COMPUTACIÓN DE ALTAS PRESTACIONES

TEMA 1

Introducción

1º ¿Qué es la Computación de altas prestaciones (HPC)?

Def. La computación de altas prestaciones (High Performance Computing) es la búsqueda del aumento de la potencia de cálculo con el fin de poder resolver problemas complejos. Para hacer esto se optimiza el uso de los recursos mediante la implementación de diversas técnicas de paralelismo.

– Actualmente se utilizan tecnologías computacionales como los clusters o los supercomputadores (todo en base al paralelismo, ya sea en una única máquina o mediante la agrupación de muchas). Sin embargo, seguir aumentando la capacidad de computo es cada vez más complicado debido a que el hardware a utilizar se vuelve cada vez más complejo.

– Un ejemplo de HPC es la creación de los primeros computadores vectoriales, los cuales son capaces de realizar una misma operación sobre un grupo completo de datos de forma simultánea. Actualmente los computadores de uso general ya integran la capacidad de hacer esto mediante las llamadas operaciones SIMD (Single Instruction Multiple Data).

1.1 Problemas abarcados por la HPC

- **Problemas de computo intensivo:** Se trata de un único problema que requiere realizar una gran cantidad de operaciones.
  - El objetivo es resolver dicho computo en el menor tiempo posible, y para llevarlo a cabo, se distribuye todo el trabajo requerido por dicho problema entre múltiples CPUs con el objetivo de reducir el tiempo de ejecución, de este modo, cada proceso realizará una parte del trabajo.
  - Este tipo de aplicaciones suelen requerir que las distintas CPUs intercambien información, lo que puede llegar a generar tiempos de espera debido a las interdependencias de las CPUs respecto a los mismos datos.
- **Problemas de memoria intensiva:** Se trata de un único problema que requiere una gran cantidad de memoria para poder emplazar los datos a utilizar.
  - Estos problemas requieren una gran cantidad de memoria, por lo que se utiliza una gran cantidad de nodos conectados mediante una red de intercomunicaciones. El objetivo es administrar correctamente el uso de las diferentes CPUs con el fin de operar utilizando los niveles de memoria más rápidos, evitando hacer accesos a los niveles más lentos.
  - En estos sistemas el acceso a memoria no es uniforme, lo que quiere decir que el tiempo necesario para obtener un mismo dato depende de la unida de computo que vaya a realizar dicho acceso. Para solventar esto es necesario mapear la información con el objetivo de emplear la la unidad de computo con el acceso más eficiente.
- **Problemas de datos intensivos:** Se trata de un único problema que requiere una gran cantidad de datos a procesar.
  - Para esto se divide el grupo de datos a procesar en múltiples subgrupos, los cuales son asignados a las diferentes CPUs, de modo que en cada uno de estos subgrupos se debe realizar el mismo trabajo. En estos problemas, el movimiento rápido de los datos en el disco es más importante que las comunicaciones entre las distintas unidades de computo.
  - Una solución común es aplicar el modelo de programación MapReduce, el cual consta de:
    - **Función Map:** Todos los nodos realizan la misma operación sobre los datos asignados.
    - **Función Reduce:** Se recolectan los resultados de todas las operaciones Map y se opera con los mismos. Esto conlleva que la información debe almacenarse de forma redundante.
- **Problemas de alto rendimiento:** Se trata de una gran cantidad de problemas que deben ser procesados de forma simultanea.

- El objetivo es poder reducir el tiempo de computo general, para lo cual se divide el grupo de problemas en diversas subgrupos y se distribuyen entre las diferentes CPUs, de modo que cada porción de trabajo se realiza en un hilo de ejecución separado. De esta manera, las diferentes tareas se ejecutan simultáneamente.
- En este tipo de problemas, la comunicación requerida entre las distintas CPUs es muy escasa o inexistente y se pone un mayor énfasis en el rendimiento global antes que en el de un problema específico. Un ejemplo de esto es la computación en la nube.

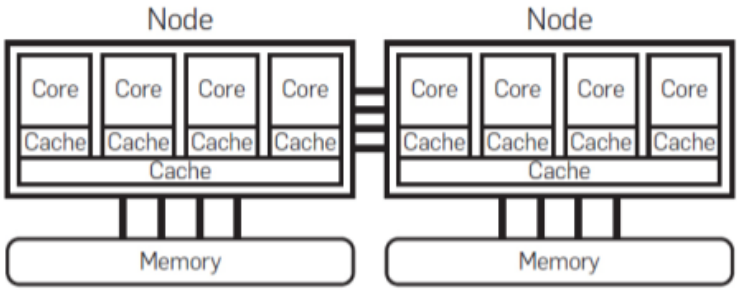
– Todos los problemas a tratar por la HPC requieren de la combinación de una gran cantidad de CPUs y módulos de memoria. Actualmente se utilizan clusters para esto, cuyo objetivo es poder resolver los cuatro tipos de problemas mediante un mismo hardware, el cual es fácilmente actualizarle.

1.2 La HPC en los computadores de carácter general actuales.

– Los computadores domésticos también integran múltiples núcleos por cada socket, donde todos ellos pueden acceder a una misma memoria caché compartida y tardan el mismo tiempo en poder obtener un mismo datos. Esto es los que se conoce como multiprocesadores simétricos (SMP).

– Los computadores mas potentes cuentan con múltiples socket, los cuales tiene varios núcleos y su propia memora memoria caché compartida, de modo que cada uno de ello se comporta como un mismo nodo.

– Cada nodo tiene acceso a una memoria compartida, lo que hace que la máquina se comporte como un multiprocesador de memoria compartida. Estos computadores tienen una arquitectura de memoria no uniforme (NUMA), lo que hace que el acceso a mismo dato pueda tardar un tiempo diferente dependiendo del nodo que quiera acceder al mismo.



– Podemos dividir los computadores actuales en dos grupos:

- **Máquinas de memoria compartida:** Tienen la ventaja de que son más fáciles de programar gracias a que únicamente contamos con un mismo bloque de memoria al cual acceden todos los nodos, pero su potencia es menos escalable debido a un mayor coste.
- **Máquinas de memoria distribuida:** Tienen la ventaja de que su coste de escalabilidad es mucho mayor, ya que únicamente necesitamos seguir aumentando el número de nodos, sin embargo, su programación es mucho más complicada debido a que cada nodo accede a una memoria diferente y esto puede conllevar implementar comunicaciones.

2º Introducción al paralelismo

– Cuando hablamos de un implementación secuencial de un programa en un computador con un solo núcleo, todas las instrucciones de dicho programa se ejecutarán de forma secuencial una detrás de otra, por lo que únicamente se podrá ejecutar una instrucción simultánea. El tiempo de ejecución sea igual al sumatorio de cada tipo de instrucción por el tiempo de CPU necesario para ejecutarla.

$$Tcpu = \Sigma(n * Tej)$$

T_{CPU} = Tempo que tarda la CPU en ejecutar el programa
n = Numero de instrucciones de un tipo concreto
t_{ej} = Tiempo que tarda la CPU en ejecutar las instrucciones de un tipo concreto

– Podemos diferenciar dos tipos básicos de paralelismos:

- **Paralelismo a nivel de instrucción (ILP):** En un mismo ciclo de reloj de la CPU se pueden ejecutar una instrucción diferentes en cada una de las etapas de la línea de procesamiento (pipeline). Este paralelismo mejora rendimiento interno de la CPU y es ajeno al programador.
- **Paralelismo a nivel de CPU:** El programa se ejecuta simultáneamente en múltiples CPUs, para lo cual, este se divide en partes discretas que pueden ejecutarse de forma independiente. Este paralelismo esta a disposición del programador.

2.1 El Speed-up

Def. El Speed-up (aceleración) es una métrica muy utilizada para medir como de rápido es una aplicación paralela con respecto a la misma aplicación implementada de forma secuencial. Se mide partiendo el tiempo de ejecución obtenido si ejecutamos la aplicación de forma secuencial entre el tiempo de ejecución de la parte paralelizable de la misma cuando lo ejecutamos de forma paralela.

$$S = \frac{Ts}{Tpar}$$

S = Speed-up de la aplicación
Ts = Tiempo de ejecución del programa ejecutado de manera secuencial
Tpar = Tiempo de ejecución de la parte paralela

– El tiempo de ejecución de un programa es la suma entre el tiempo que tarde en ejecutarse su parte secuencial más el tiempo que tarde en ejecutarse se parte paralela. Teniendo en cuenta que teóricamente, el tiempo que tarda en ejecutarse un código paralelizado desciende de forma proporcional al número de núcleos empleados, también podemos indicar el Speed-op como:

$$Ts = Tsec + Tpar \qquad S(P) = \frac{Tsec + Tpar}{Tsec + \frac{Tpar}{P}} \qquad S(max) = \frac{Tsec + Tpar}{Tsec + \frac{Tpar}{\infty}}$$

S (P) = Speed-up de la aplicación con respecto al uso de "n" núcleos
Tsec = Tiempo de ejecución del programa ejecutado de manera secuencial
Tsec = Tiempo de ejecución de la parte secuencial
Tpar = Tiempo de ejecución de la parte paralela
P = número de núcleos empleados para la ejecución

- La escalabilidad de un programa nos indica como de buena es la eficiencia del mismo, es decir, lo cerca que se encuentra del Speed-up ideal. La escalabilidad también nos muestra como incrementa el rendimiento de dicho programa conforme aumenta el número de procesadores empleados.

- De este modo, podemos calcular la eficiencia de un programa ejecutado de forma paralela como:

$$Ef = \frac{S(P)}{P}$$

Ef = Eficiencia del programación
S(P) = Speed-updel programa ejecutado con P procesos
P = Número de procesos empleados para la ejecución del programa

- Aunque aumentando la escalabilidad del programa mejoramos la eficiencia del mismo, lo normal es no llegar a la eficiencia ideal (existen pequeñas excepciones). Esto es debido a que el Speed-up no tiene en cuenta otras acciones que se dan durante la ejecución del programa y que tienen impacto en el mismo.

- Un ejemplo de esto es el tiempo de CPU destinado a la comunicación entre procesos o la diferencia entre los tiempos de acceso de los diferentes núcleos a los distintos datos del programa guardados en memoria.

2.2 Ley de Ahmdal

- La ley de Ahmdal nos da a conocer cuanto se puede acelerar un determinado programa. Teniendo en cuenta que únicamente podemos paralelizar una parte del mismo, esta ley expresa el impacto de la parte secuencial del mismo con respecto a un programa que se ejecuta de forma paralela.

$$Smax(p) = \frac{1}{f + \frac{1-f}{P}}$$

Smax (p) = Speed-up máximo usando P núcleos
P = Número de procesadores
f = Fracción secuencial, porcentaje del programa que es secuencial

- Teniendo en cuenta esto, también podemos expresar la fracción secuencial de la siguiente manera:

$$f = \frac{Ts}{Tsec + Tpar} \qquad 1 - f = \frac{Tpar}{Tsec + Tpar}$$

f = Fracción secuencial, porcentaje del programa que es secuencial
T_S = Tiempo de ejecución del programa ejecutado de manera secuencial
T_{Sec} = Tiempo de ejecución de la parte secuencial
T_{Par} = Tiempo de ejecución de la parte paralela

- Debemos tener en cuenta que el porcentaje del programa que se ejecuta de forma secuencial tiene una gran peso sobre la aceleración máxima que podemos obtener al aumentar los núcleos dedicados a la ejecución del mismo.

- Si operamos con la ley de Ahmdal llegaremos a la conclusión de que lo normal es no poder alcanzar el speed-up máximo teórico para la ejecución de un determinado programa con el uso de número de CPUs concreto. Las principales causas de esto son:

- **Gestión de la paralelización:** El coste asociado a crear y gestionar el paralelismo tiende a escalar exponencialmente con el número de hilos utilizado.
- **Comunicación entre núcleos:** La comunicación entre los distintos núcleos utilizados para ejecutar el programa depende principalmente del hardware del equipo, sin embargo, el coste computacional asociado a esto se incrementa a medida que aumenta el número de núcleos.
- **Sincronización:** Depende del modelo de programación implementado en el programa y de la eficiencia del mismo.
- **Desbalanceo de carga:** Los núcleos que han terminado su porción de trabajo quedarán esperando al resto de núcleos a que terminen la suya. El tiempo que dichos núcleos esperan es tiempo que se desaprovecha la capacidad de computo del equipo.

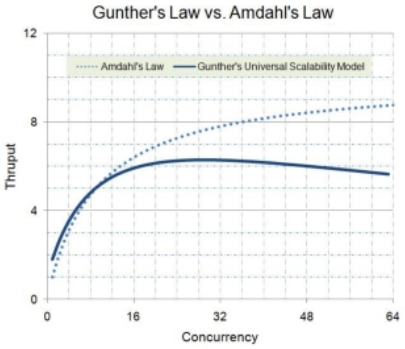
2.3 Ley de Gunther

– La ley de Gunther es una extensión de la ley de Amdal, la cual tiene en cuenta es coste computacional derivado de la comunicación entre los distintos núcleos utilizados.

– A medida que aumentamos el número de núcleos utilizados para ejecutar un programa, la aceleración del mismo se vuelve cada vez más escasa hasta llegar al momento en el que decrementa. Esto es debido a que coste de la comunicación entre los núcleos termine influyendo más que la pequeña porción de mejora que nos otorga introducir un núcleo nuevo en la ejecución.

– Para incrementar la aceleración debemos realizar dos cosas:

- Reducir la fracción del programa que se ejecuta de forma secuencial.
- Reducir el coste de computo asociado a la comunicación entre los distintos núcleos empleados para la ejecución del programa.



3º Elementos de los computadores paralelos

– Podemos diferenciar un total de tres tipos de elementos utilizados en la computación paralela que a día de hoy están implementados en los computadores de uso general:

- **A nivel Hardware:** Utilización de múltiples niveles de paralelismo (el ILP o el multicore), implementación de una jerarquía de memoria con diversos niveles, y el uso de una red de interconexión entre los distintos núcleos.
- **A nivel software:** Utilización de lenguajes de programación que permite el uso de paralelismo y programar de forma concurrente mediante el paso de mensajes.
- **Aplicaciones software:** Uso de algoritmos paralelos.

– Para el uso de la computación paralela a gran escala se utilizan los clusters, los cuales están formados por un conjunto de computadores conectados mediante una red local, de modos que funcionan como si fueran un multiprocesador a gran escala.

– El rendimiento de los clusters viene dado por la suma de todos los núcleos (existen varios núcleos por chip) que contienen en cada uno de sus nodos. Se entiende como nodo de un clusters a cada uno de los equipos que se encuentra conectado al mismo.

3.1 Ley de Gustafson

– La ley de Gustafson define que cualquier problema lo suficientemente grande puede ser paralelizable. Si aumentamos el número de procesadores destinados a la ejecución de un problema, también podremos escalar este en consecuencia.

$$S(n) = n - f(n-1)$$

S(n) = Speed-up para "n" procesadores
n = número de procesadores
f = fracción paralelizable del problema

3.2 Modelo de Roofline

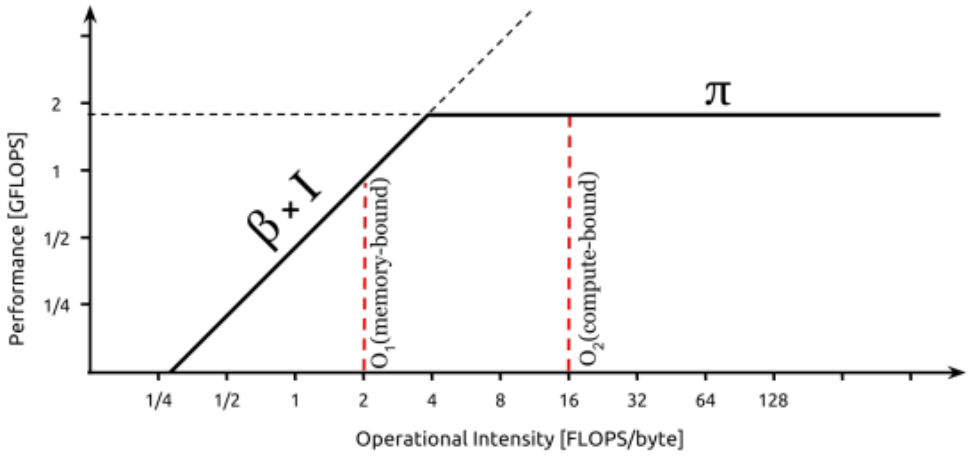
– El modelo de Roofline es un modelo intuitivo que nos puede dar una idea a cerca del rendimiento actual de la aplicación en base a evaluar el rendimiento de la aplicación en aplicaciones que realicen operaciones en punto flotante. Podemos identificar si el rendimiento de la aplicación se encuentra limitados por uno de los dos siguientes límites:

- **Límite de acceso a memoria (Memory Bound):** El rendimiento de la aplicación se encuentra limitado por el acceso a memoria a la hora de buscar o escribir datos en la misma. En este caso debemos mejorar la jerarquía de memoria y el acceso a los datos.
- **Límite de cómputo (Computer Bound):** El rendimiento de la aplicación se ve mermado por el potencial de cómputo del equipo. En este caso deberemos explotar la paralelización.

– El modelo se obtiene a base de aplicar un análisis de cuellos de botella sobre el equipo que ejecuta la aplicación, donde comparamos la potencia de cómputo del mismo con su ancho de banda. El modelo viene dado por un total de tres métricas:

- **π:** Rendimiento pico del equipo.
- **β:** Ancho de banda pico.

- **I:** intensidad aritmética, la cual viene dada por la fórmula **I = W / Q**, donde:
  - **W:** Número de operaciones de un kernel, es decir, operaciones aritméticas.
  - **Q:** Tráfico de memoria, es decir, número de bytes que se transfieren con las operaciones del kernel. Este valor es muy dependiente de las arquitecturas y la plataforma empleada.



– El valor de las métricas **π** y **β** depende una gran cantidad de características físicas del equipo en el que vayamos a aplicar el modelo. Concretamente, **π** lo podemos calcular mediante las características asociadas a la CPU mientras que **β** con las relacionadas con la memoria.

– El punto que marca el tipo de limitación que sufre el equipo en general es la intersección entre las líneas **π** y **β*I**. No existe un valor exacto que defina la frontera entre los dos tipos de limitaciones, pero cuanto mas a la izquierda se encuentre el punto, más sufrirá nuestro equipo de *Memory Bound*, mientras que cuanto más a la derecha se sitúe, más sufrirá por *Computer Bound*.

– En el caso de medir la limitación para la ejecución de una operación concreta en el equipo en el que hemos realizado el modelo, deberemos hacerlo en base a la intensidad aritmética asociada a la aplicación.

– Este modelo también nos permite comparar el rendimiento obtenido entre diferentes tipos de arquitecturas para la ejecución de una misma aplicación.