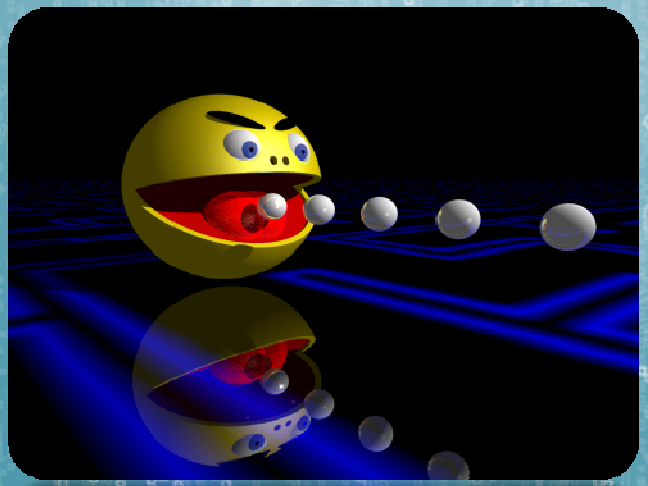
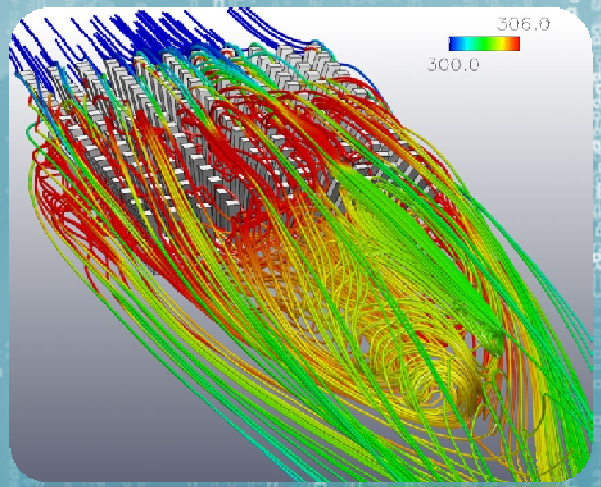
UNC Universidad Nacional de Córdoba

FaMAF

“De los Juegos, a la Ciencia”. El futuro de simulaciones y cálculos numéricos usando GPUs.





Axel D. Dente[1]– Ezequiel Ferrero[2]

[1]dente@famaf.unc.edu.ar – [2]ferrero@famaf.unc.edu.ar

NVIDIA

CONICET 50 años

Esquema

- Un poco de Historia
- Paralelismo en GPUs
- Programación en CUDA
- Aplicaciones
- Futuro de la Tecnología
- Ejemplo: Modelo de Potts de q estados

Un poco de Historia

Co-Procesadores

Destinados a realizar tareas especificas
para ayudar a los procesadores principales (CPUs)

Algunos ejemplos:

Co-procesador matemático en XTs, 286s, 386s y 486s

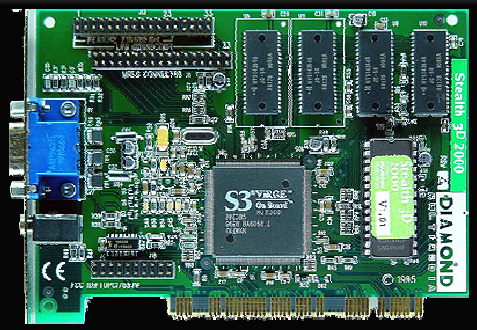
Placas de Video

Un poco de Historia

GPU -> Graphics Processing Unit

Exclusivamente para procesamiento de gráficos

Mediado de los 90° aparecen las primeras aceleradoras gráficas 3D destinadas a las PC



S3 Virtual Reality Graphics Engine (ViRGE)

fabricada en 0,35 micras - frecuencia de GPU 66 MHz

Hoy en dia se fabrican en 45 nm - frecuencia de 1400 MHz

Un poco de Historia

Tarea Principal: Renderización 3D
se usan Pixel Shaders y Vertex Shaders

Usadas principalmente para Videojuegos y Diseño Gráfico

Un poco de Historia

Ejemplo

Un poco de Historia

Desde el 2006
NVIDIA cambia la arquitectura

Figuras extraídas de: PC Perspective - NVIDIA GeForce 8800 GTX Review - DX10 and Unified Architecture

Un poco de Historia

2006 – Comienzan nuevos usos de las GPU

2.1 A Highly Multithreaded Coprocessor

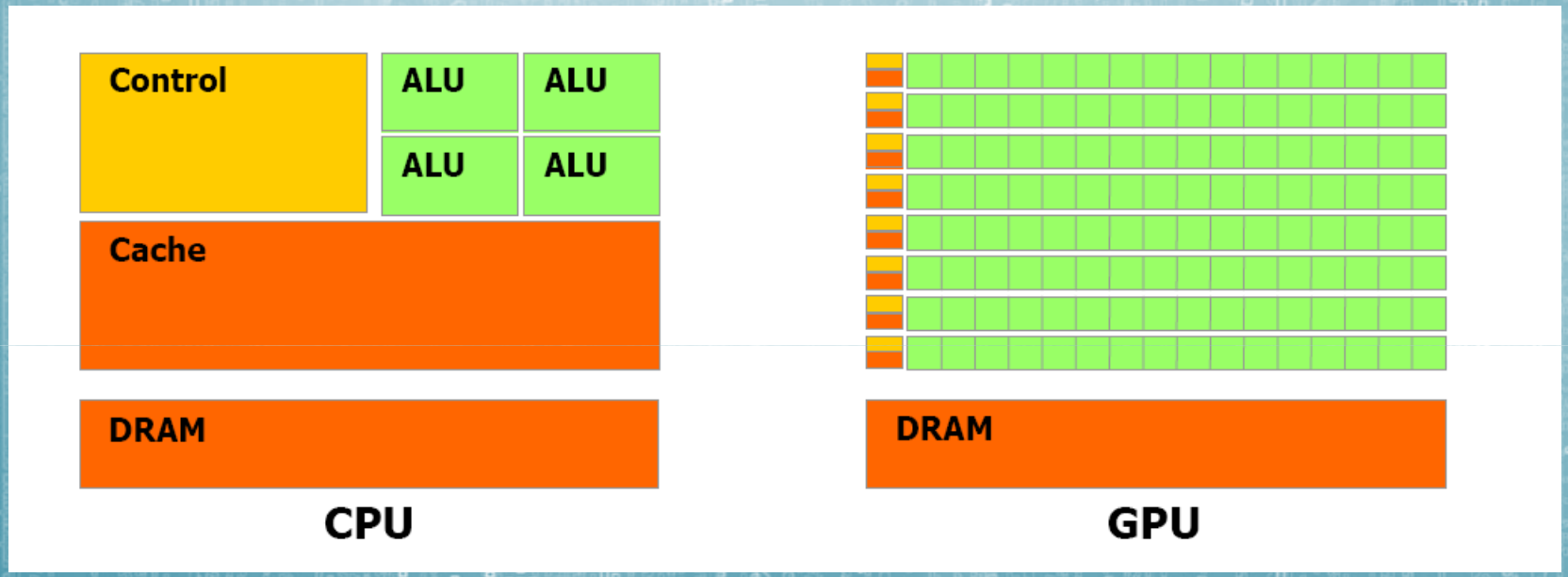
When programmed through CUDA, the GPU is viewed as a *compute device* capable of executing a very high number of threads in parallel. It operates as a coprocessor to the main CPU, or *host*: In other words, data-parallel, compute-intensive portions of applications running on the host are off-loaded onto the device.

"Cuando se programa con CUDA, la GPU es vista como un dispositivo capaz de ejecutar un gran numero de hilos en paralelo."

Texto extraído de: NVIDIA_CUDA_Programming_Guide_1.1.pdf

Paralelismo en GPUs

Diferencia contra las CPUs



Las GPUs destinan más transistores al PROCESAMIENTO de DATOS

Ventajas: Velocidad de transferencia de Memoria

Numero de procesadores (ej. 240 cores en GTX280)

Extraído de: NVIDIA_CUDA_Programming_Guide_1.1.pdf

Paralelismo en GPUs

Distribución de Procesos en CUDA (*Compute Unified Device Architecture*)

Nivel de abstracción: Hilos de Ejecución

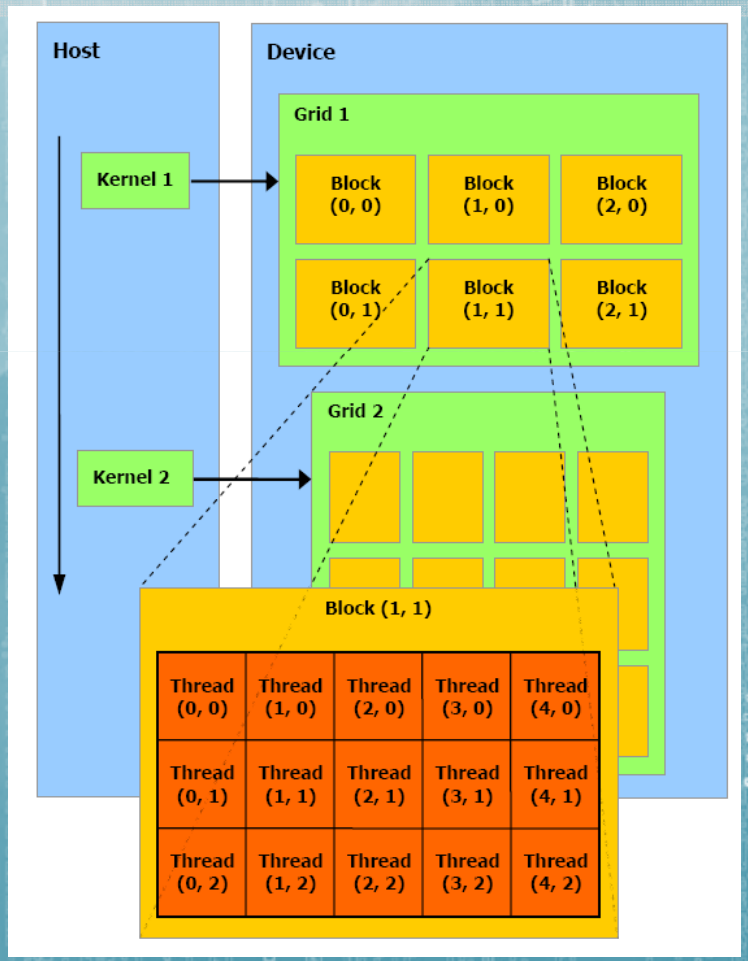
Se pueden lanzar muchos hilos independizándose del numero de cores que disponga la placa de Video

Hilos se agrupan en bloques de hasta 512 hilos

Pueden estar distribuidos en 1, 2 o 3 dimensiones.
Ej: un bloque de 32 filas y 16 columnas tiene 512 hilos

Bloques se agrupan en Grillas

Pueden estar distribuidos en 1 o 2 dimensiones
Este agrupamiento determina el orden de ejecución de los hilos



Extraído de: NVIDIA_CUDA_Programming_Guide_1.1.pdf

Programación en CUDA

Fortran

CUDA Fortran - Compilador PGIfortran (The Portland Group)

Lenguaje específico para Fortran

También se puede usar código de C incluido en fortran por medio de Librerías. Ej: FORTRAN bindings for CUDA [(1)]

Soportado por GFortran e Intel Fortran

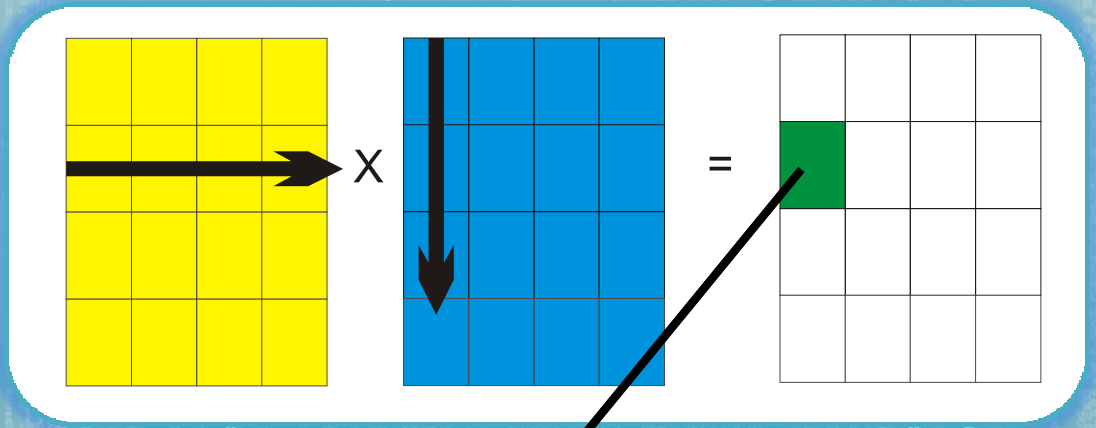
[(1)] http://www.hoopoe-cloud.com/Solutions/Fortran/Default.aspx

Programación en CUDA

Ejemplo: Multiplicación de Matrices

Código Serial

```
subroutine MaxMult(A,B,C,N)
integer N,i,j,k
real A(1:N,1:N),B(1:N,1:N),C(1:N,1:N)
do i=1,N
  do j=1,N
    C(i,j)=0.d0
    do k=1,N
      C(i,j)=C(i,j)+A(i,k)*B(k,i)
    enddo
  enddo
enddo
end subroutine MaxMult
```

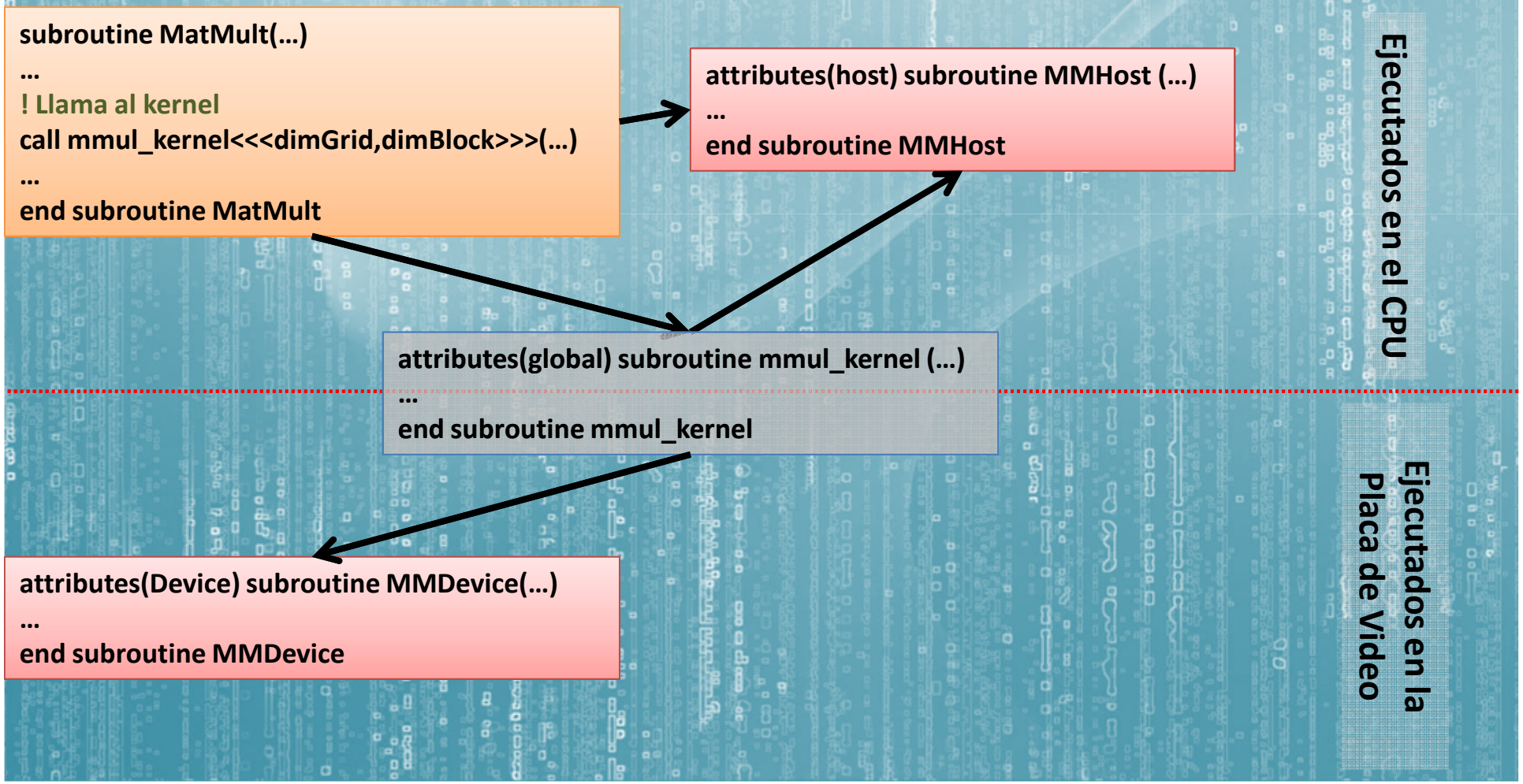


Cada punto va a ser calculado por un Hilo

Programación en CUDA

Multiplicación de Matrices versión CUDA

CUDA trabaja con Kernels



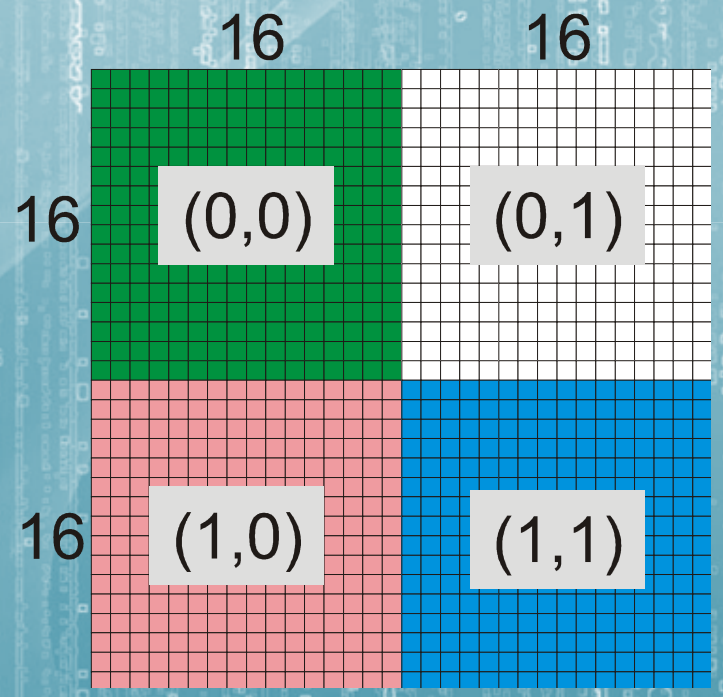
Programación en CUDA

Multiplicación de Matrices versión CUDA

```
subroutine MatMult(A,B,C,N)
integer N,i,j,k
!Matrices alojadas en el HOST
real A(N,N),B(N,N),C(N,N)
!Matrices alojadas en el DEVICE
real device, Adev(N,N),Bdev(N,N),Cdev(N,N)
! dim3 variables de la Grilla y los Bloques
type(dim3) :: dimGrid, dimBlock
!Copiamos de la memoria del Host al Device
Adev=A
Bdev=B
!Creamos las dimensiones de la Grilla y los Bloques
dimGrid = dim3( N/16, N/16, 1 )
dimBlock = dim3( 16, 16, 1 )

call mmul_kernel<<<dimGrid,dimBlock>>>( Adev, Bdev, Cdev,N)
C=Cdev
end subroutine MatMult
```

División de la grilla



Programación en CUDA

Multiplicación de Matrices versión CUDA

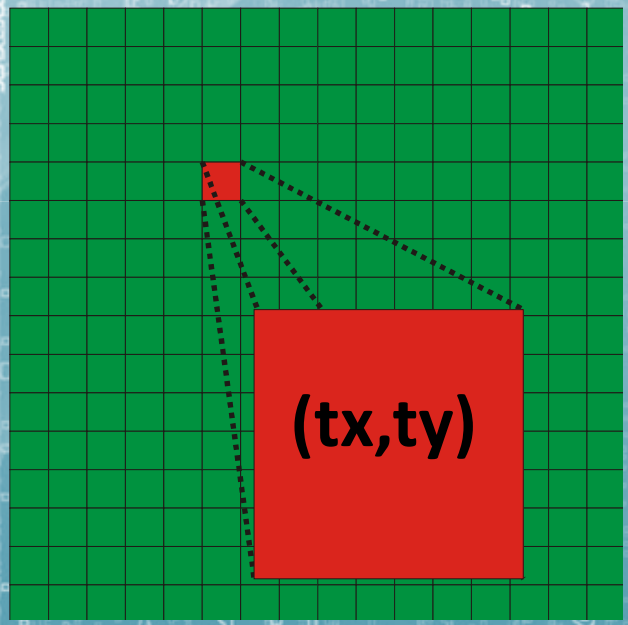
```
attributes(global) subroutine mmul_kernel( A, B, C, N)
real A(N,N),B(N,N),C(N,N)
integer N,i,j,k,tx,ty

! Obtenemos el indice del hilo
tx = threadidx%x
ty = threadidx%y
! Este hilo calcula C(i,j) = sum(A(i,:) * B(:,j))
i = (blockidx%x-1) * 16 + tx
j = (blockidx%y-1) * 16 + ty
C(i,j) = 0.d0
do k = 1,N
                C(i,j) = C(i,j) + A(tx,k) * B(k,ty)
enddo
end subroutine mmnul_kernel
```

threadidx%x => Primer índice del hilo dentro del Bloque (el segundo y tercer índice se llaman con %y y %z)

blockidx%x => Primer índice del Bloque dentro de la grilla

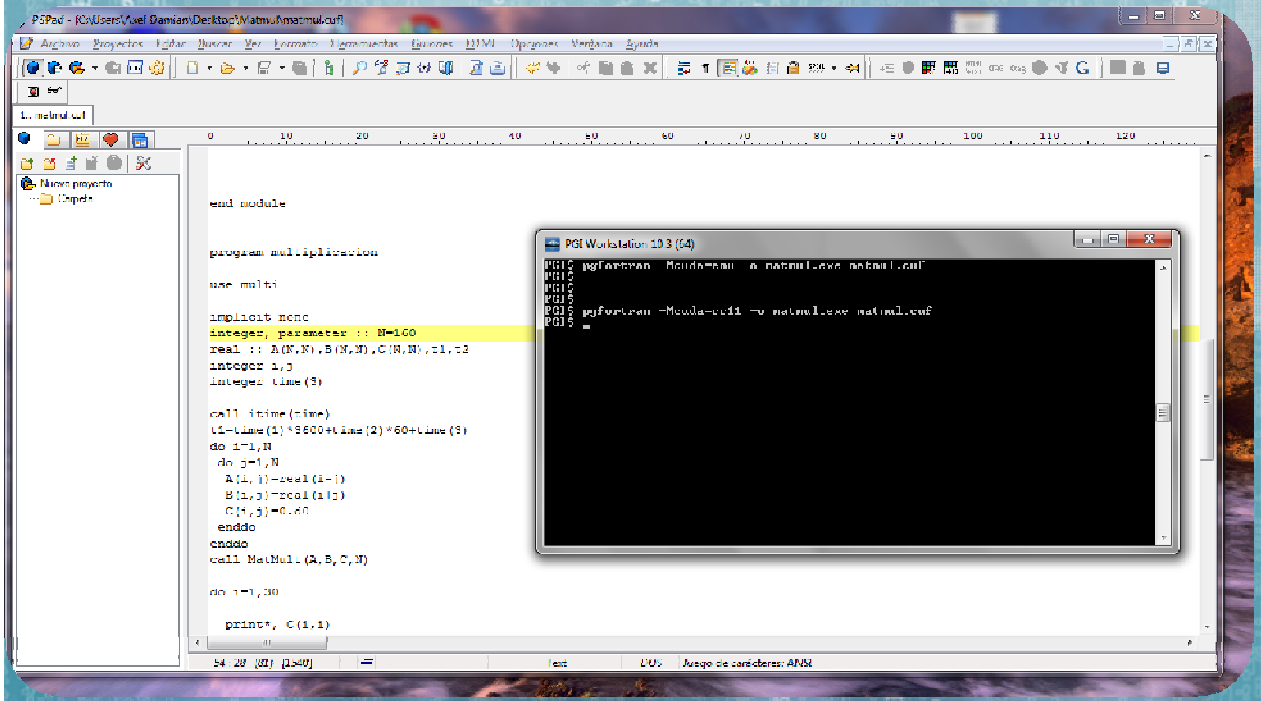
División del bloque



Programación en CUDA

Compilación

pgfortran –Mcuda=cc11 –o Matmul.out matmul.cuf



Habilita las Características de CUDA

cc=Compute Capability

10=1.0

11=1.1

13=1.3 GTX 200

20=2.0 GTX 400

emu=simulado en la CPU

Programación en CUDA

Optimizaciones

Varios Niveles de Memoria

Device Memory

Shared Memory

Texture Cache

Constant Cache

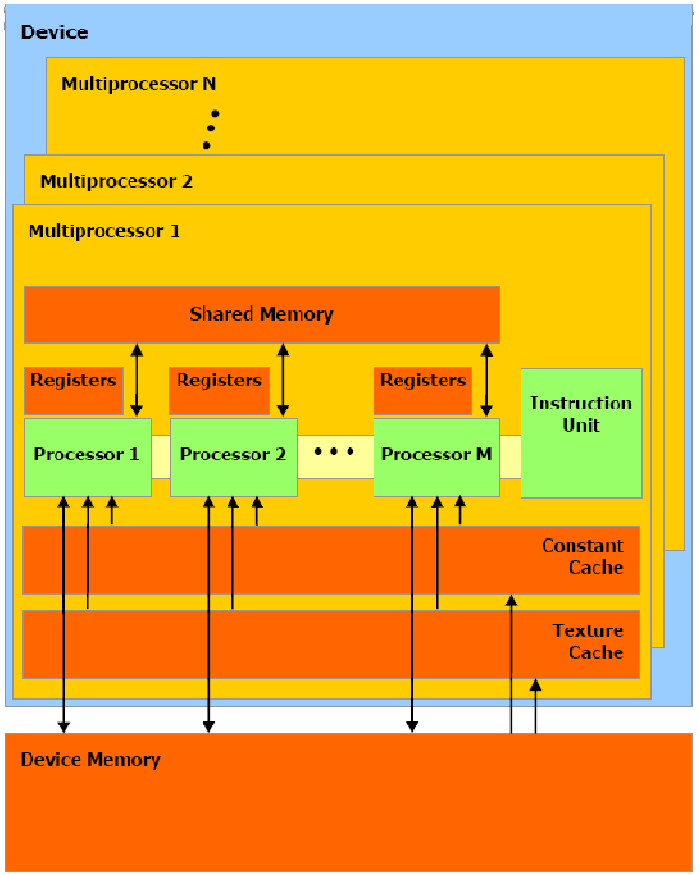
Register

Velocidad de acceso

Ejemplo en Fortran

```
real, shared :: A   !Solo se puede definir en un kernel tipo Device

real, constant :: A
```



A set of SIMD multiprocessors with on-chip shared memory.

Aplicaciones

Ejemplos

High Performance Finite Difference PDE Solvers on GPUs	38x
Accelerated Ray Tracing for Radiotherapy Dose Calculations on a GPU	6x
Massively parallel forward modeling of scalar and tensor gravimetric data	40x
Incompressible Flow Computations on the NCSA Lincoln Tesla Cluster	…
Radar Signal Processing with Graphics Processors (GPUs)	140x
Singular value decomposition on GPU using CUDA	…
Two-electron Integral Evaluation	130x

www.nvidia.com/object/cuda_home.html

Aplicaciones

Problemas Adecuados

Operaciones con matrices	→	Multiplicaciones, auto-valores, etc..
Problemas con grillas	→	Modelos de Ising, difusión del calor
FFTs	→	MRI, Imágenes Medicas
		Librerías disponibles
		MATLAB plug-in for CUDA
Tratamiento de imágenes	→	Renderización
Programas trivialmente paralelizables	→	Multiples realizaciones del mismo problema con distintas condiciones iniciales

Aplicaciones

Desventajas

Poca memoria por Hilo comparado con un Cluster

Transferencia de memoria lenta
entre el Host y el Device

Aplicaciones complicadas

Hilos con mucha inter-dependencia

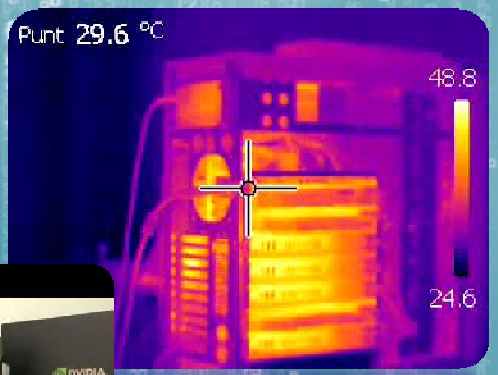
Problema de N-cuerpos (con grandes dimensiones)

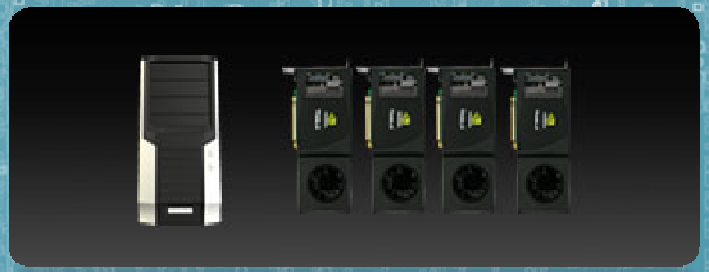
Futuro de la tecnología

Presente

Empresa	"Lenguaje"
Nvidia	CUDA (C, Fortran, etc)
Ati	OpenCL (C)

University of Antwerp stuffs 13 GPUs into FASTRA II supercomputer





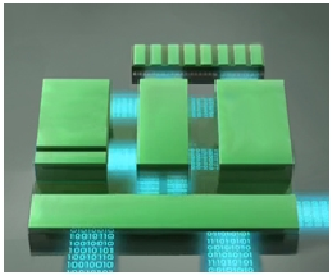
Futuro de la tecnología

Lo que se viene

AMD Fusion™ Family of APUs (Accelerated Processing Unit)

delivering powerful CPU and GPU capabilities

Combinar CPU y GPU en un solo micro central

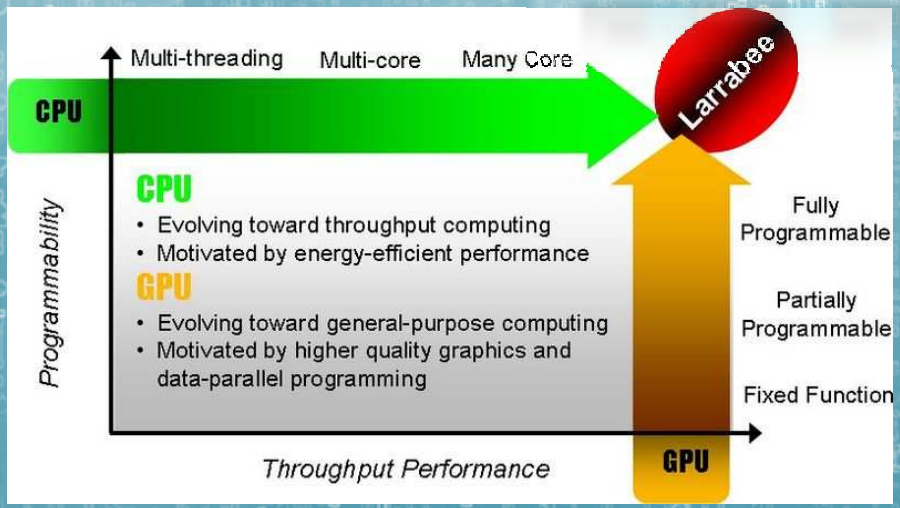


Intel (Larrabee)

↓

Mezcla entre CPU y GPU

27 de Mayo de 2010



http://sites.amd.com/us/Documents/48423B_fusion_whitepaper_WEB.pdf

El modelo Potts ferromagnético de *q* estados

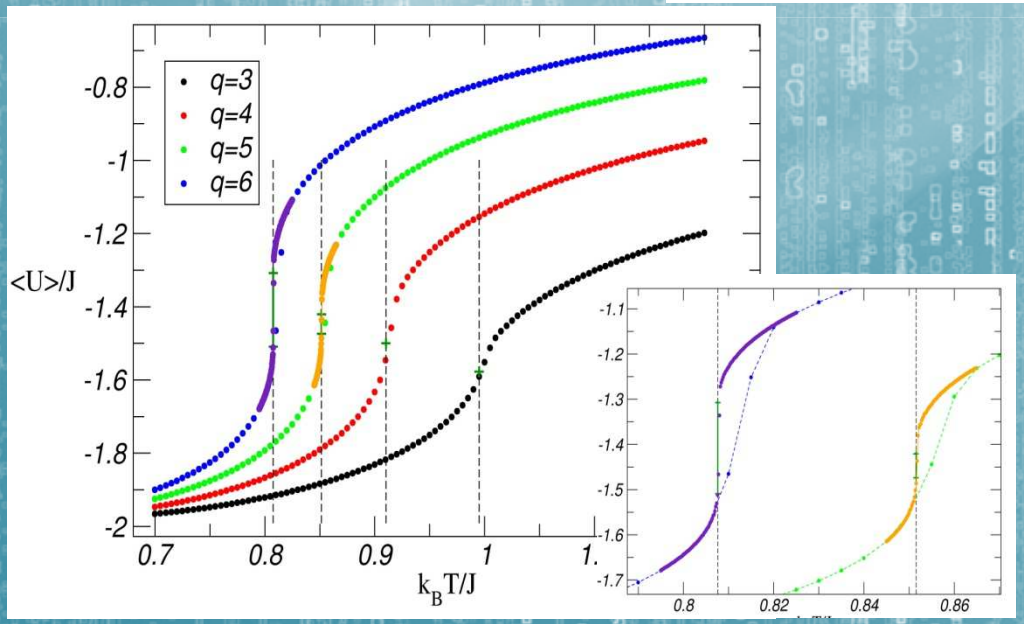
$$H = -J \sum_{nn} \delta(s_i, s_j) \qquad J > 0 \qquad s_i = 1, \ldots, q$$

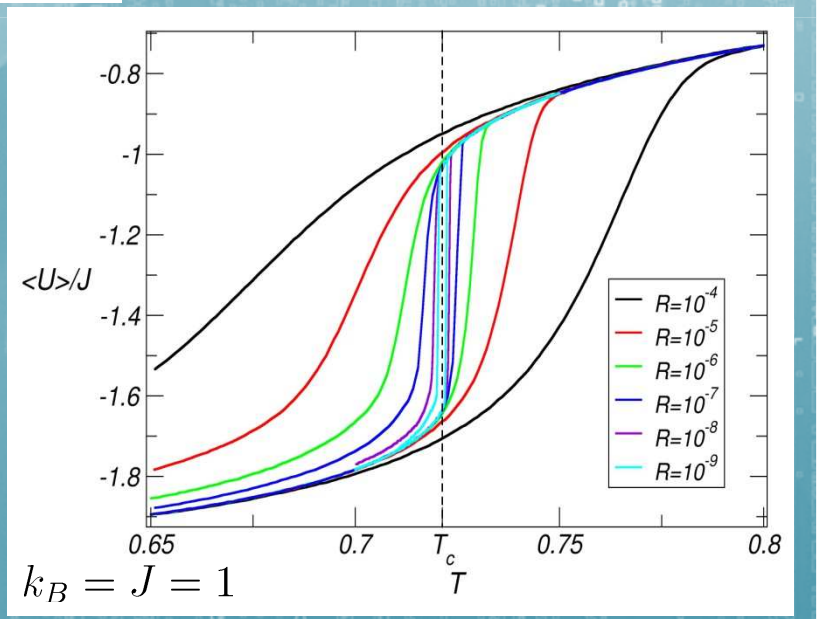
$$\delta(s, s') \begin{cases} 1 & \text{if } s = s' \\ 0 & \text{otherwise} \end{cases}$$

Transición de fase orden-desorden

En 2D: de 2° orden para q = 2,3,4
de 1° orden para q ≥ 5

$$\frac{k_B T_c}{J} = [ln(1+\sqrt{q})]^{-1}$$





Sin dinámica propia → Monte Carlo

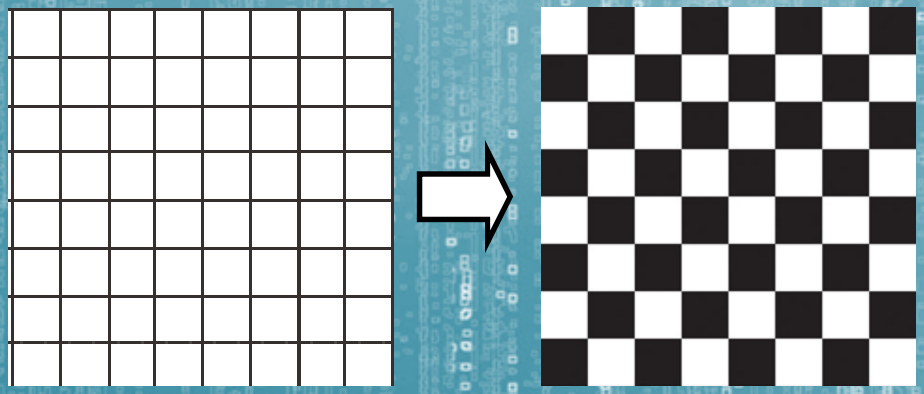
Algoritmo

Sistema de LxL=N spins c.c. periódicas → toro

-elijo un spin: S_i (*llamadas a RNG), copio su estado
-calculo la energía: *E_before*
-"flipeo" S_i (llamada a RNG)
-calculo la energía: *E_after*
-si (*DeltaE=E_after-E_before* ≤ *0*) acepto el cambio
-caso contrario sorteo un número aleatorio **r** (llamada a RNG)
y sólo si $\boldsymbol{r} \leq exp(-DeltaE/k_BT)$ acepto el cambio

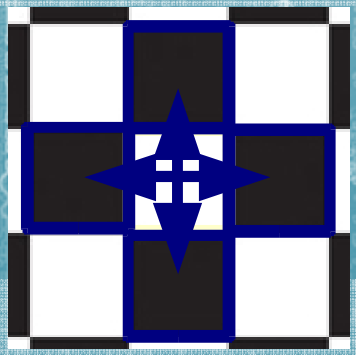
N repeticiones = 1MCS (un paso de Monte Caro, unidad de tiempo)

Estrategia de paralelización: Tablero de ajedrez



subsistemas independientes: black, white

Interacción a primeros Vecinos



Características del código

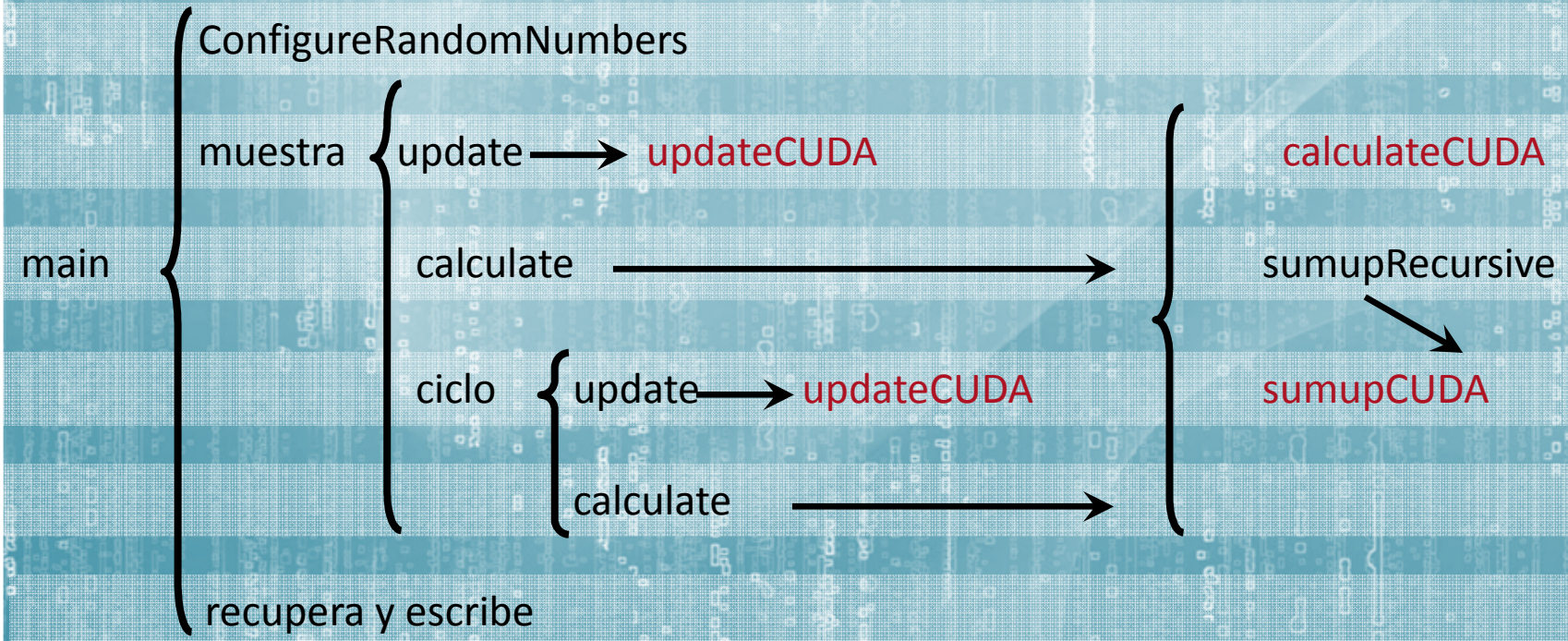
- **spin <--> thread** (asociamos cada spin a un hilo de ejecución diferente)
- **RNG independientes** (uno para cada thread)
  (tenemos **512x512/2:** cantidad disponible de pares buenos de semillas)
- **"frames" de 512x512**
  (subdividimos el sistema LxL en frames que mandamos al device secuencialmente)
- **cálculo local en paralelo + sumas recursivas en paralelo**
  (para el cálculo de cantidades físicas: energía, magnetización)

Ejecución

- El programa corre en el host y llegado el momento hace el llamdo a un kernel que se ejecuta en el device.
- Una función kernel se identifica por poseer las marcas "<<<" ">>>", conteniendo a la estructura de grillas y bloques de threads elegida para su ejecución.
- El transpaso de memoria entre el host y el device se hace explícitamente, con una directiva CUDA, sólo cuando es necesario. Ésta operación es lenta.

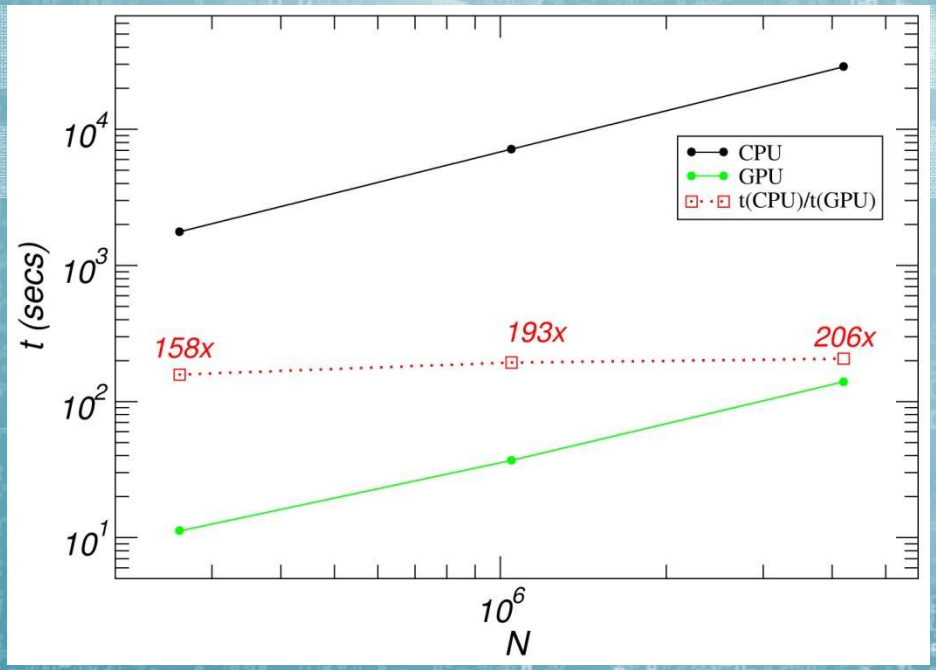
Estructura del código

pottsciclos.cu: calcula ciclos de histéresis



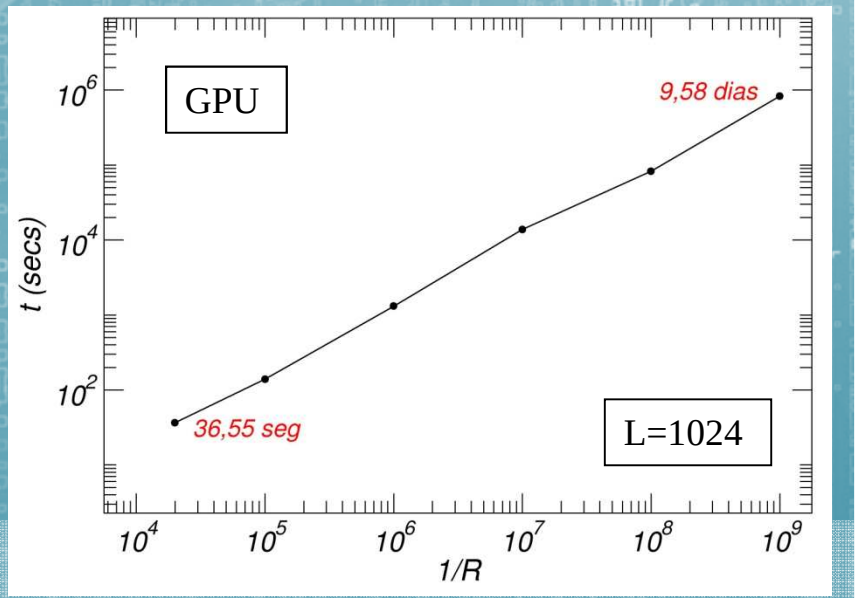
Veámoslo

Performance



Si, 200x, ¡200 veces más rápido! :D

Preis et al. 2009: Ising en GPUs (~60x)



ejemplo de cosas que ahora se pueden hacer

Algunos números (búsquedas web)

"CUDA" en arXiv: 41 resultados (desde 2007)

"GPU" en arXiv: 112 resultados (desde 2003)

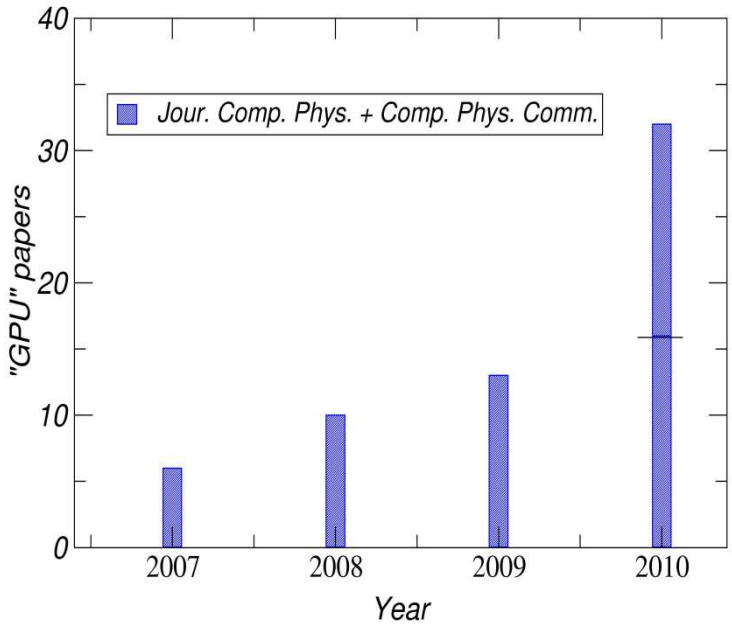
"science aplicattions" en nVidia CUDA Zone: 426 resultados (desde 2005)

(cuda+gpu) en Google Scholar : 4490 resultados

http://gpgpu.org/ "papers": 228 entradas
"research": 335 entradas

"GPU" en *Journal of Computational Physics*:
19 artículos

"GPU" en *Computer Physics Communications*:
29 artículos



en FaMAF

GPGPU Computing Group
http://www.cs.famaf.unc.edu.ar/~nicolasw/GPGPU/

Oscar Reula, Sergio Cannas, Nicolas Wolovick

NVIDIA *Professor Partner Funding Grants* (desde Septiembre 2008)

Proyectos:

-Solving evolutionary Partial Differential Equations in MHD and geometry
(D. Alonso, N. Wolovick, O. Reula)
-Post-Newtonian equations integration on GPUs
(M. Bellone, M. Tiglio)
-Monte Carlo simulations for the 2D q-state Potts model
(J.P. de Francesco, E.E. Ferrero, N. Wolovick, S.A. Cannas)
- Trotter-Suzuki simulation for a QD Tigt Binding model
(C. Bederian, A. Dente)

Llamado a una (1) beca para I+D en el área de HPC-GPGPU. **Deadline 14/06/2010**

FIN

Muchas gracias por su atención