Práctico 1 del curso GPGPU:
Impacto del acceso a los datos en CPU

Introducción

Como se vio en la Clase 2 Computación Paralela, en el desempeño computacional de un algoritmo implementado en CPU, además de la complejidad propia del algoritmo, existe otro factor que tiene una gran incidencia en el tiempo de ejecución, el acceso a los datos. Este aspecto resultará crucial en las implementaciones en GPU.

En este práctico veremos experimentalmente el efecto del acceso a los datos en varios algoritmos implementados en CPU.

Ejercicio 1

El ejercicio 1 consiste en analizar los resultados de un programa diseñado para evaluar la velocidad de acceso a los distintos niveles de la memoria. El programa recorre un arreglo lineal de distintos tamaños, con distintos tamaños de paso (distinto *stride*) y devuelve la tasa de accesos por unidad de tiempo medida en GB/s. Los parámetros del programa son el tamaño máximo del arreglo a recorrer en MB y el paso máximo (en cantidad de elementos).

El funcionamiento del programa es el siguiente:

1. Crea un arreglo lineal de enteros del tamaño máximo l_{max} recibido por parámentro.
2. Para cada tamaño l de arreglo en KBytes que sea una potencia de 2, comenzando por 16KB:
   a) Para cada *stride* s entre 1 y 16:
      1) Suma l/`sizeof(int)` elementos del arreglo salteando de a s elementos.
      2) Toma el tiempo de ejecución $t_{l,s}$ de la recorrida, guardando en la posición correspondiente de una matriz (tabla) el valor de $l/t_{l,s}$.
3. Una vez obtenida la tabla de datos, la imprime en pantalla para poder guardarla en un archivo de texto. En `script_octave.m` se incluye código para generar esta gráfica con los programas Octave o Matlab.

Analice la forma de la gráfica y explique los resultados.

1. ¿Qué puede decir sobre la forma de la gráfica en el eje correspondiente al tamaño del arreglo recorrido?
2. ¿Qué puede decir sobre la forma de la gráfica en el eje correspondiente al tamaño del paso?
3. ¿Puede sacar alguna conclusión sobre el tamaño y cantidad de niveles de la caché en base al análisis de la gráfica?

Ejercicio 2

parte a)

La multiplicación de matrices es una de las operaciones básicas de Álgebra Lineal Numérica más utilizadas. Si $A \in \mathbb{R}^{m \times p}$ y $B \in \mathbb{R}^{p \times n}$, cada elemento c_{ij} de $C = A \times B$ puede computarse como el producto escalar de la fila i de A con la columna j de B según la expresión

$$c_{ij} = \sum_{k=1}^{p} a_{ik} \times b_{kj}. \tag{1}$$

La expresión anterior da lugar a la versión conocida como ijk del producto matricial. Como los productos y actualizaciones de la matriz pueden realizarse en cualquier orden, existen otras 5 posibilidades, resultantes de intercambiar el orden de los 3 loops (*jik, ikj, kij, jki, kji*).

1. Analice el patrón de acceso a la memoria de las 6 versiones del producto matricial que se proporcionan, intentando deducir de forma teórica la tasa de cache miss en cada implementación.

2. Mida la tasa de caché miss de las versiones *ijk* e *ikj* utilizando una herramienta como *perf* o *valgrind* y contraste con el resultado teórico.

3. Ejecute el código para matrices cuadradas de distintos tamaños y analice los resultados de tiempo de ejecución, contrastándolos con el análisis teórico. Construya una gráfica de tiempo de ejecución variando el tamaño de las matrices entre 128 y 1280.

parte b)

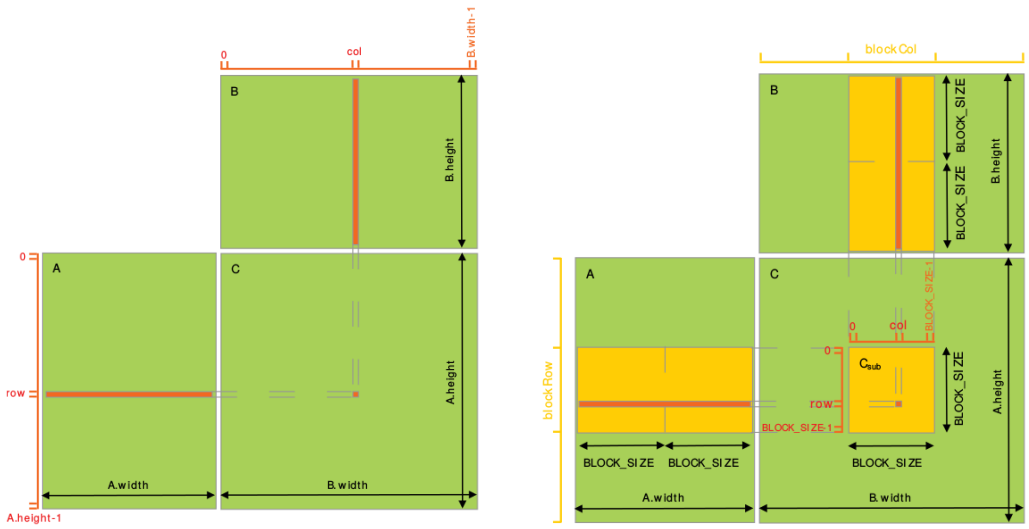

Figura 1: Multiplicación de matrices lineal y por bloques

Para maximizar la reutilización de los datos alojados en la caché, podemos dividir las matrices en *tiles* o bloques de tamaño $nb \times nb$ para cierto nb y realizar la multiplicación "por bloques". La multiplicación por bloques consiste en tomar todos los pares posibles de bloques (uno tomado de la matriz A y el otro de B), y multiplicarlos acumulando el resultado parcial en el lugar de la matriz C que corresponda. El proceso se describe en la Figura 1.

1. Ejecute las versiones a bloques para los mismos tamaños que los de la parte anterior y tamaños de bloque de entre 8×8 y 128×128.

2. Analice los resultados de las distintas versiones a bloques y compare los tiempos de ejecución con las variantes lineales correspondientes.

3. Relacione el desempeño observado de las distintas versiones (lineales y a bloques) para distintos tamaños con los tamaños de los distintos niveles de caché. Se recomienda obtener el desempeño en GFlops.

4. ¿Qué mejoras podrían ensayarse para los métodos a bloques proporcionados?

Consideraciones generales

1. En el ejercicio 1 compile el código utilizando la flag `-O0` para desactivar optimizaciones del compilador (como vectorización y desarrollo de loops) con el objetivo de que los resultados sean más fáciles de interpretar. En el ejercicio 2 puede ser conveniente compilar con las flags `-O3 -march=native -mtune=native -funroll-loops`, para habilitar las optimizaciones del compilador y el set de instrucciones de la plataforma de ejecución, de forma de obtener un mejor desempeño y poder experimentar con matrices de mayor tamaño en menos tiempo.

2. Si utiliza **perf** debe agregar también las flags `-pg`.

3. Algunas funciones matemáticas requieren compilar con la flag `-lm`.

4. Se estimula complementar el análisis en cualquiera de los dos ejercicios mediante la propuesta de otros experimentos, parámetros, gráficas, etc.

Entregar

1. Un informe que contenga el análisis de los experimentos descritos en la letra, la respuesta a las distintas preguntas, así como el planteo de otros experimentos, incluyendo una discusión sobre los resultados obtenidos.

2. Código fuente, acompañado por Makefile, en caso de que haya hecho modificaciones al código de ejemplo o haya experimentado con otras implementaciones.