GERMAN-RUSSIAN CONFERENCE

SUPERCOMPUTING IN SCIENTIFIC AND INDUSTRIAL PROBLEMS

DVM system for parallel programming

April 26, 2018 | Svetlogorsk

Vladimir Bakhtin, Alexander Kolganov, Natalya Podderugina,
Victor Krukov, Michail Pritula

Moscow State University

Keldysh Institute of Applied Mathematics RAS

DVM System

- was developed in Keldysh Institute of Applied Mathematics, Russian Academy of Sciences

- means

  Distributed Virtual Memory

  Distributed Virtual Machine

- includes two programming languages which are the extensions of standard C and Fortran languages by parallelism specifications: *C DVMH* and *Fortran DVMH*

- allows to create efficient parallel programs (DVMH-programs) for heterogeneous computational clusters

DVM programming model

- DVM model is based on special form of data parallelism: single program – multiple data streams (**SPMD**). In the model the same program is executed on all virtual processors, but each processor executes its own subset of statements according to data distribution.

- The programmer defines arrays (**distributed data**) and iterations of the loops, that should be distributed on processors. The distributed arrays are specified by data mapping directives, and parallel loops - by directives of computation distribution.

- Data distribution defines a set of local or **own variables** for each processor. The set of own variables defines the rule of **own computations**: a processor assigns values only to its own variables.

- When a processor calculates value of own variable it may need in values of as own variables as not own (remote) variables. Special directives are used for **remote data** access.

DVMH programming model

- The programmer defines the code fragments which can be executed on accelerators. These fragments are called **computational regions** or simply **regions**. A region may be performed on one or several accelerators and/or on CPU.

- Program fragments out of regions are always executed on the central processor.

- For each region data necessary for its execution (**input**, **output**, **local**) are specified.

- To control data movements between accelerators and the central processor special directives are provided.

DVM System languages

C-DVMH = C 99 language + pragmas
Fortran-DVMH = Fortran 95 language + special comments

- Special comments and pragmas are high-level specifications of parallelism in terms of a sequential program.
- Specifications of a low-level data transfer and synchronization are absent in a source code.
- Programming is accomplished in a sequential style.
- A normal compiler neglects specifications of parallelism.
- The same program is suited for sequential and for parallel execution.

DVMH parallel specifications

- Distribution of array elements on the processors:

  directives distribute / align

- Mapping of the loop iterations on the processors :

  directive parallel

- Organization of the efficient access to remote data located on other processors:

  clauses shadow / across / remote

- Organization of the efficient execution of reduction operations which are global operations on the data located on different processors:

  clause reduction: max/min/sum/maxloc/minloc/…

- Specification of the regions which are special constructions of the DVMH languages. These constructions consist of sequential parts of code and parallel loops. The regions can be executed on the accelerators:

  directive region

- Specification of the actualization directives which control data movement between a memory of CPU and memories of accelerators:

  directives actual / get_actual

DVM System components

- Fortran-DVMH compiler
- C-DVMH compiler
- DVMH runtime system library
- Tools for DVMH program functional debugging
- Tools for DVMH program performance debugging

Jacobi Iteration

```
program Jacobi
   double precision, allocatable, dimension(:,:,:) :: f, newf, r
   …
   allocate(f(mx, my, mz))
   allocate(newf(mx, my, mz))
   allocate(r(mx, my, mz))
   curf = 0
   do n = 1, NITER
      if (curf .eq. 0) then
         eps = dostep(f, newf, r, rdx2, rdy2, rdz2, beta, mx, my, mz)
      else
         eps = dostep(newf, f, r, rdx2, rdy2, rdz2, beta, mx, my, mz)
      endif
      print *, 'Iteration=' , n, 'eps=', eps
      curf = 1 - curf
    enddo
end
```

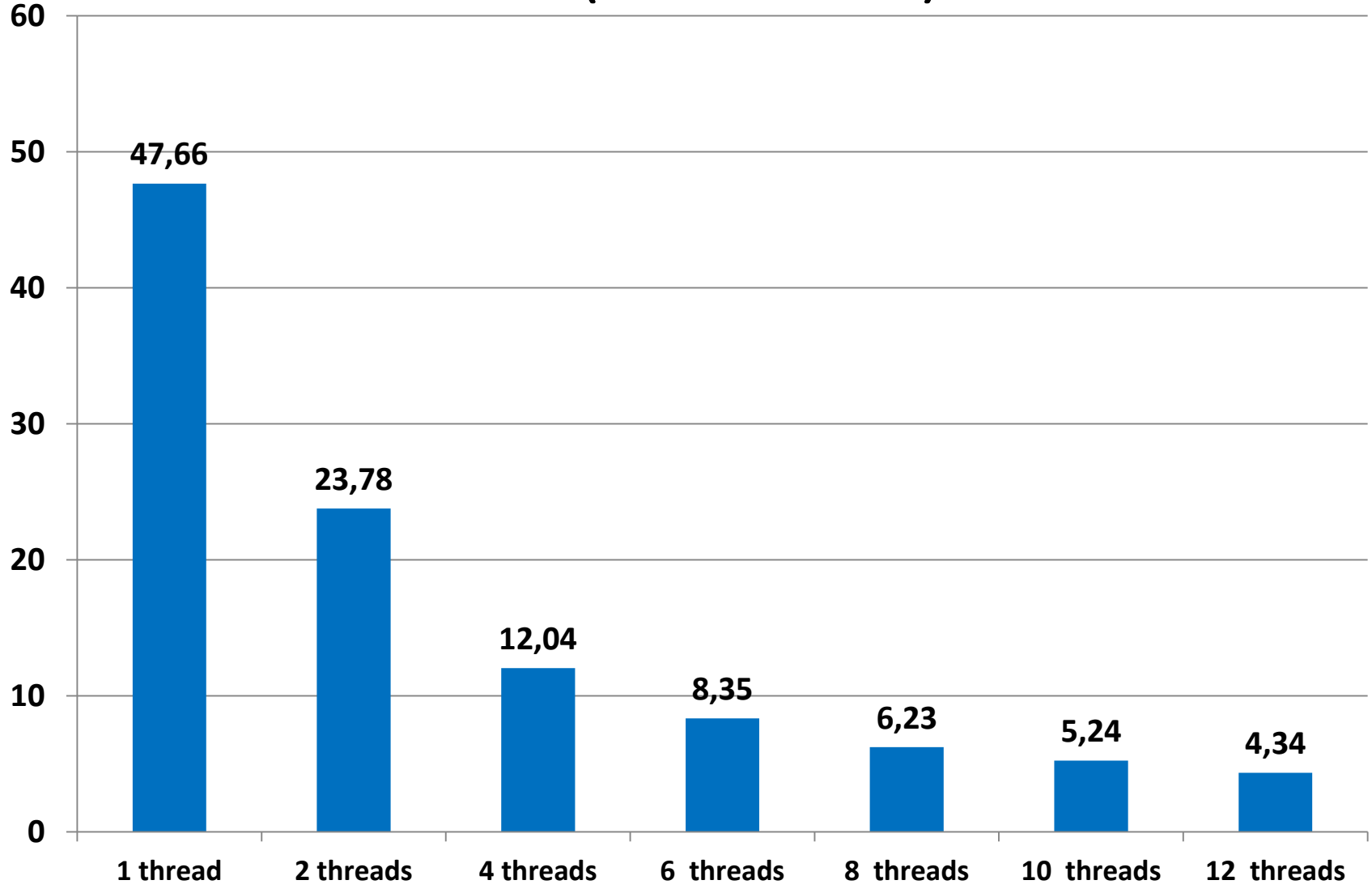
Jacobi Iteration

```
      double precision function dostep(f, newf, r, rdx2, rdy2, rdz2,
&        beta, mx, my, mz)
      integer :: mx, my, mz
      double precision, dimension(mx,my,mz) :: f, newf, r
      double precision :: rdx2, rdy2, rdz2, beta, eps
      integer :: i, j, k
      eps = 0.
      do k = 2, mz - 1
        do j = 2, my - 1
            do i = 2, mx - 1
                newf(i, j, k) = ((f(i-1,j,k)+f(i+1,j,k))*rdx2
&                   +(f(i,j-1,k)+f(i,j+1,k))*rdy2
&                   +(f(i,j,k-1)+f(i,j,k+1))*rdz2
&                   -r(i,j,k)) * beta
                eps = max(eps,abs(newf(i,j,k)-f(i,j,k)))
            enddo
        enddo
      enddo
      dostep = eps
      end function
```

```
      double precision function dostep(f, newf, r, rdx2, rdy2, rdz2,
     &                 beta, mx, my, mz)
      integer :: mx, my, mz
      double precision, dimension(mx,my,mz) :: f, newf, r
      double precision :: rdx2, rdy2, rdz2, beta, eps
      integer :: i, j, k
      eps = 0.
CDVM$ PARALLEL (k,j,i), REDUCTION(max(eps))
      do k = 2, mz - 1
        do j = 2, my - 1
            do i = 2, mx - 1
                newf(i, j, k) = ((f(i-1,j,k)+f(i+1,j,k))*rdx2
     &                          +(f(i,j-1,k)+f(i,j+1,k))*rdy2
     &                           +(f(i,j,k-1)+f(i,j,k+1))*rdz2
     &                          -r(i,j,k)) * beta
                eps = max(eps,abs(newf(i,j,k)-f(i,j,k)))
            enddo
          enddo
       enddo
      dostep = eps
      end function
```

Multi-core version

Execution time in seconds of Jacobi Iteration on Intel Xeon E5-2660 (k10.kiam.ru)



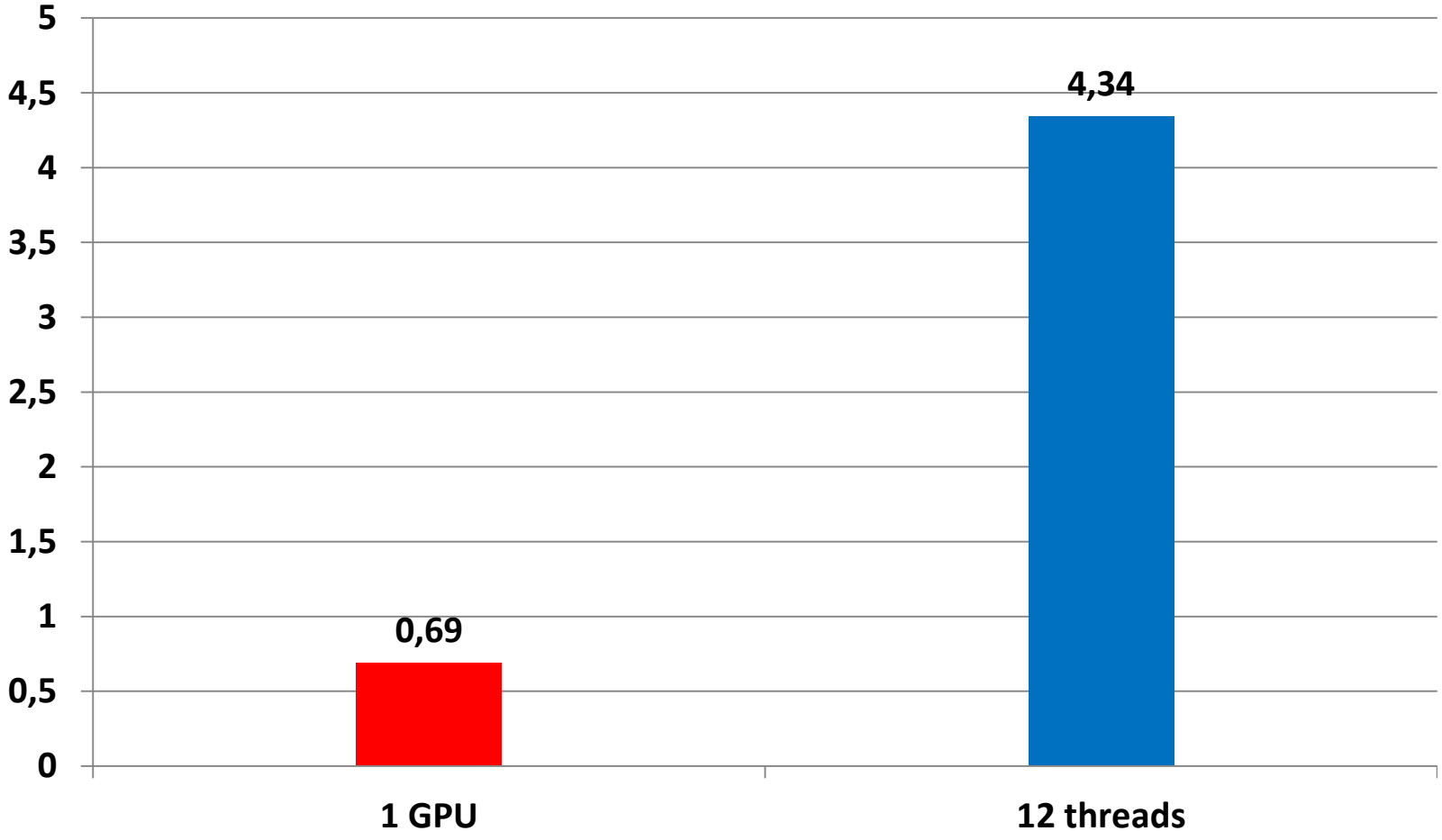
```
      double precision function dostep(f, newf, r, rdx2, rdy2, rdz2,
     &                 beta, mx, my, mz)
      …
      eps = 0.
CDVM$ ACTUAL(eps)
CDVM$ REGION INOUT(f,newf, eps), IN(r,rdx2,rdy2,rdz2,beta)
CDVM$ PARALLEL (k,j,i), REDUCTION(max(eps))
      do k = 2, mz - 1
        do j = 2, my - 1
            do i = 2, mx - 1
                  newf(i, j, k) = ((f(i-1,j,k)+f(i+1,j,k))*rdx2
     &                              +(f(i,j-1,k)+f(i,j+1,k))*rdy2
     &                        +(f(i,j,k-1)+f(i,j,k+1))*rdz2
     &                              -r(i,j,k)) * beta
                  eps = max(eps,abs(newf(i,j,k)-f(i,j,k)))
            enddo
        enddo
      enddo
CDVM$ END REGION
CDVM$ GET_ACTUAL(eps)
      dostep = eps
      end function
```

GPU version

```
      double precision function dostep(f, newf, r, rdx2, rdy2, rdz2,
     &                 beta, mx, my, mz)
      …
      eps = 0.
CDVM$ ACTUAL(eps)
CDVM$ REGION
CDVM$ PARALLEL (k,j,i), REDUCTION(max(eps))
      do k = 2, mz - 1
        do j = 2, my - 1
            do i = 2, mx - 1
                  newf(i, j, k) = ((f(i-1,j,k)+f(i+1,j,k))*rdx2
     &                                +(f(i,j-1,k)+f(i,j+1,k))*rdy2
     &                        +(f(i,j,k-1)+f(i,j,k+1))*rdz2
     &                                -r(i,j,k)) * beta
                  eps = max(eps,abs(newf(i,j,k)-f(i,j,k)))
            enddo
        enddo
      enddo
CDVM$ END REGION
CDVM$ GET_ACTUAL(eps)
      dostep = eps
      end function
```

GPU version

Execution time in seconds of Jacobi Iteration on nVidia Fermi M2090 and Intel Xeon E5-2660



Jacobi Iteration.
Version for cluster with accelerators

```
        program Jacobi
        double precision, allocatable, dimension(:,:,:) :: f, newf, r
CDVM$ DISTRIBUTE (BLOCK, BLOCK,BLOCK) :: f
CDVM$ ALIGN newf(i,j,k) WITH f(i,j,k)
CDVM$ ALIGN r(i,j,k) WITH f(i,j,k)
        …
        do n = 1, NITER
            if (curf .eq. 0) then
                eps = dostep(f, newf, r, rdx2, rdy2, rdz2, beta, mx, my, mz)
            else
                eps = dostep(newf, f, r, rdx2, rdy2, rdz2, beta, mx, my, mz)
            endif
            print *, 'Iteration=' , n, 'eps=', eps
            curf = 1 - curf
        enddo
        end
```

```
      double precision function dostep(f, newf, r, rdx2, rdy2, rdz2…)
CDVM$ INHERIT f,newf,r
       …
      eps=0.
CDVM$ ACTUAL(eps)
CDVM$ REGION
CDVM$ PARALLEL(k,j,i) ON newf(i,j,k),REDUCTION(max(eps))
       do k = 2, mz - 1
          do j = 2, my - 1
              do i = 2, mx - 1
                    newf(i, j, k) = ((f(i-1,j,k)+f(i+1,j,k))*rdx2
     &                              +(f(i,j-1,k)+f(i,j+1,k))*rdy2
     &                              +(f(i,j,k-1)+f(i,j,k+1))*rdz2
     &                              -r(i,j,k)) * beta
                    eps = max(eps,abs(newf(i,j,k)-f(i,j,k)))
              enddo
           enddo
        enddo
CDVM$ END REGION
CDVM$ GET_ACTUAL(eps)
      end function
```

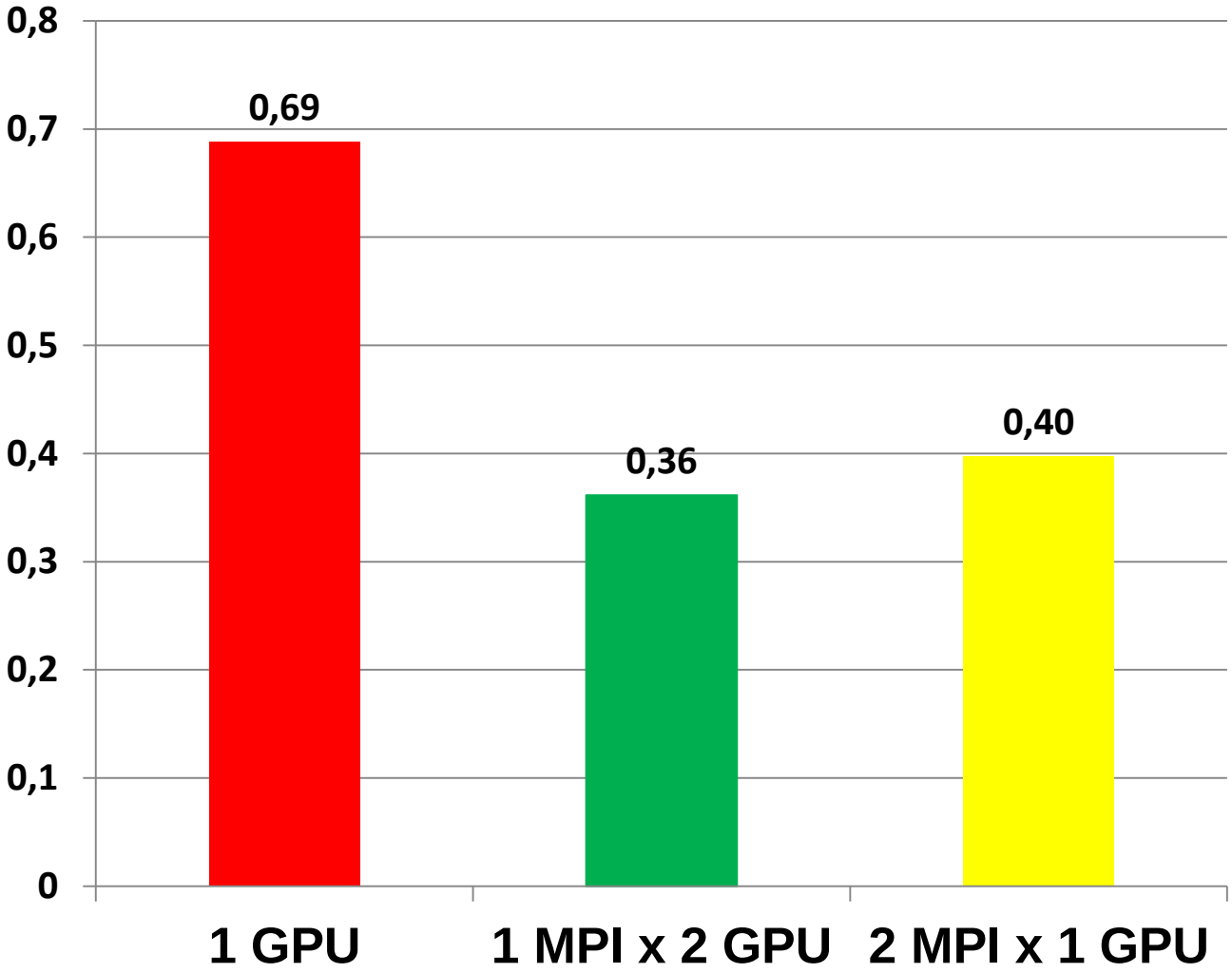
Debugging program using dynamic control

./dvm fpdeb jac.f # Program instrumentation for dynamic control
./dvm err jac # Runing dynamic control of DVMH directives

```
*** DYNCONTROL *** : Loop( No(3), Iter(1) ), Loop( No(1), Iter(2,2,2) ).
        Access to non-local element f(i - 1,j,k)
        File: jac.f  Line: 17
*** DYNCONTROL *** : Loop( No(3), Iter(1) ), Loop( No(1), Iter(2,2,2) ).
        Access to non-local element f(i + 1,j,k)
        File: jac.f  Line: 17
*** DYNCONTROL *** : Loop( No(3), Iter(1) ), Loop( No(1), Iter(2,2,2) ).
        Access to non-local element f(i,j - 1,k)
        File: jac.f  Line: 17
*** DYNCONTROL *** : Loop( No(3), Iter(1) ), Loop( No(1), Iter(2,2,2) ).
        Access to non-local element f(i,j + 1,k)
        File: jac.f  Line: 17
*** DYNCONTROL *** : Loop( No(3), Iter(1) ), Loop( No(1), Iter(2,2,2) ).
        Access to non-local element f(i,j,k - 1)
        File: jac.f  Line: 17
*** DYNCONTROL *** : Loop( No(3), Iter(1) ), Loop( No(1), Iter(2,2,2) ).
        Access to non-local element f(i,j,k + 1)
        File: jac.f  Line: 17
*** Processor 0: total errors: 6; Limit per CPU: 1000
```

```
      double precision function dostep(f, newf, r, rdx2, rdy2, rdz2…)
CDVM$ INHERIT f,newf,r
       …
      eps=0.
CDVM$ ACTUAL(eps)
CDVM$ REGION
CDVM$ PARALLEL(k,j,i) ON newf(i,j,k),REDUCTION(max(eps)),
CDVM$*  SHADOW_RENEW(f)
       do k = 2, mz - 1
          do j = 2, my - 1
              do i = 2, mx - 1
                    newf(i, j, k) = ((f(i-1,j,k)+f(i+1,j,k))*rdx2
     &                              +(f(i,j-1,k)+f(i,j+1,k))*rdy2
     &                              +(f(i,j,k-1)+f(i,j,k+1))*rdz2
     &                              -r(i,j,k)) * beta
                    eps = max(eps,abs(newf(i,j,k)-f(i,j,k)))
              enddo
           enddo
        enddo
CDVM$ END REGION
CDVM$ GET_ACTUAL(eps)
      end function
```

Execution time in seconds of Jacobi Iteration on k10.kiam.ru cluster with nVidia Fermi M2090



Performance analyzer in DVM system

```
Processor system=1*2*1
-----------------------------------------------------------------------
INTERVAL ( NLINE=45 SOURCE=jac.f ) LEVEL=0  EXE_COUNT=1
--- The main characteristics ---
Parallelization efficiency      0.6226
Execution time                  2.2143
Processors                      2
Threads amount                  2
Total time                      4.4286
Productive time                 2.7575 ( CPU= 2.7291 Sys= 0.0281 I/O= 0.0002 )
Lost  time                      1.6712
  Insufficient parallelism      1.3438 ( User= 1.3155 Sys= 0.0283 )
  Communication                 0.1272 ( Real_sync= 0.0000 Starts= 0.0000 )
  Idle time                     0.2002
Load imbalance                  0.1064
Synchronization                 0.0645
Time variation                  0.0018
Overlap                         0.0000
Productive time GPU             1.9469
Lost time GPU                   0.0963
              Nop   Communic   Synchro  Variation   Overlap
I/O            22    0.0000    0.0000    0.0003     0.0000
Reduction      20    0.0048    0.0000    0.0000     0.0000
Shadow         20    0.1223    0.0645    0.0015     0.0000
```

Performance analyzer in DVM system

--- The GPU characteristics ---

Proc: #1

GPU #1 (Tesla M2090)

	#	Min	Max	Sum	Average	Productive	Lost
[Shadow] Copy GPU to CPU	20	2.020M	2.020M	40.392M	2.020M	-	0.0239s
[Shadow] Copy CPU to GPU	20	2.020M	2.020M	40.392M	2.020M	-	0.0080s
[Region IN] Copy CPU to GPU	6	2.020M	523.070M	1.538G	262.545M	0.2684s	-
Loop execution	21	0.0247	0.0352	0.7045	0.0335	0.7045s	-
Reduction	20	0.0002	0.0005	0.0038	0.0002	-	0.0038s

Productive time: 0.9729s
Lost time : 0.0358s

Proc: #2

GPU #2 (Tesla M2090)

	#	Min	Max	Sum	Average	Productive	Lost
[Shadow] Copy GPU to CPU	20	2.020M	2.020M	40.392M	2.020M	-	0.0487s
[Shadow] Copy CPU to GPU	20	2.020M	2.020M	40.392M	2.020M	-	0.0080s
[Region IN] Copy CPU to GPU	6	2.020M	523.070M	1.538G	262.545M	0.2661s	-
Loop execution	21	0.0247	0.0354	0.7078	0.0337	0.7078s	-
Reduction	20	0.0002	0.0005	0.0039	0.0002	-	0.0039s

Productive time: 0.9740s
Lost time : 0.0606s

NAS Parallel Benchmarks

- **MG (MultiGrid)** - Approximation of the solution for a three-dimensional discrete Poisson equation using the V-cycle multigrid method.

- **CG (Conjugate Gradiant)** - Approximation to the smallest eigenvalue of a large sparse symmetric positive-definite matrix using the inverse iteration method together with the conjugate gradient method as a subroutine for solving systems of linear equations.

- **FT (Fast Fourier Transform)** - Solution of three-dimensional partial differential equation (PDE) using the fast Fourier transform (FFT).

- **EP (Embarrassingly Parallel)** - Generation of independent Gaussian random variates using the Marsaglia polar method.

- **BT (Block Tridiagonal)**, **SP (Scalar Pentadiagonal)** and **LU (Lower-Upper)** - Solution of a synthetic system of nonlinear PDEs (three-dimensional system of Navier-Stokes equations for compressible fluid or gas) using three different algorithms: block three-diagonal scheme with the method of alternating directions (BT), the scalar pentadiagonal scheme (SP) and method of symmetric successive over-relaxation (algorithm SSOR of LU).

NAS Parallel Benchmarks for GPU

- OpenCL-version

  Center for Manycore Programming at

  Seoul National University (SNU NPB Suite)

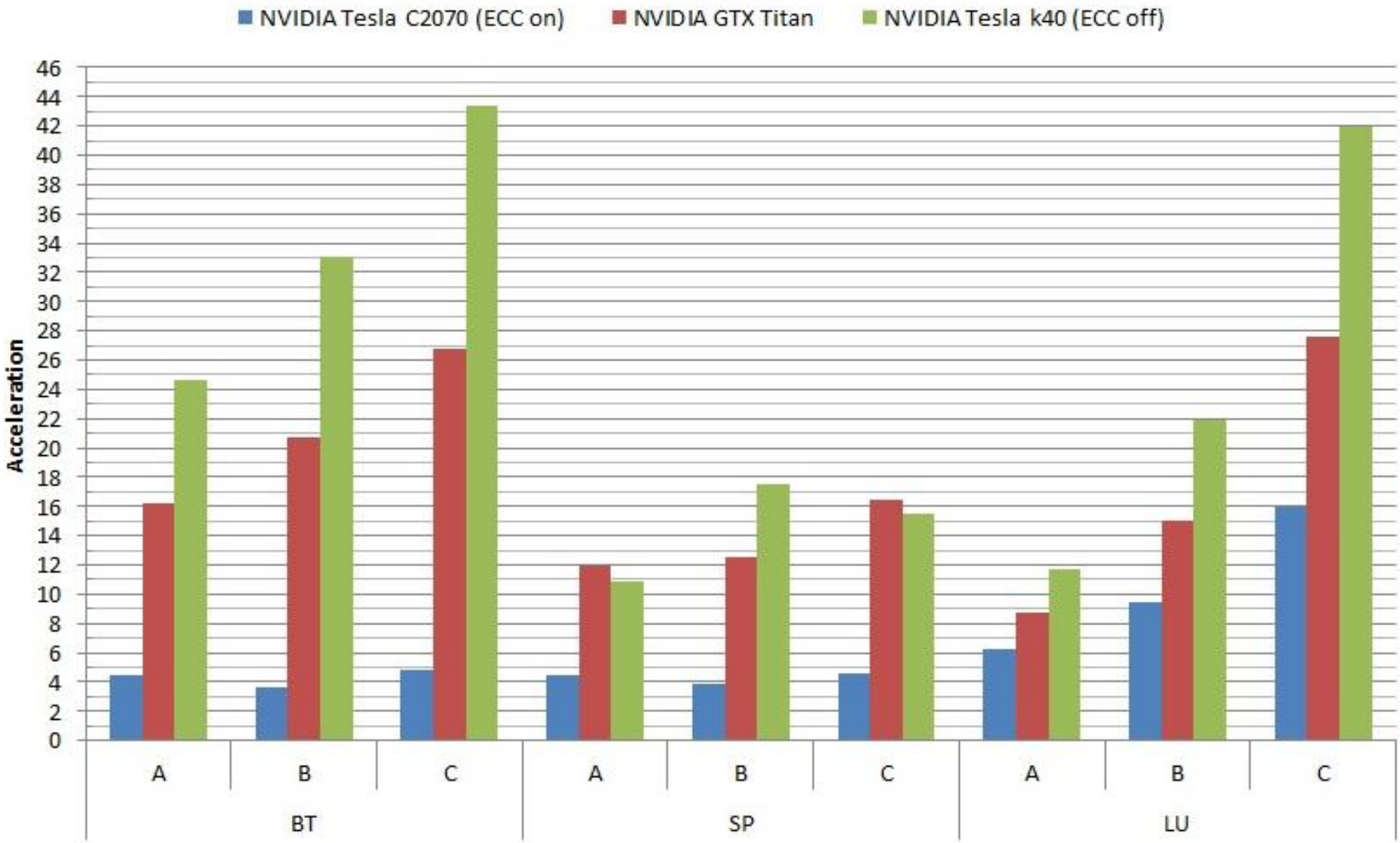
- CUDA-version

  Chemnitz University of Technology (BT, LU, SP)

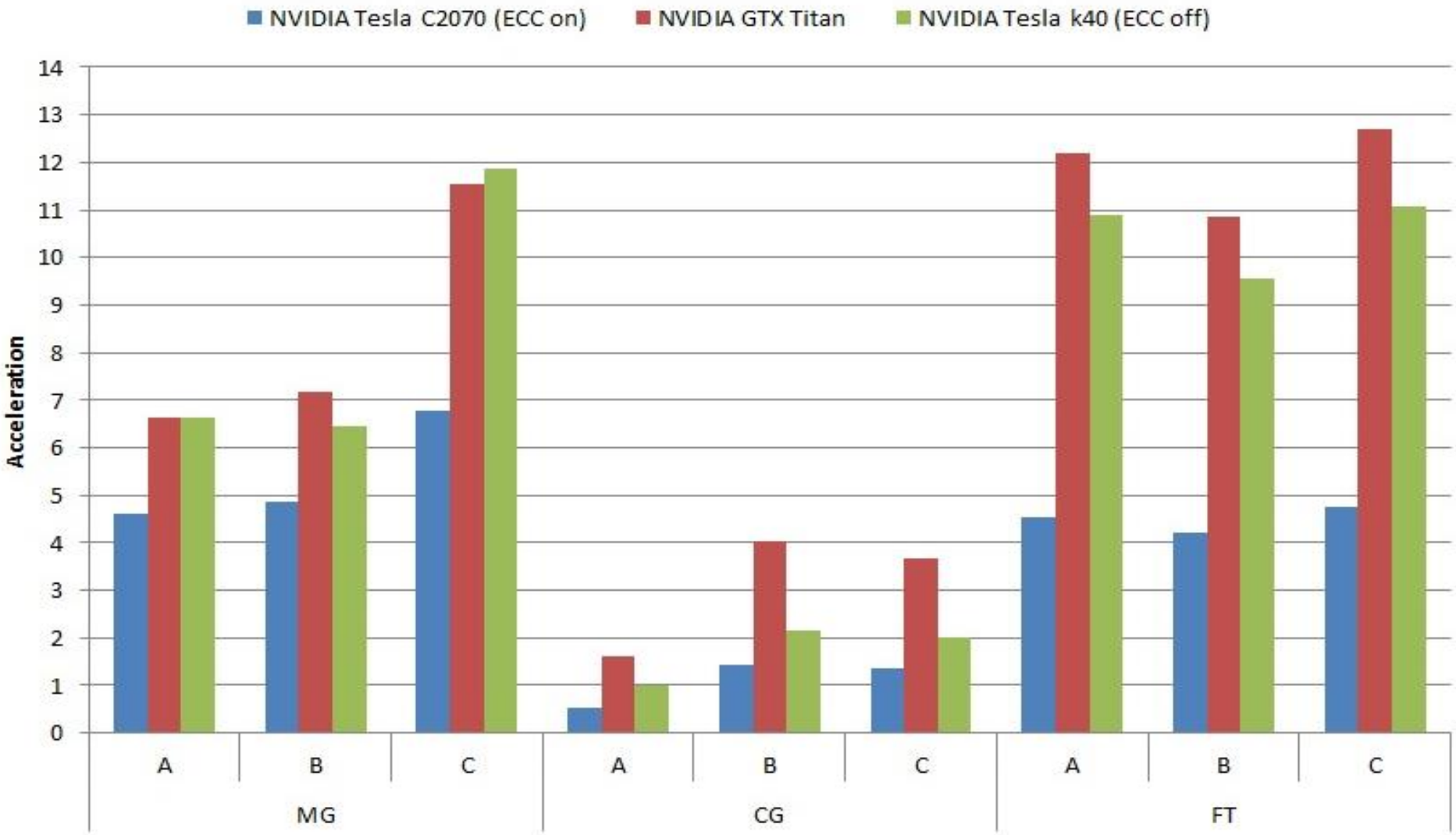
- OpenACC-версии

  SPEC ACCEL (BT, EP, CG, SP)

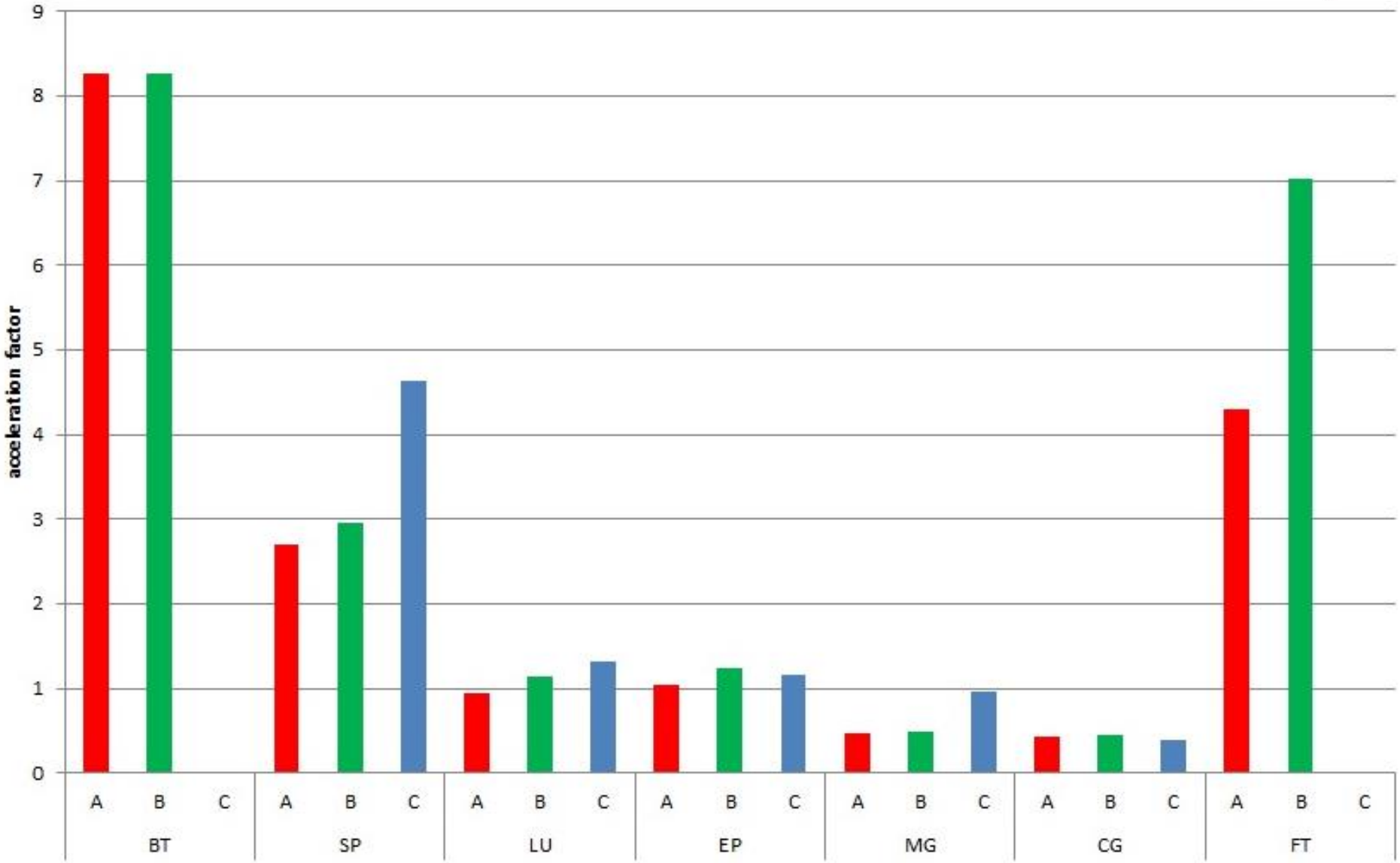
Acceleration FDVMH version of BT, SP, LU on GPU over serial version on CPU Xeon E5 1660 v2



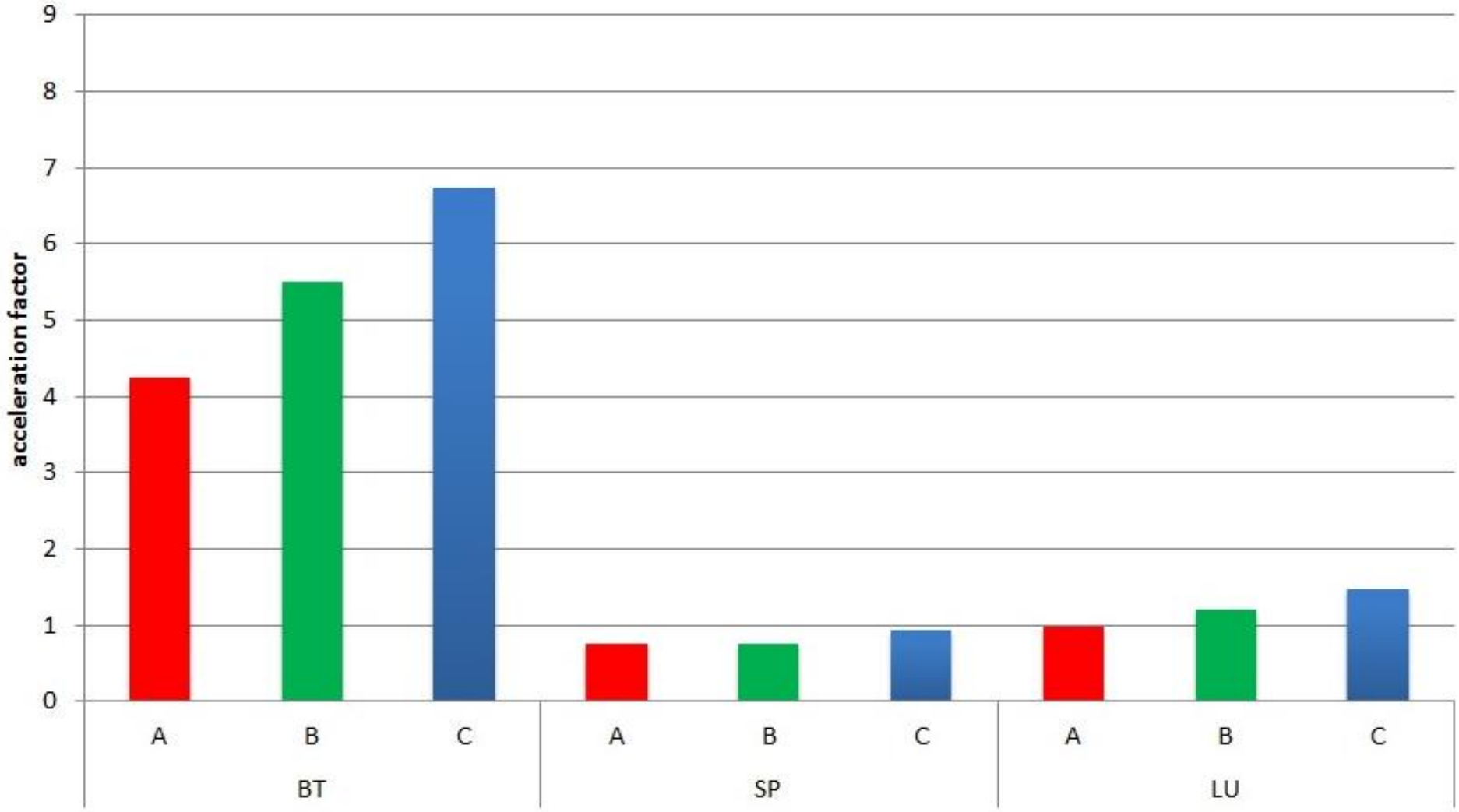
Acceleration FDVMH version of MG, CG, FT on GPU over serial version on CPU Xeon E5 1660 v2



Acceleration FDVMH version tests over OpenCL version on GPU GTX Titan

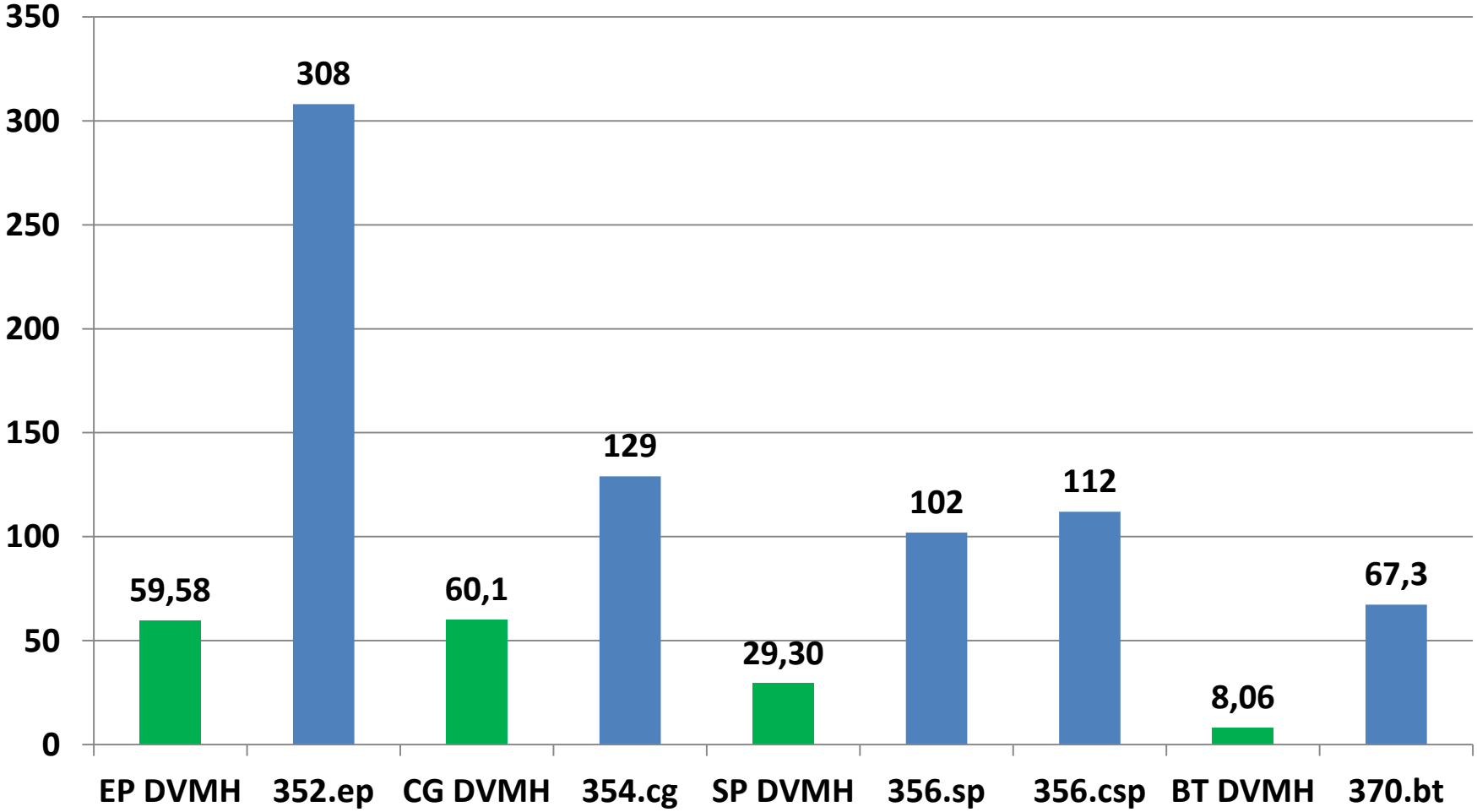


Acceleration FDVMH version tests BT, SP, LU over CUDA version on GPU GTX Titan

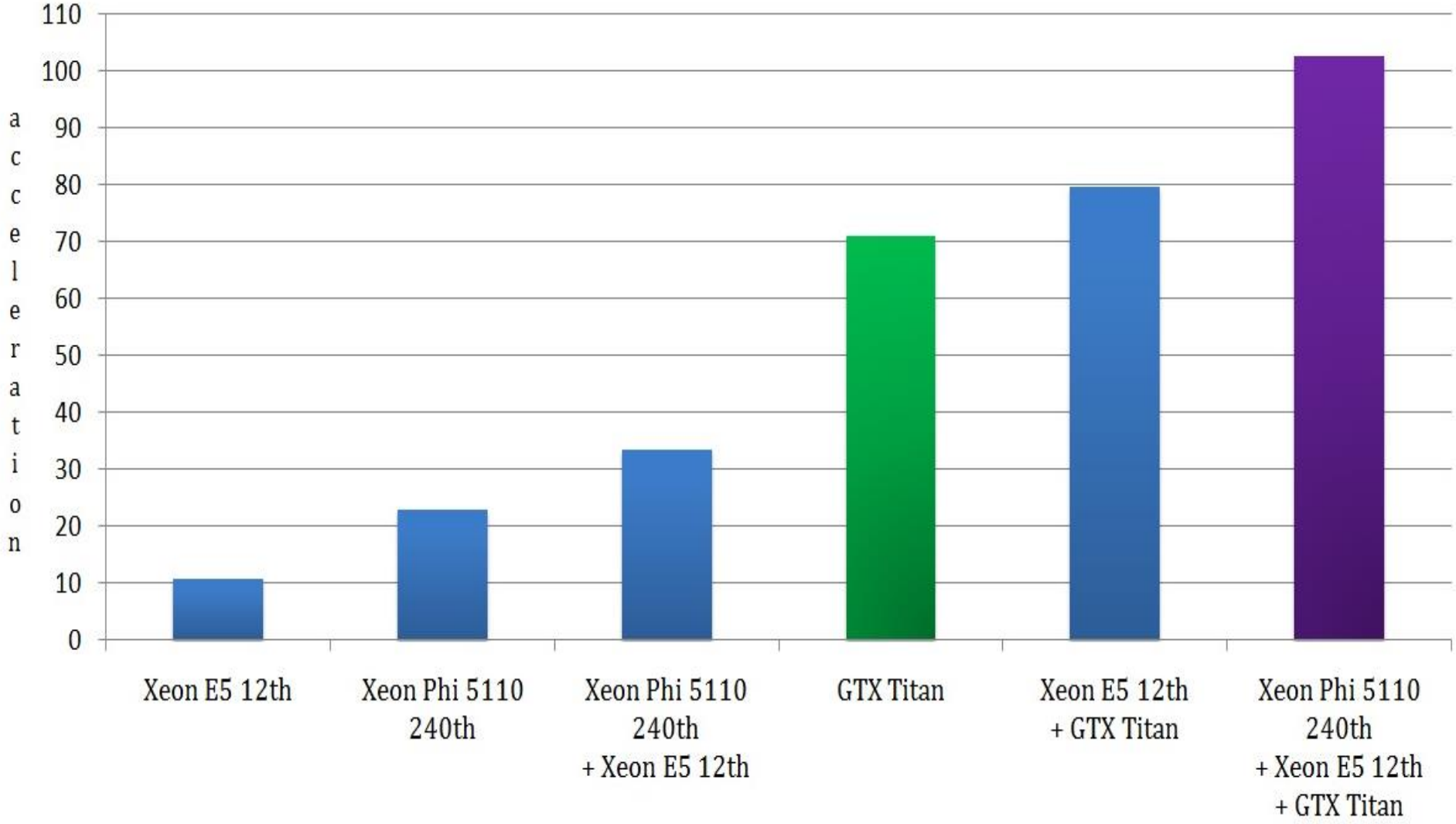


FDVMH version tests EP, CG, SP, BT over OpenACC version on GPU GTX Titan

Results submitted by Technische Universitaet Dresden, spec.org/accel/results/accel.html



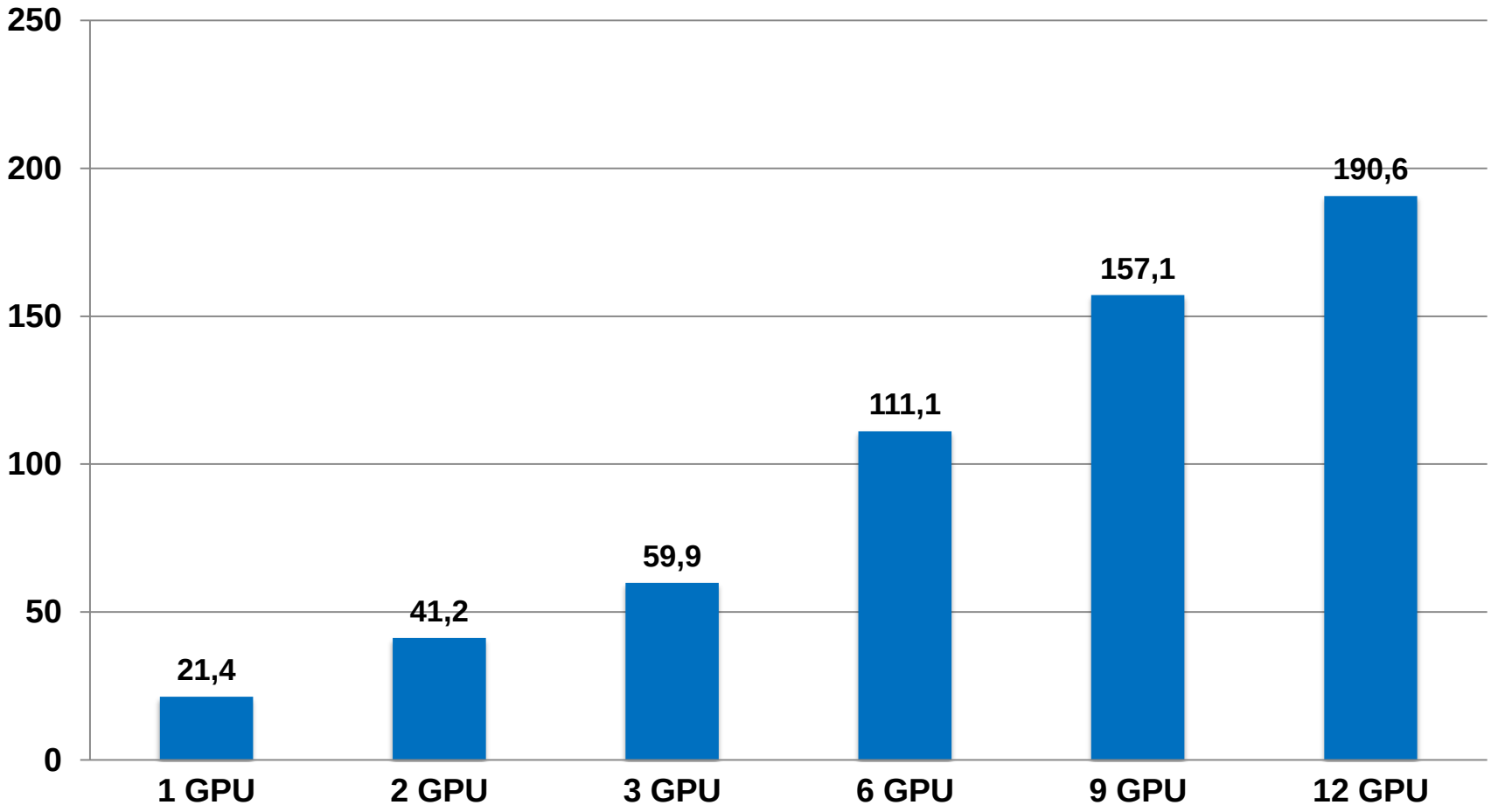
DVMH version of EP(class C) on GPU GTX Titan, Intel Xeon and Xeon Phi



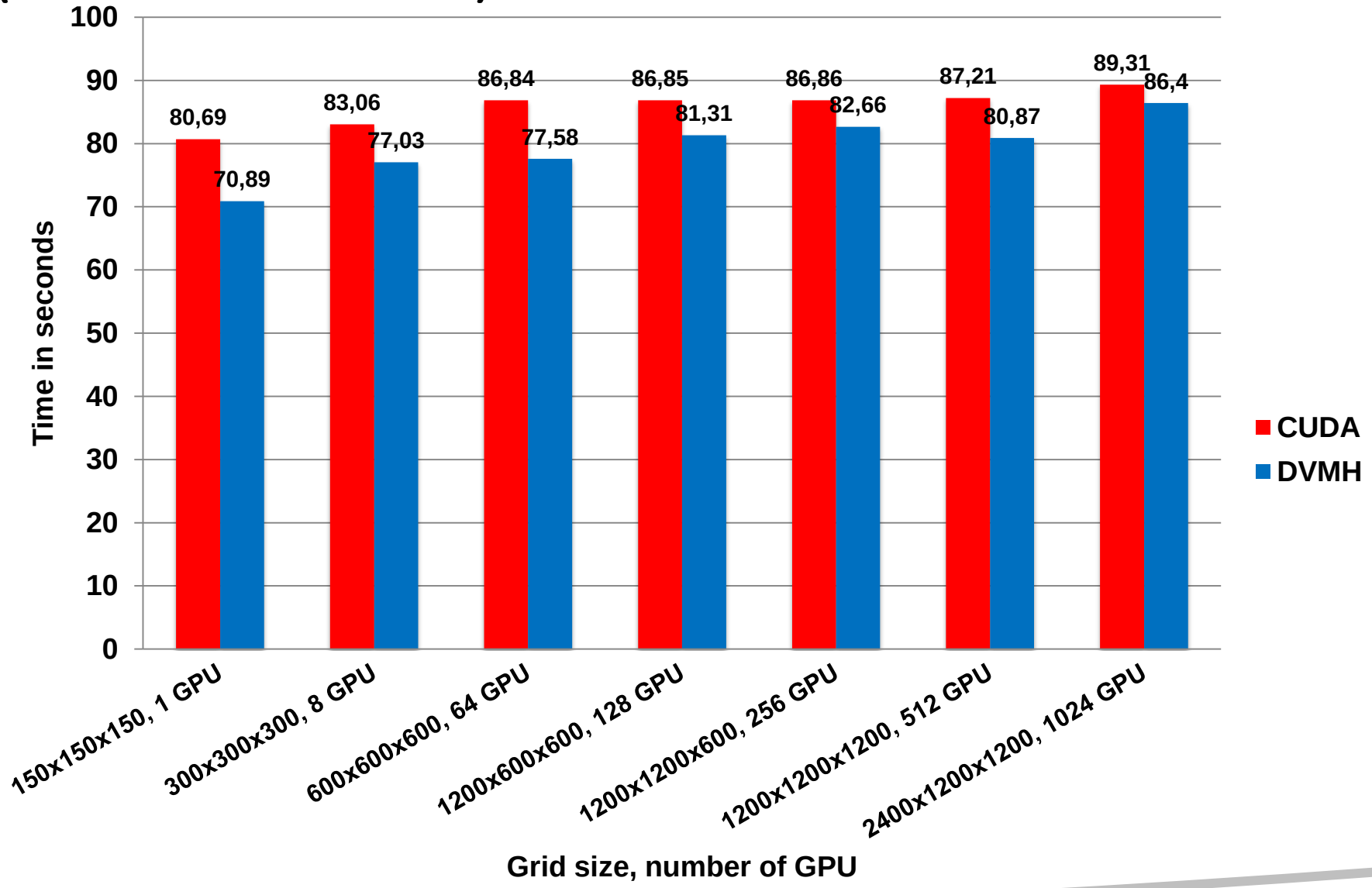
DVMH on real applications

- Program to solve the problem of flow of an incompressible fluid or weakly compressible gas around a rectangular cavity was implemented in Fortran-DVMH language
- Two versions of program were implemented - Cavity (2D, 496 LOC) and Container(3D, 855 LOC)
- Tested on K-100 supercomputer - 64 nodes with 2 Intel Xeon X5670 (6 cores, 2.93GHz) and 3 NVIDIA Tesla C2050 and Lomonosov supercomputer
- Tested using only GPUs

Speedup of Cavity (1600x1600)



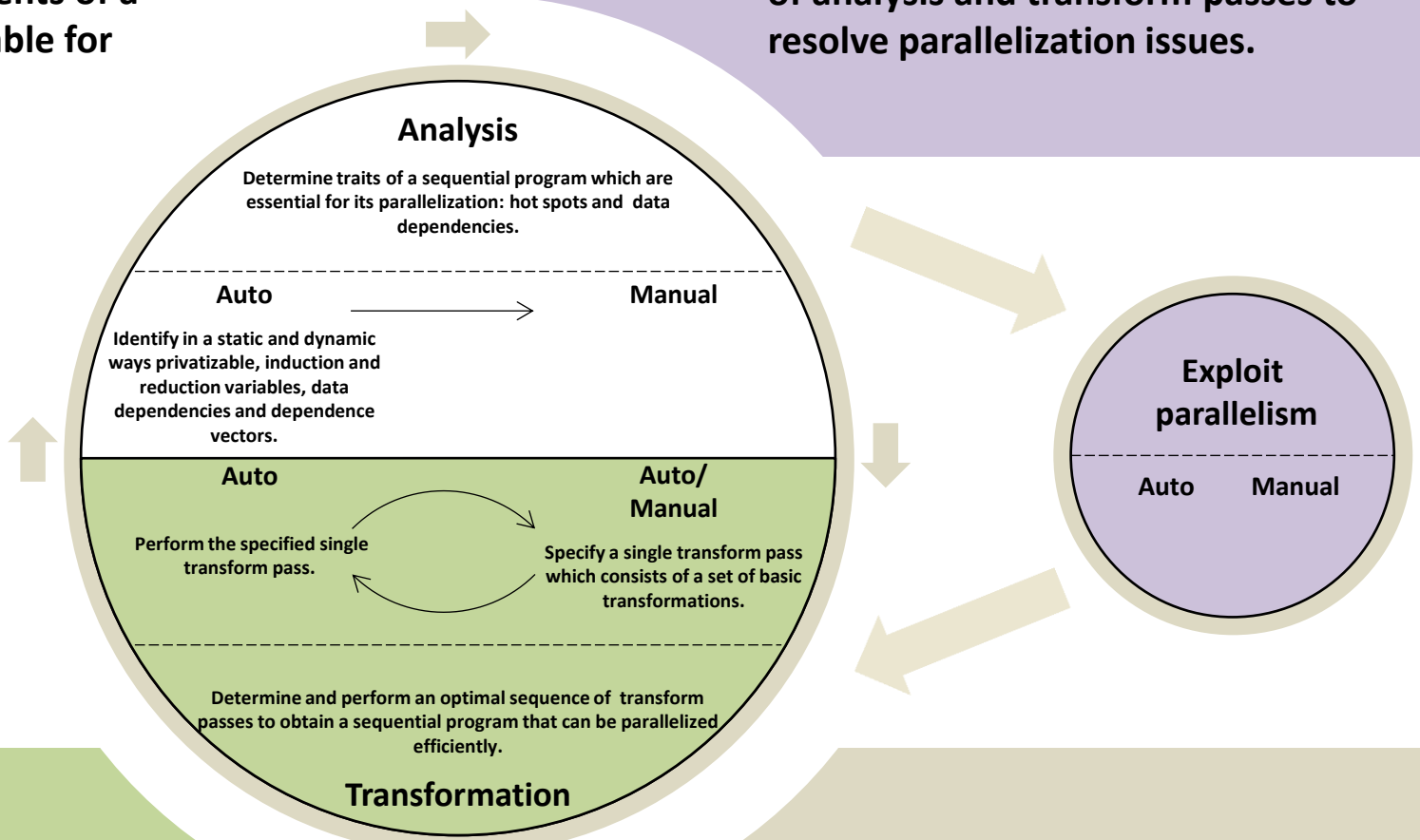
Scaling of Container vs hand-written (C+SHMEM+CUDA)



Semi-automatic parallelization with SAPFOR

Advanced profiling capability to identify fragments of a source code suitable for parallelization.

Semi-automatic search of a sequence of analysis and transform passes to resolve parallelization issues.



Automatic execution of the most frequent transformations.

Visual assistance tool to provide interaction with a user.

Conclusions

- DVM-system automates the process of parallel programs development
- The resulting DVMH programs can run efficiently on different clusters using multi-core universal processors and graphics accelerators without any changes
- This is achieved through various optimizations that are performed both statically, when DVMH programs are compiled, and dynamically. The resulting parallel programs can be configured at startup for the resources allocated for their execution - the number of cluster nodes, cores, accelerators and their performance

Thank you for attention

http://dvm-system.org
dvm@keldysh.ru

SSIP-2018