Computación de Altas Prestaciones
Seminario sobre GPGPUs
Sesión 5

Contenidos

Sesión 1 Teoría: Introducción a Computación en GPUs con CUDA.

Sesión 2:Compilación, conceptos básicos

Sesión 3: Programación de algoritmos “trivialmente paralelos”

Sesión 4: Uso de la memoria “Shared”. “Reducciones” en GPUs

Sesión 5: Optimización, temas avanzados, librerías

Ejercicios reducción

- 2) Haz un programa para calcular en la GPU las medias de las columnas de una matriz usando reducción. Cada bloque calcula la media de una columna, haciendo colaborar todos los threads del bloque. Hazlo a partir del archivo en poliformat media_matriz_incompleto.c

- 3) Haz un programa que calcule el producto Matriz-Vector. Tenéis en poliformat un archivo, matrix_vector.cu, que hace el cálculo en CPU y tiene programado todos los envíos para hacerlo en GPU .

Múltiples GPUs

Existen varias posibilidades para usar múltiples GPUs en un programa. Este código permite enumerarlas

```
int deviceCount;
cudaGetDeviceCount(&deviceCount);
int device;
for (device = 0; device < deviceCount; ++device)
   {     cudaDeviceProp deviceProp;
         cudaGetDeviceProperties(&deviceProp, device);
         printf("Device %d has compute capability %d.%d.\n",
device, deviceProp.major, deviceProp.minor);
   }
```

Múltiples GPUs

Se puede seleccionar la GPU activa con cudaSetDevice(). Si no se hace esta llamada, el “device” por defecto es el 0. Ejemplo con dos GPUs

```
size_t size = 1024 * sizeof(float);
cudaSetDevice(0); // Set device 0 as current
float* p0;
cudaMalloc(&p0, size); // Allocate memory on device 0
MyKernel<<<1000, 128>>>(p0); // Launch kernel on device 0
cudaSetDevice(1); // Set device 1 as current
float* p1;
cudaMalloc(&p1, size); // Allocate memory on device 1
MyKernel<<<1000, 128>>>(p1); // Launch kernel on device 1
```

Múltiples GPUs

La forma mas sencilla es usar varias GPUs es usar varios threads de la CPU, cada uno controlando una GPU: OpenMP

```
int i,deviceCount, N=1000;
cudaGetDeviceCount(&deviceCount);
#pragma omp parallel for
for (i=0;i<N;i++)
  { igpu=i/deviceCount;
    cudaSetDevice(igpu) ;
    cudaMalloc(&p0, size);
     MyKernel<<<1000, 128>>>(p0);
    cudaFree(p0);
  }
```

CUDA Streams

Un stream de CUDA es una secuencia de operaciones que debe ejecutarse en la GPU en un cierto orden.

Las operaciones que hemos visto se realizan en un stream: El stream por defecto.

Usando varios streams, se puede solapar cálculos con envíos de datos, (CudaMemcpyAsync())o se pueden usar para manejar varias GPUs sin usar varios threads de la CPU

CUDA Optimización

Existen muchos detalles (demasiados hoy en día) que se deben tener en cuenta para escribir código “rápido” en una GPU:

-Alto grado de paralelismo.

-Acceso “coalesced” a memoria global.

-Uso de memoria “page-locked” o “pinned”

-Minimizar sincronizaciones

...

En muchos casos puede ser mejor

-usar librerías

-usar la GPU con alguna interfaz apropiada..(MATLAB + PCT)

CUBLAS+ Magma

CUBLAS es el equivalente de BLAS para CUDA+GPUs; tiene implementada mas o menos prácticamente las mismas operaciones que BLAS.

Magma es equivalente a LAPACK; sin embargo, no es puramente para GPU.

Magma está pensado para usar la máquina en modo “híbrido”, parte en la CPU y parte en la GPU.

Cada parte del código se ejecuta en el dispositivo que se adecúa mejor.

CUBLAS+ Magma

Documentación CUBLAS:

http://docs.nvidia.com/cuda/cublas/

Documento cublas_library.pdf en poliformat

Ejemplos de uso de CUBLAS:

Documento ejemplos_CUBLAS.pdf en poliformat

CUBLAS

APIs:

- -CUBLAS : incluir “cublas_v2.h”. “the applicationmust allocate the required matrices and vectors in the GPU memory space, fill them with data, call the sequence of desired cuBLAS functions, and then upload the results from the GPU memory space back to the host.The cuBLAS API also provides helper functions for writing and retrieving data from the GPU.”

-CUBLAS legacy: incluir “cublas.h”

- -CuBLASXT: incluir “cublasxt.h” “the application must keep the data on the Host and the Library will take care of dispatching the operation to one or multiple GPUS present in the system, depending on the user request.”

CUBLAS

Ejercicio: En poliformat puedes encontrar el archivo ejemplo_matvec_cublas.cu, que realiza el producto matriz por vector de números en precisión simple(floats) en la cpu y en la gpu, con una llamada a la subrutina de cublas cublasSgemv.

Tienes también un Makefile para simplificar la compilación.

También puedes encontrar el archivo mat_mat_cublas_incompleto.cu, que tiene el producto matriz por matriz en cpu.

El ejercicio consiste en completar este archivo para que haga el producto matriz por matriz usando una llamada a la subrutina de cublas : cublas_Sgemm.