Parallel Programming Concepts
WS 2013 / 2014

Assignment 6 (Submission deadline: Feb 17th 2014, 23:59 CET)

General Rules

The assignment solutions have to be submitted at:

https://www.dcl.hpi.uni-potsdam.de/submit/

Our automated submission system is intended to give you feedback about the validity of your file upload. A submission is considered as accepted if the following rules are fulfilled:

- You did not miss the deadline.
- Your file upload can be decompressed with a zip / tar decompression tool.
- Your submitted solution contains only the source code files and a Makefile for Linux 2.6 64-bit. Please leave out any Git / Mercurial repository clones or SVN / CVS meta-information.
- Your solution can be compiled using the "make" command, without entering a separate sub-directory after decompression.
- You program runs without expecting any kind of keyboard input or GUI interaction.
- Our assignment-specific validation script accepts your program output / generated files.

If something is wrong, you will be informed via email (console output, error code). Re-uploads of corrected solutions are possible until the deadline.

Every task that you solve correctly in this assignment compensates for a full assignment that you missed / failed. Documentation should be done inside the source code.

Students can submit solutions either **alone or as team of max 2 persons**.

Assignment 6

This assignment covers a variety of challenging parallel problems. It fulfills two purposes:

1. To challenge your parallel programing skills and demonstrate your deep understanding.
2. To compensate for assignments that were missed / failed at.

Documentation should be done inside the source code.

Hasso Plattner Institute Operating Systems and Middleware Group
Dr. Peter Tröger, Frank Feinbube

Task 6.1: Heat Map with everything available

Implement the heat map example using all available hardware resources (CPUs and GPUs). The goal is to minimize the execution time of the complete simulation. You are free to use any programming language / model.

Input

As always your application has to be named “heatmap” and needs to accept five parameters:

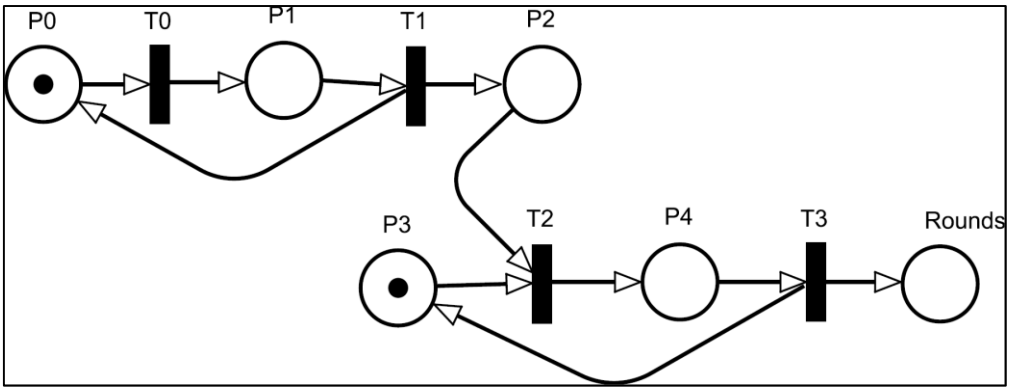
```
Example:
./heatmap 20 7 17 hotspots.csv

./heatmap 20 7 17 hotspots.csv coords.csv
```

The student achieving the lowest average runtime will be announced in the lecture.

Task 6.2: Petri Net Simulation with Erlang

Implement an Erlang module which simulates the following Petri Net[1] using Erlang’s parallelization features. It represents a simple producer-consumer setup.



The firings of transitions should be atomic events. All transitions in the model depicted above have the same priority. Make sure your implementation is starvation-free. Although for this task you only have to implement this petri-net, please design your solution in a way that supports arbitrary petri nets.

You can compile and run it from the command line as follows:

```
>> erlc petrinet.erl
>> erl -run petrinet simulate 180 marking.csv
```

`erlc` creates Erlang bytecode, producing a `.beam` file. Make sure to keep this .beam file up to date when testing your program.

Input

Your module must be named "petrinet" with a function named "simulate" of arity 1 (i.e., accepting a

[1] http://embedded.eecs.berkeley.edu/Research/hsc/class.F03/ee249/discussionpapers/PetriNets.pdf

list of command line arguments). It must accept two command line parameters: The simulation time in seconds and the path to a file containing the initial marking.

```
Example:
erl –run petrinet simulate 3 marking.csv
```

The marking file contains a comma-separated list of natural numbers defining the markings of the places, e.g. (as shown in the picture):

```
Example content of marking.csv:
1,0,0,1,0,0
```

Output

After running for the requested number of seconds, your program must produce a file named `output.csv` which contains the comma-separated markings of all places after the simulation.

```
Example content of output.csv:
0,1,0,0,1,271828
```

Task 6.3: Wator with Scala Actors – No Global Barrier Edition

Modify your parallel solution of the Wator example of Assignment 5 so that no global barrier is needed, while the concept of simulation rounds is kept. This should be realized by letting each cell keep a history of states from earlier simulation time stamps.

Task 6.4: Cooperative Hybrid Computing of Tensor Products

In *Quantum Mechanics* as well as *Quantum Computing*, the state of a system is described as a vector of complex values. When two or more quantum systems are to be combined, a special mathematical operation is performed to combine the corresponding state vectors. This operation is known as the *Tensor Product of Vectors*[2] (or *Kronecker Product*). Other applications include systems theory, matrix calculus, matrix equations, etc.

Given two vectors,

$$A = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \; and \; B = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix},$$

The tensor product of A and B is defined as:

[2] *Tensor Product of Vectors*

$$A \otimes B = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \otimes \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} a_0 \cdot \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \\ a_1 \cdot \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \\ a_2 \cdot \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \\ a_3 \cdot \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} a_0 b_0 \\ a_0 b_1 \\ a_0 b_2 \\ a_0 b_3 \\ a_1 b_0 \\ a_1 b_1 \\ a_1 b_2 \\ a_1 b_3 \\ a_2 b_0 \\ a_2 b_1 \\ a_2 b_2 \\ a_2 b_3 \\ a_3 b_0 \\ a_3 b_1 \\ a_3 b_2 \\ a_3 b_3 \end{bmatrix}$$

If A is of size m, and B is size n, the vector resulting from the tensor product is of size mn. Therefore, the size of the output vector grows very fast, even for moderately sized input vectors.

In this exercise you'll implement a CUDA kernel that computes a tensor product of two vectors (for the purpose of this exercise, the vectors will be real valued – represented by double precision floating point data type). The primary objective of the exercise is to introduce asynchronous Device to Host memory transfer using CUDA streams. For a dataset that is too big to fit into the Device memory, it must be divided into partitions. Partitions are then processed one at a time. Once a partition has been processed, the corresponding output vector must be transferred to the Host, and the Device memory for the output must be de-allocated to make room for the next partition. The default D2H memory transfer in CUDA is synchronous, i.e., the memory transfer must complete before the next partition can be processed. In this exercise, however, you must use asynchronous D2H memcpy.

Once the output vector of a partition has been copied to the Host, it must extract the maximum value of the data. The asynchronous D2H memcpy can be used to overlap the memcpy and maximum acquisition.

Note: Partitions will be required for the result of the tensor product.

Input

Your program has to be named "tensorproduct" and needs to accept one parameter indicating the input file with two input matrixes.

```
Example:
./tensorproduct input.txt
```

The first column belongs to the first vector; the second column belongs to the second vector.

```
Example content of the input.txt:
-2.6186146828319088e+000 -2.8832025844288403e+000
-2.6622582608791072e+000 -2.8262222961990213e+000
```

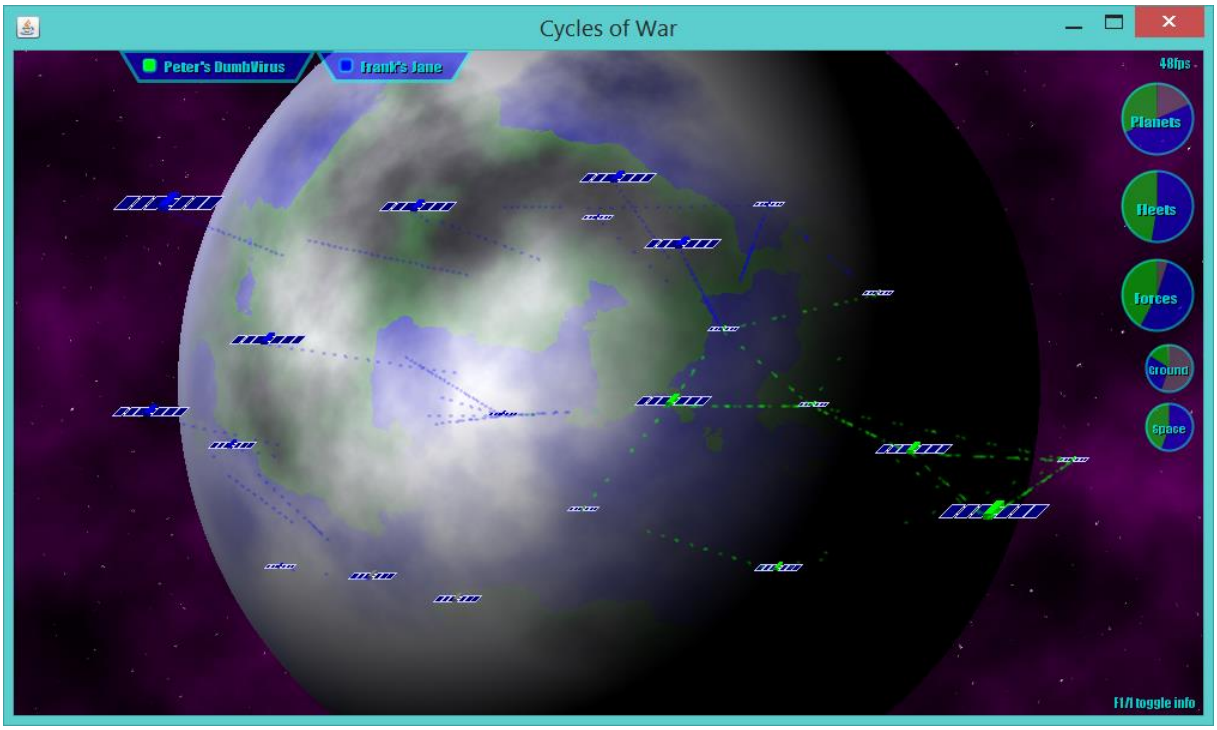
Output

The program must terminate with exit code 0 and has to produce an output file with the name "output.txt" in the same directory. The "output.txt" has to contain the maximum value in the result vector.

```
Example content of the output.txt:
-7.5499966211642673e+000
```

Task 6.5: Cycles of War with Scala

The Operating Systems and Middleware Group features a simulation of artificial intelligences competing over the restricted resources of a small sandbox universe.



The full implementation can be accessed here:

svn://code.hpi.uni-potsdam.de/cyclesofwar

The README file describes the whole game and how new bots can be contributed. Your mission, should you accept it, is to implement the core of the simulation (rules, training bots, tournaments, NO GUI) on Scala.