técnicas de programación científica

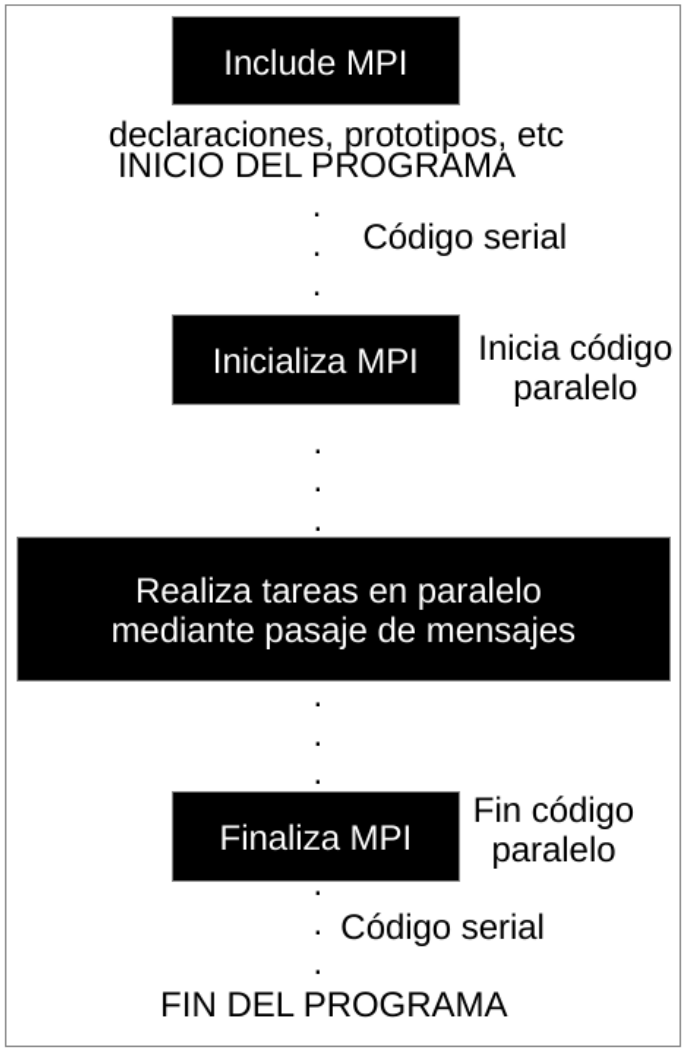
MPI: Message Passing Interface

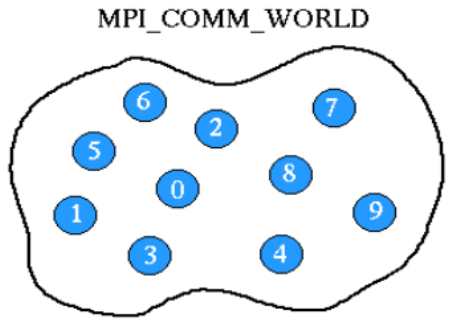


C:
int MPI_Init(int *argc, char ***argv)

Fortran:
MPI_INIT(IERROR)
INTEGER IERROR

MPI: Message Passing Interface





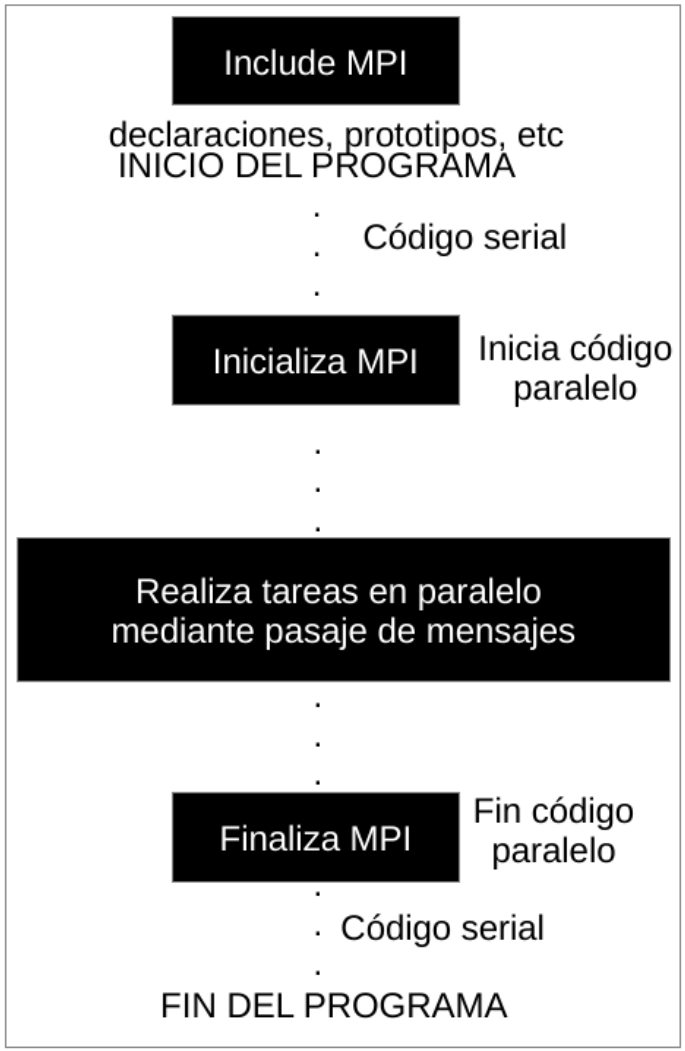
¿Cómo identifico a un proceso dentro de un comunicador?

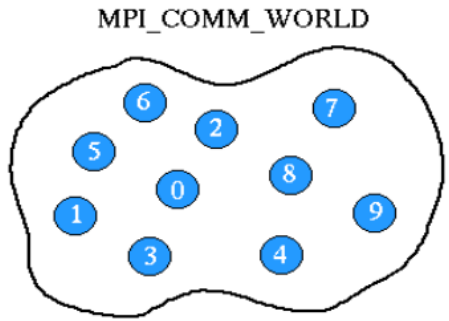
C:
ierr =MPI_Comm_rank(MPI_Comm comm, int *rank)

Fortran:
MPI_COMM_RANK(COMM, RANK, IERROR)
INTEGER COMM, RANK, IERROR

WTPC Workshop en técnicas de programación científica

MPI: Message Passing Interface





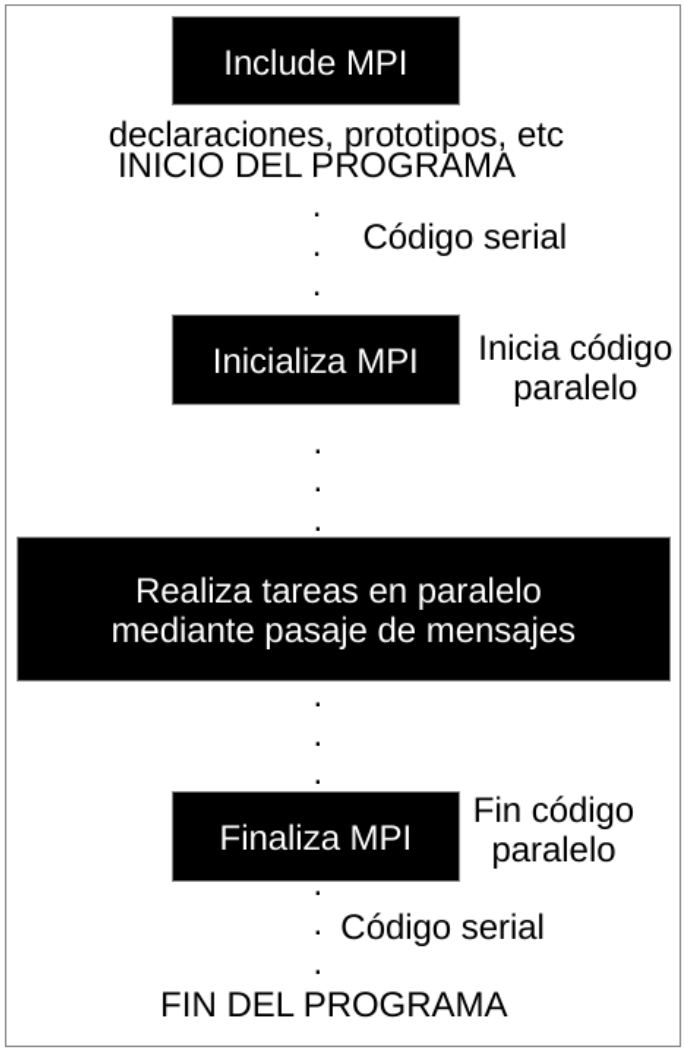
¿Cuantos procesos hay en el comunicador?

C:
ierr=MPI_Comm_size(MPI_Comm comm, int *size)

Fortran:
MPI_COMM_SIZE(COMM, SIZE, IERROR)
INTEGER COMM, SIZE, IERROR

WTPC Workshop en técnicas de programación científica

MPI: Message Passing Interface



C:
int MPI_Finalize()

Fortran:
MPI_FINALIZE(IERROR)
INTEGER IERROR

MPI: Message Passing Interface

Tipos de mensajes:

- Colectivos
- Punto a Punto

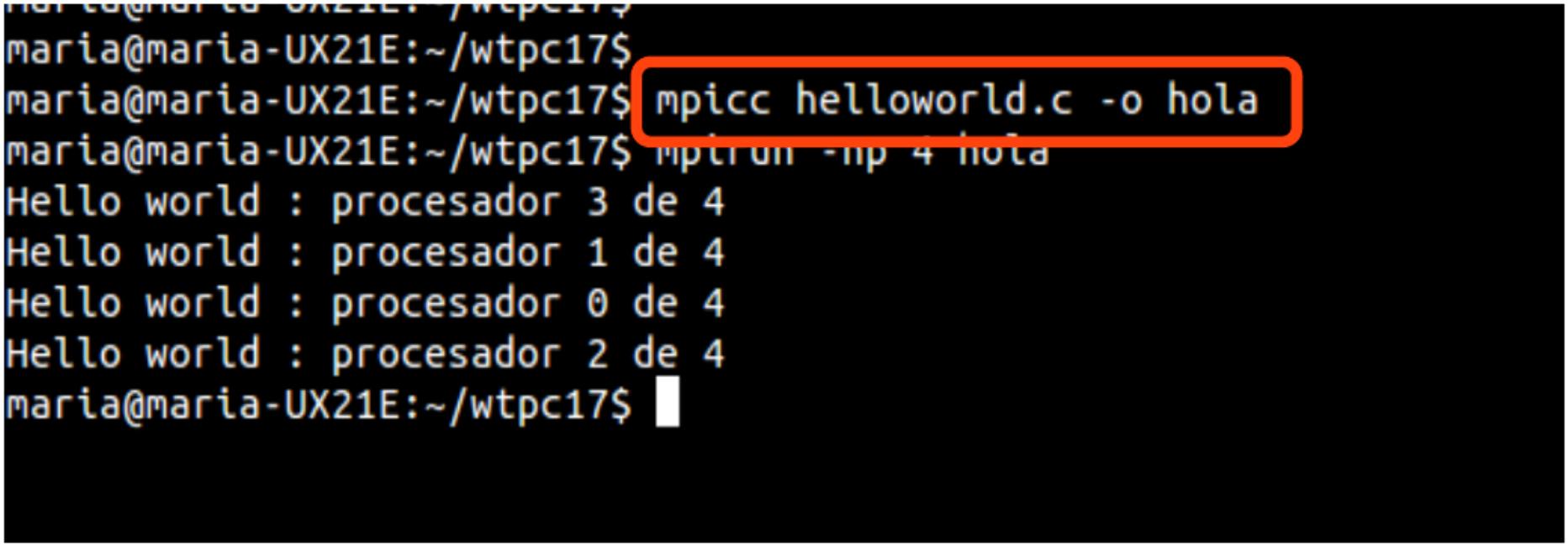
* MPI cheetsheet

MPI: Message Passing Interface

```
1  #include <stdio.h>
2  #include <mpi.h>
3
4  int main (argc, argv)
5       int argc;
6       char *argv[];
7  {
8    int rank, size;
9    double inicio,fin;
10   MPI_Init (&argc, &argv);
11   MPI_Comm_rank (MPI_COMM_WORLD, &rank);
12   MPI_Comm_size (MPI_COMM_WORLD, &size);
13   printf( "Hello world : procesador %d de %d\n", rank, size );
14   MPI_Finalize();
15   return 0;
16 }
```

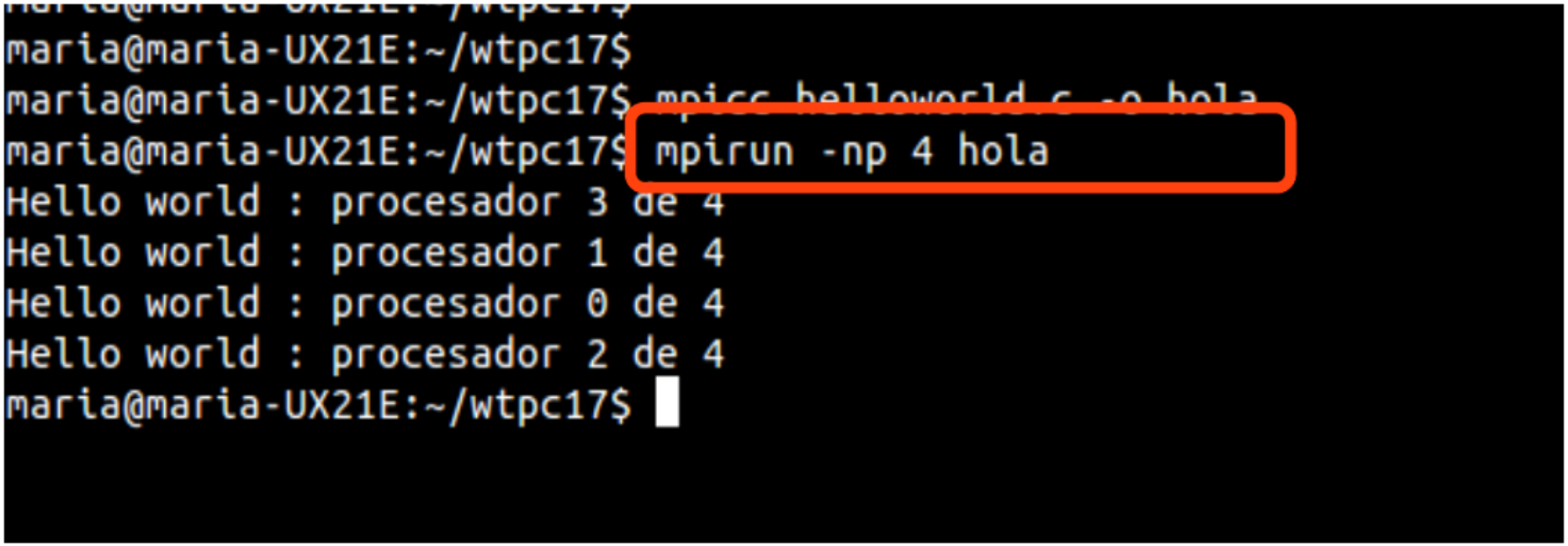
WTPC Workshop en técnicas de programación científica

MPI: Message Passing Interface



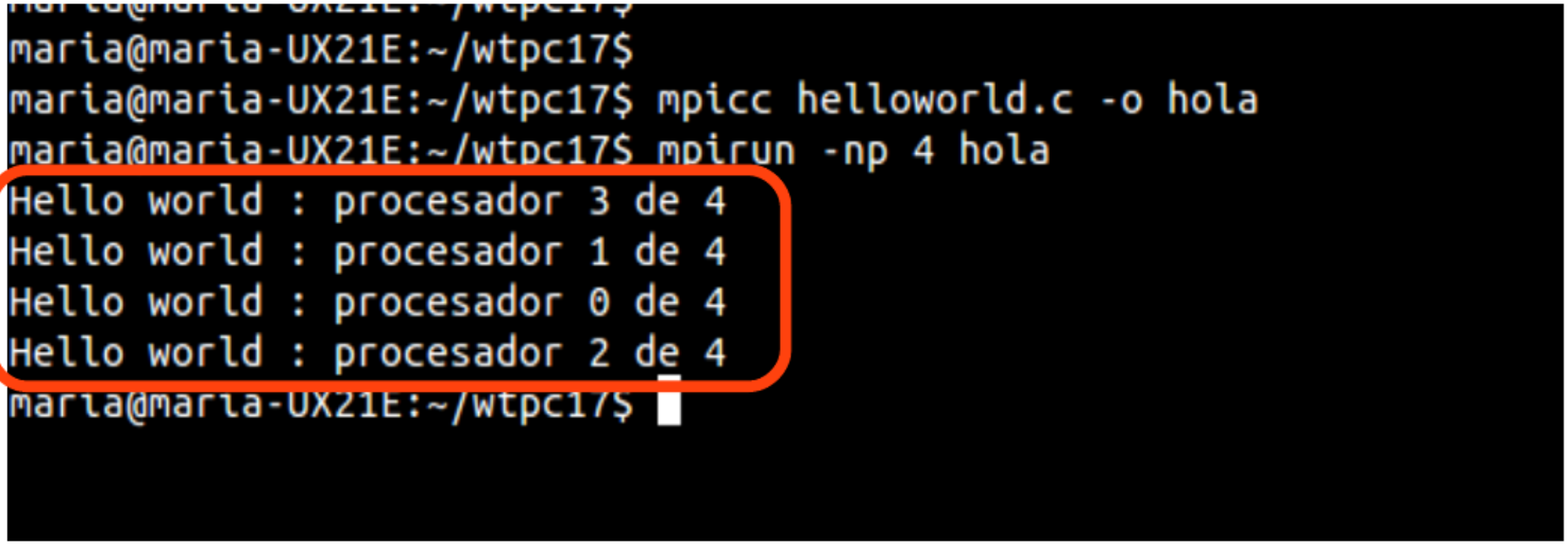
WTPC Workshop en técnicas de programación científica

MPI: Message Passing Interface



WTPC
Workshop en técnicas de programación científica

MPI: Message Passing Interface



WTPC workshop en técnicas de programación científica

Entornos Masivamente Paralelos

M. Graciela Molina

m.graciela.molina@gmail.com

WTPC workshop en técnicas de programación científica