Advanced Computer Architecture: The “Smooth” Challenge

Romain Brault[1] and Alexandre Camus[2] and Giorgos Flourentzos[3]

Abstract
The ABSTRACT is to be in fully-justified italicized text, between two horizontal lines, in one-column format, below the author and affiliation information. Use the word “Abstract” as the title, in 9-point Times, boldface type, left-aligned to the text, initially capitalized. The abstract is to be in 9-point, single-spaced type. The abstract may be up to 3 inches (7.62 cm) long.
Leave one blank line after the abstract, then add the subject categories according to the ACM Classification Index (see http://www.acm.org/class/1998/).

Contents

1. Introduction

1.1 Hardware Considerations

The chosen hardware is not one of the machine from the Lab. They do not have any interesting GPUs. But one of our laptops is very powerful with an interesting GPU. This is the chosen one.

The table 1 contains the details of this hardware.

The GPU specifications are shown in the table 2.

To summarize, the figure 1 gives a quick overview of the hardware topology. The GPU is connected on the PCI port `10de:0618`.

1.2 Software Considerations

The machine used for this coursework runs a Fedora 18 distribution. This is the exact information:

Model Name	Intel Core i7 CPU Q720
Clock Speed	1.597 GHz
Max Turbo Frequency	2.8 GHz
Cache size	6144 KB
CPU cores	4
CPU Threads	8
Integrated GPU	No
Memory Channels	2
Max Memory Bandwith	21 GB/s

Table 1. CPU Specifications

Model Name	NVIDIA GeForce GTX 260M
Clock Speed	1.375 GHz
Multiprocessors	14
Global CUDA cores	112
Allocated Memory	1 GB
Memory Clock Speed	950 MHz

Table 2. GPU Specifications

```
$ uname -a
Linux localhost.localdomain 3.8.1-201.
    fc18.x86_64 #1 SMP Thu Feb 28
    19:23:08 UTC 2013 x86_64 x86_64
    x86_64 GNU/Linux
```

2. The Sequential Issue

After running for the first time the program, it appears that

Reading the code, some sequential issues were found. There are three kinds of problems: the algorithms chosen, the coding style and the data’s representation.

2.1 Algorithm

The method used to solve an equations’ system was correct but two general to be efficient. The program needs only to solve

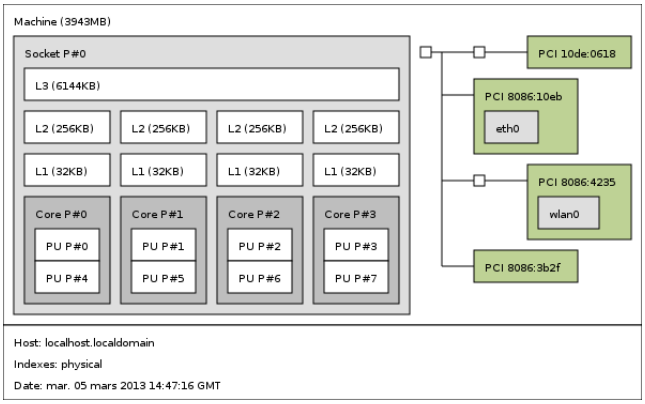

Figure 1. Topology

systems of two equations in two unknowns. The Cramer's rule based on determinants is far more efficient.

The use of the method `pow()` seems a little too heavy in the method `element_quality()`. As the number of multiplication is known, the use of simple multiplications is more efficient here, e.g. `x*x*x`.

2.2 Code Style

Few changes in the style of the code might help the compiler. Some of these changes have been done to speed up the program.

The methods `SVDsolver()` and `cornerNode()` and `isSurfaceNode()` and `element_quality()` are now written as inline functions. This gives a small boost in performance.

In loops changes have been made to avoid recomputation of the invariant (e.g. in `calling_size()`). Instead, this invariant is now stored in a local variable.

2.3 Data's Representation

The data structure used is fine and one of the most efficient to represent a graph. However, the C++ structures used seem to be a little too big in this case. So instead of using vectors of sets, the program is now using vectors of vectors. This increases a little the global performance.

3. CPU Parallelization

3.1 Analysis

Parallelization can't be done easily. The program needs to be slightly modified in order to cut the graph in groups of independent nodes. Independent nodes are nodes that can be inspected at the same time while running the program. To group nodes in such groups the graph must be colored. Then each color represents nodes that can be inspected at the same time because they are not adjacent.

But the coloring algorithm might be very expensive in computation time. This depends on the number of colors used and if this number is specified or not. In this very case, the goal of such an algorithm is to minimize the colors used in order to maximize parallel computations.

3.2 Optimization

4. GPU Acceleration

4.1 Analysis

4.2 Optimization

5. Results

5.1 Sequential improvements

5.2 GPU performance