Development and launch of parallel C-DVMH and Fortran-DVMH programs on supercomputer "LOMONOSOV"

DVM System

December 17, 2016

To compile and launch parallel DVMH-programs on supercomputer "LOMONOSOV" it is necessary:

1. To load modules for the compiler, MPI implementation , CUDA, queue system and perhaps, other modules necessary for your program:

```
module add intel/15.0.090
module add impi/5.0.1
module add cuda/6.5.14
module add slurm/2.5.6
```

   Note. The loaded modules don't keeped between sessions. If you need to use always the same set of the modules, add necessary commands in the file ~/.bashrc.

2. To initiate DVM system version installed on supercomputer "LOMONOSOV". To perform the command:

```
/mnt/msu/users/admdvm/DVM/dvminit
```

   As a result of given command performing, dvm and usr.par files necessary for DVMH-program compilation and execution will be copied to the current directory of a user, and also the symbolic reference ~dvm_current to the installed DVM system version will be created. Demonstrational DVMH-programs are available in the directory ~dvm_current/dvm_sys/demo (files with *.cdv and *.fdv extensions).

   Note. On the supercomputer "LOMONOSOV" the programs are launched only from the directory ~/_scratch and its subdirectories. Only this directory will be visible on a compute node and it will be considered as home directory. Therefore it is recommended to initiate DVM system in the directory ~/_scratch or in its subdirectories. Otherwise before start you will need to copy all data, dvm script and executable files in the directory ~/_scratch.

3. Compilation of DVMH-programs. Compilation is performed on the compiler node. Previously it is necessary to come on it using ssh compiler command. You can come on this node immediately, specifying in the ssh-client the address compiler.lomonosov.parallel.ru instead of lomonosov.parallel.ru.

   (a) Compilation of C-DVMH program is performed by the command:

```
./dvm c [-noH] [-noCuda] <program name>.<cdvm-ext>,
```

   where <cdvm-ext> = cdv|c|h
   For example,

```
./dvm c jac3d_perf.cdv
```

   The result of the command is ready-for-run parallel program in MPI/OpenMP/CUDA model with the name jac3d_perf.

```
./dvm c -noCuda jac3d_perf.cdv
```

   The result of the command is ready-for-run parallel program in MPI/OpenMP model with the name jac3d_perf.

```
./dvm c -noH jac3d_perf.cdv
```

   The result of the command is ready-for-run parallel program in MPI model with the name jac3d_perf.

   (b) Compilation of Fortran-DVMH program is performed by the command:

```
./dvm f [-noH] [-noCuda] <program name>.<fdvm-ext>,
```

   where <fdvm-ext> = fdv|f|ftn|for|f90|f95|f03
   For example,

```
./dvm f jac3d_perf.fdv
```

The result of the command is ready-for-run parallel program in MPI/OpenMP/CUDA model with the name `jac3d_perf`.

```
./dvm f -noCuda jac3d_perf.fdv
```

The result of the command is ready-for-run parallel program in MPI/OpenMP model with the name `jac3d_perf`.

```
./dvm f -noH jac3d_perf.fdv
```

The result of the command is ready-for-run parallel program in MPI model with the name `jac3d_perf`.

Notes.

i. DVMH-program can have standard extension (for example, `*.f90` or `*.c`).

ii. For Fortran-programs the record format (free or fixed) is determined automatically by the program extension. For the programs having extensions `*.f90`, `*.f95`, `*.f03`, the free format is used, and for all remaining – the fixed one. It is possible to specify a format using compiler options: `-f90` (free) and `-FI` (fixed). For example,

```
./dvm f -f90 jac.fdv
```

4. Launching DVMH-program.

Before DVMH program start it is necessary to set the following environment variables in `dvm` script (details see in documentation):

(a) Maximum runtime of the program. `maxtime` variable:
`export maxtime=30`
Default value is 30 minutes. For more long calculations (for example, 1 hour) it is necessary to set:
`export maxtime=60`

(b) the name of the queue system section in which the task will be queued. `queuemode` variable:
`export queuemode='regular4'`

(c) A number of MPI processes launched on a node. `DVMH_PPN` variable:
`export DVMH_PPN='2'`

(d) A number of the threads used by each MPI process. `DVMH_NUM_THREADS` variable:
`export DVMH_NUM_THREADS='4'`
Note. Value of this variable is ignored for the programs compiled with the option `-noH`.

(e) for each MPI process it is necessary to specify a number of used graphic accelerators. `DVMH_NUM_CUDAS` variable:
`export DVMH_NUM_CUDAS='2'`
Note. Value of this variable is ignored for the programs compiled with option `-noH` or `-noCuda`.

One node of supercomputer K-100 contains two 6-core processors Intel Xeon X5670 and three graphic accelerators NVidia Fermi C2050.

Recommended configurations:

For the section `gpu` it is possible to use the configuration:

```
export DVMH_PPN='2'
export DVMH_NUM_THREADS='0'
export DVMH_NUM_CUDAS='1'
```

For this configuration three MPI processes will be launched on each compute node of supercomputer "LOMONOSOV", and each of them will use its own graphic accelerator for computations.

For the section `regular6` it is possible to use the following configuration:

```
export DVMH_PPN='2'
export DVMH_NUM_THREADS='6'
export DVMH_NUM_CUDAS='0'
```

For this configuration two MPI processes with 6 threads per process will be launched on each compute node of supercomputer "LOMONOSOV". Graphic accelerators won't be used.

For the section `regular4` it is possible to use the following configuration:

```
export DVMH_PPN='8'
export DVMH_NUM_THREADS='1'
export DVMH_NUM_CUDAS='0'
```

For this configuration 8 MPI processes will be launched on each compute node of supercomputer "LOMONOSOV". There won't be any job distribution between threads and graphic accelerators.

After setting of environment variables listed above, the DVMH-program can be launched by the command:

```
./dvm run <processor matrix> <program name> [<task parameters>]
```

The parameter `<processor matrix>` sets rank of a processor grid on which the task will be run (`<n1> <n2>... <nK>`). During the task execution `<n1>*<n2>*...*<nK>` MPI processes will be created. For example,

```
./dvm run 2 1 2 jac3d_perf
```

As a result of this command the task which uses four MPI processes will be created. For the configuration:

```
export DVMH_PPN='2'
export DVMH_NUM_THREADS='0'
export DVMH_NUM_CUDAS='1'
```

two compute nodes will be provided for the task execution. Two MPI processes will be launched on each of them. Each from the processes will use one graphic accelerator. In total 4 graphic accelerators will be used for computations.

If there are not enough free resources and the program was queued (`Submitted batch job <JOBID>; squeue -j <JOBID>`), it is possible to delete it from queue (`scancel <JOBID>`), to get information about a number of free processors/sections (`sinfo`) and to restart the task on smaller number of processors.

Outputs of the task are redirected to the files `<task name>.<task number>.1/output` and `<task name>.<task number>.1/error`.

Note. `<task name>` is `<program name>.<number of processors>.<task number>`. The field `<task number>` is a number of the program starts on given number of processes in the current directory.

5. Obtaining efficiency characteristics of DVMH-program execution. Command:

```
./dvm pa <task name>.sts.gz+ <name of output file with characteristics>
```

6. The help information about other DVM commands is available as:

```
./dvm help
```

The detailed information about DVM system is on DVM site.
Questions and notes should be sent to the address: dvm@keldysh.ru.
The instruction how to work on supercomputer "LOMONOSOV" is available on the site.