A study of scalability performance for hybrid mode computation and asynchronous MPI transpose operation in DSTAR

Lucian Anton, **Ning Li**, *NAG*,
and **Kai H. Luo**, *University of Southampton*

May 9, 2011

ABSTRACT: A necessary condition for good scalability of parallel computation at large core counts is to minimise data communication and overlap it as much as possible with computation. Along these ideas we study the parallel performance of the code DSTAR which computes properties of reactive turbulent flows using direct numerical simulation. The studied algorithm uses a one-dimensional domain decomposition with OpenMP threads inside each local domain for numerical intensive kernels and asynchronous MPI or a specialised OpenMP thread for the transpose operation between domain decompositions. The new algorithm allows DSTAR to use around 10000 cores with good scalability, a significant improvement from hundreds of cores used by the initial algorithm.

KEYWORDS: mixed mode parallelism, MPI, OpenMP, reactive flow, DSTAR.

1 Introduction

The latest supercomputer generation, as the Cray XE6 used by UK national service HECtoR [1], provides unprecedented parallel computational capacity harnessed from thousands or ten of thousands of shared memory nodes linked together by a fast and large bandwidth network.

The rapidly increasing number of cores per shared memory node in the current hardware opens the possibility for algorithms with a larger degree of concurrency and which can avoid in the same time, to a certain extend, the penalty for data communication. This advantage can be used in scientific applications if the parallel programming model is extended beyond pure MPI, one of the most appealing solution being mixed mode programming (MPI+OpenMP) [2].

In this paper we present the implementation and performance results of the mixed mode programming model for the application DSTAR which has received six months of support from HECToR Computer Science and Engineering service.

DSTAR simulates, for example, multiphase reactive flows in which complex interactions exist among vortex dynamics, entrainment, mixing, turbulence, combustion and evaporating droplets. The macroscopic turbulent behaviour and thermodynamical properties are obtained using direct numerical simulation from the original governing equations. This method captures the energy transfer and interactions between chemical reactions and fluid dynamics over vastly disparate scales at the cost of refined grids for the spatial domain and a large number of time integration steps.

The system of partial differential equations that sets the mathematical frame for DSTAR and the numerical algorithms for their solution are described in detail in Ref [3]. In short, implicit compact finite difference schemes are employed for spatial discretisation with sixth-, fourth- and third-order schemes for internal, near-boundary and boundary points, respectively. A fourth-order Lagrangian interpolation scheme is utilised to obtain gas properties at droplet locations. A third-order explicit Runge–Kutta scheme is used for temporal advancement of flow variables, while a semi-analytical scheme is employed for droplet marching, with the consideration of numerical accuracy, stability and efficiency.

DSTAR parallel computations start with a

Cartesian grid partitioned uniformly to the MPI ranks along y axis. Because the spatial derivatives are computed with an implicit rule each MPI rank needs also a local grid which spans the whole domain along the y axis, hence a partition along z axis is also needed. Data between the two partitions are exchanged with transpose operations among all MPI ranks, see Figure 1 a).

The largest part of numerical intensive work done in DSTAR is concentrated in the subroutine which computes the right hand side terms (RHS) for the set of differential equations in time that describe the evolution of the flow variables after space discretisation of the continuum fields equations. RHS subroutine contains mainly loops that update the flow variables and derived quantities inside the domain and at the boundaries, calls to subroutines that compute the spatial derivatives, and calls to communication subroutines for the transpose operations described earlier.

This section of the code has a large degree of parallelism due to locality of the update rule and the multicomponent nature of the discretised flow variables. On the other hand, for a global grid with dimensions N_x, N_z, N_y the number of processors that can be used in a pure MPI parallel computation with 1D domain partition is limited by $min(N_y, N_z)$, which could be significantly smaller than the number of processing elements available on today's top supercomputers.

In our work we have explored two main approaches aimed at increasing DSTAR parallel scalability: i) 1D domain decomposition with OpenMP threads to accelerate the computation in each local domain, ii) 2D domain decomposition of the computational grid, with the help of the 2DECOMP library [4], which grants a larger number of MPI tasks, see Figure 1 b). In this paper we describe in detail the results obtained for the first approach and use the second approach as a reference point.

The paper is organised as follow: In Section 2 we describe the main features and the scaling behaviour of the four variations of mixed mode algorithm studied in for RHS subroutine. In Section 3 we discuss the MPI and OpenMP solutions for computation-communication overlap and present the timing results for each of them.

2 Mixed mode scaling

In order to reduce overheads we have chosen to use one OpenMP parallel region in RHS subroutine that includes almost all of its code lines. This choice implies that the MPI calls for the transpose operations are situated inside OpenMP parallel region which leads to three ways of doing MPI calls with respect to OpenMP threads: i) funneled, in which only the master thread calls MPI communication, ii) serialised, in which any OpenMP thread can call MPI communication, but one at a time, and iii) multiple, in which any OpenMP thread can call independently MPI communications.

As the RHS computation is done over uniform domains the simplest mixed mode algorithm would be to use "multiple" mode for MPI communication. In this way the OpenMP threads mimic the work of MPI tasks but with the advantage of shared memory. The transpose operation can be done with non-blocking send-receive operations, which also offers the possibility to investigate communication-computation overlap. The only technical challenging part of this algorithm is to organise the layout of tag and request variables needed by each OpenMP thread in its MPI communication.

It is not easy to predict which communication mode is the most efficient for DSTAR but we do not expect good performance from the "multiple" as it needs the threaded version of the Cray communication library which is not optimised [5]. For an overall picture we decided to explore all of the above variants by writing a general implementation in which one can select between various MPI communication modes. We have included also a version that uses 2DECOMP library in funneled mode in order to compare its efficiency with respect to non-blocking send receives.

Table 1 presents timing results for the MPI/OpenMP communication modes discussed previously for a cubic grid with 768^3 points, each grid point stores 8 field components. The runs were done with 768 MPI tasks and 1, 2, 3, 6, 12 OpenMP threads. PGI compiler version 11.0.0 was used, a couple of tests with other available compilers have shown similar behaviour. Each measurement was repeated at least three times to ensure that data are representative. A significant improvement in performance was noticed when the MPI environment variable `MPICH_GNI_MAX_EAGER_MSG_SIZE` was set to maximum allowed value.

The timing data show that the funneled communication model is the fastest. The "multiple" communication model, which is more straightforward to implement, loses scalability if the number of threads is larger than 3, presumably due to the non-optimised MPI threaded library.

We found somewhat unexpected the significantly slower computation when using 2DECOMP library with more that 1 OpenMP thread. We think that the explanation of this result is two-folded: i) 2DECOMP spends time to fill an internal buffer with contiguous data blocks for a more efficient MPI data transfer in the case of $y \rightarrow z$ transpose (see Figure 1) whereas in send-receive implementation of $y \rightarrow z$ transpose we made the observation that data reassembly in the internal buffer is not needed if the size of the local domain in y direction is 1; ii) 2DECOMP uses internally `MPI_ALLTOALL` for the transpose operation which is about 3 times faster than a transpose done with non-blocking send-receive when the nodes are fully populated but the speed gain levels out as the number of MPI tasks per NUMA node decreases, see Table 2. In the mixed mode version OpenMP threads are used to speed data copy to the internal buffer when this is needed. These two optimisations can be ported easily to 2DECOMP library for the special case of 1D decomposition.

Figure 2 presents the values from Table 1 and the scaling result for the algorithm that uses 2D domain decomposition with 24 processor rows. This algorithm has good scaling but it pays a significant penalty for the extra transpose operations it has to do in order to compute derivatives along x axis. Nevertheless, the scalability of this algorithm is not bounded by the maximum available number of OpenMP threads and it might be useful in cases in which more than $12 \times min(N_z, N_y)$ cores can be used (for XE6 nodes).

3 Overlapping computation with communication

Computation-communication overlap (CCO) in DSTAR is an interesting possibility because the transpose operation implies exchange of large amounts of data which takes approximately 30% of the time spent in RHS subroutine for the grid sizes used in this study. Besides that, the multicomponent nature of the flow variable makes the computational algorithm flexible enough to allow reordering of the code lines, which might be needed in order to achieve CCO.

Initially we have tried to achieve CCO using the non-blocking send receive MPI communication in the transpose subroutines. The initialisation and finalisation of communication is done in separated subroutines which can be called at convenient positions in the code in order to allow the progress of independent computation, if there is any available. CCO is achieved if data communication is done also between the initialisation and finalisation calls.

In this experiment we have measured the communication and computation time for two situations: i) the subroutines that initiate and finalise communication are consecutive in the source code or ii) a block of independent computation is placed between the two subroutines. The same model and grid size were employed as in Section 2 but this time the measurement was repeated on the XT4 component of HECToR service [1].

Data presented in Table 3 show that there is no significant progress of communication during computation in the case of XE6 system and a marginal one on XT4 system. These results are in agreement with other studies [6] which have found that various MPI implementation do not offer at present CCO when using non-blocking MPI communication.

The alternative solution for CCO is to use a mixed mode algorithm with one OpenMP thread for communication while the other threads are busy with computation.

There are several ways to implement this idea [6], of which we have selected for this study to use the master thread for MPI communication and to divide the computational work by splitting manually the loop trips to the remaining threads in the overlapped parallel loops. The code fragment for CCO with OpenMP has the following general structure:

```
!$OMP BARRIER ! Not always needed
  if ( omp_get_thread_num() == 0 ) then
!
!  master thread handles mpi communication
!
    call transpose_y_to_z(q1,p1)
  else
!
!  the rest of the team does computational work
!
    do i=isx,iex  ! isx, iex specify the
                  ! loop trip for each thread
    ...

    enddo
  endif
```

Following this pattern we have introduced CCO in the sector of RHS subroutine that covers all $y \to z$ transpose operations. Data were collected for two cubic grids with linear sizes 768 and 1536, the executable was produced by GNU compiler version 4.5.2. The runs were done with 768 or 1536 MPI tasks, respectively, and 1, 2, 3, 6, 12 OpenMP threads.

We have measured the total time spent in CCO region and the sum of communication times in this region. Shown in Table 4 is the difference between these two quantities, that is, the time spent in computation which is not covered by communication, and the communication time. One can see that that the non-overlapped computation is reduced by approximately half if the number of OpenMP threads is 6 or 12. The total attainable speed up for RHS subroutine by using CCO can be estimated with the following argument: The sector in which CCO was applied takes approximately 1/3 of RHS time and CCO procedure yields a speed up marginally larger than 20% for 6 OpenMP threads. RHS subroutine contains a similar sector in which the reverse transpose operations, $y \to z$, are mixed with computations for which we assume a similar speedup from CCO, hence the overall speed up for RHS should be in the $10 - 15\%$ range.

We close with the observation that the scalability of the mixed mode is helped also by the fact that the time for transpose operation decreases with the number of OpenMP threads, see Tables: (2, 4), most probably due to less intra-node communication contention. This result is somewhat counterintuitive because the amount of data transferred per MPI rank is independent of the number of OpenMP threads and the ratio between inter-node, intra-node connections increases with the number of threads.

4 Conclusion

We have studied a mixed mode (MPI+OpenMP) implementation for the main computational subroutine of DSTAR focused on the performance of funneled, serialised and multiple communication modes and the computation-communication overlap using non blocking MPI or OpenMP threads.

Data analysis shows a good scalability of the mixed mode code up to 12 OpenMP threads (efficiency larger than 50%). The funneled MPI communication offers best performance on Cray XE6 architecture. With a moderate code change further $10 - 15\%$ speed improvement can be obtained from the computation-communication overlap implemented with the help of OpenMP threads.

This study provides guidance for a 1D mixed mode specialisation of 2DECOMP library which should be useful for communication bound applications running on current hardware architectures.

Acknowledgements

The authors gratefully acknowledge HECToR - a Research Councils UK High End Computing Service and funding from the Engineering and Physical Sciences Research Council for Grant No. EP/I000801/1. LA thanks Kevin Roy for useful discussions.

About the authors

Lucian Anton is technical consultant at Numerical Algorithms Group (NAG). He is a member of the HECToR CSE team involved in user application support and training. He can be reached at NAG Ltd, Peter House, Oxford Street, Manchester, M1 5AN, United Kingdom, E-mail: lucian.anton@nag.co.uk.

Ning Li is a technical consultant at Numerical Algorithms Group (NAG). He is a member of the HECToR CSE team and he is the author of the 2DECOMP&FFT library. He can be reached at NAG Ltd, Wilkinson House, Jordan Hill Road, Oxford, OX2 8DR, United Kingdom, E-mail: ning.li@nag.co.uk.

Kai H. Luo is Professor of Energy Systems at School of Engineering Sciences, and Head of Energy Technology Research Group, University of Southampton, University Road, Southampton, SO17 1BJ, UK; `http://www.soton.ac.uk/ses/people/staff/LuoK.html`.

References

[1] http://www.hector.ac.uk . The test were done mainly on phase2b of HECToR service which has shared memory nodes with two AMD Magny-Cours 12 core processor (2.1 GHz clock rate, 16 GB of RAM on 2 NUMA nodes, 64KB L1 cache, 512 KB L2 cache, 6 MB L3 shared cache) and Gemini interconnect. In the period when this work was done a section of phase2a service was also available. This has single CPU nodes with AMD Barcelona quadcore processor (2.3 GHz clock rate, 8 GB of RAM, 64KB L1 cache, 512 KB L2 cache, 2 MB L3 shared cache) and SeaStar interconnect.

[2] E.g. at CUG 2010 the following papers addressed MPI+OpenMP mixed mode: Alice Koniges *et al, Application Acceleration on Current and Future Cray Platforms*; Lucian Anton *et al, Mixed Mode Computation in CASINO*; Mark Richardson *et al, Combining Open MP and MPI within GLOMAP Mode ...*; Vicenzo Fico *et al, A Hybrid MPI/OpenMP Code Employing a High-Order Compact Scheme for the Simulation of Hypersonic Aerodynamics* ; Iain Bethune *Improving the Performance of CP2K on the Cray XT*; Hongzhang Shan *et al, Analyzing the Effect of Different Programming Models Upon Performance and Memory Usage on Cray XT5 Platforms*; Glenn Luecke *et al, Performance Analysis of Pure MPI Versus MPI+OpenMP for Jacobi Iteration and a 3D FFT on the Cray XT5*

[3] J. Xia and K. H. Luo, *Conditional statistics of inert droplet effects on turbulent combustion in reacting mixing layers*, Combustion Theory and Modelling, 13:5, 901–920 (2009), and the references therein.

[4] N. Li and S. Laizet, *2DECOMP&FFT – A highly scalable 2D decomposition library and FFT interface*, Cray User Group 2010 conference, Edinburgh, 2010; http://www.hector.ac.uk/cse/distributedcse/reports/incompact3d/incompact3d/index.html

[5] mpi_intro man page, Cray MPT version 5.1.4 .

[6] Georg Hager *et al*, http://www.speedup.ch/workshops/w39_2010/slides/hager.pdf; G. Schubert, G. Hager, H. Fehske and G. Wellein arXiv:1101.0091v1

model	threads				
	1	2	3	6	12
funneled	232	127	98	58	39
serialized	233	128	97	58	42
multiple	232	136	104	106	172
2DECOMP	229	144	103	65	52

Table 1: Average computing time in seconds in RHS subroutine for each mixed mode model described in text.

	ranks per NUMA node			
	6	3	2	1
all to all	0.832	0.353	0.366	0.208
send-recv	2.44	0.824	0.480	0.244

Table 2: Maximum time in seconds for a transpose operation from a $768 \times 768 \times 1$ grid to a $768 \times 1 \times 768$ over 768 MPI ranks placed on NUMA nodes as in the case of mixed mode computations with 1, 2, 3 and 6 OpenMP threads. Placement done using `-S` flag of `aprun` command.

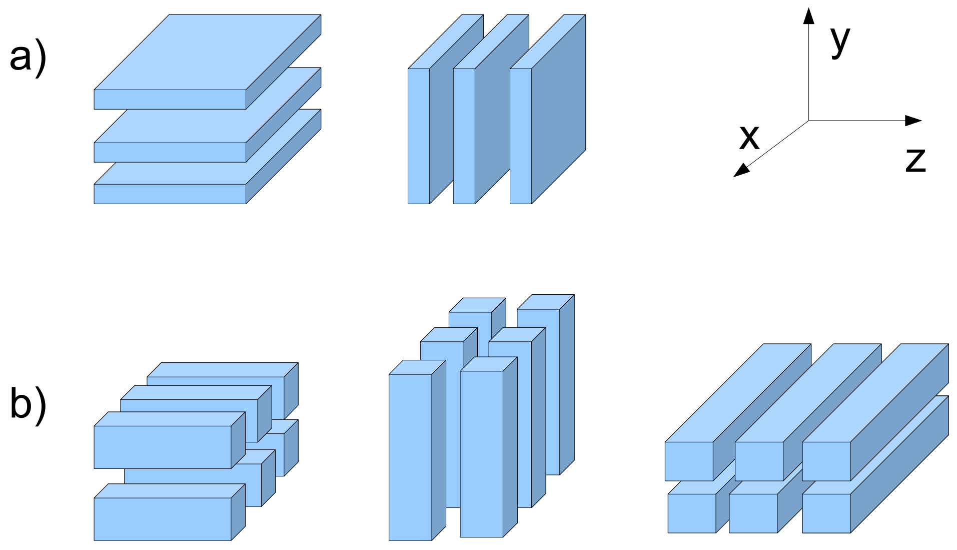

Figure 1: Schematic representation of domain decompositions: a) 1D with 3 MPI ranks along z and y directions, b) 2D with 2×3 MPI ranks along z, y and x directions. Data exchange between decompositions require all to all type of communication.

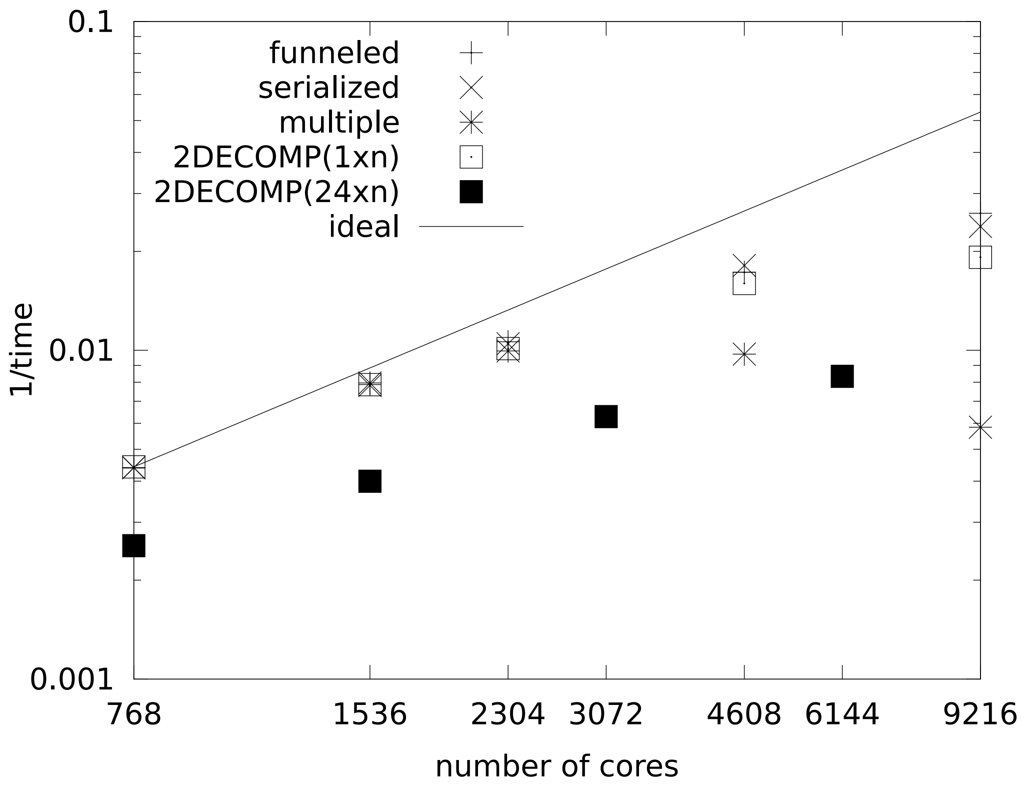

Figure 2: Mixed mode scaling for the funneled, serialised, multiple MPI communication mode (see Table 1). Timing for 2D decomposition with 24 processor rows is also shown (filled squares).

	XE6				XT4			
threads	t_{tr}	$\bar{t}_{tr}$	t_c	$\bar{t}_c$	t_{tr}	$\bar{t}_{tr}$	t_c	$\bar{t}_c$
1	2.52	16.68	13.15	13.21	2.95	17.50	15.07	15.13
2	1.57	7.81	6.64	6.62	1.51	8.63	7.37	7.39
3	1.28	5.58	4.55	4.54				
4					0.81	4.36	3.66	3.69

Table 3: Average time in seconds for transpose operation that is done as a compact block (t_{tr}) and overlapping a computation block ($\bar{t}_{tr}$) for XE6 and XT4 machines. The respective computation times t_c, $\bar{t}_c$ is approximately the same in both cases, as expected. $\bar{t}_{tr} \approx t_{tr} + t_c$ implies that none or little communication is done during the computation. This implies that the bulk of communication is done in the call of `MPI_WAITALL`.

threads	1	2	3	6	12
communication time					
786^3 grid					
t_{no}	53.80	25.32	24.61	11.08	10.21
t_{wo}		27.18	25.19	11.58	10.83
1536^3 grid					
t_{no}				73.09	38.10
t_{wo}				71.68	41.88
non-overlapped computation time					
786^3 grid					
t_{no}	58.97	30.70	21.40	12.62	7.01
t_{wo}		30.13	15.12	6.08	3.33
1536^3 grid					
t_{no}				61.70	39.22
t_{wo}				34.86	17.88

Table 4: Average time in seconds for transpose operations and computation time for regions that are not covered by the communication for the initial version of the code (t_{no}) and the code that uses CCO with help of OpenMP threads (t_{wo}). Data collected from runs on two cubic grids with linear dimensions 768 and 1536. Please note the scaling of the communication time with the number of threads, the amount of data transferred is independent of number of threads.