Computación Híbrida, Heterogénea y Jerárquica

http://www.ditec.um.es/~javiercm/curso_psba/

Curso de Programación en el Supercomputador Ben-Arabí, febrero-marzo 2012

Organización aproximada de la sesión, día 2 de marzo:

4:30-6:30	Teoría
6:30-6:45	Descanso
6:45-8:45	Prácticas

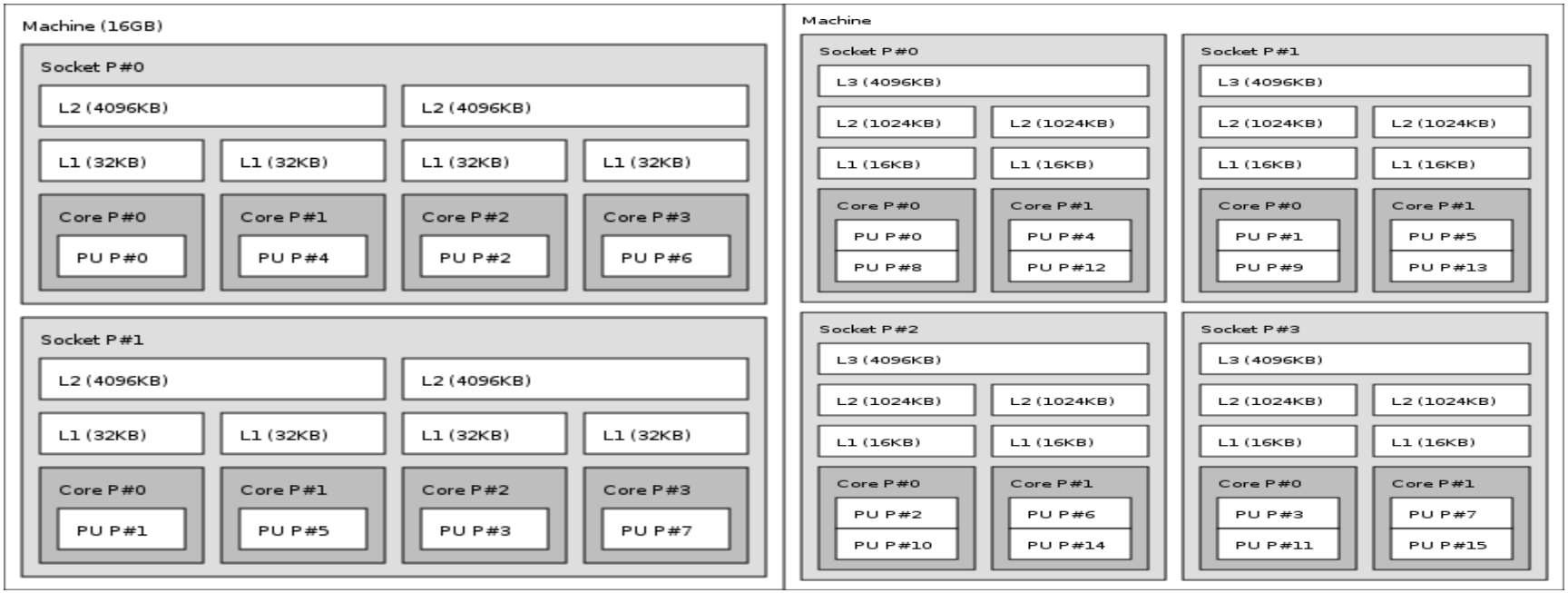
Consultas, por correo a domingo@um.es

Contenidos

Los sistemas computacionales actuales no son monolíticos:

8 cores, 2 niveles cache,
2 nodos, MC

8 cores hyperthreading, 3 niveles cache,
4 nodos, MC



Más ejemplos en **hwloc**: `http://www.open-mpi.org/projects/hwloc/`

Tenemos sistemas

- Híbridos:
  combinan distintas características: Memoria Compartida y Memoria Distribuida, multicore con GPU, varios sistemas en la red...
- Heterogéneos:
  por lo tanto sus componentes son heterogéneos: memorias de distinta capacidad y estructura de acceso, nodos a distinta velocidad y con un número distinto de cores, redes de conexión a distinta velocidad...
- Jerárquicos:
  y se estructuran de forma jerárquica: varios sistemas en la red, con cada sistema un cluster o un cc-NUMA, cada cluster con varios nodos, cada nodo varios sockets, cada socket varios cores...

Tenemos sistemas

- Híbridos:
  combinan distintas características: Memoria Compartida y Memoria Distribuida, multicore con GPU, varios sistemas en la red...
- Heterogéneos:
  por lo tanto sus componentes son heterogéneos: memorias de distinta capacidad y estructura de acceso, nodos a distinta velocidad y con un número distinto de cores, redes de conexión a distinta velocidad...
- Jerárquicos:
  y se estructuran de forma jerárquica: varios sistemas en la red, con cada sistema un cluster o un cc-NUMA, cada cluster con varios nodos, cada nodo varios sockets, cada socket varios cores...

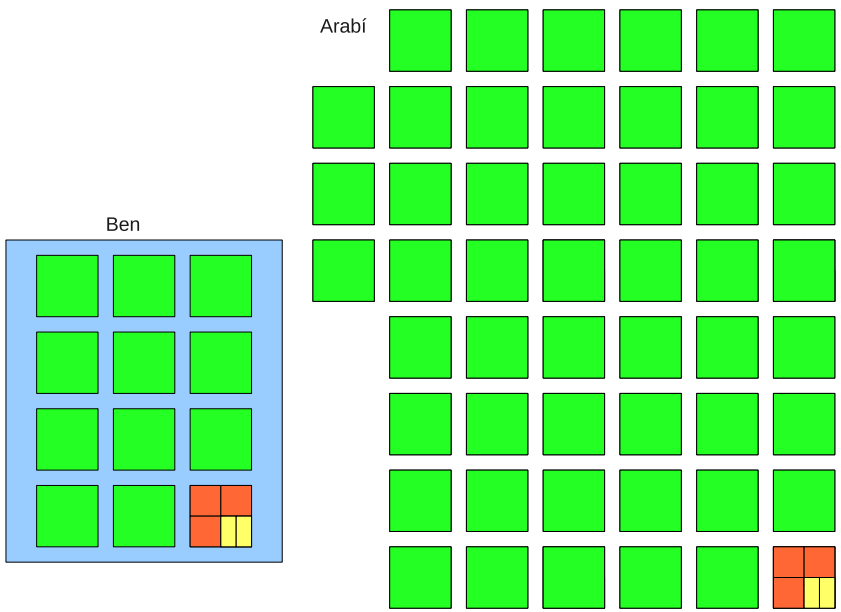
Tenemos sistemas

- Híbridos:
  combinan distintas características: Memoria Compartida y Memoria Distribuida, multicore con GPU, varios sistemas en la red...
- Heterogéneos:
  por lo tanto sus componentes son heterogéneos: memorias de distinta capacidad y estructura de acceso, nodos a distinta velocidad y con un número distinto de cores, redes de conexión a distinta velocidad...
- Jerárquicos:
  y se estructuran de forma jerárquica: varios sistemas en la red, con cada sistema un cluster o un cc-NUMA, cada cluster con varios nodos, cada nodo varios sockets, cada socket varios cores...

Ben-Arabí

Híbrido:

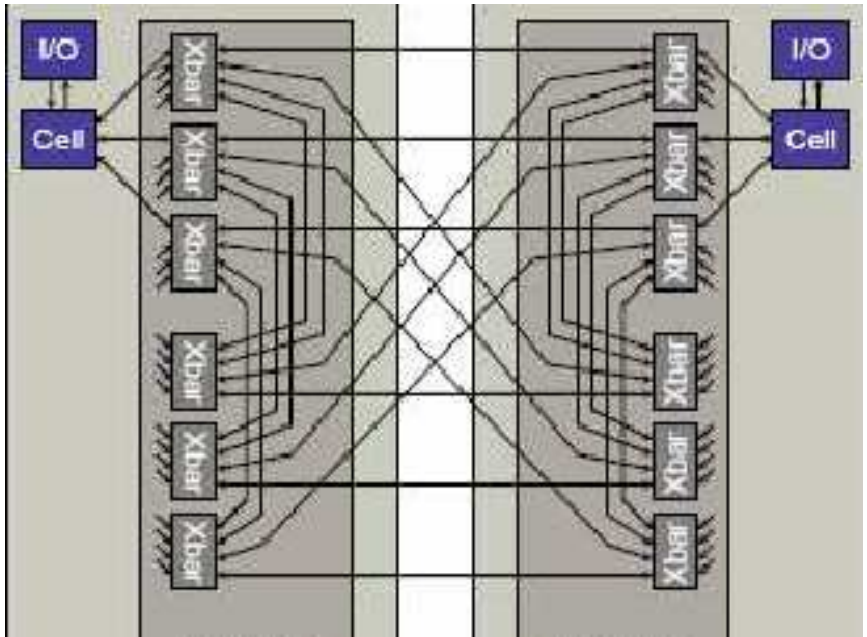
- Ben: Memoria Compartida (cc-NUMA)
  128 cores = 12 sockets * 4 procesadores * 2 cores
- Arabí: Cluster
  816 cores = 102 nodos * 4 procesadores * 2 cores



Ben-Arabí

Heterogéneo:

- Los procesadores de Ben y de Arabí van a distinta velocidad
- La memoria en Ben y en cada nodo de Arabí es distinta
- No todos los nodos de Arabí tienen la misma memoria
- Hay jerarquía de memorias, con lo que el coste de acceso a los datos depende del procesador y la zona a la que se accede, especialmente en Ben



Ben-Arabí

Jerárquico:

- Ben-Arabí está compuesto de Ben (NUMA) + Arabí (cluster)
  - Ben está formado por 12 sockets
  - Arabí por 102 nodos
    - Cada nodo por 4 procesadores
    - Cada procesador es dual

Portátiles y ordenadores personales

- Dual, quad, hexa...
- Posible hyperthreading
- Tarjeta gráfica (gforce, Tesla, AMD), con entre 96 y 512 cores
- Híbridos: multicore + GPU
- Heterogéneos:
  - distinta velocidad y memoria
  - en multicore jerarquía de memorias
  - dentro de GPU distintas memorias
  - un core del multicore, más rápido que uno de GPU
  - dependiendo del problema es más rápido el multicore o la GPU
- Jerárquico: ejecución en multicore, que lanza *kernels* para su ejecución en GPU

Portátiles y ordenadores personales

- Dual, quad, hexa...
- Posible hyperthreading
- Tarjeta gráfica (gforce, Tesla, AMD), con entre 96 y 512 cores
- Híbridos: multicore + GPU
- Heterogéneos:
  - distinta velocidad y memoria
  - en multicore jerarquía de memorias
  - dentro de GPU distintas memorias
  - un core del multicore, más rápido que uno de GPU
  - dependiendo del problema es más rápido el multicore o la GPU
- Jerárquico: ejecución en multicore, que lanza *kernels* para su ejecución en GPU

Portátiles y ordenadores personales

- Dual, quad, hexa...
- Posible hyperthreading
- Tarjeta gráfica (gforce, Tesla, AMD), con entre 96 y 512 cores
- Híbridos: multicore + GPU
- Heterogéneos:
  - distinta velocidad y memoria
  - en multicore jerarquía de memorias
  - dentro de GPU distintas memorias
  - un core del multicore, más rápido que uno de GPU
  - dependiendo del problema es más rápido el multicore o la GPU
- Jerárquico: ejecución en multicore, que lanza *kernels* para su ejecución en GPU

Portátiles y ordenadores personales

- Dual, quad, hexa...
- Posible hyperthreading
- Tarjeta gráfica (gforce, Tesla, AMD), con entre 96 y 512 cores
- Híbridos: multicore + GPU
- Heterogéneos:
  - distinta velocidad y memoria
  - en multicore jerarquía de memorias
  - dentro de GPU distintas memorias
  - un core del multicore, más rápido que uno de GPU
  - dependiendo del problema es más rápido el multicore o la GPU
- Jerárquico: ejecución en multicore, que lanza *kernels* para su ejecución en GPU

Clusters de empresas, grupos de investigación...

- Cluster de multicores
  El sistema se hace más híbrido, jerárquico y heterogéneo pero es más complicado explotar esta estructura:
  - Muchas veces uso de nodos de forma individual
  - Muchas veces ejecuciones secuenciales
  - Uso de paralelismo por llamada a rutinas paralelas
  - Raramente se utilizan las tarjetas gráficas para computación o se realiza programación paralela

Centros de supercomputación

- Sistemas de distintos tipos:
  - Marenostrum (BSC): 10000 cores en nodos duales
  - Pirineus (CESCA): 1344 cores de memoria compartida en nodos hexacore
  - Ben-Arabí (CS-FPCM): 944 cores en NUMA+cluster
- La tendencia parece ser
  - combinar MC + MD + GPU
  - pero normalmente las ejecuciones en sistemas diferenciados
- Consultar la evolución en el TOP500 (`http://www.top500.org/`)

Centros de supercomputación

- Sistemas de distintos tipos:
  - Marenostrum (BSC): 10000 cores en nodos duales
  - Pirineus (CESCA): 1344 cores de memoria compartida en nodos hexacore
  - Ben-Arabí (CS-FPCM): 944 cores en NUMA+cluster
- La tendencia parece ser
  - combinar MC + MD + GPU
  - pero normalmente las ejecuciones en sistemas diferenciados
- Consultar la evolución en el TOP500 (`http://www.top500.org/`)

Centros de supercomputación

- Sistemas de distintos tipos:
  - Marenostrum (BSC): 10000 cores en nodos duales
  - Pirineus (CESCA): 1344 cores de memoria compartida en nodos hexacore
  - Ben-Arabí (CS-FPCM): 944 cores en NUMA+cluster
- La tendencia parece ser
  - combinar MC + MD + GPU
  - pero normalmente las ejecuciones en sistemas diferenciados
- Consultar la evolución en el TOP500 (`http://www.top500.org/`)

Programación

Para poder explotar plenamente estos sistemas computacionales es necesario otro tipo de programación. Posibilidades:

- Paralelismo anidado en OpenMP.
- Paralelismo multinivel: threads OpenMP y dentro llamadas a rutinas paralelas (MKL multithreading)
- Programación híbrida: MPI+OpenMP, se crean procesos MPI y cada proceso crea varios threads OpenMP
- Combinar los tres: MPI+OpenMP+MKL
- Computación heterogénea
- Adaptativa
- *Grid computing*, *Cloud computing*, *Web computing*, *P2P computing*, *Sky computing*, *Jungle computing*...

Programación

Para poder explotar plenamente estos sistemas computacionales es necesario otro tipo de programación. Posibilidades:

- Paralelismo anidado en OpenMP.
- Paralelismo multinivel: threads OpenMP y dentro llamadas a rutinas paralelas (MKL multithreading)
- Programación híbrida: MPI+OpenMP, se crean procesos MPI y cada proceso crea varios threads OpenMP
- Combinar los tres: MPI+OpenMP+MKL
- Computación heterogénea
- Adaptativa
- *Grid computing*, *Cloud computing*, *Web computing*, *P2P computing*, *Sky computing*, *Jungle computing*...

Programación

Para poder explotar plenamente estos sistemas computacionales es necesario otro tipo de programación. Posibilidades:

- Paralelismo anidado en OpenMP.
- Paralelismo multinivel: threads OpenMP y dentro llamadas a rutinas paralelas (MKL multithreading)
- Programación híbrida: MPI+OpenMP, se crean procesos MPI y cada proceso crea varios threads OpenMP
- Combinar los tres: MPI+OpenMP+MKL
- Computación heterogénea
- Adaptativa
- *Grid computing*, *Cloud computing*, *Web computing*, *P2P computing*, *Sky computing*, *Jungle computing*...

Programación

Para poder explotar plenamente estos sistemas computacionales es necesario otro tipo de programación. Posibilidades:

- Paralelismo anidado en OpenMP.
- Paralelismo multinivel: threads OpenMP y dentro llamadas a rutinas paralelas (MKL multithreading)
- Programación híbrida: MPI+OpenMP, se crean procesos MPI y cada proceso crea varios threads OpenMP
- Combinar los tres: MPI+OpenMP+MKL
- Computación heterogénea
- Adaptativa
- *Grid computing*, *Cloud computing*, *Web computing*, *P2P computing*, *Sky computing*, *Jungle computing*...

Programación

Para poder explotar plenamente estos sistemas computacionales es necesario otro tipo de programación. Posibilidades:

- Paralelismo anidado en OpenMP.
- Paralelismo multinivel: threads OpenMP y dentro llamadas a rutinas paralelas (MKL multithreading)
- Programación híbrida: MPI+OpenMP, se crean procesos MPI y cada proceso crea varios threads OpenMP
- Combinar los tres: MPI+OpenMP+MKL
- Computación heterogénea
- Adaptativa
- *Grid computing*, *Cloud computing*, *Web computing*, *P2P computing*, *Sky computing*, *Jungle computing*...

Programación

Para poder explotar plenamente estos sistemas computacionales es necesario otro tipo de programación. Posibilidades:

- Paralelismo anidado en OpenMP.
- Paralelismo multinivel: threads OpenMP y dentro llamadas a rutinas paralelas (MKL multithreading)
- Programación híbrida: MPI+OpenMP, se crean procesos MPI y cada proceso crea varios threads OpenMP
- Combinar los tres: MPI+OpenMP+MKL
- Computación heterogénea
- Adaptativa
- *Grid computing*, *Cloud computing*, *Web computing*, *P2P computing*, *Sky computing*, *Jungle computing*...

Programación

Para poder explotar plenamente estos sistemas computacionales es necesario otro tipo de programación. Posibilidades:

- Paralelismo anidado en OpenMP.
- Paralelismo multinivel: threads OpenMP y dentro llamadas a rutinas paralelas (MKL multithreading)
- Programación híbrida: MPI+OpenMP, se crean procesos MPI y cada proceso crea varios threads OpenMP
- Combinar los tres: MPI+OpenMP+MKL
- Computación heterogénea
- Adaptativa
- *Grid computing*, *Cloud computing*, *Web computing*, *P2P computing*, *Sky computing*, *Jungle computing*...

Computación heterogénea

- Asignar los procesos de un programa homogéneo a un sistema heterogéneo con un número de procesos proporcional a la velocidad de cada nodo
- Hacer programas heterogéneos: donde la cantidad de trabajo asignada a cada proceso es proporcional a la velocidad del procesador al que se asigna
- Paralelismo multinivel con distinto número de threads en cada subgrupo de threads esclavos, dependiendo de la velocidad del nodo o del volumen de computación a realizar
- GPCPU es heterogénea por tener varios tipos de memoria
- OpenMP+GPU: posible decidir en cada parte del programa si se asigna a la GPU o al multicore, dependiendo del coste y del tipo de computación

Computación heterogénea

- Asignar los procesos de un programa homogéneo a un sistema heterogéneo con un número de procesos proporcional a la velocidad de cada nodo
- Hacer programas heterogéneos: donde la cantidad de trabajo asignada a cada proceso es proporcional a la velocidad del procesador al que se asigna
- Paralelismo multinivel con distinto número de threads en cada subgrupo de threads esclavos, dependiendo de la velocidad del nodo o del volumen de computación a realizar
- GPCPU es heterogénea por tener varios tipos de memoria
- OpenMP+GPU: posible decidir en cada parte del programa si se asigna a la GPU o al multicore, dependiendo del coste y del tipo de computación

Computación heterogénea

- Asignar los procesos de un programa homogéneo a un sistema heterogéneo con un número de procesos proporcional a la velocidad de cada nodo
- Hacer programas heterogéneos: donde la cantidad de trabajo asignada a cada proceso es proporcional a la velocidad del procesador al que se asigna
- Paralelismo multinivel con distinto número de threads en cada subgrupo de threads esclavos, dependiendo de la velocidad del nodo o del volumen de computación a realizar
- GPCPU es heterogénea por tener varios tipos de memoria
- OpenMP+GPU: posible decidir en cada parte del programa si se asigna a la GPU o al multicore, dependiendo del coste y del tipo de computación

Computación heterogénea

- Asignar los procesos de un programa homogéneo a un sistema heterogéneo con un número de procesos proporcional a la velocidad de cada nodo
- Hacer programas heterogéneos: donde la cantidad de trabajo asignada a cada proceso es proporcional a la velocidad del procesador al que se asigna
- Paralelismo multinivel con distinto número de threads en cada subgrupo de threads esclavos, dependiendo de la velocidad del nodo o del volumen de computación a realizar
- GPCPU es heterogénea por tener varios tipos de memoria
- OpenMP+GPU: posible decidir en cada parte del programa si se asigna a la GPU o al multicore, dependiendo del coste y del tipo de computación

Computación heterogénea

- Asignar los procesos de un programa homogéneo a un sistema heterogéneo con un número de procesos proporcional a la velocidad de cada nodo
- Hacer programas heterogéneos: donde la cantidad de trabajo asignada a cada proceso es proporcional a la velocidad del procesador al que se asigna
- Paralelismo multinivel con distinto número de threads en cada subgrupo de threads esclavos, dependiendo de la velocidad del nodo o del volumen de computación a realizar
- GPCPU es heterogénea por tener varios tipos de memoria
- OpenMP+GPU: posible decidir en cada parte del programa si se asigna a la GPU o al multicore, dependiendo del coste y del tipo de computación

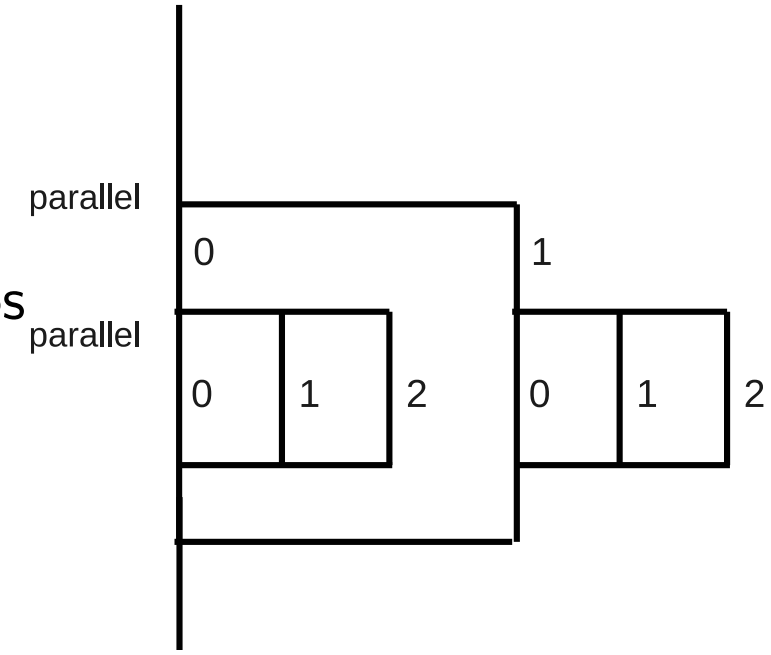
Sistemas - Práctica

Sist.1 En Ben-Arabí hay una cola heterogénea que no se usa normalmente, pero que vamos a usar para ejecutar un ejemplo simple en Ben y Arabí.
En `/project/FORMACION/Hibrida/Sistemas` hay un ejemplo de uso de la cola `hetero`.
Hay que compilar en Ben con `make sd`, para generar `mainsd`, y en Arabí con `make cc`, para generar `maincc`.
Antes de compilar hay que cargar los módulos `intel` e `impi`.
Se lanza a la cola con `bsub < bsub.main`.
La práctica consiste en lanzar a la cola y entender lo que ocurre.

Paralelismo anidado

En OpenMP:

- El thread inicial pone en marcha (`parallel`) un grupo de threads esclavos
- Cada esclavo genera un grupo de esclavos de los que es el maestro

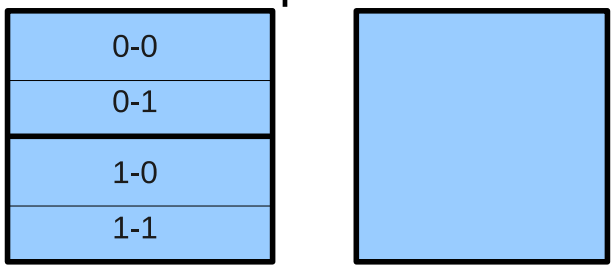


Útil para:

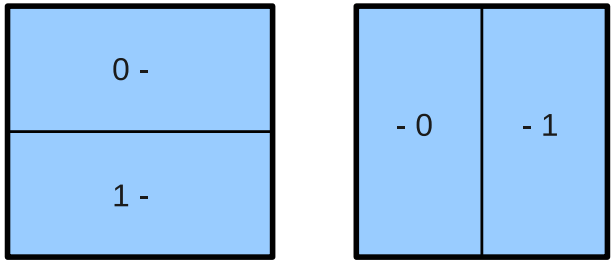
- Mejorar el uso de una memoria organizada jerárquicamente (Ben)
- Asignar unas pocas tareas muy costosas: un thread para cada tarea, y en la resolución de cada tarea trabaja un grupo de threads distinto

Paralelismo anidado - Práctica

Anid.3 Programar una multiplicación de matrices de dos niveles, con $h1$ threads en el primer nivel y $h2$ en el segundo
Experimentar si se obtiene ventaja al usar dos niveles
Probar con división sólo de la primera matriz



y con división de una matriz en cada nivel



Usar como ejemplo base el programa `codigo6-6.c` en `/project/FORMACION/Hibrida/Anidado`, u otro ejemplo de multiplicación de matrices con OpenMP.

Paralelismo anidado - Práctica

Anid.2 Hacer un programa para multiplicar m matrices A_i por otra matriz B
creando m threads en el primer nivel y otro número de threads en el segundo nivel.

Usar como ejemplo base el programa `codigo6-6.c` en `/project/FORMACION/Hibrida/Anidado`, u otro ejemplo de multiplicación de matrices con OpenMP.

Programación híbrida

Combinar MPI + OpenMP

- Se ponen en marcha procesos MPI, que se asignan a distintos nodos (Memoria Distribuida), o posiblemente varios procesos al mismo nodo
- Cada proceso genera varios threads dentro de un nodo (Memoria Compartida)

Comparación OpenMP - MPI

OpenMP	MPI
Paralelismo de grano fino	de grado grueso
Eficiencia en SMP	Más portable
Código secuencial y paralelo similar	Código secuencial y paralelo diferentes
Herramientas de desarrollo y paralelización	Desarrollo y depuración más difícil
Asignación en tiempo de ejecución	Asignación estática de procesos
La asignación de memoria puede reducir prestaciones	Memorias locales facilitan eficiencia

Ventajas de programación híbrida

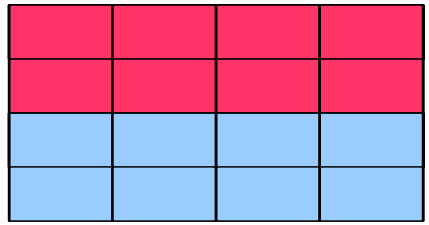
- Para mejorar la escalabilidad
- Aplicaciones que combinan paralelismo de grano grueso y fino
- Reducción del tiempo de desarrollo de código
- Cuando el número de procesos MPI es fijo
- En caso de mezcla de paralelismo funcional y de datos

Ejemplo - hello

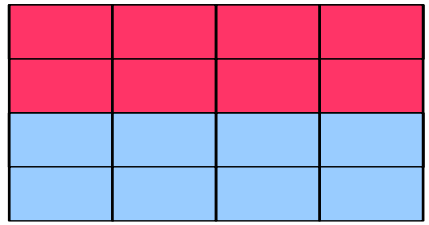
```
int main(int argc, char *argv[]) {
  int nthreads,nprocs,idpro,idthr,namelen;
  char processor_name[MPI_MAX_PROCESSOR_NAME];
  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD,&idpro);
  MPI_Get_processor_name(processor_name,&namelen);
#pragma omp parallel private(idthr)
firstprivate(idpro,processor_name)
  {
   idthr = omp_get_thread_num();
   printf("...  thread %d, proceso %d procesador
%s\n",idthr,idpro,processor_name);
   if (idthr == 0) {
    nthreads = omp_get_num_threads();
    printf("proceso %d, threads %d, procesos
%d\n",idpro,nthreads,nprocs);
   } }
  MPI_Finalize(); }
```

Ejemplo - multiplicación de matrices

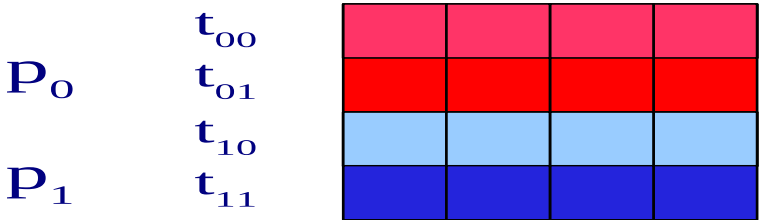
En OpenMP



En MPI



En MPI+OpenMP



Ejemplo - multiplicación de matrices (main I)

```
int main(int argc, char *argv[]) {
  ...
  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&np);
  MPI_Comm_rank(MPI_COMM_WORLD,&nodo);
  MPI_Get_processor_name(processor_name,&long_name);
  if(nodo==0) {
   N=atoi(argv[1]);
   NUMTHREADS=atoi(argv[2]);
  }
  MPI_Bcast(&N,1,MPI_INT,0,MPI_COMM_WORLD);
  MPI_Bcast(&NUMTHREADS,1,MPI_INT,0,MPI_COMM_WORLD);
  omp_set_num_threads(NUMTHREADS);
  ...
```

Ejemplo - multiplicación de matrices (main II)

```
  if(nodo==0) {
    ...
    for(i=1;i<np;i++) {

MPI_Send(&a[i*lda*N/np],fal*ca,MPI_DOUBLE,i,20,MPI_COMM_WORLD);
    }
    MPI_Bcast(b,fb*cb,MPI_DOUBLE,0,MPI_COMM_WORLD);
  }
  else {
    MPI_Recv(a,fal*ca,MPI_DOUBLE,0,20,MPI_COMM_WORLD,&estado);
    MPI_Bcast(b,fb*cb,MPI_DOUBLE,0,MPI_COMM_WORLD);
  }
```

Ejemplo - multiplicación de matrices (main III)

```
   MPI_Barrier(MPI_COMM_WORLD);
   ti=MPI_Wtime();

mm(a,fal,ca,lda,b,fb,cb,ldb,c,fcl,cc,ldc,nodo,nombre_procesador);
   MPI_Barrier(MPI_COMM_WORLD);
   tf=MPI_Wtime();
   if(nodo==0) {
    ...
    for(i=1;i<np;i++) {

MPI_Recv(&c[i*ldc*N/np],fcl*cc,MPI_DOUBLE,i,30,MPI_COMM_WORLD,&estado);
    }
   }
   else {
    MPI_Send(c,fcl*cc,MPI_DOUBLE,0,30,MPI_COMM_WORLD);
   }
```

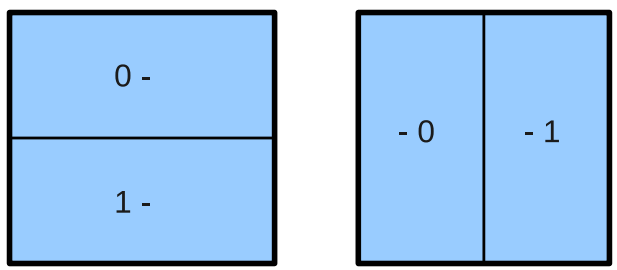
Ejemplo - multiplicación de matrices

```
void mm(...)  {
   ...
   #pragma omp parallel
   {
    #pragma omp critical
     printf("Proceso %d, nodo %s, Threads numero
%d\n",nodo,maquina,omp_get_thread_num());
     #pragma omp for private(i,j,k,s) schedule(static)
      for(i=0;i<fa;i++) {
       for(j=0;j<cb;j++) {
        s=0.;
        for(k=0;k<ca;k++)
         s=s+a[i*lda+k]*b[k*ldb+j];
        c[i*ldc+j]=s;
       }
      }
   }
}
```

Programación híbrida - Práctica

Hibr.1 Utilizar el programa ejemplo de multiplicación de matrices con MPI+OpenMP de la sesión de calentamiento del concurso de programación paralela de 2011 (mooshak en cpp.fpcmur.es, y se dará cuenta para acceder). Comprobar experimentalmente el número de procesos MPI y threads OpenMP con que se obtiene menor tiempo de ejecución.

Hibr.2 Programar una multiplicación de matrices MPI+OpenMP, con división de la primera matriz en procesos MPI y de la segunda en threads OpenMP. Comparar los resultados con los obtenidos con la implementación anterior



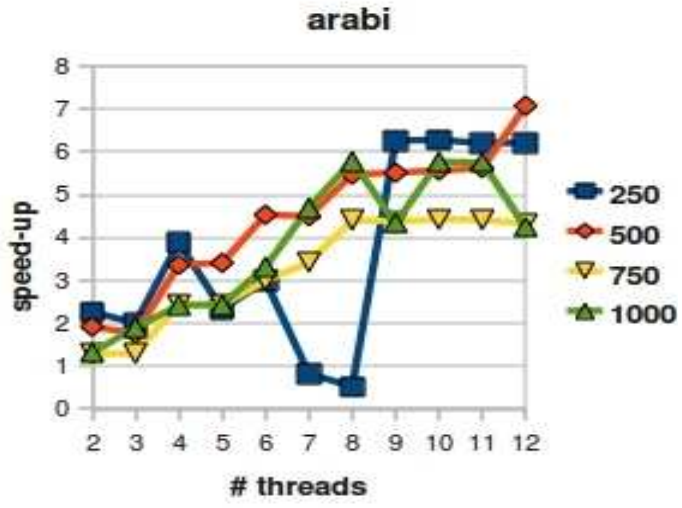
Paralelismo multinivel

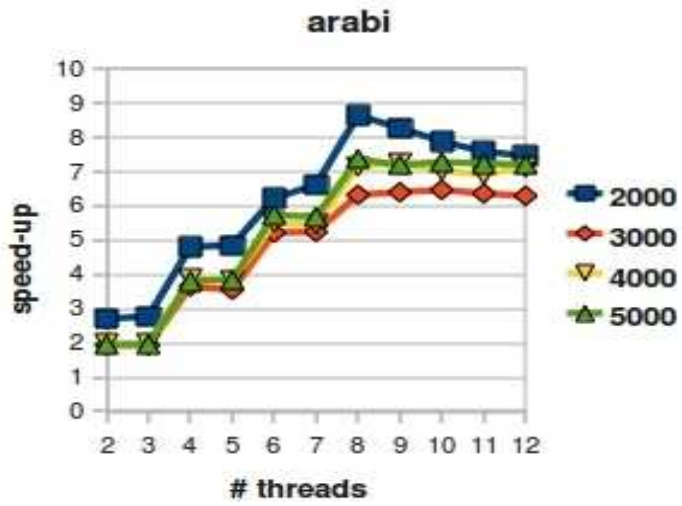
- El paralelismo anidado y el híbrido son ejemplos de paralelismo multinivel.
- En algunos casos podemos tener en un nivel paralelismo explícito (OpenMP) y en otro implícito (BLAS multithreading).
- Es posible tener más niveles de paralelismo: MPI+OpenMP+BLAS

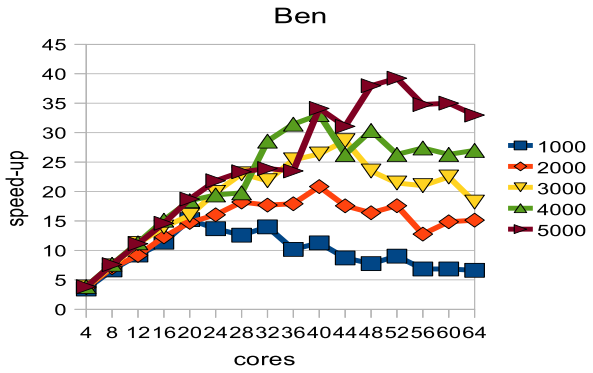
Paralelismo en MKL

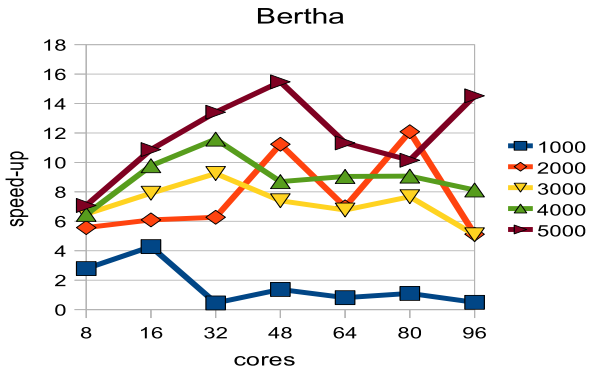
- La librería es multithreaded.
- El número de threads en las rutinas de MKL se establece con la variable de entorno `MKL_NUM_THREADS` o en el programa con la función `mkl_set_num_threads`.
- Con `MKL_DYNAMIC=true` o `mkl_set_dynamic(1)` se activa el paralelismo dinámico:
  El número de threads a usar en la rutina lo decide el sistema, y es menor o igual al valor establecido.
- Para forzar a que se use el número de threads que se indica hay que deshabilitar el paralelismo dinámico:
  `MKL_DYNAMIC=false` o `mkl_set_dynamic(0)`.

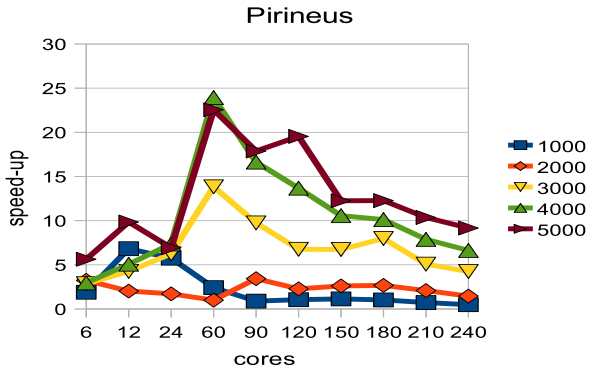
Resultados con la multiplicación de matrices











Resultados con la multiplicación de matrices

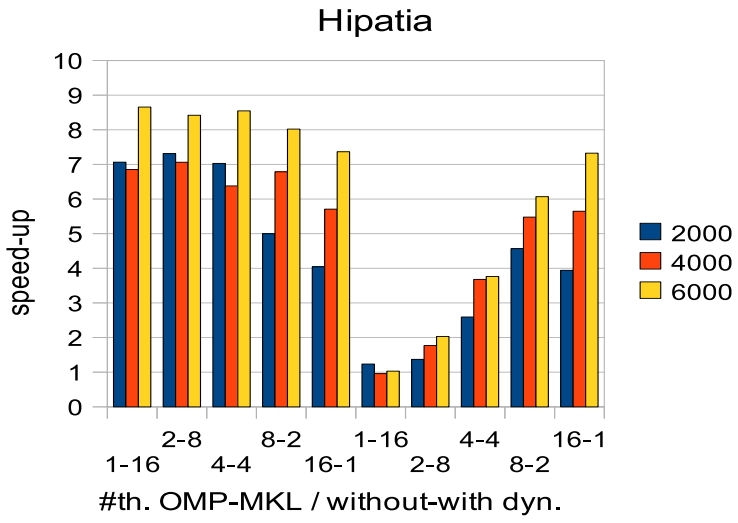
Tamaño	Secuen.	Máx. cores	Mínimo / cores	Speed-up Máx. cores/Mín. tiempo
		Ben		
500	0.042	0.033	0.0044 (19)	7.5
750	0.14	0.063	0.010 (22)	6.3
1000	0.32	0.094	0.019 (27)	4.9
2000	2.6	0.39	0.12 (37)	3.3
3000	8.6	0.82	0.30 (44)	2.7
4000	20	1.4	0.59 (50)	2.4
5000	40	2.1	1.0 (48)	2.1
		Bertha		
1000	0.25	0.50	0.058 (16)	8.6
2000	1.8	0.35	0.15 (80)	2.3
3000	6.2	1.2	0.67 (32)	1.8
4000	15	1.9	1.3 (32)	1.5
5000	30	2.0	1.9 (48)	1.1
		Pirineus		
1000	0.22	0.45	0.032 (12)	1.4
2000	1.7	1.2	0.48 (16)	2.5
3000	5.5	1.3	0.40 (60)	3.3
4000	13	1.9	0.54 (60)	3.5
5000	25	2.7	1.1 (60)	2.5

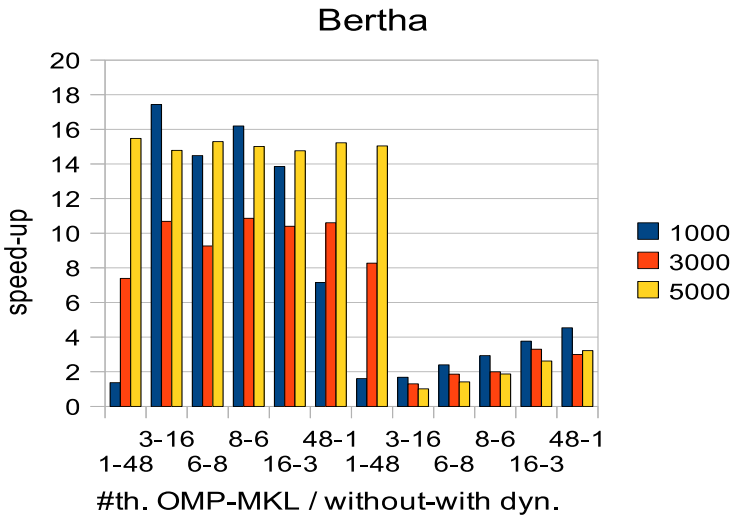
Multiplicación OpenMP+MKL

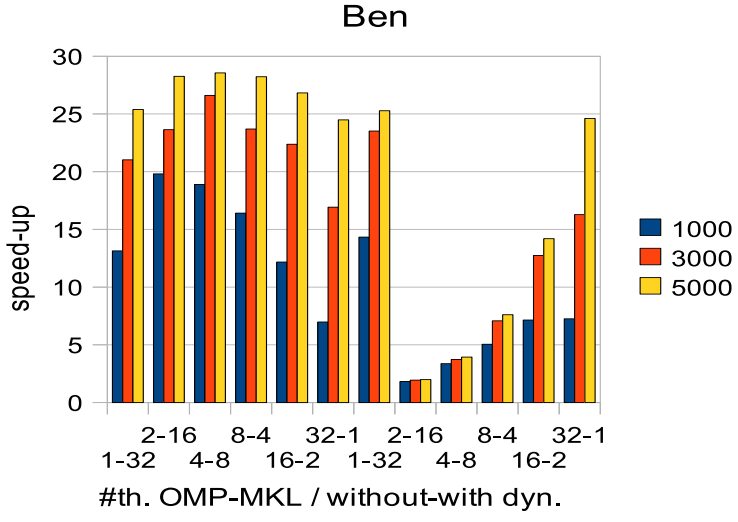
Las filas de la primera matriz se distribuyen en threads OpenMP (*nthomp*).
Se establece un número de threads a usar en las rutinas MKL (*nthmkl*).
Hay que habilitar el paralelismo anidado, con `OMP_NESTED=true` o `omp_set_nested(1)`.

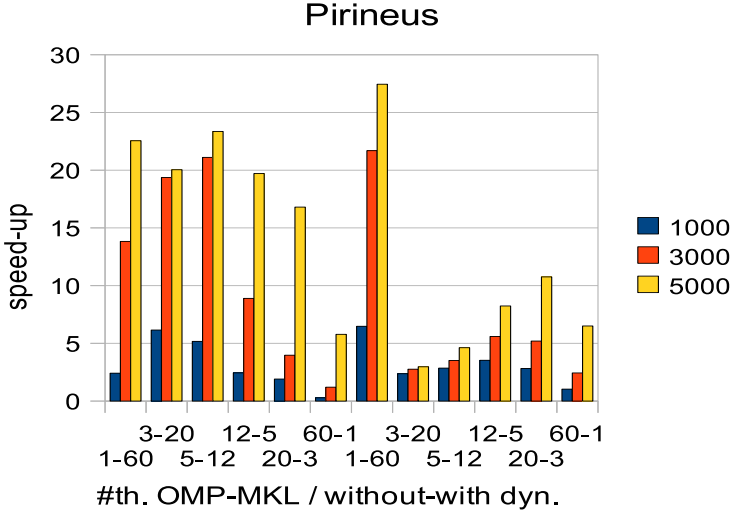
```
omp_set_nested(1);
omp_set_num_threads(nthomp);
mkl_set_dynamic(0);
mkl_set_num_threads(nthmkl);
#pragma omp parallel
    obtener tamaño y posición inicial de la submatriz de A
        a multiplicar
    llamar a dgemm para multiplicar esa submatriz por B
```

Resultados con la multiplicación OpenMP+MKL







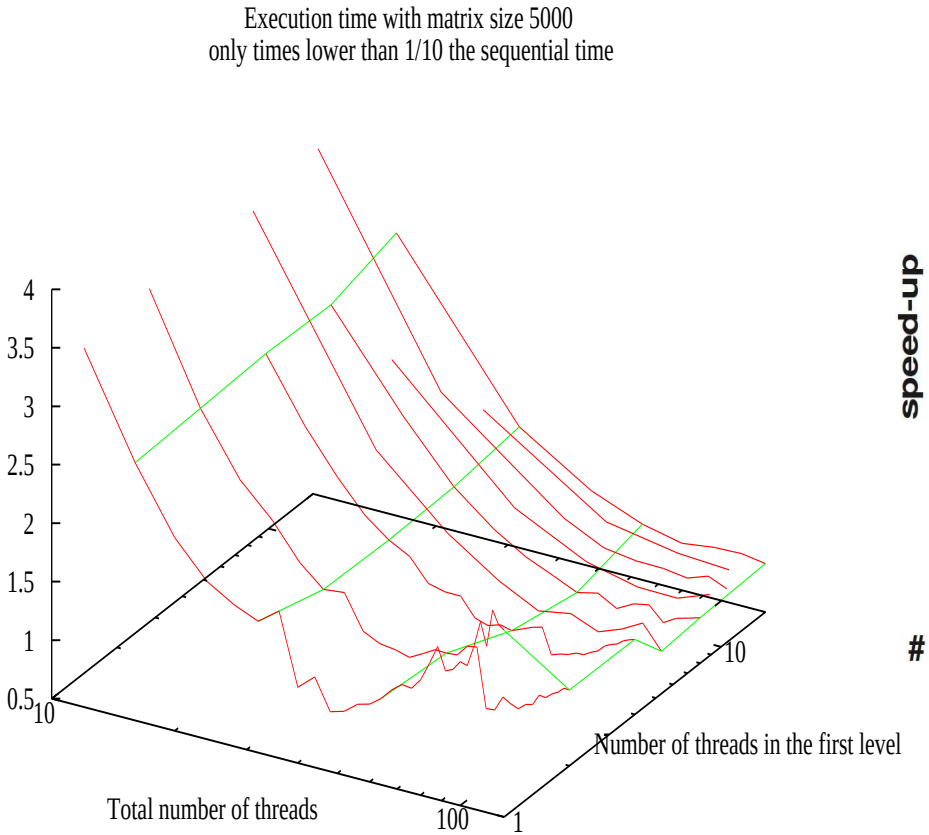


Resultados con la multiplicación OpenMP+MKL

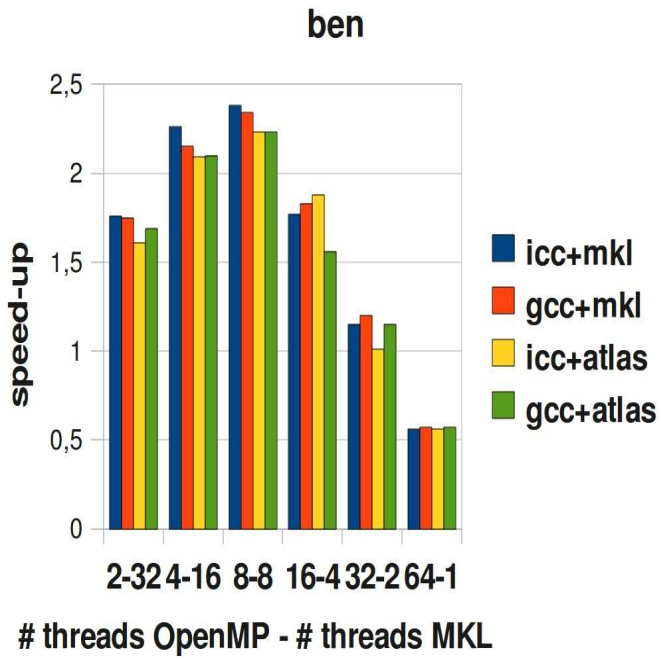
Tamaño	MKL	OpenMP+MKL	Speed-up MKL/OpenMP+MKL
		Ben	
1000	0.019 (27)	0.015 (4-10)	1.3
2000	0.12 (37)	0.072 (4-16)	1.6
3000	0.30 (44)	0.18 (4-24)	1.7
4000	0.59 (50)	0.41 (5-16)	1.4
5000	1.0 (48)	0.76 (6-20)	1.3
10000	10 (64)	5.0 (32-4)	2.0
15000	25 (64)	12 (32-4)	2.1
20000	65 (64)	22 (16-8)	3.0
		Bertha	
1000	0.058 (16)	0.012 (2-24)	4.8
2000	0.15 (80)	0.053 (5-16)	2.8
3000	0.67 (32)	0.51 (16-3)	1.3
4000	1.3 (32)	0.98 (5-16)	1.3
5000	1.9 (48)	1.7 (3-32)	1.1
		Pirineus	
1000	0.032 (12)	0.024 (2-16)	1.3
2000	0.48 (16)	0.08 (5-12)	6.0
3000	0.40 (60)	0.28 (4-15)	1.4
4000	0.54 (60)	0.47 (5-12)	1.1
5000	1.1 (60)	0.86 (10-9)	1.3
10000	8.8 (120)	5.3 (20-6)	1.7

Multiplicación OpenMP+MKL, en Ben

Tiempo de ejecución variando el número de threads OpenMP y MKL



Resultados similares con otros compiladores y librerías (gcc 4.4 y ATLAS 3.9)



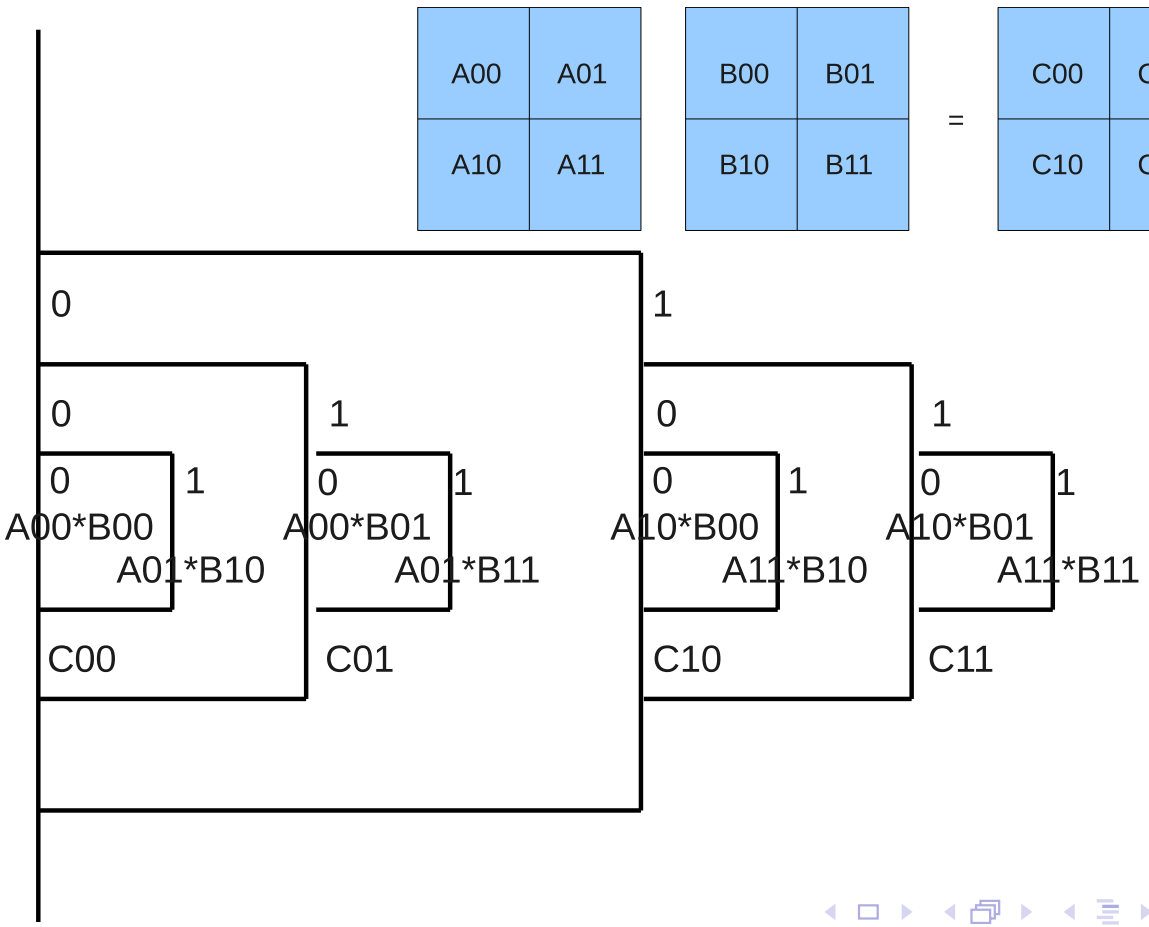
Paralelismo multinivel - Práctica

Mult.6 Programar la multiplicación de matrices cuadradas con tres niveles de paralelismo (MPI+OpenMP+MKL).
Ejecutarla en Arabí con cuatro nodos y un total de 32 cores.
Comprobar para distintos tamaños de matriz la mejor combinación de número de procesos MPI, threads OpenMP y threads MKL.

Usar como ejemplo base los ficheros en `/project/FORMACION/Hibrida/Multinivel`.

Paralelismo multinivel - Práctica

Mult.5 Programar la multiplicación de matrices cuadradas dividiendo las matrices en submatrices de la mitad de tamaño, con el esquema siguiente, donde se usan tres niveles de anidamiento OpenMP y paralelismo implícito en la multiplicación y suma de matrices con MKL



Computación heterogénea

- Cuando el sistema computacional es heterogéneo: procesadores de distinta velocidad, distinta capacidad de memoria, coste distinto de acceso a memoria, varias redes de comunicación, librerías con distintas prestaciones...
- Cuando la computación es heterogénea: tareas con distinto coste, patrones de comunicación diferentes...

Sistemas heterogéneos de memoria compartida

Podemos tener procesadores distintos en un sistema de memoria compartida (no en Ben-Arabí).
En este caso se puede:

- Usar paralelismo de dos niveles: en el primer nivel se usan tantos threads como procesadores, y en el segundo nivel el número de threads es proporcional a la velocidad del procesador.
- Con un único nivel de paralelismo, crear tantos threads como procesadores pero asignando un volumen de cómputo a cada procesador proporcional a su velocidad.

Heterogeneidad en memoria compartida - Práctica

HeMC.1 Suponiendo que tenemos 4 procesadores con velocidades 4, 3, 2 y 1, hacer un programa "hello" de manera que cada procesador salude un número de veces proporcional a su velocidad.

Usar como ejemplo base el `hello_mpi+omp.c` en `/project/FORMACION/Hibrida/Hibrida`.

HeMC.3 Suponiendo que tenemos 4 procesadores con velocidades 4, 3, 2 y 1, hacer un programa de multiplicación de matrices de manera que cada procesador calcule un número de filas de la matriz resultado proporcional a su velocidad.

Usar como ejemplo base el `mmhibrida.c` en `/project/FORMACION/Hibrida/Hibrida`.

Clusters heterogéneos

Podemos tener clusters con nodos de distintas características (en Ben-Arabí si se utilizan conjuntamente Ben y Arabí).
En centros de supercomputación, aunque se tengan distintos sistemas no se suelen utilizar como heterogéneos.
Se puede:

- Realizar programas homogéneos, con todos los procesos con la misma carga computacional, y asignar más procesos a los procesadores con más velocidad.
- Hacer programas heterogéneos: un proceso por nodo, pero cada proceso con una carga computacional proporcional a la velocidad del nodo al que se asigna.
- ¿Más posibilidades?

Cluster heterogéneos - Práctica

HePM.1 Realizar alguna ejecución de la multiplicación homogénea de matrices con MPI asignando varios procesos a un mismo nodo en Arabí.

HePM.4 Suponiendo que usamos 4 nodos de Arabí y que tienen velocidades 4, 3, 2 y 1, hacer un programa de multiplicación de matrices en MPI de manera que cada nodo calcule un número de filas de la matriz resultado proporcional a su velocidad.

HePM.2 Utilizando la cola heterogénea de Ben-Arabí, realizar alguna ejecución de la multiplicación de matrices con MPI.

Usar como ejemplos base el `codigo6.10.c` en `/project/FORMACION/Hibrida/Heterogenea`, u otro ejemplo de multiplicación de matrices con MPI, y los ficheros de `/project/FORMACION/Hibrida/Sistemas`.

General-Purpose Computation on Graphics Processing Units

- Procesadores gráficos en todos los sistemas computacionales.
- Muchas veces para juegos, pero también se pueden usar para computación.
- Disponen de muchos cores pequeños.
- Son baratos, y se puede conseguir mayores aceleraciones a un coste bajo.
- Pero la programación es más compleja.
- En algunos centros de supercomputación se incluyen GPU en los sistemas. En la actualidad no en Ben-Arabí, pero sí en el futuro. Quizás en el próximo curso se incluya.

Software para GPU

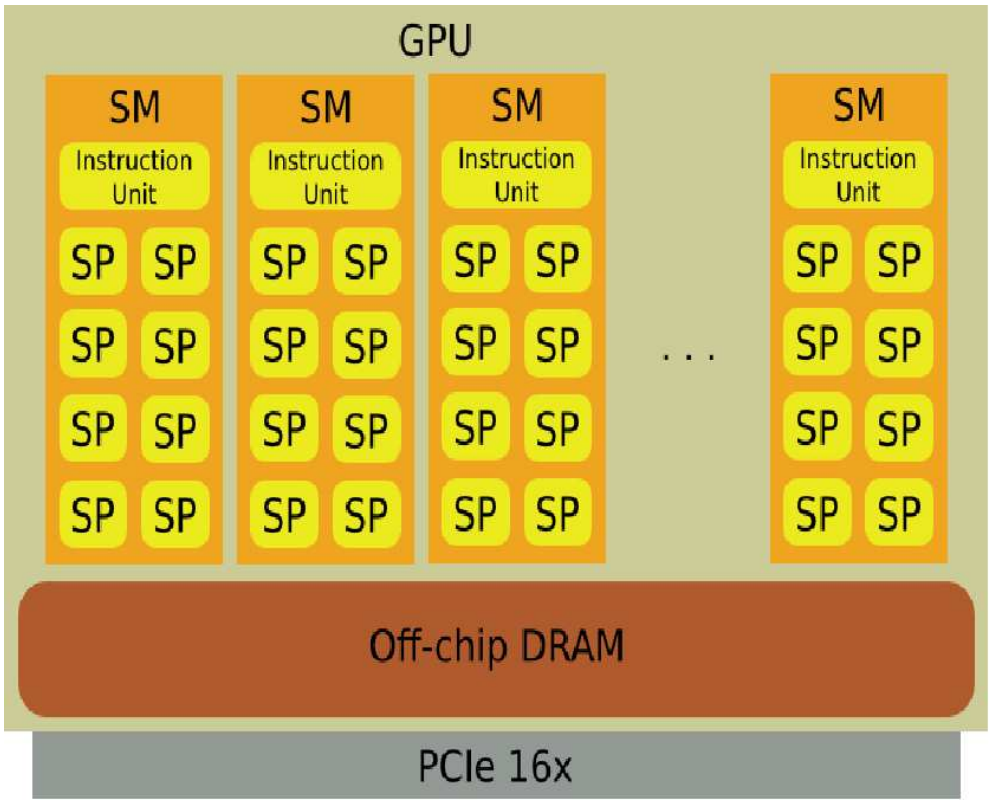
- 2006: NVIDIA introduce la arquitectura unificada (CUDA) con la presentación de la NVIDIA GeForce 8800.
  GPUs de NVIDIA compatibles con CUDA
  Modelos: GeForce, Tesla, Quadro
- 2009: OpenCL, GPGPU para plataformas gráficas
  Intenta convertirse en un estándar.
- Hay disponibles versiones de BLAS y otras librerías.

CUDA

- Arquitectura hardware y software:
  Uso de GPU, construida a partir de la replicación de un bloque constructivo básico, como acelerador con memoria integrada
  Estructura jerárquica de threads mapeada sobre el hardware
- Gestión de memoria explícita
- Creación, planificación y ejecución transparente de miles de threads de manera concurrente
- Extensiones del lenguaje C/C++ junto con CUDA Runtime API

Arquitectura Hardware y Software

GPU = N * Streaming Multiprocessors (SMs)
SM = 8 * Streaming Processors (SPs)



Arquitectura Hardware y Software

- Los procesadores (SPs)
  Realizan operaciones escalares sobre enteros/reales 32 bits
  Ejecutan threads independientes pero todos deberían ejecutar la instrucción leída por la *Instruction Unit* (IU) en cada instante:
  Single Instruction Multiple Thread (SIMT), explotación de paralelismo de datos y, en menor medida, de tareas
- Los threads son gestionados por el hardware en cada SM:
  Creación y cambios de contexto con coste despreciable
  Se libera al programador de realizar estas tareas

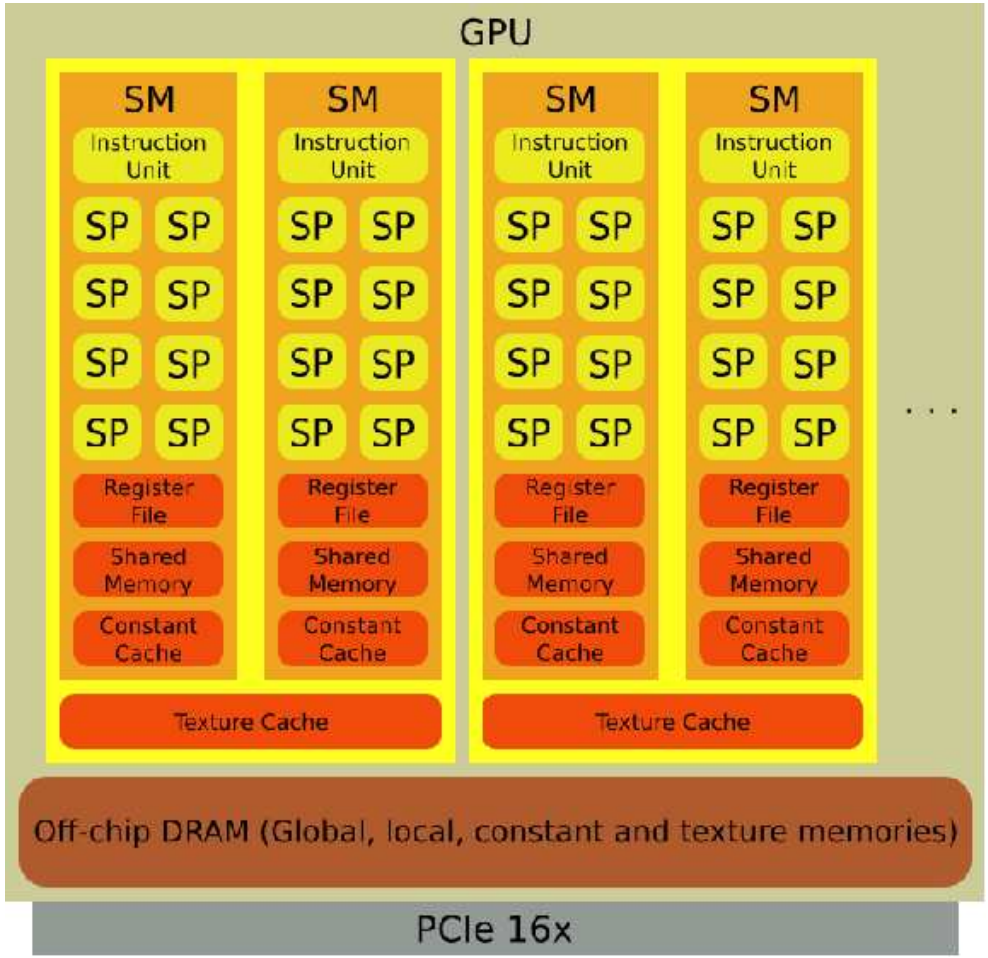
Arquitectura Hardware y Software

- Las partes del código paralelizadas para ejecutarse en la GPU se denominan *kernels*
- Un kernel descompone un problema en un conjunto de subproblemas y lo mapea sobre un *grid*
  Grid: vector 1D o 2D de *thread blocks*. Cada thread block tiene su BID (X,Y) dentro del grid
  Thread blocks: vector 1D, 2D o 3D de threads. Cada thread tiene su TID (X,Y,Z) dentro de su thread block
- Los threads utilizan su BID y TID para determinar el trabajo que tienen que hacer (SPMD)

Arquitectura Hardware y Software

- Las partes del código paralelizadas para ejecutarse en la GPU se denominan *kernels*
- Un kernel descompone un problema en un conjunto de subproblemas y lo mapea sobre un *grid*
  Grid: vector 1D o 2D de *thread blocks*. Cada thread block tiene su BID (X,Y) dentro del grid
  Thread blocks: vector 1D, 2D o 3D de threads. Cada thread tiene su TID (X,Y,Z) dentro de su thread block
- Los threads utilizan su BID y TID para determinar el trabajo que tienen que hacer (SPMD)

Modelo de memoria

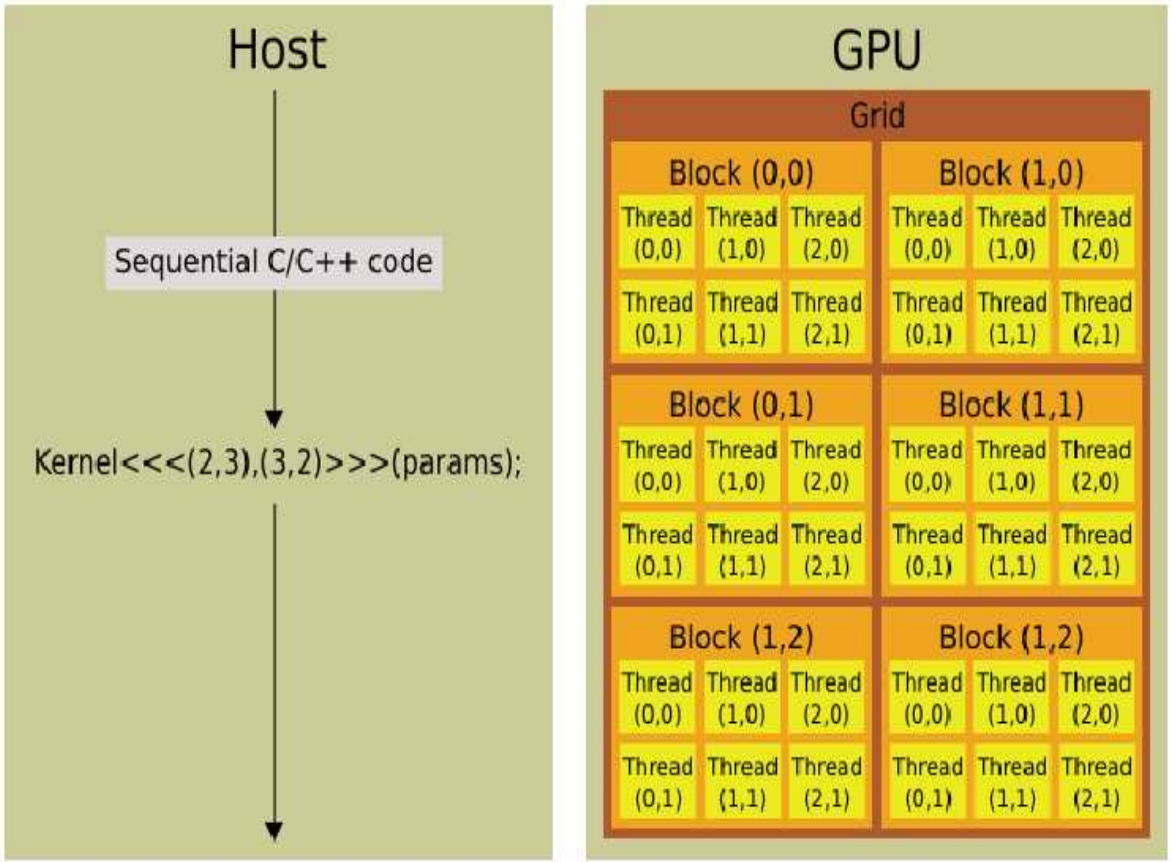


Ejemplo

Cálculo de $y = \alpha x + y$ (saxpy):

```
void saxpy_serial(int n,float *y,float alpha,float *x) {
  for(int i=0;i<n;i++)
   y[i]=alpha*x[i]+y[i];
}
/* Llamada código secuencial */
saxpy_serial(n,y,2.0,x);
__global__ /* Código GPU */
void saxpy_parallel(int n,float *y,float alpha,float *x) {
  int i=blockIdx.x*blockDim.x+threadIdx.x;
  if (i<n) y[i]=alpha*x[i]+y[i];
}
/* Llamada código paralelo desde código CPU */
int nblocks=(n+255)/256;
saxpy_parallel<<<nblocks,256>>>(n,y,2.0,x);
```

Ejecución


Host
Sequential C/C++ code
Kernel<<<(2,3),(3,2)>>>(params);
GPU
Grid
Block (0,0)
Block (1,0)
Block (0,1)
Block (1,1)
Block (1,2)
Block (1,2)
Thread (0,0)
Thread (1,0)
Thread (2,0)
Thread (0,1)
Thread (1,1)
Thread (2,1)

Prácticas

- Realizar las pácticas indicadas en los apartados de Sistemas, Paralelismo anidado, Programación híbrida, Paralelismo multinivel, y Computación heterogénea en memoria compartida y cluster.
- Se pueden usar los ejemplos que se encuentran en `/project/FORMACION`.
- Primero hacer la Hibr.1
- A continuación, cada uno puede realizar los ejercicios propuestos según el orden en que más le interesen, pero se sugiere realizarlos de menor a mayor valor de dificultad estimado (el número que aparece en cada ejercicio).
- Los que no puedan asistir a esta sesión tendrán que realizar al menos todos los ejercicios 1 y enviar un resumen del trabajo hecho. Consultas y envío a domingo@um.es.