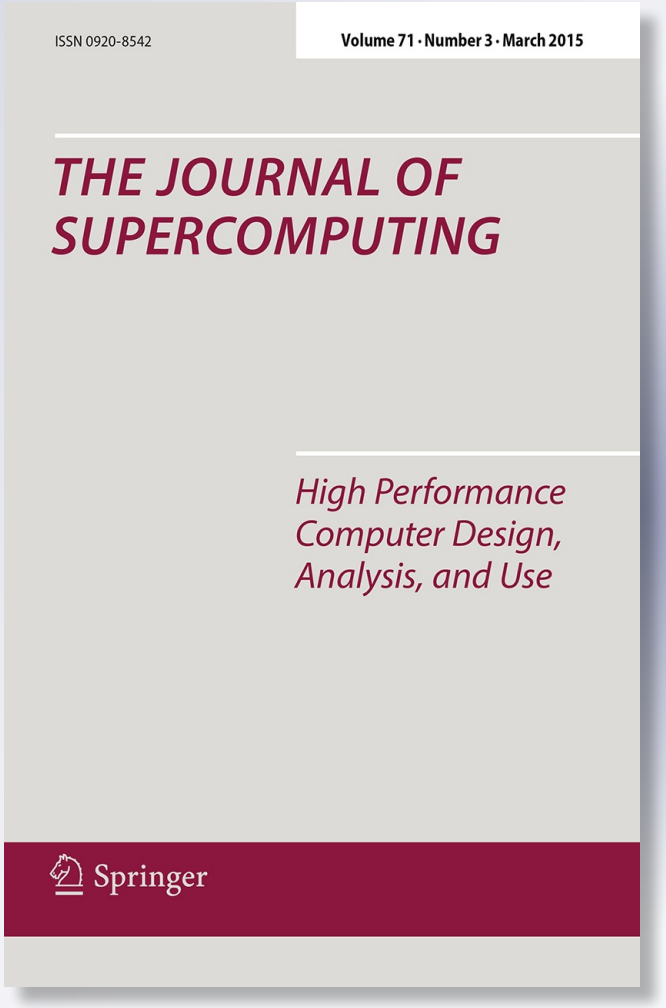
Scalability of OpenFOAM for bio-medical flow simulations

Ahmet Duran, M. Serdar Celebi, Senol Piskin & Mehmet Tuncel

The Journal of Supercomputing
An International Journal of High-Performance Computer Design, Analysis, and Use

ISSN 0920-8542
Volume 71
Number 3

J Supercomput (2015) 71:938-951
DOI 10.1007/s11227-014-1344-1



Springer

J Supercomput (2015) 71:938–951
DOI 10.1007/s11227-014-1344-1

Scalability of OpenFOAM for bio-medical flow simulations

Ahmet Duran · M. Serdar Celebi · Senol Piskin · Mehmet Tuncel

Published online: 28 November 2014

Abstract We study a bio-medical fluid flow simulation using the incompressible, laminar *OpenFOAM* flow solver *icoFoam* using iterative linear equation solver and 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 flow 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. 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 [31]). We achieved scaled speed-up for the largest matrices of 64 million $\times$64 million in our dataset to run up to 16,384 cores. In other words, we find that the scalability improves as the problem size increases for this application. As the matrix size quadrupled, the speed-up improves at least 50 %

A. Duran (✉) · M. Tuncel
Department of Mathematics, Istanbul Technical University, Istanbul 34469, Turkey
e-mail: aduran@itu.edu.tr

A. Duran
National Center for High Performance Computing of Turkey (UHeM), Istanbul Technical University, Istanbul 34469, Turkey

M. S. Celebi · S. Piskin · M. Tuncel
Informatics Institute, Istanbul Technical University, Istanbul 34469, Turkey

near speed-up saturation point. 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 16,384 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.

Keywords OpenFOAM software · Solver · Bio-medical flow · Scalability

1 Introduction

It is important 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 bio-medical flow applications. In this paper, we solve sparse linear systems with large penta-diagonal and hepta-diagonal matrices coming from the simulation of blood flow in arteries with a structured mesh domain. Incompressible, laminar *OpenFOAM* solver *icoFoam* is used as the flow solver.

We generate the structured mesh using *blockMesh* as a mesh generator tool. To decompose the generated mesh, we use the *decomposePar* tool. After the decomposition, we use *icoFoam* as a flow simulator/solver. For example, the total run time of a simple case is about 1,500 h without pre-conditioning on one core for one period of the cardiac cycle, measured on the Linux Nehalem Cluster (see [33]) available at the National Center for High Performance Computing (UHeM) (see [32]). Therefore, this important problem deserves careful consideration for usage on multi-petascale or exascale systems.

On the other hand, we completed the integration of direct solvers such as *SuperLU_DIST* (see [11]) and *SuperLU_MCDT* (Many-Core Distributed) (see [2] and [5]) into *OpenFOAM* in addition to the solvers provided by *OpenFOAM*. We performed the scalability tests for the integration of the direct solvers into *OpenFOAM*.

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 [29]). A prolonged QT interval is considered a risk factor for ventricular tachyarrhythmia disorders and sudden death (see [27]). 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 behaviors of QT interval.

The simulation of blood flow via OpenFOAM has attracted more attention recently in the literature. For example, Pal et al. [15] studied large eddy simulation of tur-

bulence for axis symmetric blood flow and compared three different subgrid scales. Moreover, Wu et al. [22] investigates two-dimensional flow of blood in a rectangular microfluidic channel. Furthermore, Kelly et al. [10] describes the use of fluid, solid and fluid–structure interaction simulations on patient-specific abdominal aortic aneurysms. Several realistic bio-fluid flow simulations have been carried out as well (see [17,18]). 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 are used to get a velocity profile of the simulation geometry inlet (see [17–19] and [21]). Turkeri et al. ([21]) investigates the effects of several rheological models on blood flow in patient-specific carotid artery geometry with measured inlet flow data. It is important to use real geometry and experimental data to analyze clinical outputs of several medicines used for humans.

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 [7,28]). Focusing on the entire performance of the many cores globally rather than focusing on particular core efficiencies may provide encouraging results (see also [8]). In the literature, there are several scalability results using OpenFOAM for some iterative solvers up to 1,000 cores (see [1,3,6,13,14,20] and references therein) for different applications and studies. Culpo [3] investigates the scaling behavior of different *OpenFOAM* versions on benchmark problems. He finds that the applications scale well up to 1,000 tasks. He also investigates in depth for several MPI routines that cause communication bottleneck and proposes solution for them. Pringle [20] presents the installation of *OpenFOAM* versions 1.6 and 1.5 on HECToR, the UK National Supercomputing service. He defines the optimum number of cores for a given simulation using iterative solvers such as *icoFoam*, *interFoam* and *pisoFoam* as the largest number of cores where the performance is greater than 1.4. He argues the absence of simple relation between the optimum number of cores to be used for a given iterative solver and a given number of cells, based on his tests up to 4,096 cores.

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

2 HPC tools, test environment and flow of approach

OpenFOAM (see [24]) is an open-source Computational Fluid Dynamics (CFD) toolbox, (see [9]). 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 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 pre-conditioned bi-conjugate gradient linear solver with diagonal incomplete LU pre-conditioner, as an option in the *icoFoam* solver. Five or seven-banded sparse matrix occurs at each time step. All the simulations in this study are obtained using the *OpenFOAM* 2.1.1. We present the flowchart in Fig. 1 to explain the flow of approach in the paper.

The simulations were conducted on TGCC Curie (a Tier-0 system) (see [31]). 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 $\times$32 million elements into 8,192 partitions was done in serial and took more than 2 h. 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 1,500 h without preconditioning on one core for one period of the cardiac cycle, measured on the Linux Nehalem Cluster (see [33]) available at the National Center for High Performance Computing (UHeM) (see [32]). The run time was reduced to 15 h 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 [16,18,19]) that we run at the Linux Nehalem Cluster (see [33]). 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 (2,000–5,000) h per case. The results of this study show 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 (s). The other matrices are obtained at time 0.00005 (s) of the simulation. mC_8M matrix means 8M of cells in the fluid domain and has matrix size of 8 million $\times$8 million.

3.1 icoFoam solver results

Some of the results obtained using *OpenFOAM icoFoam* solver are shown in Fig. 2 (see [19]). The results were obtained after establishing periodic convergence in time. The figure on the left and right shows the developed velocity and pressure distributions, respectively, for a sample time step. The distributions were taken at a cross section (symmetry of z axis) of the three-dimensional (3D) carotid artery geometry. The geometry consists of common, internal and external carotid arteries with one bifurcation. There is a stenosis (narrowing) at the sinus of internal carotid artery. The inlet of the flow is pulsatile velocity profile and the outlets are assigned as pressure outlet. The walls have no-slip boundary condition.

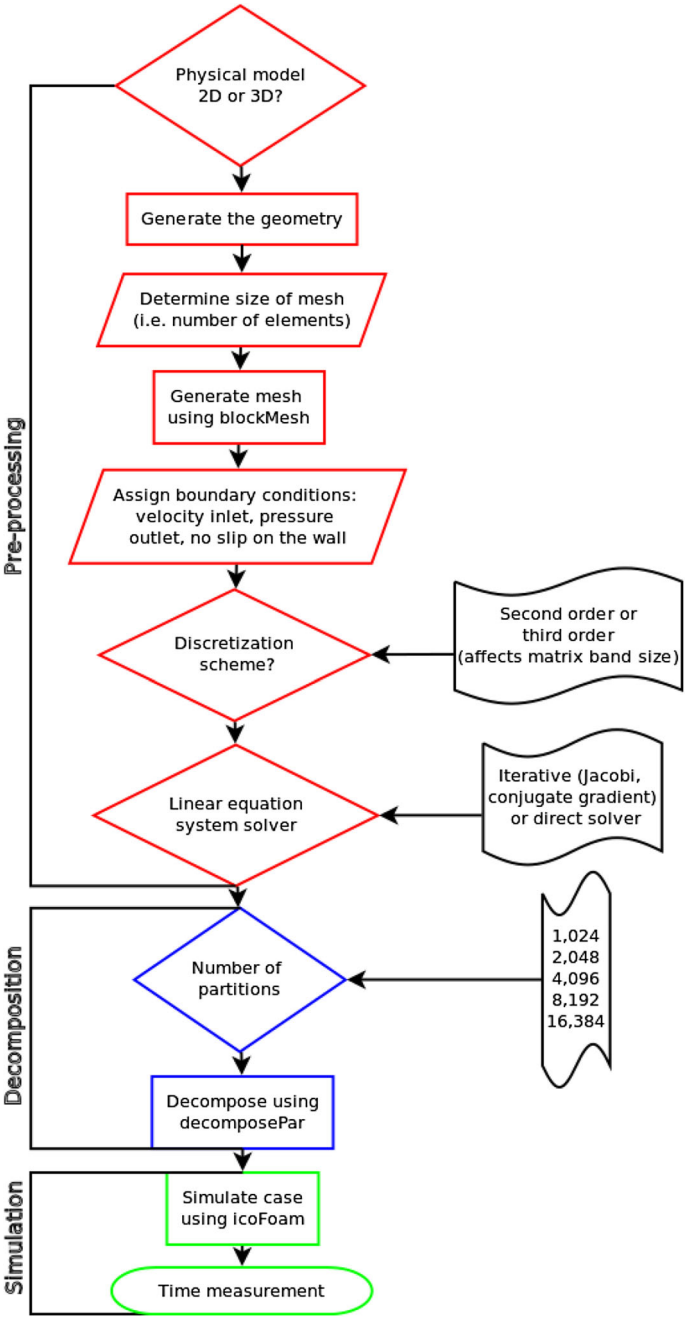


Fig. 1 Flowchart for the flow of the approach including the main tasks

Table 1 Description of matrices of different number of nonzeros (NNZ) per row

Matrix	*N*	NNZ	NNZ/N	Origin
mC_1M	1,000,000	4,996,000	4.996	UHeM
mC_8M	8,000,000	39,988,000	4.999	UHeM
mC_16M	16,000,000	79,984,000	4.999	UHeM
mC_32M	32,000,000	159,976,000	4.999	UHeM
mC_64M	64,000,000	319,968,000	5	UHeM
mC_1M_D	1,000,000	6,940,000	6.940	UHeM
mC_1M_D_t	1,000,000	6,940,000	6.940	UHeM
mC_2M_D	2,000,000	13,900,000	6.950	UHeM
mC_4M_D	4,000,000	27,840,000	6.960	UHeM
mC_5M_D	5,000,000	34,820,000	6.964	UHeM
mC_6M_D	6,000,000	41,800,000	6.967	UHeM
mC_8M_D	8,000,000	55,760,000	6.970	UHeM

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

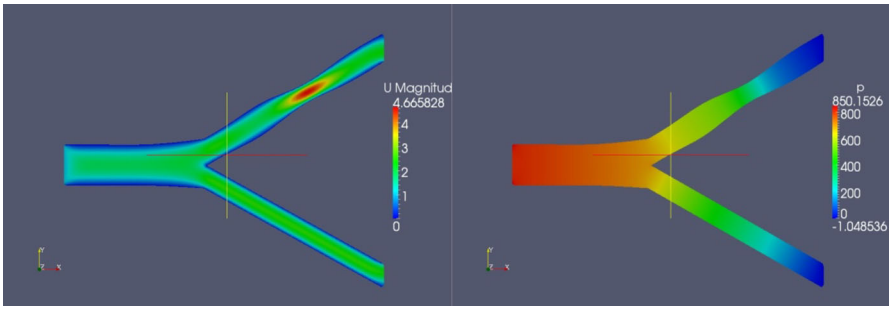


Fig. 2 The results (see [16]) obtained using *OpenFOAM icoFOAM* solver

The tests were done for only one 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 h for 8,192 partitions, while it took over 7 h for 16,384 partitions. The decomposition was run on one 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 64M $\times$ 64M) was about 1 h. 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 [25]) or Scotch (see [26]). Also, mC_64M matrix means 64M of cells in the fluid domain.

We tested several matrices of different mesh sizes. Here, we present the scalability results for three matrices of size 8 million $\times$ 8 million up to 1,024 cores in Table 2;

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

# Of cores (meshes)	Wall clock time (s)	Speed-up
128 (16 × 8)	20.7	1.00
256 (16 × 16)	10.4	1.99
512 (32 × 16)	5.3	3.91
1,024 (32 × 32)	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 (16 × 8)	43	1.00
256 (16 × 16)	23	1.87
512 (32 × 16)	11	3.91
1,024 (32 × 32)	5	8.60
2,048 (64 × 32)	2.5	17.20
4,096 (64 × 64)	2	21.50
8,192 (128 × 64)	2	21.50

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

# Of cores (meshes)	Wall clock time (s)	Speed-up
128 (16 × 8)	91	1.00
256 (16 × 16)	47	1.94
512 (32 × 16)	22	4.14
1,024 (32 × 32)	10	9.10
2,048 (64 × 32)	5	18.20
4,096 (64 × 64)	3	30.33
8,192 (128 × 64)	2	45.50
16,384 (128 × 128)	1.3	70.00

32 million×32 million up to 8,192 cores in Table 3; and 64 million × 64 million up to 16,384 cores in Table 4. The code has shown speed-up up to 16,384 cores for the largest matrix in our tests. Our observation is consistent with results of other codes which show a similar behavior. 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 2,048 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.

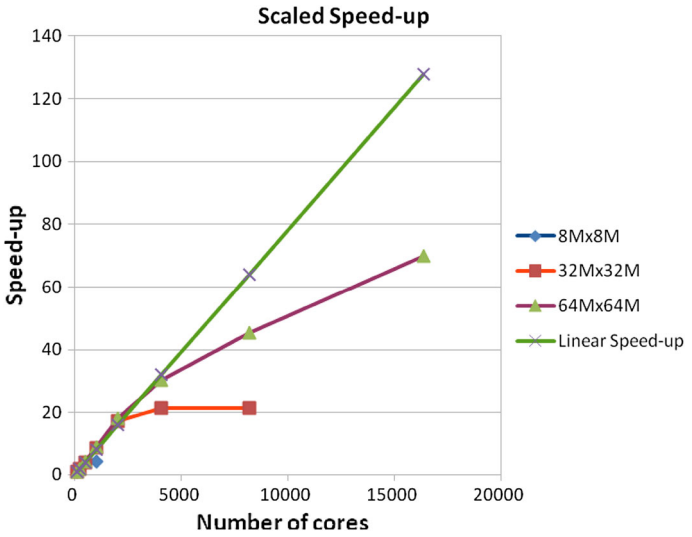


Fig. 3 Speed-up for different problem sizes

For example, for 4,096 cores the speed-up became 30.33 for 64 million × 64 million matrix while it was 21.5 for 32 million ×32 million matrix. That is, approximately 50 % improvement was achieved in speed-up as the size quadrupled. For 8,192 cores, even better improvement was obtained. Figure 3 illustrates this scaled speed-up for large matrices having sizes up to 64 million× 64 million and up to 16,384 cores.

4 SuperLU_MCDT solver results

SuperLU_MCDT is a distributed direct solver and the software will be uploaded to web site (see [32]) after academic permissions from Istanbul Technical University. Here, we used symbolic factorization, *ParMETIS* (see [25]) for column permutation and Intel MKL (see [30]) 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 [5]). 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, 2,048 cores provide minimum wall clock time for mC_4M_D, mC_5M_D, mC_6M_D and

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 (16 × 16)	512 (16 × 32)	1,024 (32 × 32)	2,048 (32 × 64)	4,096 (64 × 64)	8,192 (64 × 128)	16,384 (128×128)
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

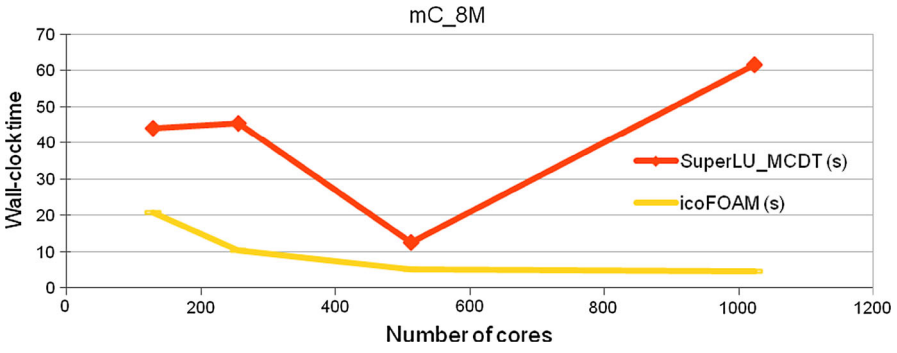


Fig. 4 The performance comparison of *SuperLU_MCDT* and *icoFoam* solvers for mC_8M matrix

mC_8M_D matrices in this portfolio of matrices (see Table 5). The minimum wall clock time values are written in bold.

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, Fig. 4 shows that *icoFoam* solver outper-

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

# Of cores (mesh)	256 (16 × 16)	512 (16 × 32)	1,024 (32 × 32)	2,048 (32 × 64)	4,096 (64 × 64)	8,192 (64 × 128)	16,384 (128 × 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

forms *SuperLU_MCDT* for mC_8M matrix. The wall clock time for *SuperLU_MCDT* becomes closest to that of *icoFoam* solver around 512 cores. Later, it diverges. On the other hand, direct solvers are more robust and may provide smaller error. For example, the error of *SuperLU_MCDT* for mC_8M matrix is around 4.06E−011, while the error for *icoFoam* solver is approximately 9.8E−07 with 2,100 iterations. Both errors seem to be sufficiently small for this application. The error of iterative solver can be sensitive to the number of iterations and the matrices. We obtained almost similar results with *SuperLU_DIST 3.3* for this set of matrices.

We find that the communication overhead coming from *ParMETIS* [25] 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. [4]. For example, Table 6 shows the distribution of wall clock time (s) for mC_8M matrix and the impact of number of super-nodes and the communication over-

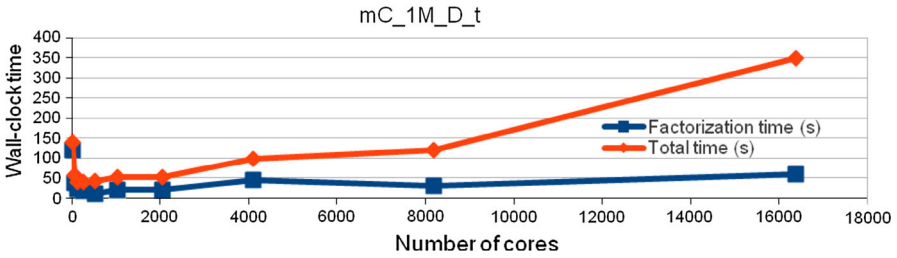


Fig. 5 The performance of *SuperLU_MCDT* for mC_1M_D_t

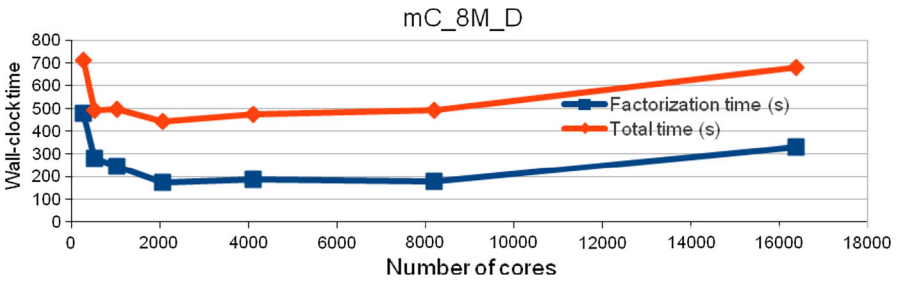


Fig. 6 The performance of *SuperLU_MCDT* for mC_8M_D

head coming from *ParMETIS* on the performance. We obtained similar results for the other matrices in Table 1. *SuperLU_MCDT* uses dense block structures called supernodes to get advantages of BLAS3 with the common technique of array padding, like *SuperLU_DIST 3.3*. Super-node detection differs as process mesh size and its square or rectangular shape. So we observe sometimes more efficient case matched to the supernode detection strategies of the algorithm like the optimal performance of mC_8M matrix for 512 cores. Figures 5 and 6 show the performance of *SuperLU_MCDT* for mC_1M_D_t and mC_8M_D, respectively.

5 Conclusions

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 16,384 cores on Curie (see [31]) for the largest matrix in our tests using *icoFoam*. Moreover, we find that the scalability becomes better as the problem size grows. We achieved scaled speed-up for large matrices having sizes up to 64 million $\times$ 64 million and up to 16,384 cores. We observed linear speed-up up to 4,096 cores on Curie (see [31]) for a 64 million $\times$ 64 million matrix. They are significant results for the problem.

Moreover, we imbedded other direct solvers (kernel class) such as *SuperLU_DIST* (see [11,12]) and *SuperLU_MCDT* (see [4,5] and references therein) in addition to default solvers provided by *OpenFOAM*. In general, we obtained reasonable perfor-

mance results with *SuperLU_MCDT*, in addition to the advantages of the direct solvers for robustness.

We define an optimal minimum number of cores under various trade-offs. 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.

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. 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 a 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 work 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 unsymmetrical matrices. Future exascale systems are expected to be having heterogeneous and many-core distributed nodes (see [23]). 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 16,384 cores for the large sparse matrices coming from the incompressible blood flow simulation without any problem, there was observable performance gain up to 2,048 cores for the sufficiently large matrices, for example, mC_4M_D, mC_5M_D, mC_6M_D and mC_8M_D matrices having seven 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 generalized geometric algebraic multi-grid and incomplete Cholesky preconditioned conjugate gradient. Also, we will test other flow simulators such as *nonNewtonianIco-*

Foam or *pisoFoam*. *nonNewtonianIcoFoam* is used to simulate non-newtonian flows while *pisoFoam* is for incompressible turbulent flow.

Acknowledgments The authors are grateful for helpful comments by the Editor-in-Chief of The Journal of Supercomputing, Prof. Hamid R. Arabnia, and two anonymous referees. This work was financially supported by the PRACE Project funded in part by the EUs 7th Framework Programme (FP7/2007–2014) under Grant agreement No. RI-312763. The work was achieved using the PRACE Research Infrastructure resource Curie at CEA, France (see [31]). Moreover, computing resources of the National Center for High Performance Computing of Turkey (UHeM) (see [32]) were used.

References

1. Behrens T (2009) OpenFOAM's basic solvers for linear systems of equations: solvers, preconditioners, smothers. Tech. Rep. DTU, Denmark. http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2008/TimBehrens/tibeh-report-fin.pdf
2. Celebi MS, Duran A, Tuncel M, Akaydin B, Oztoprak F (2013) Performance analysis of BLAS libraries in SuperLU_DIST for SuperLU_MCDT (Multi Core Distributed) development. PRACE-2IP white paper, Libraries, WP 83. http://www.prace-project.eu/IMG/pdf/wp83.pdf
3. Culpo M (2012) PRACE WP. http://www.prace-ri.eu/IMG/pdf/Current_Bottlenecks_in_the_Scalability_of_OpenFOAM_on_Massively_Parallel_Clusters-2.pdf
4. Duran A, Celebi MS, Tuncel M, Akaydın B (2012) Design and implementation of new hybrid algorithm and solver on CPU for large sparse linear systems. PRACE-2IP white paper, Libraries, WP 43. http://www.prace-ri.eu/IMG/pdf/wp43-newhybridalgorithmfo_lsls.pdf
5. Duran A, Celebi MS, Tuncel M, Oztoprak F (2013) Structural analysis of large sparse matrices for scalable direct solvers. PRACE-2IP white paper, Scalable algorithms, WP 82. http://www.prace-project.eu/IMG/pdf/wp82.pdf
6. Dagna P and Hertzer J (2013) Evaluation of multi-threaded OpenFOAM hybridization for massively parallel architectures. PRACE WP98. http://www.prace-project.eu/IMG/pdf/wp98.pdf
7. Gustafson JL (1988) Reevaluating Amdahl's law. Commun ACM 31(5):532–533
8. Hill MD, Marty MR (2008) Amdahl's law in the multicore era. IEEE Comput 41:33–38
9. Hoffmann KA, Chiang ST (2000) Computational fluid dynamics. In: Engineering education system. vol I and II, Kansas
10. Kelly S, O'Rourke M (2012) Fluid, solid and fluid-structure interaction simulations on patient-based abdominal aortic aneurysm models. Proc Inst Mech Eng Part H J Eng Med 226(4):288–304
11. Li XS, Demmel JW (2003) SuperLU_DIST: a scalable distributed-memory sparse direct solver for unsymmetric linear systems. ACM Trans Math Software 29(2):110–140
12. Li XS, Demmel JW, Gilbert JR, Grigori L, Shao M, Yamazaki I (1999) update: 2011 SuperLU Users' Guide. Report UCB, Computer Science Division, University of California, Berkeley, CA, Tech
13. Manguoglu M (2012) PRACE WP. http://www.praceproject.eu/IMG/pdf/A_General_Sparse_Sparse_Linear_System_Solver_and_Its_Application_in_OpenFOAM-2.pdf
14. Moylesa M, Nash P, Girotto I (2012) PRACE WP. http://www.prace-ri.eu/IMG/pdf/Performance_Analysis_of_Fluid-Structure_Interactions_using_OpenFOAM.pdf
15. Pal A, Anupindi K, Delorme Y, Ghaisas N, Shetty DA, Frankel SH (2014) Large eddy simulation of transitional flow in an idealized stenotic blood vessel: evaluation of subgrid scale models. J Biomech Eng 136(7):071009
16. Piskin S, Akkus A (2012) Biofuid flow applications by open-source software, 17. National Biomedical Engineering Meeting-BIYOMUT, Istanbul, Turkey
17. Piskin S, Celebi MS (2013) Analysis of the effects of different pulsatile inlet profiles on the hemodynamical properties of blood flow in patient specific carotid artery with stenosis. Comput Biol Med 43(6):717–728
18. Piskin S, Celebi MS (2012) Numerical blood flow simulation with predefined artery movement. Biomedical Engineering and Informatics (BMEI), 5th International Conference, pp 654–658. doi:10.1109/BMEI.2012.6513039
19. Piskin S, Celebi MS (2012) 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

20. Pringle GJ (2010) Porting OpenFOAM to HECToR: A dCSE Project. EPCC, The University of Edinburgh, Report
21. Turkeri H, Piskin S, Celebi MS (2011) A comparison between non-Newtonian and Newtonian blood viscosity models. J Biomech 44(Supplement):1
22. Wu WT, Aubry N, Massoudi M, Kim J, Antaki JF (2014) A numerical study of blood flow using mixture theory. Int J Eng Sci 76:56–72
23. (2013) D7.2.1 A report on the survey of HPC tools and techniques, PRACE-3IP. http://www.prace-project.eu/IMG/pdf/d7.2.1.pdf
24. OpenFOAM main site. http://www.openfoam.com
25. (Par)METIS homesite. http://www.lrz.de/services/software/mathematik/metis
26. Scotch and PT-Scotch homepage. http://www.labri.fr/perso/pelegrin/scotch
27. http://en.wikipedia.org/wiki/Electrocardiography
28. http://en.wikipedia.org/wiki/Gustafson's_law
29. http://en.wikipedia.org/wiki/QT_interval
30. http://software.intel.com/en-us/intel-mkl
31. http://www-hpc.cea.fr/en/complexe/tgcc-curie.htm
32. http://www.uybhm.itu.edu.tr
33. http://www.uybhm.itu.edu.tr/eng/inner/duyurular.html_#karadeniz

About this journal

Electronic ISSN 1573-0484

Print ISSN 0920-8542

Abstracted and indexed in

ACM Digital Library
CNKI
Current Contents/Engineering, Computing and Technology
DBLP
Dimensions
EBSCO Academic Search
EBSCO Advanced Placement Source
EBSCO Applied Science & Technology Source
EBSCO Biomedical Reference Collection
EBSCO Computer Science Index
EBSCO Computers & Applied Sciences Complete
EBSCO Discovery Service
EBSCO Engineering Source
EBSCO STM Source
EBSCO Science & Technology Collection
EI Compendex
Google Scholar
INSPEC
Institute of Scientific and Technical Information of China
Japanese Science and Technology Agency (JST)
Journal Citation Reports/Science Edition
Naver
OCLC WorldCat Discovery Service
ProQuest Advanced Technologies & Aerospace Database
ProQuest Central
ProQuest SciTech Premium Collection
ProQuest Technology Collection
ProQuest-ExLibris Primo
ProQuest-ExLibris Summon
SCImago
SCOPUS
Science Citation Index
Science Citation Index Expanded (SciSearch)
UGC-CARE List (India)
WTI Frankfurt eG

Copyright information