

SISTEMAS PARALELOS

Clase 6 – Programación en plataformas híbridas // Modelo híbrido



Facultad de
INFORMÁTICA



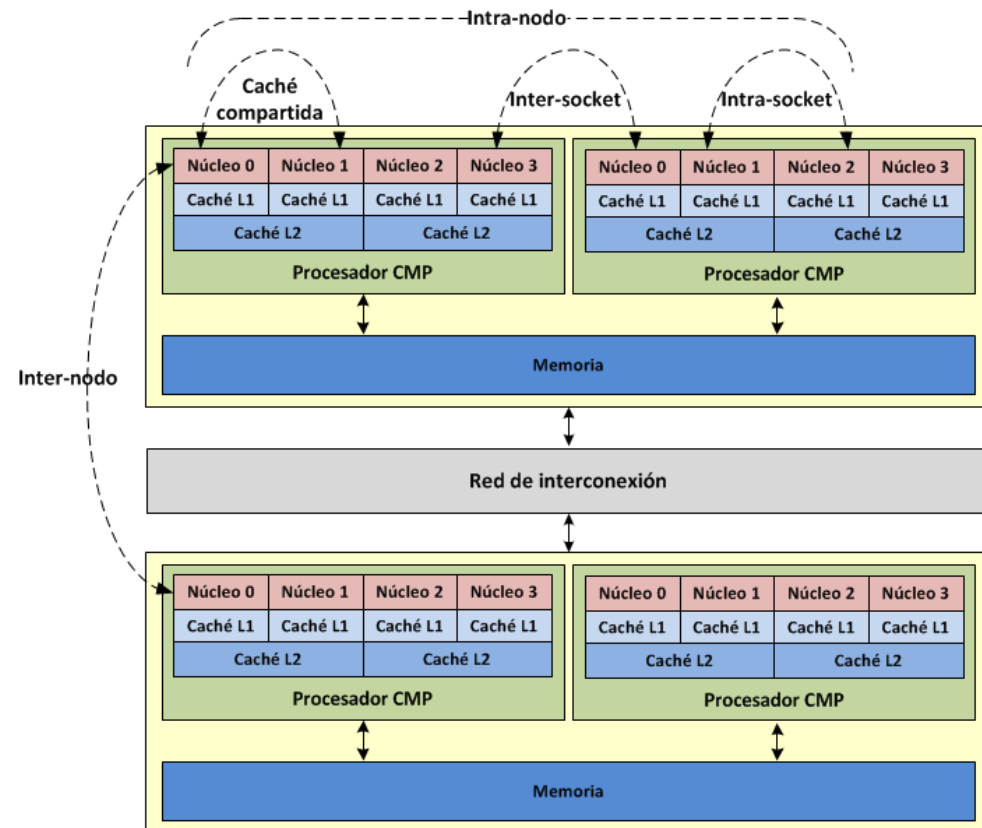
UNIVERSIDAD
NACIONAL
DE LA PLATA

PROGRAMACIÓN EN PLATAFORMAS HÍBRIDAS //

MODELO HÍBRIDO

Fundamentos del modelo híbrido

- La incorporación de procesadores multicore a las arquitecturas de clusters tradicionales dio origen a una nueva arquitectura paralela: *cluster de multicores*
 - arquitecturas híbridas
 - jerárquicas de dos niveles
- Tanto la comunidad científica como la industria se interesaron en investigar modelos de comunicación híbridos → modelos que permitan comunicarse tanto a través del pasaje de mensajes como de la memoria compartida

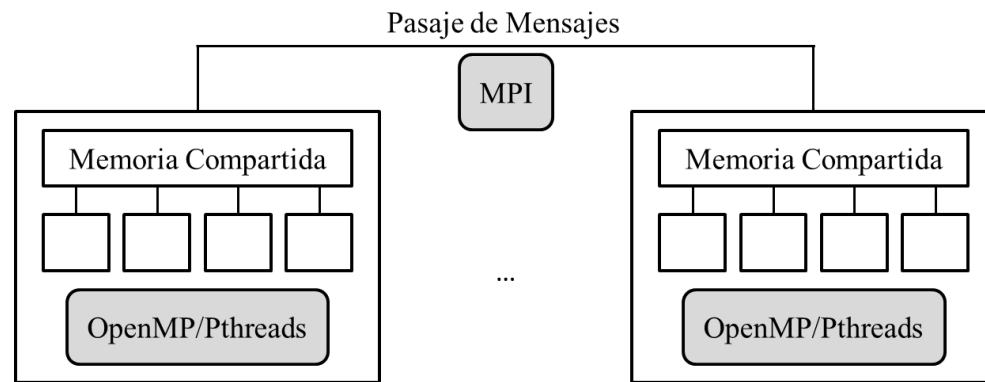


Fundamentos del modelo híbrido

- Los paradigmas de programación tradicionales (pasaje de mensajes y memoria compartida) no se adaptan naturalmente a los clusters de multicores.
- Se espera que un modelo híbrido (combinación de pasaje de mensajes con memoria compartida) explote mejor sus características.
- Idea básica:
 - Las tareas que se encuentran en el mismo nodo se comunican y sincronizan por memoria compartida.
 - Las tareas que se encuentran en diferentes nodos se comunican y sincronizan por pasaje de mensajes.

Fundamentos del modelo híbrido

- La combinación de MPI con OpenMP o Pthreads permite explotar el paralelismo jerárquico inherente a los clusters de multicores o a la aplicación.



- El modelo de programación híbrido puede incrementar el rendimiento y la escalabilidad de una aplicación.
- Sin embargo, esto no ocurre para todos los casos → antes de desarrollar una aplicación paralela empleando el modelo híbrido, debe analizarse si el mismo puede resultar útil o no.

Fundamentos del modelo híbrido

- Razones para utilizar el modelo híbrido:
 - Al aprovechar la memoria compartida dentro de cada nodo:
 - Reducen overhead de las comunicaciones MPI
 - Reducen los requerimientos de memoria de la aplicación.
- Algunas aplicaciones presentan dos niveles de paralelismo:
 - Paralelismo de grano grueso: gran cantidad de cómputo que puede ser realizado en forma independiente + algún intercambio de información ocasional entre los procesos de la aplicación → MPI
 - Paralelismo de grano fino, disponible a nivel de bucle → OpenMP
 - El modelo híbrido puede resultar adecuado para explotar estos múltiples niveles de paralelismo.

Fundamentos del modelo híbrido

- Razones para utilizar el modelo híbrido:
 - Algunas aplicaciones presentan una carga de trabajo desbalanceada al nivel de MPI, la cual puede resultar difícil de equilibrar
 - Balancear la carga en forma dinámica con OpenMP resulta más sencillo de lograr
 - Pthreads también es una opción aunque con costo de programación mayor.

Fundamentos del modelo híbrido

- Razones para no utilizar el modelo híbrido:
 - Algunas aplicaciones sólo presentan un único nivel de paralelismo
→ la introducción de paralelismo jerárquico no provee beneficios
- Al introducir OpenMP o Pthreads a un código MPI existente también se están introduciendo sus desventajas:
 - Overhead adicional por la creación, sincronización y destrucción de hilos.
 - Dependencia en la calidad del compilador y del soporte en ejecución para OpenMP/Pthreads.
 - Cuestiones relacionadas al uso de memoria compartida, como ubicación de los datos en memoria y conflictos en el acceso a los mismos.

Esquemas del modelo híbrido

- Existen diferentes esquemas para paralelizar una aplicación utilizando el modelo híbrido.
- La clasificación se realiza teniendo en cuenta qué hilo/s envía/n mensajes entre los procesos MPI y en qué momento lo hace/n.
 - Sin solapamiento de cómputo y comunicaciones
 - Con solapamiento de cómputo y comunicaciones

Esquemas del modelo híbrido – Sin solapamiento de cómputo y comunicaciones

- También conocido como *master-only* o *modo vector*.
- Emplea un proceso MPI por nodo y OpenMP o Pthreads sobre los núcleos de cada nodo.
- Las llamadas a las rutinas de MPI son realizadas fuera de las regiones paralelas de OpenMP o del código de los hilos creados con Pthreads

Esquemas del modelo híbrido – Sin solapamiento de cómputo y comunicaciones

```
...  
/* hilo maestro */  
MPI_Recv(); /* Recibir datos */  
#pragma omp parallel  
{  
    /* Ejecución multi-hilada */  
}  
/* hilo maestro */  
MPI_Send(); /* Enviar resultados */  
...
```

Esquemas del modelo híbrido – Sin solapamiento de cómputo y comunicaciones

- Ventajas:
 - No hay intercambio de mensajes dentro de cada nodo.
 - La topología de los procesos MPI ya no es una cuestión relevante a la hora de optimizar el rendimiento de la aplicación.
- Desventajas:
 - Mientras el hilo master se comunica, el resto de los hilos está ocioso → overhead.
 - Un único hilo probablemente no sea capaz de aprovechar todo el ancho de banda disponible de la red de comunicación

Esquemas del modelo híbrido – Con solapamiento de cómputo y comunicaciones

- Una forma de evitar el ocio de los hilos durante las comunicaciones MPI consiste en permitir que más de un hilo pueda comunicarse en paralelo a otros que realicen cómputo útil.

```
...  
#pragma omp parallel private(mi_id)  
{  
    mi_id = omp_get_thread_num();  
    if (mi_id ...) /* hilo de comunicación */  
        MPI_Send();  
    else  
        if (mi_id ...) /* hilo de comunicación */  
            MPI_Recv();  
        else {  
            /* cómputo */  
        }  
}  
...
```

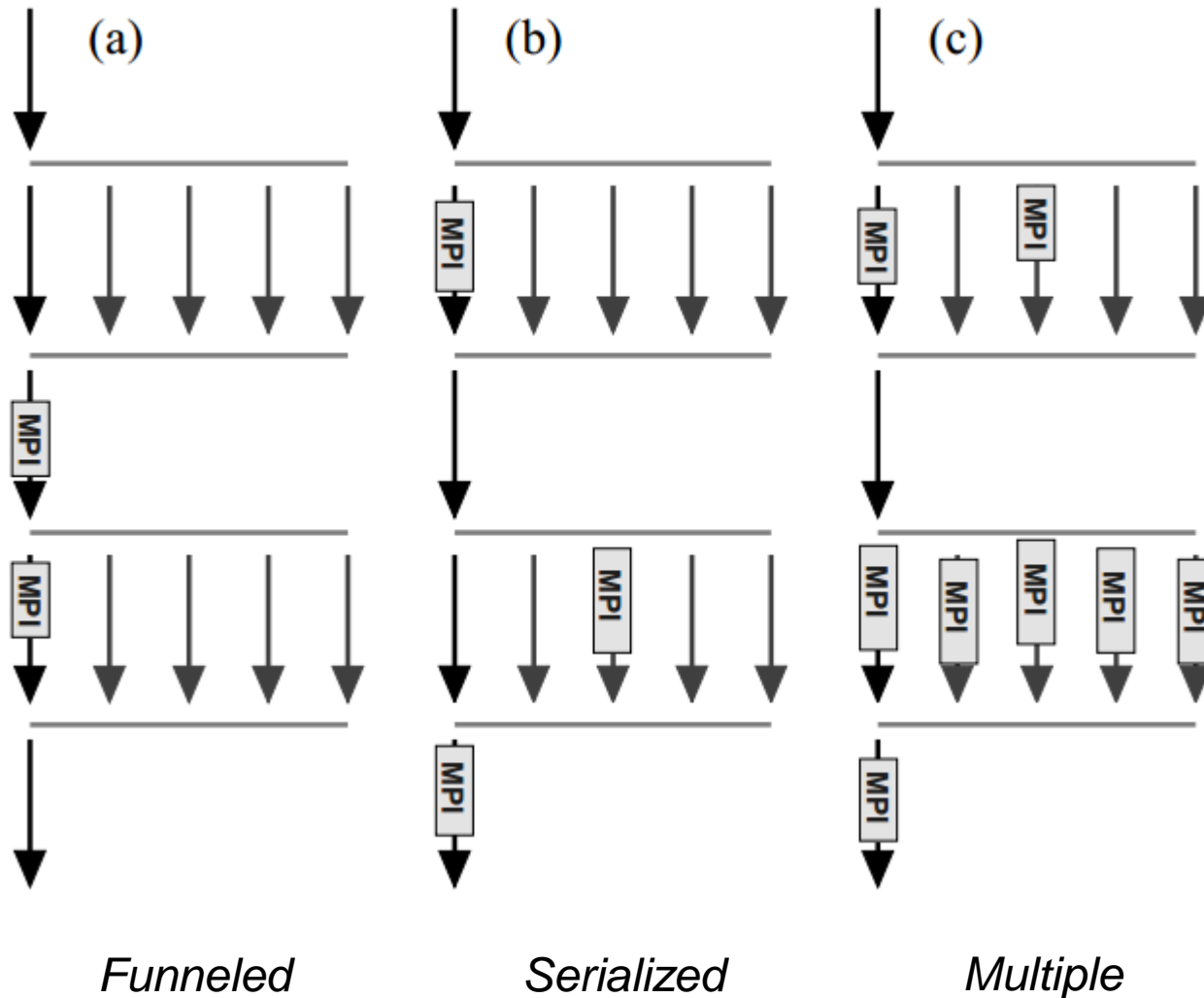
Esquemas del modelo híbrido – Con solapamiento de cómputo y comunicaciones

- Ventajas:
 - Se reduce el tiempo ocioso que los hilos podrían incurrir.
 - Se aprovecha el ancho de banda de la red.
- Desventajas:
 - Requiere mayor esfuerzo de programación.
 - Se debe equilibrar la carga de trabajo entre los hilos que comunican y los que no lo hacen.

Soporte MPI para programación híbrida

- Las librerías de MPI varían en su soporte para las comunicaciones de los hilos.
- MPI especifica 4 niveles diferentes:
 - `MPI_THREAD_SINGLE` (Nivel 0): Sin soporte para hilos
 - `MPI_THREAD_FUNNELED` (Nivel 1): Los procesos pueden ser multi-hilados pero todas las comunicaciones las realizará el hilo master
 - `MPI_THREAD_SERIALIZED` (Nivel 2): Los procesos pueden ser multi-hilados y los diferentes hilos pueden ejecutar rutinas MPI pero sólo una a la vez; los llamados a MPI no pueden ser realizados en simultáneo por 2 hilos.
 - `MPI_THREAD_MULTIPLE` (Nivel 3): Múltiples hilos pueden realizar múltiples comunicaciones, sin restricciones.

Soporte MPI para programación híbrida



Soporte MPI para programación híbrida

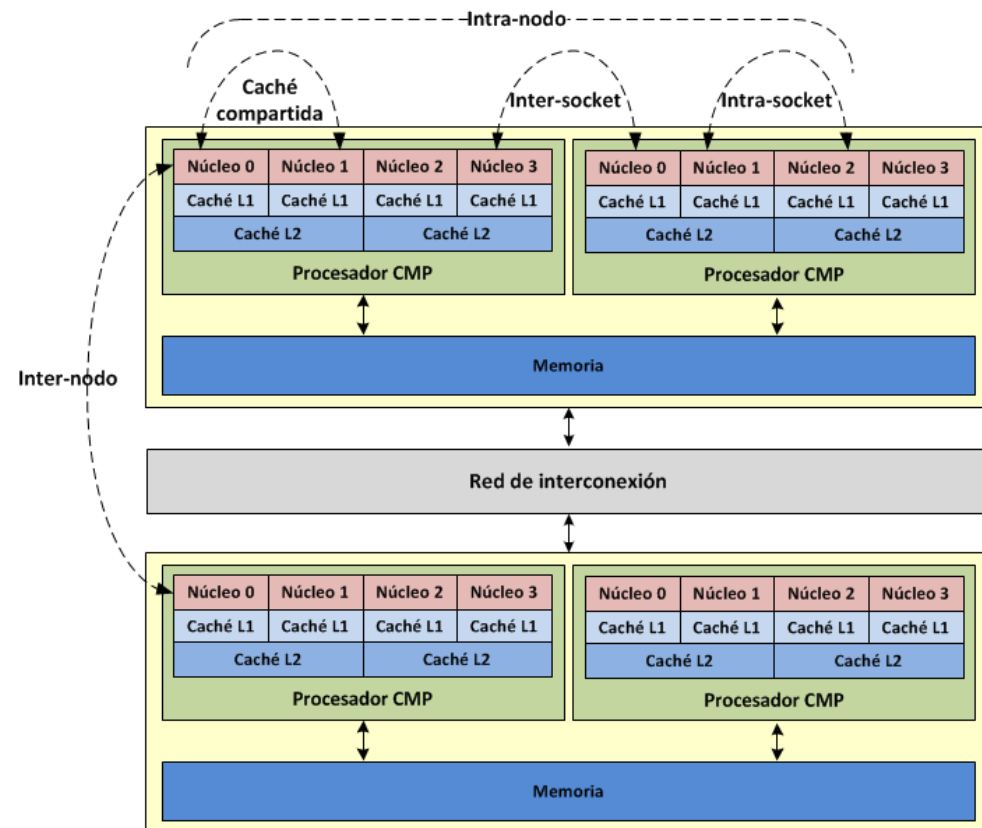
- *MPI_Init* debe reemplazarse por *MPI_Init_thread* para procesos multi-hilados:

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

```
1  #include "mpi.h"  
2  #include <stdio.h>  
3  
4  int main( int argc, char *argv[] )  
5  {  
6      int provided, claimed;  
7  
8      /*** Select one of the following  
9          MPI_Init_thread( 0, 0, MPI_THREAD_SINGLE, &provided );  
10         MPI_Init_thread( 0, 0, MPI_THREAD_FUNNELED, &provided );  
11         MPI_Init_thread( 0, 0, MPI_THREAD_SERIALIZED, &provided );  
12         MPI_Init_thread( 0, 0, MPI_THREAD_MULTIPLE, &provided );  
13     ***/  
14  
15     MPI_Init_thread(0, 0, MPI_THREAD_MULTIPLE, &provided );  
16     MPI_Query_thread( &claimed );  
17     printf( "Query thread level= %d  Init_thread level= %d\n", claimed, provided );  
18  
19     MPI_Finalize();  
20 }
```

Caso de estudio: Reducción a suma en cluster de multicores

- Debemos desarrollar un algoritmo paralelo para computar la reducción a suma de un vector.
- La arquitectura de soporte es un cluster de 2 nodos donde cada nodo tiene 2 procesadores quad-core (8 núcleos por nodo)
- ¿Opciones?
 - Algoritmo MPI
 - Algoritmo híbrido (MPI+OpenMP o MPI+Pthreads)



Caso de estudio: Reducción a suma en cluster de multicores – Usando MPI


```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <mpi.h>
4
5  #define MAX_SIZE 2000
6  #define COORDINATOR 0
7
8  int main(int argc, char* argv[]){
9      int i, numProcs, rank, size, stripSize, localSum=0, sum=0;
10     int array[MAX_SIZE];
11     MPI_Status status;
12
13     /* Lee parámetros de la línea de comando */
14     size = atoi(argv[1]);
15     size = (size < MAX_SIZE ? size : MAX_SIZE);
16
17     MPI_Init(&argc,&argv);
18
19     MPI_Comm_size(MPI_COMM_WORLD,&numProcs);
20     MPI_Comm_rank(MPI_COMM_WORLD,&rank);
21
22     if (rank == COORDINATOR)
23         for (i=0; i<size ; i++)
24             array[i] = i+1;
```

Caso de estudio: Reducción a suma en cluster de multicores – Usando MPI

```
25     stripSize = size / numProcs;
26
27     MPI_Scatter(array, stripSize, MPI_INT, array, \
28               stripSize, MPI_INT, COORDINATOR, MPI_COMM_WORLD);
29
30     for (i=0; i<stripSize; i++)
31         localSum += array[i];
32
33     MPI_Reduce(&localSum, &sum, 1, MPI_INT, MPI_SUM, COORDINATOR, MPI_COMM_WORLD);
34
35     MPI_Finalize();
36
37     if (rank==COORDINATOR)
38         printf("Sum=%d\n", sum);
39
40
41     return 0;
42 }
```

Caso de estudio: Reducción a suma en cluster de multicores – Usando híbrido (*master-only*)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <mpi.h>
4  #include <omp.h>
5
6  #define MAX_SIZE 2000
7  #define MAX_THREADS 100
8  #define COORDINATOR 0
9
10 int main(int argc, char* argv[]){
11     int i, numProcs, rank, size, stripSize, localSum=0, sum=0, threads, provided;
12     int array[MAX_SIZE];
13
14     /* Lee parámetros de la línea de comando */
15     size = atoi(argv[1]);
16     size = (size < MAX_SIZE ? size : MAX_SIZE);
17
18     threads = atoi(argv[2]);
19     threads = (threads < MAX_THREADS ? threads : MAX_THREADS);
20
21     MPI_Init_thread(&argc,&argv, MPI_THREAD_MULTIPLE, &provided);
22
23     MPI_Comm_size(MPI_COMM_WORLD,&numProcs);
24     MPI_Comm_rank(MPI_COMM_WORLD,&rank);
25
26     if (rank == COORDINATOR)
27         for (i=0; i<size ; i++)
28             array[i] = i+1;
```



Caso de estudio: Reducción a suma en cluster de multicores – Usando híbrido (*master-only*)

```
--
30     stripSize = size / numProcs;
31
32     MPI_Scatter(array, stripSize, MPI_INT, array, \
33               stripSize, MPI_INT, COORDINATOR, MPI_COMM_WORLD);
34
35     { #pragma omp parallel for num_threads(threads) reduction(+:localSum) schedule(static)
36       for (i=0; i<stripSize; i++)
37         localSum += array[i];
38     }
39     MPI_Reduce(&localSum, &sum, 1, MPI_INT, MPI_SUM, COORDINATOR, MPI_COMM_WORLD);
40
41     MPI_Finalize();
42
43     if (rank==COORDINATOR)
44         printf("Sum=%d\n", sum);
45
46
47     return 0;
48 }
```

¿Cuántos procesos se generan en cada algoritmo?

¿Qué diferencias existen en la ejecución de cada solución?
¿Sincronización? ¿Comunicación?

Bibliografía usada para esta clase

- Capítulo 11, Introduction to HPC for Scientists and Engineers. Hager, G. & Wellein, G. (2011) EEUU: CRC Press.
- Capítulo 6, Using OpenMP – Portable Shared Memory Parallel Programming. Chapman, B., Jost, G. & Van der Pas (2008). UK: MIT Press.
- Rabenseifner, R. “Hybrid Parallel Programming on HPC Platforms”. Proceedings of the Fifth European Workshop on OpenMP , EWOMP '03, (2003).
- Rabenseifner, R., Hager, G. & Jost, G. (2010). Hybrid OpenMP/MPI Parallel Programming on Clusters of Multi-Core SMP Nodes. En Proceedings of the 2010 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing (págs. 427-436). Washington, EEUU.