Proyecto Final

Análisis Comparativo de Técnicas de Paralelización de la Multiplicación de Matrices Usando CUDA

Camilo Andrés Rivera Lozano

Escuela de Graduados en Ingeniería y Arquitectura
Campus Guadalajara

3 de julio de 2015

Agenda

Tabla de contenidos

Introducción

High Performance Computing

- Alcanzar metas antes no pensadas
- Resolver problemas cuya resolución se creía imposible
- Cientos de procesadores unidos y trabajando en equipo por una misma causa

GPU

- Inicialmente co-procesadores para el trabajo sobre gráficos
- Dada su arquitectura y su alto poder de cómputo se prestaron para el procesamiento paralelo de alto poder de procesamiento
- Lenguaje CUDA basado en C para tarjetas gráficas de NVIDIA

Tabla de contenidos

Objetivos

- Encontrar diferentes mejoras ante diferentes algoritmos paralelos de multiplicación de matrices respecto al método secuencial
- Utilizar la memoria compartida y observar mejoras en los tiempos
- Generar una serie de códigos que se ejecuten en una GPU y sean funcionales para el propósito establecido

Tabla de contenidos

Justificación

La multiplicación de matrices es un paso casi inevitable en problemas multidimensionales, operaciones lineales y sistemas de ecuaciones.

El tamaño de elementos crece de manera cuadrática con la dimensión y se vuelve necesario acelerar el proceso de multiplicación.

Aplicaciones en todas las ciencias, algoritmos de optimización, solución de sistemas de ecuaciones, aprendizaje de máquinas, etc.

Tabla de contenidos

Multiplicación de Matrices

¡Se tiene que aprovechar la estructura matricial de la GPU!

Forma Tradicional

$$C_{ij} = \sum_k A_{ik} B_{kj}$$

A Bloques

$$\begin{pmatrix} A_{UL} & A_{UR} \\ A_{LL} & A_{LR} \end{pmatrix} \cdot \begin{pmatrix} B_{UL} & B_{UR} \\ B_{LL} & B_{LR} \end{pmatrix} = \begin{pmatrix} C_{UL} & C_{UR} \\ C_{LL} & C_{LR} \end{pmatrix}$$

$$C = \begin{pmatrix} A_{UL}B_{UL} + A_{UR}B_{LL} & A_{UL}B_{UR} + A_{UR}B_{LR} \\ A_{LL}B_{UL} + A_{LR}B_{LL} & A_{LL}B_{UR} + A_{LR}B_{LR} \end{pmatrix}$$

Tabla de contenidos

Mayor Cantidad de Variables

```
__global__ void matrixMultGPU(int *a_ll,int *a_lr,int *a_ul, int *a_ur,int *b_ll,int
     *b_lr,int *b_ul, int *b_ur, int *c_ll,int *c_lr,int *c_ul, int *c_ur, int *t_ll,int
     *t_lr,int *t_ul, int *t_ur,int N){

  int k, sum_cur = 0,sum_cul = 0,sum_cll = 0,sum_clr = 0,sum_tur = 0,sum_tul = 0,sum_tll
       = 0,sum_tlr = 0;
  int col = threadIdx.x + blockDim.x * blockIdx.x;
  int fil = threadIdx.y + blockDim.y * blockIdx.y;
  if (col < N && fil < N)
  {
    for (k = 0; k < N; k++)
    {
      sum_cul += a_ul[fil * N + k] * b_ul[k * N + col];
      sum_cur += a_ul[fil * N + k] * b_ur[k * N + col];
      sum_cll += a_ll[fil * N + k] * b_ul[k * N + col];
      sum_clr += a_ll[fil * N + k] * b_ur[k * N + col];

      sum_tul += a_ur[fil * N + k] * b_ll[k * N + col];
      sum_tur += a_ur[fil * N + k] * b_lr[k * N + col];
      sum_tll += a_lr[fil * N + k] * b_ll[k * N + col];
      sum_tlr += a_lr[fil * N + k] * b_lr[k * N + col];
    }
```

Mayor Cantidad de Variables

```
    c_ul[fil * N + col] = sum_cul;
    c_ur[fil * N + col] = sum_cur;
    c_ll[fil * N + col] = sum_cll;
    c_lr[fil * N + col] = sum_clr;

    t_ul[fil * N + col] = sum_tul;
    t_ll[fil * N + col] = sum_tll;
    t_lr[fil * N + col] = sum_tlr;
    t_ur[fil * N + col] = sum_tur;
    __syncthreads();

    c_ul[fil * N + col]+=t_ul[fil * N + col];
    c_ll[fil * N + col]+=t_ll[fil * N + col];
    c_lr[fil * N + col]+=t_lr[fil * N + col];
    c_ur[fil * N + col]+=t_ur[fil * N + col];
  }
}
```

- ¡Otra opción es lanzar 8 veces el kernel con dos submatrices a la vez ahorrando espacio en GPU!

Tiempos de Ejecución

Tamaño	Secuencial	Par 1	Par 2	Par 3	Compartida
100	4.09 ms	0.201ms	0.218 ms	0.105 ms	–
200	38.28 ms	2.12 ms	0.182 ms	0.78ms	–
500	979.22 ms	32.28 ms	3.52 ms	10.6 ms	–
31	–	14.3 us	14.9 us	11.6 us[1]	6.38 us

Cuadro : Tabla de tiempos

El último método pudo manejar matrices de 1000x1000 cuyo tiempo de multiplicación fue de 116ms

[1] Para el caso de la técnica de división de matrices se utilizó un tamaño de 32

Tabla de contenidos

Conclusiones

- Se observa una gran mejoría en el rendimiento de los algoritmos, para el caso de las matrices 500x500 se observa incluso 278 veces mejor tiempo que para el caso secuencial.
- Se observa una mejoría ante el uso de la memoria compartida (x2)
- Utilización de cudaMemcpu2D

Tabla de contenidos

Bibliografía

[1] Sanketh Datla and Naga Sathish Gidijala. Parallelizing motion JPEG 2000 with CUDA. In *2009 Second International Conference on Computer and Electrical Engineering*. IEEE, 2009.

[2] NVIDIA Corporation. Cuda toolkit documentation - v7.0. [online] Disponible: `http://docs.nvidia.com/cuda`, 3 2015.

Proyecto Final

Análisis Comparativo de Técnicas de Paralelización de la Multiplicación de Matrices Usando CUDA

Camilo Andrés Rivera Lozano

Escuela de Graduados en Ingeniería y Arquitectura
Campus Guadalajara

3 de julio de 2015