Storms of High-Energy Particles: An assignment for OpenMP, MPI, and CUDA/OpenCL

1st Arturo Gonzalez-Escribano
Dept. Informatica, Universidad de Valladolid
Valladolid, Spain
arturo@infor.uva.es

2nd Eduardo Rodriguez-Gutiez
Dept. Informatica, Universidad de Valladolid
Valladolid, Spain
eduardo@infor.uva.es

***Abstract*—We present an assignment used in a Parallel Computing course to teach the approaches to the same problem in different parallel programming models. It targets basic concepts of shared-memory programming with OpenMP, distributed-memory programming with MPI, and GPU programming with CUDA or OpenCL. This assignment is based on a highly simplified simulation of the impact of high-energy particle storms in an exposed surface. The idea is inspired by simulations to test the robustness of space-vessels material. The program is designed from scratch to be simply, easy to understand by students, and to include specific parallel structures and optimization opportunities. A simple parallelization in the three models considered is quite direct. But the program is plenty of opportunities for further improvements, optimizations, and more sophisticated techniques usage. It has been successfully used in parallel programming contests during our course, using the performance obtained by the students code as a measure of success.**

I. Idea and context

Different programming models use different approaches for the parallelization of basic application structures. Understanding these differences is key for students to get into more advanced techniques, and to face parallel programming in current heterogeneous platforms. We have designed a course of Parallel Programming that introduces the basic concepts and techniques to program using OpenMP, MPI, and CUDA or OpenCL.

We use the same simple but inspirational application in the assignment for each programming model. The provided material includes a sequential code and a test-bed of input files. After the lectures and laboratory sessions dedicated to teach the basic concepts and tools for each programming model, we start a programming contest of one week. During that week the students parallelize the code with the target programming model, and compete to obtain the best performance. The students can use common compilers and PC platforms to develop and test their codes. An automatic judge tool with an on-line public ranking is used to provide a fair arena, and to keep the students engaged during the contest. Other gamification tools are also included [1].

II. Concepts covered

For simplicity, the program simulates the effect of the particles impact in a cross section of the surface. It uses 1-dimensional arrays to represent the discretized space. Each storm is an unordered collection of pairs of numbers. Each one represent the impact position and the energy value of a particle. For each storm the program applies three stages: (1) Update the array cells depending on particles energy and impact distances; (2) Compute a relaxation using a stencil operator; and (3) Compute a reduction to obtain the maximum energy value and its position in the array. The output of the program is the lists of maximum values and positions after each storm. In debug mode, the program also writes the final energy values in all array positions in a graphical plain text representation. More sophisticated representations of the evolution of the energy after each storm can also be obtained with simple tools such as *gnuplot*.

The basic concepts covered in OpenMP are parallelization of loops, selecting the outer/inner loop to avoid race conditions, and non-trivial reductions. Loop reordering can also be applied. In the case of MPI, the students need to understand how to compute the limits of a distributed array in terms of the process index, using halos and neighbor communications for the stencil part, and generic reductions. For GPU programming, the students work with the concepts of memory management in the co-processor, reducing host-to-device and device-to-host communication, embarrassing parallel kernels, choosing proper thread-block sizes, and simple reductions.

Plenty of further code optimizations can be discovered and applied, such as pointer swap to avoid array copies, reverse threshold calculations to narrow loop limits, using wider halos to reduce synchronization stages in MPI, using shared memory in GPUs to implement faster reductions, etc.

III. Conclusion

We present an assignment that can be used to show the different approaches to parallelize the same application in very different parallel programming models. It covers basic concepts, and simple parallelizations are easily obtained. The assignment also provides plenty of opportunities for further optimization. It has been used in the context of programming contest and gamification scenarios to engage the students.

References

[1] J. Fresno, A. Ortega-Arranz, H. Ortega-Arranz, A. Gonzalez-Escribano, and D.R. Llanos. *Gamification-Based E-Learning Strategies for Computer Programming Education*, chapter 6. Applying Gamification in a Parallel Programming Course. IGI Global, 2017.

APPENDIX: REPRODUCIBILITY

The assignment has been used in the context of a Parallel Computing course, in the third year of the Computing Engineering grade at the University of Valladolid (Spain).

The on-line judge program used in the programming contests is named *Tablon*, and it was developed by the Trasgo research group of the University of Valladolid (https://trasgo.infor.uva.es/tablon/). The contest software uses the Slurm queue management software to interact with the machines in the cluster of our research group. During the course we used Slurm 17.02.7.

The machine of the cluster used for the OpenMP and CUDA/OpenCL contests is named *hydra*. It is a server with two Intel Xeon E5-2609v3 @1.9 GHz CPUs, with 12 physical cores, and 64 GB of RAM. It is equipped with 4 NVIDIA's GPUs (CUDA 3.5), GTX Titan Black, 2880 cores @980 MHz, and 6 GB of RAM.

During the MPI contest we use *hydra* in combination with two other servers to create a heterogeneous cluster. The other two machines are: *thunderbird*, with an Intel i5-3330 @2.4 GHz CPU and 8 GB of RAM; and *phoenix*, with and Intel QCore @2.4 Ghz CPU with 6 GB of RAM.

All machines are managed by a CentOS 7 operating system. The compilers and system software used have been GCC v6.2, and CUDA v9.0.

The assignment provides the sequential code and the input files of the test-beds for the students. Other test-beds used by the on-line judge during the contest are also provided.

The results of the contests are publicly available until the start of the next semester at http://frontendv.infor.uva.es.