An Extension of the DVM System to Solve Problems with Intensive Irregular Memory Access

Vladimir Bakhtin
KIAM RAS
Moscow, Russia
dvm@keldysh.ru

Alexander Kolganov
KIAM RAS
Moscow, Russia
dvm@keldysh.ru

Victor Krukov
KIAM RAS
Moscow, Russia
dvm@keldysh.ru

Natalia Podderyugina
KIAM RAS
Moscow, Russia
dvm@keldysh.ru

Michail Pritula
KIAM RAS
Moscow, Russia
dvm@keldysh.ru

Olga Savitskaya
KIAM RAS
Moscow, Russia
dvm@keldysh.ru

Abstract

The DVM system was designed to create parallel programs of scientific-technical calculations in CDVMH and Fortran-DVMH languages. These languages use the same model of parallel programming (the DVMH model) and are extensions of standard C and Fortran languages by parallelism specifications, implemented as compiler directives. The DVMH model allows to create efficient parallel programs for heterogeneous computational clusters, which nodes use as computing devices not only general purpose multi-core processors but also can use attached accelerators (GPUs or Intel Xeon Phi coprocessors). This article discusses new possibilities to work with irregular grids and graphs, which were implemented in the CDVMH compiler recently. Using the developed extension can considerably simplify a parallelization of irregular grid applications on a cluster.

Keywords: irregular grids, DVMH, parallel execution, GPU

In: V. Voevodin, A. Simonov (eds.): Proceedings of the GraphHPC-2017 Conference, Moscow State University, Russia, 02-03-2017, published at http://ceur-ws.org.

1 Introduction

To achieve high accuracy of calculations the researchers are forced to considerably refine a calculation grid. It leads to proportional increase of computer memory usage and increase of calculation time. Use of unstructured grids instead of structured ones allows to solve this problem partially. In this case there is an opportunity to vary a grid detailing on the calculation area, thereby to reduce both time for excessively exact calculations on some areas and random access memory used to store not needed detailed values. Also it allows to abstract numerical methods from the calculation area geometry and practically to remove requirements to it.

The class of tasks with irregular memory access is wide enough:

- Large-scale graph processing;
- Sparse matrix problems;
- Scientific and technical calculations on irregular grids.

A lot of programs are written now in more general form to apply widely and reuse the program codes [1] for irregular grids. However, such programs have much more complicated structure. When operating with regular grids it is not necessary to store explicitly the neighbourhood relations and space coordinates, as these properties and values are directly bound with multidimensional index spaces of value arrays. Such approach has obvious advantage in a memory economy, and also it sets understandable rules for parallelization of computations both at the vectorization

level and at the level of computational clusters and networks.

On the one hand, the optimizing compilers track the accesses to the elements of the arrays with constant shifts and it allows to organize simultaneous execution of several loop iterations. On the other hand, the data parallelism approach has been developed: the data arrays are divided by blocks, and different blocks are processed by separate processors which perform the same (source) program and sometimes exchange by boundary elements.

There are several approaches to solve this problem:

- The ways to write a parallel program on irregular grids are proposed [1, 4, 8];
- The approaches of manual parallelization of serial programs on unstructured grids are developed [2];
- The tools (function libraries) hiding the difficulties of a parallelization for distributed memory are developed [7];
- Automatic mechanisms of a parallelization based on inspector/performer model are developed [3,9];
- Specialized languages are developed [6].

2 What Are the DVMH Model and the DVM System

The DVM system [10] was developed in Keldysh Institute of Applied Mathematics, Russian Academy of Sciences, with active participation of graduate students and students of Faculty of Computational Mathematics and Cybernetics of Lomonosov Moscow State University. It is designed to create parallel programs of scientific-technical calculations in C-DVMH and Fortran-DVMH languages. These languages use the same model of parallel programming (the DVMH model) and are extensions of standard C and Fortran languages by parallelism specifications, implemented as compiler directives. The directives are invisible to standard compilers, so a programmer can have single program for sequential and for parallel execution on computers of different architectures.

The DVMH model allows to create efficient parallel programs (the DVMH programs) for heterogeneous computational clusters, which nodes use as computing devices not only general purpose multi-core processors but also can use attached accelerators (GPUs or Intel Xeon Phi coprocessors). In the last case computations mapped to the node can be automatically distributed between the computing devices of a node taking into account their performance. The C-DVMH and Fortran-DVMH compilers convert the source program into a parallel program using standard programming technologies MPI, OpenMP and CUDA. The DVM system includes the tools of functional debugging and performance debugging of the DVMH programs.

3 The Problems with Irregular Grids in the DVMH Model

The DVMH model is based on data parallelism [5]. The notion of distributed multidimensional array is the basis of this model. Each processor has not only a local part of a distributed array, but also so-called shadow edges – the copies of elements from local parts of adjacent processors. Main interconnection of the processors is performed via these shadow edges. The distribution of computations is performed by their mapping on the distributed arrays. Since index shifts of used arrays are known in advance, an access is performed either to own local part, or to shadow edges of known width defined as a continuation of the local part along certain dimension of the distributed array. For example, for a "cross" template with 4 neighbors, an element with (i, j) indexes is calculated using the elements with indexes (i-1, j), (i, j-1), (i+1, j), (i, j+1) and shadow edges of width 1 for both dimensions are needed.

The DVMH compilers transform references to distributed multidimensional arrays to a form independent from sizes and location of a local part on each processor, and initial index expressions are not modified. As a result each access to distributed data is performed in global (initial) indexes, but the coefficients and shifts calculated during execution are used for the access to memory for each dimension.

Such approach (unlike modifying of the index expressions) allows abstracting from the contents of parallelizable loops, but it introduces a serious restriction on a form of the distributed array part addressed by each processor. This part is called extended local part and is a union of the local part and shadow edges. Only block distributions with shadow edges are supported in the DVMH model, i.e. the extended local part is a subarray of source array of the form (A1:B1, A2:B2, A3:B3, ..., An:Bn).

Note that the DVMH model has no tools to describe even cyclic distributions (which, together with block ones, are used to parallelize the programs on regular grids) because their support would require to perform the division operations at each access to arrays and was rejected for optimization purposes.

Two ways of expansion of the DVM system opportunities to solve the problem of these restrictions are discussed in the article. The first of them allows to distribute user data manually using MPI tools or other technology of parallel programming, leaving a possibility to use the DVMH languages inside a node (multicore CPU, GPU). The second way assumes considerable extension of the DVMH languages and modifying of the DVMH compilers to introduce new types of distributed arrays, parallel loops and other auxiliary con-

structions allowing to simplify significantly the parallelization of available applications with irregular grids on a cluster.

4 Use of the DVMH Tools in MPI Programs

Now, when parallel computers are used several decades to perform the calculations, there are many programs which were already parallelized on a cluster, but have not parallel version on CPU cores, for example, with OpenMP use, and also do not use GPUs.

Traditionally in the DVMH approach a programming process (or a parallelization of available serial programs) begins with the distribution of the arrays, and then parallel computations are mapped on them. It means that to use the DVM system tools, it is necessary to convert the programs, parallelized, for example, using MPI, back to serial ones and to replace manually distributed data and computations by the distributed arrays and parallel loops described in the DVMH language.

However, at first, an author not always wants to refuse from his parallel program, and at second, it is not always possible to realize the source scheme of data and computation distribution in the DVMH language. In particular, the transformation of the tasks on irregular grids in the DVMH model will require non-trivial decisions and tricks and is not always possible.

One of the ways to solve both problems is a new mode of the DVM system operating: the DVM system does not participate in interprocessor communications, but works locally on each process. This mode is turn on by specifying a specially created MPI library when the DVM system is built. The library does not perform any communications and does not conflict with real MPI implementations. As a result an illusion is created for the DVMH runtime system that the program is executed on one processor.

In addition to such mode, the notion of non-distributed parallel loop is introduced in the CDVMH compiler. For such loop it is not needed to specify mapping on distributed array. For example, the three-dimensional parallel loop can look as follows:

```
#pragma dvm parallel(3)
for (int i = L1; i <= H1; i++)
    for (int j = L2; j <= H2; j++)
        for (int k = L3; k <= H3; k++)
...
```

By definition such loop is executed by all the processors of the current multiprocessor system, but since in described new mode the DVM system thinks that the multiprocessor system consists of only one process, such construction does not cause the replication of computations but allows to use a parallelism only within one process (of CPU or GPU). As a result, it becomes possible not to specify any distributed array in terms of the DVMH model and at the same time to use the following DVM system possibilities:

- Addition of a parallelism in shared memory (CPU cores): with OpenMP use or without its use, a possibility to specify thread binding;
- GPU use: not only "naive" porting of a parallel loop on an accelerator, but also performing of automatic reorganization of data, simplified control of data movements;
- Selection of optimization parameters;
- Convenient tools of performance debugging.

This mode can be used in particular to obtain the intermediate results in a process of full parallelization of a program in the DVMH model. It allows quickly and more easy to create the program for multi-core CPU and GPU, and also to evaluate perspectives of target program speedup on a cluster with multi-core CPUs and accelerators (there is a set of restrictions for work with distributed arrays, but it is optional to create them in such approach).

Consider as an example of the program that is a part of big advanced complex of computational programs. The program is oriented on a decision (explicit scheme) of the systems of hyperbolic equations (generally of gas dynamics) on two-dimensional areas of complex shape with use of unstructured grids. It was written on the C++ language with very wide use of object-oriented approach to provide maximum universality and simplicity for further development.

As this program is a part of whole complex, its code is based on a wide platform of the basic notions and data structures. It leads to considerable sizes (39 thousand lines) and complexity of the program if to consider it entirely. Full parallelization on a cluster is hardly possible without examination and modification of whole program, however new possibilities of the DVM system allowed to select relatively not labor-consuming first stage of the parallelization on CPU cores and graphic accelerator.

Such parallelization can be performed "locally", i.e. the modifications are required only in computation-consuming program parts of size about 3 thousand lines.

In comparison with a serial version of the program accelerations by 9.83 times on two 6 cores CPUs, and by 18 times on GPU NVIDIA GTX Titan were obtained. These results confirm the efficiency of considered program mapping on the accelerators and multi-core CPUs by the DVM-system and give the grounds to continue the parallelization of the program already with use of distributed arrays in the DVMH model.

5 New Possibilities of Operation with Irregular Grids in the CDVMH Compiler

To operate with irregular grids a new type of array and template distribution – by-element distribution – was introduced. This type of distribution does not superimpose any restrictions on what elements of the array should be located on the same processor or what elements of the array should be located on adjacent processors. On the contrary, it allows to specify arbitrary belonging of each element of the array independently.

Two new rules of by-element distribution were added: **indirect** and **derived**.

Indirect distribution is specified by an array of integer numbers, its size is equal to the size of indirectly distributed dimension, and the values specify a domain number. The quantity of domains can be either more than a number of processors as less. The DVM system guarantees that all elements of one domain belong to the same processor.

Derived distribution is specified by the rule, which form is similar to the form of alignment rule (ALIGN) of the DVMH model. However, it has more considerable flexibility. The syntax can be described as it is shown in Figure 1.

indirect-rule ::= **indirect** (var-name)
derived-rule ::= **derived** (derived-elem-list **with** derived-templ)
derived-elem ::= int-range-expr
int-range-expr ::= arbitrary integer expression+ ranges are allowed in index expressions, use of align-dummy variables.
derived-templ ::= var-name [derived-templ-axis-spec]...
derived-templ-axis-spec ::= [ ] | [@align-dummy[+shadow-name]...] | [int-expr]

Figure 1: BNF formula for new distribution rules

All references to distributed arrays in *int-range-expr* must be accessible (the element belongs to extended local part) for the corresponding element of the template (a search of template elements is performed in its local part and specified shadow edges). If according to derived rule the same element should be distributed at once on several processors, then the DVM system selects one of them where the element will be distributed actually, and adds it in shadow edge on remaining processors with "overlay" name. There should not be elements not distributed on any processor. Such cases are runtime errors and cause the program abnormal termination. Calculated nonexistent indexes of distributed array are ignored without error issue.

Overlay is introduced for possibility of coordinated distribution of a grid elements. For example, there are cells, edges, vertexes. In this case there is an opportunity to build one distribution on the base of another, and in any sequence.

As a result of such distribution an array has two types of element indexing: global (it is initial in a serial program) and local. Local indexing is continuous within one processor, i.e. there is such order of local elements that their local indexes fill fully some integer segment [Li, Hi].

Also by-element shadow edges are introduced. The shadow edge is a set of elements, not belonging to the current process (the requirement to belong to adjacent process is removed), for which, at first, an access from any point of the program is possible without special specifications, and, secondly, special tools of operation with them are introduced: updating by shadow_renew specification, the expansion of parallel loop by shadow_compute specification, etc.

Unlike traditional, by-element shadow edges are added to templates during the program execution and have name to refer to them. They are specified practically as well as derived distribution, see Figure 2.

shadow-add ::= **shadow_add** (templ-name [shadow-axis]... = shadow-name) [**include_to** (var-name-list)]
shadow-axis ::= [ ] | [derived-elem-list **with** derived-templ]

Figure 2: BNF formula for specification of by-element shadow edges

Exactly one of *shadow-axis* should be not empty. The all arrays from the list specified in **include_to** should be aligned with a template to which dimension the shadow edge is added. As a result of such directive execution the shadow edge is added to the template and it is included in specified distributed arrays. After this action shadow elements of the arrays are available for reading from the program, and also can be renewed by shadow_renew directive.

To implement by-element shadow edges and derived distribution the compiler generates a special function according to specified expressions. The parameters to bypass a local part of the template are passed to the function by the runtime system. This function, bypassing the template, fills the buffer of element indexes according to expressions in left part of mapping rule, and then returns it back to the runtime system. Then the buffer is analyzed by the runtime system.

For experimental use of these possibilities the auxiliary directive of the localization of index array values was introduced. It modifies the values of the integer array, replacing global indexes of specified target array by local ones (see Figure 3).

localize-spec ::= **localize** (ref-var-name =>
target-var-name [axis-specifier] ...
axis-specifier ::= [ ] | [:]

Figure 3: BNF formula for directive of localization of index array values

After such operation performing it becomes possible to use available method of the parallel loops compilation: they will be executed wholly in local indexes.

Together with modification of the directive of shadow exchanges and implementation of exchanges for by-element shadow edges (that now are performed not only between adjacent processors, but with arbitrary subset of processors) this set of extensions allows to parallelize and launch the applications on irregular grids on a cluster with accelerators.

The experimental application was parallelized: two-dimensional task of heat conduction in hexahedron (explicit and implicit schemes).

This application performs calculations both in grid vertexes and in its cells. It requires the coordinated distribution of arrays, the localization of index arrays, by-element shadow edges.

The results of this application execution on different number of CPUs and GPUs on the K-100 cluster are shown in Tables 1-4.

Table 1: Parallel performance of the DVMH programs on CPU, time in seconds

Scheme	Serial	Parallel execution, CPU cores				
#nodes		2	4	12	24	96
Explicit	174	87.2	50	21.7	11.1	2.75
Implicit	928	728	611	309	155	42.8

Table 2: Speedup of the DVMH programs on CPU, times

Scheme	Serial	Parallel execution, CPU cores				
#nodes		2	4	12	24	96
Explicit	1	2	3.49	8.02	15.7	63.3
Implicit	1	1.27	1.52	3	6	21.7

6 Conclusions

The steps to extend the DVM system possibilities to solve the problems using irregular grids were made. They are a possibility of manual data distribution and new experimental constructions of the CDVMH language and their implementation in the compiler and the runtime system allowing to parallelize applications on irregular grids on a cluster with multi-core processors and GPUs.

In the future it is supposed to extend new tools and to improve their integration with the DVMH programs with block distributed data.

Table 3: Parallel performance of the DVMH programs on GPUs, time in seconds

Scheme	Serial	Parallel execution, GPUs				
#nodes		1	2	6	12	24
Explicit	174	7	3.81	1.5	0.87	0.55
Implicit	928	77	42.3	16.3	9.64	6.55

Table 4: Speedup of the DVMH programs on GPUs, times

Scheme	Serial	Parallel execution, GPUs				
#nodes		1	2	6	12	24
Explicit	1	24.8	46	116	200	316
Implicit	1	12	22	57	96	142

References

[1] Kozubskaya T.K. Development of Models and Methods of Increased Accuracy for Numerical Analysis of Applied Aeroacoustic Problems. PhD thesis. Moscow, 2010. `http://oldvak.ed.gov.ru/common/img/uploaded/files/vak/2010/announcements/fiz_mat/02-08/KozubskayaTK.pdf` (accessed: 15.10.2017)

[2] Andrianov A.N., Efimkin K.N. Approach for Parallel Implementation of Numerical Methods on Unstructured Grids. Numerical Methods and Programming. 2007, vol. 8, P. 6-17. `http://www.mathnet.ru/links/8fcec8ded0a90db15452ca1f7fa4c661/vmp504.pdf` (accessed: 15.10.2017)

[3] Mahesh Ravishankar, John Eisenlohr, Louis-Noel Pouchet, J. Ramanujam, Atanas Rountev, P. Sadayappan. Code Generation for Parallel Execution of a Class of Irregular Loops on Distributed Memory Systems. SC12, November 10-16, 2012. `http://web.cse.ohio-state.edu/presto/talks/sc12.pdf` (accessed: 15.10.2017)

[4] Lizandro Solano-Quinde, Zhi Jian Wang, Brett Bode, Arun K. Soman. Unstructured grid applications on GPU: performance analysis and improvement. Proceedings of

the Fourth Workshop on General Purpose Processing on Graphics Processing Units, 2011. https://dl.acm.org/citation.cfm?id=1964197 (accessed: 15.10.2017)

[5] Bakhtin V.A., Klinov M.S., Krukov V.A., Podderyugina N.V., Pritula M.N., Sazanov Y.L. Extension of DVM Model of Parallel Programming for Clusters with Heterogeneous Nodes. Bulletin of the South Ural State University, Series Mathematical Modelling, Programming and Computer Software. No 18 (277), issue 12, 2012, P. 82-92. http://www.mathnet.ru/links/9427d06b65f8b991ede2acf397a0661d/vyuru59.pdf (accessed: 15.10.2017)

[6] Zachary DeVito, Niels Joubert, Francisco Palacios, Stephen Oakley, Montserrat Medina, Mike Barrientos, Erich Elsen, Frank Ham, Alex Aiken, Karthik Duraisamy, Eric Darve, Juan Alonso, Pat Hanrahan. Liszt: A Domain Specific Language for Building Portable Mesh-based PDE Solvers. International Conference for High Performance Computing, Networking, Storage and Analysis (SC11), 2011. http://www-cs-faculty.stanford.edu/~zdevito/a9-devito.pdf (accessed: 15.10.2017)

[7] Vasilevskiy Y.V., Konshin I.N., Kopytov G.V., Terekhov K.M. INMOST Programming Platform and Graphical Environment for Development of Parallel Numerical Models on Various Grids. Textbook. Moscow, Publishing of the Moscow State University, 2013. 144 p.

[8] Gorobets A.V., Sukov S.A., Zheleznyakov A.O., Bogdanov P.B., Chetverushkin B.N. Usage of GPU in Hybrid Two-Level MPI+OpenMP Parallelization on Heterogeneous Systems. Bulletin of the South Ural State University. Vol. 242, No 25, P. 76-86. http://omega.sp.susu.ru/books/conference/PaVT2011/short/152.pdf (accessed: 15.10.2017)

[9] J. Saltz, K. Crowley, R. Mirchandaney, and H. Berryman. Run-time scheduling and execution of loops on message passing machines. J. Parallel Distrib. Comput., vol. 8, pp. 303312, 1990. https://www.researchgate.net/publication/222449945_Run-time_scheduling_and_execution_of_loops_on_message_passing_machines (accessed: 15.10.2017)

[10] DVM system. [online]. http://dvm-system.org/en/