UCAM
UNIVERSIDAD CATÓLICA DE MURCIA

Tema 1. Introducción a la programación paralela

1. Definición de computación paralela.

Desde el comienzo de los comienzos de los primeros lenguajes de programación, el software siempre se ha desarrollado para su procesamiento secuencial; esto es la ejecución ordenada de instrucciones. El objetivo del código era ser ejecutado por un computadora que contaba con una sola unidad de procesamiento. La Figura 1 muestra la ejecución de un programa en un procesador secuencial.

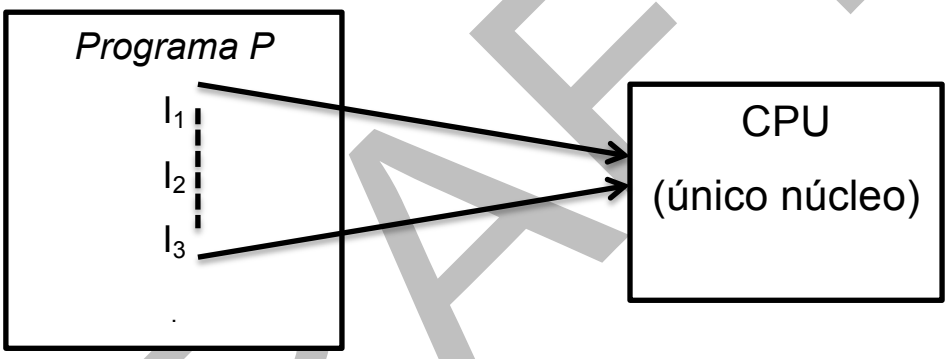

Figura 1. Ejecución de un conjunto de instrucciones en el orden establecido por el programador.

En programación paralela, por el contrario, el programa es dividido en partes para ser ejecutadas en varias CPU. Cada una de las partes en las que se divide el programa se ejecutan concurrentemente, procesando su conjunto de instrucciones de manera independiente en las diferentes unidades de procesamiento. La figura 2 muestra el enfoque de la programación paralela. El programa P se divide en una serie de **subprocesos o hilos**. Estos subprocesos tienen asignado parte del conjunto de instrucciones del programa P. Cada subproceso se ejecuta en un procesador diferente de manera concurrente.

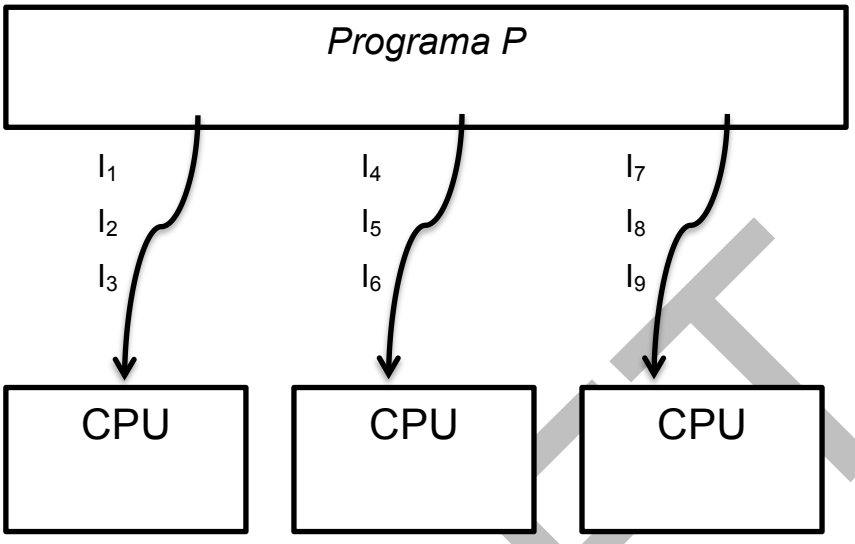

Figura 2. Ejecución de diversos subconjuntos del programa P en distintos procesadores.

El objetivo final de la programación paralela es acelerar el tiempo de ejecución del procesamiento secuencial. Este es el fin último de la programación paralela, y para ello, se deben considerar todas las capas del pensamiento computacional, desde el diseño de algoritmos que solucionen el problema tratado, la elección del lenguaje de programación que implemente dicho algoritmo, la traducción a lenguaje máquina mediante un compilador, así como el aprovechamiento de todas la arquitectura ofrecida por el procesador. Los programadores paralelos, por tanto, deben conocer en profundidad tanto los programas del sistema que permiten la programación como las arquitecturas donde se ejecutan.

2. Aplicaciones de la computación paralela

Históricamente la computación paralela, también conocida como computación de altas prestaciones (*High Performance Computing*), se ha considerado como la única herramienta para la resolución de los problemas mas complejos del campo de la ciencia y la ingeniería. Para resolver este tipo de problemas los gobiernos y grandes instituciones de los investigadores los ordenadores mas potentes del mundo. Estos ordenadores están clasificados en el top500[1], que reúne los 500 ordenadores mas potentes de la actualidad. Esta lista es actualizada dos veces al año en dos de las conferencias mas prestigiosas de supercomputación. En la figura 3 se muestra la construcción del supercomputador BlueGene/L que estuvo en el puesto número 1 desde Noviembre de 2004 hasta Noviembre de 2007.

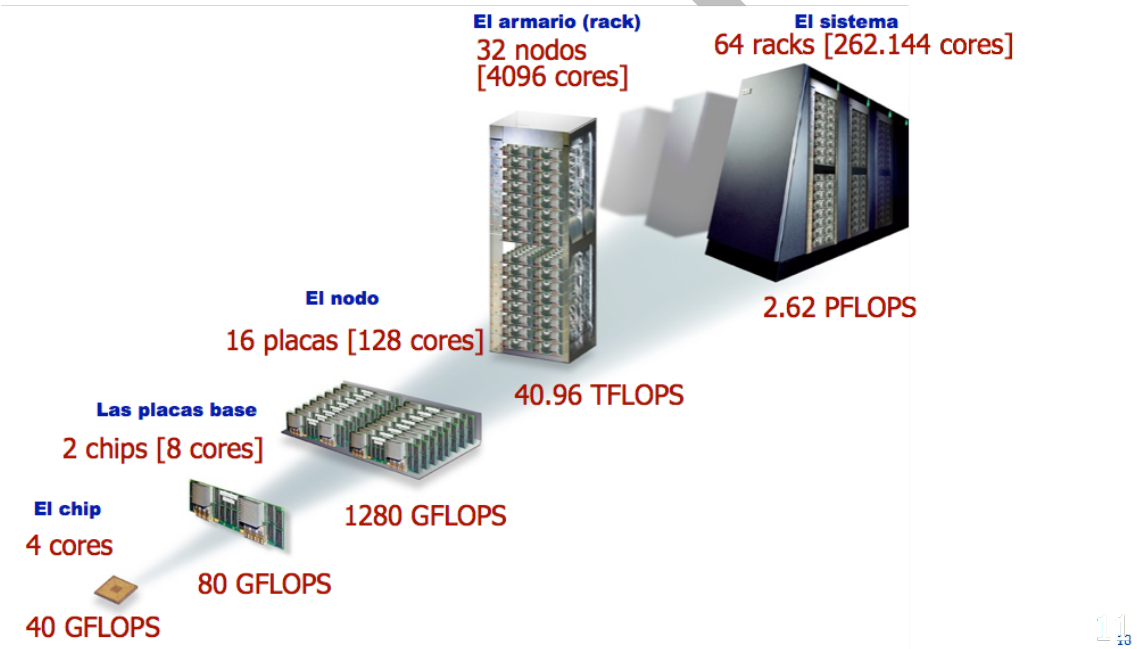

Figura 3. Organización del supercomputador de BlueGene/L perteneciente al Lawrence Livermore National Laboratory in Livermore, California, USA.

Este tipo de máquinas esta compuesta por procesadores multinúcleo, en este caso de hasta 4 núcleos o *cores,* embebidos en placas de dos sockets (total de 8 cores). Cada nodo incluye hasta 16 placas dando un total de 128 cores

[1] Top500.org: http://top500.org/

por nodo. Finalmente estos nodos se incluyen en armarios de hasta 32 nodos donde se les suministra la energía, refrigeración, conexión y monitorización que necesitan para poder funcionar.

La potencia de un ordenador se calcula en instrucciones en punto flotante por segundo que son capaces de desarrollar, **FLOPS** (*Floating Point Operations per Second*). Se considera una instrucción en punto flotante como aquella operación aritmética que trabaja con números reales. Estas operaciones tiene una precisión limitada que viene determinada por la implementación del estándar IEEE 754. Las operaciones en simple precisión utilizan una palabra de 32 bits para representar sus operandos (ver Figura 4).

Figura 4. Composición de elementos en simple presión (32 bits)

La potencia del supercomputador mas potente en la actualidad está entorno a los 33 PetaFLOPS ($33x2^{50}$ operaciones en punto flotante por segundo). Pero, ¿Qué potencia representa un petaFLOPS? Es difícil de precisar esto pero para ilustrar la potencia de un petaFLOPS, imaginad que le diésemos una calculadora científica a cada uno de los siete mil millones de habitantes del planeta, y éstos estuviesen haciendo cálculos ininterrumpidamente. Una estimación es que tardarían aproximadamente media vida (37 años y medio para ser exactos) en realizar todas las cuentas que un ordenador de un PFLOPS es capaz de completar en un solo segundo. El supercomputador más potente del planeta alcanza los 33.8 PFLOPS ejecutando un conjunto de aplicaciones (también conocido como *benchmark*) denominado LINPACK (el pico teórico de rendimiento se situaría en 54.90 PFLOPS). Esto significa que la

humanidad entera debería trabajar sin descanso durante 611 años para llevar a cabo todas las cuentas que él hace en un segundo. O visto de otra manera que él completa en 0.016 ns. lo que una persona realiza en toda su vida.

El único modo de aprovechar todo el rendimiento que ofrecen estos ordenadores es desarrollando aplicaciones paralelas que se ejecuten las aprovechando todos los recursos disponibles (cores, unidades vectoriales, etc), extrayendo así su máximo rendimiento. Si bien es cierto, la accesibilidad a estos centros es muy limitada, siendo casi imposible conseguir la dedicación exclusiva para una aplicación, el desarrollo de un programa paralelo es incuestionable para poder obtener al menos parte de su potencia. Algunos de los campos de aplicación donde se necesita esta ingente cantidad de cálculo incluyen disciplinas de rigurosa actualidad como:

- **Cambio climático**: Donde se están utilizando los patrones de cambios atmosféricos para predecir con exactitud los fenómenos alterados del tiempo.
- **Genoma**: En el secuenciamiento de la estructura del ADN del código genético.
- **Nanotecnología y química**: Para crear nuevos compuestos moleculares con propiedades microscópicas y materiales más resistentes a la fuerza y al calor.
- **Geología y sismología**: Utilizado en el estudio las propiedades geotécnicas y la predicción de movimientos sísmicos.
- **Ingeniaría electrónica**: Diseño de circuitos con más nivel de integración, más rápidos y con menos calentamiento del chip.
- **Prospección petrolífera**: Debido al agotamiento de los combustibles fósiles se ha hecho patente la necesidad de búsqueda de nuevos yacimientos para la obtención de petróleo y gas.
- **Buscadores Web**: Con la proliferación incesante de Internet en la última década se necesita una gran cantidad de recursos de cómputo que para rastrear e indexar todos los contenidos publicados en la red.

- **Diagnóstico por imagen**: El avance de la medicina requiere de sistemas de imagen capaces de diagnosticar con precisión y sin errores la existencia de enfermedades.
- **Descubrimiento de nuevos fármacos**: En el diseño de nuevos medicamentos eficaces con los mínimos efectos secundarios.
- **Modelado económico y financiero**: Las transacciones financieras realizadas en un mundo globalizado pueden poner en riesgo la economía mundial. El análisis complejo de estas interacciones de agentes puede prevenir una posible crisis económica.
- **Gráficos y realidad virtual**: Con el fin de conseguir imágenes realistas en el mundo de la infografía y los videojuegos.

3. Tipos de procesadores paralelos.

La predicción ampliamente conocida realizada por Gordon E. Moore en 1965, conocida como, la "Ley de Moore" [Moore, 1965], que esencialmente establece que el número de transistores incluidos en un circuito integrado se duplica cada dos años, ha conducido históricamente el diseño de microprocesadores. Hoy en día, más de 1. 4 Billones de transistores están disponibles en un solo chip [Riedlinger et al., 2011], y los arquitectos de computadores son los responsables de diseñar microprocesadores eficientes que aprovechen ese ingente número de transistores, respetando las leyes físicas de las arquitecturas basadas en silicio.

Existen dos métricas fundamentales para establecer el rendimiento de los procesadores [Garland y Kirk, 2010]. En primer lugar, la latencia de una tarea (o latencia) que calcula el tiempo transcurrido entre la iniciación y la finalización de una tarea determinada. En segundo lugar, el rendimiento total (*throughput*) que evalúa la cantidad de trabajo realizado por unidad de tiempo. Aunque las arquitecturas informáticas han calibrado las compensaciones entre la latencia y la optimización de rendimiento, ya que la mejora de una de ellos podría degradar la otra, la industria de los procesadores ha enfatizado

tradicionalmente una sobre la otra, dependiendo de la naturaleza de las cargas de trabajo que espera desarrollar. Este equilibrio también tiene otras consecuencias importantes más allá de la rendimiento de las aplicaciones, como por ejemplo, el consumo de energía [Grochowski et al., 2004].

Sobre la base de ambas métricas indicadores, la industria del procesador ha desarrollado dos principales tendencias en el diseño de procesadores. Las arquitecturas orientadas a latencia, cuyo principal objetivo es la reducción del tiempo de ejecución para una carga de trabajo, y las arquitecturas orientadas a rendimiento, cuyo objetivo es aumentar el número de tareas realizadas por unidad de tiempo.

A diferencia de los primeros tiempos de la informática, en las últimas dos décadas el corazón de las antiguas arquitecturas ha sido elaboradores de productos básicos de varios vendedores, tales como Intel®, AMD®, IBM®, ARM®, o Sun®. Estos microprocesadores son generalmente incluidos en sus respectivas soluciones de gama alta, y son comúnmente llamados microprocesadores de alto rendimiento. En la actualidad, la evolución de estos microprocesadores los ha convertido en arquitecturas multi-núcleo; es decir, microprocesadores que contienen varios núcleos de ejecución en el mismo chip.

Las arquitecturas orientadas a rendimiento, sin embargo, han seguido una trayectoria diferente [Garland y Kirk, 2010], centrándose principalmente en el aumento del rendimiento de ejecución de cargas de trabajo con abundante paralelismo. Es digno de mención destacar que el rendimiento se ha entendido tradicionalmente, desde el punto de vista de la arquitectura del ordenador, como la cantidad de tareas independientes realizadas por diferentes usuarios completados por unidad de tiempo, lo que conduce a la reducción del tiempo total de ejecución. Por ejemplo, un servidor web es una plataforma en la que se ejecutan las tareas de diferentes usuarios del sistema. El aumento del rendimiento en este tipo de entornos se ha conseguido tradicionalmente

añadiendo más procesadores, incrementando el trabajo total que es capaz de procesar el servidor en cada instante de tiempo; sin embargo esta estrategia no reduce el tiempo de ejecución de una tarea independiente.

Las arquitecturas orientadas a rendimiento se han convertido, sin embargo, en plataformas capaces de reducir el tiempo de ejecución de una sola tarea con abundante carga de trabajo, mediante la ejecución simultánea de muchos hilos que pertenecen al mismo proceso. Para conseguir esto, estas arquitecturas dedican su área de silicio a tener un ingente número de núcleos mucho más sencillos que los procesadores tradicionales, con la expectativa, también de duplicar el número de núcleos con cada generación de semiconductores. Un ejemplo actual de esta tendencia son las Unidades de Procesamiento Gráfico de NVIDIA (GPU, *Graphics Processing Units*), que contiene hasta 2880 núcleos muy simples, cada uno de los cuales permite un alto número ejecución de hilos, cuyo procesamiento es en orden y que comparten su unidad de control y la caché de instrucciones con otros 192 núcleos, etc [NVIDIA Corporation, 2013]. Los procesadores orientados a rendimiento, también conocidos como *manycores* por el ingente número de cores que poseen sus arquitecturas han liderado la carrera del número de operaciones en punto flotante por segundo, así como el ancho de banda de sus memoria (ver Figura 5 y 6).

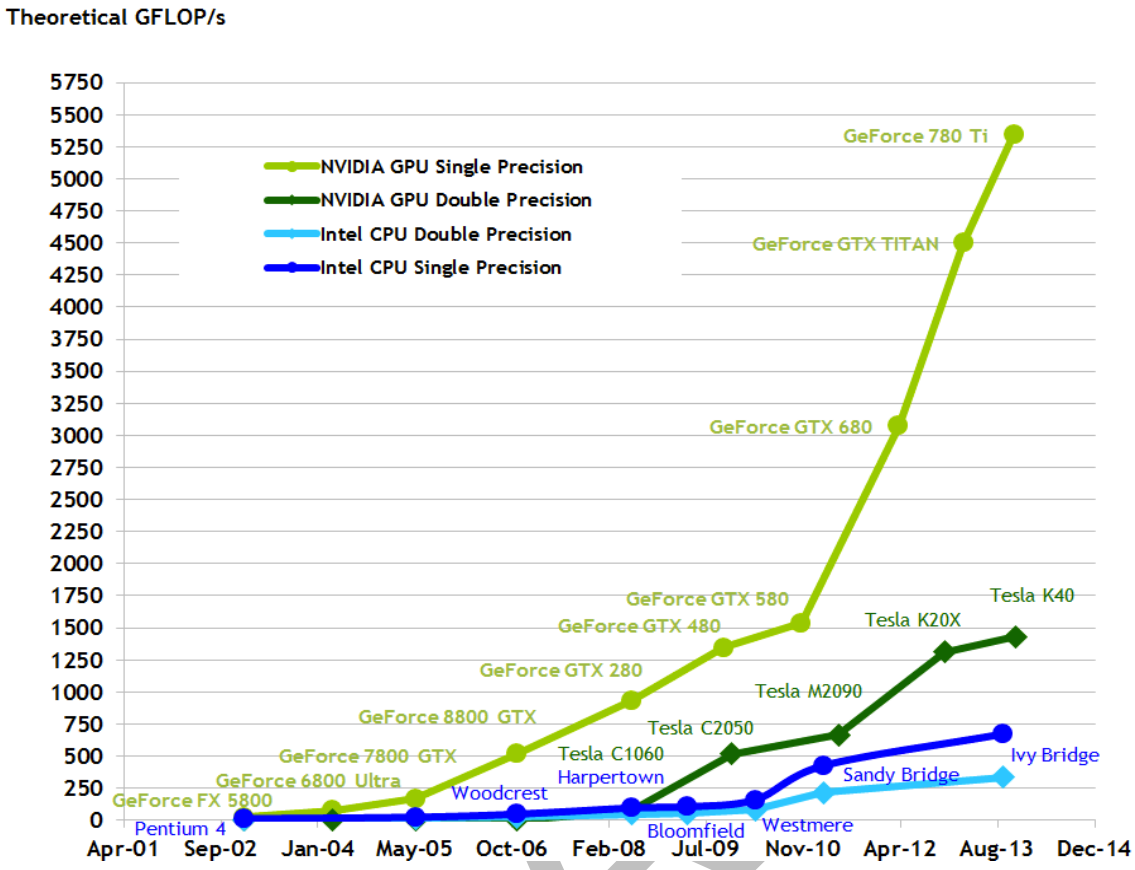

Figura 5. Comparación de FLOPS entre CPU y GPU. Fuente Nvidia.

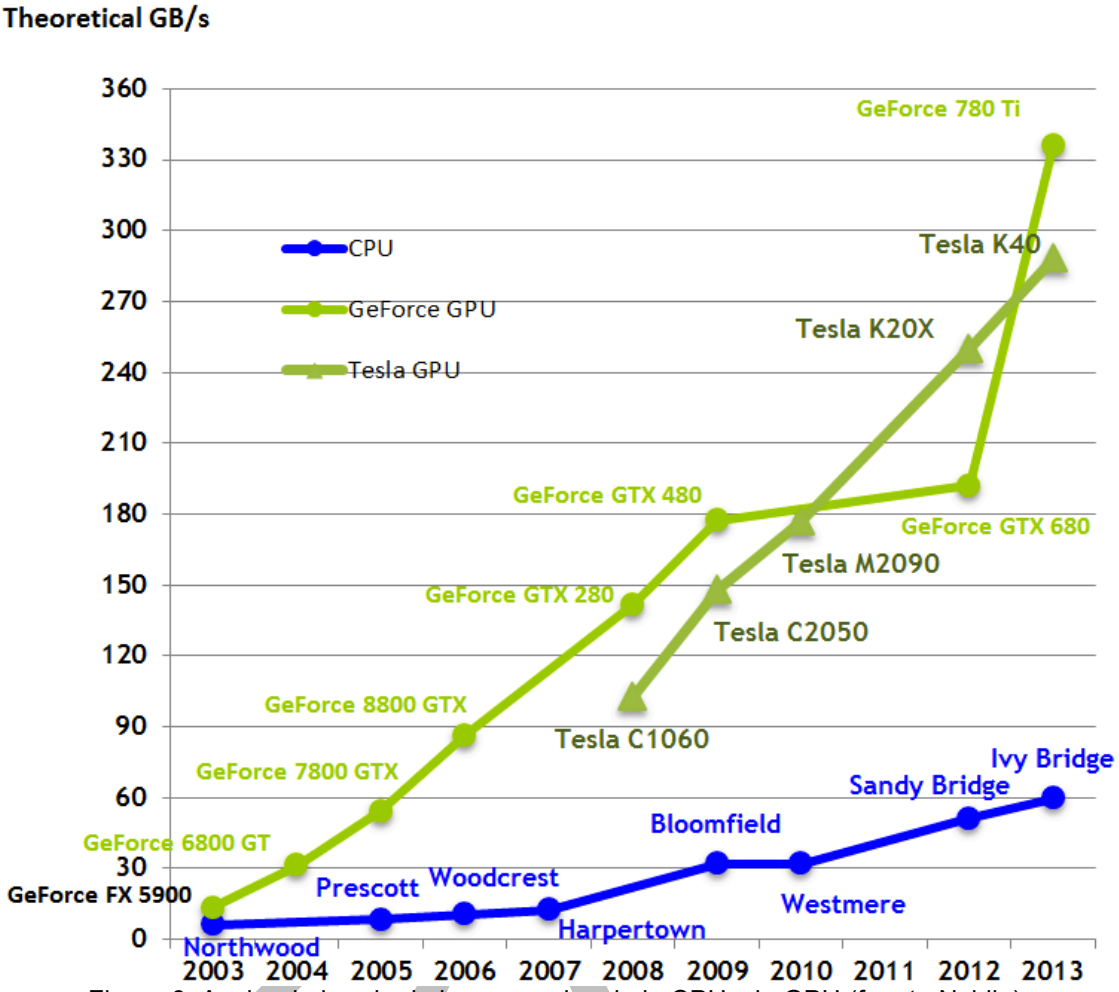

Figura 6. Ancho de banda de las memorias de la CPU y la GPU (fuente Nvidia)

3.1 El camino de los microprocesadores de altas prestaciones.

Muchas estrategias de arquitectura de computadores se han propuesto desde los primeros estadios del desarrollo de microprocesador para reducir al mínimo el tiempo de ejecución de una tarea secuencial. Entre ellas destacamos la ejecución fuera de orden de instrucciones, la predicción de saltos que permite un ejecución de las instrucciones especulativa, y el uso sofisticado cachés de memoria. El principal objetivo de estas técnicas es principalmente explotar el paralelismo a nivel de instrucción (*Instruction Level Paralelism,* ILP) , con el soporte del aumento continuo de la frecuencia del procesador. Estos microprocesadores son esencialmente arquitecturas orientadas latencia, cuyo

principal objetivo es reducir el tiempo de ejecución de la tarea (latencia). Por ejemplo, los procesadores, como la familia Intel ® Pentium ® o la AMD Opteron ™ han estado a la vanguardia de estas arquitecturas orientadas latencia.
Esta tendencia fue determinante durante muchos años, ya que estos microprocesadores aportaban un gran rendimiento que se duplicaba en cada generación de procesadores, permitiendo mejoras de rendimiento muy atractivas para las aplicaciones software de manera transparente al programador. Esto es lo que se conoce como la edad de oro del software.

Desde 2003, la confianza de los programadores en que su software se iba a ver acelerado con la compra de un nuevo microprocesador se vio frustrada. El desarrollo de microprocesadores ya no estuvo basado en el incremento de frecuencia, en cambio se tendió hacia diseño que incluían varios cores dentro del mismo procesador. Esto fue debido a varias razones, todas ellas relacionadas con los límites físicos de las arquitecturas basadas en el silicio, y debido principalmente a la disminución de la capacidad de integración de los procesadores siguiendo la senda de la ley de Moore. Entre estos factores destacamos [Borkar et al., 2007]:

- Energía (el consumo de energía, disipación de potencia, etc.),
- Variabilidad,
- Fiabilidad,
- Envejecimiento,
- Testado.

Como consecuencia de estos límites físicos, el aumento de la frecuencia se convirtió en poco práctico, causada principalmente por la excesiva complejidad y reducción de los beneficios de explotación de ILP más allá de un cierto punto.
La industria de la computación cambió de rumbo en 2005, cuando Intel siguió a la tendencia marcada por IBM (Power 4) y el procesador Niagara de Sun Microsystems, marcando la tendencia hacía múltiples unidades de procesamiento. Esta es la aparición de una nueva línea de mercado de procesadores denominada "*multicore*"; plataformas donde múltiples unidades

de procesamiento se utilizan en cada chip para aumentar la potencia de procesamiento.
Las plataformas *multicore* ayudaron a las cargas de trabajo multiprogramadas; es decir que contienen una mezcla de tareas secuenciales independientes, haciendo caso omiso de la aceleración de las tareas individuales.

Evidentemente, esto produjo un cambio de la programación puramente secuencial a una programación modestamente paralela que hacía muchos más difícil la tarea de programación sin tener unos rendimientos significativos [Sutter y Larus, 2005]. Por otra parte, la expectativa histórica que una sola aplicación se ejecuta más rápido con cada nueva generación de microprocesadores ya no fue válida. Un programa secuencial que utilice sólo uno de los múltiples procesadores en el chip, que se convirtió significativamente más lento que las unidades de proceso de generaciones anteriores. Esta problema limitaba el desarrollo de aplicaciones mas potentes que cubrieran los retos científicos del siglo XXI [Kirk y Hwu, 2010].

Los programadores tenían que empezar a jugar un papel vital en la era multicore que nunca habían desarrollado antes si querían obtener mejoras de rendimiento en sus aplicaciones. Las aplicaciones tenían que ser redefinidas como cargas de trabajo paralelas, en el que múltiples hilos de ejecución cooperaban para completar el trabajo más rápido, aprovechando los múltiples núcleos disponibles en el chip. Esta fue la llamada revolución de concurrencia [Sutter y Larus, 2005].

Esta revolución fue un poco frustrante para los programadores y a día de hoy todavía sigue siéndolo. La práctica de la computación paralela se ha extendido ampliamente, pero sigue todavía limitada a la comunidad científica, en la que algunos desarrolladores han estado desarrollando programas en gran escala, equipos costosos que son en realidad limitado a las instituciones grandes, empresas o naciones que están en condiciones de pagar todos relacionados fijo y costos variables [Madera y Hill, 1995]. Por lo tanto, la programación

paralela es hoy en día la única manera de crear programas eficientes en todas las plataformas, incluyendo PC de las materias primas, ya que todos los procesadores son máquinas basadas paralelas.

3.2 Arquitecturas orientadas a throughput

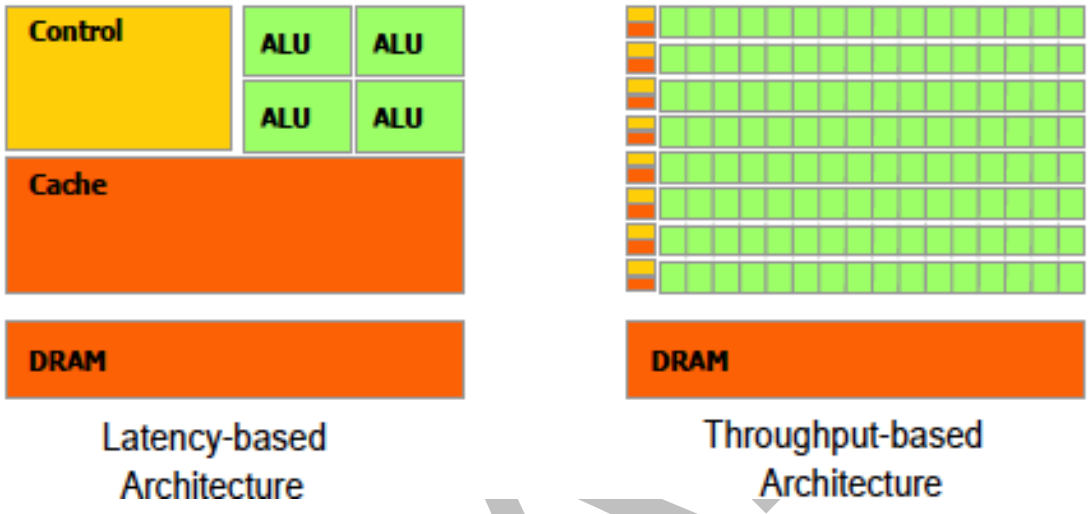

Figura 7. Uso diferentes del espacio de silicio. Las arquitecturas basadas en throughput dedican mas transistores a elementos de procesamiento (fuente:Nvidia)

La promesa de paralelismo tiene fascinado a los investigadores de computación de altas prestaciones desde hace al menos tres décadas [Asanovic et al., 2006]. Sus expectativas se han basado sobre las cargas de trabajo en el que el paralelismo es abundante, sacrificando de buena gana la velocidad de ejecución de un solo hilo para aumentar el rendimiento total de cómputo a través de todos los elementos de procesamiento, acelerando así el tiempo total de ejecución de una aplicación. Es por ello que este tipo de arquitecturas se denominan orientadas a rendimiento o throughput **[Kirk and Hwu, 2010].**

Las cargas de trabajo masivamente paralelas que se esperan desarrollar se espera procesadores orientados rendimiento han provocado varias decisiones del diseño de la arquitectura de estos procesadores muy diferentes de los procesadores orientados a latencia tradicionales, entre las que podemos

destacar el énfasis en el uso de muchos núcleos de procesamiento simples en lugar de tener procesadores muy complejos como los multicores.

Las arquitecturas orientadas a rendimiento incluyen en un área de silicio equivalente, procesadores muchos mas simples, y por tanto, mas pequeños, para aumentar el rendimiento total de la aplicación, en lugar de la latencia de una sola tarea (ver Figura 7). Esta alternativa logra grandes niveles de rendimiento en cargas de trabajo con abundante paralelismo. Por otra parte, las arquitecturas orientadas a rendimiento también utilizan otras técnicas de arquitectura de computadores, como hardware multithreading y la ejecución de una instrucción sobre múltiples datos (SIMD), que también se aplica a las arquitecturas orientadas a tareas. Estas estrategias se verán un poco mas detalladamente en el siguiente tema.

3.3 La unidad de procesamiento gráfico como arquitectura de altas prestaciones

Las emergente Unidades de Procesamiento de Gráficos (GPU) son los principales ejemplos de arquitecturas orientadas a rendimiento [Garland y Kirk, 2010]. Están ofreciendo un gran funcionamiento en diversos campos de computación, tales como simulaciones científicas, Ciencias de la Vida, Visión por Computador y así sucesivamente [CUDAZone, Hwu, 2011].

Motivado por el mercado de los videojuegos, las GPUs ofrecen un gran rendimiento a precios muy competitivos en comparación con las arquitecturas basadas en la latencia. Las principales decisiones de diseño de arquitectura de las GPUs modernas han sido las siguientes; masivo multithreading, memorias caché relativamente y un diseño de la interfaz de memoria con un gran ancho de banda. En las siguientes secciones, se va a explicar la razón de estas decisiones, y también responder a la pregunta de por qué GPUs han experimentado un movimiento revolucionario hacia la computación de alto rendimiento de propósito general. Comenzamos nuestro análisis muestra la histórica evolución de las GPU.

3.3.1 Perspectiva histórica

La notable evolución del rendimiento de los procesadores gráficos ha evolucionado de acuerdo a la demanda de alta calidad de los gráficos en tiempo real de las aplicaciones informáticas. Dichas aplicaciones suelen modelar una realidad, la producción de varias imágenes o marcos que son procesados por varias primitivas geométricas como triángulos y produciendo así de alguna manera una realidad virtual. Para producir esas imágenes a base de triángulos, varias técnicas -normalmente llamados técnicas de renderizado se aplican a cada triángulo, triángulo esquina (o vértice) y pixel cubiertos por un triángulo. Además, cuanto mas pequeño es el tamaño de los triángulos, mejor será la calidad de la imagen, produciendo un enorme cantidad de trabajo en paralelo, ya que todos los triángulos, sus vértices y píxeles son procesado independientemente de todos los demás. Además, en una de las aplicaciones gráficas líderes como la electrónica del videojuego, la tasa de aplicación es la representación entre 30-70 fotogramas por segundo para imágenes de alta calidad en tiempo real, por lo que la arquitectura de gráficas se espera que generen, procesen y muestren imágenes en el rango de 14,2-33,3 milisegundos. Por lo tanto, el objetivo principal de la arquitectura de gráficos es aumentar el rendimiento total de la aplicación, dejando la aceleración de un solo hilo a un lado. Los dos principales características de las aplicaciones gráficas: orientadas a rendimiento y paralelismo masivo, han impulsado la evolución de la arquitectura de gráficos durante últimas décadas.

3.3.2 GPGPU: General Purpose computation on the Graphics Processing Unit

El hardware de gráficos ha sido un área activa de investigación para el desarrollo de la computación de propósito general durante muchos años. Las primeras máquinas especificas para gráficos en los que algunas aplicaciones

de uso general donde se desarrollaron fueron la Ikonas [Inglaterra, 1978], la máquina de píxeles [Potmesil y Hoffert, 1989] y Pixel-Planes 5 [Rhoades et al., 1992]. Estos primeros aceleradores gráficos eran típicamente aplicaciones que computan en servidores en lugar de las estaciones de trabajo de escritorio. Además, se hicieron otros intentos después de que el amplio despliegue de las GPU, pero aún con pipelines fijos que se clasificaron en [Trendall y Stewart, 2000]. Por ejemplo, Lengyel et al. [1990] utiliza el hardware de rasterización para la planificación de movimiento del robot. Hoff et al. [2001] describió el uso de técnicas de la memoria z para el cálculo de diagramas de Voronoi. Kedem y Ishihara [1999] utilizan un hardware de gráficos SIMD para romper el cifrado de contraseñas UNIX. Bohn [1998] también se utiliza el hardware de gráficos en el cálculo de las redes neuronales artificiales. Convolución y la transformada se llevaron a cabo por Hopf y Ertl [1999], Hopf y Thomas [1999]. Sin embargo, el hito para difundir GPU como una plataforma de uso general fue motivada por primera vez por Larsen y McAllister [2001], que demostró la capacidad de un procesador de gráficos para acelerar un producto típico densa matriz a través de operaciones regulares de texturas.

Este resultado atrajo a la comunidad científica en una carrera por el uso de la GPU como un co-procesador, y de inmediato el número de aplicaciones mejoradas con este arquitectura dio lugar a la iniciativa GPGPU (cómputo de propósito general en unidades de procesamiento de gráficos, también conocida como GPU Computing y GPGPU.org en la Web) acuñada por Mark Harris en 2002 como un intento de recopilar todos estos logros [Luebke et al., 2006]. Poco después, se desarrollo el lenguaje Cg para facilitar la programación de aplicaciones gráficas que iban de la mano de un hardware totalmente programable. [Lindholm et al., 2001]. Las GPUs comenzó a ser considerado seriamente en la comunidad HPC principalmente debido al rendimiento y el su paralelismo masivo. Las GPUs programables fueron expresamente diseñadas para procesar múltiples primitivas de datos en paralelo al mismo tiempo. Por otra parte, las GPU normalmente tenía múltiples procesadores de vértices y fragmentos. Por ejemplo, la NVIDIA GeForce 6800 Ultra tenían 6 procesadores

para vértices y 16 para fragmentos. Sin embargo, el hardware de gráficos era muy limitado para el desarrollo de aplicaciones de propósito general por varias razones que se resumen principalmente en dos limitaciones principales: las limitaciones de hardware y el modelo de programación dedicado a gráficos. En cuanto al hardware, los conjuntos de instrucciones de cada etapa procesador eran muy limitados en comparación con las CPU; eran principalmente las operaciones matemáticas, muchas de las cuales eran específicas para gráficos y sólo habían disponibles unas pocas operaciones de control de flujo. Por otra parte, cada etapa programable puede acceder a los registros constantes de todas las primitivas y también leer-escribir registros por primitiva, pero estos recursos eran muy limitados en su número de entradas, salidas, constantes, registros e instrucciones. Los procesadores de fragmentos tenían la capacidad de obtener los datos de las texturas, por lo que fueron capaces de reunir la memoria (memory gather). Sin embargo, la dirección de salida de un fragmento siempre estaba asignada antes del procesamiento del fragmento en sí. Es decir, el procesador no podía cambiar la ubicación de salida de un píxel, por lo que los procesadores de fragmentos no eran inicialmente capaz de hacer la dispersión de memoria (memory scatter). Los procesadores de vértices evolucionaron adquiriendo capacidades de las texturas, y por lo tanto eran capaces de cambiar la posición de los vértices de entrada, que en última instancia afectaban donde los píxeles de la imagen se podían extraer. Por lo tanto, los procesadores de vértices se convirtieron en realizar ambos memory gather y scatter. Por desgracia, el memory scatter todavía podía conducir a problemas de memoria y de coherencia rasterización en etapas posteriores del procesado. Combinado con el menor rendimiento de los procesadores de vértices, esto limitaba el rendimiento de las aplicaciones de propósito general en la GPU [Owens et al., 2007].

Al inicio de esta nueva era GPGPU en 2002, la API disponible para interactuar con las GPUs eran DirectX 9 y OpenGL 1.4, ambos diseñados sólo para que coincida con las características requeridas por las aplicaciones gráficas. Para acceder a los recursos computacionales, los programadores tenían que emitir

sus problemas en operaciones de gráficos nativos, por lo tanto la única manera de poner en marcha su cálculo era a través de las llamadas al API deOpenGL o DirectX. Por ejemplo, para ejecutar las instancias simultáneas de una función de cálculo, el cálculo se escribía como un sombreado de píxeles. La colección de datos de entrada se almacena en las imágenes de textura y cargada en la GPU mediante el envío de triángulos. La salida era mostrada como un conjunto de píxeles generada a partir de la operaciones de trama con las limitaciones de hardware mencionadas anteriormente [Kirk y Hwu, 2010].

A pesar de este escenario tan engorroso para la programación, algunas aplicaciones científicas de diferentes campos fueron portadas a la GPU [Owens et al., 2007] por investigadores intrépidos. Algunos primeros trabajos fue presentado por Thompson et al. [2002] en el que se utilizó el procesador de vértices programables de una GPU NVIDIA GeForce 3 para resolver el problema de Satisfacibilidad y llevar a cabo la multiplicación de matrices. Además, Strzodka mostró la combinación de canales de textura 8 bits múltiples para crear virtual de 16 bits operaciones de punto flotante [Strzodka, 2002], y Harris analizó el acumulado error en las operaciones de simulación de ebullición causada por la baja precisión [Harris, 2002] en la primera generación de GPUs. Strzodka construyó y analizó los esquemas discretos especiales que, por cierto tipos de PDE, permiten la reproducción del comportamiento cualitativo de la solución continua, incluso con muy baja precisión computacional, por ejemplo, 8 bits [Strzodka, 2004].

Se hicieron otros esfuerzos en campos como simulaciones físicas, interpretación de señales y procesamiento de imágenes, segmentación, etc [Pharr y Fernando, 2005, Owens et al., 2007]. Con la llegada de CUDA en 2006, la programación de aplicaciones de propósito general en las GPU se hizo más accesible. NVDIA ha vendido millones de GPUs CUDA. Desarrolladores de software, los científicos y los investigadores están encontrar campos de aplicación de amplio alcance para CUDA, incluyendo imágenes y video procesamiento, la biología y la química computacional, simulación de dinámica

de fluidos, reconstrucción de imágenes, análisis sísmico, el trazado de rayos y muchos más [CUDAZone, Nguyen, 2007, Sanders y Kandrot 2010, Hwu, 2011].