COMPUTACIÓN DE ALTAS PRESTACIONES

TEMA 2

Arquitecturas y aplicaciones paralelas

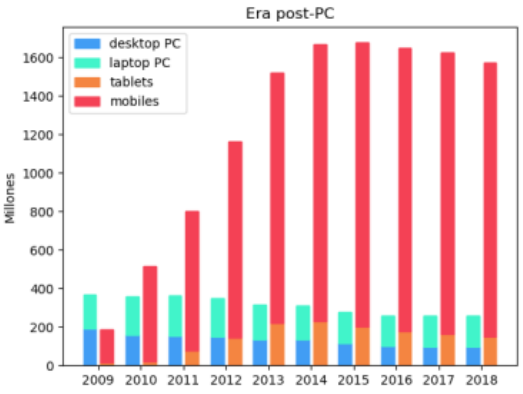
1º Historia de la arquitectura de computadores

1.1 Surgimiento de las primeras ISAs (1960 -1975)

- En 1960 IBM era la empresa más importante dentro del ámbito de la computación y poseía cuatro gamas distintas de computadores, las cuales abarcaban desde modestos ordenadores para pequeños negocios hasta computadores de altas prestaciones destinados al ámbito científico.
- Estas cuatro gamas implementaban arquitecturas distintas, de modo unas utilizaban rutas de datos de 8 bits, otros llegaban a usarlas de 64 bits. Esto suponía un gran problema, debido a que las mejoras implementadas en una de las arquitecturas no podían trasladarse directamente a las otras.
- La solución adoptada fue implementar el mismo juego de instrucciones ISA para todos los computadores. Para que dichas instrucciones se pudieras ejecutar en las diferentes arquitecturas, el primer paso era convertirlas en una secuencia de µ-instrucciones mediante hardware, de modo que estas fueran compatibles con la arquitectura en la que se estaba ejecutando el juego ISA.
- Implementar el proceso de traducción del juego ISA conllevo la implementación de tablas de traducción cada vez más grandes y complejas dentro del procesador y la necesidad de utilizar memorias más grandes. Esto fue el origen de las arquitecturas de procesadores CISC.

1.2 Las arquitecturas CISC y RISC (1975 -2000)

- Tras el surgimiento e implementación de los primeros procesadores CISC, a mediados de los años 80s aparece una nueva tendencia de arquitectura, la cual se centra en la utilización de un repertorio de instrucciones más reducido y simples de utilizar. Surgiendo así las arquitecturas RISC.
- Este planteamiento hace que no sea necesario realizar la traducción previa a µ-instrucciones, de modo que la sección de memoria del procesador destinada a la implementación del interprete puede ser utilizada para dotar al procesador de una pequeña caché de instrucciones. Esto conlleva la implementación de registros más eficientes y rápidos en el procesador.
- Pese a las ventajas de los computadores RISC, Intel termina imponiéndose en el mercado con el uso de su arquitectura x86. Esta arquitectura, pese a utilizar un juego de instrucciones CISC, cuenta con un tamaño de transistor mucho más pequeño, el cual sigue reduciéndose y otorga a estos procesadores la capacidad de tener una mayor potencia ce cómputo.
- Esta tendencia cambiará con la aparición del primer Iphone y el surgimiento de un nuevo mercado móvil cada vez más importante, el cual provocará un descenso en las ventas de equipos de sobremesa. Esto provocó que la arquitectura x86 de Intel perdiera la supremacía del mercado y que los procesadores RISC obtuvieran cada vez una mayor importancia.



- Actualmente, Intel utiliza en sus procesadores instrucciones RISC, las cuales dependiendo de su complejidad, pueden traducirse a instrucciones CISC. Esto hace que ambos tipos de instrucciones convivan dentro de la misma arquitectura.

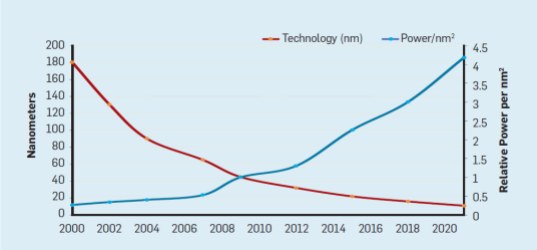
1.2.1 La arquitectura VLIW

- Contemporáneamente a la disputa entre las arquitecturas CISC y RISC, Intel desarrollo la arquitectura denominada VLIW, la cual se centraba en el uso de instrucciones de gran tamaño que se formaban mediante la búsqueda y empaquetamiento de instrucciones más sencillas. Esto tenía la ventaja de que los procesadores contaban con un pipeline más sencillo y de menor consumo.
- Estos procesadores estaban dotados de un procesamiento de instrucciones estrictamente paralelo denominado EPIC, el cual permitía la ejecución simultánea de 2 paquetes de instrucciones, cada uno con un máximo de 6 instrucciones individuales. De este modo logramos que en un mismo ciclo se ejecuten todas las instrucciones empaquetadas en una misma instrucción.
- La implementación del paralelismo recae en el compilador, debido a que tiene que realizar la búsqueda de instrucciones a empaquetar. Esto hace que la arquitectura tenga un mayor rendimiento en códigos estructurados (los cuales trabajan en punto flotante), pero que recaigan en trabajo con enteros y en códigos menos predecibles.
- Finalmente, el proyecto (denominado Itanic), termino fracasando aún utilizando la arquitectura en aplicaciones con saltos sencillos y predecibles.

1.3 El surgimiento de la Power Wall (2000 – 2002)

Def. **Ley de Moore:** El número de transistores que puede integrarse en un chip se duplica cada año. A partir de los años 70 la duplicidad se aplica cada aproximadamente 18 meses.

- A principio de los años 2000 la diferencia entre la densidad de transistores predicha por la ley de Moore y la realidad empieza a aumentar. Esto es debido a que aumentar el número de los transistores de un chip provoca que el consumo de energía en dicha área sea más elevado y por lo tanto que aumente la temperatura. En este momento se marca como uno de lo los principales objetivos llegar a tener una computación energéticamente eficiente.
- El otro componente que contribuye al aumento de la energía consumida en los procesadores es el ritmo de trabajo de los mismos, es decir, la frecuencia de reloj. Esto ha llevado a que actualmente nos encontremos estancados en lo que a frecuencia de reloj se refiere, provocando que los nuevos procesadores no puedan aumentar su frecuencia con respecto a los anteriores (Power Wall).



- Desde 1986 hasta el año 2002 el incremento de la paralelización en los procesadores se llevo a cabo explotando el paralelismo a nivel de instrucción. Esto provocó la creación de pipelines cada vez mas largos con un número de instrucciones por ciclo cada vez mayor.
- Aunque se trataba de procesadores con una gran potencia gracias a su capacidad de paralelismo, eso no se traducía en la practica debido a que tener un pipeline de instrucciones tan largo provocaba que cuando se realizaba un salto indeterminado, una gran cantidad de procesamiento fuera desechado.
- Desechar las instrucciones que se están ejecutando en un pipeline no solo conlleva perdida de tiempo de ejecución, sino un aumento necesario en el consumo del procesador. Para evitar esto se vuelve necesario el uso de predictores de instrucción muy potentes y fiables.

1.4 La época Multicore (2002 - 2012)

– Los problemas derivados del surgimiento de la Power Wall y el excesivo uso de la ILP, conlleva el cambio de enfoque desde el uso de un núcleo muy potente y con alta capacidad de paralelismo al uso de varios núcleos más sencillos dentro de un mismo procesador. Esto desplaza el trabajo asociado a la explotación del paralelismo del hardware al propio programador.

– El uso del multicore no acata el problema de la computación eficiente, en parte debido a que todos los núcleos consumen energía, ya estén trabajando o en espera. Por otra parte, el incremento del número de núcleos también contribuye al incremento del consumo del procesador, lo cual lleva a limitaciones en el aumento del rendimiento.

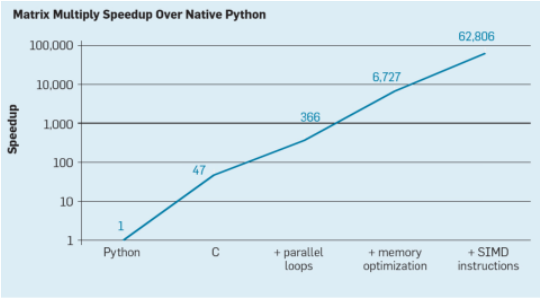
1.5 El surgimiento de nuevas oportunidades

– Los enfoques actuales para aumentar el rendimiento abarcan distintas áreas de acción, tanto software como hardware. Sin embargo, todo se traduce en que se construirán cada vez procesadores más complejos y será objetivo del programador aprovechar el paralelismo de los mismos.

1.5.1 Nuevas oportunidades a nivel software

– Las técnicas para crear software con tipos y gestión de memoria dinámicos suelen ser ineficientes. Por otra parte, el lenguaje utilizado influye drásticamente en el rendimiento del programa ejecutado.

– El uso de lenguajes interpretados como python reduce enormemente la eficiencia debido a que el uso de un interprete imposibilita la predicción de instrucciones.



1.5.2 Nuevas oportunidades a nivel hardware

– El nuevo enfoque avanza hacia la implementación de diversas Arquitecturas de Dominio Específico (DSA), también conocidas como aceleradores, dentro del procesador. El uso de DSAs en cada dominio aumenta enormemente el rendimiento y la eficiencia energética. Dentro de estas DSA podemos ver que se llegar a realizar ciertas implementaciones:

- **Respecto al paralelismo:**
  - Se implementan procesadores SIMD, los cuales son más eficientes que los MIMD, aunque sean menos flexibles debido a que las instrucciones se ejecutan de forma secuencial.
  - Se utiliza la arquitectura VLIW debido a que es más eficiente que un superescalar fuera de orden. Esto es gracias a tener pipelines más cortos, no necesitamos hacer predicciones y las instrucciones son más simples.
- **Respecto a la memoria:**
  - Debido a que el acceso a memoria es muy costoso, se reduce el tamaño de caché para implementar mayores sumadores. Esto implica tener buenos predictores con los que reducir los accesos a memoria y aprovechar mejor la caché.
- **Respecto a la precisión:**
  - El trabajo con redes neuronales no precisa de todo el rendimiento de la CPU, puesto que para la inferencia es suficiente con enteros de 16 bits y para el entrenamiento con fp de 32 bits.

1.5.3 Nuevas oportunidades arquitectónicas

- **GPUs:** Los aceleradores gráficos utilizan menos caché y una cantidad muy superior de unidades aritmético-lógicas.
  - **Ej.** Nvidia utiliza una gran cantidad de núcleos e hilos de ejecución en sus tarjetas gráficas.
- **TPUs:** Se trata de pequeños procesadores para tareas específicas.
  - **Ej.** Circuitos integrados especializados (ASIC) en para operaciones con matrices de 8 bits.
- **FPGAs:** Contienen una gran cantidad de unidades funcionales y es objetivo del programador definir las conexiones entre las mismas. Esto permite obtener un gran rendimiento gracias a hacer un circuito concreto para un trabajo determinado, pudiendo prescindir de etapas de ejecución en el pipeline, además de tener un bajo consumo.
  - **Ej.** Microsoft ha desarrollado productos para redes neuronales con FPGAs mediante Azure.
- **CPUs:** Dotación a los procesadores de carácter general con operadores vectoriales para la realización de tareas más específicas.
  - **Ej.** Intel ofrece procesadores SIMD con baja precisión para su uso en redes neuronales.

– Con el paso del tiempo, los distintos aceleradores han terminado utilizándose para ámbitos distintos a los que fueron concebidos. Esto plantea el reto de poder adaptar el hardware específico para su uso en ámbitos más generalistas.

1.6 La especialización en µarquitectura

– La tendencia actual se basa en implementar dentro de un computador de uso general una gran cantidad de DSAs especializadas, de modo que se utilicen para realizar aquellos cálculos para los que aportan un incremento en el rendimiento respecto a si los realizara la propia CPU.

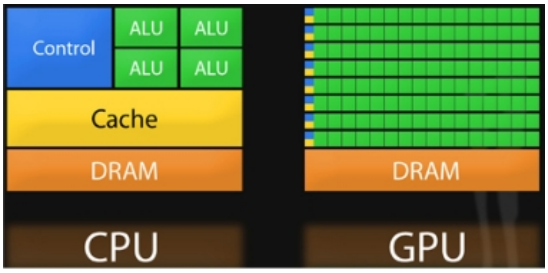
1.6.1 GPUs

– Si analizamos la arquitectura de una CPU general podemos ver como una gran partes de los transistores están destinados a la implantación de las memorias caché y a la sección de control. Esto quiere decir que únicamente tienen una pequeña parte de las mismas destinadas a la realización de operaciones aritmético-lógicas (ALUs), lo cual se debe al propósito generalista de las CPUs.

– Por otra parte, si observamos una GPU actual podremos ver como casi todos los transistores de la misma están destinados a la implantación de unidades aritmético-lógicas. Estas unidades son mucho más simples que las implementadas en una CPU, pero debido a la gran cantidad de las mismas, la CPU consta de una mayor capacidad de paralelización.

– Los núcleos de una GPU se agrupan en bloques denominados denominados StreamMultiprocessors, cada uno de los cuales cuenta con una sección de memoria caché compartida, la cual solo pueden acceder los núcleos que lo componen. Cuando se ejecuta una instrucción esta se lanza en todos los núcleos que forman el SM, lo cual se conoce como SIMT (Simple Instruction Multiple Thread).

– Para mitigar el coste temporal de los accesos a memoria las CPUs implementan los distintos niveles de memorias caché, sin embargo las GPU a penas cuentan con este tipos de memoria, por lo que en su lugar utilizan diversas técnicas.

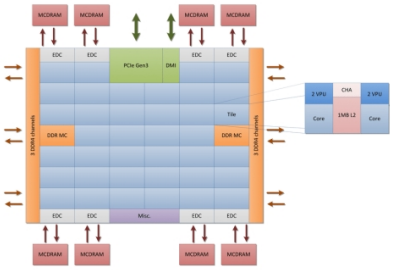


– El Multihilo de grado fino (fine-grained multithreading) consiste en tener una gran cantidad de hilos de ejecución al mismo tiempo, muchos más que el número de núcleos de la GPU. Cuando un hilo quiere realizar un acceso a memoria este deja de ser ejecutado y pasa al final de la cola de procesos, de modo que cuando le vuelve a ejecutarse lo normal es que ya haya obtenido el dato buscado.

1.6.2 Xeon-KNL

– Se trata de una arquitectura de Intel (proyecto cerrado en 2018) basada en el uso de núcleos con un pipeline muy corto (lo cual permite tener runa mayor cantidad de núcleos) y que implementan una unidad de computo vectorial, permitiendo así conseguir una gran cantidad de paralelismo.

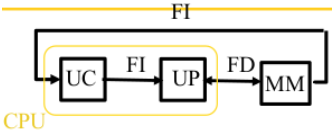
– Estos procesadores también contaban con una caché intermedia entre los niveles de caché privadas o compartidas parcialmente entre los núcleos y la memoria principal. Esta caché esta dotada de un gran ancho de banda y puede ser programable, de modo que el usuario puede decidir si utilizarla de forma privativa, compartida o híbrida.



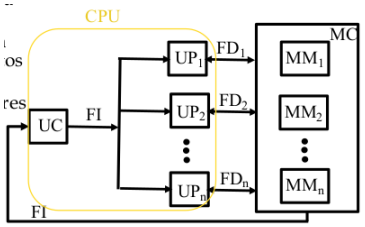
2º Clasificación de Flynn

– La clasificación de Flynn se basa en organizar los diferentes arquitecturas dependiendo del hardware empleado para atender los flujos de datos e instrucciones. Estas son:

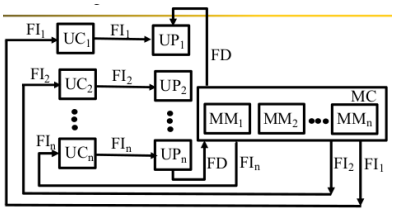
- **SISD (Simple Instruction Simple Dato):** Las instrucciones se ejecutan secuencialmente, aunque pueden estar solapadas en distintas etapas de ejecución y se pueden tener varias Unidades de Procesamiento (UP) bajo el control de una misma Unidad de Control (UC). Es la arquitectura de los procesadores mononúcleo.



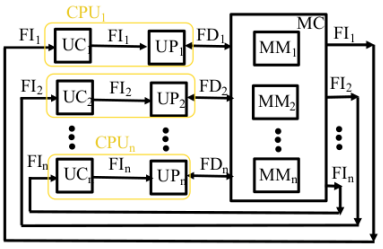
- **MISD (Simple Instruction Multiple Dato):** Tenemos múltiples Unidades de Procesamiento (UP) que reciben la misma instrucción emitida por una única Unidad de Control (UC) y aplican la misma sobre diferentes datos procedentes de diferentes flujos. Es la arquitectura de los procesadores matriciales o GPUs.



- **MISD (Multiple Instruction Simple Dato):** Tenemos múltiples Unidades de Procesamiento (UP) que reciben las instrucciones de Unidades de Control (UC) diferentes y operan sobre un mismo flujo de datos, de modo que la salida de una de una pasa a ser la entrada de la siguiente. Frente a los diversos desacuerdos, asumimos que este tipo de arquitectura nunca ha sido implementada.



- **MIMD (Multiple Instruction Multiple Dato):** Se trata se sistemas con más de un procesador, de modo que tenemos varias Unidades de Control (UC) y varias Unidades de Procesamiento (UP), las cuales operan sobre distintos flujos de datos. Estos procesadores son capaces de ejecutar varios programas de forma simultánea y se trata de sistemas multiprocesadores y multicomputadores.



3º Grados de paralelismo

- **Paralelismo Masivo:** Se refiere al hardware que comprende un sistema paralelo dado, el cual tiene una gran cantidad de núcleos y estos siguen aumentando conforme pasa el tiempo. El aumento de los número de núcleos incide en el aumento del rendimiento de la aplicación, sin hacer inciso sobre la comunicación.
- **Embarrassingly Parallel:** Referente a la ejecución de una gran cantidad de tareas similares pero que son independientes entre si, de modo que tienen muy poca o ninguna necesidad de comunicación.
- **Scalabilidad:** Capacidad de un sistema paralelo de demostrar un aumento proporcional de la aceración con respecto a la adición de núcleosg.

4º Pasos para llevar a cabo una paralelización

1. **Identificar que parte es paralelizable:** Es importante identificar el mayor porcentaje posible del código que es paralelizable, es decir, que ejecutado en un orden diferente aporta el mismo resultado. Para ello debemos realizar:
   - **Descomposición funcional:** Identificar que partes pueden ejecutarse en paralelo por que sus instrucciones no tienen dependencias con instrucciones anteriores.
   - **Descomposición de datos:** Normalmente aplicada se aplica a los bucles, de modo que si no hay dependencias entre los datos del mismo lo podemos paralelizar, de modo que cada iteración del mismo sea ejecutada por un núcleo diferente.
2. **Elegir la granularidad correcta:** La granularidad hace referencia al tamaño de los fragmentos en los que vamos a descomponer el problema y los cuales se van a ejecutar de forma paralela. Dependiendo del problema a abarcar, el paralelismo puede ser de grano más grueso, más fino o utilizarse un paralelismo multinivel, de modo que se empleen a la vez distintas ganularidades.
   - **Granularidad de grano grueso:** Produce menos penalizaciones de tiempo debido a los accesos a memoria (overheads) y menor número de comunicaciones y sincronización entre los distintos procesos, sin embargo, pueden producirse un mayor desbalanceo de carga.
   - **Granularidad de grano fino:** Se produce un menor desbalanceo de carga pero un mayor número de overheads y requiere de una mayor comunicación y sincronización.
3. **Elegir el modelo de paralelización a aplicar.** Debemos elegir el tipo de paralelización que queremos aplicar, es decir, el modelo. También podemos combinar la aplicación de varios modelos de forma simultánea.
4. **Evaluación de los resultados:** En el momento de evaluar los resultados tenemos que comprobar tanto que estos sean correctos como que se obtengan con un rendimiento adecuado.
   - **Resultados incorrectos:** Paralelización incorrecta, los errores dependen de la secuencia de ejecución, se producen interbloqueos y los errores no se pueden reproducir con facilidad.
   - **Rendimiento pobre:** Se produce un desbalanceo de carga, el porcentaje de serialización es muy elevado, poco aprovechamiento de la localidad de los datos y se produce false sharing.

– Cuando dos o más núcleos comparten un mismo dato y uno de ellos lo modifica, este debe de poner toda la sección de memoria caché en estado de inválido, de modo que los demás núcleos deban averiguar que dato ah sido modificado y preguntar por el valor actual del mismo.

– El false shasing se produce cuando una variable compartida se modifica continuamente y los núcleos pierden una gran cantidad de tiempo en comprobar la modificación y pedir el dato actualizado. Este problema de produce debido a la aplicación de una granularidad incorrecta.