Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Direct Numerical Simulation of a high-Reynolds-number Homogeneous shear turbulence

Siwei Dong[a]*, Vegard Eide[b]**, Jeroen Engelberts[c]

[a]*Universidad Politecnica Madrid, Spain*
[b]*Norwegian University of Science and Technology, Norway*
[c]*SURFsara, Amsterdam, Netherlands*

Abstract

The SHEAR code is developed at the School of Aeronautics, Universidad Politécnica de Madrid, for the simulation of turbulent structures of shear flows. The code has been well tested on smaller clusters. This white paper desbribes the work done to scale and optimise SHEAR for large systems like the Blue Gene/Q system JUQUEEN in Jülich.

1. Introduction

Turbulence is often induced by shear and one important problem in the study of fluid mechanics is to find the interaction between the mean flow and the kinetic energy of the turbulent fluctuations. The simplest flow in which to analyse this interaction is the so-called *homogeneous shear turbulence* (HST) which has a constant velocity gradient. The logarithmic layer of general wall-bounded turbulent flows has been investigated for a long time, but the mechanisms of how large-scale motions are generated and collapse into smaller eddies are not well understood. These multi-scale interactions among eddies are of great interest. The objective is to investigate if HST contains the basic energy-production and momentum-transfer characteristics of general turbulent shear flows. To do so, the key point is to achieve a high enough Reynolds number to include a sufficient range of scales. Preliminary tests have been run to determine the necessary test box sizes and Reynolds numbers. To be able to run the required simulations the model code needs to be ported to larger machine systems. Data from the computations will be compared with those of turbulent wall-bound flows at high Reynolds numbers, and the results will be made available to the community.

The SHEAR code [1] is written to do direct numerical simulation of homogeneous shear turbulence of logarithmic layers. The code is a modification of an earlier model code written for turbulent channel flow [2]. In the model, the governing equations of velocities for an incompressible flow are reduced to yield a fourth-order equation for the normal velocity component, and a second-order equation for the normal vorticity component. The third-order explicit Runge-Kutta method is applied and explicit time stepping is adopted. The model uses Fourier-expansions in the streamwise and spanwise directions, and compact finite differences in the shear direction. Domain decompositions are done in two ways: in the xz-plane for performing fast Fourier transformations along the x- and z-axis, and in the xy-plane to perform the derivatives using compact finite differences along the y-axis. In all of the computation, these domains must be transposed to each other by using two collective transposing algorithms. The baseline code is parallelised using MPI and OpenMP. The fast Fourier transformations and computing derivatives are done independently for each MPI process.

* Principal investigator. *E-mail*: dongsiwei@torroja.dmt.upm.es
** Corresponding author. *E-mail*: vegard.eide@ntnu.no

The goal of this project is to improve the performance of the SHEAR code on the Blue Gene/Q architecture, focusing on the following tasks:

- Implement parallel I/O using the HDF5 library
- Investigate scalability of the MPI communication when data need to be transposed
- Test scalability of 1D and 2D FFT using the FFTW library

2. Porting and Optimisation

The native IBM XL Fortran compiler was used to compile the SHEAR code on JUQUEEN. The `-qsmp=omp` option and was used to enable OpenMP parallelisation, and `–qnosave` was added to ensure thread-safety. Different levels of optimisation were tested, and although the `–O3` option is faster than `–O2`, it turned out to be too aggressive and affected the accuracy of the results. This was corrected by adding the `–qstrict` flag which ensures that optimisations done at the `–O3` level (and higher) do not alter the semantics of a program. The `–qhot` option was included for higher order optimizations on loops and array handling (e.g. array padding). With the `–qassert=refalign` option the compiler is requested to apply aliasing assertions to the compilation unit. The compiler will take advantage of the aliasing assertions to improve optimizations where possible. Additional compiler flags were added to generate optimised code instructions, `-qarch=qp`, and for tuning for the Blue Gene/Q architecture, `-qtune=qp`. To take advantage of the vector instructions for the Blue Gene/Q processor the code was compiled with the `–qsimd` option. The combined options used for compiling the SHEAR code on JUQUEEN was

```
-qsmp=omp –qnosave –O3 –qhot –qstrict –qarch=qp –qtune=qp –qsimd –qassert=refalign
```

The code is linked with the threaded version of the IBM ESSL library for Blue Gene/Q by adding `–lesslsmpbg` at the link stage.

The only part of the code that is slower when adding the `–qstrict` flag is the matrix solver (by 15%). This part of the code accounts for 7% of the total time, which leads to a difference of 1% of the total wall time.

3. Parallel I/O

There are two types of outputs in the SHEAR model. The first file output contains full flow fields needed for the restart of simulations, and a second file written is for time-resolved structure data. The I/O of the SHEAR code was initially done by one process only. Parallel I/O has in the scope of this project been implemented using the HDF5 library.

To take advantage of GPFS file system features the following MPI I/O hint is used

```
MPI_INFO_SET(info,”IBM_largeblock_io”,”true”,ierror)
```

To prevent conflicting access to file blocks by multiple tasks, GPFS binds each file block to a single I/O agent which control all accesses to the block, a technique called *data shipping*. For an I/O write operation, an MPI task must distribute the data to be written to I/O agents who in turn issue the write calls to GPFS. Data shipping is targeted at applications that do fine-grain writing to a file in a GPFS file system. Enabling the `IBM_largablock_io` setting will disable data shipping, i.e. write calls are issued directly to GPFS saving the overhead of going via I/O agents. This setting is recommended when writing large contiguous I/O chunks [3].

HDF5 applications can access HDF5 files using different low-level file drivers. The selected MPI POSIX file driver was specified through the call to

```
H5Pset_fapl_mpiposix_f(plist_id,MPI_COMM_WORLD,use_gpfs_hints,error)
```

This driver uses MPI communication for coordinating the actions of several tasks, and POSIX file-system calls to do the actual I/O to the disk. Available GPFS hints are known to the HDF5 library (they are not user configurable). Specifying the `use_gpfs_hints` parameter equal to `TRUE` in the function call above indicates that these hints should be used, if possible. A second parallel file system driver, the MPI-I/O driver, was tested but did not perform better than the MPI POSIX driver selected.

Despite the above settings initial runs on the JUQUEEN Blue Gene/Q system show that I/O performance is not very good, achieving only 1-2 GB/s when writing data. Additional tuning of the HDF5 I/O was investigated testing different file access settings.

The HDF5 file access property lists control properties and ways of accessing a file such as the data sieve buffer size, metadata block and cache sizes, and data alignment settings. When the file is opened, the access properties specify settings for the current access to the file. The access properties can be changed in between closing and reopening of the file.

The `H5Pset_sieve_buf_size_f` function can be used to set the size of the data sieve buffer. This is the buffer in memory that holds part of the dataset raw data. During I/O operations data is put in this buffer first, and then one big I/O request occurs. The default buffer size is 64KB. Different values in the range of 128 KB to 4 MB were tested, and the best result was achieved when increasing the size to 1MB:

```
    H5Pset_sieve_buf_size_f(plist_id,1024*1024,ierror)
```

For one test case, running a total of 1024 MPI tasks writing a 32 GB file, the following results were obtained:

Buffer size [KB]	Time to write [s]
64	32
1024	26

Table 1. File write times using different sieve buffer sizes.

The listed runtimes are averaged over several runs as results can vary when sharing the I/O infrastructure with other jobs. The results show an average reduction of 19% in the time to write the file. The HDF5 function `H5Pset_alignment_f` for specifying alignment properties of objects in a file was also tested, but did not give any noticeable change in write time.

The I/O performance of the SHEAR code on JUQUEEN can still not be considered very good. The support team in Jülich has reported that they have been and are still investigating I/O-related problems on JUQUEEN. This is still an open issue.

4. MPI communication

The communication when transposing data in SHEAR was done using calls to the `MPI_ALLTOALLV` function. This function was used since MPI tasks can have different message sizes if the global dataset cannot be evenly distributed among the processes. But test runs on JUQUEEN show that the performance using this function is not good, causing poor scalability of the code. The execution time using `MPI_ALLTOALLV` has increased dramatically from earlier test runs. A request regarding this has been issued to the Jülich support team.

The code was rewritten to use `MPI_ALLTOALL` instead by padding smaller messages with extra bytes so that each process will send the same amount of data. One test job, running 2048 MPI tasks with the total size of the matrix to be transposed equal to 32 GB, shows the following results:

	Time [s]
`MPI_ALLTOALLV`	205
`MPI_ALLTOALL`	0.61

Table 2. Communication times using MPI collectives.

The expected scaling of collective communications like `MPI_ALLTOALL` on JUQUEEN is, according to the Jülich support team, limited because of network congestion due to the network topology of Blue Gene/Q systems. The network is a 5-D torus so each node has 5 network connections. Once a job goes past 6 nodes, a message going from one node to another has to hop through all the nodes in between. As the number of nodes grow, the number of hops a message has to make increases and therefore the number of messages competing for connections between the nodes increases too.

5. Fast Fourier Transforms

The Fast Fourier Transforms of the SHEAR code can be done by either 1D or 2D FFT using functions from the FFTW library. The 1D FFT can use both the basic and advanced interface implemented in FFTW. Results show that using the basic interface in the 1D FFT is faster than the advanced interface, and also faster than the 2D FFT, see tables 3 and 4 below. The FFT in the SHEAR code is multi-threaded by hand. Testing the OpenMP version of the FFTW library did not improve performance.

# threads	16	32	64
basic	0.0110s	0.0076s	0.00748s
advanced	0.0133s	0.0101s	0.00860s

Table 3. Comparison of 1D FFT using the basic and advanced interface.

The tests in table 3 were done with a grid size of 2048x2048x32 points, running 32 MPI tasks.

# threads	16	32	64
1D FFT	11.95s	7.34s	5.81s
2D FFT	13.96s	8.54s	6.90s

Table 4. Comparison of 1D and 2D FFT of the Fourier transform of a 2D matrix.

The tests in table 4 were done with grid size of 3Kx2Kx1.5K points, with 512 MPI tasks. The 1D FFT is done using the basic interface, and the timing includes the time for transposing the matrix by hand.

6. Scaling

Scalability testing is done running a test case of 3072x2048x1536 grid points. Multi-threading performance is tested using 512 nodes, with one MPI task per node, and changing the number of threads from 1 to 64, see figure 1. Scalability up to16 threads, i.e. running on physical cores, is nearly linear. Increasing the number of

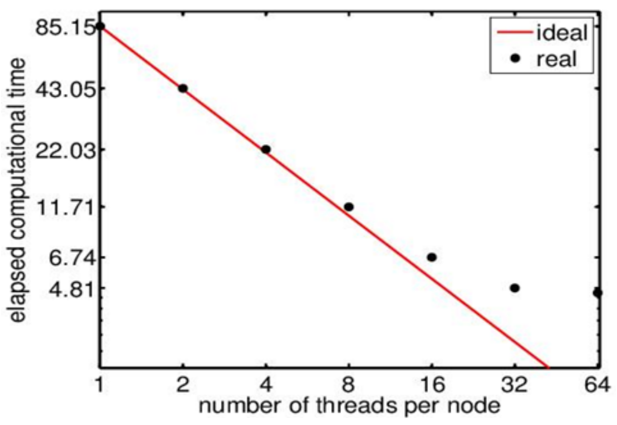

Figure 1. Multi-threading scalability.

threads further, i.e. utilizing hyper-threading, scalability decreases. Tracing of the code has shown that this is because of the FFTs that obtain only a speedup of x1.3 from x2 or x4 hyper-threading using FFTW.

Strong scaling is obtained running on 128 to 1K nodes with 1 MPI task and 64 threads per node, see figure 2. The results show that efficiency drops to 87.5% when running on 1K nodes. This is due to the communication time that does not change going from 512 to 1K nodes, and the amount of computation for the test case may not be large enough to compensate for this.

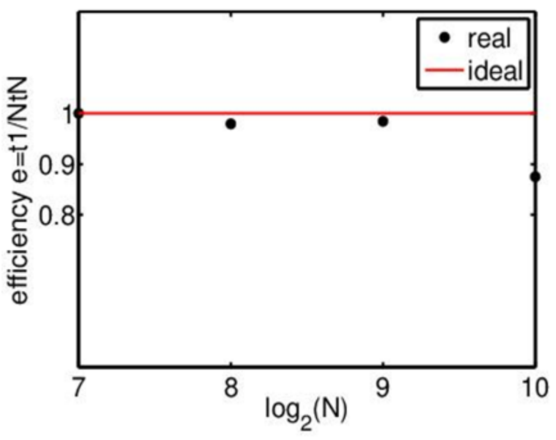

Figure 2. Strong scaling (N nodes).

Weak scaling was tested running on 128, 256, 512 and 1K nodes with a constant work load of 9M grid points per node, showing a nearly perfect scaling, see figure 3.

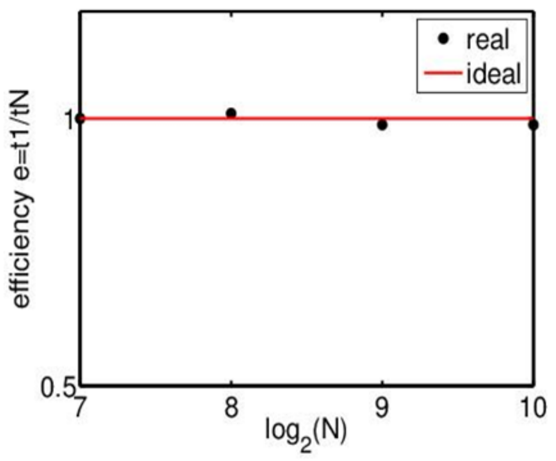

Figure 3. Weak scaling (N nodes).

7. Conclusions

The hybrid MPI/OpenMP SHEAR code is ported to the Blue Gene/Q system architecture. Parallel I/O using HDF5 is implemented for the output of restart data. The parallel I/O performance on JUQUEEN is however not very good. This may be due to I/O-related problems on the system as reported by the Jülich support team. The code shows good intra-node thread scaling running on physical cores but can only obtain approx 1.5 speedup from utilising hyper-threading. Inter-node strong scaling is good up to 1K nodes, while drop-off in efficiency when increasing the number of nodes further can be accounted for by the size of the test case, and also network congestion of the Blue Gene/Q system. Weak scaling shown is very good.

References

[1] SHEAR, code and version history can be obtained from Siwei Dong, *dongsiwei@torroja.dmt.upm.es*
[2] Guillem Borrell, Juan A. Sillero, Javier Jiménez, “A code for direct numerical simulation of turbulent boundary layers at high Reynolds numbers in BG/P supercomputers”, Computers & Fluids, 2012
[3] Katie Antypas, “Parallel IO Library Benchmarking on GPFS”, NERSC 2007

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement Nos. RI-283493 and RI-312763. The work was achieved using the PRACE Research Infrastructure (JUQUEEN) at Jülich.