Simulating Heat Dissipation with Chapel

Pieter Hijma, Clemens Grelck

October 31, 2017

1 Introduction

As a representative example for the parallelization of numerical problems we study the (much simplified) simulation of heat dissipation on the surface of a cylinder. In this assignment we will use the modern parallel programming language Chapel, a partitioned global address space language. We discuss the heat dissipation problem (next section) that you have to implement (Sec. 3) and parallelize with the provided framework (Sec. 5).

2 Heat Dissipation on a Cylindrical Surface

The cylinder surface shall be represented as temperature matrix t of $N \times M$ grid points. Each grid point is associated with a temperature value, specified as a double precision floating point number. More precisely, the M columns of the matrix shall make up the cyclic dimension of the cylinder (i.e. the grid points in the first column are neighbors of those in the last column) and the N rows of the matrix shall represent the other dimension of the cylinder that has fixed boundary conditions. More precisely, each boundary grid point is supposed to have a neighboring halo grid point that is initialized to the value of the boundary grid point and remains fixed over the iterations.
The simulation of heat dissipation is an iterative process. At each iteration step the temperature value of each grid point shall be recomputed as the weighted average of the previous iteration's value at that point, the previous iteration's values of the 4 direct neighbor grid points and the previous iteration's values of the 4 diagonal neighbor grid points.
Thermal conductivity is a property of the material that makes up the surface of the cylinder. Hence, conductivity coefficients are constant in the time domain, but different in the spatial domain. An $N \times M$ conductivity coefficient matrix c shall contain one (double precision floating point) conductivity coefficient for every simulation grid point. This conductivity coefficient in the interval [0,1] denotes the weight of the previous iteration's value at the same grid point for the current iteration's value. The joint weight of the direct neighbors shall be $\frac{\sqrt{2}}{\sqrt{2}+1}$ of the remaining weight and those of the diagonal neighbors $\frac{1}{\sqrt{2}+1}$. Weights of the four direct neighbors are the same, likewise for the four diagonal neighbors. A coefficient of 1 indicates no conductivity whatsoever, i.e. the temperature at the corresponding grid point remains constant; a coefficient of 0 represents maximum conductivity.
The simulation shall be finished after a given maximum number of iterations *maxiter* or when the simulation has sufficiently converged (i.e. all absolute differences between old temperature value and new temperature value across the entire temperature matrix remain below a given threshold ϵ in one iteration), whichever occurs first.
After completion of the simulation your program shall report a number of characteristic values: the number of iterations processed, the minimum, maximum and average temperature across all grid points as well as the maximum absolute difference between old and new temperature at any grid point as an indicator of convergence.

3 Requirements

Implement your program for the simulation of heat dissipation on the surface of a cylinder using the provided Chapel framework conforming to the specification in the section above. Implement four versions:

- `seq`, a sequential version,
- `par-global-single`, a global-view, single-locale, parallel version (runs on one DAS-4 node),
- `par-global-multi`, a global-view, multi-locale, parallel version (runs on multiple DAS-4 nodes), and
- `par-local-multi`, a local-view, multi-locale, parallel version.

The goal is to create efficient versions of the `par-global-single` and `par-global-multi` versions using `seq` as a reference, while optimizing the `par-local-multi` version for high scalability and performance.
For the local-view version, we recommend to take a structured approach by creating several versions and document your progress to explain your investigation later in the report, for example:

- `par-local-multi/initial_attempt`, your initial attempt;
- `par-local-multi/my_optimization1`, your first optimization (rename to a representative name)
- `par-local-multi/my_optimization2`, your second optimization (rename)
- `par-local-multi/my_optimization3`, etc.

Besides implementing the heat dissipation simulation, the goal of this assignment is to explain in a 5-10 pages report the design choices you made for the implementation, how you evaluated the performance/scalability, why you have the performance/scalability that you measured, why this is good performance/scalability and what you have done to achieve it.

4 Grading

A correct implementation, compliant with all the requirements above (both for code and documentation) is graded with 8. Up to 2 bonus points (to grade 10) can be given for extra work on implementation, optimizations, testing and benchmarking (e.g. making a more detailed evaluation of communication overheads and proposing solutions to minimize them). In addition, we organize a contest for the fastest implementation, for details see below.
We encourage creative attempts to improve the implementation, the performance, or the analysis of the parallel application beyond the mandatory requirements.
Important: You receive bonus points if you find interesting and/or creative ways to improve the parallel application (its implementation, performance, or analysis). Furthermore, make sure that the basic requirements for the assignment are still met! Additional features are only graded for working solutions.
The points are distributed as follows:

- 5 points for correct sequential and parallel implementations, compliant with all the requirements
- 1 point for coding style (use clear, commented code, according to the Coding Style section from the Canvas Pages)
- 2 points for the report

- 2 points bonus for extra work

Examples for scoring bonus points are:

- optimize the `par-global-multi` version.
- detailed analysis of the performance difference between `par-global-multi` and `par-local-multi`.
- participate in the contest and have one of the three fastest implementations.

4.1 Contest

We organize a contest for the fastest multi-locale version. To participate in the contest, send an email with the source code of your solution and the output for a run of multi-locale execution with 16 (`-nl 16`) nodes with the following parameters:

- `--N` 5000
- `--M` 5000
- `--T /home/hphijma/images/plasma_5000x5000.pgm`
- `--C /home/hphijma/images/pat2_5000x5000.pgm`
- `--I` 500
- `--E` 0.01

If your implementation is correct and indeed faster than one of the three fastest implementations, we will announce your name on Canvas. If, at the end of the course you have one of the three fastest you will get for places 1, 2, and 3 1.0, 0.7, and 0.3 bonus points respectively.

5 Framework

We provide you with a code framework that allows you to focus on the interesting and relevant code sections. The following description serves as a documentation of the user interface of the framework, or, if you want, as a specification of the behavior of your code.

5.1 Program command line parameters

The interface to your program should present itself as an executable named `heat_dissipation` accepting the following command-line parameters:

- `--N` specifies N, the number of rows;
- `--M` specifies M, the number of columns;
- `--I` specifies *maxiter*, the maximum number of iterations;
- `--E` specifies ϵ, the convergence threshold;
- `--C` specifies the input file to read conductivities from;
- `--T` specifies the input file to read initial temperature points from;
- `--H` specifies H, the highest temperature in the input file;
- `--L` specifies L, the lowest temperature in the input file;

The framework provides default values for these parameters in the form of `config` variables.

5.2 Program input and parameters

Input files are available on the DAS-4 file system in `/home/hphijma/images` in the ASCII Portable Graymap format[1] (P2). This format supports the definition of 2D point grids where each point receives a value between 0 and a maximum *max*. To translate an input grid to conductivities, the input value 0 should be mapped to conductivity 0 and the value *max* to conductivity 1. To translate an input matrix to temperatures, the input value 0 should be mapped to the temperature L specified with `--L` and the value *max* to the temperature H specified with `--H`. The mapping should be linear.
[1] `http://en.wikipedia.org/wiki/Netpbm_format`
To ease your task and to ensure uniform usability across solutions, the framework contains utility functions to read in these files.

5.3 Program output

The framework provides a `results` record with the following fields:

- `niter`, the number of iterations
- `tmin`, the minimum temperature of the surface
- `tmax`, the maximum temperature of the surface
- `tavg`, the average temperature of the surface
- `maxdiff`, the maximum temperature difference
- `tmin`, the time in seconds

The `heat_dissipation` application should print its parameters, its header, and the result. Your task is to focus on the procedure `do_compute` that should perform the heat dissipation simulation and return the results in the `results` record. The framework already takes care of printing the output. Initially, the framework prints the parameters, the header, the temperature file that has been read and the result values that are all 0 except for the time.

5.4 Source tree layout

The source tree can be extracted with:

```
tar xvzf heat_dissipation-1.2.tar.gz
```

The framework contains the file `set_env.bash` that, when sourced with (for bash) will setup your environment for use with the Chapel distribution located in `/home/hphijma/chapel-1.14.0/`. For using other shells, see `/home/hphijma/chapel-1.14.0/util/README`.

```
source set_env.bash
```

The source tree already contains the four directories `seq`, `par-global-single`, `par-global-multi`, and `par-local-multi/initial_attempt` with a working setup. Each of these directories contains a `das4.job` file that describes how a job should be run, the `heat_dissipation.chpl` file to which you add your implementation, a `Makefile` that describes how to compile the program, and `util.chpl` that contains utility functions such as reading in `.pgm` files.
To compile one of the versions, go to the directory and run:

```
make
```

To run it on the head node and print information about the parameters provided by Chapel and the application, you can do the following:

`./heat_dissipation --help`

To see documentation about the `heat_dissipation` parameters, run:

`./heat_dissipation --help_params`

The `das4.job` file contains a job description where you can choose several parameters for the file. Loading the Chapel environment also adds the `chpl2das4` script to your path which will search for a `das4.job` and submit it to the DAS-4, so to run the heat dissipation, do:

`chpl2das4`

For multi-locale execution, it is necessary to specify the number of locales, so, for using one node:

`chpl2das4 -nl 1`

To see the output of the job, do:

`cat HEAT_DISSIPATION.o<job-number>`

To wait for the output of the job interactively, you can do:

`tail -f HEAT_DISSIPATION.o<job-number>`

To clean do:

`make clean`

To clean the jobs do:

`make jobclean`

6 Submitting

You have to submit both the code (containing useful comments that illustrate how the application works) and the report. Make sure that your submission has the same directory structure as the provided framework. You are free to create different versions as discussed in the previous section. Create the report as a PDF file, and make sure you place the file in the pre-created docs directory. Rename the `heat_dissipation` directory to match your VUNetID and first and lastname, for example, `jsh400_JohnSmith_heat_dissipation`, make sure there are no binaries and that you have working `Makefile` and `das4.job` (see Sec. 5) files. Archive the directory as a `.tar.gz` file (i.e. `jsh400_JohnSmith_heat_dissipation.tar.gz`) and submit the archive via Canvas. To create the archive, use the following command:

```
tar cvzf jsh400_JohnSmith_heat_dissipation.tar.gz \
  jsh400_JohnSmith_heat_dissipation/
```

7 Further information

For further information, we refer you to the Canvas Syllabus section which contains information about domain maps, installing Chapel on your local machine and using the Chapel visualizer tool that can help you find bottlenecks. For questions, send an email to pieter@cs.vu.nl and note that we expect you to ask questions.
Please make sure that your questions are specific, so for example:

- what you try to achieve,
- what you already have tried,
- what you expect to happen (your hypothesis),
- what actually happens.