University of Amsterdam
BSc Informatica

Concurrency and Parallel Programming

Assignment 1: OpenMP and pthreads

Parallel Programming for Shared Memory Systems

November 2020

Assignment 1.1: Wave equation simulation with ptThreads

Consider the following 1-dimensional wave equation:

$$A_{i,t+1} = 2 \times A_{i,t} - A_{i,t-1} + c \times (A_{i-1,t} - (2 \times A_{i,t} - A_{i+1,t}))$$

The wave equation describes the movement of a wave in a time- and space-discretized way. Space discretization here means that we represent the wave amplitude not as a continuous function, but as a vector of values. Time discretization means that we simulate continuous motion as a sequence of equidistant time steps. Thus, the above formula defines the wave's amplitude $A_{i,t+1}$ at location i and time step $t + 1$ as a recurrence relation of the current (t) and previous $(t - 1)$ amplitude at the same location (i) and the current amplitude at the neighbouring locations to the left $(i - 1)$ and to the right $(i + 1)$. The amplitude at locations with the least and the greatest index shall be fixed to zero, as illustrated below. In the above wave equation c is a constant that defines the spatial impact, i.e. the weight of the left and right neighbours' values on the new value. Its concrete value is irrelevant from the perspective of parallel program organization; let us work with 0.15.

Write a C program that uses multiple threads to simulate the above wave equation in parallel. The program shall be parameterized over the number of discrete amplitude points and the number of discrete time steps to be simulated, as well as the number of parallel threads to be used. Use three equally sized buffers to store the three generations of the wave needed simultaneously. Rotate the buffers after each time step.

Two initial generations of the wave shall be read in from file before starting the simulation, and the final wave shall be written to file after completing the simulation. Parallelize your program by creating the given number of additional threads and let all threads collaboratively simulate each time step. Divide the work appropriately among the threads. A high resolution timer shall be used to measure the time needed for simulation, which does not include the setup and file input/output times. Print the measured execution time and the normalized time (i.e., the measured time divided by the number of amplitude points and time steps simulated), to the standard output stream.

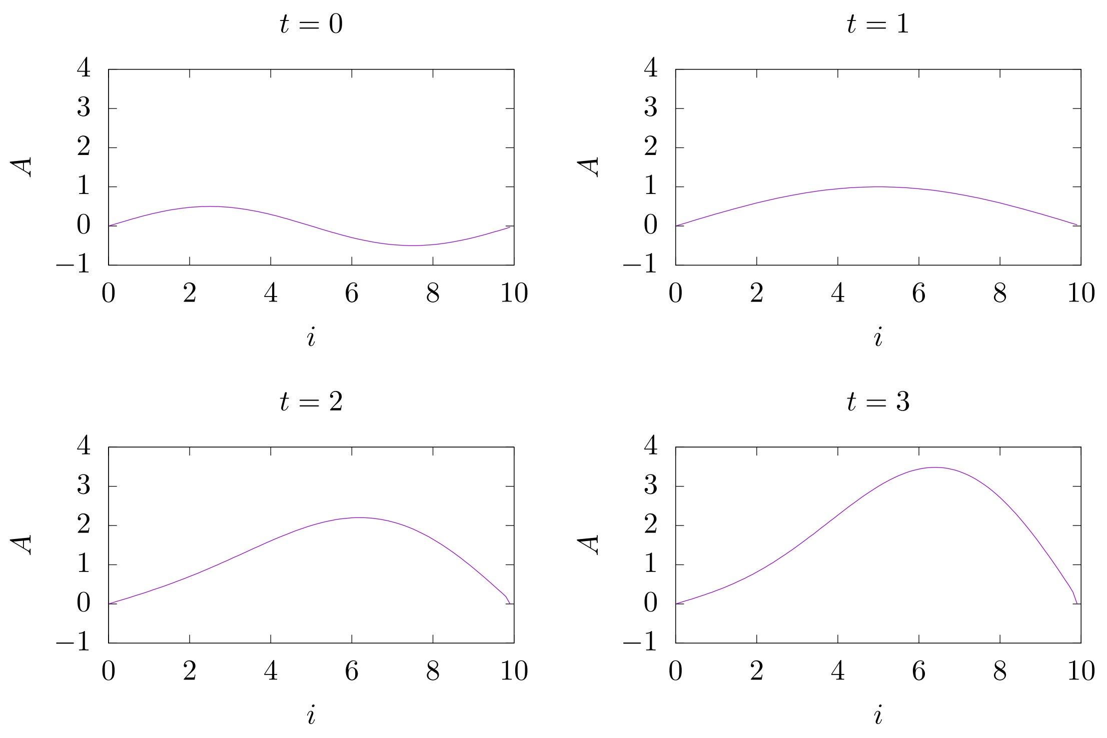

Figure 1: Example execution of the wave equation simulation. The first two graphs use fixed input data while the other two are calculated using previous timesteps.

On Canvas we provide an implementation framework that takes care of parameter interpretation and timing; the framework also includes the reading and writing of wave data from/to files. The use of this framework is mandatory: your code must be embedded in the framework.

Run experiments with different problem sizes, i.e. number of amplitude points being 10^3, 10^4, 10^5, 10^6, 10^7 . Adjust the number of time steps to yield a simulation time that allows for reliable timing of parallel execution without excessively waiting for results, i.e., roughly between 10 and 100 seconds. Report your results as speedup graphs: sequential execution time divided by parallel execution time using 1, 2, 4, 6, 8, 12, 14, and 16 threads on DAS-5. Optionally, you can also report speed-up for execution on your own machine or another development platform, potentially commenting on the difference in performance.

As expected, besides the code itself, you have to also document (in your report) how you solved the problem - your design and your implementation - and the empirical analysis of your solution - experiments, results, and their interpretation (see below for more details about how to write your report). Specifically, for this assignment, make sure you explain how you designed your parallel version of the application - i.e., what is being divided across threads and how - and reflect, if needed, on potential challenges/performance problems with your workload division. Please also explain any specific implementation details and/or their effects on performance. For example, any choices you considered about how to create/destroy threads and how to synchronize them, if at all necessary, should be discussed. Moreover, extra (empirical) research on the efficiency of different synchronization primitives is appreciated. Finally, do not forget to include the results of the proposed experiments and their analysis - specifically, consider reporting and analyzing speed-up and/or efficiency and/or scalability.

Assignment 1.2: Wave equation simulation with OpenMP

Revisit the 1-dimensional wave equation studied in assignment 1.1 and parallelize the sequential simulation code using OpenMP.

Please document the design of your parallel version (specifically focusing on the differences from the pthreads version, if any) and the choices you made for the implementation. Make sure you explain you choices for different pragmas (and briefly explain what they actually do). Finally, consider the theoretical and practical aspects of using multiple scheduling policies, as well as the performance impact of a few such policies.

Please repeat the experiments of assignment 1.1. Furthermore, experiment with different scheduling techniques and block sizes (preferably the same ones as considered in the analysis above). Report your results and comment on the achieved performance compared with what was expected from the theoretical analysis. Finally, compare your OpenMP performance results with those obtained by your pthread-based code developed for assignment 1.1.

Assignment 1.3: Sieve of Eratosthenes

The Sieve of Eratosthenes is an ancient algorithm to compute prime numbers, attributed to the Greek mathematician Eratosthenes of Cyrene (276 BC – 194 BC). Write a multithreaded C program that prints the unbounded list of prime numbers to the standard output stream following the approach of Eratosthenes as detailed below.

The (multithreaded) sieve consists of one generator thread that generates the (a-priori) unbounded sequence of natural numbers starting with the number two (by definition the smallest prime number) and a pipeline of filter threads that each filters out multiples of a certain prime number from the sequence of prime number candidates (initially all natural numbers). The threads shall communicate with each other solely by means of bounded queues, as introduced in the lecture.

The generator thread has a single output queue to which it sends the natural numbers generated: it is a producer. Each filter thread is connected to two distinct queues: an inbound queue, from where it receives natural numbers as prime number candidates, and an outbound, queue to which it sends all received numbers that are not multiples of its own filter number, and, therefore, remain prime number candidates. Consequently, filter threads are both consumers and producers.

Upon creation, a filter thread receives the address of its input queue as an argument. The first number it receives from the input queue is a prime number, which it prints to the standard output stream. Any subsequently received number that is a multiple of the initial prime number is discarded. All other numbers received on the input queue are forwarded to the output queue.

The first time a filter needs to forward a candidate prime to its output queue, the filter thread instantiates a new output queue and creates a further filter thread, which uses the new queue as input queue.

The Sieve of Eratosthenes is meant to generate the infinite list of prime numbers. Thus, it is not mandatory to terminate threads or gracefully shutdown the communication links between threads -

for a crude termination, use CTRL-C to terminate program execution when you have seen enough prime numbers. Of course, elegant termination counts as extra research/implementation.

Please make sure you detail, in your report, how you designed and implemented your parallel algorithm, and analyse, if possible, the impact of the implementation choices you make on the performance of your application.

Once your code functions correctly, please validate your application empirically, by running experiments to determine the first 100, 1000, and 5000 prime numbers, and timing the execution. Report the timing in your report. Any extra analysis/(empirical) research on the impact of the number of available cores/hardware threads on running these three tests is appreciated.

Please note: we are aware that this interpretation of the Sieve of Eratosthenes is not the most efficient way of computing prime numbers; however, we propose this approach as a programming exercise to study pipeline parallelism and event handling in multithreaded programming.

Grade weights

Table 1: Weights of different parts of the assignments used in calculating the final grades.

Assignment	Name	Code	Report
1.1	Wave equation simulation with pthreads	20%	20%
1.2	Wave equation simulation with OpenMP	15%	15%
1.3	Sieve of Eratosthenes	15%	15%
		50%	50%

Instructions for submission

For this assignment, you should submit two kinds of deliverables:

1. Source code in the form of a tar-archive of all relevant files that make up the solution of a programming exercise, with an adequate amount of comments, ready to be compiled.
2. A report in the form of a pdf file that explains the developed solution at a higher level of abstraction, illustrates and discusses the outcomes of experiments and draws conclusions from the observations made.

Please submit one archive containing your code for all three parts, and your report (one report for the entire assignment).

Instructions for writing reports

Your report should explain your solution to someone who is familiar with programming, in our case in C, including the parallel programming paradigms used throughout the course. Please find below a possible (recommended) structure for your report; you can use this structure per assignment.

Introduction

Briefly summarize the assignment as an introduction. *Note:* Although defining a research question is a major part of a research report, this is less relevant for this assignment. Thus, we recommend not to waste too much time to rephrase the assignment in all details, as the research goal/objective is clearly stated in the assignment itself [1].

Design and implementatation

Then, describe your solution *design* at a high level of abstraction. Please describe *how* you have parallelized the algorithm, and, when needed, *why* you selected certain solutions when you had more options. Make sure that anything you find remarkable or super smart about your solution is elaborated on here (that is, feel free to brag about interesting ideas and/or solutions).

Next, talk about your solution's *implementation*: *how* you have implemented your parallel algorithm using specific constructs, and, when needed, *why* you selected certain constructs when you had more options. Here it is recommended to support the explanations with code snippets. In other words, it is really not a good idea to copy the whole program code into the report, but it is often relevant and interesting to use pseudocode to highlight the core of the solution to the given problem, or specific implementation details that you find interesting to talk about. Please make sure you clearly state where in the code archive (i.e., which file) the snippet originates from.

If you think your solution is not quite the best thing since sliced bread, also discuss that. Explain potential limitations and failures - in the design and/or implementation - and explain why you could not solve them (e.g. submission deadline was 5 minutes ahead when you figured it out); ideally, please sketch out what you think would be a way forward towards solving those limitations. This can be the basis for a good report, even if the programming exercise did not work out that well for you this time around.

Experiments and results

Finally, assignments ask you to run certain experiments. Reporting on the experiments results *and analysing them* are critical parts of your report. For every experiment, we recommend that you provide *a description of the goal of the experiment, a hypothesis (usually strongly correlated with the goal), the results of the empirical tests, and how they match or not your hypothesis*. Whenever possible, please present the results in a graphical way.

[1]In other words, we consider the general research questions for each of the three assignments to be on the lines of "How can we design and implement <insert application> using <programming model>, and what is the performance we observe?". Feel free to paraphrase these "template questions" in your report.

For example, think of documenting an experiment as follows: "We run tests with our application on 1,2,...,32 threads to see the impact the number of threads has on performance. We report performance as speed-up over the sequential case, and we expect the speed-up to ... as we add more threads. The results are presented in figure ... We observe that ... which confirms/infirms/... our hypothesis. However, we also notice that ... "

Please note that we strongly recommend you present your experiments and results together - i.e., each experiment setup followed by its own results and analysis. This is the common practice in parallel processing, and it makes it easier to follow your analysis. Also note that explanations and analysis are highly appreciated. For example, explain why does code A perform better than code B, or why does the speed-up for code A increase linearly and for code B it does not increase at all, or why using more threads shows lower efficiency. All these are interesting and relevant questions, and give you excellent opportunities for you to demonstrate your knowledge.

Important: If you have ideas of other relevant experiments, that can showcase specific parts or features of your solution, please feel free to run those as well, and analyse their results - these are all examples of extra research.

Conclusion

Finally, please conclude your report with a short conclusion section, where you reflect briefly on what you have learned and what were the challenges you encountered, focused, as much as possible, on those aspects relevant to parallel algorithm design, implementation, and empirical analysis.

Note: We do not prescribe minimum or maximum page numbers, but it should be clear by now that writing the report is a significant part of each assignment series. Take this into account when planning your time. Your job is not done when your code compiles without errors!