14th International Conference on Parallel Computing Technologies

Parallel Computing Technologies 2017

Automated Parallelization of a Simulation Method of Elastic Wave Propagation in Media with Complex 3D Geometry Surface on High-Performance Heterogeneous Clusters

September 8, 2017 | Nizhni Novgorod

Nikita Kataev, Alexander Kolganov, Pavel Titov

Moscow State University

Keldysh Institute of Applied Mathematics RAS

Institute of Computational Mathematics and Mathematical Geophysics SB RAS

How to simplify parallel programming?

- Program parallelization is the process of transforming a computer program in order to tap the full potential of parallel computers (multi-core or heterogeneous computational systems, etc.). This implies either rewriting of a program in a special language or inserting special directives.

- Auto-parallelization is a compiler optimization of a computer program in order to utilize multiple processors simultaneously, for example, in a shared-memory multiprocessor (SMP or NUMA) machine.

- Semi-automatic parallelization is a compiler optimization of a computer program in order to utilize multiple processors simultaneously which requires the user to provide the vital information about the program or to perform some transformations manually.

Parallel programming tools

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 System components

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

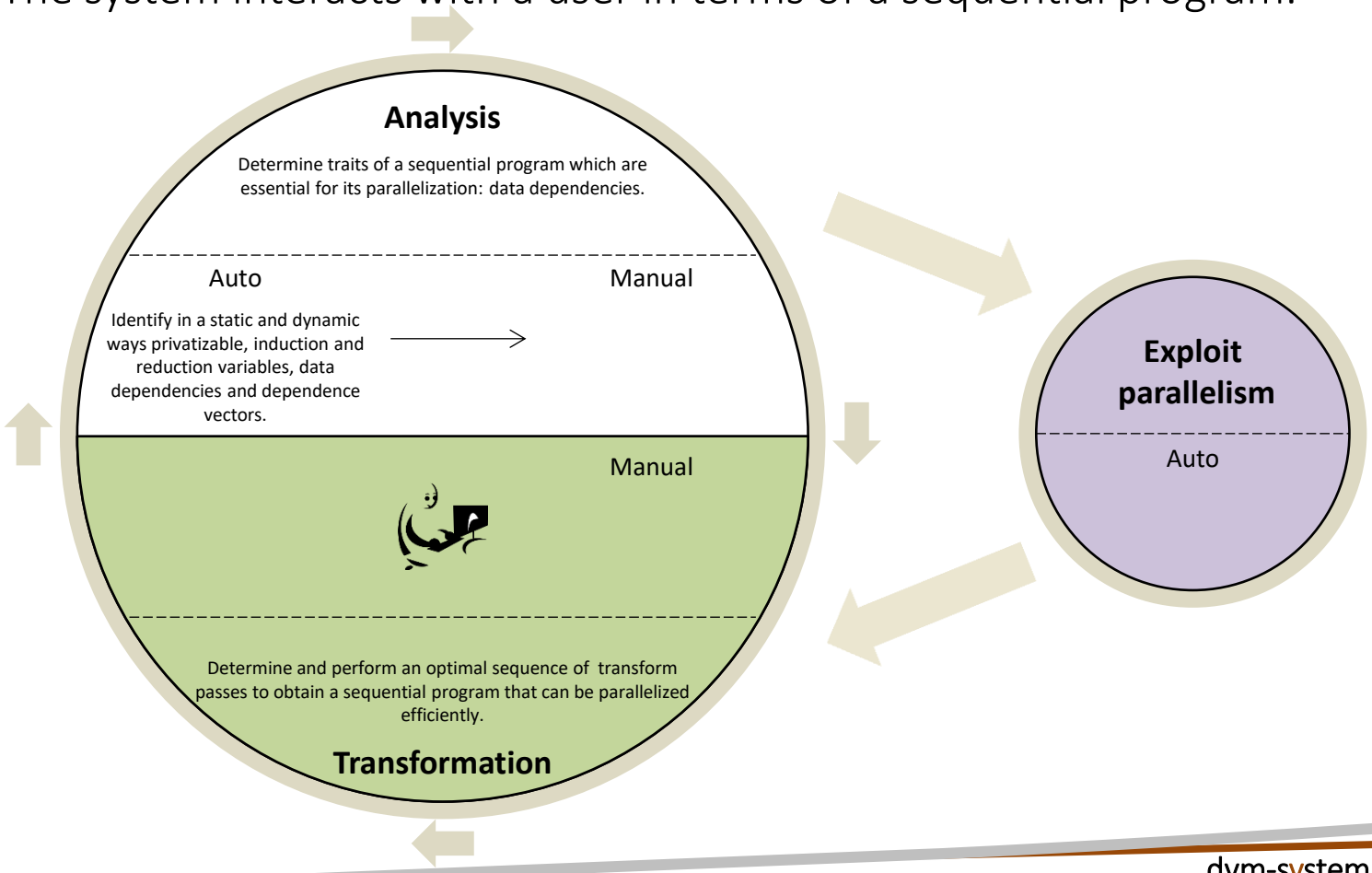
DVM System languages

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.

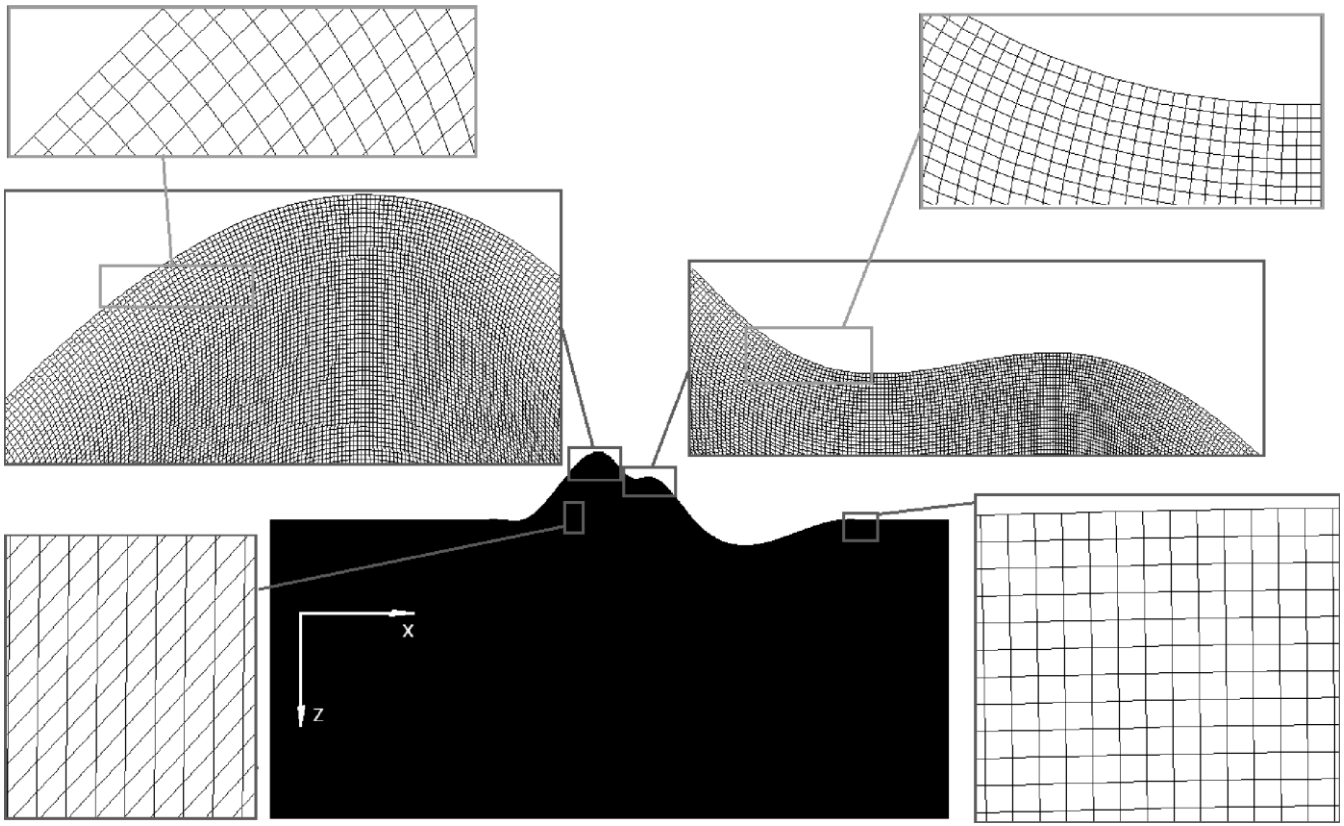
Semi-automatic parallelization with SAPFOR

- The system is focused on cost reduction of manual program parallelization for heterogeneous computational clusters.
- The parallelism exploitation in an automatic way relies on semi-automatic analysis techniques.
- The system interacts with a user in terms of a sequential program.



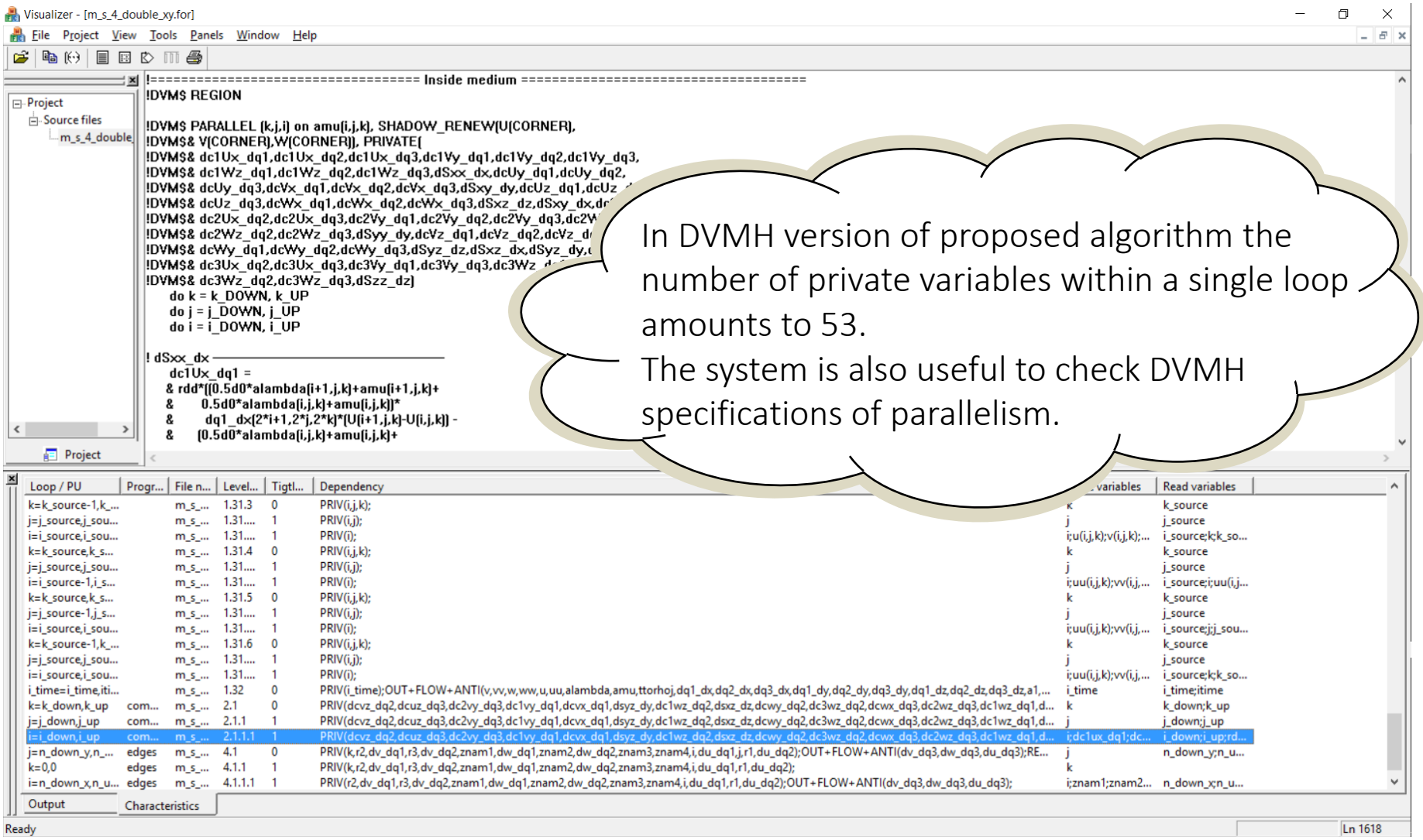
Simulation Method of Elastic Wave Propagation in Media with Complex 3D Geometry Surface

- This is related to the solution of equations of elasticity theory in the case of 3D medium.
- Transformation of a curvilinear grid, 2D-section



Program exploration with SAPFOR

- ~ 3000 lines of code in Fortran 95, > 100 allocatable arrays, > 90 loops



Parallelism exploitation with DVM

- Distribution of array elements on the processors:

  directives distribute / align

- Mapping of the loop iterations on the processors in accordance with data distribution:

  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

Data distribution

Distribution of allocatable arrays

```
double precision, allocatable::
      & dq1_dx(:,:,:),dq2_dx(:,:,:),dq3_dx(:,:,:),
      & dq1_dy(:,:,:),dq2_dy(:,:,:),dq3_dy(:,:,:),
      & dq1_dz(:,:,:),dq2_dz(:,:,:),dq3_dz(:,:,:),
        ...
```

Distribution for any number of processors

```
!DVM$ DISTRIBUTE dq1_dx(BLOCK, BLOCK, *)
!DVM$ ALIGN (i,j,k) WITH dq1_dx(i,j,k)::dq2_dx, dq3_dx, dq1_dy, dq2_dy
!DVM$ ALIGN (i,j,k) WITH dq1_dx(i,j,k)::dq3_dy, dq1_dz, dq2_dz, dq3_dz
        ...
```

Memory allocations by normal Fortran statements

```
        allocate(
      & dq1_dx(2*N_down_x-3:2*N_up_x+3,2*N_down_y-3:2*N_up_y+3,
      & 2*N_down_z-3:2*N_up_z+3),
      & dq2_dx(2*N_down_x-3:2*N_up_x+3,2*N_down_y-3:2*N_up_y+3,
      & 2*N_down_z-3:2*N_up_z+3),
      & dq3_dx(2*N_down_x-3:2*N_up_x+3,2*N_down_y-3:2*N_up_y+3,
      & 2*N_down_z-3:2*N_up_z+3),
        ...)
```

Loop mapping

Privitizable and reduction variables are discovered by SAPFOR, so there is no trouble to add the necessary clauses (PRIVATE and REDUCTION)

```
!DVM$ PARALLEL (k,j,i) on amu(i,j,k), PRIVATE(
!DVM$& dc1Ux_dq1,dc1Ux_dq2,dc1Ux_dq3,dc1Vy_dq1,dc1Vy_dq2,dc1Vy_dq3,
!DVM$& dc1Wz_dq1,dc1Wz_dq2,dc1Wz_dq3,dSxx_dx,dcUy_dq1,dcUy_dq2,
       ...)
      do k = k_DOWN, k_UP
      do j = j_DOWN, j_UP
      do i = i_DOWN, i_UP
       ...


!DVM$ PARALLEL (k,j,i) ON X(2*i,2*j,2*k),
!DVM$& REDUCTION(MINLOC(amax,newSources,3))
      do k=N_down_z,N_up_z
      do j=N_down_y,N_up_y
      do i=N_down_x,N_up_x

      if(((X(2*i,2*j,2*k)-x0)**2 +
     &     (Y(2*i,2*j,2*k)-y0)**2 +
     &     (Z(2*i,2*j,2*k)-z0)**2 < amax)) then
       ...
```

Organization of access to remote data

Dynamic control of DVMH directives proved to be useful for functional debugging of program

./dvm fpdeb M_S_4_double_xy.for

./dvm err M_S_4_double_xy

```
!DVM$ PARALLEL (k,j,i) on amu(i,j,k), PRIVATE(
!DVM$& dc1Ux_dq1,dc1Ux_dq2,dc1Ux_dq3,dc1Vy_dq1,dc1Vy_dq2,dc1Vy_dq3,
!DVM$& dc1Wz_dq1,dc1Wz_dq2,dc1Wz_dq3,dSxx_dx,dcUy_dq1,dcUy_dq2,
       ...)
      do k = k_DOWN, k_UP
      do j = j_DOWN, j_UP
      do i = i_DOWN, i_UP
       dc1Ux_dq1 =
       ...
     & 0.25d0*rdd*( (alambda(i+1,j,k)+2.0d0*amu(i+1,j,k))*
     &  dq2_dx(2*i+2,2*j,2*k)*(U(i+1,j+1,k)-U(i+1,j-1,k)) -
     &                 (alambda(i-1,j,k)+2.0d0*amu(i-1,j,k))*
     &  dq2_dx(2*i-2,2*j,2*k)*(U(i-1,j+1,k)-U(i-1,j-1,k)) ) +
       ...
```

*** DYNCONTROL *** : Loop(No(46), Iter(0)), Loop(No(54), Iter(1,2,2)).

Access to non-local element dq2 _dx(2 * i + 2,2 * j,2 * k)

File: M_S_4_double_xy.for Line: 1710

Organization of access to remote data

```
!DVM$ SHADOW(1:1, 1:1, 1:2) :: U,V,W
!DVM$ SHADOW(2:2, 2:2, 2:2) :: dq1_dx,dq2_dx, dq3_dx, dq1_dy, dq2_dy
!DVM$ SHADOW(2:2, 2:2, 2:2) :: dq3_dy, dq1_dz, dq2_dz, dq3_dz,X,Y,Z
        ...
!DVM$ PARALLEL (k,j,i) on amu(i,j,k), PRIVATE(
!DVM$& dc1Ux_dq1,dc1Ux_dq2,dc1Ux_dq3,dc1Vy_dq1,dc1Vy_dq2,dc1Vy_dq3,
!DVM$& dc1Wz_dq1,dc1Wz_dq2,dc1Wz_dq3,dSxx_dx,dcUy_dq1,dcUy_dq2,
        ...)
      do k = k_DOWN, k_UP
      do j = j_DOWN, j_UP
      do i = i_DOWN, i_UP
       dc1Ux_dq1 =
        ...
     & 0.25d0*rdd*( (alambda(i+1,j,k)+2.0d0*amu(i+1,j,k))*
     &  dq2_dx(2*i+2,2*j,2*k)*(U(i+1,j+1,k)-U(i+1,j-1,k)) -
     &                (alambda(i-1,j,k)+2.0d0*amu(i-1,j,k))*
     &  dq2_dx(2*i-2,2*j,2*k)*(U(i-1,j+1,k)-U(i-1,j-1,k)) ) +
        ...
```

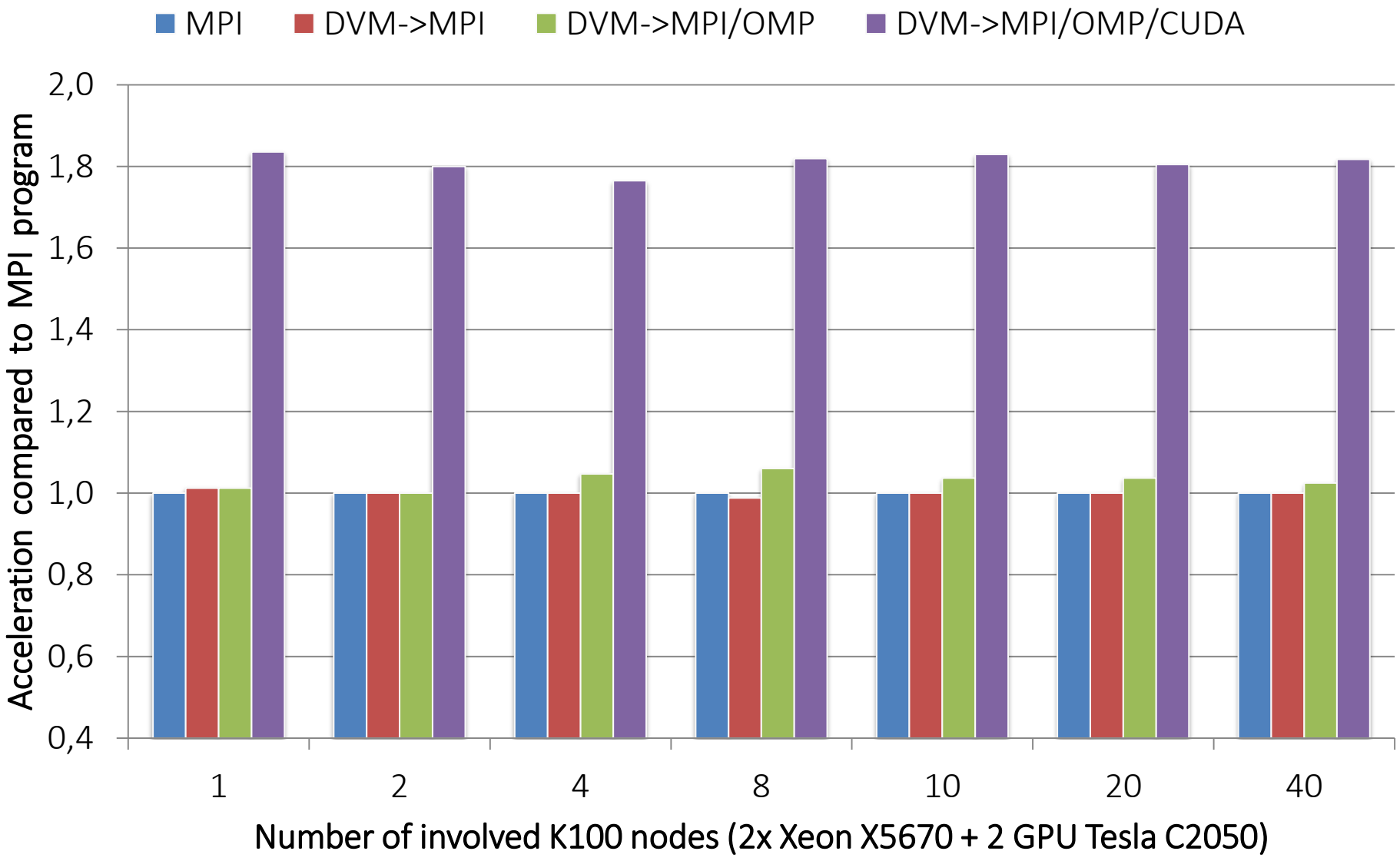
Just extend the shadow edges to access 2 remote elements of the array (1 by default).

Specification of computational regions

Now the program may be executed on a multiprocessor or on other devices (GPU, Xeon Phi)

```
!DVM$ REGION
!DVM$ PARALLEL (k,j,i) on amu(i,j,k), PRIVATE(
!DVM$& dc1Ux_dq1,dc1Ux_dq2,dc1Ux_dq3,dc1Vy_dq1,dc1Vy_dq2,dc1Vy_dq3,
!DVM$& dc1Wz_dq1,dc1Wz_dq2,dc1Wz_dq3,dSxx_dx,dcUy_dq1,dcUy_dq2,
       ...)
      do k = k_DOWN, k_UP
      do j = j_DOWN, j_UP
      do i = i_DOWN, i_UP
       dc1Ux_dq1 =
        ...
     & 0.25d0*rdd*( (alambda(i+1,j,k)+2.0d0*amu(i+1,j,k))*
     &  dq2_dx(2*i+2,2*j,2*k)*(U(i+1,j+1,k)-U(i+1,j-1,k)) -
     &                (alambda(i-1,j,k)+2.0d0*amu(i-1,j,k))*
     &  dq2_dx(2*i-2,2*j,2*k)*(U(i-1,j+1,k)-U(i-1,j-1,k)) ) +
        ...
      end do
      end do
      end do
!DVM$ END REGION
```

Comparison of DVMH and MPI programs



Results

- ~ 3000 lines of code in fixed source form in Fortran 95
- > 100 allocatable arrays, > 90 loops

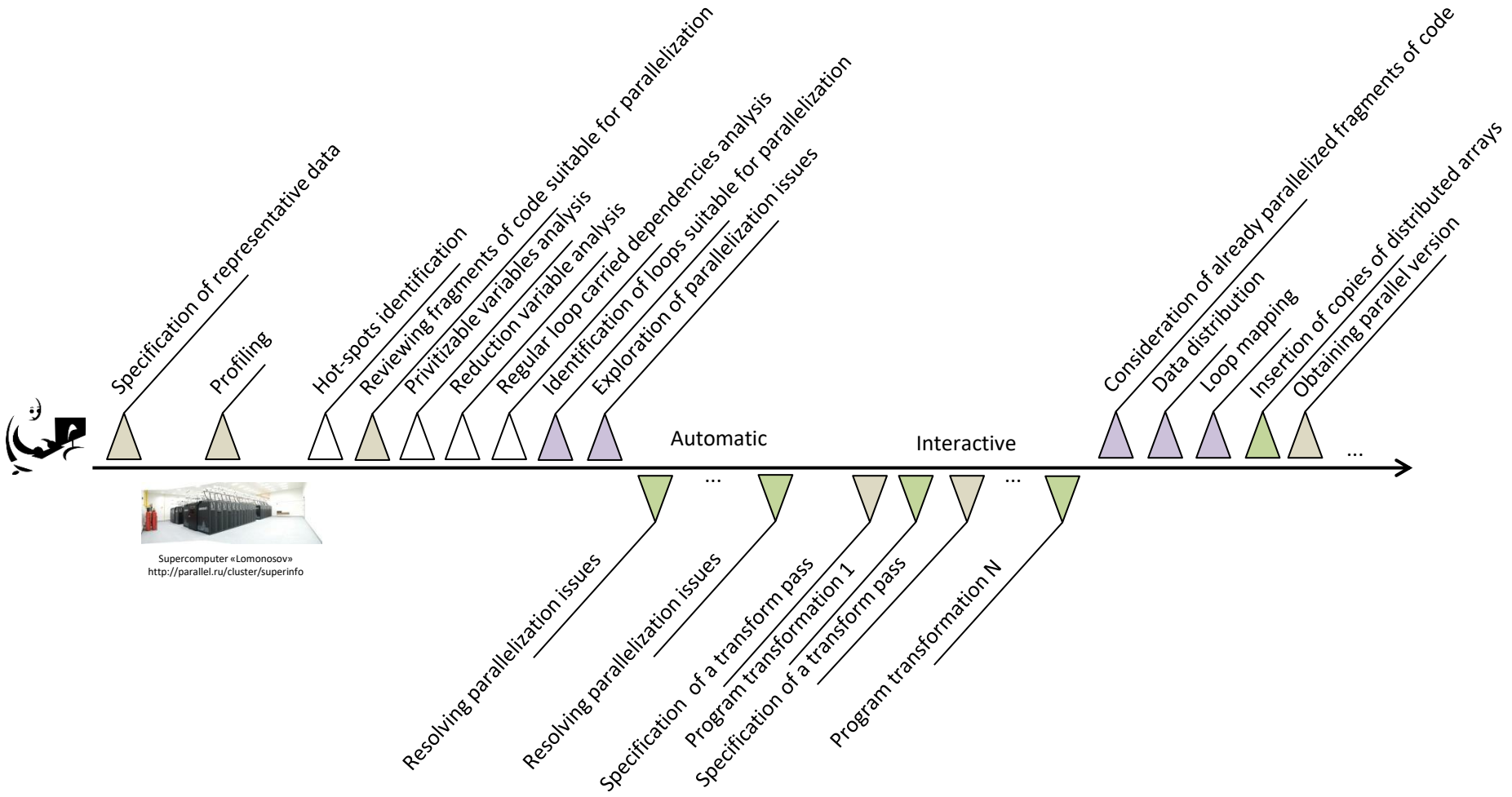
- 79 DVM directives have been placed
- 1 day for parallelization and 2 days to obtain results
- Parallel execution on heterogeneous computational cluster with GPU
- In total, we were able to use 480 CPU cores and 80 GPUs (40 nodes of K100), the size of the problem in this case was approximately of 1000 GB
- Almost linear acceleration of DVMH program execution:

strong and weak scaling

- Source code https://bitbucket.org/dvm-system/elastic-wave-3d
  - SEQ_VER contains a sequential program
  - MPI_VER contains a parallel program based on MPI
  - DVMH_VER contains a parallel program based on DVMH

SAPFOR development suit: future

– *interactive , iterative and incremental parallelization*

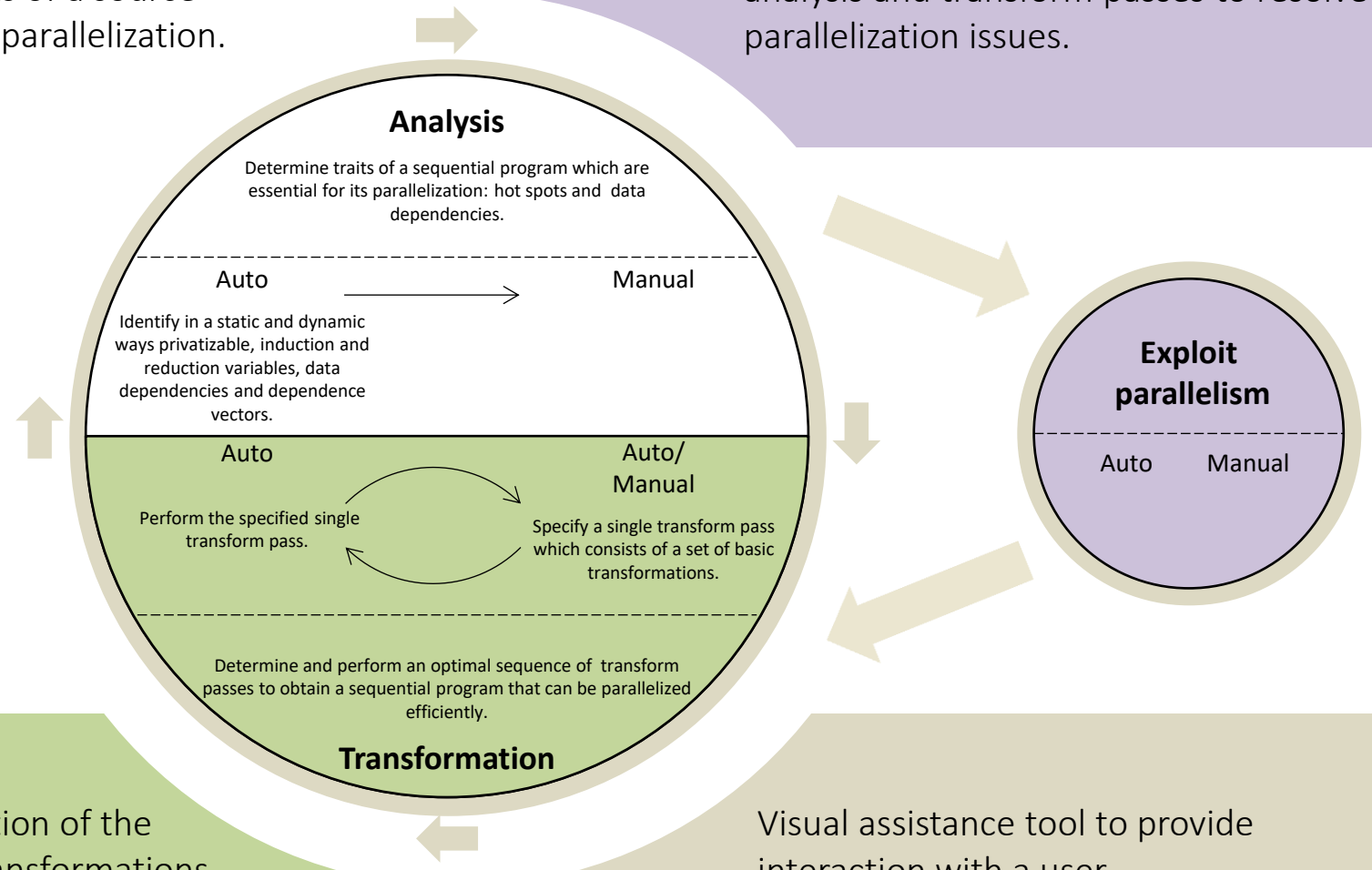


Program parallelization relies on the ability to identify a sequence of analysis and transform passes to obtain a computer program which effectively taps full potential of parallel computers.

SAPFOR development suit: future

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.

Thank you for attention

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