International Journal of Engineering, Mathematical and Physical Sciences
ISSN: 2517-9934
Vol:3, No:9, 2009

High Performance Computing Using Out-of-Core Sparse Direct Solvers

Mandhapati P. Raju and Siddhartha Khaitan

***Abstract*—**In-core memory requirement is a bottleneck in solving large three dimensional Navier-Stokes finite element problem formulations using sparse direct solvers. Out-of-core solution strategy is a viable alternative to reduce the in-core memory requirements while solving large scale problems. This study evaluates the performance of various out-of-core sequential solvers based on multifrontal or supernodal techniques in the context of finite element formulations for three dimensional problems on a Windows platform. Here three different solvers, HSL_MA78, MUMPS and PARDISO are compared. The performance of these solvers is evaluated on a 64-bit machine with 16GB RAM for finite element formulation of flow through a rectangular channel. It is observed that using out-of-core PARDISO solver, relatively large problems can be solved. The implementation of Newton and modified Newton's iteration is also discussed.

***Keywords*—**Out-of-core, PARDISO, MUMPS, Newton.

I. Introduction

THE use of sparse direct solvers in the context of finite element discretization of Navier-Stokes equations for three dimensional problems is limited by its huge memory requirement. Nevertheless, direct solvers are preferred due to their robustness. The development of sparse direct solvers based on algorithms like multifrontal [1], supernodal [2] etc. have significantly reduced the memory requirements compared to the traditional frontal solvers [3]. The superior performance of multifrontal solvers has been demonstrated for different CFD applications [4]-[7] and also in power system simulations [8]-[10]. It has been identified [4]-[7] that the memory requirement is a bottleneck in solving large three-dimensional CFD problems. There are different viable alternatives for overcoming the huge memory requirements. One alternative is to run on a 64 bit machine having large RAM. The second alternative is to use out-of-core solver, where the factors are written to the disk, thereby minimizing the in-core requirements. The third alternative is to use parallel solvers in a distributed computing environment where the memory is distributed amongst the different processors. Recent efforts by the authors show that by using a 64 bit and 16GB RAM machine, relatively larger problems can be handled in-core.

Mandhapati P. Raju is currently with the General Motors Inc., Warren, MI 48093USA (phone: 586-986-1365; e-mail: raju192@gmail.com).
Siddhartha Khaitan, is with Iowa State University, Ames, IA 50011 USA. (e-mail: skhaitan@iastate.edu).

However, as the problem size increases, the in-core memory requirement quickly exceeds 16GB. To increase the size of RAM is cost wise very expensive. On the other hand out-of-core solvers can handle very large problems with smaller in-core memory requirements. The disadvantage of using out-of-core solvers is that the computational time increases due to the I/O operations on the disk. In the recent past there has been lot of research to reduce the time for I/O and make the out-of-core solvers efficient. The capability and performance of out-of-core solvers in the context finite element Navier-Stokes code is assessed in this paper. Three state-of-the art out-of-core solvers - MUMPS, HSL_MA78 and PARDISO are evaluated. To the best of author's knowledge no such comparison of the performance of out-of-core has been reported in the literature.

MUMPS [11]-[13] is a parallel direct solver with out-of-core functionality and is available in the public domain. PARDISO [2], [14]-[17] also has an out-of-core solver and it is available as a part of the INTEL Math Kernel Library [18]. HSL_MA78 [19], an out-of-core solver, is available as part of HSL 2007, which is available free for any UK researchers. An evaluation version of HSL_MA78 is used in this paper.

In finite element Navier-Stokes formulations, the set of linear equations generated usually generate a matrix that zero diagonal entries. Penalty formulation yields non-zero diagonal entries but it is observed that the diagonal entries are few orders of magnitudes smaller than the other non diagonal entries. The iterative solution methods fail or pose severe convergence problems for such ill conditioned matrices. Although the iterative solvers are memory efficient, the resolution of convergence issues is not straightforward and results in lack of robustness. The performance of a suite of iterative solvers is compared with the out-of-core direct solvers to demonstrate the superiority of direct solvers.

II. Mathematical Formulation

A benchmark rectangular channel flow problem is chosen for evaluating the out-of-core solvers. The governing equations for laminar flow inside a rectangular channel are presented below in the non-dimensional form. In three-dimensional calculations, instead of the primitive formulation, penalty approach is used to reduce the memory requirements. The equations are all presented in the non-dimensional form.

$$\frac{\partial \hat{u}}{\partial \hat{x}}+\frac{\partial \hat{v}}{\partial \hat{y}}+\frac{\partial \hat{w}}{\partial \hat{z}}=0, \tag{1}$$

$$\frac{\partial}{\partial \hat{x}}\left(\hat{u}^2\right)+\frac{\partial}{\partial \hat{y}}\left(\hat{u}\hat{v}\right)+\frac{\partial}{\partial \hat{z}}\left(\hat{u}\hat{w}\right)=\lambda\frac{\partial}{\partial \hat{x}}\left(\frac{\partial \hat{u}}{\partial \hat{x}}+\frac{\partial \hat{v}}{\partial \hat{y}}+\frac{\partial \hat{w}}{\partial \hat{z}}\right)+\frac{\partial}{\partial \hat{x}}\left(\frac{2}{\mathrm{Re}}\frac{\partial \hat{u}}{\partial \hat{x}}\right) \tag{2}$$
$$+\frac{\partial}{\partial \hat{y}}\left(\frac{1}{\mathrm{Re}}\left(\frac{\partial \hat{u}}{\partial \hat{y}}+\frac{\partial \hat{v}}{\partial \hat{x}}\right)\right)+\frac{\partial}{\partial \hat{z}}\left(\frac{1}{\mathrm{Re}}\left(\frac{\partial \hat{u}}{\partial \hat{z}}+\frac{\partial \hat{w}}{\partial \hat{x}}\right)\right),$$

$$\frac{\partial}{\partial \hat{x}}\left(\hat{u}\hat{v}\right)+\frac{\partial}{\partial \hat{y}}\left(\hat{v}^2\right)+\frac{\partial}{\partial \hat{z}}\left(\hat{v}\hat{w}\right)=\lambda\frac{\partial}{\partial \hat{y}}\left(\frac{\partial \hat{u}}{\partial \hat{x}}+\frac{\partial \hat{v}}{\partial \hat{y}}+\frac{\partial \hat{w}}{\partial \hat{z}}\right)+\frac{\partial}{\partial \hat{x}}\left(\frac{1}{\mathrm{Re}}\left(\frac{\partial \hat{u}}{\partial \hat{y}}+\frac{\partial \hat{v}}{\partial \hat{x}}\right)\right) \tag{3}$$
$$+\frac{\partial}{\partial \hat{y}}\left(\frac{2}{\mathrm{Re}}\frac{\partial \hat{v}}{\partial \hat{y}}\right)+\frac{\partial}{\partial \hat{z}}\left(\frac{1}{\mathrm{Re}}\left(\frac{\partial \hat{v}}{\partial \hat{z}}+\frac{\partial \hat{w}}{\partial \hat{y}}\right)\right),$$

and

$$\frac{\partial}{\partial \hat{x}}\left(\hat{u}\hat{w}\right)+\frac{\partial}{\partial \hat{y}}\left(\hat{v}\hat{w}\right)+\frac{\partial}{\partial \hat{z}}\left(\hat{w}^2\right)=\lambda\frac{\partial}{\partial \hat{z}}\left(\frac{\partial \hat{u}}{\partial \hat{x}}+\frac{\partial \hat{v}}{\partial \hat{y}}+\frac{\partial \hat{w}}{\partial \hat{z}}\right)+\frac{\partial}{\partial \hat{x}}\left(\frac{1}{\mathrm{Re}}\left(\frac{\partial \hat{u}}{\partial \hat{z}}+\frac{\partial \hat{w}}{\partial \hat{x}}\right)\right) \tag{4}$$
$$+\frac{\partial}{\partial \hat{y}}\left(\frac{1}{\mathrm{Re}}\left(\frac{\partial \hat{v}}{\partial \hat{z}}+\frac{\partial \hat{w}}{\partial \hat{y}}\right)\right)+\frac{\partial}{\partial \hat{z}}\left(\frac{2}{\mathrm{Re}}\frac{\partial \hat{w}}{\partial \hat{z}}\right).$$

where $\hat{u},\hat{v},\hat{w}$ are the components of velocity,. The bulk flow Reynolds number, Re, and λ is the penalty parameter. Velocities are non-dimensionalized with respect to inlet velocity and the coordinates are non-dimensionalized with respect to channel length.

The boundary conditions are prescribed as follows:

(1) Along the channel inlet:

$$\hat{u}=1;\ \hat{v}=0;\hat{w}=0. \tag{5}$$

(2) Along the channel exit :

$$\frac{\partial \hat{u}}{\partial \hat{x}}=0;\ \frac{\partial \hat{v}}{\partial \hat{x}}=0;\ \frac{\partial \hat{w}}{\partial \hat{x}}=0. \tag{6}$$

(3) Along the walls:

$$\hat{u}=0;\ \ \hat{v}=0;\ \ \hat{w}=0. \tag{7}$$

The flow Reynolds number is taken as 50 to simulate laminar flow inside the channel.

III. Numerical Formulation

Galerkin finite element method (GFEM) is used for the discretization of the above penalty based Navier Stokes equations. Three dimensional brick elements are used; the velocity components are interpolated bilinearly. The nonlinear system of equations obtained from GFEM is solved by Newton's method. Let be the available vector of field unknowns for the *i*th iteration. Then the update for the iteration is obtained as

$$\underset{\sim}{X}^{(i+1)}=\underset{\sim}{X}^{(i)}+\alpha\,\delta\underset{\sim}{X}^{(i)}, \tag{8}$$

where α is an under-relaxation factor, and $\delta\underset{\sim}{X}^{(i)}$ is the correction vector obtained by solving the linearized system

$$[J]^{(i)}\left\{\delta\underset{\sim}{X}^{(i)}\right\}=-\left\{\underset{\sim}{R}_X\right\}^{(i)}. \tag{9}$$

Here, $[J]^{(i)}$ is the Jacobian matrix at the $(i+1)^{st}$ iteration,

$$[J]^{(i)}=\frac{\partial \underset{\sim}{R}_X^{(i)}}{\partial \underset{\sim}{X}^{(i)}}. \tag{10}$$

$\{\underset{\sim}{R}_X\}^{(i)}$ is the residual vector. Newton's iteration is continued till the infinity norm of the correction vector $\delta\underset{\sim}{X}^{(i)}$ converges to a prescribed tolerance of 10^{-10}. A modified Newton's method is also used in this study. For modified Newton eq. (9) is modified as shown in eq. 11

$$[J]^{(0)}\left\{\delta\underset{\sim}{X}^{(i)}\right\}=-\left\{\underset{\sim}{R}_X\right\}^{(i)}. \tag{11}$$

In modified Newton's method the Jacobian is evaluated only during the first iteration. Consequently the Jacobian is factorized only once. For all subsequent iterations, the same Jacobian (and hence its *LU* factors) is used repeatedly. This algorithm is referred as modified Newton. Since factorization is the most expensive part of the computations, by using modified Newton's algorithm, the expensive factorization step can be skipped after the first iteration.

We can see that the discretizations of the governing partial differential equations from (7)-(10) by the GFEM scheme results in a set of nonlinear equations. However the core of the resulting nonlinear equations is the solution of a sparse linear systems (eq. 9), which is the most computationally intensive part of the solver both in terms of CPU time and memory requirement. Here three different out-of-core solvers, MUMPS, HSL_MA78 and PARDISO are implemented and compared.

To gain maximum computational efficiency the codes are optimized at three levels.

(a) The first is at the hardware level by using an optimized Intel MKL BLAS library. This is highly optimized for Intel processors.

(b) The second level is the choice of an efficient state-of-the-art out-of-core solver. Three different out-of-core solvers are evaluated for their performance. The efficiency of an out-of-core solver not only depends on the factorization algorithms of the solvers but also on the handling of different I/O operations. For an out-of-core solver, the I/O operations can be a bottleneck depending on how the I/O operations are performed.HSL_MA78 handles efficiently using virtual memory management package HSL_OF01 which facilitates reading and writing from direct-access files. Real and integer data have their own buffers associated with it. Each buffer can be associated with more than one direct-access file.

(c) The third level is the choice of an efficient algorithm for solving the system of non-linear equations. The choice of the non-linear algorithm can affect the rate of convergence and hence the computational time. The system of non-linear equations is either solved using Newton or Picard iteration. Newton iteration is quite popular and efficient due to its quadratic convergence behavior. If the initial guess is chosen properly, then Newton iteration can give convergence in a few iterations. However the limitation is that the formation of Jacobian matrices involving derivatives is not always straightforward to compute. In addition the choice of the initial guess will affect the convergence behavior. Picard

iteration however is more robust in terms of its flexibility for choosing an initial guess. However, the rate of convergence will be linear and hence would result in more computational time.

The choice of modified Newton or modified Picard can further significantly reduce the computational time. In the modified Newton method, the left hand side matrix is factorized only once and the factors are reused. Since factorization is the bottleneck, by avoiding the factorization in the subsequent steps can reduce the computational time. The rate of convergence will no longer be quadratic but linear. Nevertheless, there will be significant savings in computational time. This paper discussed only Newton and modified Newton implementation.

IV. Results and Discussion

In this paper, flow inside a three dimensional rectangular channel is considered. Three dimensional finite element brick elements are used for generating the grid. Weak Galerkin finite element formulation is used to discretize the Navier-Stokes equations to form a large set of non-linear equations. Newton's iteration is used to generate a set of linear algebraic equations. The matrices generated from such discretization are usually very sparse and hence a good sparse solver is used to reduce the computational efforts. It is to be noted that for three dimensional grids, the matrices generated are less sparse compared to the matrices generated from two-dimensional grid.

Typically an interior node in a three-dimensional grid is connected to 27 nodes including it. Since there are 3 dof's at each node, a typical row consists of 81 non-zero entries. In a two-dimensional grid, a typical row consists of 27 non-zero entries. This would increase the frontal size considerably. Hence solving three-dimensional problems using direct solvers is quite challenging both in terms of computational time and memory requirements. Large problems cannot be solved on a 32-bit machine using in-core techniques [4], [7]. This paper studies the performance of out-of-core direct solvers on a 64 bit machine with 16GB RAM. All the computations are run a windows machine with Intel Xeon processor.

Before comparing the various solvers for their relative performances, each individual solver is tuned for its optimal performance, specifically the choice of the ordering package. Each solver has inbuilt ordering packages, whose choice can affect the performance of the solver. In addition there are other parameters like pivot tolerance etc which will affect the performance of the solver.

MUMPS solver

The sequential version of out-of-core MUMPS solver is built on a 64 bit machine. The choice of the out-of-core solver can be invoked by setting the value of mumps_par%ICNTL(22) as 1. MUMPS has different inbuilt ordering packages (AMD [20], QAMD [21], AMF, PORD [22]). In addition, there is a provision to link METIS [23] as an external package. Memory relaxation is taken as 100%. MUMPS out-of-core solver is used for all the cases. Table 1 shows the comparison of the performance of the various ordering methods. All the cases are run for 30x30x30 mesh. The CPU time and memory for each of the solver are compared. The CPU time reported is the CPU time for the first Newton iteration. The CPU time and memory requirement for the complete in-core solution is also included in the brackets for a quick comparison. It is to be first noted that the out-of-core solution is around 3-5 times slower compared to the in-core solution. Of all the ordering packages, METIS gives best results. Compared to AMD, METIS results in almost one-third of the floating point operations. The computational time and memory requirements are the lowest for the METIS ordering. Nested bisection algorithm of METIS is found to generate good ordering for three dimensional meshes. Based on this result, METIS ordering is used for all subsequent runs using MUMPS solver.

Table 1: Comparison of ordering methods for the MUMPS solver

ordering	#dof's	Cpu time (sec)	Memory (GB)	
			in-core arrays	out-of-core files
AMD	89373	438.4 (142.8)	1.4 (4.06)	2
QAMD	89373	446.6 (142.75)	1.4 (4.04)	2
AMF	89373	352 (105.7)	1.34 (3.48)	1.67
PORD	89373	309 (86.6)	1.09 (3.18)	1.5
METIS	89373	250.3 (55.01)	0.78 (3.02)	1.28

HSL_MA78 solver

The HSL MA78solver does not any internal ordering techniques but however the HSL solver package has other routines which do the function of ordering the finite element entries to reduce the fill-in during factorization. HSL MC68 is generally used for efficient ordering of finite element matrices. In addition, external ordering packages can be hooked to the HSL MA78 solver. In this paper METIS ordering is also used by hooking the METIS library to the solver. Table 2 shows the performance of HSL MC68 and METIS ordering on the CPU time and memory of the HSL MA78 solver. It is found that METIS performs better than HSL MC68. Hence METIS is used for all subsequent runs for the HSL MA78 solver.

Table II: Comparison of ordering methods for the HSL-MA78 for 30x30x30 grid

ordering	#dof's	Cpu time (sec)	Memory (GB)	
			in-core arrays	out-of-core files
HSL_MC68	89373	536 (524)	1.4 (6.88)	3.5
METIS	89373	321 (318)	0.79 (4.72)	2

PARDISO solver

PARDISO has minimum dissection (MD) and METIS

ordering hooked internally within the solver. The user has the choice to use either of the ordering techniques. Table 3 compares the effect of MD and METIS ordering on the performance of PARDISO solver. It is found that METIS performs better than the MD ordering technique. Hence METIS is used for all subsequent runs for the PARDISO solver.

TABLE III: COMPARISON OF ORDERING METHODS FOR THE PARDISO SOLVER FOR 30X30X30 GRID

ORDERING	#DOF'S	CPU TIME (SEC)	MEMORY (GB)	
			IN-CORE ARRAYS	OUT-OF-CORE FILES
MD	89373	284 (162)	0.5 (2.71)	2.8
METIS	89373	97 (59)	0.3 (1.42)	1.25

Table 4 and Table 5 show the time split between different phases of the solver for in-core and out-of-core solution. Interestingly, the performance of in-core and out-of-core for HSL_MA78 solver is almost the same in terms of computational time. This may be because of the efficient I/O operation used in HSL_MA78 package. It used virtual memory management using HSL_F01 packages to handle efficient I/O operations. This strategy is found to be very effective in developing good out-of-core solvers. Although the out-of-core HSL is performing well in comparison to the in-core, the overall computation time is much larger compared to the other solvers. MUMPS out-of-core solver is almost 4 times slower than in-core solver. PARDISO out-of-core solver is around 1.6 times slower than its in-core solver. Another interesting observation is that out-of-core PARDISO has a relatively large solve phase compared to out-of-core MUMPS. PARDISO has much less in-core memory requirement compared to MUMPS or HSL.

TABLE IV: COMPARISON OF TIME SPLIT FOR IN-CORE SOLUTION OF 30X30X30 GRID

In-core solvers	Computational time (Seconds)					Memory (GB)
	Matrix assembly	Analysis phase	Numeric phase	Solve phase	Total time	
MUMPS	4	1.51	53.3	0.58	59.39	3.02
PARDISO	4	2.19	52.6	0.47	59.26	1.42
HSL_MA78	4	0.64	313.14		317.78	4.72

TABLE 5: COMPARISON OF TIME SPLIT FOR OUT-OF-CORE SOLUTION OF 30X30X30 GRID

Out-core solvers	Computational time (Seconds)					Memory (GB)	
	Matrix assembly	Analysis phase	Numeric phase	Solve phase	Total time	Incore files	Out-of-core files
MUMPS	4	1.6	243.2	1.45	250.25	0.78	1.28
PARDISO	4	2.2	87.76	3.07	97.03	0.3	1.25
HSL_MA78	4	0.64	314		318.64	0.79	3.174

Tables 6-8 shows the performance of out-of-core MUMPS, HSL_MA78 and PARDISO solvers for different grid sizes. The performance of in-core solution is also presented for relative comparison. Table 4 shows that out-of-core MUMPS solver is always greater than 4 times slower compared to the in-core solver. This shows that the out-of-core implementation of MUMPS solver is less efficient. The in-core memory requirement is maintained low. Surprisingly the out-of-core HSL_MA78 solver is very efficient with respect to the in-core solver. The computational times for both in-core and out-of-core implementations are almost similar. The in-core memory for the out-of-core solver is maintained low. The out-of-core memory requirement is larger for HSL_MA78 solver compared to the other two solvers.

TABLE VI: PERFORMANCE OF MUMPS SOLVER ON DIFFERENT GRID SIZES

				MUMPS in-core		MUMPS out-of-core		
nex	ney	nez	#dof's	cpu time (min)	Memory (GB)	cpu time (min)	Memory (GB) incore	Out-of-core
50	10	10	18513	0.047	0.23	0.31	0.075	0.11
100	10	10	36663	0.089	0.52	0.63	0.12	0.23
200	10	10	72963	0.177	1.1	1.25	0.2	0.465
50	20	10	35343	0.14	0.65	0.82	0.17	0.285
100	20	10	69993	0.278	1.4	1.74	0.27	0.612
100	20	20	133623	1.127	3.87	5.31	0.72	1.72
100	50	20	324513	6.4	13.42	21.39	2.54	6.13
100	50	50	788103	*	*	126.1	10.2	24.1
50	20	20	67473	0.45	1.81	2.36	0.47	0.78
50	50	10	85833	0.495	2.11	2.72	0.45	0.925
50	50	20	163863	2.228	5.93	8.67	1.3	2.64
50	50	50	397953	*	*	42.5	5	10.12

TABLE VII: PERFORMANCE OF HSL_MA78 SOLVER ON DIFFERENT GRID SIZES

				HSL in-core		HSL out-of-core		
nex	ney	nez	#dof's	cpu time (min)	Memory (GB)	cpu time (min)	Memory (GB) In-core	Out-of-core
50	10	10	18513	0.18	0.24	0.193	0.14	0.32
100	10	10	36663	0.35	0.5	0.363	0.14	0.625
200	10	10	72963	0.59	0.79	0.612	0.14	1.15
50	20	10	35343	0.78	0.65	0.8	0.23	0.84
100	20	10	69993	1.06	1.53	1.074	0.23	1.475
100	20	20	133623	4.83	3.91	3.936	0.5	3.54
100	50	20	324513	28.9	13.94	23.52	1.56	12.67
100	50	50	788103	*	*	542.6	8.1	49.95
50	20	20	67473	2.42	1.98	2.46	0.5	1.93
50	50	10	85833	2.65	2.38	2.69	0.5	2.37
50	50	20	163863	11.9	7.12	11.3	1.56	6.45
50	50	50	397953	*	*	358	5.8	24

Table 8 shows the performance of PARDISO solver. The out-of-core solver is around twice slower compared to the in-core solver. Overall PARDISO out-of-core solver is must faster compared to the other two solvers. The in-core memory requirement is kept very low. For a 100x50x50 mesh, out-of-core MUMPS requires around 10 GB of in-core memory and out-of-core HSL_MA78 requires around 8 GB of in-core memory and out-of-core PARDISO requires around 4 GB of in-core memory. Hence PARDISO can solve much finer grid sizes compared to the other two solvers. The finest grid sizes chosen to solve with PARDISO in this paper are 150x75x30 and 200x80x40, which consists of around 1 million and 2 million degrees of freedom. The 150x75x30 grid requires around 5.5 GB of in-core memory. The out-of-core memory is around 31 GB. It takes around 172 seconds for one Newton iteration. The 200x80x40 grid requires around 10.5 GB of in-core memory and around 75 GB out-of-core memory. One Newton iteration takes around 16.5 hours of CPU time. Thus we observe that out-of-core PARDISO can solve very large three dimensional problems in the context of using direct solvers on a single desktop. Both in terms of computational time and memory requirement, PARDISO is found to the best solver.

TABLE VIII: PERFORMANCE OF PARDISO SOLVER ON DIFFERENT GRID SIZES

				PARDISO in-core		PARDISO out-of-core		
nex	ney	nez	#dof's	cpu time (min)	Memory (GB)	cpu time (min)	Memory (GB) incore	out-of-core
50	10	10	18513	0.038	0.08	0.067	0.087	0.1
100	10	10	36663	0.073	0.25	0.15	0.17	0.22
200	10	10	72963	0.157	0.57	0.304	0.34	0.47
50	20	10	35343	0.105	0.29	0.209	0.17	0.27
100	20	10	69993	0.243	0.69	0.515	0.337	0.593
100	20	20	133623	1.073	1.92	1.758	0.665	1.693
100	50	20	324513	6.517	6.62	9.5	1.65	6.035
100	50	50	788103	*	*	114.2	4.1	24.5
50	20	20	67473	0.432	0.85	0.731	0.33	0.763
50	50	10	85833	0.457	1.02	0.84	0.42	0.895
50	50	20	163863	2.233	2.86	3.4	0.83	2.6
50	50	50	397953	17.650	10.67	25.4	2.03	10.15
150	75	30	1067268	*	*	171.95	5.52	31
200	80	40	2002563	*	*	993	10.5	74.8

Correlations are generated for the CPU times and memory requirements of all the solvers with respect to the grid size. Equation 12-15 shows the correlations for the MUMPS solver. In these equations T refers to the CPU time in minutes taken by the solver for one Newton iteration. It includes the time for generation of the matrix, the analysis phase, factorization phase and the solve phase. M refers to the memory requirement in GigaBytes, the subscripts in-core and out-of-core refers to the in-core and out-of-core memory requirements for the out-of-core solver, *nex, ney and nez* refer to the grid elements in the *x,y* and *z* directions respectively, n refers to the total number of degrees of freedom, *ar* and *ar* refers to the grid aspect ratio's *nex/ney* and *nex/nez*.

MUMPS

$$T = 3.56\times10^{-7} n^{1.447} ar_1^{-0.127} ar_2^{-0.127};\quad R^2 = 0.95 \tag{12}$$

$$M_{incore} = 8.28\times10^{-7} n^{1.214} ar_1^{-0.197} ar_2^{-0.197};\quad R^2 = 0.98 \tag{13}$$

$$M_{outofcore} = 2.11\times10^{-7} n^{1.377} ar_1^{-0.127} ar_2^{-0.127};\quad R^2 = 0.98 \tag{14}$$

HSL

$$T = 3.53\times10^{-8} n^{1.707} ar_1^{-0.362} ar_2^{-0.362};\quad R^2 = 0.7 \tag{15}$$

$$M_{incore} = 2.6\times10^{-4} n^{0.736} ar_1^{-0.33} ar_2^{-0.33};\quad R^2 = 0.98 \tag{16}$$

$$M_{outofcore} = 3.42\times10^{-6} n^{1.219} ar_1^{-0.155} ar_2^{-0.155};\quad R^2 = 0.99 \tag{17}$$

PARDISO

$$T = 4.07\times10^{-9} n^{1.757} ar_1^{-0.174} ar_2^{-0.174};\quad R^2 = 0.85 \tag{18}$$

$$M_{incore} = 3.84\times10^{-6} n^{1.02} ar_1^{-0.01} ar_2^{-0.01};\quad R^2 = 0.99 \tag{19}$$

$$M_{outofcore} = 1.39\times10^{-7} n^{1.407} ar_1^{-0.112} ar_2^{-0.112};\quad R^2 = 0.99 \tag{20}$$

The correlations will give an idea of how the solver requirements vary as the grid size is modified. The exponents of n is greater than 1 indicating that as the number of degrees of freedom increases, the CPU time and memory requirement are going to increase superlinearly. For the out-of-core solvers, the exponents of n are similar for all the three solvers, with MUMPS being the lower of around 1. 45. The CPU time (and memory requirement are not only a function of number of degrees of freedom but also a function of the grid aspect ratio's. The absolute values of the exponents of the aspect ratio's is larger for HSL solver compared to MUMPS and PARDISO. This indicates that the solver performance is also a strong function of the grid distribution also. The memory requirements of the out-of-core solver consists of in-core memory (for holding the frontal matrices and other working arrays) requirement and out-of-core memory (consists of *LU* factors written to the disk) requirements. Correlations are presented for both the memory requirements. An interesting observation is that the in-core memory requirement for PARDISO the least amongst the three solvers and that the memory requirement varies linearly with the number of degrees of freedom and is almost independent of the grid distribution. This behavior of out-of-core PARDISO is very conducive for choosing it for solving large three dimensional finite element problems.

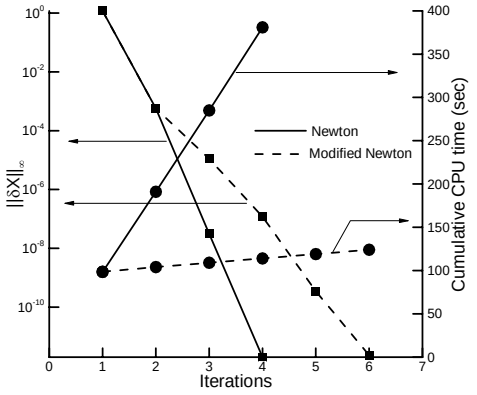

Fig. 1 Comparison of CPU time and residual norm for Newton and modified Newton's method for 30x30x30 grid

Figure 1 shows the performance of Newton and modified Newton algorithms using out-of-core PARDISO as the linear solver for a 30x30x30 grid. No under-relaxation is used for both Newton and modified Newton's method. Newton's method converges in 4 iterations and quadratic convergence is observed. Modified Newton's method converges in 6 iterations. Linear convergence is observed. It is observed that significant computational time savings can be achieved using modified Newton's method. Newton's iterations converge in 381 seconds, whereas modified Newton's iterations converge in 128 seconds.

TABLE IX: COMPARISON OF CPU TIMES FOR NEWTON AND MODIFIED NEWTON FOR DIFFERENT GRIDS

PARDISO out-of-core				Newton	Modified Newton
nex	ney	nez	#dof's	cpu time (min)	cpu time (min)
50	10	10	18513	0.34	0.152
100	10	10	36663	0.8	0.327
200	10	10	72963	1.78	0.77
50	20	10	35343	0.99	0.405
100	20	10	69993	2.35	0.937
100	20	20	133623	7.65	2.53
100	50	20	324513	39.51	11.83
100	50	50	788103	468.1	122.5
50	20	20	67473	3.27	1.09
50	50	10	85833	3.57	1.247
50	50	20	163863	14.4	4.37
50	50	50	397953	106.4	27.98
150	75	30	1067268	693	181.8

Table 9 shows the comparison of CPU times for Newton and modified Newton methods for different grid sizes. It is clearly observed that the implementation of modified Newton methods leads to significant savings in computational time. Further it is observed that the number of degrees of freedom increases, the percentage of computational savings using modified Newton's method as compared to the Newton's method increases.

V. CONCLUSIONS

Three different out-of-core solvers (MUMPS, HSL_MA78, PARDISO) are evaluated for the solution of finite element Navier-Stokes formulation of laminar flow in a rectangular channel. METIS is found to be the best choice of ordering algorithm for reducing the fill in of LU factors. Of the three solvers, PARDISO is found to the best solver with lower computational time and lower in-core and out-of-core memory requirements.

It is observed that out-of-core HSL_MA78 is found to perform almost identically with that of the in-core HSL_MA78 solver. HSL_F01 facilitates the efficient I/O operations for the HSL_MA78 solver. However the out-of-core HSL_MA78 is much slower than MUMPS and PARDISO out-of-core solvers. Out-of-core strategy can help in solving large three dimensional finite element problems. Out-of-core PARDISO could solve around 2 million equations resulting from three dimensional finite element formulations on a single desktop. Further it is observed that the use of modified Newton's algorithm can significantly reduce the computational time as compared to the Newton's algorithm.

REFERENCES

[1] T. A. Davis and I.S. Duff, "A combined unifrontal/multifrontal method for unsymmetric sparse matrices," *ACM Trans. Math. Soft.*, vol. 25, no. 1, 1997, pp. 1–19.
[2] O. Schenk, K. Gartner, and W. Fichtner, "Efficient Sparse LU Factorization with Left-right Looking Strategy on Shared Memory Multiprocessors," BIT, vol. 40, no. 1, 2000, pp. 158–176.
[3] B. M. Irons, "A frontal solution scheme for finite element analysis," *Numer. Meth. Engg.*, vol. 2, 1970, pp. 5-32.
[4] M. P. Raju, and J. S. T'ien, "Development of Direct Multifrontal Solvers for Combustion Problems," *Numerical Heat Transfer-Part B*, vol. 53, 2008, pp. 1-17.
[5] M. P. Raju, and J. S. T'ien, "Modelling of Candle Wick Burning with a Self-trimmed Wick," Comb. Theory Modell., vol. 12, no. 2, 2008, pp. 367-388.
[6] M. P. Raju, and J. S. T'ien, "Two-phase flow inside an externally heated axisymmetric porous wick," vol. 11, no. 8, 2008, pp. 701-718.
[7] P. K. Gupta, and K. V. Pagalthivarthi, "Application of Multifrontal and GMRES Solvers for Multisize Particulate Flow in Rotating Channels," *Prog. Comput Fluid Dynam.*, vol. 7, 2007, pp. 323–336.
[8] S. Khaitan, J. McCalley, Q. Chen, "Multifrontal solver for online power system time-domain simulation," *IEEE Transactions on Power Systems*, vol. 23, no. 4, 2008, pp. 1727–1737.
[9] S. Khaitan, C. Fu, J. D. McCalley, "Fast parallelized algorithms for online extended-term dynamic cascading analysis," *PSCE*, 2009, pp. 1–7.
[10] J. McCalley, S. Khaitan, "Risk of Cascading outages", Final Report, PSrec Report, S-26, August 2007. Available at http://www.pserc.org/docsa/Executive_Summary_Dobson_McCalley_Cascading_Outage_ S-2626_PSERC_ Final_Report.pdf
[11] P. R. Amestoy, and I. S. Duff, "Vectorization of a multiprocessor multifrontal code," *International Journal of Supercomputer Applications*, vol. 3, 1989, pp. 41–59.
[12] P. R. Amestoy, I. S. Duff, J. Koster and J. Y. L'Excellent, "A fully asynchronous multifrontal solver using distributed dynamic scheduling," *SIAM Journal on Matrix Analysis and Applications,* vol. 23, no. 1, 2001, pp. 15–41.

[13] P. R. Amestoy, I. S. Duff, and J. Y. L'Excellent, "Multifrontal parallel distributed symmetric and unsymmetric solvers," *Comput. Methods Appl. Mech. Eng.*, vol. 184, 2000, pp. 501–520.
[14] O. Schenk, "Scalable Parallel Sparse LU Factorization Methods on Shared Memory Multiprocessors," Ph.D. dissertation, ETH Zurich, 2000.
[15] O. Schenk, and K. Gartner, "Sparse Factorization with Two-Level Scheduling in PARDISO," in *Proc. 10th SIAM conf. Parallel Processing for Scientific Computing*, Portsmouth, Virginia, March 12-14, 2001.
[16] O. Schenk, and K. Gartner, "Two-level scheduling in PARDISO: Improved Scalability on Shared Memory Multiprocessing Systems," *Parallel Computing,* vol. 28, 2002, pp. 187-197.
[17] O. Schenk, and K. Gartner, "Solving Unsymmetric Sparse Systems of Linear Equations with PARDISO," *Journal Future Generation Computer Systems*, vol. 20, no. 3, 2004, pp. 475-487.
[18] *Intel MKL Reference Manual*, Intel® Math Kernel Library (MKL), 2007. Available: http://www.intel.com/software/products/mkl/
[19] J. A. Scott, *Numerical Analysis Group Progress Report,* RAL-TR-2008-001, Rutherford Appleton Laboratory, 2008.
[20] P. R. Amestoy, T. A. Davis, and I. S. Duff, "An approximate minimum degree ordering algorithm," *SIAM Journal on Matrix Analysis and Applications,* vol. 17, 1996, pp. 886–905.
[21] P. R. Amestoy, "Recent progress in parallel multifrontal solvers for unsymmetric sparse matrices," in *Proc. 15th World Congress on Scientific Computation, Modelling and Applied Mathematics, IMACS,* Berlin, 1997.
[22] J. Schulze, "Towards a tighter coupling of bottom-up and top-down sparse matrix ordering methods," *BIT*, vol. 41, no. 4, 2001, pp. 800–841.
[23] G. Karypis, and V. Kumar, "METIS – A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices – Version 4.0," University of Minnesota, September 1998.