Team 16 -   Max Falk Nilsson, Per Albin Mattsson, Jakob Sennerby

LAB 2

Contributions

Max has written most of the code. Per Albin and Jakob have spent much time on trying to figure out how to use OpenCL and CUDA but has not made large contributions to the actual program in this assignment.

How to run the program

To chose between sequential, pthread and OpenMP mode, use the “-mode” flag followed by a string for the three different alternatives:

SEQ: Sequential mode

PTHREAD: Pthread mode

OMP: OpenMP mode

VECTOR: Vector mode

CUDA: Cuda mode

To chose the number of threads use the “-threads” flag followed by the number of threads.

Example:

“./demo -mode OMP -threads 4”

The example above runs OpenMP with 4 threads.

Example:

“./demo -mode VECTOR”

The example above runs vectorization.

Computer used for tests

CPU: AMD Phenom(tm) 8450 Triple-Core Processor
Memory: 3GB of DIMM DDR2 1639 MHz (0,6 ns)
Graphics card: GF114 [GeForce GTX 560 Ti]

Questions

A. What kind of parallelism is exposed with your code refactorization?

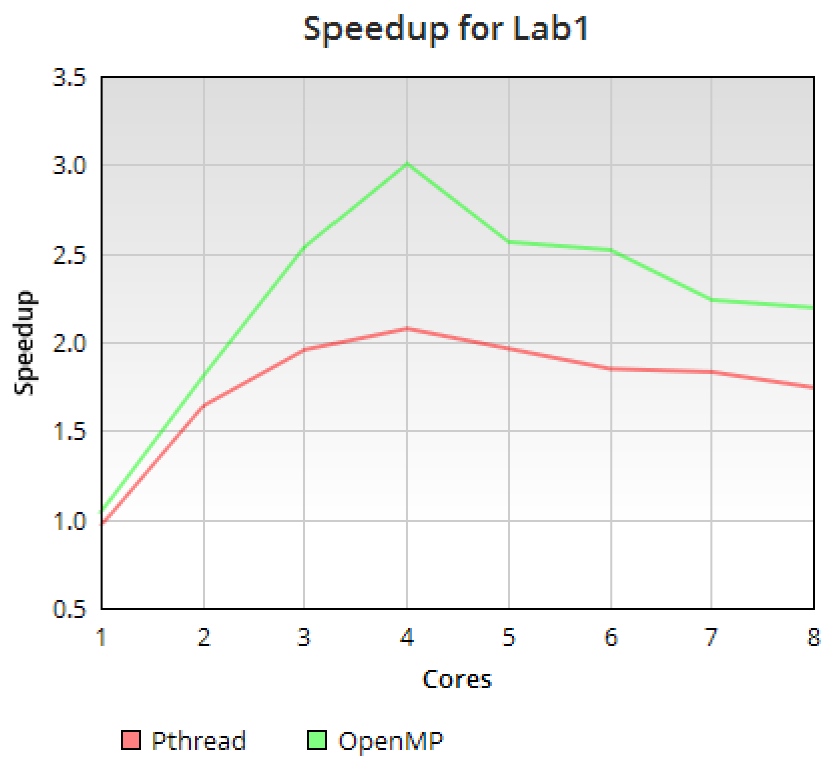
The refactorization made the program more suited for data parallelism. Since, for example, SIMD executes the same instruction to different data in parallel. We needed to make the code more suited for these kind of instructions.

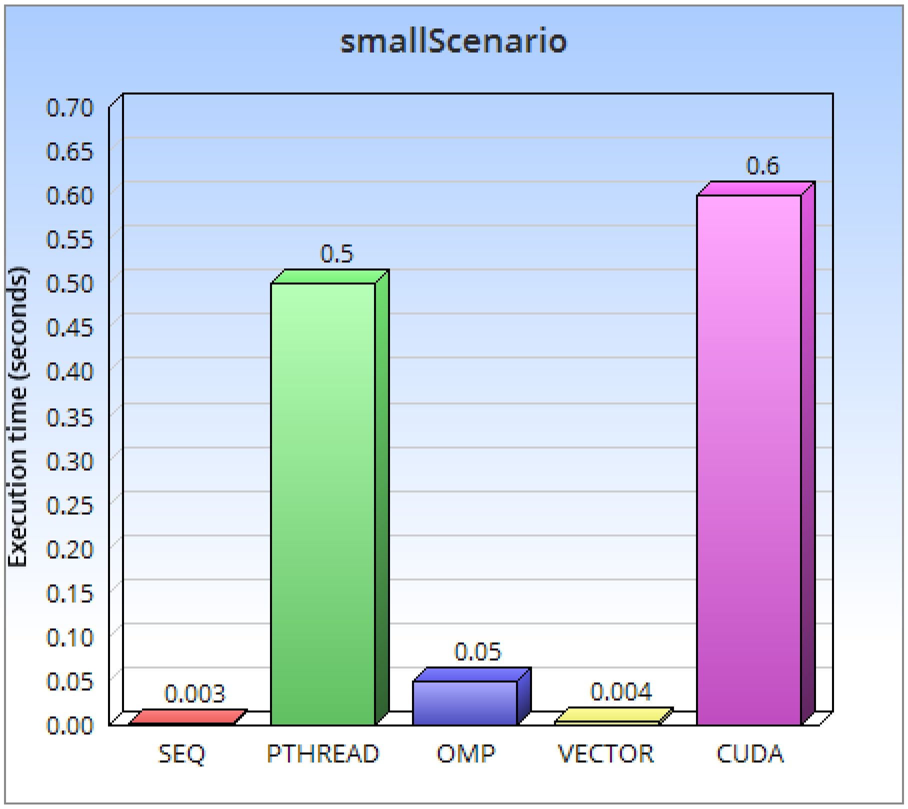
B. Give a short summary of similarities and differences between SIMD and CUDA.

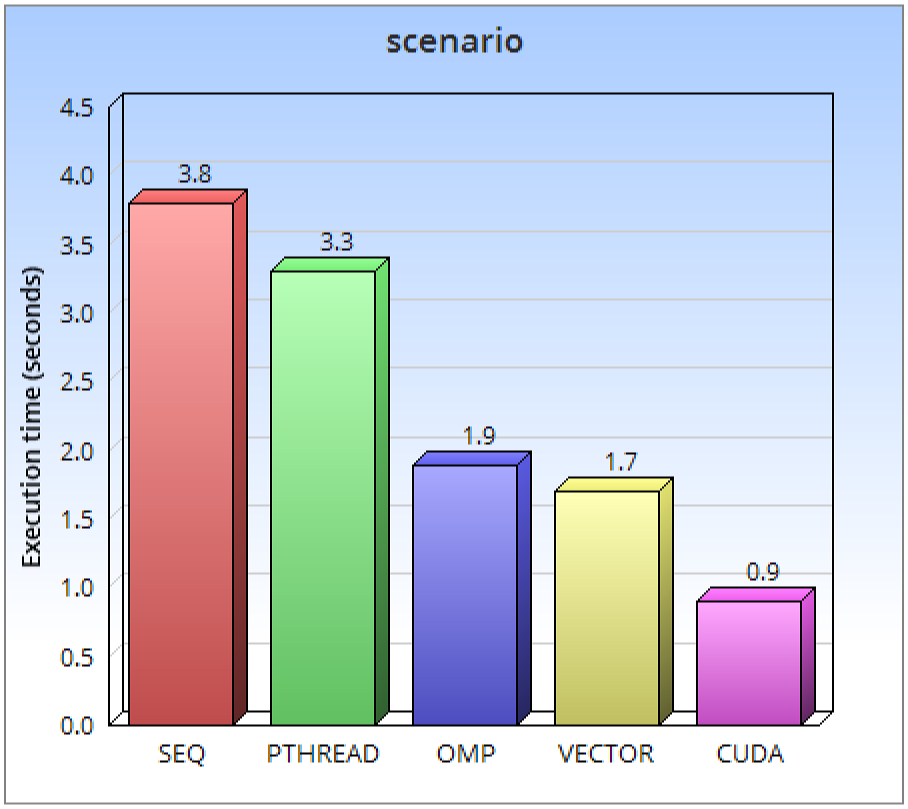
Both techniques are, when in seperate use, suited for data parallel programs when high throughput is desired. The difference is that CUDA is a programming platform that gives developers the possibility to run SIMD instructions on the GPU. This makes it possible to use the SIMT execution model.

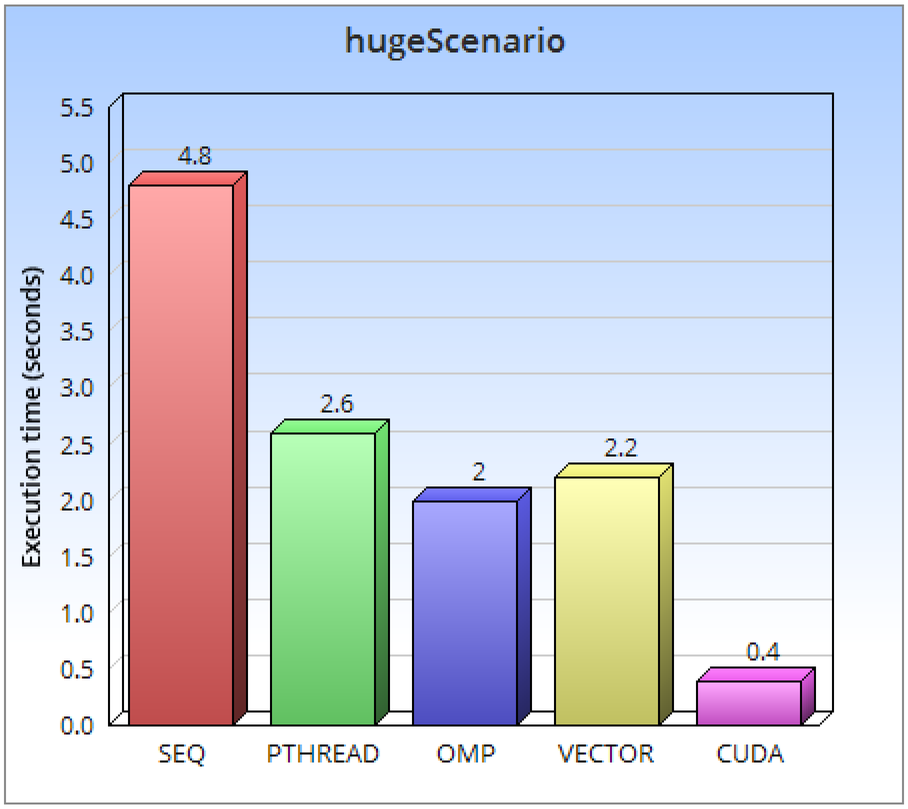
C. Which parallelization has been most beneficial in these two labs? Support your case using plotted timing measurements.

As we can see below, both from lab 1 and lab 2, OpenMP is very beneficial. Although, vectorization and CUDA is a bit faster when running the larger scenarios. We can also see that CUDA is slow when operating on small problems as there are more overhead with beginning the computation. This is also true for pthreads though there is not as much. For the huge scenario we changed the number of ticks to 1000 and agents to 2*24000.


Speedup for Lab1
Speedup
3.5
3.0
2.5
2.0
1.5
1.0
0.5
1
2
3
4
5
6
7
8
Cores
Pthread
OpenMP


smallScenario
Execution time (seconds)
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.003
0.5
0.05
0.004
0.6
SEQ
PTHREAD
OMP
VECTOR
CUDA


scenario
Execution time (seconds)
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
3.8
3.3
1.9
1.7
0.9
SEQ
PTHREAD
OMP
VECTOR
CUDA



D. What kind of code transformations were required for this lab, and why were they needed?

We needed to put most of the data in aligned vectors to make better use of the low level instructions that can do computations in parallel. Thus some of the object oriented programming paradigm used in the given code needed to be dissolved.

E. Compare the effort required for each of the four parallelization techniques. Would it have made a difference if you had to build the program from scratch, instead of using given code?

OpenMP is the easiest to implement. Pthread has been, up until now at least, fairly easy to implement as well. The other methods need more complicated programming techniques. If the program were to be built from scratch, vectorization and GPU parallelization probably would have been easier to implement. We then could have made the code more suitable for these techniques from the start instead of translating already existing code that followed an undesirable programming paradigm. Cuda had a very confusing and hard installation process that created many problems for us. The installation process was very different depending on hardware and even though you followed the official installation instructions it did not work as explained.Debugging in CUDA was also very hard as you couldn’t use gdb-cuda if you were using the same graphics card to power your monitor.