Tema 1 Conceptos Básicos y Motivación

Programación Paralela

José María Cecilia

Grado en Ingeniería informática

José M. Cecilia- Tlf: (+34) 968 278 587
Universidad Católica San Antonio de Murcia - Tlf: (+34) 968 00 00 00 info@ucam.edu - www.ucam.edu

Bases Fundadas

"La programación paralela está basada en el rendimiento. De otra forma escribirías una aplicación secuencial. Para aquellos interesados en aprender programación paralela, el problema es donde encontrar una arquitectura paralela dedicada a tu aplicación. Es difícil obtener aceleraciones interesantes (speedups), compartiendo la máquina o si no es realmente paralela. Una respuesta es la Unidad de procesamiento gráfico (GPU), que puede tener cientos de cores, y se encuentra en millones de PC's y portátiles"

David Patterson
Director, The Parallel Computing Research Laboratory y
profesor de informática en U.C Berkeley.

José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

Agenda

Motivación

Historia del paralelismo

GPUs como arquitecturas paralelas

Arquitectura básica de ordenadores

Algunos conceptos básicos de C

José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

Agenda

Motivación

Historia del paralelismo

GPUs como arquitecturas paralelas

Arquitectura básica de ordenadores

Algunos conceptos básicos de C

José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

Antecedentes

- **Programadores confiaban en los progresos del hardware.**
  - Aplicaciones secuenciales corren más rápido mágicamente en cada generación de procesadores (Incremento de la frecuencia).
- **Barrera de frecuencia en los procesadores.**
  - Consumo de Energía.
  - Disipación de Calor.
  - Límite 8,67 GHz (ver video AMD)
- **Ley de Moore se sigue cumpliendo (Almost Die).**
  - Número de transistores se dobla cada 1,5-2 años
  - **Solución: Múltiples unidades de procesamiento (cores) en el chip.**
- **El rendimiento depende del programador.**

Proyecciones Futuras

	Tecnología de 2010	Tecnología de 2020
A nivel de core	1 GFLOPS	5 GFLOPS
A nivel de nodo y placa	10 x	1000 x
A nivel de cluster y rack	100.000 x	500.000 x
Potencia total:	1 PetaFLOPS (15 ceros)	1 ExaFLOPS (18 ceros)

Pregunta: ¿Por qué avanzaremos tan poco a nivel de core?

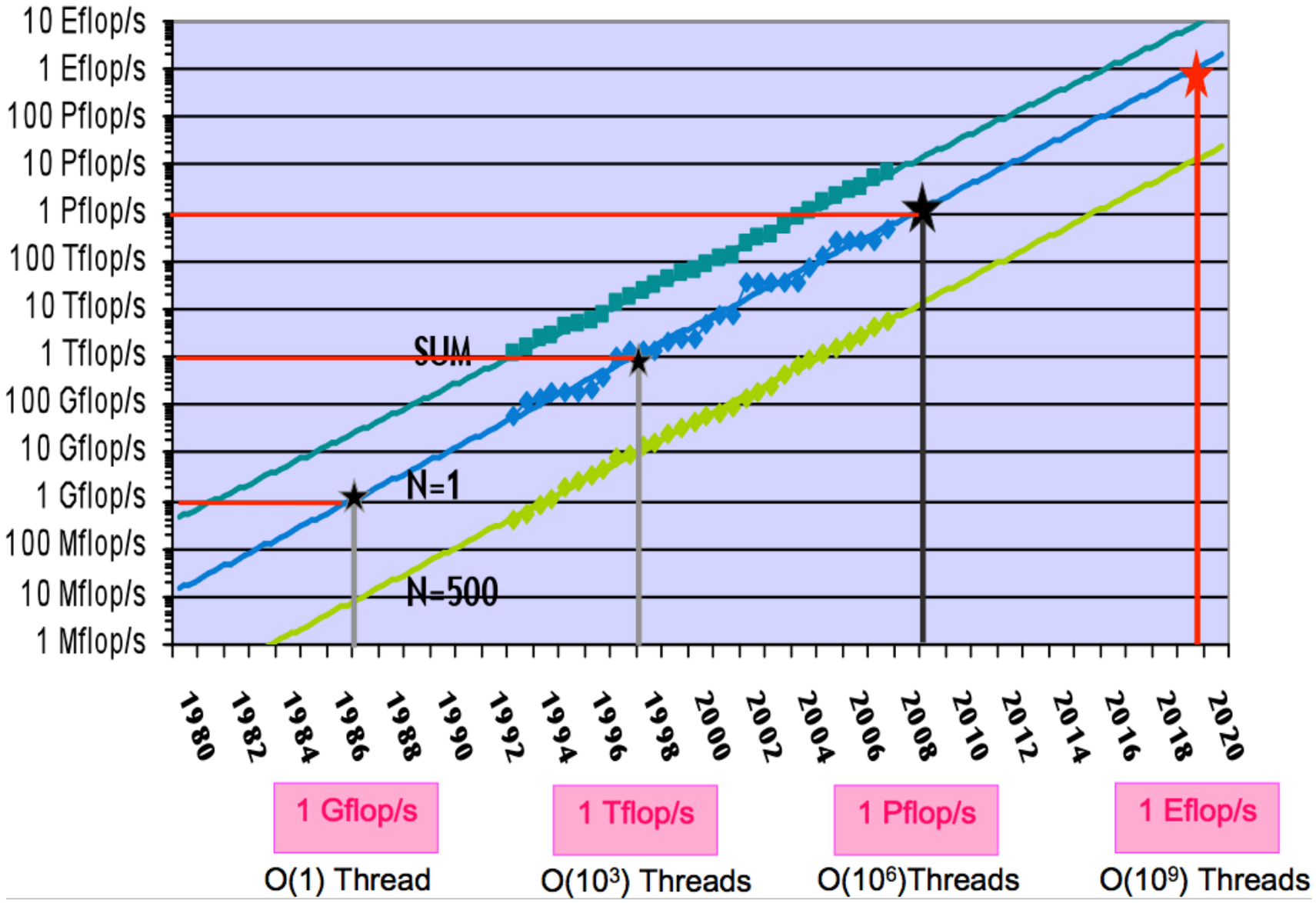
José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

Respuesta: Porque la frecuencia no ayuda. Predicciones de la ITRS en MHz

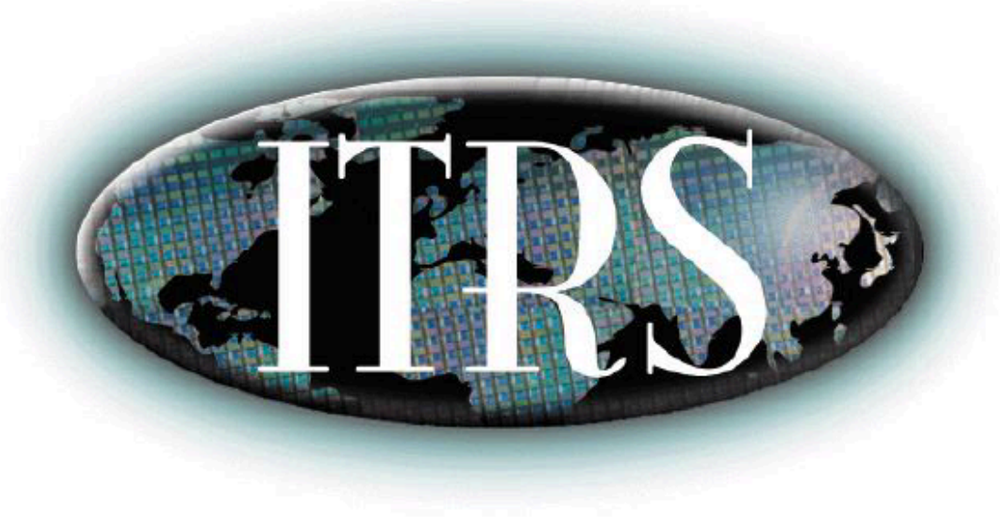
Año	Según el informe ITRS de 2007	Según el informe ITRS de 2011
2010	5 875	3 600
2011	6 329	3 744
2012	6 817	3 894
2013	7 344	4 050
2014	7 911	4 211
2015	8 522	4 380
2016	9 180	4 555
2017	9 889	4 737
2018	10 652	4 927
2019	11 475	5 124
2020	12 361	5 329
2022	14 343	5 764
2024	16 640	6 234

José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

Rendimiento esperado:
Proyecciones futuras según el ritmo evolutivo



El oráculo tecnológico: http://www.itrs.net
Intl Technology Roadmap for Semiconductors



The ITRS is Jointly Sponsored by

European Semiconductor Industry Association

Japan Electronics and Information Technology Industries Association

Korea Semiconductor Industry Association

Taiwan Semiconductor Industry Association

Semiconductor Industry Association

INTERNATIONAL TECHNOLOGY ROADMAP FOR SEMICONDUCTORS

2011 EDITION

EXECUTIVE SUMMARY

THE ITRS IS DEVISED AND INTENDED FOR TECHNOLOGY ASSESSMENT ONLY AND IS WITHOUT REGARD TO ANY COMMERCIAL CONSIDERATIONS PERTAINING TO INDIVIDUAL PRODUCTS OR EQUIPMENT.

ACKNOWLEDGMENTS

INTERNATIONAL ROADMAP COMMITTEE
- *Europe*—Patrick Cogez, Mart Graef, Bert Huizing, Reinhard Mahnkopf
- *Japan*—Hidemi Ishiuchi, Junji Shindo
- *Korea*—Siyoung Choi, JaeSung Roh,
- *Taiwan*—Carlos H. Diaz, Burn Lin
- *U.S.A.*—Bob Doering, Paolo Gargini, Ian Steff

TECHNOLOGY WORKING GROUP KEY CONTRIBUTORS

2011 Cross TWG Study Group (Technology Pacing)—Alan Allan, Sitaram Arkalgud, Dave Armstrong, Joel Barnett, Roger Barth, Herb Bennett, Bill Bottoms, Juan-antonio Carballo, Chris Case, David Chan, Bill Chen, Alain Diebold, Bob Doering, Denny Fandel, Paul Feeney, Mike Gaitan, Mike Garner, Christina Hacker, Dan Herr, Jim Hutchby, Hirofumi Inoue, Raj Jammy, Scott Jones/ICKnowledge, Andrew Kahng, Leo Kenny, Larry Larson, Marcus Lentz, Mike Lercel, Rich Liu, Jurgen Lorenz, Ichiro Matsuo, Andreas Neuber, Kwok Ng, Andreas Nutsch, Dilip Patel, Jack Pekarik, Lothar Pfitzner, Gopal Rao, Mike Rodgers, Thomas Skotnicki, Hitoshi Wakabayashi, Linda Wilson, Osamu [Sam] Yamazaki, Victor Zhirnov, Paul Zimmerman

Programación paralela hasta ahora

- **Hasta ahora:**
  - Programación paralela en clusters desde décadas
  - Solo para grandes organizaciones.
  - Muy pocos programadores paralelos.
- **Ahora**
  - Todos los chips son paralelos (2, 4, 8, 12(Magny-cours)).
  - Incremento del mercado de la programación paralela.
  - Oportunidad de trabajo.

José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

Del lenguaje natural a los electrones

Natural Language (e.g, English)

Algorithm

High-Level Language (C/C++…)

Compiler →

Instruction Set Architecture (ISA)

Microarchitecture

Circuits

Electrons

©Yale Patt and Sanjay Patel, *From bits and bytes to gates and beyond*

José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

Agenda

Motivación

Historia del paralelismo

GPUs como arquitecturas paralelas

Arquitectura básica de ordenadores

Algunos conceptos básicos de C

José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

El programa al nivel del ISA

- Un programa es un conjunto de instrucciones almacenadas en memoria, que pueden ser leídas, interpretadas y ejecutadas por el hardware.

- Las instrucciones del programa operan con los datos almacenados en memoria o dados por operaciones de I/O.

José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

A nivel de programa

- Toda instrucción necesita ser cargada de memoria, decodificada, y entonces ejecutada.
- Tres tipos: Operaciones, Transferencias de datos, y Control de flujo de programa.

Ejemplo de un ciclo de instrucción (Pipeline):

Fetch | Decode | Execute | Memory

José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

Ejemplo ISA

- Ejemplo de una instrucción de operación
  ADD R1, R2, R3
  Ciclo de Instrucción para ADD:
  Fetch | Decode | Execute | Memory
- Ejemplos de transferencias de datos:
  LDR R1, R2,
  STR R1, R2, #2
  Ciclo de Instrucción para LDR y STR
  Fetch | Decode | Execute | Memory
- Ejemplo de control de flujo:
  BRp #-4
  Si la condición es positiva, salta 4 ins atrás
  Ciclo de Intrucción para BRp
  Fetch | Decode | Execute | Memory

José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

Procesadores en orden

- 1ª gen – Instrucciones son ejecutadas en el orden secuencial del programa, una cada vez.

Ejemplo:

Ciclo	1	2	3	4	5	6
Instruccion1	Fetch	Decode	Execute	Memory		
Instruccion2					Fetch	Decode

Pipeline

- 2ª gen – Instrucciones son ejecutadas secuencialmente, en el orden del programa, pero con pipeline

Ejemplo:

Ciclo	1	2	3	4	5	6
Instruccion1	Fetch	Decode	Execute	Memory		
Instruccion2		Fetch	Decode	Execute	Memory	
Instruccion3			Fetch	Decode	Execute	Memory

José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

ILP: Instruction Level Parallelism

3ª gen – Instrucciones son ejecutadas en paralelo.

Ejemplo de código 1:

```
c = b + a;
d = c + e;
```

No paralelizable

Ejemplo de código 2:

```
a = b + c;
d = e + f;
```

Paralelizable

José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

ILP – (Cont)

Dos formas de ILP:

- –Superscalar: En tiempo de ejecución se ejecutan las fases: fetch, decode, and execute, de múltiples instrucciones al mismo tiempo. La ejecución puede ser fuera de orden.
- –VLIW: En tiempo de compilación, empaquetar múltiples instrucciones independientes en una gran instrucción y procesar esas instucciones como unidades atómicas.

Ciclo	1	2	3	4	5
Instruction1	Fetch	Decode	Execute	Memory	
Instruction2	Fetch	Decode	Execute	Memory	
Instruction3		Fetch	Decode	Execute	Memory
Instruction4		Fetch	Decode	Execute	Memory

José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

Multithreading y Multicore

4ª gen – Multi-threading: Múltiples hilos se ejecutan alternando o de manera simultanea en el mismo procesador o core.

5ª gen - Multi-Core: Múltiples hilos se ejecutan en multiples procesadores o cores.

Agenda

Motivación

Historia del paralelismo

GPUs como arquitecturas paralelas

Arquitectura básica de ordenadores

Algunos conceptos básicos de C

José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

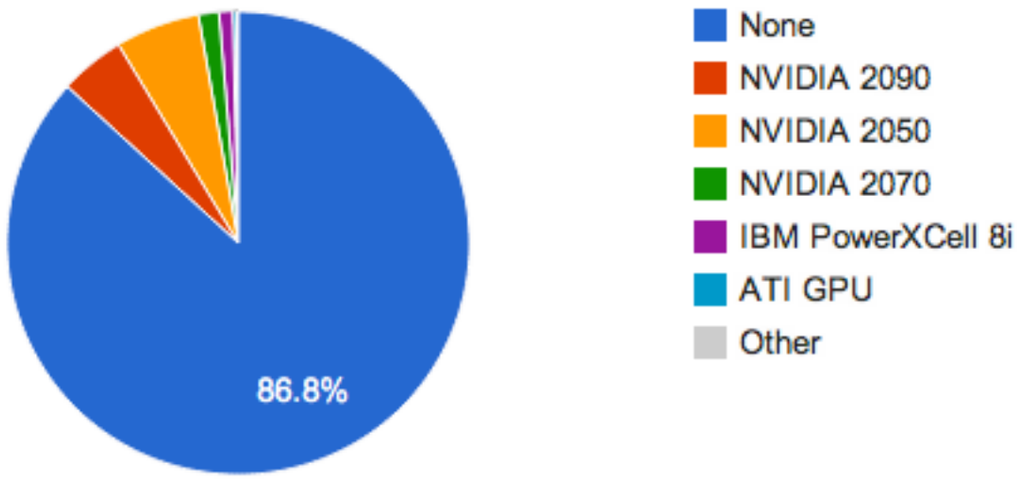
El estado actual de la ciencia desde una perspectiva jocosa…¡pero realista!

José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

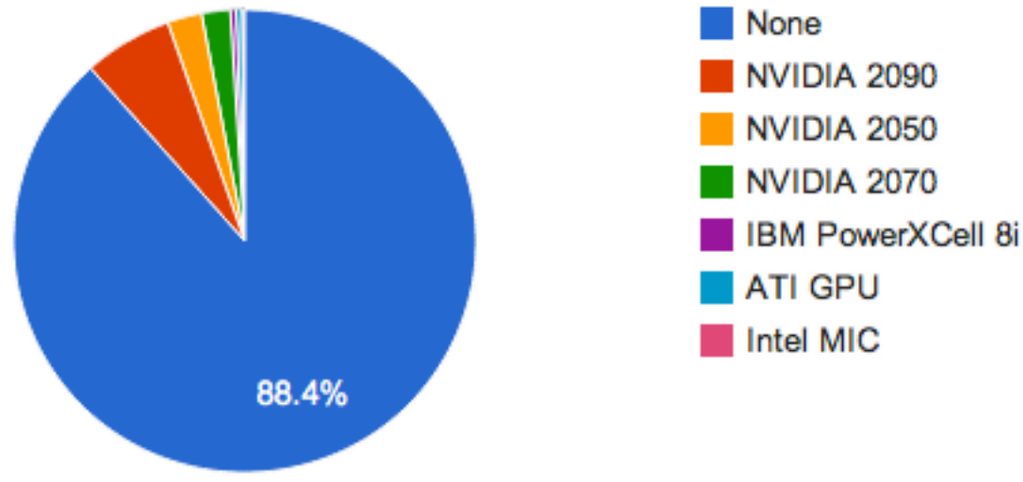
Aportación de las GPUs (o aceleradores gráficos) a la supercomputación

Accelerator/Co-Processor	Count	System Share (%)	Rmax (GFlops)	Rpeak (GFlops)	Cores
None	442	88.4	107103147.87	140616821.21	12004293
NVIDIA 2090	31	6.2	5626155.43	11318582.04	572312
NVIDIA 2050	12	2.4	7262060	14304324.8	532298
NVIDIA 2070	10	2	1775173.41	3265356.28	146174
IBM PowerXCell 8i	2	0.4	1168500	1537632	136800
ATI GPU	2	0.4	360710	647429.2	26268
Intel MIC	1	0.2	118600	180992	9800

Accelerator/Co-Processor Performance Share



Accelerator/Co-Processor System Share



José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

GPUs como arquitecturas paralelas

- CPUs: 2 cores, 4 cores muy complejos:
  - Implementación de todo el ISA x86.
  - Ejecución fuera de orden.
  - Hyperthreading.
- GPUs: hasta 512 cores muy simples:
  - Implementación reducida del ISA.
  - Ejecución en orden.
  - Carga de una sola instrucción por procesador.
  - UC y Cache instrucciones compartidas entre 8 cores

José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

GPUs como Manycores

- GPUs contiene un gran paralelismo de datos (SPMD):

  Múltiples hilos ejecutan el mismo programa sobre múltiples cores con diferentes datos SPMD (Single Program Multiple Data)

- Definición Relacionada:

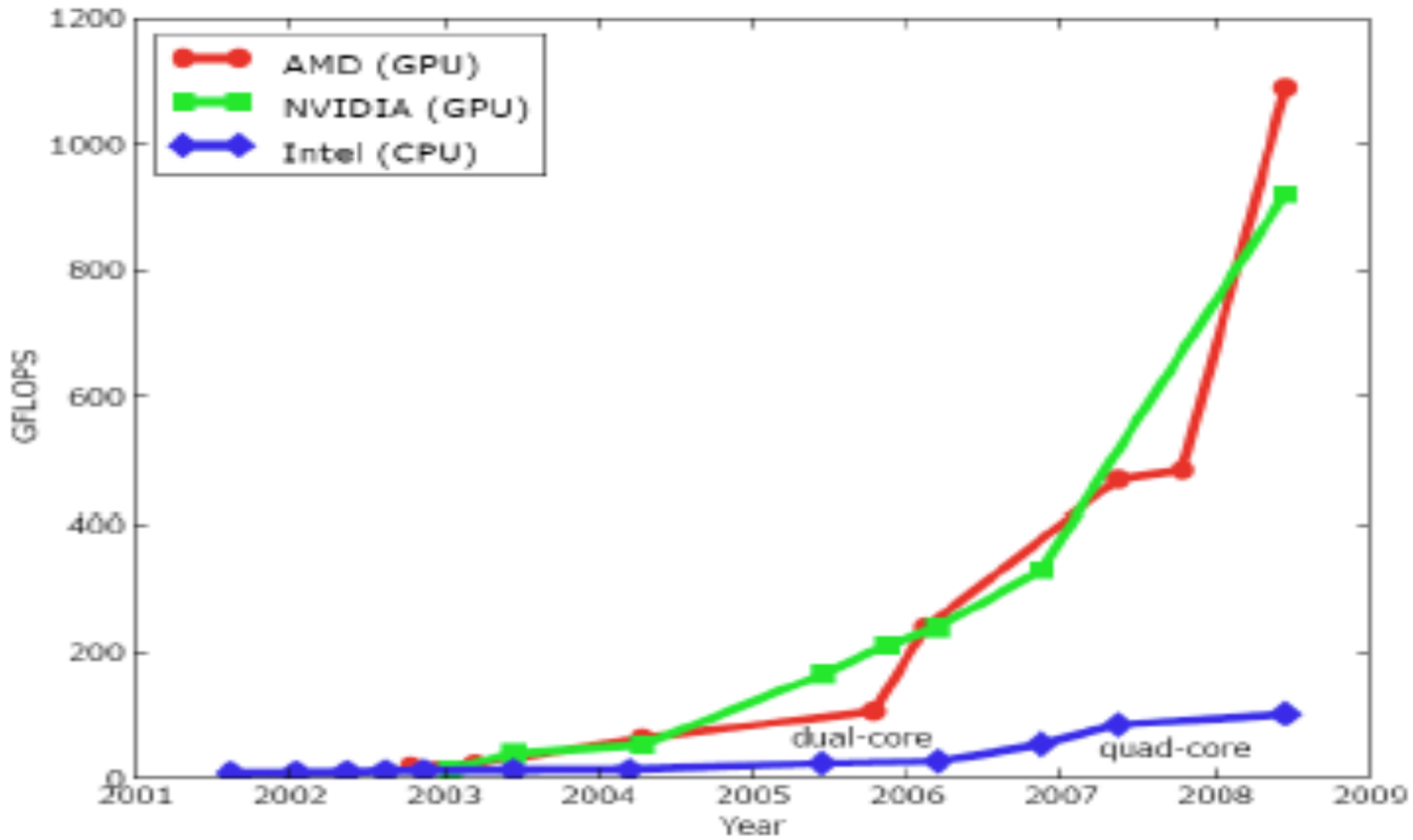
  SIMD: Single Instruction Multiple Data

  Todas las unidades funcionales ejecutan la misma instrucción pero sobre diferentes datos.

José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

Ventaja de Rendimiento de las GPUs

- Una gran ventaja en el rendimiento pico de las GPUs
  - Cálculos: 1 TFLOP Vs. 100 GFLOPS
  - Ancho de banda en memoria: 100-150 vs. 32-64 GB/s



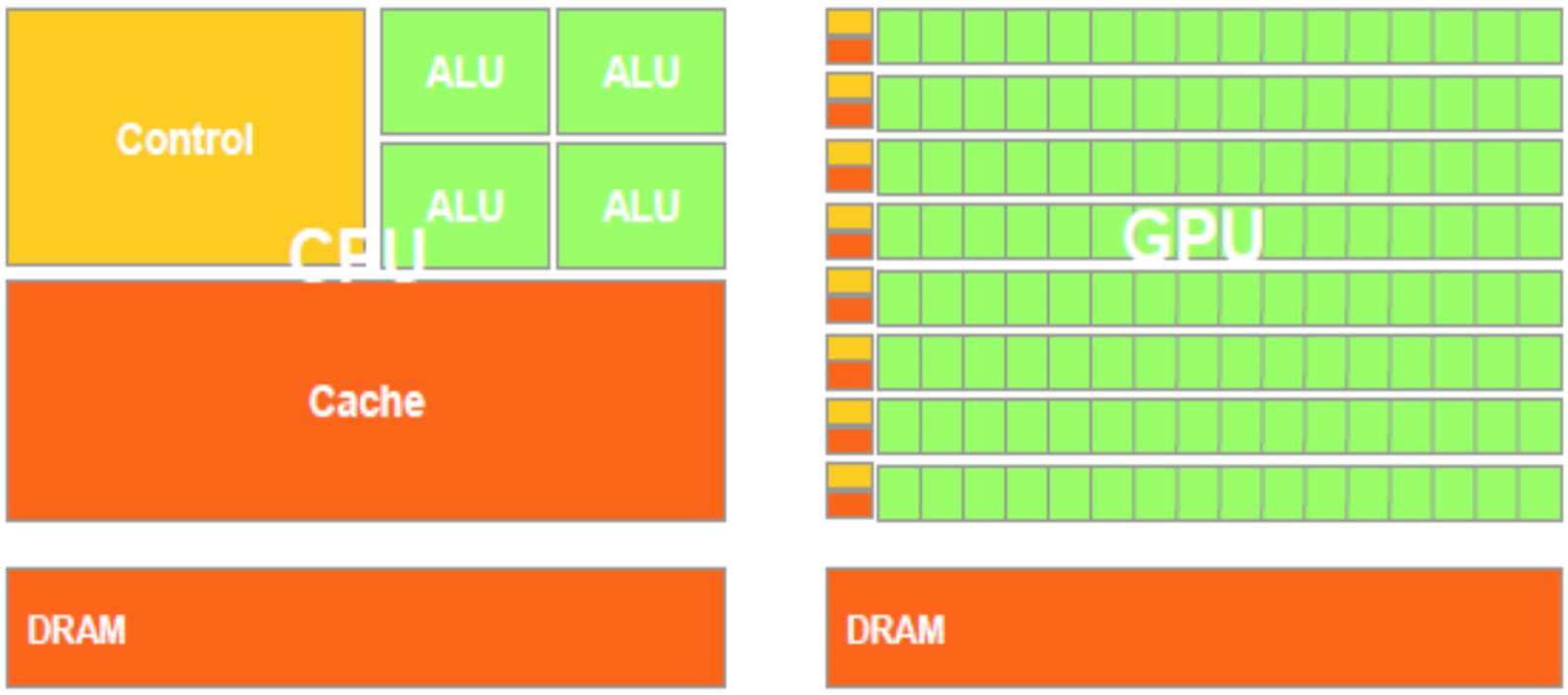
- Las GPUs están en todos los PCs y estaciones de trabajo.

José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

Motivos de la victoria de los FLOPS

- ¿ Por qué esa gran diferencia en rendimiento de cómputo?
- Diferencias en la filosofía de diseño:
  - CPU optimizado para código secuencial.
    - Unidad de control: Ejecución en paralelo, fuera de orden, etc → Pero resultado secuencial.
    - Más memoria cache.
  - GPU optimizado para videojuegos
    - Gran número de operaciones aritméticas
    - Ingente número de hilos. Unos esperan (latencias, etc) y otros ejecutan

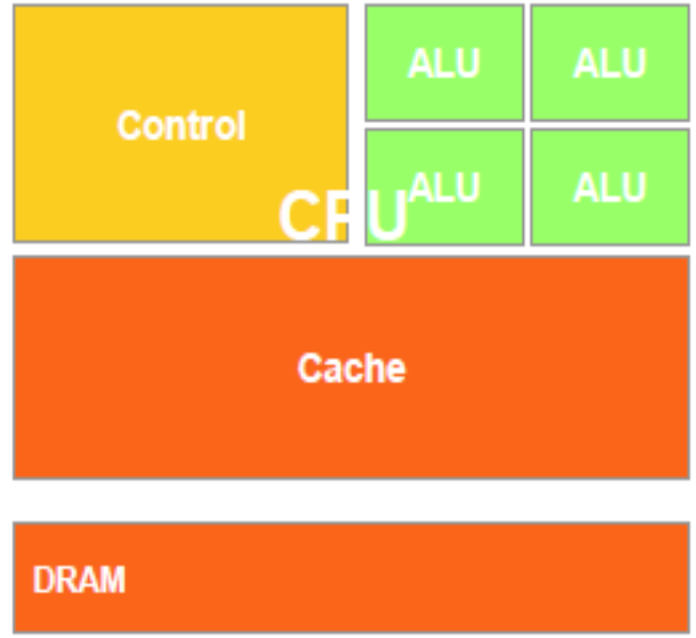
Distintas filosofías en el diseño (ver video)



José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

CPUs: Diseño orientado a latencia

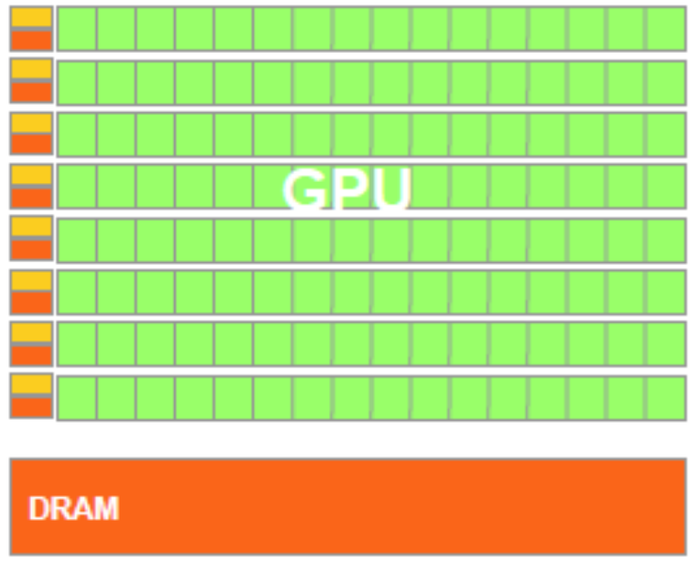
- Caches muy grandes
  - Convierten accesos muy lentos a memoria en accesos muy cortos
- Unidad de control sofisticada
  - Predicción de saltos para reducir la latencia.
  - Reenvío de datos para aliviar la latencia en los datos
- Pocas ALUs muy potentes
  - Reducción de la latencia de operaciones



José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

GPUs: Diseño orientado a throughput

- Caches muy pequeñas
  - Para impulsar el uso del gran ancho de banda a memoria
- Unidad de control muy simple
  - No hay predicción de saltos
  - No hay reenvío de datos
- ALUs eficientes energéticamente
  - Muchas ALUs, con largas latencias pero altamente pipelined para conseguir un gran throughput
- Requieren un número de hilos muy elevado



José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

Entonces…

- Conclusión: GPUs diseñadas para cálculos numéricos.
- GPUs no son CPUs. No son incompatibles.
- Computación híbrida CPU+GPU → CUDA.
  - CPU partes secuenciales.
  - GPU partes intensivas en cómputo.

José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

Las mejores aplicaciones usan las dos: CPU y GPU

- CPUs para las partes secuenciales donde la latencia importa
  - CPUs pueden ser 10+x mas rápido que las GPUs para códigos secuenciales
- GPUs para las partes paralelas donde gana el throughput
  - GPUs pueden ser 10+x mas rápido que las CPUs para códigos paralelos

Performance y más…

- No todo es rendimiento. Hay otros factores:
  - Mercado:
    - Cuanto mas se use, mas se extiende.
    - Problema anterior del mercado paralelo.
    - Esto ha cambiado con las GPUs. G80 200 millones.
  - Fácil accesibilidad:
    - ¿Aplicación médica en clusters?
    - Hospitales: PC + Aceleradores Hw.
  - Soporte IEEE standar floating point.
  - Problema de las GPUs. Doble Precisión

José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

Agenda

Motivación

Historia del paralelismo

GPUs como arquitecturas paralelas

Arquitectura básica de ordenadores

Algunos conceptos básicos de C

Poniendo las piezas del ordenador juntas

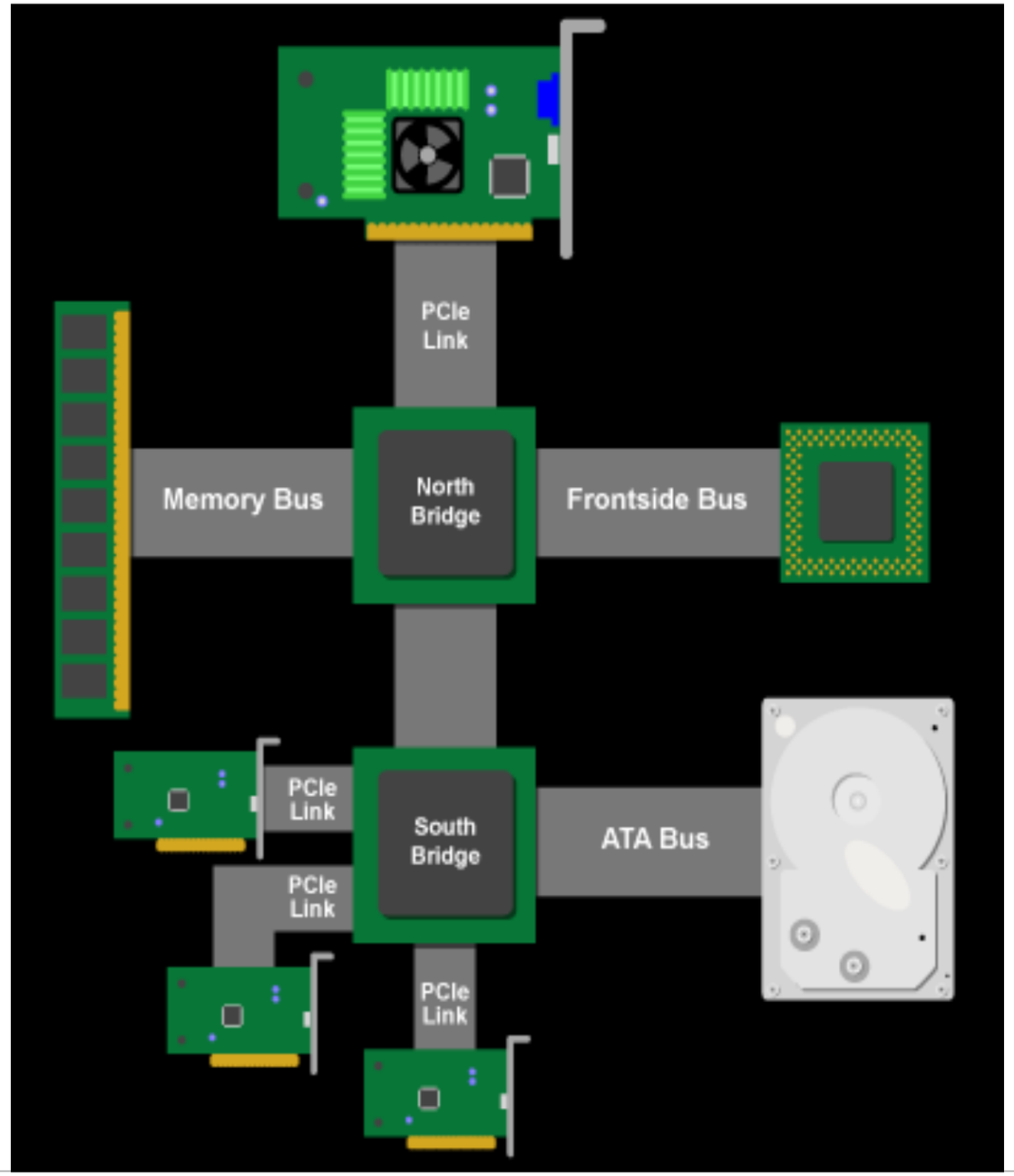
- La GPU es parte del ordenador.
- Vamos a ver como está compuesto el ordenador.

Arquitectura del PC

- Northbridge conecta 3 componentes que se deben comunicar a una alta velocidad
  CPU, DRAM, video

- Southbridge sirve como enlace de los dispositivos.

- PCIe une la GPU al sistema 8Gb/sec



José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

Agenda

Motivación

Historia del paralelismo

GPUs como arquitecturas paralelas

Arquitectura básica de ordenadores

Algunos conceptos básicos de C

José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

Floating point operation (FLOP)

- Float es un tipo primitivo de C/C++-
- Son usados para representar números racionales.
- Simple precisión se representa con 32 bits.
- Doble precisión se representa con 64 bits.

Números representables

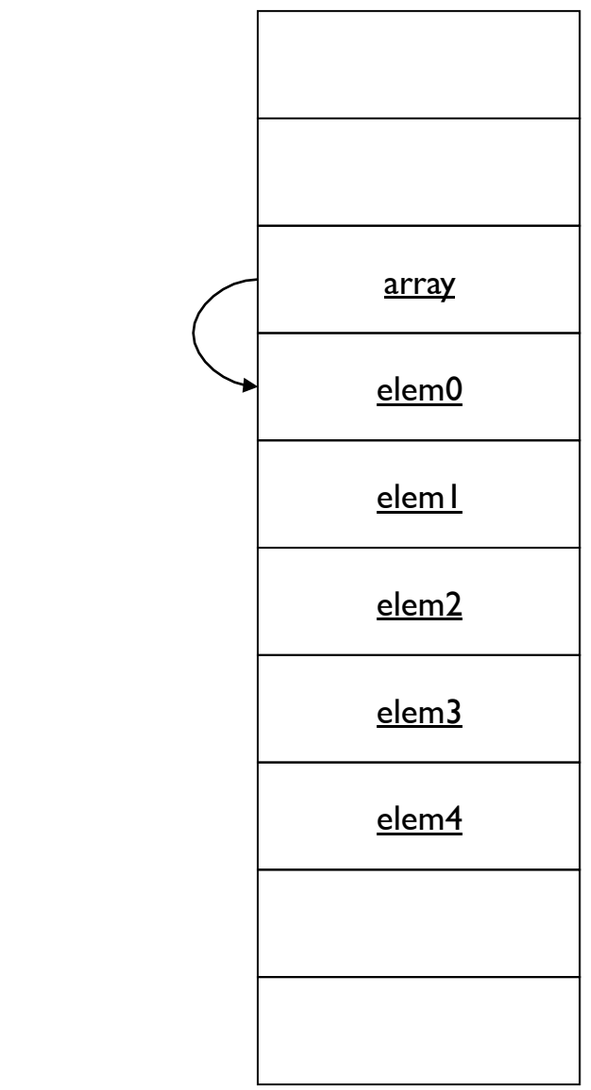
- Los números representables para un formato dado es el conjunto de todos los números que pueden ser representados exactamente en ese formato.
- Ver tabla para enteros con 3 bits

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

Arrays

- int array[5];
- La variable "array" es un puntero a un bloque de memoria que contiene 5 enteros.
- En otras palabras, el array es de tipo int *.
- "array" es en si mismo una variable que contiene la posición de inicio de ese bloque de memoria.



Arrays en 2D

Qué pasa con:

int matrix[3][3]; // ¿Qué es esto?

Respuesta: Un array de dos dimensiones.

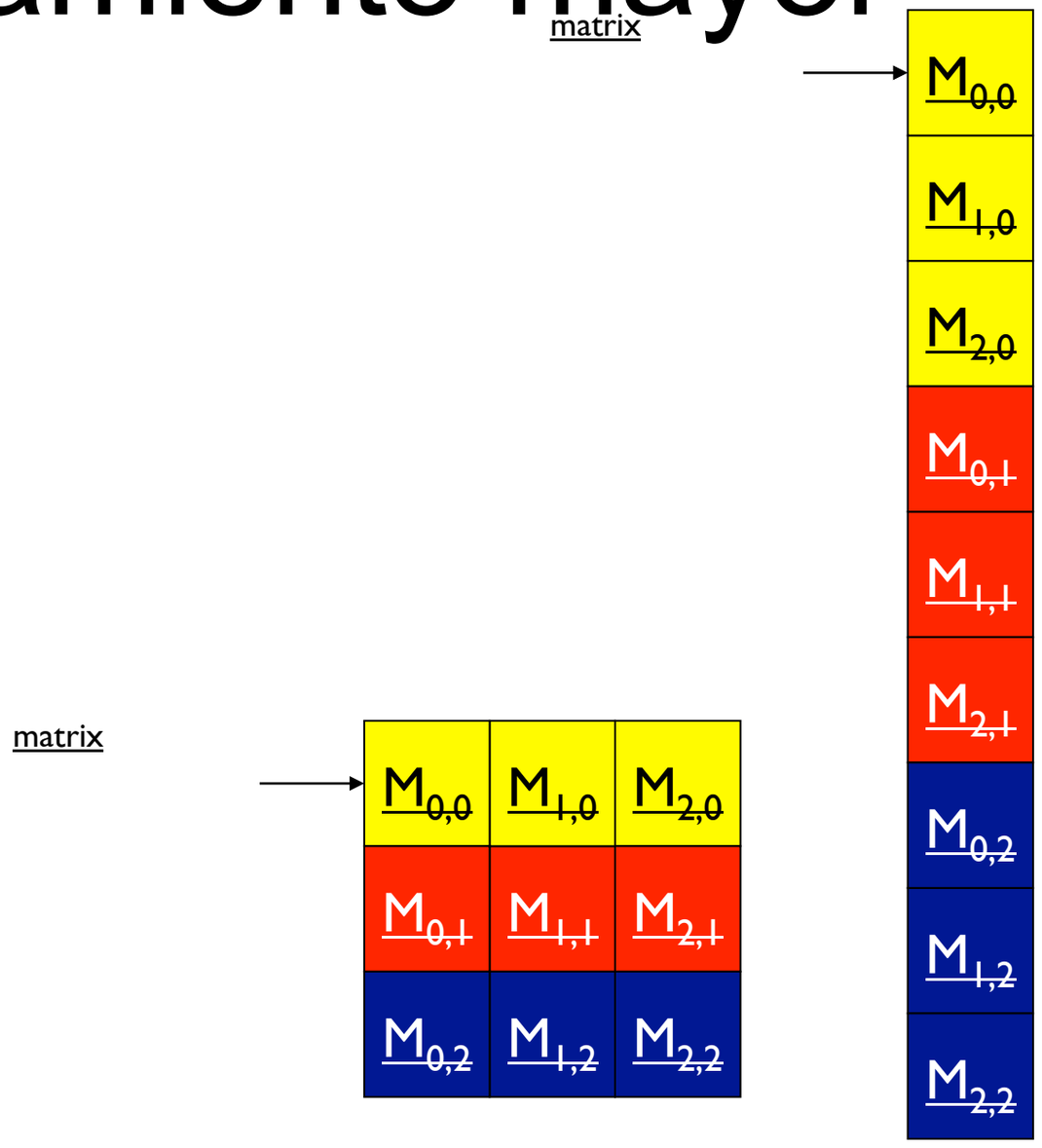
Pero la memoria es de 1 Dimensión. ¿Cómo se almacena?

Respuesta: Ordenamiento mayor.

José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

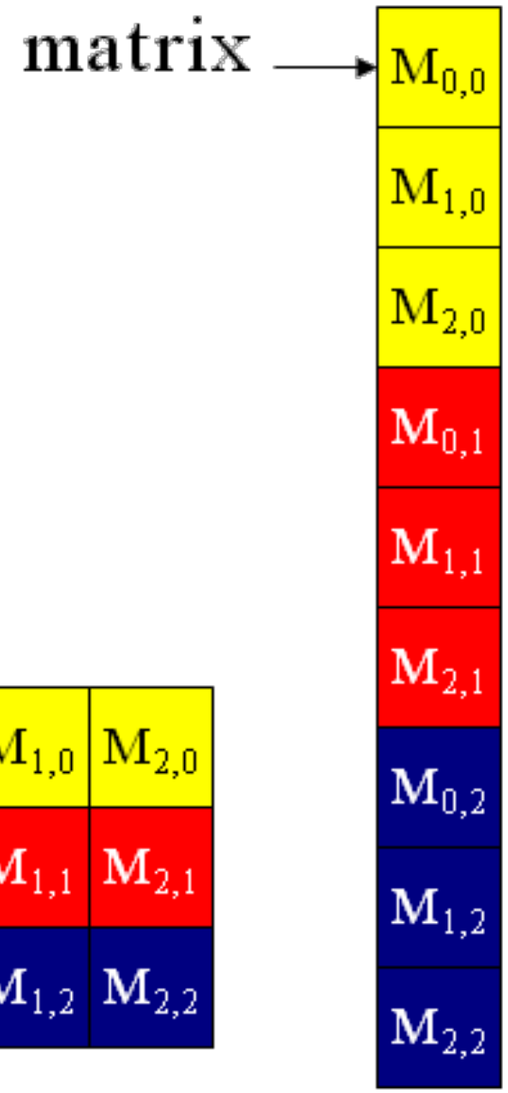
C usa ordenamiento mayor

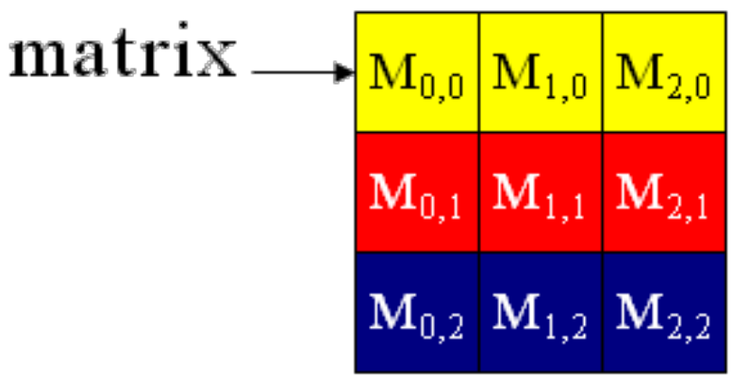
- Fila Mayor: Empieza desde la esquina superior izq, y almacena columnas consucutivas primero
- Nota: Fortran es a la mayor columna



Aritmética de punteros para acceso a memoria

- Fila Mayor: Empieza desde la esquina superior izq, y almacena columnas consucutivas primero
- Nota: Fortran es a la mayor columna





José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

Operaciones atómicas

thread 1: Reg ← Mem[x]	thread 2: Reg ← Mem[x]
Reg ← Reg + 1	Reg ← Reg + 1
Mem[x] ← Reg	Mem[x] ← Reg

Si x inicialmente es 0, ¿Qué tendría x después de que x e y terminen su ejecución?

La Respuesta es el problema de las condicones de carrera. Para evitarlas se necesitan operaciones atómicas.

José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

Operaciones atómicas

thread 1: Reg ← Mem[x]
Reg ← Reg + 1
Mem[x] ← Reg

thread 2: Reg ← Mem[x]
Reg ← Reg + 1
Mem[x] ← Reg

- Thread 1 lee 0 y el hilo 2 lee 1
- Mem[x] acaba siendo 2

José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu

¿Preguntas?

José M. Cecilia - Tlf: (+34) 968 27 85 87 jmcecilia@ucam.edu