ISSN 1064-5624, Doklady Mathematics, 2011, Vol. 84, No. 3, pp. 879–881. © Pleiades Publishing, Ltd., 2011.
Original Russian Text © V.A. Bakhtin, V.A. Kryukov, B.N. Chetverushkin, E.V. Shil'nikov, 2011, published in Doklady Akademii Nauk, 2011, Vol. 441, No. 6, pp. 734–736.

COMPUTER SCIENCE

Extension of the DVM Parallel Programming Model for Clusters with Heterogeneous Nodes

V. A. Bakhtin, V. A. Kryukov,
Corresponding Member of the Russian Academy of Sciences B. N. Chetverushkin, and E. V. Shil'nikov

Presented by Academician S.K. Korovin July 20, 2011

Received July 6, 2011

DOI: 10.1134/S1064562411060408

INTRODUCTION

The appearance of clusters with heterogeneous nodes, which use graphics processors (GPUs) as accelerators, has seriously complicated programming, because such clusters require employing, in addition to the MPI and SHMEM low-level technologies, the CUDA or the OpenCL low-level technology. In development and coming soon are new processors with a large number of cores (such as the 48-core Intel SCC processor, which has recently become available for a wide circle of researchers [1]); efficiently using them requires new programming models.

Developers of new parallel programming languages have, evidently, not enough time to trace the architectural diversity of multicore processors. The possibility of designing and popularizing a new high-level language in 5–10 years is very unlikely. Therefore, in the nearest future, programmers will have to use hybrid languages uniting various models of parallel programming.

APPLICATION OF HYBRID MODELS AND LANGUAGES

At present, programs for high-performance computations on modern clusters extensively use three programming models, MPI, OpenMP, and CUDA. Technically, low-level programming models realized as libraries are easier to combine than high-level models realized as languages and the corresponding compilers. But such programs are much harder to program, debug, accompany, and transfer to other computers. For example, the passage from an NVIDIA GPU to an AMD GPU requires replacing CUDA by OpenCL. Thus, it is important to use high-level models and programming languages.

Among high-level programming models a special place is occupied by models implemented by augmenting programs in standard successive languages with specifications which control transferring these programs to parallel machines. These specifications, written as comments in FORTRAN programs or compiler directives (pragmas) in C and C++ programs, are not seen by usual compilers, which significantly facilitates adopting new models of parallel programming. Such models for clusters are HPF and DVM [2], and for multiprocessors, OpenMP.

In recent years, similar models (such as HMPP [3], hiCUDA [4], and PGI_APM [5]) have also been developed for GPUs. These three models have much in common, which makes it possible to assert the emergence of a new approach to programming accelerators of GPU type. A common feature of these models with the DVM and OpenMP models is also that the programmer has full control over the mapping of data and computations to hardware. Thus, it is quite natural to use the suggested approach for extending DVM and OpenMP models in order to facilitate and speed up the development of programs for clusters with heterogeneous nodes.

Such OpenMP extension projects have already appeared; some authors [6] suggest to take the HMPP model for the base, and other authors (Cray)—suggest using the PGI_APM model.

The appearance of clusters with GPUs, such as *K-100* (designed at Keldysh Institute of Applied Mathematics, Russian Academy of Sciences) and *Lomonosov* (Research Computing Center, Moscow State University) sets the urgent task of extending the DVM model and developing the DVMH (DVM for Heterogeneous systems) model.

Keldysh Institute of Applied Mathematics, Russian Academy of Sciences, Miusskaya pl. 4, Moscow, 125047 Russia
e-mail: chetver@imamod.ru, bakhtin@keldysh.ru, krukov@keldysh.ru, shiva@imamod.ru

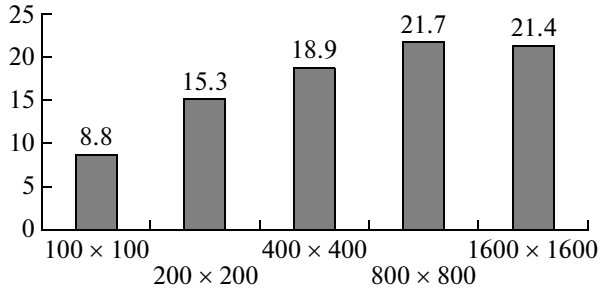

Fig. 1. Speedup of the cavern program on grids of various sizes for one *K-100* GPU.

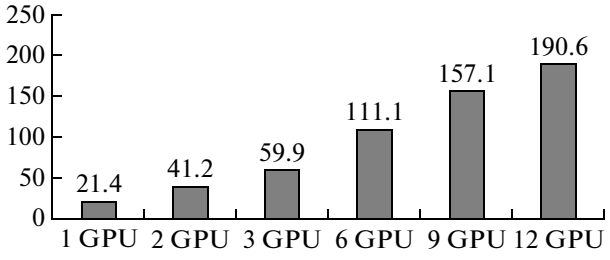

Fig. 2. Speedup of the on cavern program on a 1600 × 1600 grid depending on the number of *K-100* GPUs.

DVMH MODEL: THE FORTRAN DVMH LANGUAGE AND A COMPILER

The development of the DVMH model was based on the DVM model augmented by the following facilities.

1. Determination of program fragments to be implemented on a particular accelerator. Such fragments of programs (called computational regions, or simply regions) can be separate DVM-cycles or their sequences. For these fragments, accelerator types and choice conditions for each of them can be specified. If there is no suitable accelerator for implementing a region, then this region is implemented on the universal processor.

2. Determination of data required by regions. For each region, the required data and their form of application (input, output, local) are specified.

3. Specification of cycle properties and rules for mapping cycle coils to an accelerator. For each DVM cycle, a thread block configuration (in the CUDA terminology) can be specified. If such a configuration is not specified in the program, then it is determined automatically.

4. Controlling data transfer between the main memory of the universal processor and the memories of accelerators. Data transfer is mainly performed automatically in accordance with regions execution on accelerators and an information about the data which they use. For fragments of programs executed on the universal processor (outside computational regions), there are special directives for specifying what data from the accelerators are used and corrected by these fragments.

In order to implement the DVMH model, the FORTRAN DVM language was augmented by new directives. At present, the implementation of the first version of a compiler for the extended language is being completed. The compiler translates a FORTRAN DVMH program into a PGI FORTRAN CUDA program with calls of functions of the DVM support system (Lib-DVM). In the current version of the compiler, calls of user procedures/functions from computational regions and processing remote references of REMOTE type are not implemented.

EXPERIMENTAL RESULTS

The existing version of the compiler has already made it possible to perform experiments with real-life applications on the *K-100* cluster. In the FORTRAN DVMH language, the flow of an incompressible fluid or weakly compressible gas near a rectangular cavity was simulated; for the initial mathematical model the hyperbolic version of the quasi-gas dynamic system was used. A two- and a three-dimensional problem, the cavern problem and the container problem, were solved.

The speedups achieved for the cavern problem by computations on an NVIDIA Fermi C2050 GPU in comparison with computations on an Intel Xeon X5670 processor for grids of various sizes are shown in Fig. 1.

The influence of the grid dimension on the accelerator performance is largely caused by that relatively small amounts of processed data do not completely load the hardware. Moreover, in processing arrays of small dimension, the universal processor wins by the use of caching, which does not happen for large amounts of data.

Figures 2 and 3 show the speedup of the cavern program (for a 1600 × 1600 computational grid) and the container program (for a 120 × 120 × 120 computational grid) depending on the number of graphics processors.

An analysis of the obtained results shows that the performance of programs in the high-level DVMH hybrid model differs very little from that of programs designed by using the CUDA low-level technology [7].

CONCLUSIONS

The HMPP, hiCUDA, and PGI_APM high-level models for GPUs developed in recent years were studied; a new model DVMH for clusters with heterogeneous nodes was suggested. Extensions of the FORTRAN DVM language and of the support system Lib-DVM for parallel implementation of programs were described. The first version of a compiler for the FORTRAN DVMH language was designed.

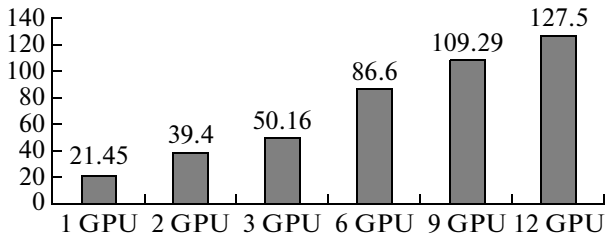

Fig. 3. Speedup of the container program on a 120 × 120 × 120 grid depending on the number of *K-100* GPUs.

A study of characteristics of test and real-life applications was performed, which confirmed the applicability of the language and the efficiency of its mapping to clusters with heterogeneous nodes. Successfully applying the developed language requires convenient tools for functional debugging and analyzing the performance of GPU program implementation.

ACKNOWLEDGMENTS

This work was supported by Presidium of the Russian Academy of Sciences (program nos. 14, 15, and 17) and by the Russian Foundation for Basic Research (project nos. 10-07-00211 and 11-01-00246).

REFERENCES

1. T. G. Mattson, R. F. van der Wijngaart, F. Rob, et al., in ACM/IEEE International Conference on High-Performance Computing, Networking, Storage and Analysis, Washington, D.C., U.S.A., 2010 (IEEE Computer Soc., 2010), pp. 1–11.

2. N. A. Konovalov, V. A. Kryukov, S. N. Mikhailov, et al., Programmirovanie, No. 1, 49–54 (1995).

3. F. Bodin and S. Bihan, Sci. Program. Software Develop. Multi-Core Comput. Syst. **17** (4), 325–336 (2009).

4. T. D. Han and T. S. Abdelrahman, in *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units* (ACM, New York, 2009), pp. 52–61.

5. M. Wolfe, in *Proceedings of 3rd Workshop on General-Purpose Computation on Graphics Processing Units* (ACM New York, New York, 2010), pp. 43–50.

6. E. Ayguade, R. M. Badia, D. Cabrera, et al., in *Proceeding of 5th International Workshop on OpenMP: Evolving OpenMP in an Age of Extreme Parallelism* (Springer-Verlag, Berlin, 2009), pp. 154–167.

7. A. A. Davydov, B. N. Chetverushkin, and E. V. Shil'nikov, Vychisl. Mat. Mat. Fiz. **50** (12), 2275–2284 (2010).