Trabajo Práctico 1: Optimización secuencial

Computación paralela

Marzorati Denise
M-6219/7

10 de mayo de 2018

Docentes de la materia

Nicolás Wolovick
Carlos Bederián

Características del hardware y del software

- CPU: Intel Core i7-3632QM @ 2.20GHz.
- Memoria: 8GB, DDR3, 2 canales.
- L1 cache: 256KB
- L2 cache: 1MB
- L3 cache: 3MB
- Compiladores: GCC 7.1.0, Clang 5.0, PGI, AOCC (compilador de AMD).
- Sistema operativo: Ubuntu 16.04.4, x86_64.

IntegralImage

Para obtener la versión más rápida posible se ha compilado el código con:

clang-5.0 integralimage.c -o integral -O2 -march=ivybridge -funroll-loops -Wall -Wextra -std=c99 -lm -lgomp.

Una de las primeras cosas que se hizo fue la de probar usando distintos compiladores. Clang fue con el que se obtuvieron los mejores resultados, con GCC y AOCC devolviendo valores ligeramente inferiores a los del mencionado compilador. Por otro lado, PGI tuvo un rendimiento pésimo, puesto que para FRAMES=300 la ejecución del programa tomó alrededor de ciento cincuenta segundos.

Para lograr autovectorización se escribieron varias versiones de la función "integral_image", usando las técnicas mostradas en la clase correspondiente al tema. Es decir, el uso de restrict, se indicó el alineamiento de las variables en memoria, se redujeron los múltiples loops a un único loop. Sin embargo nada de esto logró que el compilador autovectorizara.

Por el motivo antes mencionado se prosiguió intentando vectorizar la función a mano, pero cada uno de los intentos bajó el rendimiento del programa notablemente, así que la opción se descartó.

Como Carlos envió un mail avisando de que había un problema para usar ISPC, ni siquiera se intentó usar el compilador.

En resumen, con respecto al trabajo anterior lo único que dio resultado fue cambiar el compilador, que pasó de ser GCC-7 a Clang-5. Nada fue modificado del código.

Aclaración: sin haber visto el tema en las clases, para el primer trabajo se utilizó una función generadora de números aleatorios que hace uso de SSE4, y está escrita con intrinsics, lo que significó una mejora importante en el rendimiento del programa.

FRAMES = 300

	Práctico 1	Práctico 2
Avg IPC	2.61	2.25
Avg % cache misses	16.16%	14.02%
Avg time	0.991633s	0.8807468s
Avg % stalled cycles, front end	18.61%	21.21%

Table 1: Resultados para FRAMES = 300

FRAMES = 3000

	Práctico 1	Práctico 2
Avg IPC	2.69	2.48
Avg % cache misses	12.02%	11.69%
Avg time	9.420287s	8.802669s
Avg % stalled cycles, front end	17.61%	16.67%

Table 2: Resultados para FRAMES = 3000

FRAMES = 30000

	Práctico 1	Práctico 2
Avg IPC	2.68	2.53
Avg % cache misses	12.61%	9.40%
Avg time	95.767781s	82.109516s
Avg % stalled cycles, front end	17.89%	15.46%

Table 3: Resultados para FRAMES = 30000

Con las optimizaciones realizadas se ha podido observar que ahora el tiempo que el programa tiene de ejecución crece linealmente con respecto al tamaño del mismo, en este caso dado por el valor FRAMES.

No han surgido ideas para potenciales mejoras en la vectorización.