UNIVERSITY OF AMSTERDAM
Informatics Institute
Systems and Networking Lab
Parallel Computing Systems Group

Dr. Ana Varbanescu
Dr. Clemens Grelck
Julius Roeder, MSc
Misha Mesarcik, MSc
February 8, 2021

Programming Multi-core and Many-core Systems

— Assignment 2 —

Assignment 2.1: Parallelisation of heat dissipation code with OpenMP

Parallelise your program for the simulation of heat dissipation on the surface of a cylinder using OpenMP. You may assume a maximum number of threads to be fixed as a (symbolic) compile time constant.

There is a wide range of design choices in the parallelisation of your code and we encourage you to experiment with them. Again, we urge you to strive for "fast" code that aims at keeping the overhead inflicted by multithreaded execution small. OpenMP makes it particularly simple to experiment with different strategies; make use of it.

Assignment 2.2: Experimentation with OpenMP heat dissipation code

Run your OpenMP multithreaded heat dissipation code (at least) for the following grid sizes: 100×100, 1000×1000, 100×20000, 20000×100 and 5000×5000. Report speedup graphs for different numbers of processors/cores and use

a) your sequential code,

b) your multithreaded code running with only one thread

as two independent baselines. In addition show absolute execution times and FLOP/s for varying numbers of threads up to the number of cores available. What is the shortest execution time that you achieve, and what is the best FLOP/s rate? What happens if you use more threads than the machine has cores?

Assignment 2.3: Task parallelism

MergeSort is a popular and fairly easy to parallelise sorting algorithm, see wikipedia or the like if you do not recall the details from previous courses. Start out with the top-down implementation found at

http://en.wikipedia.org/wiki/Merge_sort

and parallelise it using OpenMP task parallel constructs. Optimise your code for high performance and good speedups over the original sequential solution from wikipedia.

Submit your code in the sub-directory `omp` of your heat dissipation source tree. Your source could shall be organised into two files, one named `sort.c` that your sorting code exporting a function

```
void mergesort(A[], n)
```

and the other containing the `main` function with appropriate initialisation code for the arguments and experimentation in general. The `make` command should compile your code into an executable with the name `mergesort`.

Assignment 2.4: Experimentation with task parallelism

Experiment with to be sorted vectors of numbers of different length starting at 10^4 and advancing in steps of orders of magnitude until execution time becomes infeasible. Fill argument vectors with

a) numbers in ascending order, i.e. already sorted;

b) numbers in descending order, i.e. sorted the wrong way around;

c) random numbers.

Determine the execution time of the relevant program parts, i.e. the pure sorting step without initialisation of the to be sorted vector or possibly output of the sorted vector, using a high precision wall clock timer.

Assignment 2.5: Combining loop and task parallelism

Based on your solution for Assignment 2.3, write a parallel OpenMP program that sorts a vector of vectors of numbers of different lengths, i.e. each inner vector shall be sorted in ascending order.

What does that mean for the scheduling of the outer data parallel operation? Does it pay off to combine data and task parallelism for overall performance? Or, is it better to use either or?

Please, submit your code in the sub-directory `omp` of your heat dissipation source tree. The `make` command should compile your code into an executable with the name `vecsort`. Follow the guidelines given under Assignment 2.3 and name the sorting function `vecsort`.

Submission due date: Tuesday, February 23, 2021