

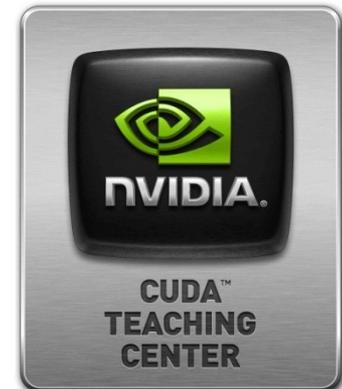
Computación de Propósito General en Unidades de Procesamiento Gráfico (GPGPU)

M. Freire

E. Dufrechou, P. Ezzatti y M. Pedemonte



UNIVERSIDAD
DE LA REPUBLICA
URUGUAY



Clase 12a

Ecosistema CUDA (extendido)

Contenido

- **Ecosistema CUDA**
- **Nvidia Performance Primitives (NPP)**
- **OpenACC**

Ecosistema CUDA

Ecosistema CUDA

- **CUBLAS, NVBLAS**
- **Cusparse**
- **CuSolver**
- **CUDA Profiling Tools Interface (CUPTI)**
- **CURAND**

Conceptos previos a CUBLAS

BLAS (Basic Linear Algebra Subprograms):

- Es una especificación!
- Las operaciones se dividen en tres niveles:
 - BLAS-1, BLAS-2 y BLAS-3.
- Cuanto mayor sea el nivel de las operaciones, mejores resultados de optimización se obtienen.
- Matrices generales y estructuradas (triangulares, banda)

BLAS

- Mantiene una nomenclatura estándar.
- 5 letras agrupadas en 3 bloques:
 - TMMOO: tipo de datos, tipo de matriz, código de operación
- Nomenclatura:

	Real	Complejo
Simple Pre.	S	C
Doble Pre.	D	Z

Densa
Banda

	General	Simétrica/ Hermitiana	Triangular
Densa	GE	SY/HE	TR
Banda	GB	SB/HB	TB

BLAS

BLAS1

- Implementa operaciones tipo vector – vector y escalares.
- $O(n)$ datos y $O(n)$ operaciones.
- Movimiento de Datos:
 - Copia, intercambio.
- Operaciones vectoriales:
 - Escalado vectorial.
- Operaciones de reducción:
 - Producto escalar, norma vectorial, sumatoria, máximos.
- Nomenclatura operaciones:
 - xYYYYY.
 - x {S,C,D,Z}, Y: operación.
 - Ej:xCOPY(.....).

BLAS

BLAS2

- **Implementa operaciones tipo matriz - vector.**
- **$O(n^2)$ datos y $O(n^2)$ cálculos**
- **Ejemplos destacados:**
 - **Producto matriz - vector.**
 - **Actualizaciones de rango 1 y 2**

BLAS

BLAS3

- Implementa operaciones tipo matriz - matriz.
- $O(n^2)$ datos y $O(n^3)$ cálculos
- Operaciones destacadas:
 - Producto matriz – matriz.
 - Actualizaciones de rango k y $2k$.
 - Resolución de sistemas triangulares.

BLAS

Distintas versiones según plataformas:

- **ACML de AMD**
- **ESSL de IBM**
- **MKL de Intel**
- **Sun Performance Library de SUN**
- **GotoBLAS para Intel Pentium, SPARC, IBM PowerPC (discontinuada)**
- **OpenBLAS (basada en GotoBLAS2)**
- **ATLAS (implementación portable)**

BLAS

- **MKL es la implementación de referencia en arquitecturas Intel.**
- **Es la biblioteca de álgebra lineal utilizada por MATLAB (en gral).**
- **Desempeño de OpenBLAS comparable con MKL para algunas arquitecturas**
- **Algunas versiones proporcionan paralelismo a nivel de hilos para arquitecturas con memoria compartida.**
 - Intel MKL
 - OpenBLAS
 - ATLAS, etc.

CUBLAS

Bases para CUBLAS

- [2003, Kruger y Westermann] Primeras ideas tendientes sobre implementación de BLAS en GPU.

Con el surgimiento de CUDA

- Se incluye CUBLAS
 - Implementación en GPU de BLAS.
 - Comienza en simple precisión, luego evoluciona ...

CUBLAS

Mejoras a CUBLAS

- [2008, Volkov y Demmel] Importantes mejoras en la operación gemm (multiplicación de matrices generales). Luego incorporado a CUBLAS.
- [2008, Barrachina y otros] Uso de padding y estrategias híbridas (CPU+GPU).

CUBLAS

CUBLAS v2

- La forma de acceder incluye un handler.
- Se puede acceder a los escalares por referencia tanto en GPU como en CPU.
- Se han agregado funciones multi-GPU.
- Se han agregado funciones batch.

NVBLAS

- Ejecución dinámica de operaciones BLAS-3 en más de una GPU (y CPU).

CuSparse

En las últimas versiones incluye funciones de alto nivel:

Ofrece spmv y sptrsv en distintos formatos de matriz dispersa, funciones de conversión.

Para preconditionamiento

- IC, ILU, etc.

Para reordenamiento

- Coloreado de grafos

cuSolver

Se incluyen funciones para resolver operaciones de álgebra densa y dispersa:

- LU, Cholesky
- SVD
- eig
- QR

CUDA Profiling Tools Interface (CUPTI)

- **Permite crear herramientas de profiling y tracing sobre CUDA.**
- **Provee varias APIs:**
 - » **Activity API**
 - » **Callback API**
 - » **Event API**
 - » **Metric API**

Activity API

Permite guardar en forma asincrónica la traza de una aplicación.

Elementos:

- **Activity record:** estructuras de C donde se reportan las actividades.
- **Activity Buffer:** para almacenar los activity records.
- **Activity Queue:** cola con los activity buffers (global, context, stream).

Callback API

Permite guardar un callback en el código.

**El callback es invocado cuando una aplicación llama al
CUDA runtime, a una función del driver o ciertos
eventos del CUDA driver.**

CUPTI

- **Event API**

Permite consultar, configurar, lanzar, parar y consultar el event counter de un device.

- **Metric API**

Permite consolidar métricas de aplicaciones calculadas por uno o varios valores de evento.

CURAND

Biblioteca para la generación de números pseudo-aleatorios y quasi-aleatorios.

- **Una secuencia pseudo-aleatoria satisface la mayoría de las propiedades estadísticas pero es generada por un algoritmo determinístico.**
- **Una secuencia quasi-aleatoria de puntos n-dimensional es generada por un algoritmo determinístico diseñado para completar un espacio posiblemente n-dimensional.**

CURAND

CURAND incluye dos componentes:

- **Una API en el host (CPU).**
- **Una API (archivo cabecal) para el device (GPU).**

CURAND

Biblioteca

Los números aleatorios pueden ser generados en el device o en el host.

- **Cuando se generan en el device: la llamada a la biblioteca se realiza en el host (pero se ejecuta en el device). Los números generados se guardan en memoria global del device.**
- **Cuando se genera en CPU: el trabajo se realiza en CPU y los números se almacenan en memoria del host.**

CURAND

Archivo cabezal

- **Define las funciones en el device para inicializar los estados del generador de números aleatorios y generar las secuencias de números aleatorios.**
- **Utilizando las funciones directamente, se puede evitar escribir en memoria (consumiendo al mismo tiempo que se van generando los números).**

CURAND

Resumen del funcionamiento:

- Los números aleatorios son calculados por los generadores.
- Los generadores encapsulan el estado interno para seguir la secuencia.
- Se pueden crear varios generadores al mismo tiempo.
- La secuencia producida por cada generador es determinística.
Dada los mismos parámetros iniciales se genera la misma secuencia !!

CURAND

Secuencia normal de uso:

1. Crear un nuevo generador con `curandCreateGenerator()`.
2. Especificar las opciones del generador.
3. Allocar memoria en el device con `cudaMalloc()`.
4. Generar números aleatorios con `curandGenerate()` (u otra función generadora).
5. Usar los números ...
6. Si es necesario, generar más números llamar nuevamente a `curandGenerate()`.
7. Finalizar con `curandDestroyGenerator()`.

CURAND

- **Para generar números aleatorios en el host:**
 - En el paso 1, llamar a `curandCreateGeneratorHost()`,
 - En el paso 3, alocar memoria en el host para recibir los números.
- **Notar que `curandGenerate()` en el paso 4 , lanza el kernel y retorna en forma asincrónica.**
 - Si se lanza otro kernel en otro stream, y se necesitan los resultados de `curandGenerate()`, se necesita llamar a `cudaThreadSynchronize()` o sincronizar los streams, para garantizar que el generador finalizó antes de que el kernel sea lanzado.
- **Obviamente no es válido pasar un puntero a memoria del host cuando se trabaja en el device y viceversa.**

NPP

NVIDIA Performance Primitives (NPP)

- **NPP es una biblioteca de funciones para acelerar en forma sencilla aplicaciones con GPUs.**
- **Las funciones incluidas se centran en procesamiento de imágenes y video.**
- **Más de 5000 primitivas**

OpenACC

OpenACC

- **Paralelismo sencillo.**
- **Paralelización mediante cláusulas.**
- **Extensión del estilo OpenMP.**
- **Incluye las transferencias**