

SISTEMAS PARALELOS

Clase 1 – Introducción

Prof. Enzo Rucci



Facultad de
INFORMÁTICA



UNIVERSIDAD
NACIONAL
DE LA PLATA

Metodología

- Clases
 - Teorías: Jueves de 15 a 17hs (aula 1-2)
 - Prácticas: Jueves de 17 a 19:30hs (sala de PC Postgrado). Inicio: 21/03.
- Aprobación de la cursada: a partir de la entrega y aprobación de 2 trabajos prácticos y sus respectivos coloquios (obligatorios). Fecha recuperatoria.
- Aprobación del final:
 - Rendir examen final escrito en alguna mesa de examen.
 - Aprobar un parcial teórico al final de la cursada + [desarrollo de un trabajo dado por la cátedra y su posterior coloquio en mesa de final (hasta mesa de Marzo de 2020)].
 - Desarrollo y prueba de soluciones paralelas + elaboración de informe de análisis de rendimiento + coloquio

Metodología

- Material y comunicación: Plataforma Ideas (ideas.info.unlp.edu.ar)
 - Buscar curso «Sistemas Paralelos» y solicitar inscripción
- Correo de la cátedra: sparalelos@lidi.info.unlp.edu.ar

Objetivos

- Plantear los fundamentos del procesamiento paralelo
- Caracterizar las arquitecturas de hardware
- Describir los modelos de programación
- Estudiar métricas de rendimiento
- Analizar y trabajar casos concretos de procesamiento paralelo, resolubles sobre distintas arquitecturas multiprocesador.

Cronograma, sala y entregas

- En el curso de IDEAS, encontrarán el cronograma tentativo de la materia.
- La disponibilidad de la sala de PC depende de Postgrado. Cuando no dispongamos de la misma, se avisará mediante la cartelera de IDEAS.
- Sobre las entregas:
 - Se recomienda “desarrollar en casa” y usar el tiempo de consulta para despejar dudas y realizar las “pruebas de producción”.
 - Las pruebas para la primera entrega (memoria compartida) puede realizarse en sus máquinas personales, si disponen del sistema adecuado.
 - Las pruebas para la segunda entrega (memoria distribuida) deben realizarse en la sala de PC de Postgrado.

Bibliografía básica

- “Introduction to Parallel Computing”. Grama, Gupta, Karypis, Kumar. Addison Wesley (2003).
- “Introduction to High Performance Computing for Scientists and Engineers”. Georg Hager, Gerard Wellein. CRC Press (2011).
- “Parallel Programming for Multicore and Cluster Systems”. Thomas Rauber, Gudulla Runger. Springer (2010).
- “An introduction to parallel programming”. Peter Pacheco. Elsevier (2011).
- “Parallel Programming”. Wilkinson, Allen. Prentice Hall 2005.

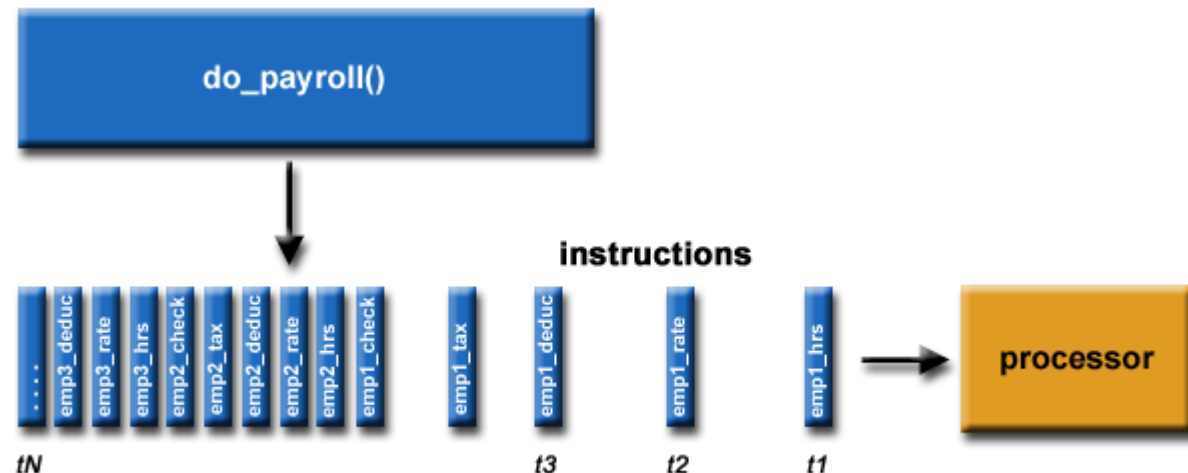
Agenda de esta clase

- Fundamentos de procesamiento paralelo
- Plataformas de procesamiento paralelo
- Modelos de programación paralela
- Evolución de los procesadores
- Clusters

FUNDAMENTOS DE PROCESAMIENTO PARALELO

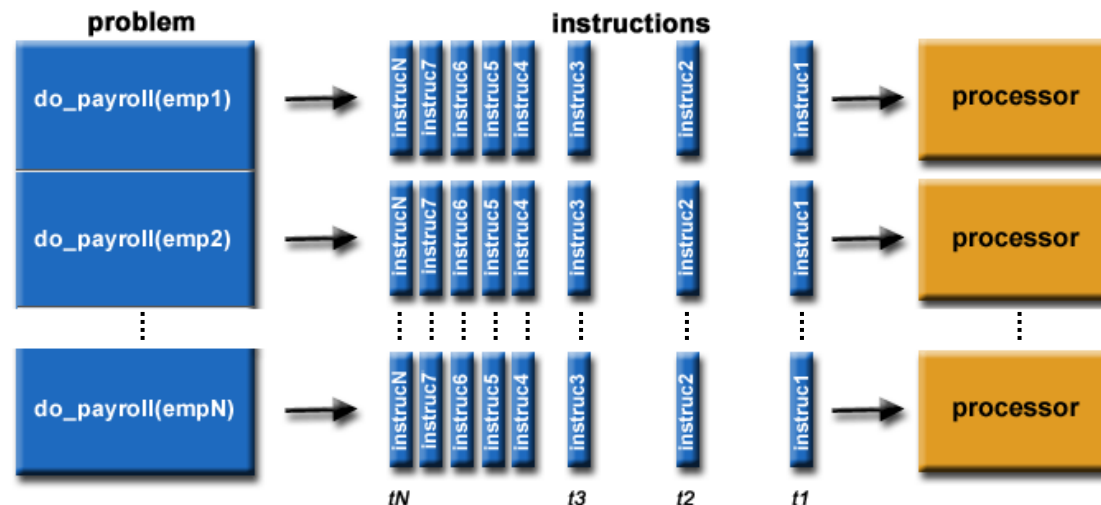
¿Qué es el procesamiento paralelo?

- Históricamente, el software ha sido desarrollado para ejecución secuencial
 - El problema se divide en conjuntos separados de instrucciones
 - Las instrucciones se ejecutan secuencialmente, una después de la otra.
 - La ejecución se realiza en un procesador único
 - Sólo una instrucción es ejecutada en un instante determinado.
- Ejemplo:



¿Qué es el procesamiento paralelo?

- En forma sencilla, el *procesamiento paralelo* es el uso de múltiples unidades de procesamiento para resolver un problema computacional.
 - El problema se divide en partes separadas que pueden ser resueltas en forma concurrente.
 - Cada parte es luego dividida en una serie de instrucciones.
 - Las instrucciones de cada parte se ejecutan simultáneamente sobre diferentes procesadores.
 - Un mecanismo de control/coordinación global es necesario
- Ejemplo:



¿Por qué es importante el procesamiento paralelo?

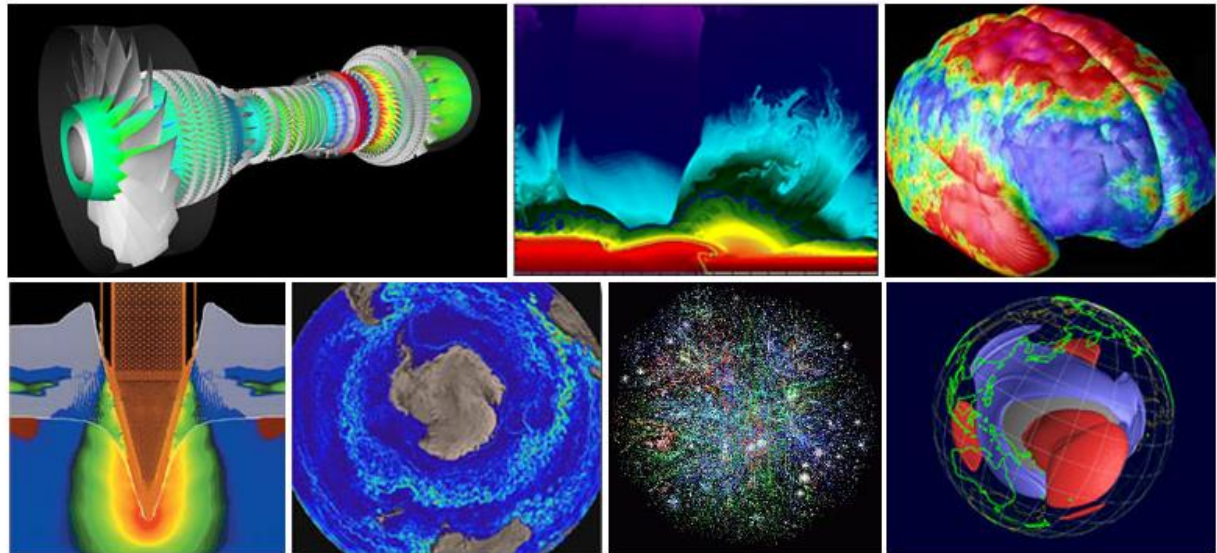
- Resolver problemas más grandes o más complejos
 - Algunos problemas son tan grandes y/o complejos que resulta poco práctico (o imposible) de resolver empleando una única computadora
- Proveer concurrencia:
 - Una sola unidad de procesamiento puede hacer una única tarea en un instante dado. Múltiples unidades de procesamiento pueden hacer múltiples tareas al mismo tiempo.

¿Por qué es importante el procesamiento paralelo?

- Ahorrar tiempo y/o dinero
 - En general, destinar más recursos a una tarea conlleva a completarla en menor tiempo, lo que a su vez puede ahorrar dinero.
 - Las computadoras paralelas pueden ser construidas a partir de componentes básicos (*commodities*).
- Hacer mejor uso de los recursos de hardware
 - Hoy todas las computadoras son paralelas con múltiples unidades de procesamiento.
 - El software paralelo se diseña específicamente para aprovechar el hardware subyacente.
 - En la mayoría de los casos, los programas secuenciales *desperdician* el poder de cómputo que ofrecen las computadoras actuales.

¿Qué problemas requieren procesamiento paralelo?

- Ciencia e Ingeniería
 - Históricamente, el procesamiento paralelo ha sido considerado como un recurso de lujo, y ha sido empleado para modelar problemas complejos de la ciencia y de la ingeniería.
- Física – aplicada, nuclear, partículas, materia condensada
- Biociencia, bioingeniería, genómica
- Química, ciencias moleculares
- Geología, sismología
- Ingeniería mecánica, ingeniería electrónica
- Defensa
- Entre otras

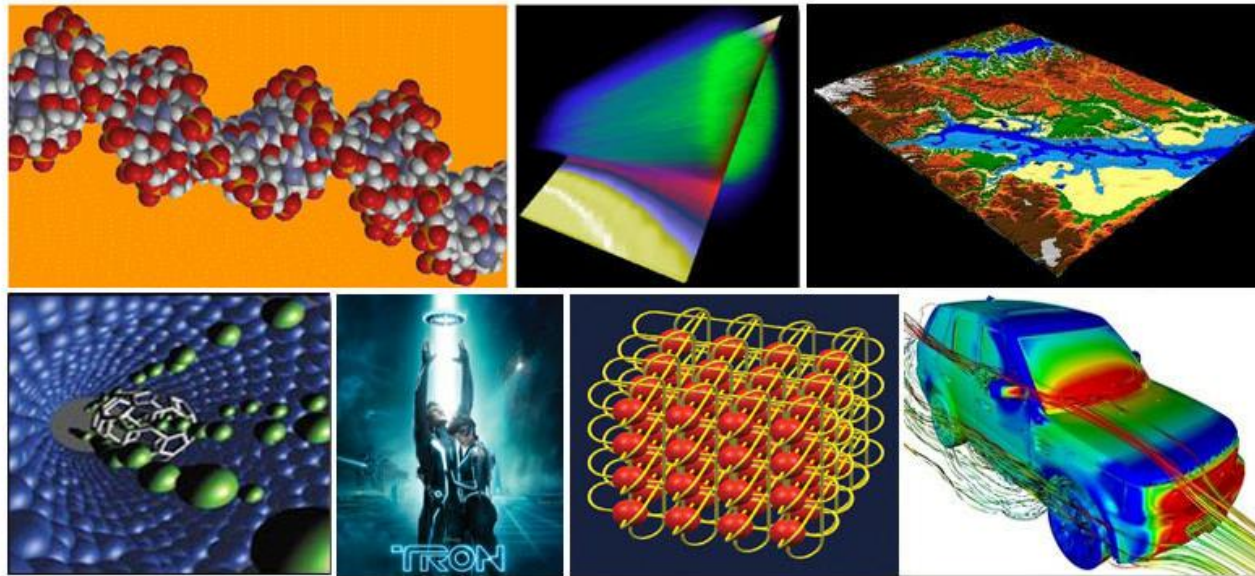


¿Qué problemas requieren procesamiento paralelo?

- Industria y comercio

- En la actualidad, diferentes área de la industria y el comercio requieren procesar grandes conjuntos de datos de forma sofisticadas.

- *Big data*, minería de datos
- Exploración de hidrocarburos
- Motores de búsqueda web
- Diagnóstico médico por imágenes
- Diseño de fármacos
- Modelización económica y financiera
- Computación gráfica, realidad virtual
- Muchas más



PROCESAMIENTO CONCURRENTE, PARALELO Y DISTRIBUIDO

Procesamiento concurrente, paralelo y distribuido

- No hay un completo acuerdo sobre la diferencia entre procesamiento concurrente, paralelo y distribuido.
- Algunos autores hacen las siguientes distinciones:
 - Un programa *concurrente* es aquel en el que múltiples tareas puede estar avanzando en cualquier instante de tiempo.
 - Un programa *paralelo* es aquel en el que múltiples tareas que se ejecutan simultáneamente cooperan para resolver un problema.
 - Un programa *distribuido* es aquel en el que múltiples tareas que se ejecutan físicamente en diferentes lugares cooperan para resolver uno o más problemas.

Procesamiento concurrente, paralelo y distribuido

- De las definiciones anteriores:
 - Los programas paralelos y distribuidos son entonces programas concurrentes
 - La concurrencia es un concepto de software y especificar la concurrencia implica *especificar los procesos concurrentes, su comunicación y sincronización*.

Procesamiento paralelo y distribuido: similitudes y diferencias

- El procesamiento paralelo busca reducir el tiempo de ejecución de un programa empleando múltiples procesadores al mismo tiempo.
 - El hardware brinda respuesta pero lo fundamental sigue siendo el software → Cómo desarrollar software que sea capaz de aprovechar el hardware subyacente
 - Una consecuencia inevitable e indeseada es la gran dependencia entre el software y el hardware subyacente para obtener alto rendimiento
 - Al mismo tiempo, el hardware evoluciona: de los multiprocesadores a los clusters, multiclusters, multicores, aceleradores...

Procesamiento paralelo y distribuido: similitudes y diferencias

- Un sistema distribuido es un conjunto de computadoras autónomas interconectadas que cooperan compartiendo recursos (físicos y datos).
 - Pueden ejecutar múltiples aplicaciones de diferentes usuarios.
 - La granularidad de los nodos y el grado de acoplamiento de los procesadores determina las características y aplicaciones de un sistema distribuido
 - En ocasiones, los sistemas distribuidos no se fabrican como tales, son que evolucionan a partir de redes LAN y WAN → Heterogeneidad
 - Problemas: no hay reloj único, se requiere planificación, la escalabilidad depende de las comunicaciones, heterogeneidad, seguridad de los datos, entre otros...

Procesamiento paralelo y distribuido: similitudes y diferencias - Resumen

- Características comunes:
 - Se usan múltiples procesadores
 - Los procesadores se encuentran interconectados por algún tipo de red
 - Múltiples tareas evolucionan al mismo tiempo y cooperan/compiten.
- Diferencias básicas:
 - Los programas paralelos se descomponen en tareas que se ejecutan al mismo tiempo
 - Los programas distribuidos se descomponen en tareas que se ejecutan físicamente en lugares diferentes con diferentes recursos locales

A veces se usan ambos términos con el mismo significado e incluso hay autores que consideran al paralelismo una sub-área del procesamiento distribuido.

Cómputo de Alto Rendimiento

- También llamado Cómputo de Alto Desempeño o HPC por *High-Performance Computing*
- Una posible definición es la siguiente:

Es el uso de supercomputadoras y técnicas de procesamiento paralelo para la resolución de problemas complejos con alta de demanda computacional

- HPC involucra una amplia gama de temas: algoritmos, técnicas de programación, aplicaciones, hardware, redes, herramientas, entre otros
- Diferencia con procesamiento paralelo

HPC - Ejemplos



**Barcelona
Supercomputing
Center**
Centro Nacional de Supercomputación

DESCUBRE EL BSC RESEARCH & DEVELOPMENT MARENOSTRUM TECH TRANSFER ÚNETE EDUCACIÓN NOTICIAS

Buscar ...



Inicio / Computer Applications / **Alya Red - HPC-based Computational Biomechanics for Supercomputers**

< Inicio

Alya Red - HPC-based Computational Biomechanics for Supercomputers

Overview:

*No man-made structure is designed like a heart. Considering the highly sophisticated engineering evidenced in the heart, it is not surprising that our understanding of it comes so slowly.
Daniel D. Streeter Jr. - The Handbook of Physiology Section 2.
The Heart (American Physiology Society). 1979.*

Lead by M. Vázquez and G. Houzeaux, CASE Department, BSC-CNS.

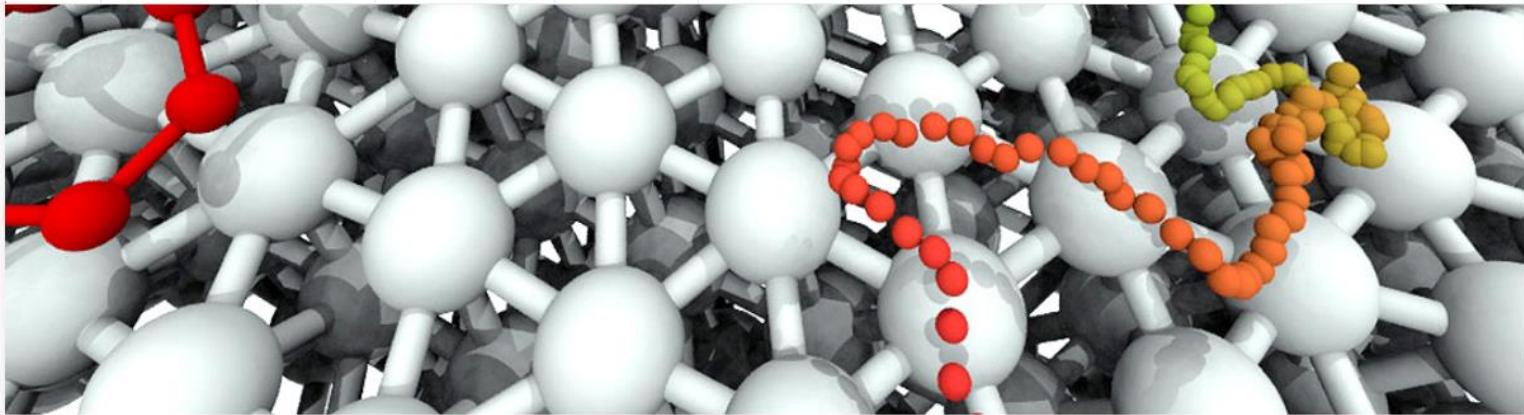
Aplicación paralela que modela y simula el comportamiento del corazón

<https://www.bsc.es/es/computer-applications/alya-red-ccm>

HPC - Ejemplos

- ¿Cuántas veces late un corazón por segundo?
 - De 60 a 200 pulsaciones por segundo → Esto significa una pulsación cada 1000 a 300 ms
- ¿Cuánto se tarda *aproximadamente* en simular un latido del corazón?
 - Usando 200 núcleos del Marenostrium → 100 minutos
 - Si usamos 1 único núcleo, nos tomaría aproximadamente 200 veces más → $100 \cdot 200 = 20.000$ minutos = 333,33 hs = **13,8 días**
- ¿A cuánto equivale 1 día completo de simulación del Marenostrium?
 - Con 10.000 núcleos se puede simular 720 latidos
 - Usando 1 único núcleo, se requerirían 10.000 días = **27,4 años**

HPC - Ejemplos



29

MAR

El grafeno que crece

En el Instituto de Investigaciones en Fisicoquímica de Córdoba simularon en computadoras el proceso de formación del grafeno en una superficie metálica. El trabajo, publicado en Science, permitiría encontrar formas de producción menos costosas de un material con gran potencial para la industria por su gran resistencia, bajo peso y buena conductividad.

Simulación del proceso de formación del grafeno en una superficie metálica

<http://www.unsam.edu.ar/tss/el-grafeno-que-crece/>

HPC - Ejemplos

La investigación se dividió en una parte de experimentación, que se hizo en Europa, y otra de análisis y simulación computacional, que se hizo en la Argentina. En Italia, incorporaron un módulo de escaneo de alta velocidad (de 60 imágenes por segundo) a un microscopio altamente sofisticado, con el objetivo de registrar, en tiempo real, el comportamiento de los átomos individuales de carbono sobre la superficie de níquel. Con la filmación del proceso y los datos recabados durante la fase experimental en Italia, los científicos cordobeses generaron los modelos teóricos y realizaron las simulaciones computacionales para intentar explicar este proceso.

Para la simulación computacional del crecimiento de láminas de grafeno (imagen que abre esta nota) se utilizó la [supercomputadora Mendieta](#), del [Centro de Cómputo de Alto Desempeño](#) de la UNC. Esta máquina tiene 424 núcleos que pueden trabajar en simultáneo y las simulaciones demandaron un mes de procesamiento. Los investigadores estiman que, con una computadora estándar, esa tarea habría llevado más de seis años y medio de uso continuo. “Lo que hicimos fue simular todos los procesos que se llevan adelante durante el crecimiento del grafeno”, explicó Mariscal. Y agregó: “Con eso pudimos ayudar a entender el mecanismo por el cual el grafeno crece sobre la superficie de níquel. Experimentamos con níquel porque es muy utilizado en la industria y tiene la particularidad de que tiene una muy buena adhesión con el grafeno, pero a partir de ahora la idea es optimizar los procesos de síntesis con el níquel y continuar la investigación con otros metales”.



HPC - Ejemplos

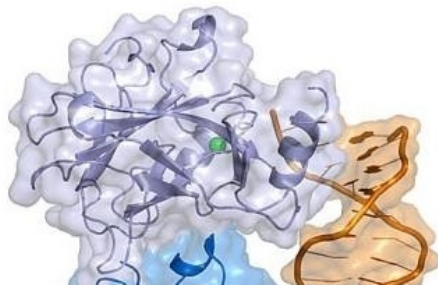
Home / News / Supercomputer Models Critical Protein That Suppresses Cancer

Supercomputer Models Critical Protein That Suppresses Cancer

Michael Feldman | October 29, 2016 00:27 CEST



A new report by Texas Advanced Computing Center (TACC), describes how the center's Stampede supercomputer is being used by a researchers to find out how the p53 protein helps prevent cancer cells from developing. The p53 protein is considered one of the most critical natural defenses the body has to protect itself against a wide array of human cancers, which has earned it a great deal of attention from medical researchers and other life scientists.

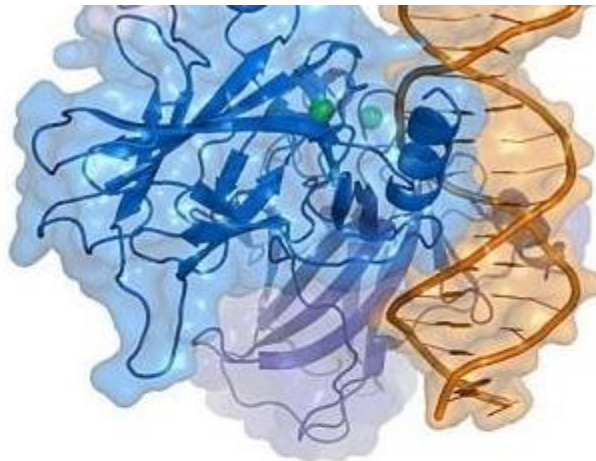


At the high level, p53 seems to work by detecting DNA damage that would lead to cancerous growth and then inducing the cell to self-destruct before it spreads. Besides acting as a tumor suppression protein, p53 also helps to regulate other critical aspects of cells, including cellular senescence and DNA repair associated with aging. As a result, there has been a lot of wet lab experimental work focused on how this molecule does its job, as well as how mutated versions of it leads to cancer and other undesirable health outcomes.

Simulación de sistemas biológicos

<https://www.top500.org/news/supercomputer-models-critical-protein-that-suppresses-cancer/>

HPC - Ejemplos



CHRONICLE NEWS SERVICE.

According to the TACC report, the work there to study p53 in silico is being led by computational biophysicist Dr. Rommie Amaro from the University of California, San Diego. Using Stampede, the center's 5-petaflop supercomputer, she and her team observed the 1.5-million-atom molecule in action for nearly a millisecond. By doing so, for the first time they were able to see how p53 interacts with cellular DNA in a number of ways they had not expected, providing some clues on how the protein does its cancer prevention magic.

From the TACC report:

"As the systems get bigger, much more computational time is required. Previously the simulations were run for a few nanoseconds. Now we have microseconds of dynamical data, which is 1,000x more. In this case we were looking at over a dozen domains and in complex with the DNA. It gives us a much more complete picture of what is actually happening. It's a few steps closer to reality than anything we've been able to accomplish yet," Amaro explained.

She is hoping that the knowledge gained from these supercomputing simulations will help lead to novel cancer therapies that could be delivered at the early stages of the disease.


HPC - Ejemplos








Renderización de la película «Metegol»

<https://www.infotechnology.com/revista/Campanella-cuenta-como-se-hizo-Metegol-20130619-0003.html>

HPC - Ejemplos

 **INFOTECHNOLOGY** Lunes, 1 de Octubre de 2018



películas de Pixar y es algo que está manejado muy inteligentemente, por algo Pixar casi nunca filma en pantalla ancha. En ese sentido, son películas más económicas y sencillas que la nuestra.

¿Cuáles fueron las principales dificultades que enfrentaron? ¿Las barreras fueron presupuestarias o técnicas?

El presupuesto fue creciendo a lo largo del tiempo. En tres años, fácil tenés un 50 por ciento de crecimiento por inflación y más también. Y hubo que gastar más por la necesidad de render de la película. En los últimos meses nos ayudaron de todos lados: [Telefónica](#), [Microsoft](#)... el otro día también vinieron de Arsat para ver si nos podían dar algo de poder de cómputo adicional. Estamos pidiendo render como limosneros. Si alguien tiene una notebook que nos pueda prestar, se la pedimos. Cada fotograma nos está llevando un promedio de unas cuatro horas de render. Como la película también se proyectará en 3D, cada fotograma es doble, uno para cada ojo. Así que ahí tenemos ocho horas de render. A eso hay que multiplicarlo por 24 para llegar a un segundo de película, por 60 para tener un minuto y por 95 para tenerla entera. Es algo brutal y no tiene comparación con nada que se haya hecho hasta ahora en la Argentina ni en América latina.

Renderización de película completa → 45600 días = 124,9 años

HPC - Ejemplos

In addition, PerMed applications have a significant impact in the planning and evaluation of clinical trials. Several research and development initiatives have been launched to build data and software resources able to combine text analysing and genome mining tools. This approach is considered the most promising strategy in the research for new treatment methods.

All these approaches of PerMed require the processing of large amounts of data and often rely on systems requiring a fast in- and output. Benchmarking tests at the Luxembourg Centre for Systems Biomedicine (LCSB) have shown that the use of high performance computing can reduce the time needed to process a full genome - as needed for rare diseases - from one day to 20 minutes.

Using big data in the medical field has more advantages: it leads to more efficient data handling and new security technologies for a trusted management and transfer of clinical data.



Prof. Dr. Rejko Krüger, Neurologist and Neuroscientist at the Luxembourg Centre for Systems Biomedicine said: "Understanding the factors that influence human health and cause disease and using that understanding to develop treatments has always been the driving force behind medical research. Modular, flexible computing performance addressing also high-content and high-throughput computing will trigger a disruptive change that will dramatically alter aspects of personalised medicine. Computational tasks that today would take weeks will be completed in a few days; hour-long computations suddenly become interactive because they could be completed in seconds."

Centro para Sistemas Biomédicos de Luxemburgo:
Medicina de precisión. Análisis del genoma de una
persona para diagnóstico y tratamiento personalizado.

HPC - Ejemplos

In addition, PerMed applications have a significant impact in the planning and evaluation of clinical trials. Several research and development initiatives have been launched to build data and software resources able to combine text analysing and genome mining tools. This approach is considered the most promising strategy in the research for new treatment methods.

All these approaches of PerMed require the processing of large amounts of data and often rely on systems requiring a fast in- and output. Benchmarking tests at the Luxembourg Centre for Systems Biomedicine (LCSB) have shown that the use of high performance computing can reduce the time needed to process a full genome - as needed for rare diseases - from one day to 20 minutes.

Using big data in the medical field has more advantages: it leads to more efficient data handling and new security technologies for a trusted management and transfer of clinical data.

Prof. Dr. Rejko Krüger, Neurologist and Neuroscientist at the Luxembourg Centre for Systems Biomedicine said: "Understanding the factors that influence human health and cause disease and using that understanding to develop treatments has always been the driving force behind medical research. Modular, flexible computing performance addressing also high-content and high-throughput computing will trigger a disruptive change that will dramatically alter aspects of personalised medicine. Computational tasks that today would take weeks will be completed in a few days; hour-long computations suddenly become interactive because they could be completed in seconds."



Uso de HPC permite reducir el tiempo necesario para procesar el genoma completo de una persona de 1 día a 20 minutos

HPC - Ejemplos

Boeing Catches a Lift with High Performance Computing

Boeing Corporation saved many millions of dollars by using supercomputers. Boeing physically tested 77 prototype wing designs for the 767 aircraft, but for the new Boeing 787 Dreamliner only 11 wing designs had to be physically tested (a 7 fold reduction in the needed amount of prototyping), mainly because over 800,000 hours of computer simulations on supercomputers had drastically reduced the amount of needed physical prototyping.

Uso de HPC permitió reducir costos al reemplazar costos prototipos físicos por horas de cómputo de simulación

HPC – Modelización financiera

10,623 views | Mar 20, 2012, 04:22pm

Supercomputer Manages Fixed Income Risk At JPMorgan

 **Tom Groenfeldt** Contributor 
I write about finance and technology.

f JPMorgan is using a [Maxeler](#) supercomputer to measure risk in its fixed income operations by assessing tens of thousands of possible market

t scenarios, constantly examining the time path and structure of the associated risk. With the supercomputer, which went live in December, the

in investment bank can calculate complex scenarios which used to take hours in just a few minutes. The investment bank was awarded the 'Most Cutting Edge IT Initiative' at the prestigious [American Financial Technology Awards \(AFTA\)](#).

Cálculo del riesgo de operaciones financieras en la empresa JPMorgan

<https://www.forbes.com/sites/tomgroenfeldt/2012/03/20/supercomputer-manages-fixed-income-risk-at-jpmorgan/#a7f945100196>

HPC – Modelización financiera

10,623 views | Mar 20, 2012, 04:22pm

Supercomputer Manages Fixed Income Risk At JPMorgan

 **Tom Groenfeldt** Contributor ⓘ
I write about finance and technology.

f JPMorgan is using a [Maxeler](#) supercomputer to measure risk in its fixed income operations by assessing tens of thousands of possible market

tw scenarios, constantly examining the time path and structure of the

in associated risk. With the supercomputer, which went live in December, the investment bank can calculate complex scenarios which used to take hours in just a few minutes. The investment bank was awarded the 'Most Cutting Edge IT Initiative' at the prestigious [American Financial Technology Awards \(AFTA\)](#).

Uso de HPC permite reducir el tiempo necesario para calcular escenarios complejos de horas a minutos

HPC – Analítica de datos

Baseball analytics: Mystery MLB team uses a Cray supercomputer to help win more games

BY TAYLOR SOPER on April 6, 2015 at 10:18 am

For starters, Bolding notes that 95 percent of baseball stats have been created over the last five years thanks to the growing amount of data sensors and innovative methods of analyzing players.



"They are gathering so much data that a single person with an Excel spreadsheet can no longer analyze, in a sophisticated way, all the data they have," Bolding said. "They need bigger and bigger computers to be able to analyze the data."

A mystery MLB team uses the Urika-GD appliance, which ranges from a few hundred thousand to well over \$1 million.

As popularized by Michael Lewis' *Moneyball* and the subsequent movie, using baseball data to drive decisions about player personnel — and ultimately win more games — was a strategy first used successfully by the Oakland in 2003.

Predicción del rendimiento de un bateador contra diferentes tipos de *pitchers*

<https://www.geekwire.com/2015/baseball-analytics-mystery-mlb-team-uses-a-cray-supercomputer-to-crunch-data/>

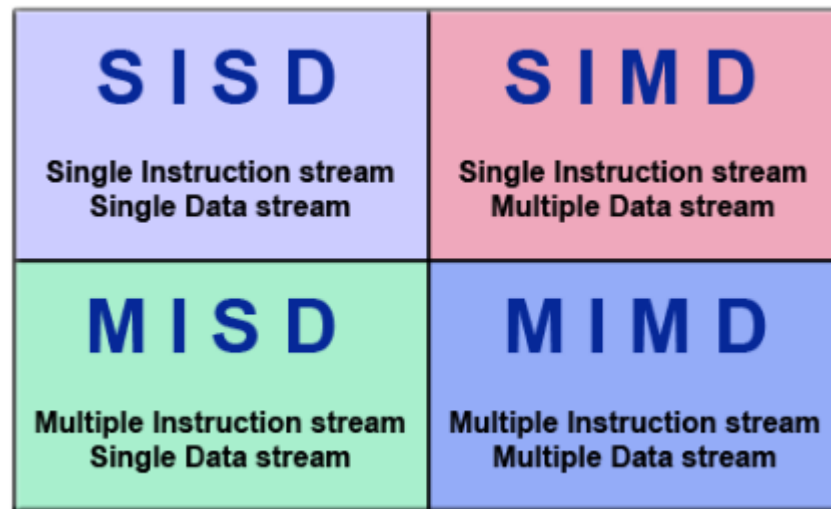
PLATAFORMAS DE CÓMPUTO PARALELO

Clasificación de las plataformas de cómputo paralelo

- Para alcanzar alto rendimiento, es fundamental conocer las características de la arquitectura subyacente.
- Existen diversas formas de clasificar a las plataformas de cómputo paralelo. Entre las más usadas se encuentran:
 - Por el mecanismo de control
 - Corresponde a la Taxonomía de Flynn
 - De acuerdo a cómo se especifica el paralelismo entre instrucciones y datos
 - Por la organización física:
 - De acuerdo al espacio de direcciones que tiene cada procesador (es decir, cómo ve la memoria principal)

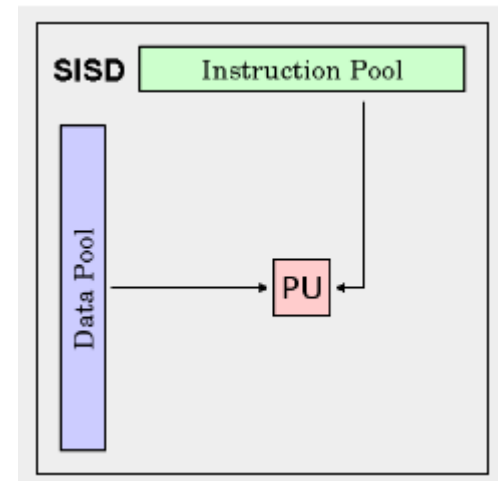
Clasificación por mecanismo de control

- Flynn diseñó una taxonomía para clasificar a las computadoras en **1966**
- Se clasifican según el número de instrucciones y datos que se pueden procesar simultáneamente



Clasificación por mecanismo de control: SISD

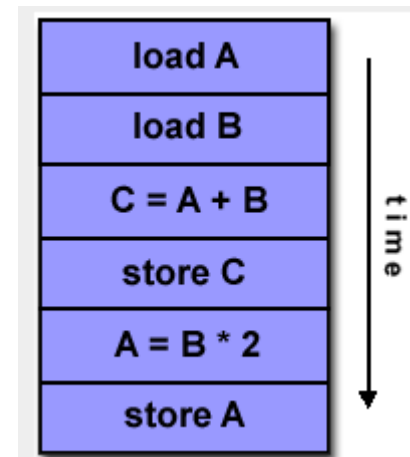
- Las instrucciones son ejecutadas en forma secuencial, una por ciclo de reloj
- En cualquier ciclo de reloj, los datos afectados son aquellos que hace referencia la instrucción en cuestión
- La ejecución se vuelve determinística
- Es el tipo más antiguo de computadoras
- Ejemplos: mainframes, computadoras monoprocesador



UNIVAC1

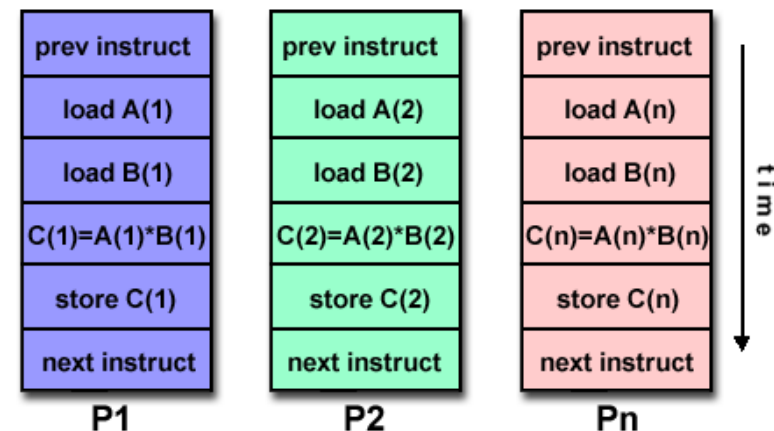
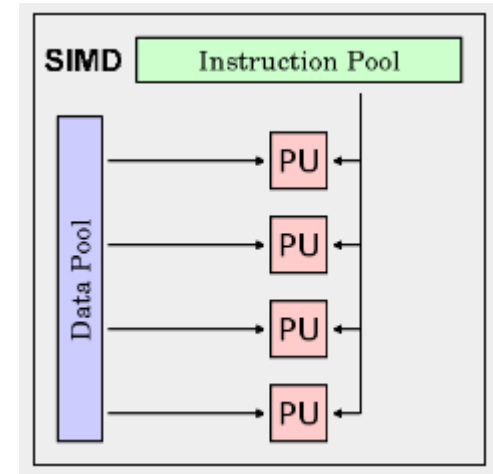


IBM 360



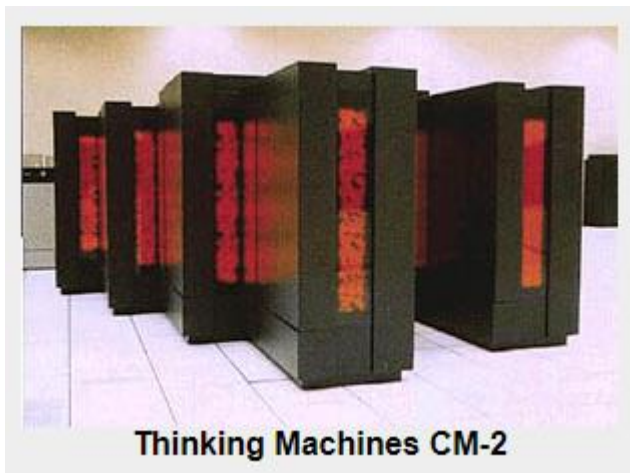
Clasificación por mecanismo de control: SIMD

- Todas las unidades de procesamiento ejecutan la misma instrucción sobre diferentes datos
- Ejecución sincrónica y determinística
- Hardware simplificado (comparten control)
- Pueden deshabilitarse selectivamente algunas unidades para que ejecuten o no instrucciones
- Resulta adecuado para problemas con alto grado de regularidad (por ejemplo, procesamiento de imágenes)



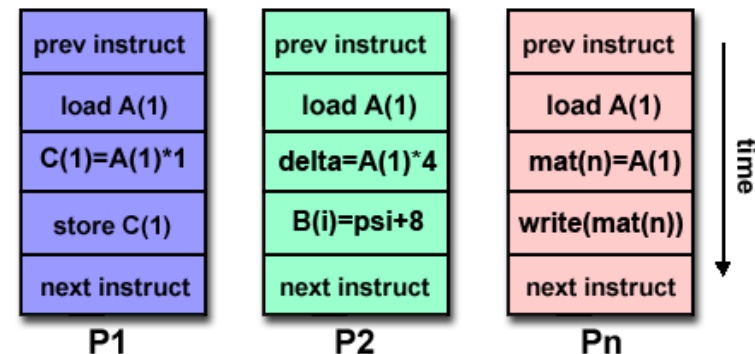
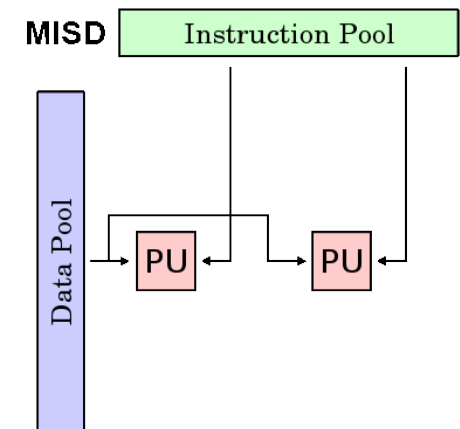
Clasificación por mecanismo de control: SIMD

- Ejemplos: Arreglos de procesadores (*Processor Arrays*) como el Thinking Machines CM-2
- En la actualidad, no existen procesadores SIMD “puros” pero sí hay algunos que incluyen componentes basados en este modo de procesamiento
 - Unidades vectoriales (MMX,SSE,AVX,AVX512,etc), GPUs



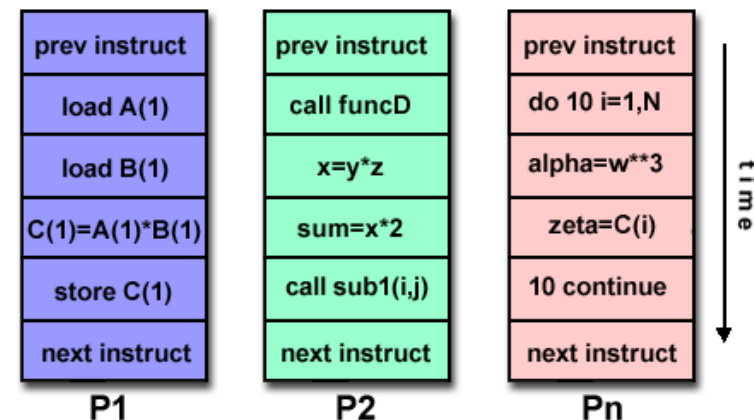
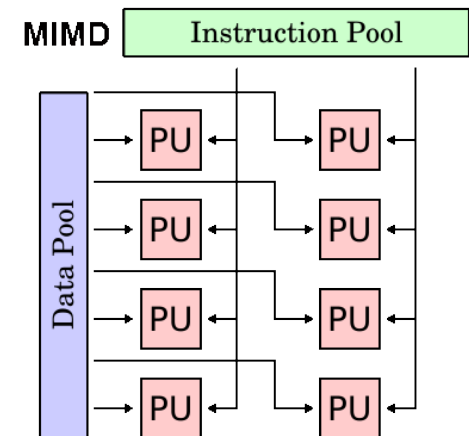
Clasificación por mecanismo de control: MISD

- Las unidades de procesamiento ejecutan diferentes instrucciones sobre el mismo dato
- No existen máquinas reales basadas en este modo de procesamiento
- Podrían usarse para:
 - Múltiples filtros de frecuencia aplicados a una misma señal
 - Múltiples algoritmos de cifrado intentando decodificar el mismo mensaje



Clasificación por mecanismo de control: MIMD

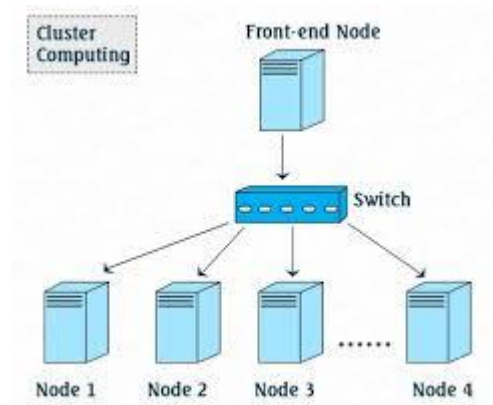
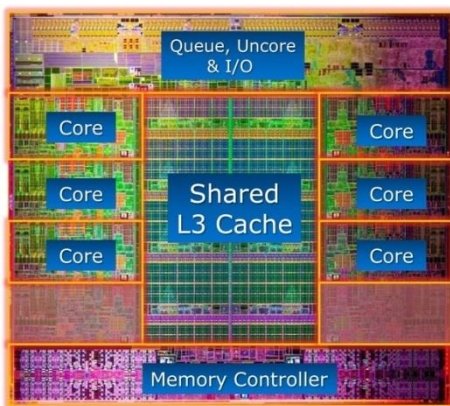
- Las unidades de procesamiento ejecutan diferentes instrucciones sobre diferentes datos
- La ejecución puede ser sincrónica o asincrónica, determinística o no determinística
- Pueden ser máquinas de memoria compartida o de memoria distribuida
- Es la clase más común de máquina paralela



Clasificación por mecanismo de control:

MIMD

- Ejemplos: procesadores multicore, clusters, multiprocesadores, grids, supercomputadoras



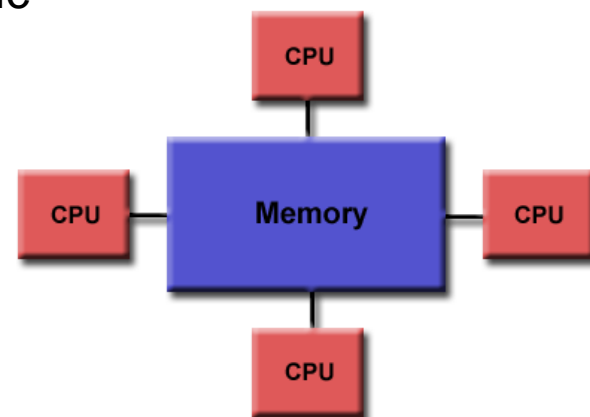
Clasificación por la organización física

- Se realiza de acuerdo al espacio de direcciones que tiene cada procesador → En otras palabras, tiene en cuenta la visión de la memoria principal por parte de cada procesador
- Las opciones son:
 - Memoria compartida
 - Memoria distribuida
 - Memoria híbrida

Clasificación por la organización física:

Memoria compartida

- Estos sistemas pueden tener variantes pero se caracterizan por la capacidad que tienen sus procesadores de acceder a toda la memoria como un único espacio de direcciones global
- Cuentan con múltiples procesadores que operan en forma independiente pero que comparten los mismos recursos de memoria → Esto significa que los cambios realizados por un procesador serán visibles para el resto
- Se necesita un mecanismo de coherencia de caché
- Sub-clasificación por modo de acceso a memoria
 - Acceso uniforme a memoria (UMA)
 - Acceso no uniforme a memoria (NUMA)



Clasificación por la organización física:

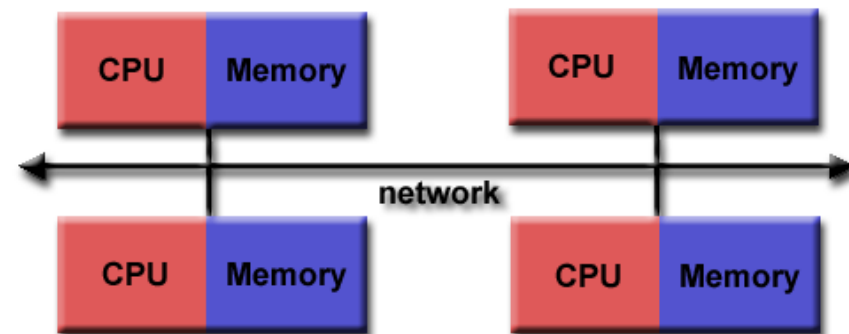
Memoria compartida

- Ventajas:
 - La programación de estos sistemas suele ser más fácil debido al espacio de direcciones global
 - La comunicación de datos entre las tareas es rápida y uniforme debido a la cercanía entre procesadores
- Desventajas:
 - La principal desventaja es la falta de escalabilidad entre la memoria y los procesadores → Agregar más procesadores incrementa el tráfico de memoria y complejiza los mecanismos de coherencia de caché
 - El programador se vuelve responsable de asegurar un «correcto» acceso a los datos

Clasificación por la organización física:

Memoria distribuida

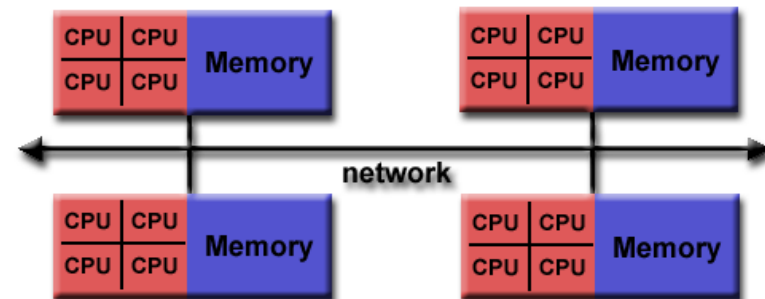
- Estos sistemas pueden tener variantes pero se caracterizan por requerir de una red de comunicación para poder conectar a sus procesadores
- Cada procesador opera en forma independiente y tiene su propia memoria, por lo que no hay un espacio de direcciones global → Esto significa que los cambios realizados por un procesador NO serán visibles para el resto
- Cuando un procesador necesita un dato que tiene otro, el programador se vuelve responsable de definir cómo y cuándo será comunicado
- No se requiere un mecanismo de coherencia de caché, ya que las memorias son locales
- Redes de interconexión: Ethernet, Infiniband, Myrinet



Clasificación por la organización física:

Memoria híbrida

- Debido al surgimiento de los procesadores multicore, los sistemas más grandes y rápidos del mundo de la actualidad combinan características de ambos modelos
- Múltiples máquinas de memoria compartida son inter-conectadas entre sí para permitir que sus procesadores puedan comunicarse
- Ventajas y desventajas
 - Lo que es común a ambos modelos
 - Soluciona el problema de escalabilidad de memoria compartida
 - Aumenta la complejidad de programación



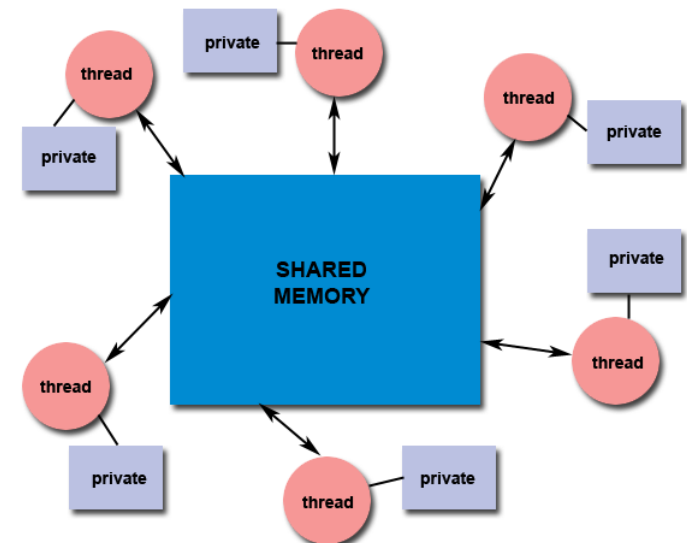
MODELOS DE PROGRAMACIÓN PARALELA

Modelos de programación paralela

- Existen diferentes modelos para desarrollar programas paralelos.
- Estos difieren básicamente en la forma en que las tareas de un programa comparten datos y se sincronizan.
- Son 2 las alternativas que se han mantenido a través de los años:
 - Memoria compartida
 - Pasaje de mensajes
- A comienzos de la década del 2000, con la aparición de los procesadores multicore, un nuevo modelo fue desarrollado el cual combina características de los 2 anteriores

Modelo de memoria compartida

- En el modelo de memoria compartida, múltiples tareas se ejecutan en forma concurrente (con la misma o diferente funcionalidad).
- Todas las tareas acceden a un mapa de memoria común y además cada una puede tener memoria local “exclusiva”.
- La comunicación y sincronización de estas tareas se realiza escribiendo y leyendo áreas de memoria compartida.
- Generalmente usado en plataformas de memoria compartida como multiprocesadores o multicores.



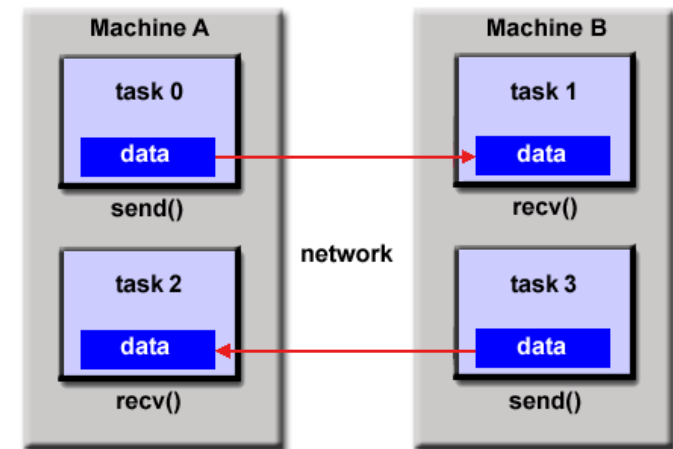
Modelo de memoria compartida

- El programador en general no maneja la distribución de los datos ni lo relacionado a la comunicación de los mismos.
- Ventaja: Transparencia para el programador. La ubicación de los datos, su replicación y su migración son transparentes.
- Desventaja: A veces es necesario trabajar sobre esos aspectos para mejorar el rendimiento. Además, resulta difícil la predicción de performance a partir de la lectura del algoritmo.



Modelo de pasaje de mensajes

- Consiste de p procesos (eventualmente procesadores), cada uno de ellos con su espacio de direcciones exclusivo.
- Característica clave: Espacio de direcciones particionado.
 - Cada dato pertenece a una partición.
 - Toda interacción requiere la cooperación de dos procesos.
- El intercambio de mensajes sirve para varios propósitos:
 - Intercambio explícito de datos (programador).
 - Sincronizar procesos.
- Generalmente usado en plataformas de memoria distribuida como clusters



Modelo de pasaje de mensajes

- Ventajas:

- El programador tiene total control para lograr sistemas más eficientes y escalables.
- Puede implementarse eficientemente en muchas arquitecturas paralelas.
- Más fácil de predecir el rendimiento.



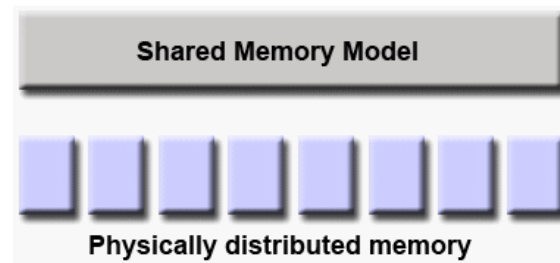
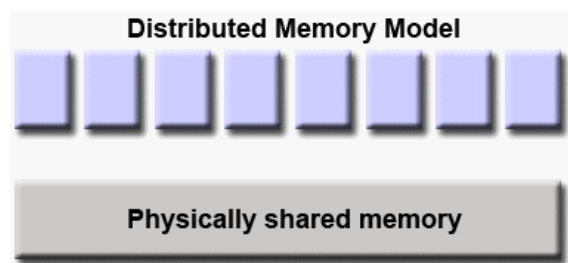
- Desventajas:

- Mayor complejidad al implementar estos algoritmos para lograr alto rendimiento.



Modelos de programación y arquitecturas paralelas

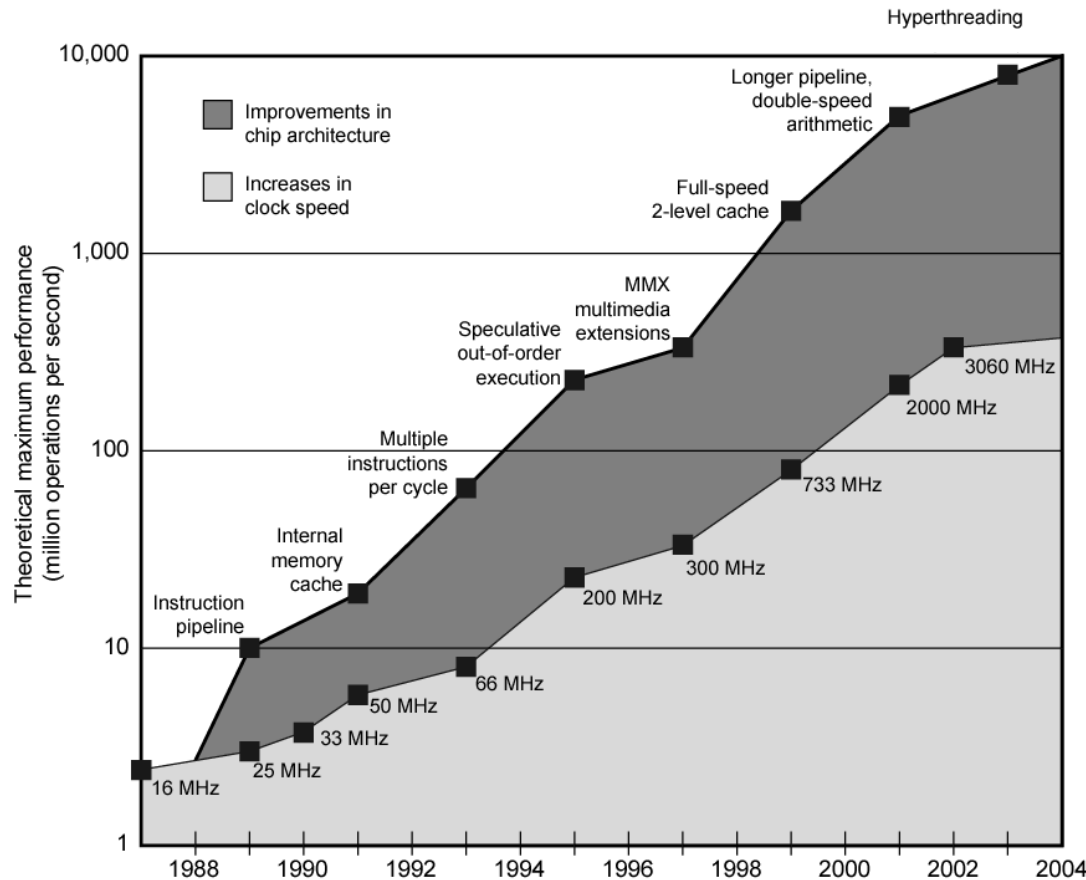
- Se suele asociar cada modelo con su correspondiente arquitectura paralela.
- Sin embargo, estos modelos son abstracciones y, en principio, pueden emplearse en cualquier plataforma paralela. Por ejemplo:
 - Un modelo de memoria compartida sobre una máquina de memoria distribuida
 - Un modelo de pasaje de mensajes sobre una máquina de memoria compartida



- El uso de un modelo que no resulte “natural” para la arquitectura subyacente, probablemente provoque una degradación del rendimiento.

EVOLUCIÓN DE LOS PROCESADORES

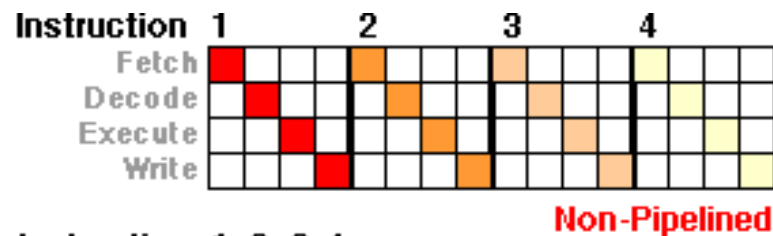
Modelo tradicional de mejora de los procesadores



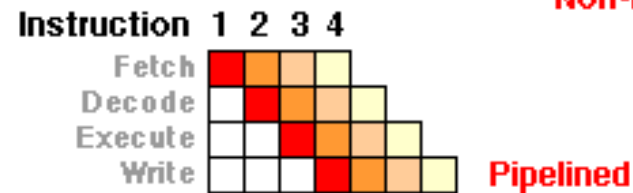
- Guiado por:
 - Aumento en el número de transistores en el chip
 - Aumento de la frecuencia del reloj
- Permitted implement advanced processing techniques that improve the performance of applications (Parallelism at Instruction Level, ILP)

ILP - Pipelining

- El *pipelining* consiste básicamente en solapar las diferentes etapas de la ejecución de instrucciones, reduciendo el tiempo de ejecución total



16 ciclos de reloj



7 ciclos de reloj

Cycles/Time →

ILP - Pipelining

- El modelo de referencia es el de “línea de montaje”
 - Supongamos que el armado de un auto requiere 100 unidades de tiempo y que puede ser dividido en 10 etapas en forma de pipeline de 10 unidades cada una → se armaría 1 auto cada 10 unidades de tiempo
- Para aumentar la velocidad del pipeline, podemos dividirlo en etapas más pequeñas incrementando su profundidad.
 - En el contexto de los procesadores, esto permite a su vez mayores frecuencias de reloj.
 - Tener en cuenta que la velocidad de un pipeline está limitado por la duración de su etapa más costosa → Etapas más pequeñas aceleran el pipeline.

ILP - Pipelining

- Si dividir nos permite aumentar la velocidad del pipeline, ¿podemos hacerlo infinitamente? ¿es útil?

Microarquitectura Intel	Número de etapas en el pipeline
P5 (Pentium)	5
P6 (Pentium 3)	10
P6 (Pentium Pro)	14
NetBurst (Willamette)	20
NetBurst (Northwood)	20
NetBurst (Prescott)	31
NetBurst (Cedar Mill)	31
Core	14
Bonnell	16
Sandy Bridge	14
Silvermont	14-17
Haswell	14
Skylake	14
Kabylake	14

- Prescott mejoró modestamente respecto a su predecesor a costo de una organización más compleja y un mayor consumo energético.
- Uno de los principales problemas para pipelines profundos es que estadísticamente hay un salto condicional cada 5-6 instrucciones!

ILP - Pipelining

- Consecuencias de los saltos condicionales:

Time →

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

Ejemplo de pipeline de 6 etapas sin saltos condicionales → Máximo aprovechamiento

ILP - Pipelining

- Consecuencias de los saltos condicionales:

	Time →							← Branch penalty						
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO							
Instruction 5					FI	DI	CO							
Instruction 6						FI	DI							
Instruction 7							FI							
Instruction 15								FI	DI	CO	FO	EI	WO	
Instruction 16									FI	DI	CO	FO	EI	WO

Ejemplo de pipeline de 6 etapas donde la instrucción 3 es un salto condicional a la instrucción 15 → Pérdida de rendimiento

ILP - Pipelining y ejecución especulativa

- Pipelines profundos requieren de técnicas efectivas que sean capaces de predecir los saltos en forma especulativa
 - La penalización de un salto mal predicho se incrementa a medida que el pipeline es más profundo → más instrucciones adelantadas son las que se pierden
 - Estos factores limitan la profundidad de pipeline y su posible ganancia de rendimiento
- ¿Cómo mejorar entonces la tasa de ejecución de instrucciones?
- Una forma obvia consiste en usar múltiples pipelines
 - Durante cada ciclo de reloj, múltiples instrucciones son emitidas en paralelo, las cuales son ejecutadas en múltiples unidades funcionales

ILP - Pipelining y ejecución superescalar

1. load R1, @1000
2. load R2, @1008
3. add R1, @1004
4. add R2, @100C
5. add R1, R2
6. store R1, @2000

(i)

1. load R1, @1000
2. add R1, @1004
3. add R1, @1008
4. add R1, @100C
5. store R1, @2000

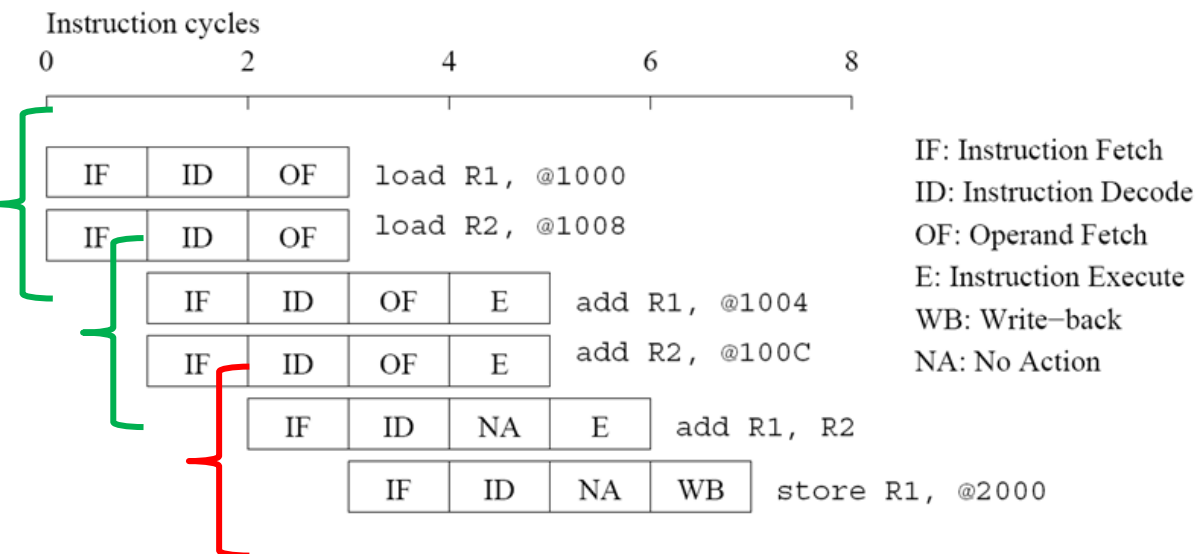
(ii)

1. load R1, @1000
2. add R1, @1004
3. load R2, @1008
4. add R2, @100C
5. add R1, R2
6. store R1, @2000

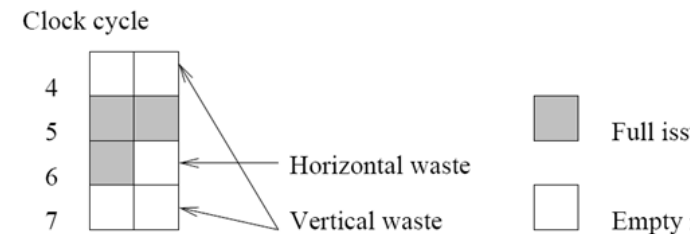
(iii)

(a) Three different code fragments for adding a list of four numbers.

Procesador con 2 pipelines
y la habilidad de emitir
simultáneamente 2
instrucciones
(*superescalar*)



(b) Execution schedule for code fragment (i) above.



Adder Utilization

(c) Hardware utilization trace for sched

ILP - Pipelining y ejecución superescalar

- No hay una única manera de escribir un programa y la misma tiene incidencia en el rendimiento final
- El ideal sería que todas las etapas estén activas en todo momento (máximo paralelismo).
 - En la práctica, es muy difícil que ocurra
- Al momento de realizar la planificación de instrucciones, se deben tener en cuenta:
 - Dependencia verdadera de datos (*True data dependency*): el resultado de una instrucción es la entrada para la siguiente
 - Dependencia de recurso (*Resource dependency*): dos operaciones requieren el mismo recurso (por ejemplo, unidad de punto flotante)
 - Dependencia de salto (*Branch dependency*): las instrucciones a ejecutar después de un salto condicional no pueden ser determinadas a priori sin tener margen de error.

ILP - Pipelining y ejecución superescalar

- El planificador (hardware) analiza un conjunto de instrucciones de la cola de instrucciones a ejecutar y emite aquellas que pueden ser ejecutadas en forma concurrente, teniendo en cuenta las dependencias.
 - Si las instrucciones son ejecutadas en el orden en que aparecen en la cola, se dice que la emisión es *en orden* → Simple pero limita significativamente la emisión de instrucciones
 - Si el procesador tiene la habilidad de reordenar las instrucciones en la cola, entonces se puede alcanzar el máximo rendimiento posible → este modelo se conoce como *fuera de orden* (o de *emisión dinámica*) y, aunque es más complejo, es el que se usa en la actualidad

ILP - Pipelining y ejecución fuera-de-orden

Cycle	1	2	3	4	5	6	7	8	9
Instr ₁	Fetch	Decode	Execute			Write			
Instr ₂		Fetch	Decode	Wait		Execute	Write		
Instr ₃			Fetch	Decode	Wait		Execute	Write	
Instr ₄				Fetch	Decode	Wait		Execute	Write
Instr ₅					Fetch	Decode	Wait		Execute
Instr ₆						Fetch	Decode	Wait	

*Ejecución
en-orden*

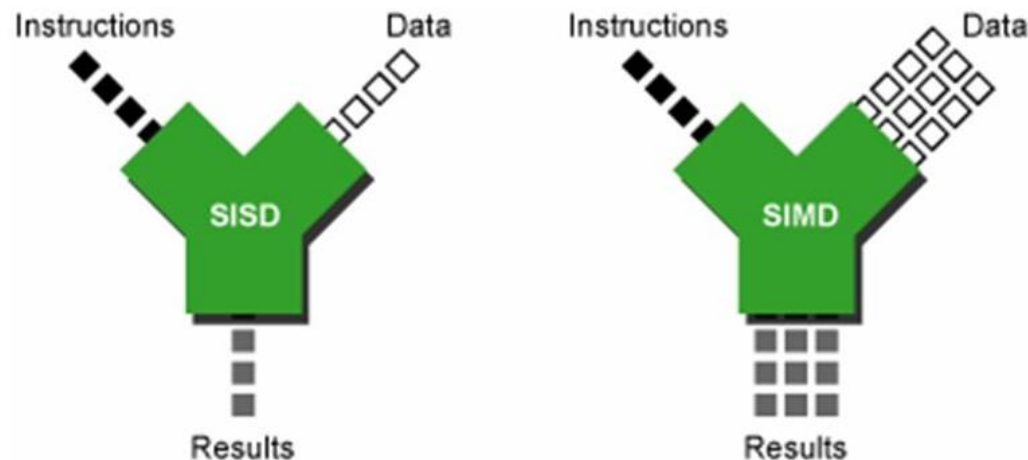
Cycle	1	2	3	4	5	6	7	8	9
Instr ₁	Fetch	Decode	Execute			Write			
Instr ₂		Fetch	Decode	Wait		Execute	Write		
Instr ₃			Fetch	Decode	Execute	Write			
Instr ₄				Fetch	Decode	Wait	Execute	Write	
Instr ₅					Fetch	Decode	Execute	Write	
Instr ₆						Fetch	Decode	Execute	Write

*Ejecución
fuera-de-orden*

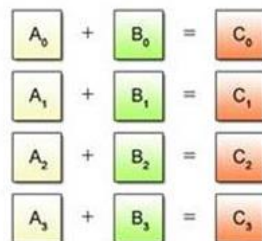
En resumen, el rendimiento de un procesador superescalar está limitado por la cantidad disponible de paralelismo a nivel de instrucciones.

ILP - Instrucciones SIMD

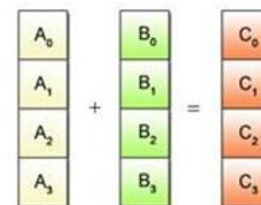
- Incorporación de unidades de procesamiento vectorial que permiten ejecutar una instrucción sobre diferentes datos en un ciclo de reloj



(a) Scalar Operation



(b) SIMD Operation

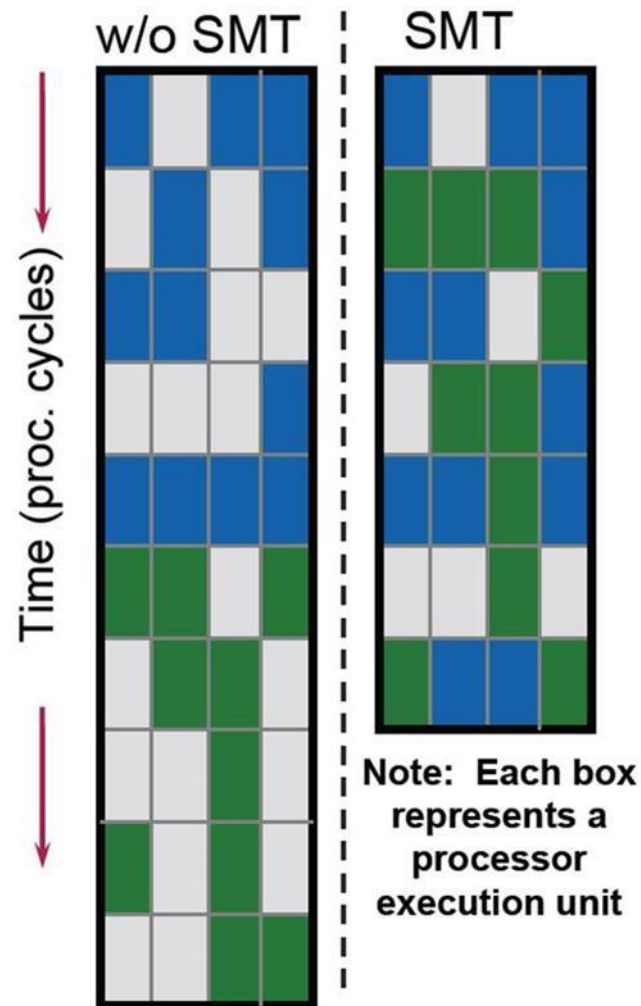


ILP - Multi-hilado a nivel hardware

- En general, un único hilo de ejecución no resulta suficiente para aprovechar la potencia de los procesadores superescalares
- La técnica *Simultaneous Multi-Threading* (SMT) consiste en mantener más de un hilo de ejecución al mismo tiempo en el procesador¹
 - Los recursos asociados al estado del procesador son replicados una o más veces (contador de programa, registros, punteros, pila, etc) manteniendo el número original de recursos de ejecución (unidades funcionales, cachés, interfaces de memoria, etc) → sólo requiere un pequeño incremento en el tamaño del chip
 - Con esta replicación, el procesador parece tener múltiples núcleos (a veces llamados *procesadores lógicos*) y por lo tanto puede ejecutar múltiples flujos (hilos) en paralelo, sin importar si pertenecen al mismo programa o a diferentes
 - El número de replicas de estados determina el número de procesadores lógicos del procesador

¹En los procesadores de Intel esta técnica se denomina Hyper-Threading

ILP - Multi-hilado a nivel hardware

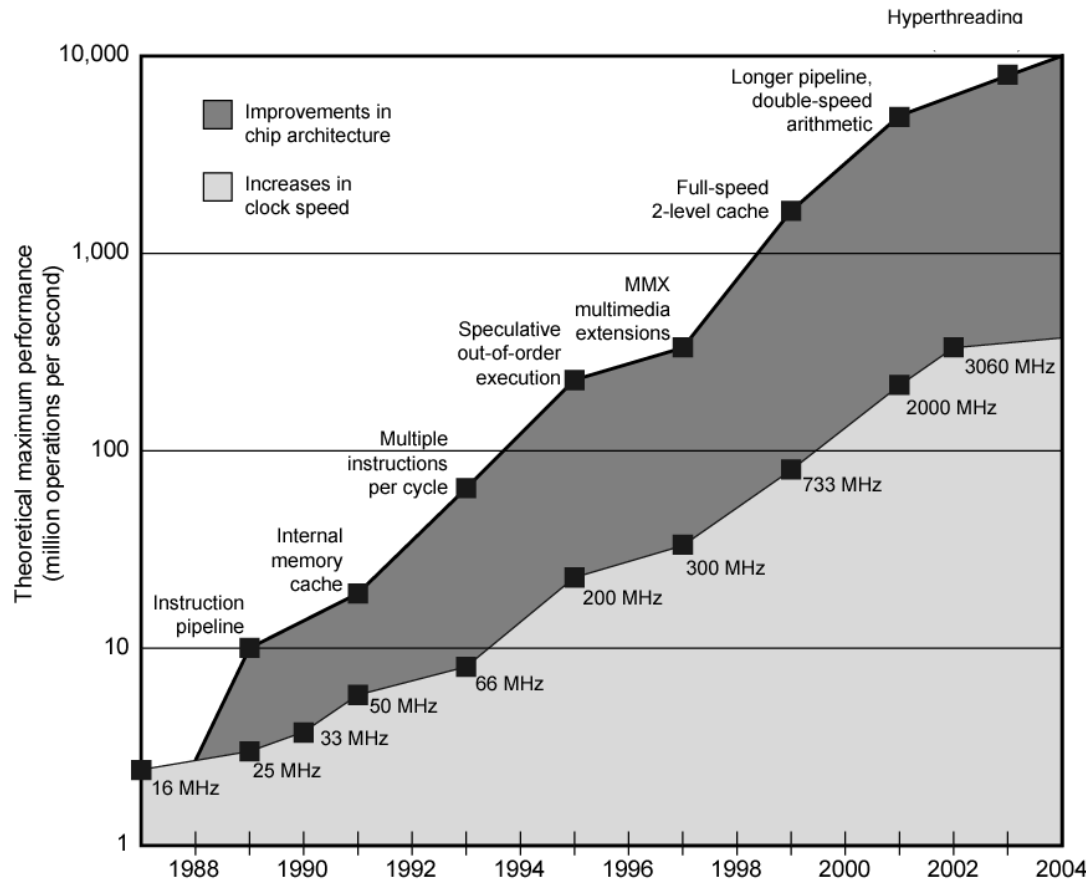


ILP - Multi-hilado a nivel hardware

- SMT puede mejorar la productividad del procesador (instrucciones ejecutadas por ciclo) siempre y cuando sea posible intercalar instrucciones de múltiples hilos entre los pipelines
- El escenario ideal sería tener múltiples hilos que usen recursos de ejecución diferentes → Lamentablemente esto no es común en la práctica
 - En ocasiones, el número de referencias a memoria de un programa escala con el número de hilos por lo que se puede dar un mejor aprovechamiento del ancho de banda si tenemos un gran número de hilos
- Desventaja de SMT: Si los hilos usan exactamente los mismos recursos, podría haber pérdida de rendimiento por la competencia entre ellos. Por ejemplo, programas sensibles al tamaño de caché.

La ganancia por el uso de SMT depende fuertemente del programa a ejecutar. Generalmente, lo que conviene es hacer pruebas con y sin uso de SMT para evaluar su posible beneficio.

Modelo tradicional de mejora de los procesadores

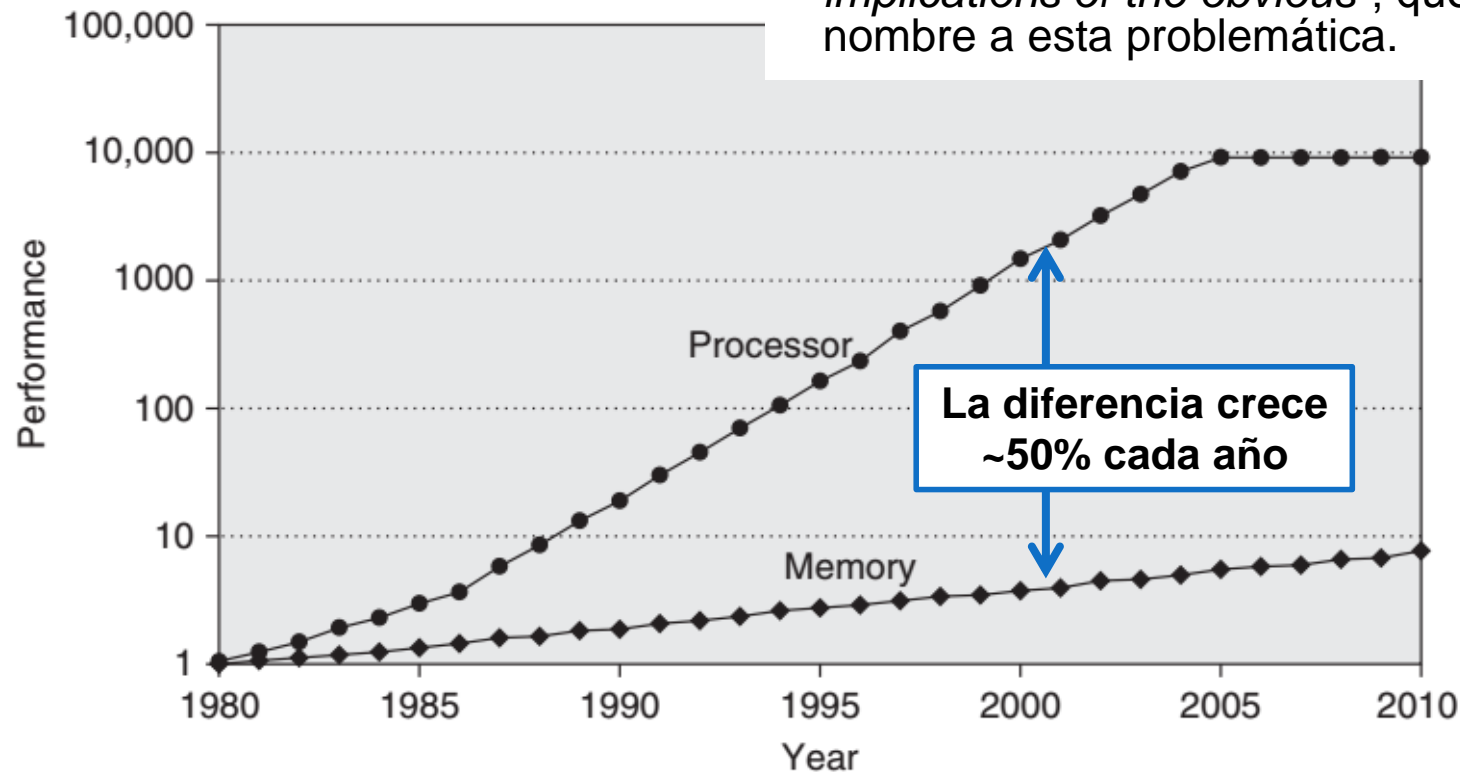


- A principios de la década del 2000 llegó a su límite, debido a 3 factores:

- *Memory Wall*
- *ILP Wall*
- *Power Wall*

Memory Wall

- En 1994, Bill Wulf y Sally McKee publican el artículo "*Hitting the memory wall: implications of the obvious*", que pone nombre a esta problemática.



- El número de ciclos de reloj por acceso a memoria se incrementa año a año (~50%).
- En el artículo, los autores concluyen que, de no mediar cambios disruptivos, llegará un momento en que el rendimiento estará dominado por la velocidad de memoria.

ILP Wall

- Si bien es posible agregar más unidades funcionales al chip, no se alcanzaría una mejora de rendimiento para la mayoría de las aplicaciones debido a que no es posible extraer más ILP de los programas.
- Problemas:
 - Limitaciones de los compiladores
 - Dependencias entre instrucciones
 - Imposibilidad de predecir saltos
 - Cantidad limitada de paralelismo intrínseco

Power Wall

- La potencia usada por el procesador se transforma en calor que debe disiparse
- Al aumentar la frecuencia de reloj, se incrementa la temperatura y se requiere más energía para mantener refrigerado al procesador → El problema es que la relación no es lineal

Model Number	Cores	Frequency	L2 cache	L3 cache	TDP	Socket	I/O bus
Xeon E5-4640	8	2.4 GHz	8 × 256 KB	20 MB	95 W	LGA 2011	2 × 8.0 GT/s QPI
Xeon E5-4650	8	2.7 GHz	8 × 256 KB	20 MB	130 W	LGA 2011	2 × 8.0 GT/s QPI

Incremento en la frecuencia de reloj: 12.8%

Incremento de la potencia: 36.8%

Power Wall

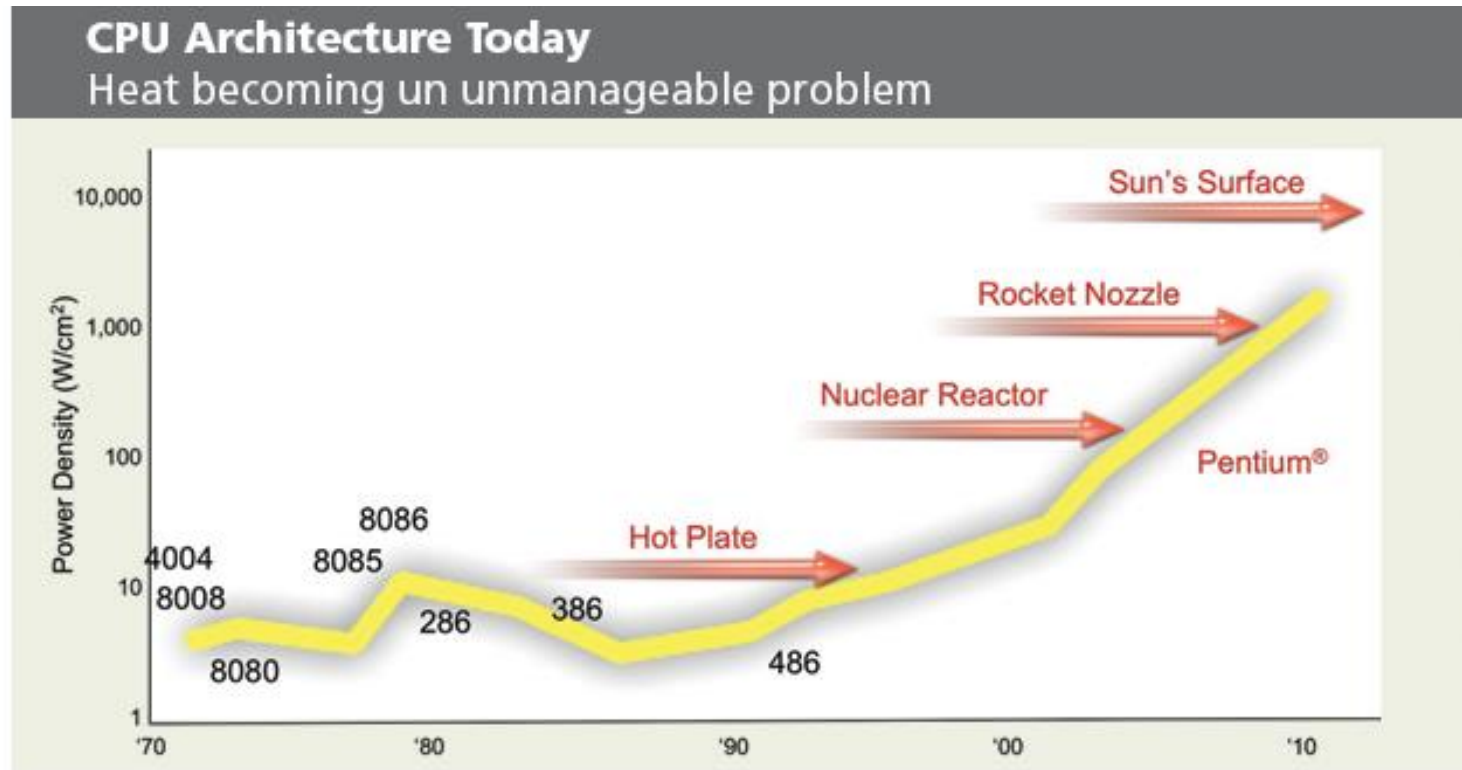


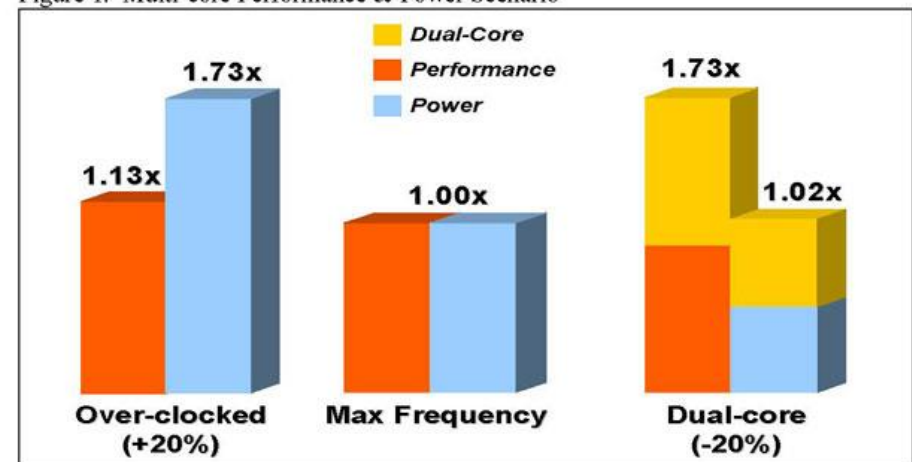
Figure 1. In CPU architecture today, heat is becoming an unmanageable problem.
(Courtesy of Pat Gelsinger, Intel Developer Forum, Spring 2004)

- Conclusión: no es posible incrementar la frecuencia de reloj con los mecanismos de refrigeración actuales

Cambio de paradigma en la mejora de los procesadores

- Fue necesario buscar otras alternativas en el diseño de los procesadores que permitieran continuar incrementando su rendimiento.
- En lugar de seguir aumentando la compleja organización interna del chip, las empresas fabricantes optaron por integrar dos o más núcleos computacionales (también llamados *cores*) más simples en un sólo chip (*multicores*).
- Los procesadores multicore proveen el potencial de mejorar el rendimiento sin necesidad de aumentar la frecuencia de reloj, lo que los vuelve más eficientes energéticamente.

Figure 1. Multi-core Performance & Power Scenario



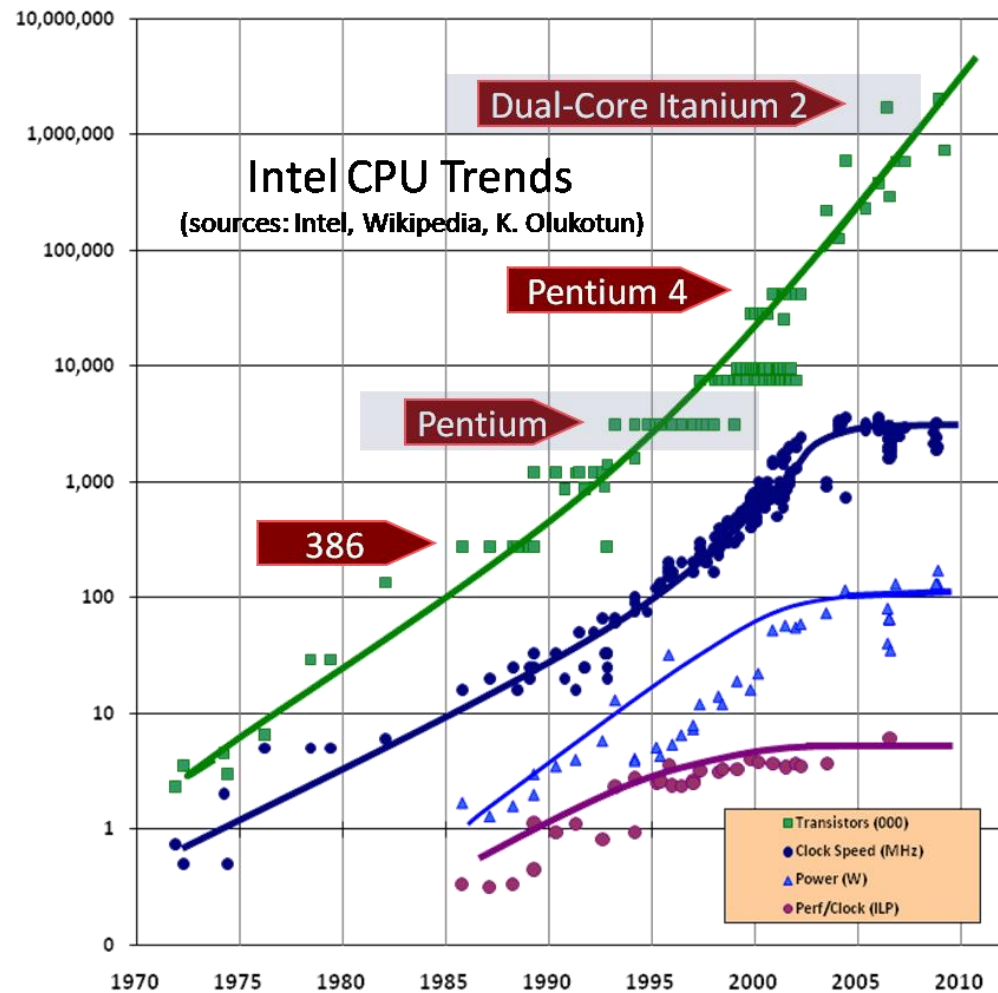
Cambio de paradigma en la mejora de los procesadores

¿Qué impacto tuvo la transición hacia los procesadores multicore para el software?

Necesidad de explotar paralelismo explícito para aprovechar potencia del procesador

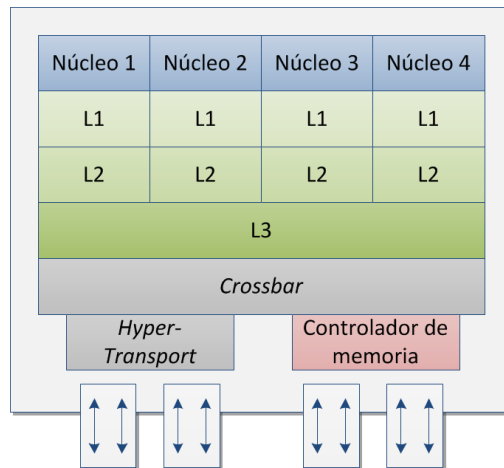
Aplicaciones, sistemas operativos, librerías, middlewares, etc, debieron ser re-programados

Evolución de los procesadores

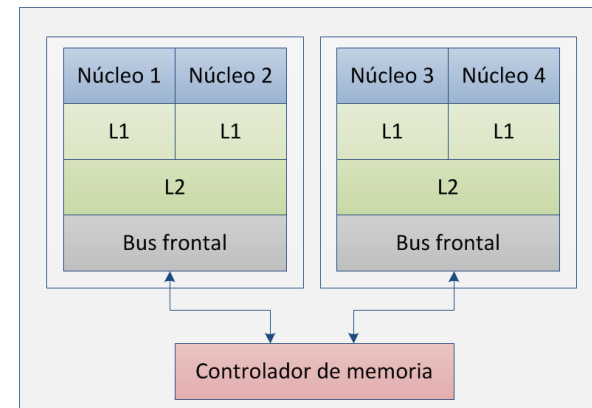


Procesadores multicore

- Desde su desarrollo, los procesadores multicore han sofisticado su diseño en las sucesivas familias.
- Los primeros procesadores de este tipo eran prácticamente dos procesadores mononúcleo en la misma oblea.
- Las siguientes generaciones han incrementado el número de núcleos e incorporado niveles de caché L2 y L3 (con o sin compartición).



AMD Opteron



Intel Xeon

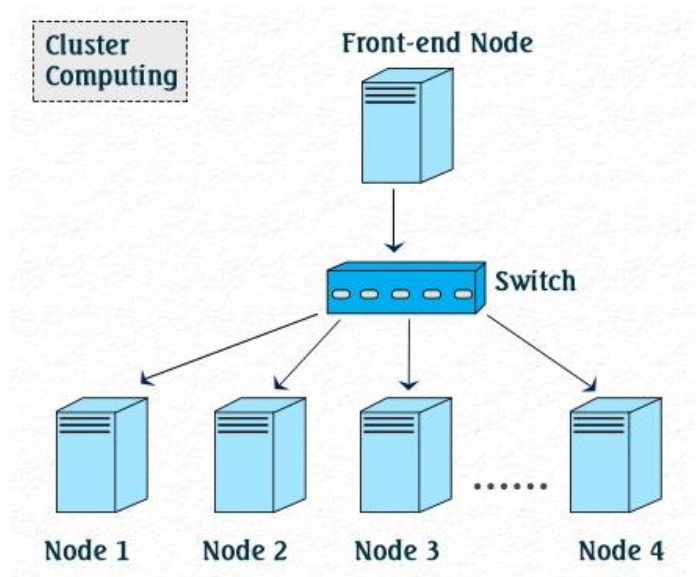
CLUSTERS

Clusters

- Se puede definir como:

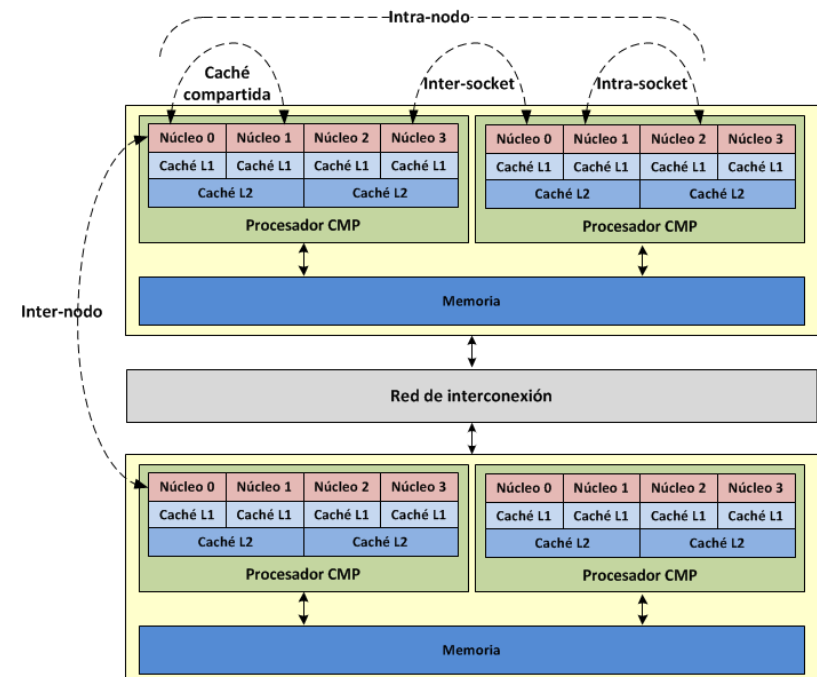
Colección de computadoras individuales interconectadas vía algún tipo de red, que trabajan en conjunto como un único recurso integrado de cómputo

- Cada nodo de procesamiento es un sistema de cómputo en sí mismo, con hardware y sistema operativo propio.
- La red de interconexión puede ser Ethernet o redes específicas de alta velocidad como Infiniband o Myrinet
- Puede ser homogéneo o heterogéneo
- Según clasificación vista: ?
- Ofrecen una buena relación costo-rendimiento y son fáciles de expandir



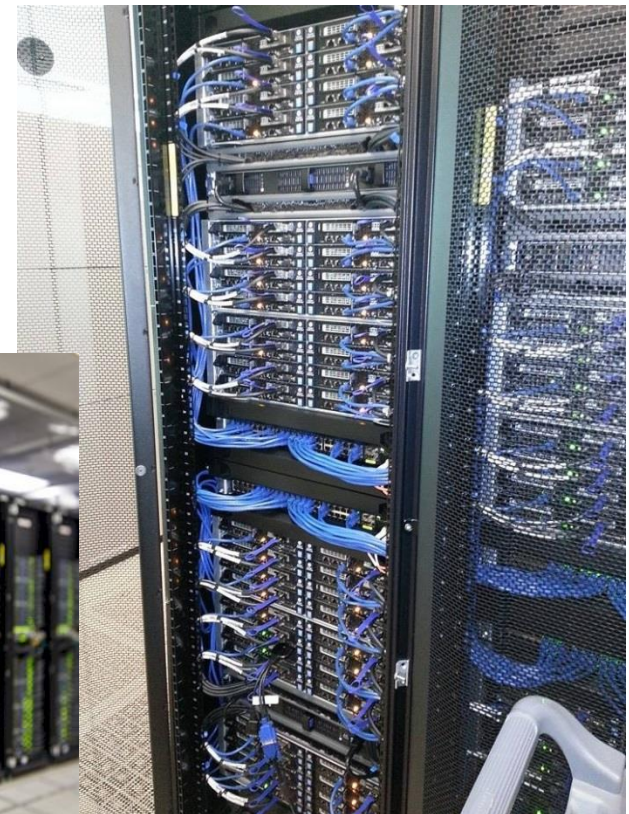
Clusters

- Se volvieron populares en la década del '90, por la posibilidad de lograr alto rendimiento a bajo costo
- Cluster de tipo *Beowulf*
- En la actualidad, la mayoría de los grandes sistemas de cómputo se basan en clusters de nodos multi/many-core, conformado una arquitectura híbrida



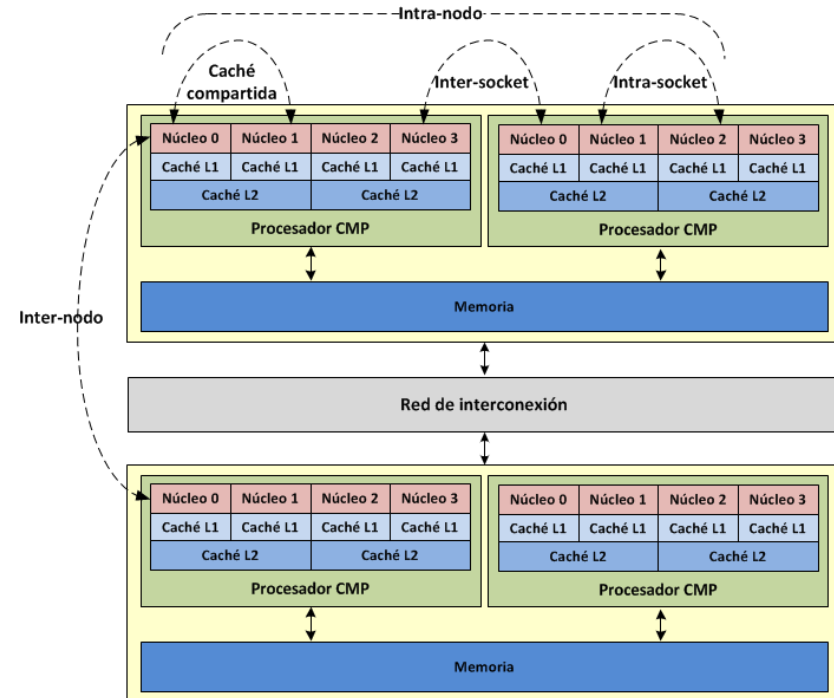
Clusters

- En la actualidad, la mayoría de los grandes sistemas de cómputo se basan en clusters de nodos multi/many-core
- En el último listado del TOP500 (Noviembre de 2018), 87% de los supercomputadores en el ranking se componen de clusters



Clusters

- Dado que los clusters son arquitecturas distribuidas, el modelo de programación más utilizado suele ser pasaje de mensajes
- También es posible emplear el modelo de memoria compartida, aunque a costo de menor rendimiento
- Con la incorporación de los procesadores multicore a los clusters, surgió un nuevo modelo de programación que combina pasaje de mensajes con memoria compartida



Bibliografía usada para esta clase

- Capítulo 1, “Introduction to High Performance Computing for Scientists and Engineers”. Georg Hager, Gerard Wellein. CRC Press (2011).
- Capítulo 1, “An introduction to parallel programming”. Peter Pacheco. Elsevier (2011).
- “Introduction to parallel computing” Blaise Barney, Lawrence Livermore National Laboratory.
https://computing.llnl.gov/tutorials/parallel_comp
- Capítulo 5, “Introduction to Parallel Computing”. Grama, Gupta, Karypis, Kumar. Addison Wesley (2003).
- Capítulo 1, “Computer Organization and Architecture. Designing for Performance”. W. Stallings (2010)