Aplicaciones bioinformáticas en GPUs: Imágenes de alta resolución y análisis genómico

Manuel Ujaldón Martínez
Nvidia CUDA Fellow
Departamento de Arquitectura de Computadores
Universidad de Málaga

NLHPC National Laboratory for High Performance Computing Chile

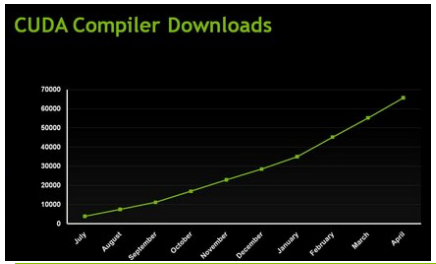
CMM Center for Mathematical Modeling

Indice de contenidos [70 diapositivas]

1. Introducción

En apenas 5 años, la programación CUDA está muy extendida y plenamente consolidada

- Se publican más de 500 artículos científicos cada año.
- Más de 500 universidades incluyen CUDA en sus cursos.
- Más de 350 millones de GPUs se programan con CUDA.
- Más de 120.000 programadores CUDA en activo.
- Más de un millón de descargas del compilador y toolkit.



Muchos de los supercomputadores más potentes se construyen con GPUs

- 18 de Junio de 2012:

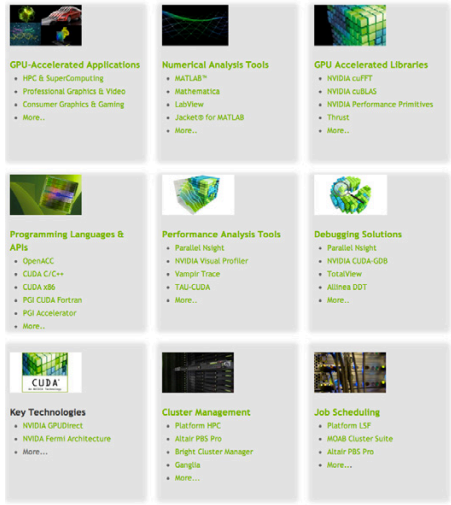
Rank	Site	Computer/Year Vendor	Cores	R_{max}	R_{peak}	Power
1	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom / 2011 IBM	1572864	16324.75	20132.66	7890.0
2	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect / 2011 Fujitsu	705024	10510.00	11280.38	12659.9
3	DOE/SC/Argonne National Laboratory United States	**Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom / 2012 IBM	786432	8162.38	10066.33	3945.0
4	Leibniz Rechenzentrum Germany	**SuperMUC** - iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR / 2012 IBM	147456	2897.00	3185.05	3422.7
5	National Supercomputing Center in Tianjin China	**Tianhe-1A** - NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050 / 2010 NUDT	186368	2566.00	4701.00	4040.0
6	DOE/SC/Oak Ridge National Laboratory United States	**Jaguar** - Cray XK6, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA 2090 / 2009 Cray Inc.	298592	1941.00	2627.61	5142.0
7	CINECA Italy	**Fermi** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom / 2012 IBM	163840	1725.49	2097.15	821.9
8	Forschungszentrum Juelich (FZJ) Germany	**JuQUEEN** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom / 2012 IBM	131072	1380.39	1677.72	657.5
9	CEA/TGCC-GENCI France	**Curie thin nodes** - Bullx B510, Xeon E5-2680 8C 2.700GHz, Infiniband QDR / 2012 Bull	77184	1359.00	1667.17	2251.0
10	National Supercomputing Centre in Shenzhen (NSCS) China	**Nebulae** - Dawning TC3600 Blade System, Xeon X5650 6C 2.66GHz, Infiniband QDR, NVIDIA 2050 / 2010 Dawning	120640	1271.00	2984.30	2580.0

- 9 de Noviembre de 2011:

Rank	Site	Computer/Year Vendor	Cores	R_{max}	R_{peak}	Power
1	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect / 2011 Fujitsu	705024	10510.00	11280.38	12659.9
2	National Supercomputing Center in Tianjin China	NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050 / 2010 NUDT	186368	2566.00	4701.00	4040.0
3	DOE/SC/Oak Ridge National Laboratory United States	Cray XT5-HE Opteron 6-core 2.6 GHz / 2009 Cray Inc.	224162	1759.00	2331.00	6950.0
4	National Supercomputing Centre in Shenzhen (NSCS) China	Dawning TC3600 Blade System, Xeon X5650 6C 2.66GHz, Infiniband QDR, NVIDIA 2050 / 2010 Dawning	120640	1271.00	2984.30	2580.0
5	GSIC Center, Tokyo Institute of Technology Japan	HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU, Linux/Windows / 2010 NEC/HP	73278	1192.00	2287.63	1398.6
6	DOE/NNSA/LANL/SNL United States	Cray XE6, Opteron 6136 8C 2.40GHz, Custom / 2011 Cray Inc.	142272	1110.00	1365.81	3980.0
7	NASA/Ames Research Center/NAS United States	SGI Altix ICE 8200EX/8400EX, Xeon HT QC 3.0/Xeon 5570/5670 2.93 Ghz, Infiniband / 2011 SGI	111104	1088.00	1315.33	4102.0
8	DOE/SC/LBNL/NERSC United States	Cray XE6, Opteron 6172 12C 2.10GHz, Custom / 2010 Cray Inc.	153408	1054.00	1288.63	2910.0
9	Commissariat a l'Energie Atomique (CEA) France	Bull bullx super-node S6010/S6030 / 2010 Bull	138368	1050.00	1254.55	4590.0
10	DOE/NNSA/LANL United States	BladeCenter QS22/LS21 Cluster, PowerXCell 8 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband / 2009 IBM	122400	1042.00	1375.78	2345.0

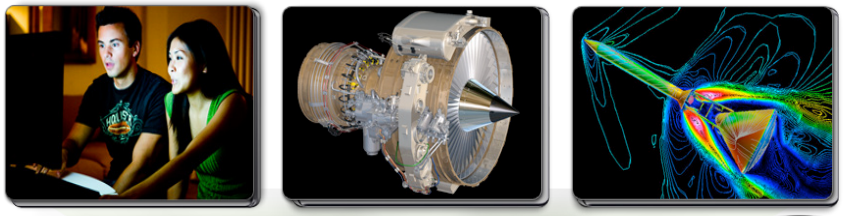
Y las herramientas de programación están también a la altura de estas exigencias

- Librerías.
- Lenguajes y APIs.
- Debuggers.
- Profilers.
- Gestión de tareas y recursos.



Existe una plataforma hardware para cada perfil de usuario...

Existen tres perfiles de usuario en Nvidia dentro de la base de la pirámide anterior



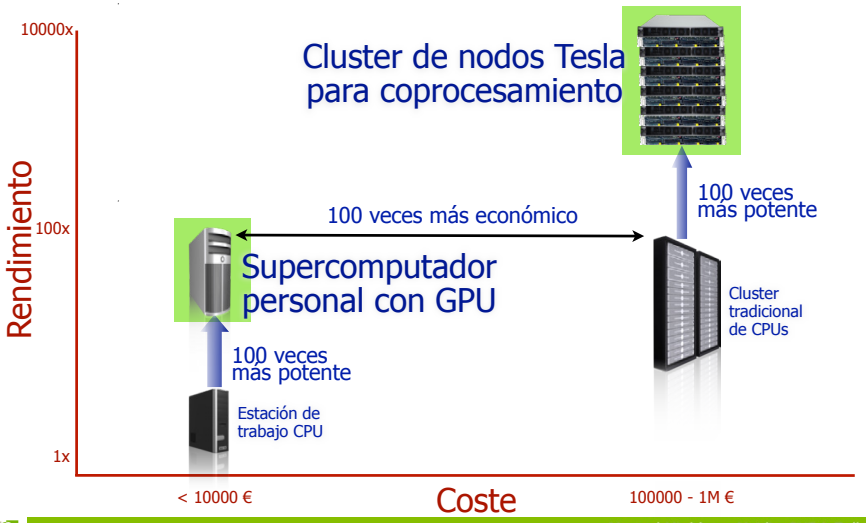
GeForce®
Jugones: Ocio y entretenimiento

Quadro®
Gráficos profesionales: Diseño y creación

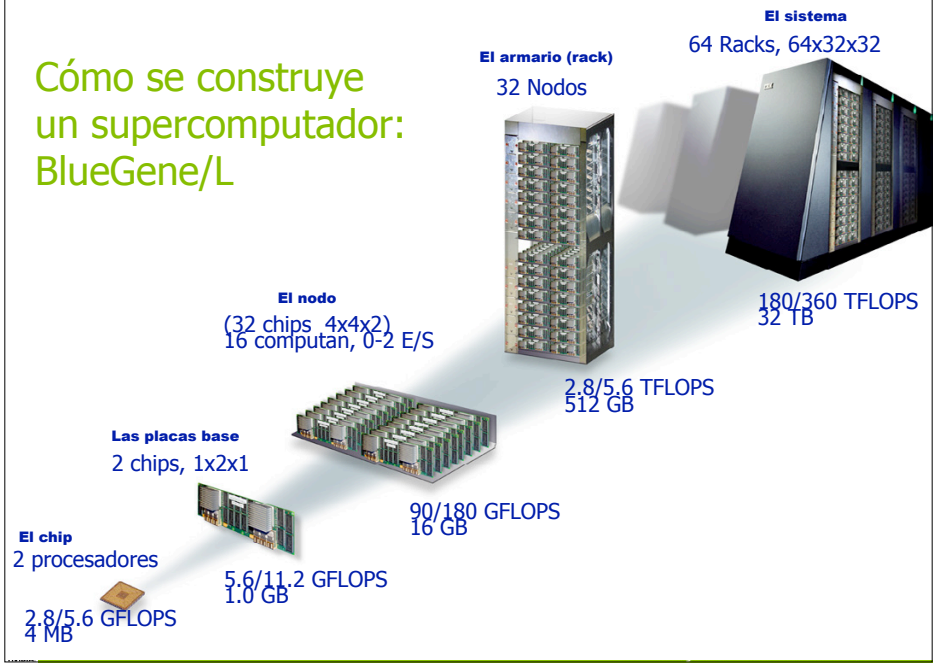
Tesla
Computación de altas prestaciones

Todos ellos basados en una misma microarquitectura

La relación rendimiento/coste es muy favorable ...



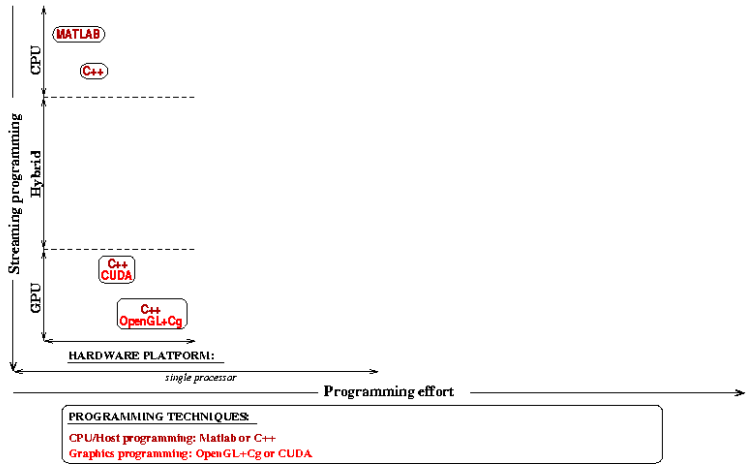
Cómo se construye un supercomputador: BlueGene/L



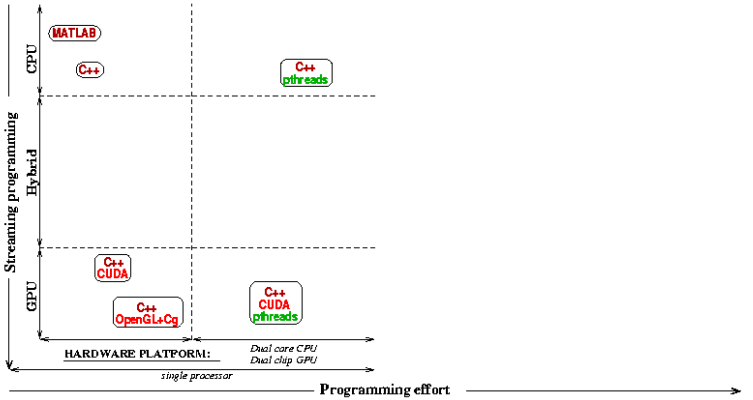
Lenguajes y herramientas útiles para el programador y su evolución

- En el contexto de la programación gráfica:
  - 2001: HLSL (programación de shaders - Microsoft).
  - 2003: OpenGL y Cg (C vectorial para shaders - Nvidia).
  - 2006: CUDA (GPGPU para plataformas Nvidia).
  - 2009: OpenCL (GPGPU para todo tipo de plataformas gráficas).
  - 2012: OpenACC (para programadores menos avezados).
- En el contexto de la programación paralela de CPUs:
  - P-threads para las plataformas multi-core.
  - MPI/OpenMP para las plataformas multi-socket.
- Veamos cómo se relaciona todo esto basándonos en la programación de nuestras aplicaciones.

Migración a Cg del código en CPU

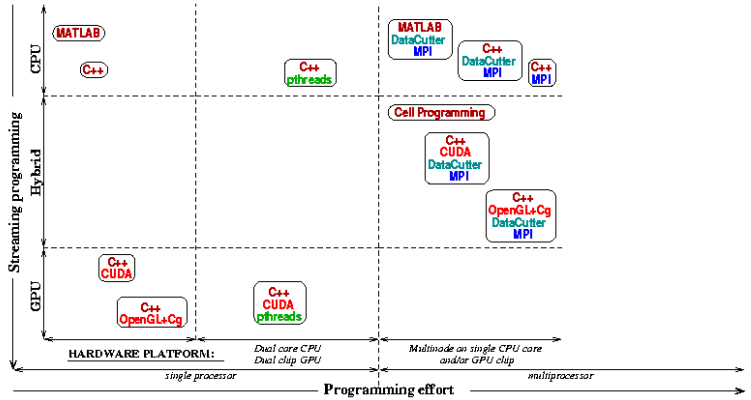


Versiones multicore en CPU y GPU



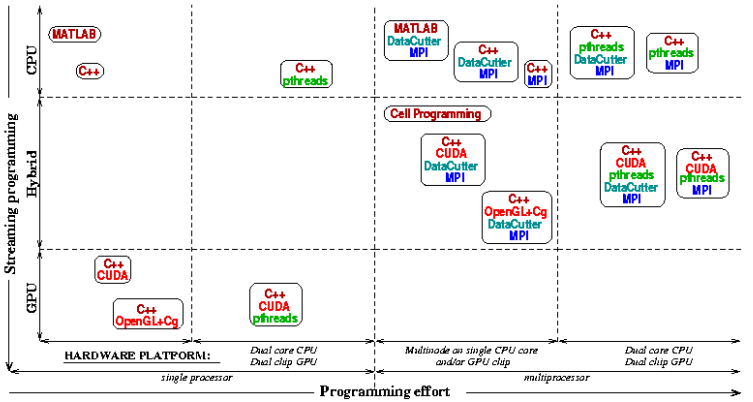
PROGRAMMING TECHNIQUES:
CPU/Host programming: Matlab or C++
Graphics programming: OpenGL+Cg or CUDA
Multicore programming: pthreads or OpenMP

Versiones multinodo



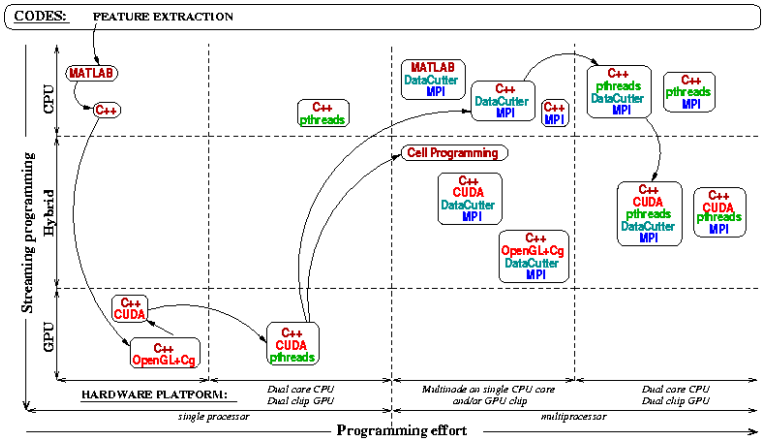
PROGRAMMING TECHNIQUES:
CPU/Host programming: Matlab or C++
Graphics programming: OpenGL+Cg or CUDA
Multicore programming: pthreads or OpenMP
Data distribution/partitioning: Manual or using DataCutter
Parallelism & communication: Message Passing or MPI

Versiones multicore sobre múltiples nodos



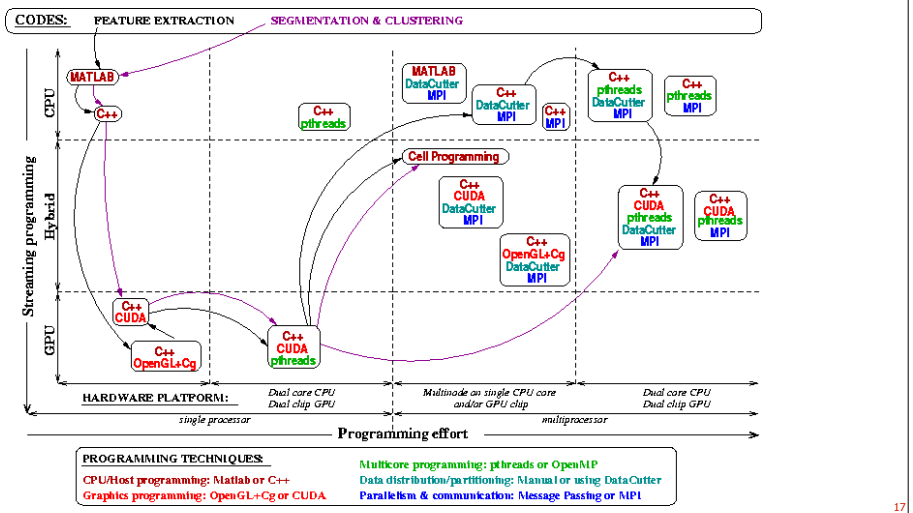
PROGRAMMING TECHNIQUES:
CPU/Host programming: Matlab or C++
Graphics programming: OpenGL+Cg or CUDA
Multicore programming: pthreads or OpenMP
Data distribution/partitioning: Manual or using DataCutter
Parallelism & communication: Message Passing or MPI

Ruta seguida por nuestra primera aplicación: Extracción de rasgos de una imagen

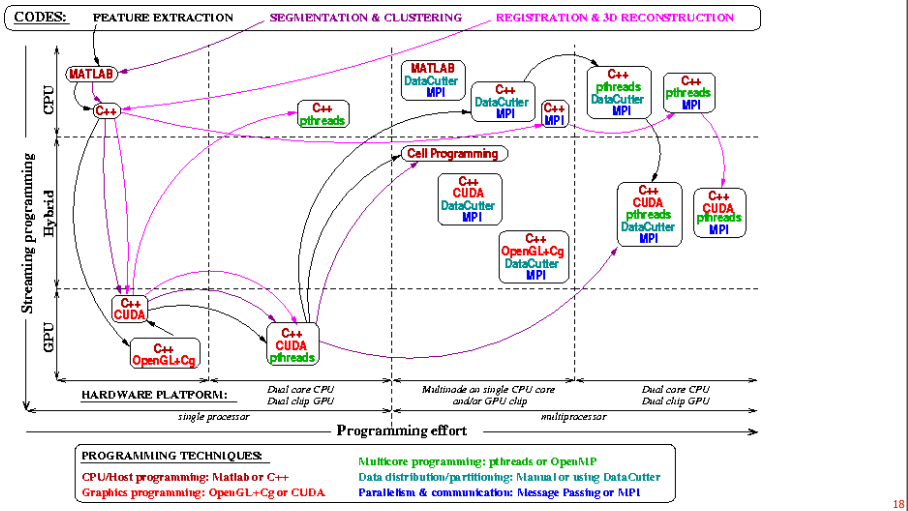


PROGRAMMING TECHNIQUES:
CPU/Host programming: Matlab or C++
Graphics programming: OpenGL+Cg or CUDA
Multicore programming: pthreads or OpenMP
Data distribution/partitioning: Manual or using DataCutter
Parallelism & communication: Message Passing or MPI

Ruta seguida por nuestra segunda aplicación: Segmentación y clustering de imágenes



Ruta seguida por nuestra tercera aplicación: Registro y reconstrucción 3D



2. Procesamiento de imágenes en GPU con CUDA

2.1. Caracterización de la imagen: Extracción de rasgos

2.1.1. Kernels desarrollados

Operadores que vamos a implementar en CUDA para caracterizar imágenes

- Operadores streaming: Conversiones de formato para el tratamiento del color. Opera sobre cada píxel de forma exclusiva.
  - Casos estudio: 8 conversiones entre diferentes formatos.
- Operadores stencil: Aplicados a un entorno de vecindad de cada uno de los píxeles. Dependencias de datos.
  - Caso estudio: LBP (Local Binary Pattern).
- Operadores con indirecciones (en el acceso a vectores):
  - Caso estudio: Matrices de co-ocurrencia (histogramas 2D).
- Operadores recursivos:
  - Caso estudio: Momentos de Zernike (métodos iterativos y directos).

Conversiones de formato para el color

- Son operadores "streaming", los más afines a la GPU.
- Utilizados para representar con mayor fidelidad el color en el procesamiento de imágenes. Por ejemplo:
  - Color por adición (luz en monitores): RGB, XYZ.
  - Color por sustracción (tinta en papel): CMYK.
  - Gestión del color: LUV, LA*B*, HSV.
  - Almacenamiento digital y fotografía: sRGB, JPEG, PNG.
  - Representación de texturas para renderización (bitmaps).
- Se requiere una conversión si el formato de origen y el más útil para la aplicación no coinciden. Para nuestro estudio, elegimos los formatos origen RGB y sRGB, y los formatos destino XYZ, Luv, LA*B* y HSV (8 operadores de conversión).

Conversiones de formato de color: Un ejemplo

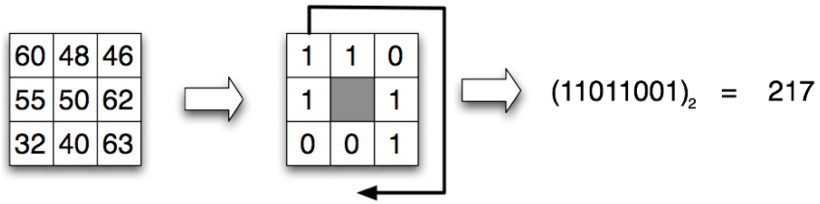
- Conversión de RGB (formato de entrada) a XYZ (formato de salida). Cada componente de la terna XYZ se calcula como una combinación lineal de los tres elementos RGB.

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0.4124 & 0.3575 & 0.1804 \\ 0.2126 & 0.7151 & 0.0721 \\ 0.0193 & 0.1191 & 0.9502 \end{pmatrix} * \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

- El nuevo resultado XYZ para cada píxel depende únicamente de los valores RGB de ese mismo píxel.
- Los cálculos son independientes y podemos aplicar paralelismo, aunque la fórmula no tiene mucha intensidad aritmética.

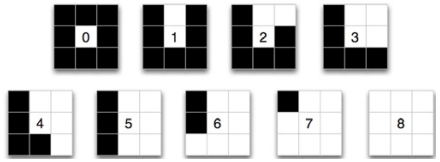
Operadores en el entorno de vecindad: LBP (Local Binary Pattern) (1)

- Basado en patrones binarios locales (Ojala, 2002)
  - Recoge la proporción de micro-rasgos tales como aristas, brillo y puntos oscuros.
  - Resulta muy utilizado en aplicaciones como el reconocimiento de expresiones faciales.



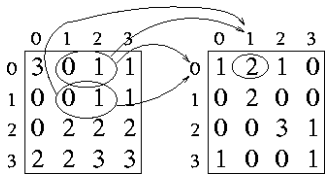
Operadores en el entorno de vecindad: LBP (Local Binary Pattern) (2)

- Es invariante a la rotación de la imagen y a variaciones de intensidad local o global.
- Para ello, cada valor obtenido se caracteriza dentro de una de las nueve clases siguientes:



Matrices de co-ocurrencia

- Ideadas por Haralick (1973).
  - Histograma de intensidades de pares de píxeles que guardan una determinada relación espacial [dx,dy].
  - Recoge la variación espacial de intensidades.
  - Se usa como estructura de datos intermedia para calcular ciertos rasgos: Contraste, correlación, energía, …
  - Ejemplo para una pequeña imagen con cuatro niveles de intensidad:



Variantes para el cálculo de las matrices de co-ocurrencia

- Dependiendo de la aplicación que las utilice:
  - Se calculan para cada píxel o subimagen a tratar.
  - Tamaño de ventana a procesar centrada en cada píxel o subimagen: Desde pequeña (4x4) a grande (256x256).
  - Espacio de color discretizado o completo [0..255].
  - Calculado para cada canal de color de forma separada o sobre escala de grises de forma conjunta.
- Configuración experimental:
  - Resolución de las imágenes de entrada.
  - Tratamiento hardware: CPU vs GPU.
  - Estructuras de datos: Matrices densas/dispersas.

Momentos de Legendre y Zernike

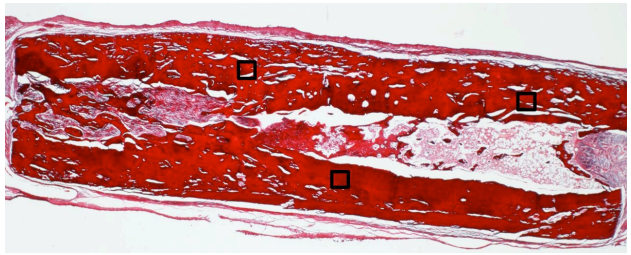
- Son filtros definidos en el dominio espacial como una forma directa de capturar las propiedades de una textura.
- Los polinomios de Legendre y Zernike representan una imagen mediante un conjunto de descriptores mutuamente independientes.
- Los polinomios de Zernike son más exigentes computacionalmente que los de Legendre, pero también son invariantes a transformaciones lineales (escalado, rotación), siendo más atractivos para procesamiento de imágenes.

Momentos de Legendre y Zernike (2)

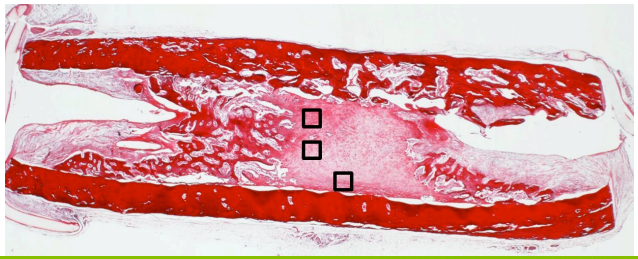
- El momento calculado para una ventana centrada en un píxel puede interpretarse como una convolución de la imagen con una máscara.
- Cuanto mayor es el número de momentos, mejor es la reconstrucción.
- Las implementaciones existentes en CPU se han optimizado utilizando algoritmos recursivos, lo que plantea un reto a la GPU debido a las dependencias de datos entre iteraciones de los lazos.
- La alternativa de cálculo utilizando métodos directos (vs. iterativos) resulta más rápida en GPU, ya que aunque se requieren más computos, se explota mucho más paralelismo.

Ejemplo: Momento de Zernike de orden 0 aplicado al canal rojo de 6 ventanas 64x64

Tejido óseo:
- 166.33
- 146.91
- 148.53



Cartílago:
- 231.12
- 234.75
- 226.33



Computación del momento de orden N y repetición M usando métodos directos

- La carga computacional es mucho más elevada que utilizando métodos iterativos, pero también admite mayor paralelismo, lo que resulta decisivo para reducir el tiempo en la GPU.

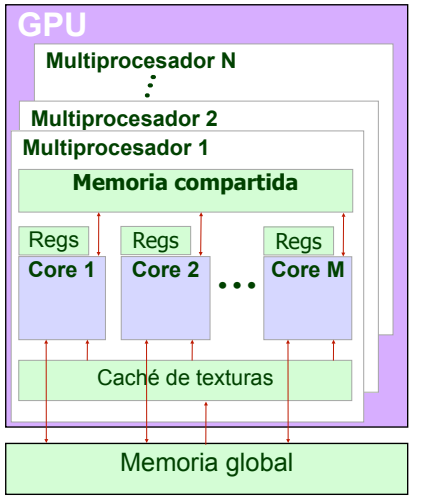
FUNCTION Radial Polynomial (ρ, n, m)
$radial = 0$
for $s = 0$ **to** $(n-m)/2$
$c = (-1)^s \frac{(n-s)!}{s!\left(\frac{n+|m|}{2}-2\right)!\left(\frac{n-|m|}{2}-s\right)!}$
$radial = radial + c \cdot \rho^{n-2s}$
end for **return** $radial$

FUNCTION Zernike Moments (n, m)
$Z_r = 0$
$Z_i = 0$
$cnt = 0$
for $y = 0$ **to** $N-1$
for $x = 0$ **to** $N-1$
$\rho = \frac{\sqrt{(2x-N+1)2+(2y-N+1)^2}}{N}$
if $\rho \leq 1$
$radial =$ RadialPolynomial (ρ, n, m)
$theta = \tan^{-1}\left(\frac{2y-N+1}{2x-N+1}\right)$
$z_r = z_r + f(x,y) \cdot radial \cdot \cos(m \cdot theta)$
$z_r = z_i + f(x,y) \cdot radial \cdot \sin(m \cdot theta)$
$cnt = cnt + 1$
endif
endfor
endfor
return $\frac{n+1}{cnt}(z_r + jz_i)$

2.1.2. Implementación en CUDA

Las bazas de la GPU para mejorar implementaciones ya existentes en CPU

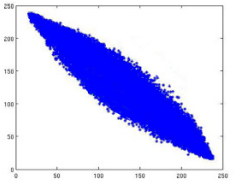
- Explotar el paralelismo inherente a la aplicación: Elegir el número de threads óptimo y su despliegue en bloques.
- Utilizar mecanismos específicos para el acceso a memoria en GPU.
  - Evitar conflictos en el acceso a bancos de memoria.
  - Grandes latencias en el acceso a memoria global, poca capacidad de memoria local y muy cercana a la velocidad del banco de registros.
  - Elección de la estructura de datos más adecuada.
- Crear kernels que reutilicen datos.

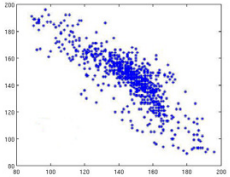


Ejemplo de optimización sobre la jerarquía de memoria de CUDA

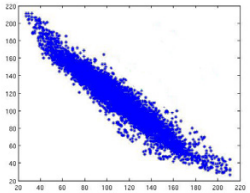
- Las matrices de co-ocurrencia tienden a concentrar sus elementos en torno a la diagonal principal.
- Algunos ejemplos ilustrativos:

Regeneración ósea según diferentes tinciones:





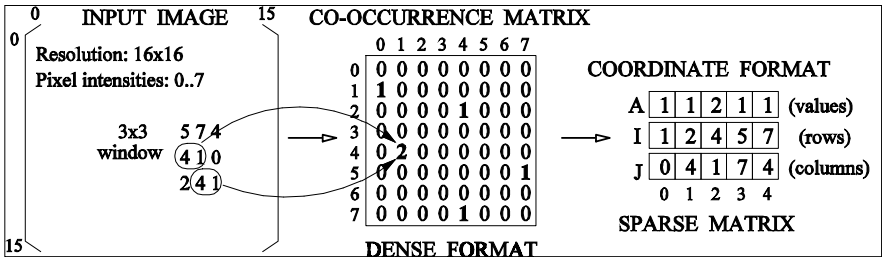
Diagnóstico del cancer



- Solución: Utilizar matrices dispersas para minimizar el consumo de memoria.

Implementación mediante matrices dispersas: Utilizamos el formato coordenado

- Ejemplo:



- Resulta algo más difícil de acceder que las listas simple o doblemente enlazadas, pero también es más compacto, que es nuestra prioridad aquí.

2.1.3. Análisis de rendimiento

Procesadores utilizados en la evaluación experimental de todos estos algoritmos

Parámetro hardware	CPU	GPU
Modelo comercial	Intel Core 2 Duo	Nvidia GeForce
Código de referencia	E6400	8800 GTX (G80)
Frecuencia de reloj	2.13 GHz	575 MHz
Potencia bruta de cálculo	10 GFLOPS	520 GFLOPS
Anchura del bus de memoria	64 bits	384 bits
Reloj para la memoria	2x333 MHz	2x900 MHz
Ancho de banda de la memoria	10.8 GB/sg.	86.4 GB/sg.
Tamaño y tipo de memoria	2 GB DDR2	768 MB GDDR3

Resultados para las conversiones de color

Conversión	Tiempo en CPU	Tiempo en GPU	Aceleración en GPU
RGB to XYZ	140.01 ms.	1.27 ms.	109.47x
RGB to Luv	273.83 ms.	1.42 ms.	191.62x
RGB to LA*B*	267.92 ms.	2.23 ms.	119.66x
RGB to HSV	16.60 ms.	0.57 ms.	28.98x
RGB to sRGB	123.51 ms.	1.23 ms.	99.84x
sRGB to XYZ	16.50 ms.	0.43 ms.	37.59x
sRGB to Luv	150.31 ms.	0.57 ms.	263.25x
sRGB to LA*B*	144.41 ms.	1.29 ms.	111.68x

Resultados para el operador LBP

Tamaño de la ventana	CPU/C++	GPU/Cg	GPU/CUDA	Aceleración en GPU
128x128	3.95 ms.	1.01 ms.	0.07 ms.	54.86 x
256x256	17.83 ms.	1.09 ms.	0.14 ms.	127.35 x
512x512	76.70 ms.	1.92 ms.	0.41 ms.	184.81 x
1K x 1K	310.65 ms.	6.88 ms.	1.56 ms.	198.62 x
2K x 2K	1234.96 ms.	23.91 ms.	6.11 ms.	201.98 x

- Se consiguen mayores aceleraciones a medida que aumenta el tamaño de la ventana.
- La GPU es más efectiva en ventanas grandes, mientras que CUDA se luce más en tamaños de ventana pequeños.

Resultados para las matrices de co-ocurrencia. Mejoras en GPU y usando matrices dispersas

Tamaño de la ventana	CPU/ densa	GPU/ densa	GPU/ dispersa	% de datos no nulos	Mejora dispersa	GPU/CPU
4x4	1.36	7.61	0.10	0.024%	76.10x	13.60x
8x8	2.82	7.62	0.16	0.097%	47.62x	17.62x
16x16	2.82	7.58	0.39	0.390%	19.43x	7.23x
32x32	3.04	7.63	0.74	1.562%	10.31x	4.10x
64x64	3.08	7.76	1.74	6.250%	4.45x	1.77x
128x128	2.94	8.54	7.70	25%	1.10x	0.38x
256x256	2.96	9.19	46.49	100%	0.19x	0.32x

La GPU es más efectiva cuanto más dispersa es la matriz.

Resultados para las matrices de co-ocurrencia. Mejoras según formato de la matriz dispersa

Tamaño de la ventana	Formato coordenado	Listas simplemente enlazadas	Listas doblemente enlazadas
4 x 4	0.10	0.13	0.30
8 x 8	0.14	0.25	0.58
16 x 16	0.36	1.02	1.52
32 x 32	0.45	2.31	4.91
64 x 64	1.13	3.46	6.52
128 x 128	6.58	19.85	23.33
256 x 256	43.19	65.99	78.19

La GPU es más efectiva utilizando formatos sencillos de matrices dispersas.

Resultados para los momentos de Zernike para toda una imagen de 1K x 1K

Momentos calculados (nº)	Método recursivo óptimo en CPU	Método directo en GPU	Aceleración en GPU
A4,* (3)	62.5 ms.	19.0 ms.	3.28 x
A8,* (5)	54.5 ms.	36.6 ms.	1.48 x
A12,* (7)	62.5 ms.	50.5 ms.	1.23 x
A16,* (9)	78.0 ms.	68.2 ms.	1.14 x
A20,* (11)	93.5 ms.	90.0 ms.	1.03 x

Mayores aceleraciones en GPU en los momentos de orden inferior.

Síntesis de resultados. Caracterización de cada algoritmo sobre la GPU

Rasgo a cuantificar	Operadores streaming: Conversiones de color	Operador stencil: LBP	Operador con indirecciones: Matrices de coocurrencia	Operador recursivo: Momentos de Zernike
Entrada	Un píxel	Ventana 3x3	Subimagen	Ventana variable
Salida	Un píxel	Un solo valor	Cto. de valores	Matriz de tam. var.
Canales de color	Tres	Uno	Uno	Tres
Rango computacional	Por píxel	Por píxel	Por subimagen	Por píxel
Peso computacional	Muy ligero	Ligero	Pesado	Muy pesado
Tipo de operador	Streaming	Stencil	Accesos indirectos	Recursivo
Reutilización de datos	Ninguno	Bajo	Muy alto	Alto
Localidad de acceso	Ninguna	Baja	Alta	Muy alta
Intensidad aritmética	Muy alta	Media	Alta	Baja
Acceso a memoria	Bajo	Medio	Alto	Muy alto
Aceleración en GPU	25-250x	50-200x	1-2x	0.7-1x

Recomendaciones finales para seleccionar los kernels que son más apropiados en GPU

- Implementar sobre todo kernels de computación:
  - Regulares.
  - Masivamente paralelos.
  - Aritméticamente intensivos.
- Adaptar el algoritmo para expresarlo mediante:
  - Un número elevado de hilos.
  - Hilos muy ligeros.
- Rediseñar el algoritmo para evitar recursividades y recurrencias.
  - Adoptar variantes no recursivas si es posible.
- Pensar en las optimizaciones específicas de CUDA:
  - Maximizar el ancho de banda: Coalescing y conflictos en bancos.
  - Maximizar el uso del tamaño de memoria compartida disponible.

2.2. Segmentación y clasificación de imágenes biomédicas de alta resolución

Supercomputador BALE: Cluster de 55+18 nodos híbridos CPU-GPU

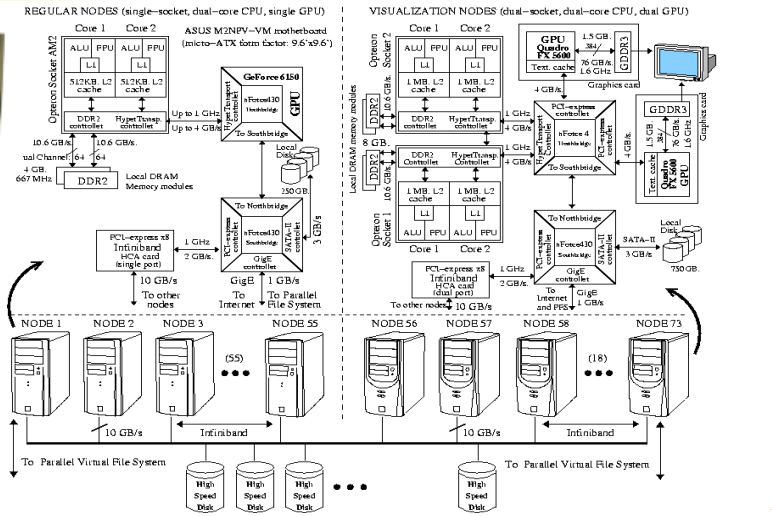
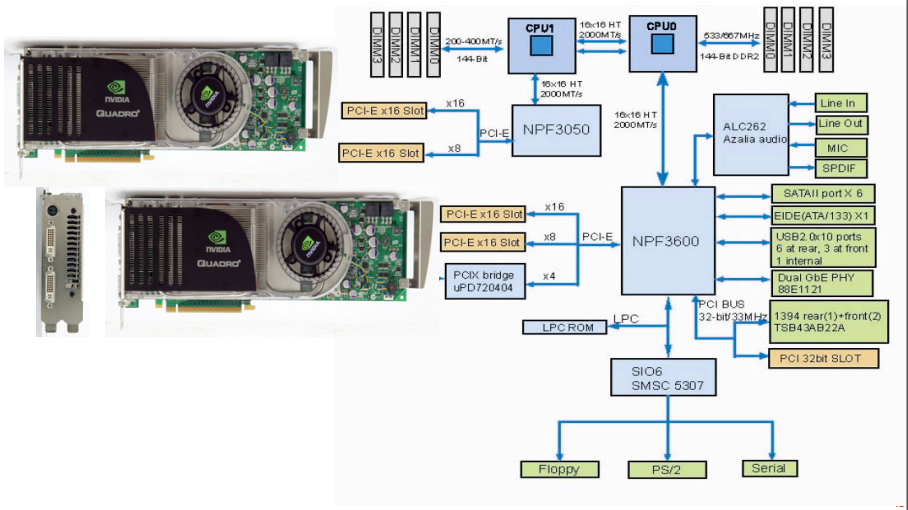


Diagrama de bloques de cada uno de los 18 nodos más potentes de BALE



Prestaciones comparativas CPU/GPU en cálculo y acceso a memoria

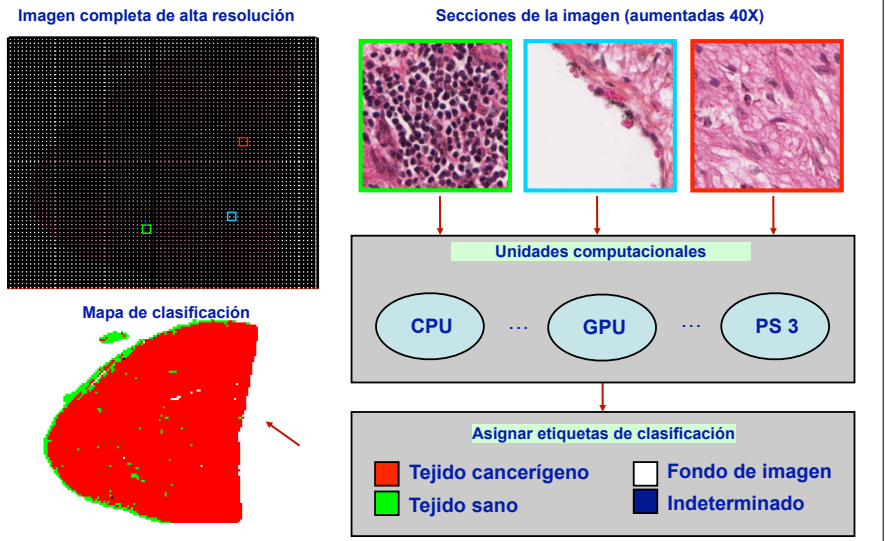
Procesador	CPU (AMD)	GPU (Nvidia)
Modelo arquitectural	Opteron X2 2218	Quadro FX 5600
Frecuencia de reloj	2.6 GHz	600 MHz / 1.35 GHz
Número de núcleos	2 cores	128 stream processsors
Potencia de cálculo	2 cores x 4.4 GFLOPS = 8.8 GFLOPS	madd(2 FLOPS) x128 SP x 1.35 GHz = 345.6 GFLOPS
En el total de 18 nodos x 2 zócalos	316.8 GFLOPS	¡¡ 12.4 TFLOPS !!

Memoria	CPU (AMD)	GPU (Nvidia)
Capacidad y tipo	8 Gbytes de DDR2	1.5 Gbytes de GDDR3
Frecuencia de reloj	2x 333 MHz	2x 800 MHz
Anchura del bus	128 bits (doble canal)	384 bits
Ancho de banda	10.8 Gbytes/sg.	76.8 Gbytes/sg.

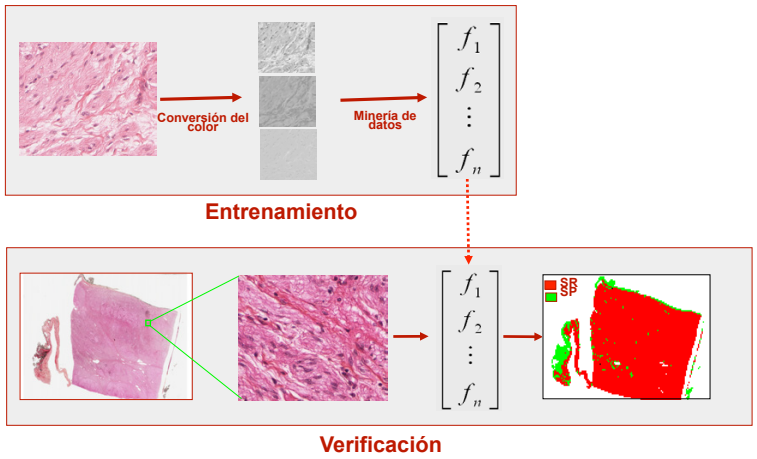
Paralelismo y programación

- Niveles de paralelismo:
  - Multi-nodo: Los 18 nodos de visualización de BALE.
  - Multi-zócalo: SMP (Symmetric MultiProcessing): 2 CPUs y 2 GPUs.
  - Multi-core: Hilos concurrentes en la CPU.
  - Many-core: SIMD (Simple Instruction Multiple Data) en GPU.
  - PNI en las instrucciones máquina (procesadores segmentados y superescalares).
- Herramientas de programación:
  - MPI: Para alojar recursos entre los nodos y comunicarlos en BALE.
  - POSIX threads (P-threads): Para CPUs multi-core.
  - CUDA: Para GPUs many-core.
  - C/C++: Para programación secuencial en el host.

Visión general



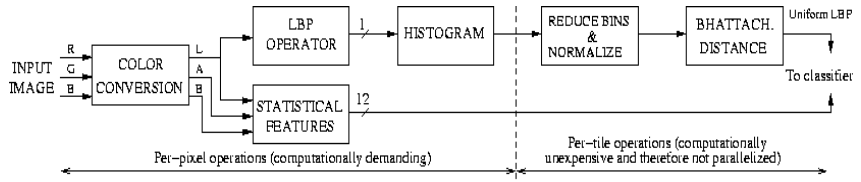
Descripción del proceso



Ganancias en GPU para el clasificador KNN
(N= número de ventanas; L= número de momentos computados)

N	L	K = 5			K = 10			K = 20		
		CPU	GPU	Mejora	CPU	GPU	Mejora	CPU	GPU	Mejora
200	4	6,06	112,16	0,05x	8,38	113,39	0,07x	13,22	115,42	0,11x
1000	4	89,97	113,17	0,79x	122,37	114,75	1,06x	183,33	121,40	1,51x
2000	4	350,73	114,06	3,07x	472,58	116,91	4,04x	714,36	128,57	5,55x
4000	4	1383,85	109,61	12,62x	1826,74	121,12	15,08x	2814,86	139,47	20,18x
8000	4	5506,27	124,72	44,14x	7396,22	133,93	55,24x	11211,92	188,72	59,41x
8000	8	5953,49	126,78	46,95x	7902,55	136,44	57,91x	11694,30	193,00	60,59x
8000	16	6661,26	131,44	50,67x	8653,35	145,49	59,47x	12464,66	200,68	62,11x
8000	32	8255,37	151,32	54,55x	10156,24	160,37	63,33x	13908,83	206,61	67,31x
8000	64	11660,05	182,02	64,05x	13455,99	191,04	70,43x	17162,53	243,27	70,53x

El proceso completo de caracterización y clasificación



- Conversión de color: De RGB a LA*B*
  - Operador típico de GPU, pero la memoria compartida de CUDA aún ofrece un potencial de mejora.
- Matriz de co-ocurrencia para el cálculo de párs. estadísticos:
  - Baja reutilización de datos.
  - El incremento de los contadores obliga a leer y escribir en memoria.
- Operador LBP (Local Binary Pattern):
  - Utiliza una ventana de 3x3 píxeles vecinos.
- Histograma:
  - Caracterización similar a la de las matrices de co-ocurrencia.

Resultados experimentales utilizando un solo procesador de BALE (ver HW. más adelante)

Tamaño imagen	Resolución en píxeles	Número de mosaicos de 1K x 1K que contiene la imagen
Pequeña	32980 x 66426	33 x 65 = 2145
Mediana	76543 x 63024	75 x 62 = 4659
Grande	109110 x 80828	107 x 79 = 8453

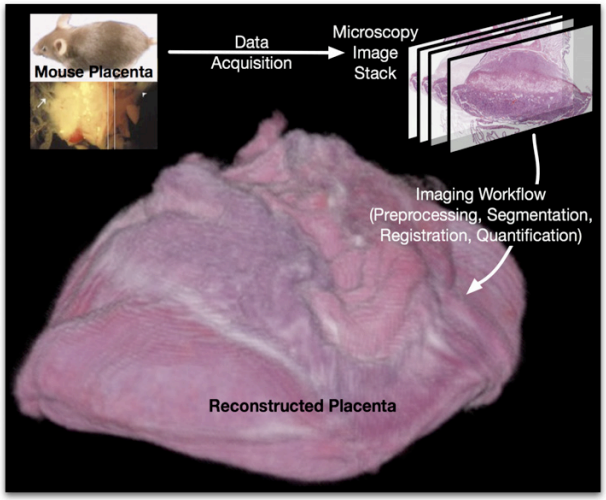
Tamaño imagen	En la CPU		En la GPU	
	Matlab	C++	Cg	CUDA
Pequeña	2h 57' 29"	43' 40"	1' 02"	27"
Mediana	6h 25' 45"	1h 34' 51"	2' 08"	58"
Grande	11h 39' 28"	2h 51' 23"	3' 54"	1' 47"

Aceleración sobre la carga computacional anual del hospital (OSU Medical Center)

- 400 pacientes, 8-10 muestras registradas de cada uno.
- En una sola CPU, el tiempo de procesamiento total para la caracterización y clasificación de todas las muestras es de:
  - 2 años, utilizando Matlab.
  - 3.4 meses, utilizando C++.
- En una sola GPU, el tiempo queda reducido a:
  - 5.3 días, utilizando Cg.
  - 2.4 días, utilizando CUDA.
- Paralelizando sobre BALE (ver HW. a continuación):
  - Menos de 2 horas, utilizando 16 nodos CPU-GPU.
- La aceleración total supera los seis órdenes de magnitud.

2.3. Registro y reconstrucción 3D de imágenes biomédicas

Registro de imágenes para el análisis de volúmenes microscópicos de alta resolución

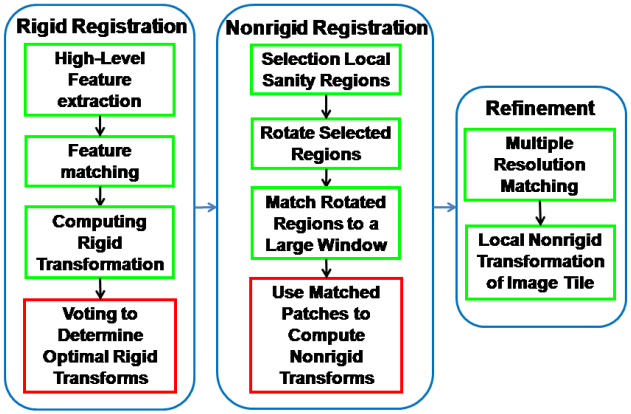


Desafíos y objetivos

- Complejidad: El registro de una muestra de placenta mamaria de ratón compuesta de 500 imágenes tarda 181 horas en una CPU de gama alta.
- Objetivo: Reducir este tiempo combinando paralelismo.
- Retos a resolver en el camino:
  - La distorsión no rígida es una tarea compleja de acometer.
  - Aparecen gran cantidad de rasgos a correlar.
  - Las imágenes son enormes: Desde 16K x 16K a 23K x 62K pixels, ocupando cada una entre 0.7 y 1.5 Gbytes. Para una media de 1,200 imágenes por cada placenta mamaria, necesitamos 3 Tbytes.
  - La carga de trabajo es intensiva y los métodos iterativos inviables.
- Mejora final: Factor de aceleración de 49x en 16 nodos.

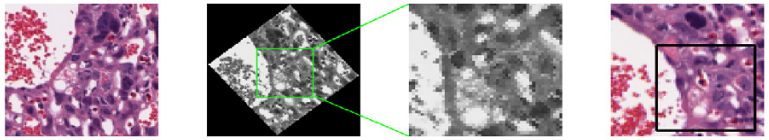
Descripción del proceso global

- Los bloques verdes son operadores locales independientes que pueden procesarse en paralelo.

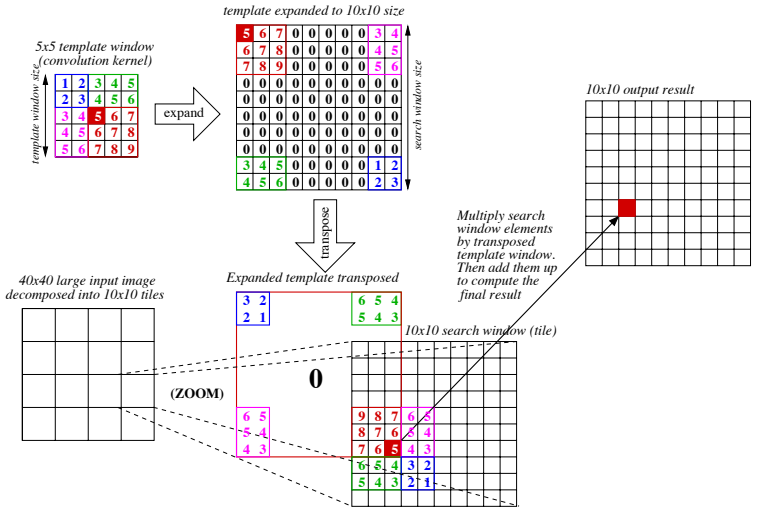


Coincidencias entre imágenes

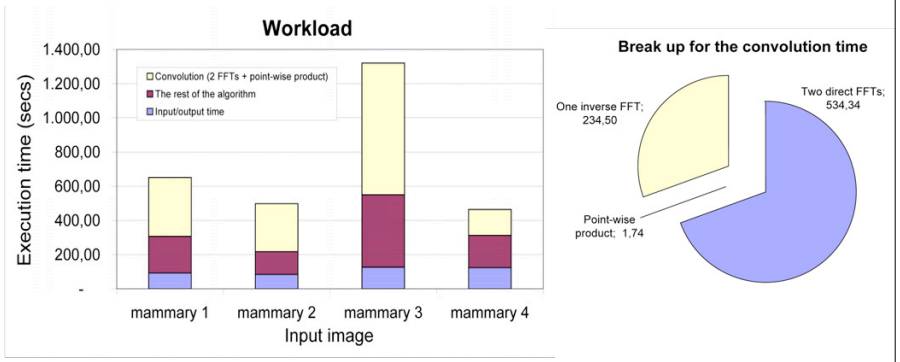
- Un fragmento de la primera imagen se sitúa sobre la posición inicial de la segunda imagen (registro rígido).
- Este fragmento se utiliza como plantilla para las coincidencias dentro de una ventana de búsqueda más grande centrada en dicha ubicación.
- El grupo de píxeles dentro de la ventana de búsqueda con mayor afinidad se toma como candidato para la coincidencia.
- Las coordenadas de coincidencia se toman como centros finales.



La computación de la correlación cruzada normalizada (NCC) basándonos en la FFT

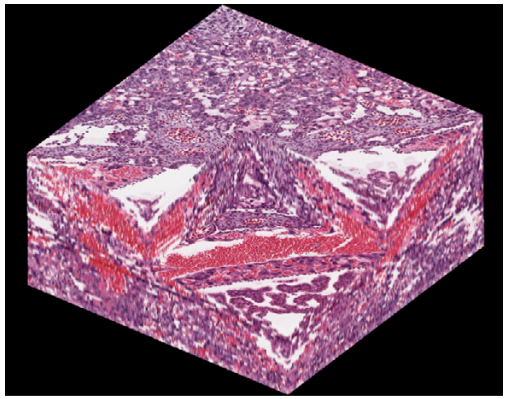


Distribución de la carga computacional para cada una de las fases computacionales

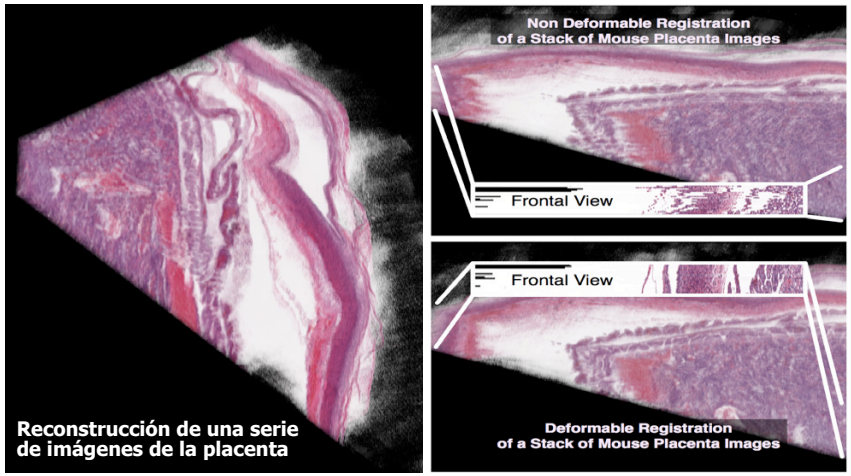


El resultado para una pila de 20 imágenes de 16K x 16K píxeles cada una

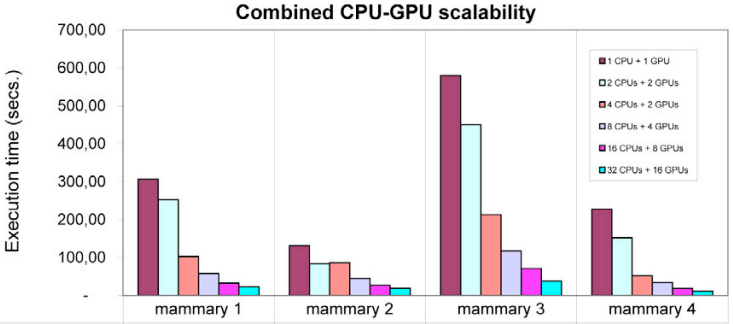
- Hemos troquelado una esquina para apreciar las secciones entre las imágenes:



Resultado final

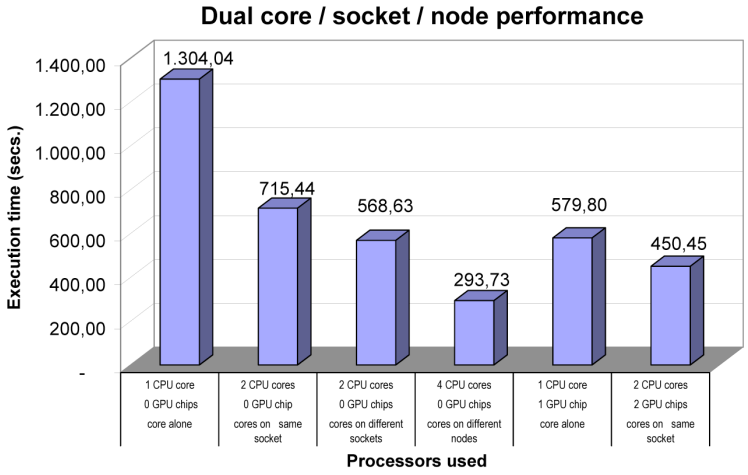


Resultados experimentales: Escalabilidad de BALE

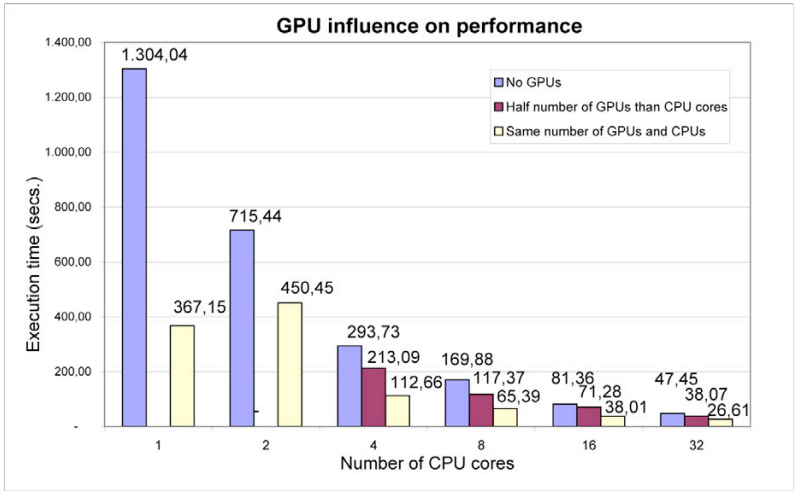


	mammary 1	mammary 2	mammary 3	mammary 4
1 CPU + 1 GPU	306,02	131,13	579,80	227,51
2 CPUs + 2 GPUs	252,02	83,89	450,45	152,13
4 CPUs + 2 GPUs	102,81	85,83	213,09	51,72
8 CPUs + 4 GPUs	57,99	45,36	117,37	34
16 CPUs + 8 GPUs	32,24	26,74	71,28	18,54
32 CPUs + 16 GPUs	22,62	18,87	38,07	11,89

Rendimiento según la configuración de cada nodo del supercomputador BALE



Influencia de la GPU en el rendimiento



Conclusiones de la paralelización del registro de imágenes

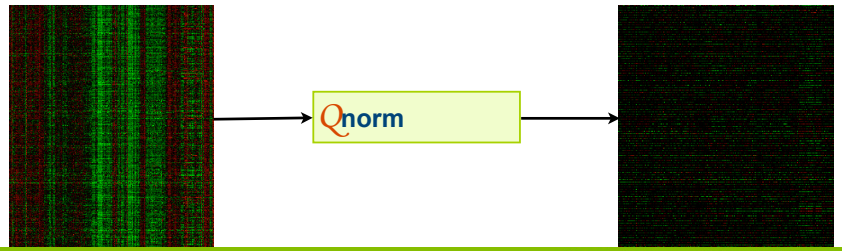
- La paralelización sobre BALE permite explotar conjuntamente múltiples niveles de paralelismo:
  - Multinodo y multiprocesamiento simétrico (SMP) [vía MPI].
  - Paralelismo de tareas (multi-hilo) [vía P-threads].
  - Paralelismo de datos (SIMD) [vía CUDA].
- Los núcleos computacionales se comportan de forma muy distinta en CPU y GPU según su:
  - Intensidad aritmética.
  - Patrón de acceso a memoria.
  - Reutilización de los datos.
- Se consiguen grandes mejoras en un solo nodo gracias a la GPU, y aceleración casi lineal en el multiprocesador.

3. Análisis genómico

3.1. Q-norm: Comparativa frente a supercomputadores de memoria compartida y distribuida

Un algoritmo básico: Q-norm

- Se trata de un proceso de normalización por cuantiles estadísticos de muestras genéticas que comprenden 470 vectores de 6.5 millones de datos (12 Terabytes en total).
- Colaboradores de mi Departamento en Málaga:
  - A. Rodríguez (implementación en GPU).
  - J.M. Mateos, O. Trelles (implem. en mem. compartida y distribuida).

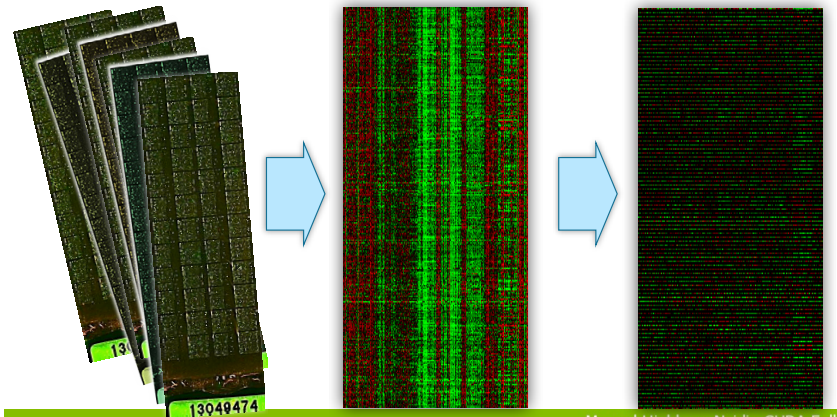


Q-norm:
El proceso de normalización por cuantiles

Entrada: Múltiples microarrays de DNA

Datos de intensidad global (Un array por columna)

Salida: Datos Q-normalizados



Descripción del proceso por fases

1	5	3	5
2	1	6	7
3	2	2	6
4	6	1	8

1	1	1	5
2	2	2	6
3	5	3	7
4	6	6	8

2
3
4.5
6

2	2	2	2
3	3	3	3
4.5	4.5	4.5	4.5
6	6	6	6

2	4.5	4.5	2
3	2	6	4.5
4.5	3	3	3
6	6	2	6

Ordenar cada columna de la matriz (cada vector)

Calcular la media por filas

Actualizar cada valor con la media aritmética de su fila

Recolocar los datos de las columnas en su posición original

Plataformas hardware sobre las que ejecutamos el código

- Multiprocesador de memoria distribuida ("Pablo"):
  - Cluster de 256 blades JS20 de IBM (512 CPUs). Cada blade tiene:
    - 2 CPUs PowerPC 970FX @ 2 GHz.
    - 4 Gbytes de memoria, para un total de 1 Terabyte de memoria distribuida.
  - Red de interconexión: MIRYNET.
- Multiprocesador de memoria compartida ("Picasso"):
  - Superdome de 16 nodos de HP (128 cores). Cada nodo tiene:
    - 4 CPUs Intel Itanium-2 (alias "Montecito") de doble núcleo @ 1.6 GHz.
  - Total de memoria disponible en conjunto: 384 Gbytes.
  - Red de interconexión: Propietaria.
- Un PC de nuestro laboratorio ("Antonio Banderas"):
  - CPU: Intel Core 2 Quad Q9450, 2.66 GHz, 1.33 GHz FSB, 12 MB L2.
  - GPU: GeForce 9800 GX2, 600/1500 MHz, 512 MB de DDR3 a 2 GHz.
  - Discos duros: 2 x 72 GB (RAID 0) Raptors de WD a 10000 RPM.

Pablo (1/8 del viejo Mare Nostrum - 2004). Coste aproximado: 600.000 €

pab scbi PLATAFORMA ANDALUZA DE BIOINFORMÁTICA

SC scbi

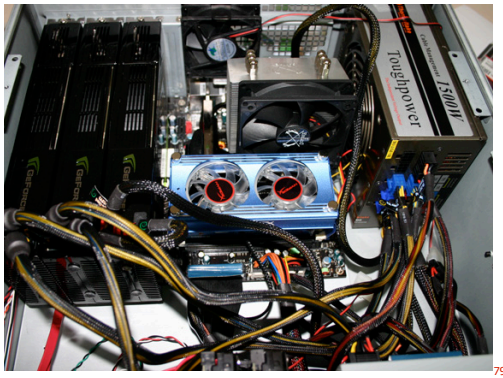
Picasso (Superdome de HP - 2008).
Coste superior a los 2 millones de euros

Nuestro PC "Antonio Banderas":
Dotado con una GPU GeForce 9800 GX2 (2008)

- Doble GPU GeForce 8800 (G80).
- Doble placa de circuito impreso.
- Doble memoria de vídeo de 512 MB y 256 bits.
- Un solo conector PCI-express.

Potencial de mejora en GPU

- Hemos utilizado una sola GPU, pero disponemos de hasta seis, lo que nos otorga un gran potencial si montamos arquitecturas multichip y/o multitarjeta.



Versión OpenMP para Picasso
(multiprocesador de memoria compartida)

```
nE = LoadProject(fname, fList);  // Paso 1: E/S

for (i=0;i< nE;i++) { // Paso 2: Ordenar columnas
  LoadFile(fList, i, dataIn);
  Qnorm1(dataIn, dIndex, fList[i].nG);
  PartialRowAccum(AvG, dataIn, nG);
  // Gestionar el vector índice en memoria o disco
}

for (i=0;i<nG;i++)    // Paso 3: Computar medias
  AvG[i].Av /=AvG[i].num;

// Producir el fichero de salida con
// el vector de datos ordenados para cada columna

for (i=0;i<nE;i++)
{
  Obtener el vector índice del disco para cada col.
  for (j=0;j<nG;j++) { // preparar vector salida
    dataOut[dIndex[j]]=AvG[j].Av;
    // Pos. en fichero para escribir vector salida
  }
}
```

#pragma omp parallel shared
From, To, Range
// Open general parallel section

SOLO HAY QUE AÑADIR ESTAS DOS DIRECTIVAS AL CODIGO ORIGINAL

#pragma omp parallel shared
From, To, Range

Versión con pase de mensajes para Pablo (multiprocesador de memoria distribuida)

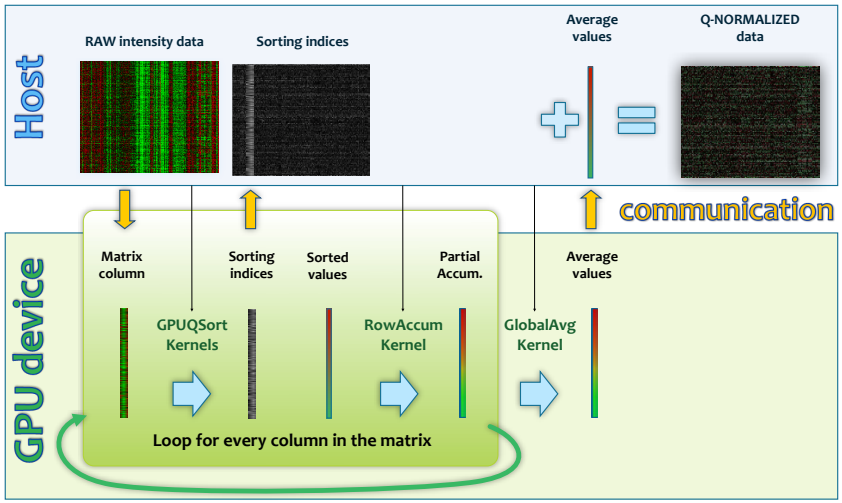
```
                                  Master
                                  Slave(s)
Get Parameters, Initialize                  Start with params
CalculateBlocks(nP,IniBlocks)
Broadcast(IniBlocks)                        Receive (Block)
                                            while(!ENDsignal) {
                                            for each experiment in block {
                                               LoadExperiment(Exp);
                                               SortExperiment(Exp);
                                               AcumulateAverage(Exp);
                                            }
while (ThereIsBlocks) {                     AverageBlock(ResultBlock);
  Receive(ResultBlock,who)                  Send(ResultBlock);
  AverageBlock(ResultBlock)
  if(!SendAllBlocks) {
    CalculateNextBlock(NextBlock)
    Send(who,NextBlock)                     Receive(Block);
                    }
}                 }
Broadcast(ENDsignal)                        ReportResults
```


Versión CUDA para Antonio Banderas

```
__global__ void QSortGPU(int *dataIn, int *dIndex)  // The heavy kernel on GPU
{  // Index to the element to be computed by each thread
  int idx = (blockIdx.x * blockDim.x) + threadIdx.x;
   // Sort the dataIn array and return dIndex as output permutation
}
```

```
__host__ void main()  // The host program running on CPU
{ // N is the number of input samples; p is the number of gene expression values
  // Bi (threads/block) and NBi (number of blocks) determine parallelism
  int N = LoadProject(fname,fList);  // p is read from file similarly
  for (i=0; i<N; i++) {
    LoadFromFile(fList,i,dataIn); CPUtoGPU(dataIn);  // Move data from file to GPU
    B1 = 512; // NB1 depends on p and B1
    QSortGPU <<<NB1,B1>>>(p,*dataIn,*dIndex);        // CUDA kernel invocation
    GPUtoCPU(p,dIndex); WriteToFile(p,dIndex);  // Move data from GPU back to file
    RowAccum <<<NB2,B2>>>(p,*dataIn,*dataAvg);       // 2nd CUDA kernel
  }
  GlobalAvg <<<(NB3,B3)>>>(*dataAvg,N);              // 3rd CUDA kernel
  GPUtoCPU(*dataAvg);
  // For all samples, read *dIndex file and update *dataIn file according to *dataAvg
}
```

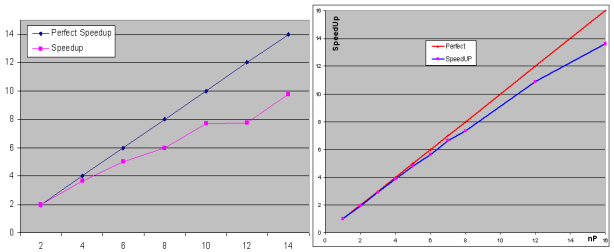

Funcionamiento interno en GPU



Tiempos de ejecución de Q-norm (en segs.): Comparativa con supercomputadores

Pablo (fracción del extinto Mare Nostrum).
Incluye el tiempo de E/S

Picasso (Superdome de HP).
Incluye el tiempo de E/S



Nuestro Antonio Banderas (PC con GeForce 9800 GX2)

PC	Sólo CPU	Con GPU	Acel.
Procesam.	807	115	7.03x
E/S	179	179	
Comun.	0	25	
Total	986	319	3.09x

# CPUs	1	2	3	4	5	6	7	8	10	12	14	16
Pablo y	1708	867		469		339		285	221	220	175	
su aceler.		1.97x		3.64x		5.03x		5.99x	7.71x	7.76x	9.76x	
Picasso y	210	108	72	54	43	37	31	28		19		15
su aceler.		1.94x	2.91x	3.67x	4.82x	5.64x	6.66x	7.32x		10.9x		13.6x

Desglose de la paralelización en GPU

El tamaño del bloque CUDA resulta decisivo, pero la GPU no luce más porque Q-norm carece de intensidad aritmética.

Hilos por bloque	Tiempo de GPU	Mejora parcial	Mejora acumul.
32	204,38		
64	170,24	17%	17%
128	141,21	18%	31%
256	122,13	13%	40%
512	114,92	6%	44%

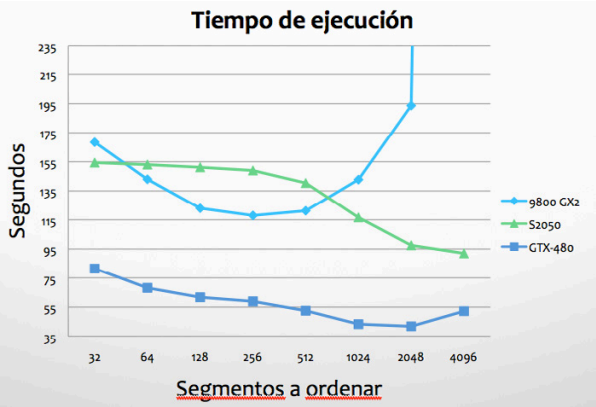
Transferencias	Segs.
1. De disco a CPU (SAS y RAID 0 de 2)	162,32
2. De CPU a GPU (PCI-express 2)	13,89
3. De GPU a CPU (PCI-express 2)	11,51
4. De CPU a disco (asíncrono)	16,70
Tiempo total de E/S (1+4)	179,02
Tiempo total de comunic. CPU-GPU	25,40
Tiempo total de transferencia	204,42

Q-norm se encuentra limitado por E/S.

Comparativa CPU vs. GPU: Resumen de prestaciones hardware

Procesador	Q9450	9800 GX2	C2050	GTX 480
Arquitectura	Intel Core 2	Nvidia G92	Nvidia Fermi	Nvidia Fermi
Número de cores	4	128	448	480
Frecuencia core	2.66 GHz	1.35 GHz	1.15 GHz	1.40 GHz
Potencia de cálculo	32.56 GFLOPS	345.6 GFLOPS	515.2 GFLOPS	672 GFLOPS
Tamaño DRAM	4 GB DDR3	512 MB GDDR3	3 GB GDDR5	1.5 GB GDDR5
Ancho de banda	28.8 GB/s.	64 GB/s.	144 GB/s.	177 GB/s.
Coste aproximado	€ 300	€ 200	€ 1500	€ 500

Comparativa CPU vs. GPU: Tiempos de ejecución de Q-norm (segs.)



- Mejora en GPU vs. CPU: Hasta 20x (4x si contamos E/S).

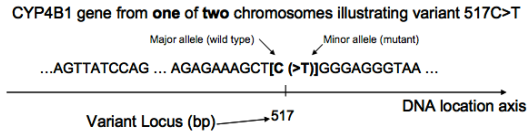
Para descargarse los códigos

- Las tres versiones analizadas, así como documentación anexa, pueden descargarse de nuestro servidor Web:
  - https://chirimoyo.ac.uma.es/qnorm

3.2. Interacciones genéticas en GWAS (Genomic-Wide Association Studies)

Minería de datos en SNPs

- El análisis de SNPs (Single Nucleotide Polymorphisms) en muestras genéticas de pacientes (GWAS) permite inferir la predisposición a ciertas enfermedades degenerativas como el Alzheimer.



- El problema una vez más es el elevado volumen de datos a procesar: Más de 2.5 billones de interacciones para una muestra de 1329 individuos (1014 sanos y 315 enfermos).

Resultados experimentales. Aceleración y relación rendimiento/coste.

	CPU Xeon E5645	GPU Tesla C1060	GPU Tesla C2050	GPU GeForce GTX 480
Número de cores	6	240	448	480
Velocidad del core	2.40 GHz	1.30 GHz	1.15 GHz	1.40 GHz
GFLOPS	144	312	515	672
T. cómputo (segs)	45 234	1 195 (38x)	507 (89x)	270 (167x)
Comunic. CPU-GPU	Ninguna	187	183	170
Tiempo total	45 234 segs. (más de 12 horas)	1 385 (32x) (23 minutos)	699 (65x) (11 minutos)	441 (103x) (7 minutos)
Coste estimado	$ 400	$ 1600	$ 1600	$ 400
Escalabilidad	Referencia	Buena	Mejor	Optima
Rendimiento/coste	1	8,17	16,17	103

Reflexiones finales

- La computación de altas prestaciones ha avanzado de forma uniforme en los últimos 20 años, demostrando escalabilidad hacia un futuro donde el paralelismo tiene una importancia decisiva.
- Existe un déficit en la programación paralela de aplicaciones, donde la bioinformática suele ser el referente. Debemos suplirlo con una buena oferta formativa.
- Aunque la curva de aprendizaje es alta al principio, se suaviza mucho después, y termina recompensando sobremanera el esfuerzo realizado.
- La programación de plataformas gráficas se erige como uno de los caballos ganadores para lograr esta meta.