Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Scalability of OpenFOAM for Bio-medical Flow Simulations

Ahmet Duran[a,b]*, M. Serdar Celebi[a,c], Senol Piskin[a,c], and Mehmet Tuncel[a,c]

[a]*Istanbul Technical University, National Center for High Performance Computing of Turkey (UHeM), Istanbul 34469, Turkey*
[b]*Istanbul Technical University,Department of Mathematics, Istanbul 34469, Turkey*
[c]*Istanbul Technical University, Informatics Institute, Istanbul 34469, Turkey*

9 June 2014

Abstract

We study a bio-medical fluid flow simulation using the incompressible, laminar *OpenFOAM* solver *icoFoam* and other direct solvers (kernel class) such as *SuperLU_DIST 3.3* and *SuperLU_MCDT* (Many-Core Distributed) for the large penta-diagonal and hepta-diagonal matrices coming from the simulation of blood flow in arteries with a structured mesh domain. A realistic simulation for the sloshing of blood in the heart or vessels in the whole body is a complex problem and may take a very long time, thousands of hours, for the main tasks such as pre-processing (meshing), decomposition and solving the large linear systems. We generated the structured mesh by using *blockMesh* as a mesh generator tool. To decompose the generated mesh, we used the *decomposePar* tool. After the decomposition, we used *icoFoam* as a flow simulator/solver. For example, the total run time of a simple case is about 1500 hours without preconditioning on one core for one period of the cardiac cycle, measured on the Linux Nehalem Cluster (see [28]) available at the National Center for High Performance Computing (UHeM) (see [5]). Therefore, this important problem deserves careful consideration for usage on multi petascale or exascale systems. Our aim is to test the potential scaling capability of the fluid solver for multi petascale systems. We started from the relatively small instances for the whole simulation and solved large linear systems. We measured the wall clock time of single time steps of the simulation. This version gives important clues for a larger version of the problem. Later, we increase the problem size and the number of time steps to obtain a better picture gradually, in our general strategy. We test the performance of the solver *icoFoam* at TGCC Curie (a Tier-0 system) at CEA, France (see [21]). We consider three large sparse matrices of sizes 8 million x 8 million, 32 million x 32 million, and 64 million x 64 million. We achieved scaled speed-up for the largest matrices of 64 million x 64 million to run up to 16384 cores. In other words, we find that the scalability improves as the problem size increases for this application. This shows that there is no structural problem in the software up to this scale. This is an important and encouraging result for the problem.

Moreover, we imbedded other direct solvers (kernel class) such as SuperLU*_DIST 3.3* and *SuperLU_MCDT* in addition to the solvers provided by *OpenFOAM*. Since future exascale systems are expected to have heterogeneous and many-core distributed nodes, we believe that our SuperLU_MCDT software is a good candidate for future systems. SuperLU_MCDT worked up to 16384 cores for the large penta-diagonal matrices for 2D problems and hepta-diagonal matrices for 3D problems, coming from the incompressible blood flow simulation, without any problem.

1. Introduction

It is essential to have a fast, robust and scalable library having a potential for multi-petascale systems or future exascale systems to solve a sparse linear system AX=B coming from many science and engineering applications. In this paper, we solve sparse linear systems with large penta-diagonal and hepta-diagonal matrices coming from

* Corresponding author. *E-mail address*: aduran@itu.edu.tr.

the simulation of blood flow in arteries with a structured mesh domain. Incompressible, laminar *OpenFOAM* solver *icoFoam* is used as the flow solver.

The sloshing of blood in heart is important to understand heart rate turbulence, potential damage of the sloshing to walls of vessels and heart attack. When we examine the sinus rhythm with Q wave and T wave for a human heart seen on electrocardiography, the QT interval is a measure of the time between the beginning of the Q wave and the end of the T wave in the heart's electrical cycle (see [25]). A prolonged QT interval is considered a risk factor for ventricular tachyarrhythmias disorders and sudden death (see [26]). The QT interval is used as one of the input parameters for devices used to regulate the heart, such as cardiac pacemakers. We believe that this study is useful to contribute to the development of artificial blood vessel and temporary or permanent cardiac pacemakers which are sensitive to the sloshing of blood and various behaviours of QT interval.

Several realistic bio-fluid flow simulations have been carried out (see [1-2]). The geometries were extracted from real patients or generated using real patient data from CT or MRI scans. Measured flow rates at the vessel inlet by ultrasound technique is used to get a velocity profile of the simulation geometry inlet (see [1-4]).

In this study, we describe the speed-up based on variable problem sizes as well. In other words, we not only deal with the linear speed-up for a fixed problem size, but also scaled speed-up which is consistent with the Gustafson's law (see [22] and [23]). Focusing on the entire performance of the many cores globally rather than focusing on particular core efficiencies may provide encouraging results (see also [24]). We achieved scaled speed-up for large matrices up to 64 million x 64 million matrices and speed-up up to 16384 cores on Curie (see [21]). We observed linear speed-up up to 4096 cores on Curie for a 64 million x 64 million matrix. They are significant results for the problem. In literature, there are several scalability results using *OpenFOAM* for some iterative solvers up to one thousand cores (see [8-12] and references therein) for different applications and studies.

On the other hand, we completed the integration of direct solvers such as SuperLU_DIST (see [16]) and *SuperLU_MCDT* (see [14] and [15]) into *OpenFOAM*. We performed the scalability tests for the integration of the direct solvers into *OpenFOAM*.

The remainder of this work is organised as follows: In Section 2, the test environment is presented. In Section 3, simulation test results with *icoFoam* and *SuperLU_MCDT* solvers are discussed. Section 4 summarises this work.

2. HPC Tools and Test environment

OpenFOAM (see [7]) is an open source Computational Fluid Dynamics (CFD) toolbox, (see [29]). It is useful to simulate complex fluid flows involving turbulence, heat transfer and solid dynamics. It is a generic CFD software package with many tools for several main tasks of the simulation such as pre-processing (meshing), decomposition and solution. Here, the solver refers to not only linear system solver but also Navier Stokes solver and simulator. In this project, specifically, we used the *OpenFOAM* to simulate blood flow in arteries as an application.

Here, we generated a structured mesh by using *blockMesh* as the mesh generator. To decompose the generated mesh, we employed *decomposePar* tool. After the decomposition, we used *icoFoam* as an incompressible laminar flow simulator/solver tool. It can use several iterative linear system solvers with different pre-conditioners. Up to now, we tested a preconditioned bi-conjugate gradient linear solver with diagonal incomplete LU pre-conditioner, as an option in the *icoFoam* solver. 5 or 7 banded sparse matrix occurs at each time step. All the simulations in this study are obtained using the *OpenFOAM* 2.1.1.

The simulations were conducted on TGCC Curie (a Tier-0 system) (see [21]). It is a Linux environment with InfiniBand QDR Full Fat Tree network and a 100 GB/s bandwith disk system. Each node has 2 eight-core Intel® processors Sandy Bridge EP (E5-2680) 2.7 GHz, 64 GB of RAM and one local SSD disk. The *simple* decomposition method was used for partitioning the mesh into sub-domains. The decomposition of a matrix with the size of 32 million x 32 million elements into 8192 partitions was done in serial and took more than 2 hours. So, parallel decomposition techniques are needed when we increase the matrix size and the number of partitions.

3. Test results

The total run time of a simple case took about 1500 hours without preconditioning on one core for one period of the cardiac cycle, measured on the Linux Nehalem Cluster (see [28]) available at the National Center for High Performance Computing (UHeM) (see [5]). The run time was reduced to 15 hours with preconditioning techniques on eight cores for one period of cardiac cycle. This was a laminar, rigid wall case with a small portion

of the geometry. There were more complex simulations with longer periods (see [2, 3, 5]) that we run at the Linux Nehalem Cluster (see [28]). Also, we needed at least 10 periods of the cardiac cycle for several cases because the periodic convergence of the case occurs after 10 cycle of the simulation. So the necessary CPU time went up to a few thousand (2000-5000) hours per case. The results of this white paper shows that increasing the mesh size produces a good scale on parallel computing.

Table 1 describes a dozen of matrices coming from the simulation of blood flow. For example, mC_1M_D_t is a matrix encountered at the twelfth time step, at time 0.0006 s of the simulation where the time step size is 0.00005. The other matrices are obtained at time 0.00005 of the simulation. mC_8M matrix means 8M of cells in the fluid domain and has matrix size of 8 million x 8 million.

Table 1. Description of matrices of different number of nonzeros (NNZ) per row. The matrices come from 2D-3D meshes obtained for simple incompressible blood flow. The structured mesh produces penta or hepta-diagonal matrices. The rest of the matrices has zero elements.

Matrix	N	NNZ	NNZ/N	Origin
mC_1M	1000000	4996000	4,996	UHeM
mC_8M	8000000	39988000	4,999	UHeM
mC_16M	16000000	79984000	4,999	UHeM
mC_32M	32000000	159976000	4,999	UHeM
mC_64M	64000000	319968000	5	UHeM
mC_1M_D	1000000	6940000	6,940	UHeM
mC_1M_D_t	1000000	6940000	6,940	UHeM
mC_2M_D	2000000	13900000	6,950	UHeM
mC_4M_D	4000000	27840000	6,960	UHeM
mC_5M_D	5000000	34820000	6,964	UHeM
mC_6M_D	6000000	41800000	6,967	UHeM
mC_8M_D	8000000	55760000	6,970	UHeM

3.1. icoFoam solver results

Some of the results obtained using *OpenFOAM icoFoam* solver are shown in Figure 1 (see [3]).

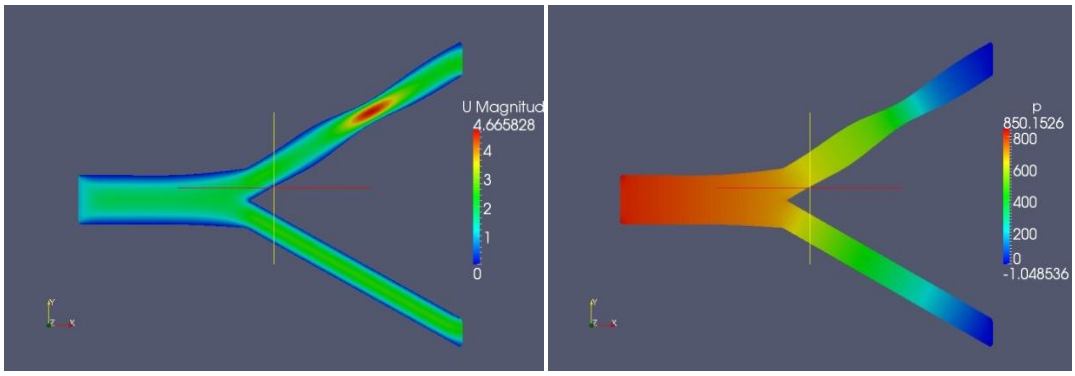

Fig. 1. The results (see [5]) obtained using *OpenFOAM icoFOAM* solver.

The tests were done for only 1 time step due to time limitations, while the real case runs are conducted for more than millions of time steps. The most time consuming part of the simulation was the decomposing of the mesh. For example, when we considered the decomposing 64M cells of data, it took over 3 hours for 8192 partitions, while it took over 7 hours for 16384 partitions. The decomposition was run on 1 core since blockMesh does not support parallel decomposition. Meshing time was the same for all partition numbers. The total meshing time for 64M cells (i.e. having problem matrix size of 64Mx64M) was about one hour. The simple decomposition method was preferred since the running cases were for a structured mesh. This technique simply splits geometry into pieces by direction, such as 32 pieces in x direction and 32 pieces in y direction. When the geometry is more complex and an unstructured mesh is used, more advanced techniques can be selected such as *METIS* (see [18]) or Scotch (see [19]). Also, since the mesh is structured, mC_64M matrix means 64M of cells in the fluid domain.

Table 2. Wall clock time and normalized speed-up for mC_8M matrix.

# of cores (meshes)	Wall clock time (s)	Speed-up
128 (16x8)	20.7	1.00
256 (16x16)	10.4	1.99
512 (32x16)	5.3	3.91
1024 (32x32)	4.7	4.40

Table 3. Wall clock time and normalized speed-up for mC_32M matrix.

# of cores (meshes)	Wall clock time (s)	Speed-up
128 (16x8)	43	1.00
256 (16x16)	23	1.87
512 (32x16)	11	3.91
1024 (32x32)	5	8.60
2048 (64x32)	2.5	17.20
4096 (64x64)	2	21.50
8192 (128x64)	2	21.50

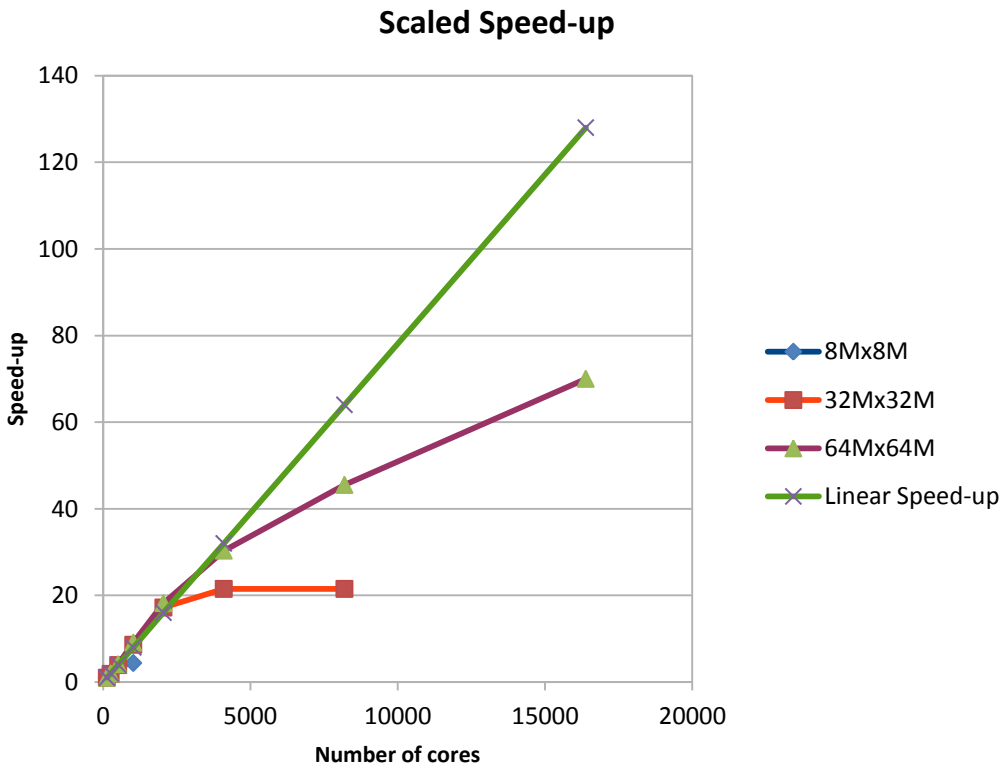

Fig. 2. Speed-up for different problem sizes

We tested several matrices of different mesh sizes. Here, we present the scalability results for three matrices of size 8 million x 8 million up to 1024 cores in Table 2; 32 million x 32 million up to 8192 cores in Table 3; and 64 million x 64 million up to 16384 cores in Table 4. The code has shown speed-up up to 16384 cores for the largest matrix in our tests. Our observation is consistent with results of other codes which show a similar behaviour. To exploit massively parallel architectures on large number of cores we need bigger size of problems. On the other hand, there may not be right match between the problem size and the available memory for the small number of cores. In calculation of speed-up, the irregular memory access of reference point for specific

core numbers and the more memory necessity for small number of cores can affect negatively in wall clock time and they cause superlinear speed-up, up to 2048 cores, since the more consuming time than the expected value on numerator. Another reason for the superlinear speed-up is due to the nonlinear characteristics of the iterative solver and the pre-conditioner performance. Moreover, we observe that the scalability becomes better as the problem size increases. Figure 2 illustrates this scaled speed-up for large matrices having sizes up to 64 million x 64 million and up to 16384 cores.

Table 4. Wall clock time and normalized speed-up for mC_64M matrix.

# of cores (meshes)	Wall clock time (s)	Speed-up
128 (16x8)	91	1.00
256 (16x16)	47	1.94
512 (32x16)	22	4.14
1024 (32x32)	10	9.10
2048 (64x32)	5	18.20
4096 (64x64)	3	30.33
8192 (128x64)	2	45.50
16384 (128x128)	1.3	70.00

3.2. SuperLU_MCDT solver results

SuperLU_MCDT is a distributed direct solver and the software will be uploaded to website (see [17]) after academic permissions from Istanbul Technical University. Here, we used symbolic factorization, *ParMETIS* (see [18]) for column permutation and Intel MKL (see [27]) as the BLAS library, among several options. The tuning of super-nodal storage parameters is important for the performance and we selected the tuned parameters relax:100 and maxsuper:110 (see [14]). Table 5 illustrates the time for the factorisation and the total time for each matrix.

We define an optimal minimum number of cores as the number of cores that provides the minimum wall clock time for a given size of problem, where a right match occurs between the problem size and the available resources such as memory, in presence of communication overhead. We find that the optimal minimum number of cores required depends on the sparsity level and size of the matrix. As the sparsity level of matrix decreases and the order of matrix increases, we expect that the optimal minimum number of cores increases slightly. For example, while 512 cores is the required minimum number of cores for mC_8M, mC_16M, mC_1M_D and mC_2M_D matrices, 2048 cores provides minimum wall clock time for mC_4M_D, mC_5M_D, mC_6M_D and mC_8M_D matrices in this portfolio of matrices.

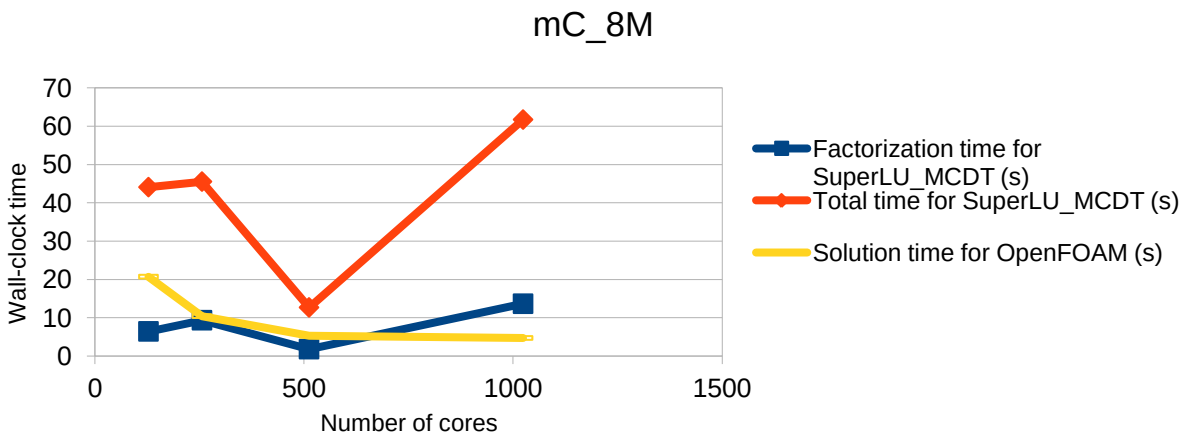

Fig. 3. The performance comparison of *SuperLU_MCDT* and *icoFoam* solver for mC_8M matrix

Table 5. Wall clock times (s) of *SuperLU_MCDT* for the large penta-diagonal matrices for 2D problems and hepta-diagonal matrices for 3D problems, described in Table 1, coming from the incompressible blood flow simulation

Matrices	# of cores (meshes)	256 (16x16)	512 (16x32)	1024 (32x32)	2048 (32x64)	4096 (64x64)	8192 (64x128)	16384 (128x128)
mC_1M	Factor Time	1,96	1,97	3,29	4,62	10,77	14,71	32,95
	Total Time	**9,45**	12,71	17,83	25,6	72,04	75,59	197,61
mC_8M	Factor Time	9,34	1,79	13,64	13,87	25,33	43,46	90,98
	Total Time	45,53	**12,68**	61,71	97,08	145,22	201,96	418,53
mC_16M	Factor Time	15	15,68	25,65	25,29	46,73	70,07	156,88
	Total Time	80,53	**67,04**	95,88	141,78	198,08	308,80	461,76
mC_1M_D	Factor Time	21,01	2,71	18,23	15,14	17	26,79	53,57
	Total Time	40,86	**14,77**	45,29	59,67	55,61	106,7	172,1
mC_1M_D_t	Factor Time	16,23	10,77	19,77	19,62	43,04	28,98	57,70
	Total Time	**38,43**	42,78	51,07	50,55	97,66	120,29	349,97
mC_2M_D	Factor Time	43,35	28,11	27,83	23,85	37,99	48,56	98,32
	Total Time	87,64	**82,1**	84,18	87,23	118,98	160,26	377,6
mC_4M_D	Factor Time	148,99	95,7	82,51	56,88	80,91	80,81	219,4
	Total Time	224,67	201,77	212,51	**173,71**	224,44	403,14	561,99
mC_5M_D	Factor Time	194,31	118,73	113,44	66,22	103,82	104,74	271,77
	Total Time	328,13	246,09	255,6	**215**	284,61	301,6	659,86
mC_6M_D	Factor Time	265,07	163,34	150,03	117,96	129,19	136,56	226,23
	Total Time	438,46	318,71	330,46	**303,24**	340,88	378,73	558,22
mC_8M_D	Factor Time	478,27	278,96	244,23	175,19	184,89	179,49	330.24
	Total Time	712,99	493,45	494,23	**441,53**	473,72	491,27	678.04

Overall, we observed that the wall clock time with *SuperLU_MCDT* is longer than that of *icoFoam* solver, as expected, because generally direct solvers take longer time than iterative solvers. For example, *icoFoam* solver outperforms *SuperLU_MCDT* for mC_8M matrix (see Figure 3). We obtained almost similar results with *SuperLU_DIST 3.3* for this set of matrices.

Table 6. Distribution of wall clock time (s) for mC_8M matrix using *ParMETIS* for column permutation

# of cores	256	512	1024	2048	4096	8192	16384
(mesh)	(16 X 16)	(16 X 32)	(32 X 32)	(32 X 64)	(64 X 64)	(64 X 128)	(128 X 128)
Nonzeros in L	736867161	**80858737**	759889256	765376719	692260216	700475156	690287571
Nonzeros in U	736867161	**80858737**	759889256	765376719	692260216	700475156	690287571
nonzeros in L+U	1465734322	**160717474**	1511778512	1522753438	1376520432	1392950312	1372575142
nonzeros in LSUB	102386047	**11558966**	106262844	108045660	94662608	97338383	96491385
# of super nodes	204238	**26847**	207025	208620	215465	214535	217216
Equil time	0,39	0,27	0,53	1,41	2,07	2,23	6,05
RowPerm time	2,18	0,27	2,17	2,18	2,18	2,2	2,17
ColPerm time	5,54	**8,63**	**31,12**	**66,29**	**102,04**	**139,54**	**301,12**
SymbFact time	3,92	0,41	4,07	4,1	3,57	3,66	3,63
Distribute time	1,07	0,24	0,75	0,76	0,69	0,92	1,68
Factor time	9,34	1,79	13,64	13,87	25,33	43,46	90,98
Solve time	3,33	0,01	1,59	1,88	1,59	1,85	2,05
Refinement time	19,76	1,06	7,84	6,59	7,75	8,1	10,85
||X-Xtrue||/||X||	1,18E-012	4,06E-011	1,80E-012	2,35E-012	1,12E-012	1,08E-012	1,10E-012
Total time (s)	45,53	**12,68**	61,71	97,08	145,22	201,96	418,53

We find that the communication overhead coming from *ParMETIS* [18] becomes one of the dominating factors in the distribution of wall clock time on the large sparse matrices for certain large numbers of cores, for example greater than 256 cores, depending on the pattern, sparsity level and order of matrix, consistent with the results of

Duran et al. [13]. For example, Table 6 shows the distribution of wall clock time (s) for mC_8M matrix and the impact of number of supernodes and the communication overhead coming from *ParMETIS* on the performance. We obtained similar results for the other matrices in Table 1. Figure 4 and Figure 5 show the performance of *SuperLU_MCDT* for mC_1M_D_t and mC_8M_D, respectively.

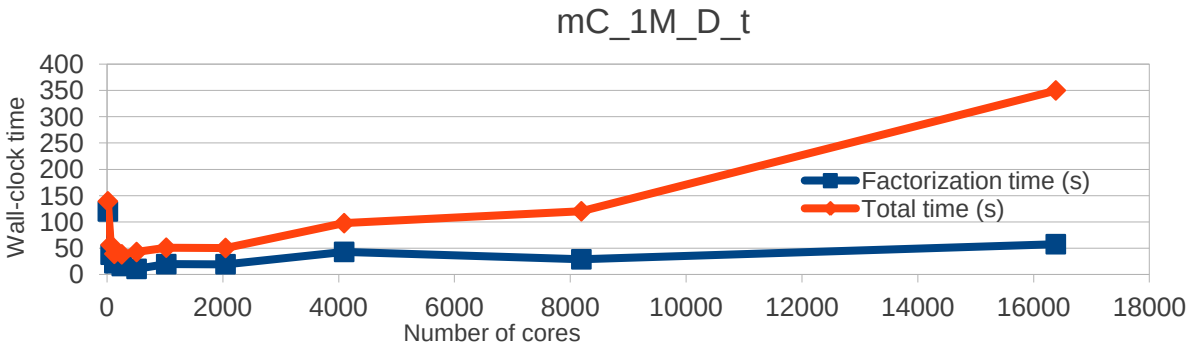

Fig. 4. The performance of *SuperLU_MCDT* for mC_1M_D_t

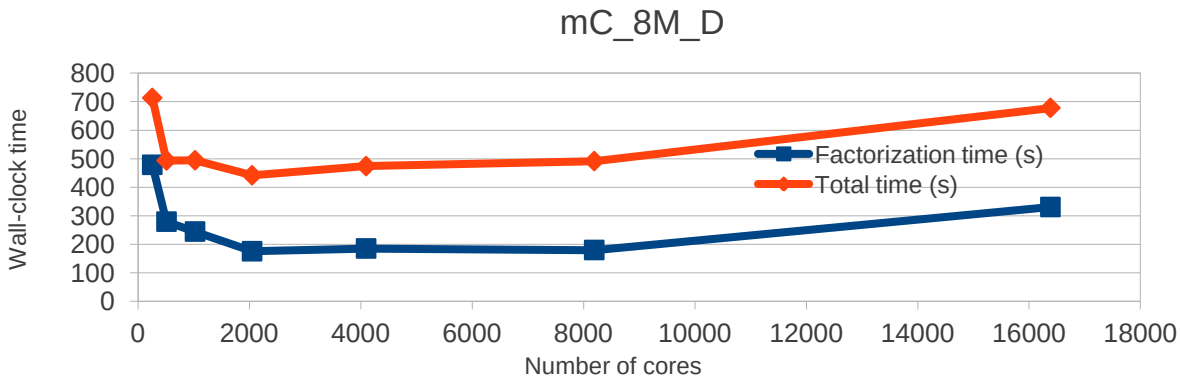

Fig. 5. The performance of *SuperLU_MCDT* for mC_8M_D

4. Summary and future work

In this study, we tested the scalability of the existing *OpenFOAM* solver *icoFoam* and *SuperLU_MCDT* for various sizes of penta-diagonal and hepta-diagonal matrices coming from the simulation of blood flow in arteries with structured mesh domain. We observe speed-up up to 16384 cores for the largest matrix in our tests using *icoFoam*. Moreover, we find that the scalability becomes better as the problem size grows. Figure 2 shows this scaled speed-up for large matrices having sizes up to 64 million x 64 million and up to 16384 cores. We observed linear speed-up up to 4096 cores on Curie (see [21]) for a 64 million x 64 million matrix.

Moreover, we imbedded other direct solvers (kernel class) such as SuperLU_DIST (see [16] and [20]) and *SuperLU_MCDT* (see [13], [14] and references therein) in addition to default solvers provided by *OpenFOAM*. In general, we obtained reasonable performance results with *SuperLU_MCDT*, in addition to the advantages of the direct solvers for robustness.

In order to show better usability of our direct method compared with iterative methods in blood flow simulations, the coefficient matrices with high condition number in transient flow conditions need to be tested. It is important to observe that, for high condition numbers, the time difference between direct and iterative solvers for the solution of linear set of equations will be reduced. For more complex flow conditions the solution time of iterative solvers will increase based on fixed solution precision. In this case, the cost of direct methods is still fixed and the potential gap is expected to be reduced. In order to show better usability of our many-core enabled direct method compared with iterative methods in blood flow simulations, the coefficient matrices with high condition number in transient flow conditions have to be tested. It is well known fact that during the flow simulations both coefficient matrices and right hand side vector are changing. This change is especially drastic

during the severe flow dynamics conditions in simulation. This drastic change, in most cases, shows itself as an ill-conditioned spectral space and high condition numbers. It is important to observe that, for high condition numbers, the time complexity gap between direct and iterative solvers for the solution of linear set of equations will be reduced. For more complex flow conditions the solution time of iterative solvers will increase based on fixed solution precision. In this case direct method's cost is still fixed and the potential gap in solution time is expected to be reduced. It is also worth noting that iterative methods works on both coefficient matrix and the right hand side vector changing at each time step but our direct solver works only on coefficient matrix. This is also a potential advantage for our direct solver in case of large simulation times.

Our many-core aware direct sparse solver has a capability of exploiting the potential benefits of many-core distributed systems than any other sparse direct solvers especially for unsymmetric matrices. Future exascale systems are expected to be having heterogeneous and many-core distributed nodes. We believe that our *SuperLU_MCDT* software is a good candidate for these future systems with its scalability on the servers we tested. While *SuperLU_MCDT* worked up to 16384 cores for the large sparse matrices coming from the incompressible blood flow simulation without any problem, there was observable performance gain up to 2048 cores for the sufficiently large matrices, for example, mC_4M_D, mC_5M_D, mC_6M_D and mC_8M_D matrices having 7 number of nonzeros per row. Potential challenges for our many-core aware software is resilience, accelerator support and hyper graph partitioning for both better scalability and sustainability**.** We make efforts to minimize the communication overhead coming from *ParMETIS* for large number of cores and search for alternative solutions.

As a future work we will test *icoFoam* simulator with other iterative solvers such as generalised geometric algebraic multi-grid and incomplete Cholesky preconditioned conjugate gradient. Also, we will test other flow simulators such as *nonNewtonianIcoFoam* or *pisoFoam*. *nonNewtonianIcoFoam* is used to simulate non-newtonian flows while *pisoFoam* is for incompressible turbulent flow.

References

[1] S. Piskin, M. S. Celebi, Analysis of the effects of different pulsatile inlet profiles on the hemodynamical properties of blood flow in patient specific carotid artery with stenosis, Computers in Biology and Medicine, Volume 43, Issue 6, 1 July 2013, Pages 717-728, ISSN 0010-4825, http://dx.doi.org/10.1016/j.compbiomed.2013.02.014

[2] S. Piskin, M. S. Celebi, "Numerical blood flow simulation with predefined artery movement," Biomedical Engineering and Informatics (BMEI), 2012 5th International Conference, pp.654,658, 16-18 Oct. 2012 doi: 10.1109/BMEI.2012.6513039

[3] S. Piskin and M. S. Celebi, Bir boyutlu damar hareketi ile sayısal kan akısı benzetimi (The analogy between one dimensional blood vessel movement and numerical blood flux), Tıp Teknolojileri Ulusal Kongresi - TIPTEKNO 12, Antalya, Turkey, November 1-3, 2012

[4] H. Turkeri, S. Piskin, and M. S. Celebi, A comparison between non-Newtonian and Newtonian blood viscosity models, Journal of Biomechanics, 44, Supplement 1, 2011

[5] S. Piskin and A. Akkus¸ Biofluid flow applications by open-source software, 17. National Biomedical Engineering Meeting - BIYOMUT 2012, Istanbul, Turkey, October 3-5, 2012

[6] D7.2.1 A Report on the Survey of HPC Tools and Techniques, PRACE-3IP, April 25, 2013, http://www.prace-project.eu/IMG/pdf/d7.2.1.pdf

[7] *OpenFOAM* main site, http://www.openfoam.com

[8] P. Dagna and J. Hertzer, Evaluation of multi-threaded OpenFOAM hybridization for massively parallel architectures, PRACE WP98, Aug. 20, 2013, www.prace-project.eu/IMG/pdf/wp98.pdf

[9] M. Manguoglu, PRACE WP, Sep. 6, 2012, http://www.praceproject.eu/IMG/pdf/A_General_Sparse_Sparse_Linear_System_Solver_and_Its_Application_in_OpenFOAM-2.pdf

[10] M. Culpo, PRACE WP, Sep. 6, 2012, http://www.prace-ri.eu/IMG/pdf/Current_Bottlenecks_in_the_Scalability_of_OpenFOAM_on_Massively_Parallel_Clusters-2.pdf

[11] M. Moylesa, P. Nash, and Ivan Girotto, PRACE WP, Sep. 6, 2012, http://www.prace-ri.eu/IMG/pdf/Performance_Analysis_of_Fluid-Structure_Interactions_using_OpenFOAM.pdf

[12] T. Behrens, OpenFOAM's basic solvers for linear systems of equations: Solvers, preconditioners, smoothers, Tech. Rep. DTU, Denmark, Feb. 18, 2009, http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2008/TimBehrens/tibeh-report-fin.pdf

[13] A. Duran, M.S. Celebi, M. Tuncel and B. Akaydın, Design and implementation of new hybrid algorithm and solver on CPU for large sparse linear systems, PRACE-2IP white paper, Libraries, WP 43, July 13, 2012, http://www.prace-ri.eu/IMG/pdf/wp43-newhybridalgorithmfo_lsls.pdf
[14] A. Duran, M.S. Celebi, M. Tuncel, and F. Oztoprak. Structural analysis of large sparse matrices for scalable direct solvers. PRACE-2IP white paper, Scalable algorithms, WP 82, August 20, 2013, http://www.prace-project.eu/IMG/pdf/wp82.pdf
[15] M.S. Celebi, A. Duran, M. Tuncel, B. Akaydin and F. Oztoprak, Performance analysis of BLAS libraries in SuperLU_DIST for SuperLU_MCDT (Multi Core Distributed) development, PRACE-2IP white paper, Libraries, WP 83, August 20, 2013, http://www.prace-project.eu/IMG/pdf/wp83.pdf
[16] Xiaoye S. Li and James W. Demmel, SuperLU_DIST: A Scalable Distributed-Memory Sparse Direct Solver for Unsymmetric Linear Systems, ACM Trans. on Math. Software, Vol. 29, No. 2, June 2003, pp. 110-140.
[17] http://www.uybhm.itu.edu.tr
[18] (Par)METIS homesite: http://www.lrz.de/services/software/mathematik/metis
[19] Scotch and PT-Scotch homepage http://www.labri.fr/perso/pelegrin/scotch
[20] X. S. Li, J. W. Demmel, J. R. Gilbert, L. Grigori, M. Shao, and I. Yamazaki, SuperLU Users' Guide, Tech. Report UCB, Computer Science Division, University of California, Berkeley, CA, 1999, update: 2011
[21] http://www-hpc.cea.fr/en/complexe/tgcc-curie.htm
[22] J. L. Gustafson, Reevaluating Amdahl's Law, Communications of the ACM 31(5), 1988. pp. 532-533
[23] http://en.wikipedia.org/wiki/Gustafson's_law
[24] M. D. Hill and M. R. Marty, Amdahl's Law in the Multicore Era, IEEE Computer, vol. 41, pp. 33–38, July 2008
[25] http://en.wikipedia.org/wiki/QT_interval
[26] http://en.wikipedia.org/wiki/Electrocardiography
[27] http://software.intel.com/en-us/intel-mkl
[28] http://www.uybhm.itu.edu.tr/eng/inner/duyurular.html#karadeniz
[29] K.A. Hoffmann and S.T. Chiang, Computational Fluid Dynamics, Engineering Education System Vol. I and II, 2000.

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 no. RI-312763. The work was achieved using the PRACE Research Infrastructure resource Curie at CEA, France (see [21]). Moreover, computing resources of the National Center for High Performance Computing of Turkey (UHeM) (see [17]) were used under grant number 1001682012.