Methods of dynamic tuning of DVMH programs on clusters with accelerators *

V.A. Bakhtin[1,2], A.S. Kolganov[1,2], V.A. Krukov[1,2], N.V. Podderugina[1], M.N. Pritula[1]

Kelsysh Institute of Applied Mathematics RAS[1],
Lomonosov Moscow State University[2]

DVM system is intended for development of parallel programs of scientific-technical calculations in C-DVMH and Fortran-DVMH languages. These languages use single model of parallel programming (DVMH model) and they are extensions of standard C and FORTRAN languages by the parallelism specifications issued in the form of directives to the compiler. DVMH model allows to develop efficient parallel programs for heterogeneous computational clusters with accelerators. Using DVMH model the programmer doesn't use explicit copy operations of the data located in memory of central processor (CPU) or accelerators. For program fragments (regions) which can be executed on accelerators, he specifies input and output data, and also those data which are updated or used out of regions. It allows to select dynamically the devices where the region will be executed, to distribute job between devices according to their performance, to execute repeatedly regions to select optimum configuration. The influence of the listed methods on performance of some tests (from NAS NPB benchmarks) and real applications is shown in the article.

1. Introduction

A lot of computational clusters with accelerators of different architecture attached to their nodes [1] are emerging in recent years. This tendency significantly complicates the programming process due to requirement to know good several programming models and languages (MPI, Pthreads, CUDA or OpenCL) at once. To simplify exploiting of accelerators the programming models and appropriate directive-based extensions for programming languages, such as OpenACC [2], OpenMP 4.x [3] and DVMH [4] have been proposed.

DVMH programming model allows to develop parallel programs for clusters in which nodes NVidia accelerators (hereinafter GPU) and Intel Xeon Phi coprocessors are attached in addition to universal multi-core processors. The model supports use of all listed architectures, both separately and simultaneously in frameworks of one program.

One of the important aspects of functioning of such programming model as DVMH is the question of effective mapping of the source program on all levels of parallelism and heterogeneous computational devices. The important tasks of mapping mechanism are to provide correct execution of all constructions supported by the language on heterogeneous computational devices, loading balance between computational devices, and also a choice of optimum method of each code fragment execution on this or that device.

Some optimizations applied to DVMH program during its execution on clusters with accelerators are considered in the article.

2. DVMH model. Scheme of DVMH program execution.

DVMH program is the program in Fortran-DVMH [5] language or the program in C-DVMH language.

When a programmer writes DVMH program he represents a target computer as a multidimensional grid of virtual processors on which data and computation will be mapped. A number of distributed dimensions of the arrays depens on this grid rank (a number of its dimensions). The sizes of each dimension may not be specified in the program, but are

* The research is performed with financial support of grants of RFBR No. 13-07-00580, 14-01-00109.

specified when the program is launched. MPI processes are considered as virtual processors. One or several MPI processes can be mapped on the same node of cluster and each of the processes uses multi-core processors and accelerators of the node.

At first in the program it is required to specify arrays which should be distributed on virtual processors (distributed data). The distribution can be performed by blocks of different sizes, according to distinctions in computation times of different array elements, and also differences in virtual processor performance. The other variables (distributed by default) are mapped by one copy per each virtual processor (replicated data).

The computations are distributed by their mapping on the distributed arrays.

The model allows to specify several levels of parallelism:

- parallel execution of computations (of the program fragments issued as parallel tasks) on different sections of virtual processor grid. For example, it is used in the implementation of multi-block methods;
- parallel execution of distributed loops - execution of iterations of tightly nested loop by virtual processors of the task. The rules of distribution of the loop iterations over the processors are specified in a declaration of the distributed loop;
- parallel execution of the loops on a shared memory - execution of iterations of a tightly nested loop on the computational devices used by MPI process.

Parallel execution of the loops is possible also when there are dependences between the loop iterations. It can be regular dependence on array elements, dependence on reduction variables, dependence on privatized variables. All such dependences should be specified.

Virtual processor can be heterogeneous – it can consist of multiprocessor and accelerators. Each accelerator has its own RAM memory. Only the regions specified by the programmer can be executed on the accelerators. The regions are in a special way issued code fragments with one entrance and one exit, consisting of serial statement groups and parallel loops.

DVMH program execution begins from execution of the initial task on all virtual processors simultaneously. This task can launch a group of child subtasks that will be executed on the provided for them sections of virtual processor grid. After completion of all launched child tasks the initial task continues its execution and can launch the same or other group of child tasks. During execution of any task the regions can be launched. They will be executed on the subset of accelerators provided for them or on the multiprocessor. The task is executed on multiprocessors before entering in the region and after exit from it. The iterations of the distributed loops of the task are distributed between the virtual processors provided for the task. Entering in the computational region each process independently performs additional distribution of its data: in addition to distribution between processes the data are distributed on the computational devices provided for the region execution. At this stage dynamic planning is performed to balance loading and minimize time overheads on data movement. The method of the loop part processing called a handler and also optimization parameters for this handler are selected for each device. To minimize execution time on each separate device the dynamic tuning of optimization parameters, including amount of threads for the handlers on the multiprocessor, and also the size and a form of thread block for CUDA handlers is performed.

During execution the parallel loop inside the region is split into several parts, each of them is processed by some handler on some computational device. The main information for DVMH program performance debugging is accumulated on this stage.

When exiting the computational region additional information for the purposes of dynamic planning of the region execution is accumulated. Also comparative debugging can be performed to check a correctness of the results obtained on accelerators.

Such scheme of DVMH program execution becomes possible since there are no explicit operations of data coping from CPU memory to memory of accelerators (and back) in the program. Using DVMH model a programmer specifies what data are input and output for each region, and also defines data which are updated or used out of regions. It allows to change

dynamically devices, the region is executed on, to distribute job among devices according to their performance and to execute regions repeatedly to select an optimum configuration. It is important facility of DVMH model which favourably distinguishes it from OpenACC and OpenMP 4.x models.

3. Methods of dynamic tuning of DVMH programs

One of DVM system advantages is that obtained parallel programs can be tuned dynamically at startup on provided for their execution resources (quantity of cluster nodes, cores, accelerators and their performance).

Below main methods of dynamic tuning of DVMH programs used in DVM system are considered.

3.1 Mapping of subtasks on cluster nodes according to their performance

In DVM model [6-7] for each subtask a user explicitly specified a set of virtual processors where the subtask should be executed (**Fig. 1**).

```
! description of an array of virtual processors
!DVM$ PROCESSORS P(NUMBER_OF_PROCESSORS( ))
! description of the array of tasks
!DVM$ TASK MB (2)
! determining of a number of virtual processors
NP = NUMBER_OF_PROCESSORS( ) / 2
! explicit mapping of the tasks on the virtual processors
!DVM$ MAP MB( 1 ) ONTO P( 1 : NP )
!DVM$ MAP MB( 2 ) ONTO P( NP+1 : 2*NP )
```

Fig. 1. Parallel tasks in DVM model

As a result of such mapping each of two subtasks will be executed on the section containing a half of all nodes of a cluster.

Thus, in DVM model the distribution of the subtasks was made manually, and it was often complicated due to:

- large number of subtasks;
- significant difference in their complexity;
- neccessity to perform the distribution on different number of nodes.

DVMH model hasn't this disadvantage. In DVMH runtime system the algorithm of automatic distribution of virtual processors between subtasks is implemented.

The constructions to support automatic distribution of subtasks have been added in C-DVMH and Fortran-DVMH languages. The programmer should to specify the relative complexity of the subtasks, minimum and maximum number of virtual processors for each subtask, and also the parameters of a dependence of each subtask runtime on a number of provided virtual processors (the parts of parallel computations in the total volume of computations for the task).

Two modes of subtask distribution are possible when multi-block DVMH program is launched:

- one accelerator is selected as a logical processor for distribution algorithm and thus loading on each accelerator is balanced;
- whole node of a cluster is selected as the logical processor for distribution algorithm and thus loading on each accelerator is balanced, and during subtask execution all computational devices of the nodes are used according to general mechanisms of data and computation distribution.

3.2 Mapping of arrays and loops on cluster nodes according with their performance

One of possible modes to use clusters with Intel Xeon Phi coprocessors is so-called "symmetric mode". In this mode the central processor and the coprocessor can be considered as separate computational nodes of the cluster. The same program is compiled separately for CPU and separately for the coprocessor. The compiled programs are launched simultaneously on the coprocessor and CPU and can be synchronized using MPI technology.

When this mode is used it is necessary to balance loading of MPI processes executed on different devices. For MPI programs, as a rule, this problem is solved by an implementation of one more level of parallelism with use of threads (Pthreads or OpenMP). Different number of MPI processes is launched on CPU and the coprocessor, the arrays are distributed on MPI processes by equal chunks, but due to use of different number of threads it is possible to balance their loading.

Such mechanism of balancing can be used also for DVMH programs. There are environment variables which allow to set a number of MPI processes launched on different nodes of a cluster and number of threads for different MPI processes:

```
export DVMH_PPN='2,1,1' # Number of processes per node
export DVMH_NUM_THREADS='8,8,240,240' # Number of CPU threads per process
```

Moreover, the possibilities to set performances of processors (or their automatic determining at the program startup) and to take them into account during data and computation distribution on the cluster nodes are implemented in DVM system. This possibility was implemente d in 2002 and allowed to efficiently execute DVM programs on heterogeneous clusters [8].

3.3 Mapping of arrays and loops on the node devices according with their performance

At present in DVMH programs data are distributed by blocks: each distributed dimension is divided into segments by several points. Along each dimension of the distributed array it is possible to specify either a distribution by equal blocks, or distribution by weighed blocks, i.e. according to the given vector of weights. These directives are immediately performed during data distribution on MPI processors. The nested distributions appearing when entering the computational region are built with use of this information, but, because of heterogeneity of computational devices, can have distribution scheme differing from external one.

DVMH runtime system supports three modes of data and computation distribution on computational devices in points of entry to regions:

1. Simple static mode.
2. Dynamic mode with selection of distribution scheme.
3. Static mode with use of selected distribution scheme.

Consider these distribution modes in more details.

In the simple static mode the distribution is performed identically in each region. Using environment variables a user specifies a vector of weights of computational devices available in each node of the cluster (or they can be roughly determined automatically). Then these weights are superimposed on parameters of external data distribution. In this mode the data movements due to their redistribution are minimized, but different ratio of performances of computational devices in different code fragments isn't considered.

The selection modes are based on the assumption that a program is iterative i.e. the same computations are repeated large number of times (in respect of performed operations, but not calculated values). In the dynamic mode with selection of distribution scheme in each region the distribution is selected on the basis of permanently replenished history of starts of the region and its neighbors.

During execution DVMH programs keep processing times of all parallel loops on used devices, supporting in actual state a table function of the performance dependence (in iterations per a second) on the volume of computations (in iterations). After each execution of the loop this dependence is updated - new measurement replaces old ones, being in the specified neighborhood. So, in case of quite smooth (in time) changing of iteration weight, the function will remain actual, at least, in the neighborhood of the points corresponding to the current starts of the loop.

The existence and accumulation of this information enables to optimize distribution weights.

Formally, the problem is formulated as follows:

There is a set of computational devices D1,D2,D3,...,DN.

There is a set of loops L1,L2,L3,...,LM.

For each loop there are known usual number of iterations, number of entrances, the dependence of execution time on a particular device for specified number of the loop iterations.

It is required to determine a vector of weights W1,W2..., WN, sum {Wi} =1.0, such that it minimizes time functionality T=sum { Li.execCount * max { Li.Time (Dj, Li.typicalPart * Wj); 1 <= j <= N};1 <= i <= M}, where execCount - a number of the loop executions; typicalPart - average quantity of the loop iterations; Time - time function of the computational device and quantity of iterations.

This problem is solved approximately by modification of an algorithm of gradient descent.

The built distribution scheme can be kept in the file and used in the next starts of the program. The vector of weights of computational devices can be used as initial approximation, as for the simple static mode.

The transition from this mode to the third mode - static mode with use of selected distribution scheme - is possible in any point of the program execution. In this mode in each region the distribution is selected on the basis of the provided distribution scheme built during the program execution in the second mode and there is an possibility as to transfer to this mode directly from the second one, as to use the distribution scheme from the file obtained during the program execution in the second mode. If the distribution scheme from the file is used its correct applying isn't guaranteed if parameters of the program were changed. In particular it concerns those parameters which affect a way of the program execution (the choice of other calculation method, disable or enable the calculation stages).

3.4 Transformation of arrays

One of advantages of DVMH programs is that access to arrays in the parallel loops executed on GPU are programmed in generalized form that allows to have other order of the array dimensions during execution than it was specified in the source program. For loops with regular data dependencies more complex reordering - diagonalization - is supported at the level of array elements.

This feature gives one more degree of freedom for optimization during execution - a selection of the array representation during execution of a concrete loop.

For implementation of these possibilities the DVMH runtime system keeps the current representation of the array on all computational devices in the form of dimension reordering and existence and method of diagonalization. There are two forms of diagonalization: parallel to the anti-diagonal or parallel to the main diagonal.

Before execution of each loop its mapping rule is analyzed and compared with alignment rules of the arrays used in the loop. This information allows to match the loop dimensions with the array dimensions and to determine optimum for this loop form of the array representation in the computational device memory. After the optimum form had been determined, the transformation from the current representation to new one is performed by the following rules:

1. If the current form is not the same, as new one, then:

1.1. If the current form is diagonalized then to undiagonalize it => updated current form.

1.2. If the current form has other order of dimensions, than it is required, then to calculate a quotient of reorderings (difference) and to apply to the current representation => updated current form.

1.3. If the new form requires a diagonalization, then to apply required diagonalization to the current representation => updated current form.

2. End. The current form was transformed to new required.

The icon "=>" means that as a result of this action the current form of the array representation is changed and the next steps are already applied to it as to the current form.

So flexible mechanism of automatic reordering of arrays in the memory of computational devices allows to accelerate considerably some loops. For example, in case of the implementation of a method of alternating directions in which the array is located in GPU memory at the same manner during all runtime (the order of dimensions isn't important) one of the loops will be executed approximately 10 times slower than others because of impossibility to joint an accesses to global memory, requested by threads of one warp. If during runtime the array is able to change its representation in the memory, then it will be possible to make all loops of the program equally fast.

3.5 Use of dynamic compilation

This mechanism allows to compile CUDA handlers generated by Fortran-DVMH or C-DVMH compilers during the program execution.

To implement this possibility there was used new library NVidia Run Time Compilation (NVRTC) [9] which allows to compile a code for GPU on cluster nodes in the moment of the program execution. The library is available in CUDA Toolkit 7.0.

NVRTC is dynamic compilation library for CUDA-C++ programs. It gets the source CUDA code as a character line and creates special objects that are used further for creation of PTX code and loading of the obtained code on GPU.

This approach can promote to increase performance because all actions are performed during the program execution when many data are already known that is impossible in case of static compilation.

To use this approach CUDA programmers should to transform CUDA kernels to character lines, to add compilation and the call of kernels using low level CUDA Driver API [10] that significantly complicates not simple debugging of GPU code, worsens a convenience of its development and further support.

In DVMH compiler this possibility is implemented in DVMH runtime system. To enable the possibility the optimization option -rtc was added in DVMH compilers. The programmer can use both old version of the program (without dynamic compilation), and new version of the program (with dynamic compilation), changing compilation options.

There are the following advantages of dynamic compilation usage.

1) During the program execution the values of scalar variables (int, double, long types, etc.) become known. Thereby, before CUDA kernel compilation it is possible to execute code transformation as follows: instead of a passed parameter, for example, int param_1 to declare in CUDA kernel body the variable const int param_1 = dyn_value where dyn_value is the value of this variable, known at compilation time. As a result used by GPU resources (for example, registers) are reduced that can lead to increasing of this kernel performance.
2) During static compilation it is necessary to specify the architecture of GPU (for example, Fermi, Kepler, Maxwell). If different generations of GPU are used in one node it can affect negatively on performance because old generations don't support new possibilities of the modern GPU. If dynamic compilation is used in DVMH compiler there is no such problem because the compilation is performed for target GPU (taking into account architectural features) at execution time.

3) DVMH compiler generates several CUDA kernels and CUDA handlers for a parallel loop execution. During static compilation it is necessary to compile all CUDA kernels regardless of that, they will be executed or not. During dynamic compilation only those from the kernels will be compiled that should be executed on GPU. Thereby, the time of the program compilation can be reduced.

4. Use of methods of dynamic tuning of DVMH programs

To estimate an affect of different optimizations applied during DVMH program execution on clusters with accelerators DVMH versions of NAS tests (BT, EP, SP and LU) from NPB 3.3.1 benchmarks [11] and real user applications (Composite, Cavity and Container programs) were used.

Composite program is intended for modeling of process of heat distribution in composite material. Cavity and Container [12] programs solve the problem of flow of incompressible fluid or weakly compressible gas near rectangular recess in which the hyperbolic version of quasi-gasdynamic system is used as source mathematical model (two-dimensional and three-dimensional problem definition).

Testing was performed on MIC server with attached to it 6-core (12 flow) Intel Xeon E5-1660v2 processor with 48 GB of DDR3 RAM, the Intel Xeon Phi 5110P coprocessor with 8 GB of GDDR5 RAM and NVidia GTX Titan GPU with 6 GB of GDDR5 RAM, and also on hybrid supercomputer K100 [13].

4.1 Mapping of subtasks on cluster nodes according to their performance

To demonstrate this possibility Composite program developed in Fortran-DVMH language was used. In this application the computational physical area is divided into 36 blocks, for each of them the rectangular grid of different detailing is introduced because the required detailing of calculations is various in different blocks of computational area. Computational complexity of one block processing has a range more than 30 times. Efficiency of this application parallelization was estimated on K-100 cluster. 1, 2, 4, 8, 12, 24, 36, 48, 60, 72, 84, 96, 108, 120, 132, 144 processors were used.

The researching of parallelization efficiency showed:

1. At each step of processor quantity increase the application was accelerated, including the cases, when the number of processors was more than subtasks quantity.
2. When 4 processors are used the acceleration is increased 2.4 times only. Parallelization efficiency on 4 processors is 60%.
3. Since 8 processors parallelization efficiency (in relation to the result on 8 processors) doesn't fall lower than 87%. In comparison with serial version - it isn't lower than 47%.
4. The maximum reached acceleration is 68 times in comparison with the serial program.

The obtained results show that the mechanism of subtask distribution implemented in DVM system copes with the task of the application acceleration even if the number of processors exceeds a number of subtasks by four times.

4.2 Mapping of arrays and loops on cluster nodes according to their performance

An acceleration of EP test on the MIC server in comparison with the serial version of this program executed on one core of CPU is shown in **Fig. 2**. This test was executed on different architectures separately, and also in the following combinations: CPU + GPU, CPU + coprocessor and coprocessor + CPU + GPU. The cases marked by red and magenta colors are the cases when additionally loading balance was used by specifying of the weight ratio of all CPU and GPU cores and the weight ratio of MPI processes mapped on CPU and the coprocessor.

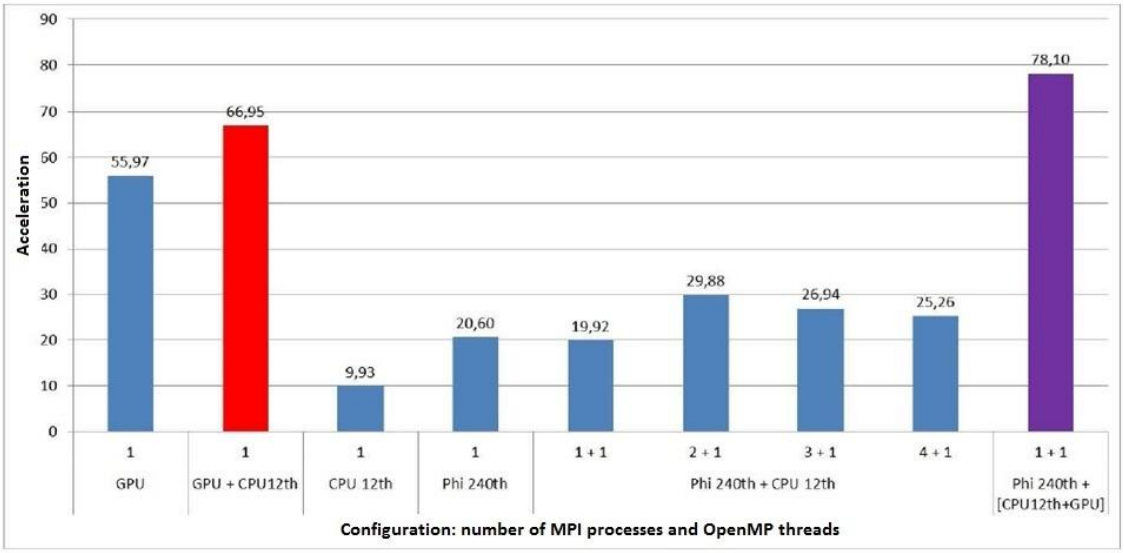

Fig. 2. Use of loading balance in DVMH program on the example of EP test, class C

As a result, on this test in case of simultaneous use of GPU and CPU it was succeeded to reach performance which is 17% more than performance of the test execution only on GPU.

When the coprocessor and CPU were used simultaneously it was succeeded to reach performance which is 30% more than the test performance on one coprocessor. The most favorable ratio of MPI processes is following: two MPI processes on Xeon Phi and one MPI process on Xeon E5 and MPI processes have equal weights, or one MPI process on CPU and on the coprocessor, and ratio of the process weights is 1: 2 correspondently.

In case of simultaneous use of all devices of the node the flexible mechanism of balancing implemented in DVM system allowed to obtain the performance which is 28% more than performance of EP test on GPU and 3.7 times exceeds performance of this test on the coprocessor.

4.3 Mapping of arrays and loops on node devices according to their performance

As it was already mentioned, several modes of data and computation distribution on computational devices are implemented in DVMH runtime system. The mode is set by DVMH_SCHED_TECH environment variable. Existence of such possibility allows to determine automatically the best scheme of mapping and essentially improve the performance of DVMH programs due to use of all resources of a node. Execution times of Cavity (1600x1600, 200 iterations) and Container (150x150x150, 100 iterations) programs on MIC server, K-100 computer system and a test server where "budgetary" graphic card is used are shown in table 1. The dynamic mode with selection of distribution scheme was used (DVMH_SCHED_TECH=2).

Table 1. Automatic distribution of job between node devices

Program	Execution time, Sec.		Combination of job (GPU+CPU)			
	GPU	CPU	CPU weight	GPU weight	Time1, s	Time2, s
Server MIC	GTX Titan	E5-1660 (6 threads)				
Cavity	13.37	41.91	0.256	0.743	12.81	12.43
Container	27.44	60.12	0.268	0.732	23.30	22.72
K-100	C2050	X5670 (12 threads)				
Cavity	16.87	9.76	0.350	0.649	15.21	14.72

Container	30.12	56.24	0.291	0.708	28.31	24.42
Test server	GTX 550Ti	i7-980X (12 threads)				
Cavity	45.59	55.87	0.532	0.468	31.21	30.85
Container	72.25	78.61	0.462	0.537	45.23	43.88

During DVMH program execution the table function of performance dependence (in iterations per second) on the volume of computations (in loops) is built for all parallel loops on all used devices. The fragment of this table is shown in **Fig. 3**.

```
Performance statistics for parallel loop at cavity.fdv(601):
    Device #0:
        Handler #0:
            Best parameters: threads=11
            Table function (iterations => performance):
                      894400 => 2.21928e+08
                       1e+06 => 2.20961e+08
                    2.56e+06 => 2.22738e+08
    Device #1:
        Handler #0:
            Best parameters: thread-block=(32,8,1)
            Table function (iterations => performance):
                    1.56e+06 => 6.3996e+08
                  1.6656e+06 => 6.44597e+08
                    2.56e+06 => 6.74815e+08
Simple dynamic run performances:
DVMH_CPU_PERF='0.35016'
DVMH_CUDAS_PERF='0.64984'
```

Fig.3. Dependence of performance on the volume of computations

Existence and accumulation of such information allows to determine optimum distribution weights. CPU and GPU weights shown in table 2 were determined automatically. The selection of optimum distribution scheme demanded to execute 3-7 iterations for these programs. Time1 is the program runtime on a node in case of simultaneous use of all cores of CPU and one GPU in dynamic selection mode. Time2 is the program runtime when automatically determined weights of distribution were used in the program restart from the beginning.

Use of CPU cores allowed to accelerate Container program by 1.2 times (1.6 times on old GPU) in comparison with startup only on GPU.

4.4 Transformation of arrays

The acceleration of BT, SP and LU tests on 1 GPU is shown in **Fig. 4**. The mechanism of automatic array reordering in the memory of computational devices is used. Use of compilation option –autoTfm enabling this mode allows to accelerate significantly the programs (for example, for LU test, class C, on Kepler architecture an acceleration more than 2.7 times was obtained, in comparison with the program execution without such reordering). It is necessary to note that for the array transformation the additional memory on GPU may be required and if such reordering is performed very often, then overhead costs of its performing can exceed an effect of the optimization therefore this mechanism isn't used by default.

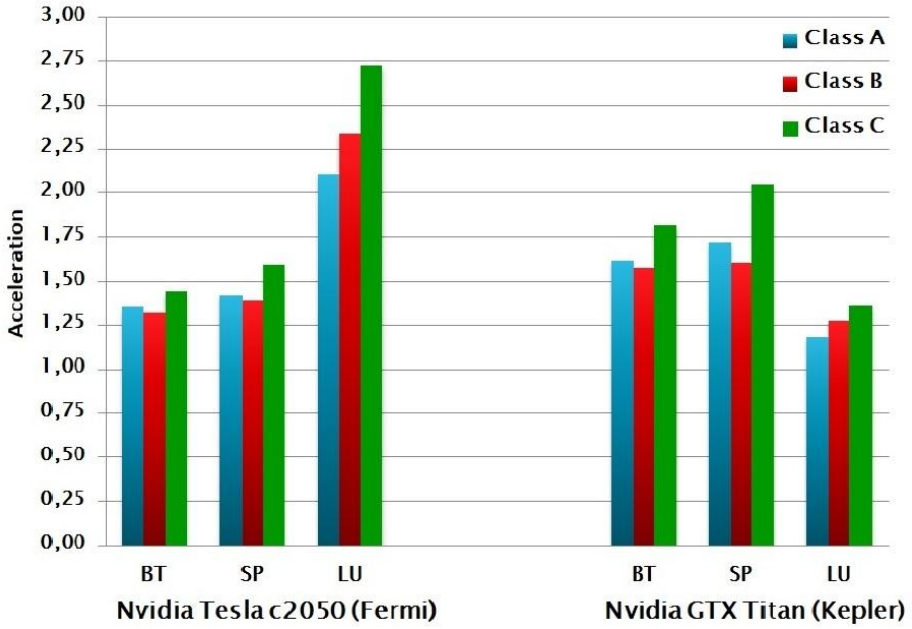

Fig. 4. An acceleration of NAS tests as a result of automatic reordering of arrays in memory

4.5 Use of dynamic compilation

The advantages of dynamic compilation can be demonstrated on SP and LU tests. It is favorable to use this optimization in iterative methods when the same parallel loop can be executed many times. At first execution of the loop, tuning and compilation of corresponding CUDA kernel are performed in DVMH runtime system. At repeated execution of the loop the compilation isn't performed, if parameters of CUDA kernel are the same.

Table 2. Characteristics of SP test launch

	Number of registers			Execution time, ms		
	Was	Became	Reducing	Was	Became	Acceleration
compute_rhs	125	82	52%	74	66.5	11%
x_solve	153	112	37%	64.5	42.9	50%
y_solve	132	107	23%	64.4	42.6	51%
z_solve	128	107	20%	53	48	10%

Table 3. Characteristics of LU test launch

	Number of registers			Execution time, ms		
	Was	Became	Reducing	Was	Became	Acceleration
Loop ACROSS+	148	35%	110	0.22	0.14	57%
Loop ACROSS–	141	110	28%	0.22	0.14	57%
Loop	120	89	35%	125.4	126.5	-1%

The results of procedure execution without applying of dynamic compilation and with its applying are shown in tables 2 and 3. The programs were compiled with use of CUDA Toolkit 7.0 for Kepler architecture. The computational grid 250*250*250 was used for SP test, and for LU the grid 270*270*270 was used. Each of grids required approximately 4 GB of GPU memory.

It is visible that due to decreasing of a number of used registers it is possible to improve performance up to 50%. But the program is decelerated in some cases. It may be explained by the fact that the current version of NVRTC library is experimental. It is worth note that in LU test first two loops have data dependencies and for execution of one parallel loop multiple

launches of the same handler are performed. In table 3 the average time of such handler execution is shown (there were about 20800 starts of this loop on 20 iterations).

As a result of this optimization total performance of SP program was increased by 28%, and LU program - by 36%.

Conclusions

The system of automation of parallel program development (DVM system) significantly simplifies a process of parallel program development for hybrid computational clusters. The obtained DVMH programs without any changes can be executed effectively on the clusters of different architecture using multi-core universal processors, graphic accelerators and Intel Xeon Phi coprocessors. It is reached as due to different optimizations which are performed statically during DVMH program compilation [14-16], and due to dynamic optimizations which were considered in this article.

At startup parallel programs can dynamically be tuned on the provided for their execution resources (quantity of cluster nodes, cores, accelerators and their performance). Such property of the programs allows to launch them on arbitrary configuration (a number of processors, threads, accelerators), to combine complex programs from available simple programs, and also to increase efficiency of usage of parallel systems of collective use due to more flexible distribution of resources between separate programs.

References

1. Top500 List – November 2014. URL: http://top500.org/list/2014/11/ (accessed 12.06.2015).
2. OpenACC. URL: http://www.openacc-standard.org/ (accessed 12.06.2015).
3. OpenMP 4.0 Specifications. URL: http://openmp.org/wp/openmp-specifications/ (accessed 12.06.2015).
4. Bakhtin V.A. Rasshirenie DVM-modeli parallel'nogo programmirovaniya dlya klasterov s geterogennymi uzlami [Extension of the DVM parallel programming model for clusters with heterogeneous nodes] / Bakhtin V.A., Klinov M.S., Krukov V.A., Podderyugina N.V., Pritula M.N., Sazanov Y.L.. // Vestnik Yuzhno-Ural'skogo gosudarstvennogo universiteta, seriya "Matematicheskoe modelirovanie i programmirovanie". 2012. No. 18 (277), P. 82–92.
5. Description of Fortran-DVMH language. URL: http://dvm-system.org/static_data/docs/FDVMH-user-guide-en.pdf (accessed 12.06.2015).
6. Konovalov N.A., Krukov V.A., Mihailov S.N., Pogrebtsov A.A. Fortran DVM – yazyk razrabotki mobil'nyh parallel'nyh programm [Fortran DVM – a language for mobile parallel programs development]. // Programmirovanie [Programming]. 1995. No. 1. P. 49–54.
7. Konovalov N.A. C-DVM – yazyk razrabotki mobil'nyh parallel'nyh programm [C-DVM – a language for mobile parallel programs development] / Konovalov N.A., Krukov V.A., Sazanov Y.L. // Programmirovanie [Programming]. 1999. No. 1. P. 54–65.
8. Krukov V.A. Razrabotka parallel'nyh program dlya vychislitel'nyh klasterov i setei. [Development of parallel programs for computer clusters and networks] / Krukov V.A. // Information technology and computer systems, Moscow: Institute of microprocessor computer systems RAS, 2003, No. 1-2, P. 42-61 URL: ftp://ftp.keldysh.ru/dvm-distr/journ1-2_page42_61.pdf (accessed 12.06.2015).
9. NVIDIA CUDA Runtime Compilation Library. URL: http://docs.nvidia.com/cuda/nvrtc/index.html (accessed 12.06.2015).

10. NVIDIA CUDA Driver API. URL: http://docs.nvidia.com/cuda/cuda-driver-api/index.html (accessed 12.06.2015).
11. NAS Parallel Benchmarks. URL: http://www.nas.nasa.gov/publications/npb.html (accessed 12.06.2015).
12. Bakhtin V.A. Rasshirenie DVM-modeli parallel'nogo programmirovaniya dlya klasterov s geterogennymi uzlami. [Extension of the DVM parallel programming model for clusters with heterogeneous nodes] / Bakhtin V.A., Krukov V.A., Chetverushkin B.N., Shil'nikov E.V. // DOKLADY MATHEMATICS, Moscow: Pleiades Publishing, Ltd, 2011, Vol. 84, Issue 3, P. 879-881
13. Hybrid computational cluster K-100. URL: http://www.kiam.ru/MVS/resourses/k100.html (accessed 12.06.2015).
14. Bakhtin V.A. Otobrazhenie na klastery s graphicheskimi processorami DVMH-programm s regulyarnymi zavisimostyami po dannym [Mapping DVMH-programs with regular data dependencies on clusters with graphics processors] / Bakhtin V.A., Kolganov A.S., Krukov V.A., Podderyugina N.V., Pritula M.N. // Vestnik Yuzhno-Ural'skogo gosudarstvennogo universiteta, seriya "Vychislitel'naya matematika i informatika". 2013. V.2 No. 4, P. 44–56.
15. Aleksahin V.F. Rasparallelivanie na graphicheskie processor testov NAS NPB3.3.1 na yazhyke Fortran DVMH [GPU parallelization of the tests from NAS NPB3.3.1 using Fortran DVMH programming language] / Aleksahin V.F., Bakhtin V.A., Zhukova O.F., Kolganov A.S., Krukov V.A., Podderugina N.V., Pritula M.N., Savitskaya O.A., Shubert A.V. // Vestnik Ufimskogo gosudarstvennogo aviatsionnogo texnicheskogo universiteta, 2015, V. 19, №1(67). P. 240-250.
16. Aleksahin V.F. Rasparallelivanie na yazhyke Fortran DVMH dlya soprocessora Intel Xeon Phi testov NAS NPB3.3.1. [Parallelization of NAS NPB3.3.1 tests in Fortran-DVMH for Intel Xeon Phi coprocessor] Aleksahin V.F., Bakhtin V.A., Zhukova O.F., Kolganov A.S., Krukov V.A., Ostrovskaya I.P., Podderugina N.V., Pritula M.N., Savitskaya O.A. // Proceedings of the international scientific conference "Parallel Computational Technologies (PCT'2015)", Chelyabinsk: Izdatel'skiy centr UUGU 2015, P. 19-30