Sparse Matrix Factorization in the Implicit Finite Element Method on Petascale Architecture

Seid Koric [1, 2] and Anshul Gupta [3]

[1]National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, USA
[2]Department of Mechanical Science and Engineering, University of Illinois at Urbana-Champaign, USA
[3]Mathematical Sciences, IBM T.J. Watson Research Center, USA

Corresponding author:
Seid Koric, National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign
1205 W. Clark St., Urbana, IL, 61801, USA
Email: koric@illinois.edu

Abstract

The performance of the massively parallel direct multifrontal solver Watson Sparse Matrix Package (WSMP) for solving large sparse systems of linear equations arising in implicit finite element method on unstructured (free) meshes in solid mechanics was evaluated on one of the most powerful supercomputers currently available to the open science community-the sustained petascale high performance computing system of Blue Waters. We have performed full-scale benchmarking tests up to 65,536 cores using assembled global stiffness matrices and load vectors ranging from 11-40 million unknowns extracted from "real-world" commercial implicit finite element analysis (FEA) applications. The results show that a direct multifrontal factorization method with a hybrid parallel implementation in WSMP performs exceedingly well on a petascale high-performance computing (HPC) system, and delivers superior factorization time and parallel scalability, thus opening the door for the high fidelity modeling of complex industrial structures and assemblies in real scale.

Keywords Sparse linear solvers, factorization, petascale high performance computing, finite element method, unstructured mesh

1. Introduction

1.1 High Performance Computing in Engineering

Across a range of engineering fields, the use of simulation and computational models is pervasive for designing engineered systems. High Performance Computing (HPC) systems play an essential role in simulations and modeling. Researchers and manufacturing teams depend on HPC to create safe cars and energy-efficient aircraft as well as effective communication systems and efficient supply chain models. Availability of advanced HPC technologies has also fundamentally altered the investigative paradigm in the field of biomechanics. While emerging peta-scale computing is already a strategic enabler of large-scale simulations in many scientific areas such as astronomy, biology and chemistry [1-3], paradoxically for many engineers and researchers, the existing hardware and software often cannot be used to solve their problems. On one hand, current HPC systems in production often lack the computational power, network bandwidth and data storage needed for solving tomorrow's real-world engineering challenges. On the other hand, even the most powerful hardware will fail to deliver on its full potential unless matched with appropriate algorithms designed specifically for such environments. Sparse matrix factorization, a critical algorithm in many science, engineering, and optimization applications, has traditionally had difficulty tuning to and leveraging the ever increasing computational power of HPC [4].

The main objective of this work is to demonstrate that the multifrontal sparse factorization algorithm with hybrid parallelization, such as the one in the WSMP solver code, can scale efficiently in today's large-scale supercomputers, opening a new horizon of high fidelity and robust finite element simulations in the engineering academic and industrial realms.

1.2 Sparse Linear Solvers in Implicit Finite Element Methods, Background and Previous Work

Solving linear system of equations:

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{1}$$

is responsible for 70-80% of the total computational time in many problems in computational science and engineering. When discretizing continuous solid mechanics problems with implicit finite element method, the associated matrix $\boldsymbol{A}$ is sparse, symmetric and often positive definite. A single solution of equation (1) suffices for linear problems. For nonlinear problems, however, within each quasi-static time step, a system of nonlinear equations is linearized and solved with a Newton-Raphson (NR) iteration scheme [5,6], which requires several linear solver solutions of global equilibrium iterations (subscript i) as follows:

$$\left[\mathrm{K}_{\mathrm{i-1}}^{\mathrm{t+\Delta t}}\right]\left\{\Delta \mathrm{u}_{\mathrm{i-1}}^{\mathrm{t+\Delta t}}\right\}=\left\{\mathrm{R}_{\mathrm{i-1}}^{\mathrm{t+\Delta t}}\right\}. \tag{2}$$

Here $\left\{\Delta \mathrm{u}_{\mathrm{i-1}}^{\mathrm{t+\Delta t}}\right\}$ is the incremental change to the solution vector (displacements in mechanical problems), and $\left\{\mathrm{R}_{\mathrm{i-1}}^{\mathrm{t+\Delta t}}\right\}$ is the residual error vector. A linear solver is used to solve equation (2) for $\left\{\Delta \mathrm{u}_{\mathrm{i-1}}^{\mathrm{t+\Delta t}}\right\}$, which is used to update the solution vector in equation (3), until convergence is achieved everywhere at time $\mathrm{t+\Delta t}$ (i.e., when the update vector is sufficiently small).

$$\left\{\mathrm{u}_{\mathrm{i}}^{\mathrm{t+\Delta t}}\right\}=\left\{\mathrm{u}_{\mathrm{i-1}}^{\mathrm{t+\Delta t}}\right\}+\left\{\Delta \mathrm{u}_{\mathrm{i-1}}^{\mathrm{t+\Delta t}}\right\} \tag{3}$$

The tangent stiffness matrix $\left[\mathrm{K}^{\mathrm{t+\Delta t}}\right]$ is defined in equation (5) from the consistent tangent operator, also known as the material Jacobian, [J], which is defined in equation (4) for mechanical problems, taking $\Delta\hat{\boldsymbol{\varepsilon}}^{\mathrm{t+\Delta t}}$ as a guessed mechanical strain increment, based on the current best displacement increment.

$$\underline{\underline{\mathbf{J}}} = \frac{\partial \Delta \boldsymbol{\sigma}^{\mathrm{t+\Delta t}}}{\partial \Delta \hat{\boldsymbol{\varepsilon}}^{\mathrm{t+\Delta t}}} \tag{4}$$

$$\left[K^{t+\Delta t}\right]=\int_V [B]^T[J][B]dV \quad (5)$$

.

Here $[B]=\partial[N]/\partial \mathbf{x}$ contains the spatial derivatives of the element shape functions $[N]$.

There has been considerable interest in the development of numerical algorithms for solving large sparse linear systems of equations and their efficient parallel implementation on HPC systems for more than three decades. The algorithms may be grouped into two broad categories: direct methods and iterative methods.

Iterative method algorithms repeatedly apply a sequence of operations at each step attempting to improve upon its current approximation to a solution. Krylov subspace methods are an important class of iterative methods. This class includes the Conjugate Gradient (CG) method [7,8] and its variants, which are robust for Symmetric Positive Definite (SPD) matrices. In solving the large systems in finite element method, combining a Krylov subspace method such as CG with a preconditioner is essential to accelerating the convergence rate and avoiding divergence of solution especially for ill-conditioned linear systems.

The most widely used direct methods [9] are variants of Gaussian elimination and involve the explicit factorization of the system matrix ***A*** (or, more usually, a permutation of ***A***) into a product of lower and upper triangular matrices ***L*** and ***U***. In the symmetric case, $\boldsymbol{U} = \boldsymbol{D}\boldsymbol{L}^{\boldsymbol{T}}$, where ***D*** is a block diagonal matrix with 1 × 1 and 2 × 2 matrix blocks. The ordering phase reorders the rows and columns such that the factors have reduced fill-in. Symbolic factorization phases analyze the matrix structure to determine an optimal pivoting sequence and strategy for optimal factorization. Forward elimination (numerical factorization) is by far the most computationally expensive part in solving a sparse linear system and the primary focus of this work. Numerical factorization is followed by backward substitution that completes the solution process for each given right-hand side ***b***.

For some tough (ill-conditioned) linear systems that arise in a number of application areas with finer unstructured computational meshes on irregular geometries and model coupling (e.g. multi-scale, multi-physics concurrent modeling), explicit time discretization imposes a constraint on the

maximum time step directly proportional to the grid edge length. Increasing the level of detail of the geometric models inevitably leads to shorter time steps in explicit schemes. Implicit methods with iterative solvers often diverge, or finding and computing a good preconditioner for use with an iterative method can be computationally more expensive than using a direct method. Therefore, implicit methods with direct solvers are often the only feasible methods.

In the case of nonlinear problems, moreover, the matrix structure does not change for the linear solvers within each NR non-linear iteration in the equation (2). Ordering and symbolic factorization phases are performed only for the first NR iteration for each step. For every subsequent nonlinear iteration, only the factorization and backward-solve need to be called with direct solvers. Furthermore, in a modified NR [5] the reordering/analysis and factorization is done only once during the first iteration. The tangent stiffness matrix [K] on the left hand side of equation 1 is retained and is not changed during the subsequent iterations. Only the right hand side vector is updated during each iteration and solved with extremely cheap backward solve until convergence is achieved. The savings in computational time per iteration compensates for the lower convergence rate compared to the full NR method.

The main advantages of direct methods are their generality and robustness. A significant weakness of direct methods, however, is that the matrix factors are often significantly denser than the original matrix, and for large problems such as those that arise from discretization of three dimensional partial differential equations, insufficient memory for both forming and then storing the factors can prevent the use of direct methods. The limitation on CPU computing power and memory requirements had made the use of direct solvers uneconomical in the past, resulting in broad use of iterative solvers. The recent rise of terascale and petascale computational resources has greatly increased the efficiency and practicality of using direct solvers for large sparse systems.

Kilic et al. [10] showed that direct solvers provide a faster solution than iterative solvers in implicit structural dynamics with ill-conditioned coefficient matrices. Multifrontal parallel distributed symmetric and unsymmetric solvers [11] in MUMPS were evaluated by Amestoy [12]. Gould et. al.[13] assessed the performance of direct solvers for symmetric matrices. The solvers were executed

on a single processor for matrices of order greater than 10,000. Parallel performance of both direct and iterative solvers on proprietary IBM HPC platforms with matrices of 1-2 million unknowns were studied by Gupta et al. [14]. Wozniak et al. [15] have analyzed theoretically and experimentally the performance of multi-frontal direct solvers on distributed memory parallel machines. A general comparison of several sparse linear solvers on a modern multi-core HPC cluster with large sparse matrices originating from practical 3D FEA discretization has recently been performed lately by Koric et al. [16], showing that Watson sparse matrix package (WSMP) [17, 18] is the only direct solver that has shown sufficient scalability and robustness to tackle problem sizes of tens of millions of ill-conditioned FEA equations on many thousands of processor cores.

Besides global matrix solver methods, direct sparse methods are also used in domain decomposition methods, such as Finite Element Tearing and Interconnecting FETI [19-20] and its dual prime follow up FETI-DP [21]. The global problem is partitioned in local subproblems, which are solved independently of each other, often by direct methods. Besides the implementation complexity of these methods, solving some additional coarse global problem such as the primal Schur matrix in FETI-DP is the price for decoupling computations to introduce parallelism. The Schur matrix is a distributed matrix with a very low ratio of rows per processor, thus the amount of communication is extremely high in contrast to the computation that can be done locally and is mostly responsible for observed decreasing computational efficiency of the FETI-DP implementation on large number of domains [21].

2. WSMP: A Hybrid Shared and Distributed-Memory Parallel Solver

The Watson Sparse Matrix Package, WSMP [17,18], is a high-performance, robust, and easy-to-use software package for solving large sparse systems of linear equations. It has been under constant improvement and development for over two decades, and can be used as a serial package, in a shared-memory multicore environment, or as a scalable parallel solver in a message-passing environment. A distinctive aspect of WSMP is that it exploits both shared-memory (SMP) and distributed-

memory parallelism using Pthreads and MPI, respectively, while mostly shielding the user from the details of the architecture.

WSMP can perform either Cholesky ($\boldsymbol{LL^T}$) or $\boldsymbol{LDL^T}$ factorization on symmetric sparse matrices. A highly scalable parallel algorithm is used for this step. The parallel symmetric factorization in WSMP is based on the multifrontal algorithm [11,23]. Cholesky ($\boldsymbol{A = LL^T}$) factorization [11] is the simplest of direct methods applicable to the important class of linear systems with symmetric positive-definite (SPD) coefficient matrices. The algorithmic descriptions and experimental results in this paper pertain to sparse Cholesky factorization only. However, the basic approach is applicable to other factorization variants for symmetric indefinite and unsymmetric matrices fully supported in WSMP. Given a sparse matrix and the associated elimination tree, the multifrontal algorithm can be recursively formulated as follows.

2.1 Multifrontal Algorithm in WSMP

Consider an $N \times N$ matrix $\boldsymbol{A}$. The algorithm performs a postorder traversal of the elimination tree associated with $\boldsymbol{A}$. There is a frontal matrix $\boldsymbol{F^i}$ and an update matrix $\boldsymbol{U^i}$ associated with any vertex i. The row and column indices of $\boldsymbol{F^i}$ correspond to the indices of row and column i of $\boldsymbol{L}$, the lower triangular Cholesky factor, in increasing order. In the beginning, $\boldsymbol{F^i}$ is initialized to an $s \times s$ matrix, where s is the number of nonzeroes in the lower triangular part of column i of $\boldsymbol{A}$. The first row and column of this initial $\boldsymbol{F^i}$ is simply the upper triangular part of row i and the lower triangular part of column i of $\boldsymbol{A}$. The remainder of $\boldsymbol{F^i}$ is initialized to all zeroes.

After the algorithm has traversed all the subtrees rooted at a vertex i, it ends up with a $(m+k) \times (m+k)$ frontal matrix $\boldsymbol{F^i}$, where $m+k$ is the number of nonzeroes in the lower triangular part of column i in $\boldsymbol{L}$. The row and column indices of the final assembled $\boldsymbol{F^i}$ correspond to $m+k$ (possibly) non-contiguous indices of row and column i of $\boldsymbol{L}$ in increasing order. If i is a leaf in the elimination tree of $\boldsymbol{A}$, then the final $\boldsymbol{F^i}$ is the same as the initial $\boldsymbol{F^i}$. Otherwise, the final $\boldsymbol{F^i}$ for eliminating vertex i is ob-

tained by merging the initial $\boldsymbol{F}^i$ with the update matrices obtained from all the subtrees rooted at i via an extend-add operation. The extend-add is an associative and commutative operator on two update matrices such the index set of the result is the union of the index sets of the original update matrices. After $\boldsymbol{F}^i$ has been assembled, a single step of the standard dense Cholesky factorization is performed with vertex i as the pivot. At the end of the elimination step, the column with index i is removed from $\boldsymbol{F}^i$ and forms the column i of $\boldsymbol{L}$. The remaining $m \times m$ matrix is called the update matrix $\boldsymbol{U}^i$ and is passed on to the parent of i in the elimination tree.

We assume that the supernodal tree is binary in the top log p levels. The portions of this binary supernodal tree are assigned to the nodes using a subtree-to-subcube strategy illustrated in Figure 1(b), where eight nodes are used to factor the example matrix of Figure 1(a). The subgroup of nodes working on various subtrees are shown in the form of a logical mesh labeled with P. The frontal matrix of each supervertex is distributed among this logical mesh using a bitmask based block-cyclic scheme. Figure 1(b) shows such a distribution for unit block size. This distribution ensures that the extend-add operations required by the multifrontal algorithm can be performed in parallel with each node exchanging roughly half of its data only with its partner from the other subcube. Figure 1(b) shows the parallel extended process by showing the pairs of nodes that communicate with each other. Each node sends out the shaded portions of the update matrix to its partner. The parallel factor operation at each supervertex is a pipelined implementation of the dense block Cholesky factorization algorithm. Often, each MPI process is multithreaded, and the portion of multifrontal factorization assigned to each process if further parallelized.

The multithreaded algorithm for numerical factorization is similar to its message-passing counterpart. There are, however, significant differences in implementation. Just like the message-passing algorithms, tasks at each subroot of the elimination tree are assigned to independent groups of processors until each processor ends up with its own subtree. The most important difference is that a mapping of rows and columns to core processors based on the binary representation of the indices is not used in the SMP implementation because all core processors can access all rows and columns with the same overhead. This lifts the restriction that the subtree assigned to a group of P processors be

binary in its top $\log_2 P$ levels. In the portion of the elimination tree that is executed in the SMP mode, therefore, the number of threads assigned to work on a subtree at a branching point is roughly proportional to the amount of work associated with that subtree. This ensures a high degree of load-balance. Since the relative ratio of work in a subtree with respect to its siblings in the factorization and solve phases is different, moreover, the mapping of subtrees to subgroups of processors in these two phases can be different - a flexibility that is not available in the message-passing portion of the code where the same subtree-to-subcube mapping is used in both factorization and triangular solves.

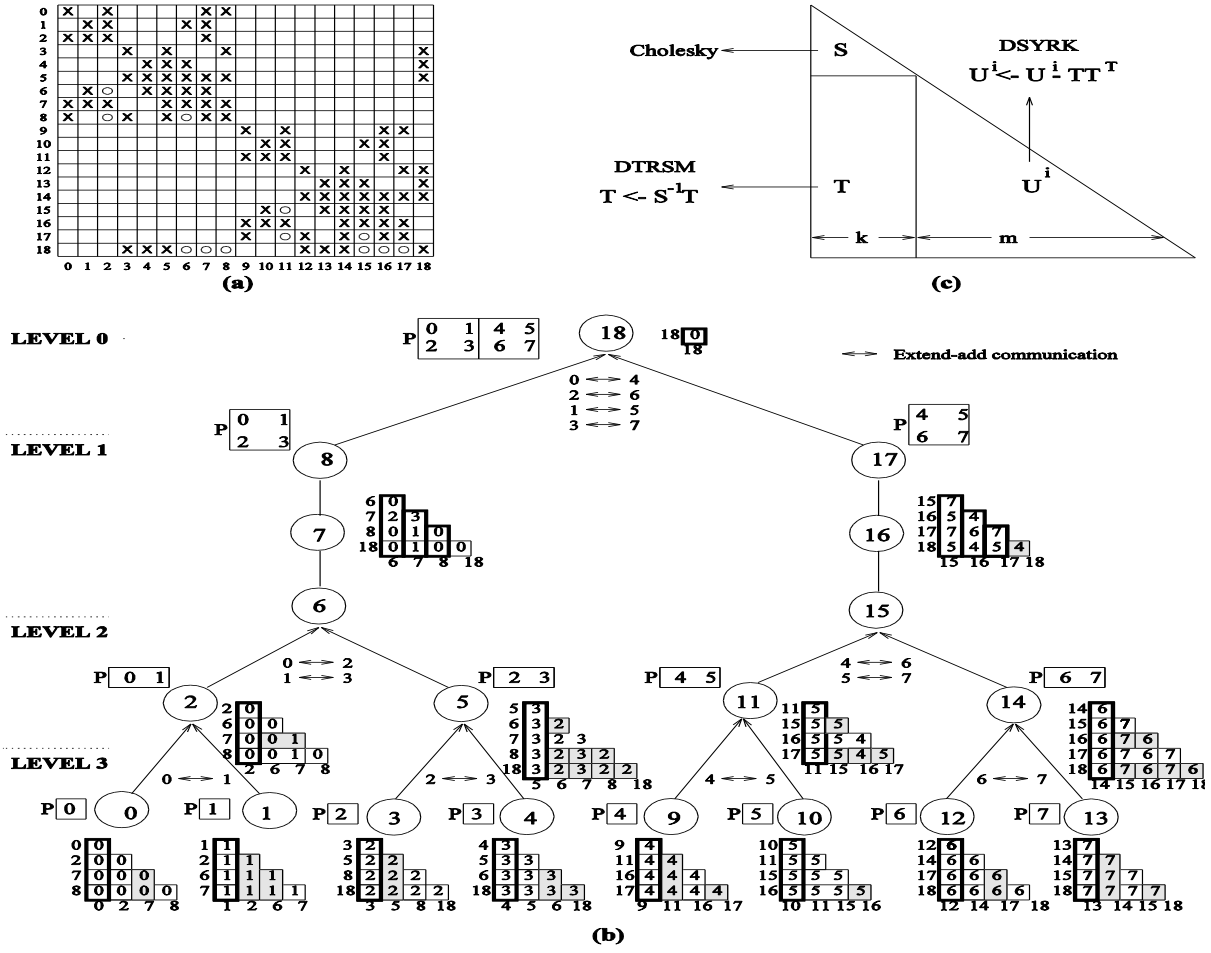

Figure 1: (a) An example symmetric sparse matrix. The nonzeroes of A are shown with symbol "x" in the upper triangular part and nonzeroes of L are shown in the lower triangular part with _fill-ins denoted by the symbol "o". (b) The process of parallel multifrontal factorization using 8 nodes. At each supervertex, the factored frontal matrix, consisting of columns of L (thick columns) and update matrix (remaining columns), is shown. (c) Computation at a typical supernode.

3. Test Cases and Computing Platform

CAD models of solid geometries are commonly used to design parts and assemblies and create their corresponding part drawings for manufacturing. Conveniently, these models can be imported into FE packages for subsequent numerical analysis. While structured meshing with hexahedral elements exhibit higher convergence rate and accuracy, it frequently requires user intervention and is labor intensive. Automatic unstructured mesh generation with tetrahedral elements is quick and often preferred under a tight industrial project schedule. In biomechanics, however, finite element meshes generated from computed tomography (CT), are almost always consisting of tetrahedral elements due to complex geometry of bone segments. Comparisons between linear T4 and quadratic T10 tetrahedral elements, [24-25], have shown that linear element should be avoided due to their stiff nature and volumetric and shear locking.

The global stiffness matrices $\boldsymbol{A}$, and the load vectors $\boldsymbol{b}$ used in this study are extracted from a commercial FEA software NX Nastran [26] via a DMAP procedure. They represent real industrial CAD geometries and loads, and are automatically meshed with T10 elements. Figures 2 and 3 show two of those geometries: A symmetric machine part cutter with asymmetrical loads, and a header part of a Charge Air Cooler (CAC) with complex geometry whose elements have higher aspect ratios and therefore a higher condition number. The element size control has produced different levels of automatic refinement. No additional intervention was applied to improve these unstructured meshes to emulate a standard accelerated FEA workflow. Table 1 summarizes the three (3) test matrices varying from 11 to 40 million degrees of freedom (DOFs). The condition number, as a direct indicator of ill-conditioning, is driven mainly by irregular element shapes discretizing complex details of the geometries, and it varies from 10^6 to 10^7 for the mildly ill-conditioned cutter meshes to 10^9 for the M11 system extracted from the CAC mesh. The size of the largest system, with over 40 million DOFs and 3.3 billion nonzeros, is the largest ever to be benchmarked with direct solver on this scale as best as we can discern from our review of existing literature

Table 1 Test matrix characteristics

Matrix	Source Model	Dimension	Nonzeros	Condition Number
M11	CAC	11,562,627	937,454,416	6.80E+09
M20	Cutter	20,056,050	1,634,926,088	2.70E+07
M40	Cutter	39,979,380	3,290,344,248	5.80E+07

The hardware we used is the sustained peta-scale system of Blue Waters [27] hosted at the University of Illinois' National Center for Supercomputing Applications (NCSA). Blue Waters is one of the most powerful supercomputers currently available for the open science community. Sponsored by the US National Science Foundation (NSF) and installed at the National Center for Supercomputing Applications (NCSA) in Illinois, Blue Waters is also the largest machine to date ever built by Cray.

Blue Waters consists of traditional Cray XE6 compute nodes (each containing two AMD Interlagos processors with 16 floating point cores/XE6 node) and accelerated XK7 compute nodes (each containing a single AMD Interlagos processor with 8 floating point cores and a single Nvidia Kepler GPU) in a single Gemini interconnection fabric. The solver library is ported and tuned to take full advantage of increased memory bandwidth and SSE instructions of XE6 architecture and is linked with AMD ACML math library for Basic Linear Algebra Operation (BLAS) [28] operations.

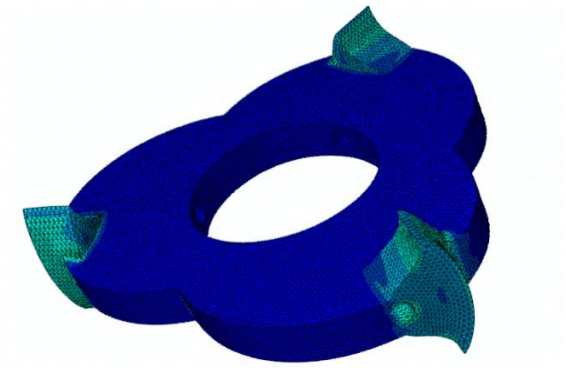

Figure 2. Finite element mesh of the M20 and M40 cutter models

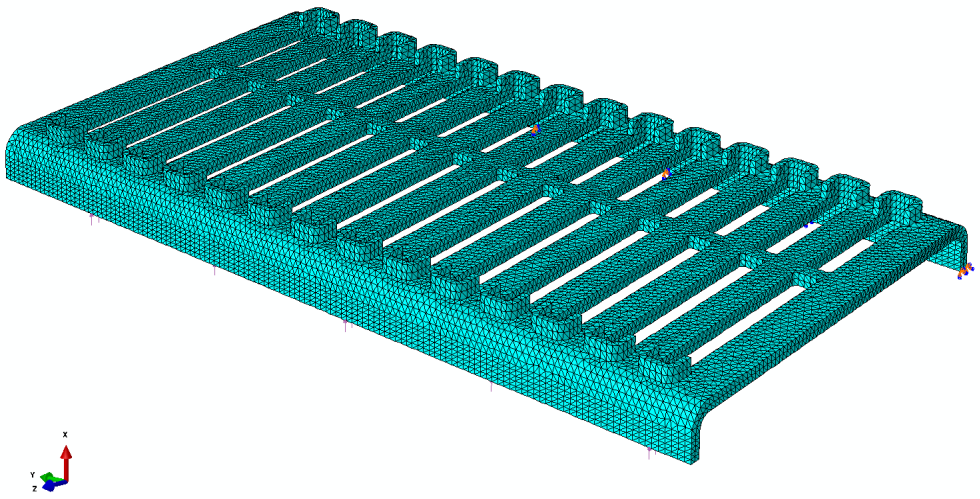

Figure 3. Finite element mesh of the M11 CAC model

4. Results and Discussion

Table 2 shows factorization time, parallel Speedup and performance for all 3 cases, We have started benchmarking all three (3) cases with the minimum number of nodes that can fit the largest M40 case, which is 64 nodes with 1 thread per node for the total of 64 threads. For other runs we have fully utilized 8 or 16 floating point units on XE6 nodes by spawning one or two MPI ranks per node each with 8 threads. In fact, we have compared the jobs with two MPI ranks per node with 8 threads per MPI rank against the comparable jobs on twice as many nodes with a single MPI rank per node and 8 threads and found no significant differences in performance. This indicates that WSMP does not suffer from memory access bandwidth saturation on Blue Waters.

$\boldsymbol{LL^T}$ factorization wall clock time (on the log scale) for all cases is given in Figure 4. It took less than 18 seconds to factor the largest M40 matrix on 65,536 cores opening the possibilities to solve large scale linear and nonlinear FEA problems extremely efficiently. Parallel Speedup, equation (6), Figure 5, is defined as the ratio of sequential wall clock time over wall clock time on p cores, and is a direct indicator of how much benefit we get by solving the systems in parallel, and how well the solvers scale on parallel computers.

$$S_p = \frac{T(1)}{T(p)} \quad (6)$$

An ideal Speedup is assumed for all the cases on the minimum number of nodes (cores) that can fit M40, (Sp = 64 on 64 cores). Based on this assumption, sequential wall clock times are calculated for each problem size and then equation (6) is used to calculate Speedup on larger core counts. Note that this assumption is close to reality since the factorization in WSMP indeed scales ideally or even super-linearly on lower number of cores for multi-million equation problem sizes given enough memory is provided [16].

All three test cases experience a small super-linear Speedup (Sp > p) at 256 and 512 cores, while it extends to 1024 cores for M40. On these scales computation is still more important than communication, and with more nodes more cache is available and therefore more data is stored to cache than to memory. Thus, the memory access time is dramatically reduced, which causes the extra speedup in addition to that from the actual computation. Scaling wider incurs more communication and synchronization overheads, especially for the smaller problem M11, that lowers the Speedup to under-linear (ideal) values. Nevertheless, the Speedup remains high for larger problems. This is particularly the case for the M40 case where Sp reaches 13,179 and 76.4 TFlops (Figure 6) at the largest number of threads (cores) 65,536 used in this study. This indicates that the factorization algorithm in WSMP is exceedingly well-parallelized, while the load is well-balanced among the MPI ranks and their threads. This is also due to the Cray's proprietary Gemini interconnect between XE6 nodes of Blue Waters, having lower latency and better bandwidth than the interconnect fabric found on most modern Linux x86 clusters. Figure 7 shows the total measured data traffic to and from Gemini interconnect when solving the largest M40 case. A sharp increase in the interconnect traffic by over 30 times is observed going from 64 to 4096 nodes indicating that communication becomes more important than computation on the large scale.

The results for the M20 and M11 cases on 65,536 cores are not shown since the workload assigned to each processing element significantly decreases for these cases reaching the scalability limit

– that is, the point at which the wall clock time stops decreasing. In general, it is harder to achieve good strong-(fixed-size) scaling at larger process counts since the communication overhead for many parallel algorithms increases in proportion to the number of processing cores used. Even though the commercial FEA codes nowadays can directly solve larger then M40 problems on the latest HPC hardware, the M40 scaling on 65,536 Blue Waters cores represents, to the author's best knowledge, the highest sparse matrix factorization parallel scaling reported to date in the literature.

In Figure 8, we also compare the peak memory used per MPI rank for all 3 test cases.. It takes at least 64 nodes to fit the M40 under the 64 GB RAM available on XE6 nodes signifying that the HPC clusters with smaller number of nodes would need more memory per node to directly solve large FEA problems with tens of millions of equations (DOFs). The memory usage significantly drops to 9 GB/MPI rank, 18GB/node, on 4096 nodes (65536 cores) showing that the systems larger than M40 can be solved with WSMP on Blue Waters or other similar modern large HPC systems. Theoretical computational complexity of sparse matrix factorization in 3D problems is $O(N^2)$ [9]. Since M40 is roughly twice as large then M20, and since they are both discretizing the identical domain, the theoretical computational cost of factoring M40 should be approximately four times higher than for M20. Figure 9 shows the ratio of factorization times for M40 over M20 as a function of number of parallel threads. As both problems are scaled wider, the superior parallel efficiency of M40 over M20 significantly reduces the computational complexity of sparse matrix factorization.

Finally from Figure 5, the transition from superliner/ideal to sublinear speedup incurs a visible drop in the speedup values for all 3 test cases. This speedup gradient is somewhat recovered at larger scales and until the scalability limits are approached for each case when the speedup starts gradually dropping again. Since the transition happens for M40 at 2048 cores, and later then for M20 which is already partially recovered at that scale, the difference between the M40 and M20 speedups is smallest at 2048 cores, and thus a local maximum is present in Figure 9 at 2048 cores. The partial speedup recovery at around and larger than 128 nodes (2048 cores) is due to a stricter 3D node topology enforced in Gemini interconnect [27] that uses more of fast X and Z directional links for larger node jobs compared to the slower Y-directional links preferred by small node count jobs. This is another

proof that a fast interconnect is a crucial requirement for efficient sparse matrix factorization on large scale.

Table 2, Factorization times in seconds, Speedup and Performance in TFlop/sec

Threads	M11			M20			M40		
	Time	Sp	Perf	Time	Sp	Perf.	Time	Sp	Perf.
64	121	64	0.29	891.3	64	0.39	3645	64	0.39
128	57.9	136.1	0.62	457	124.8	0.78	1696	137.6	0.85
256	28.4	272.6	1.21	215.3	265	1.67	857	272.2	1.67
512	15.7	527.1	2.28	113	504.8	3.13	421	554.1	3.38
1024	10.8	717	3.1	73.5	775.7	4.95	223.1	1045.6	6.38
2048	7.6	1019	4.5	43	1326.6	8.33	147.2	1684.8	9.62
4096	4.8	1596.7	6.9	31.1	2034.2	11.9	76.3	3057.4	18.36
8192	3.7	2070.6	8.78	17.6	3233.8	20.0	45.5	5166.7	30.9
16384	3.1	2458.4	10.4	12.6	4527.4	27.7	33	7060.5	41.1
32768				11.8	4813.9	29.73	22.7	10272	59.99
65536							17.7	13179.7	76.40

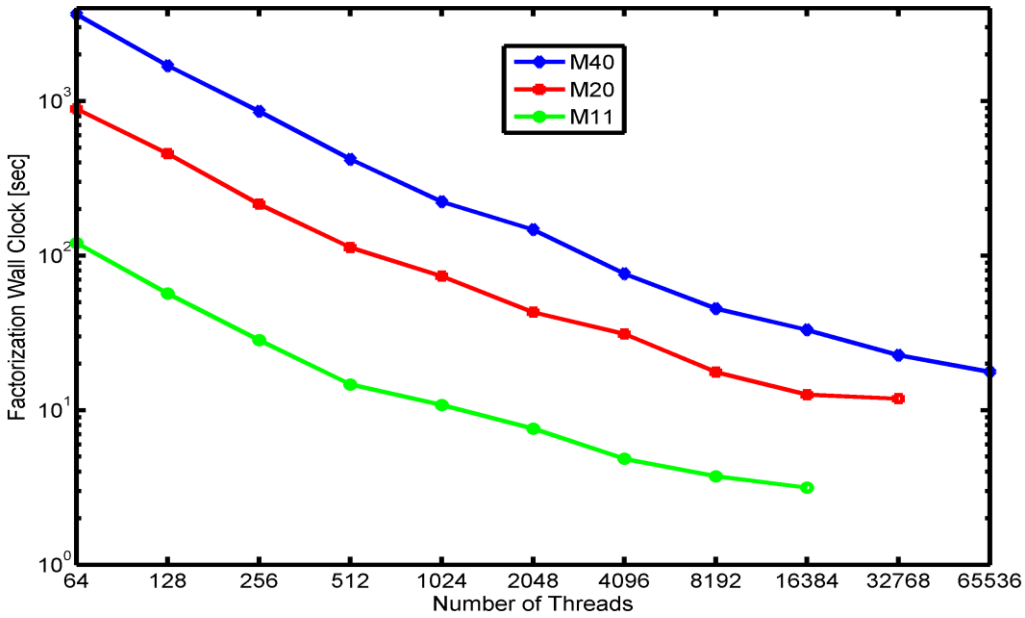

Figure 4. Factorization times

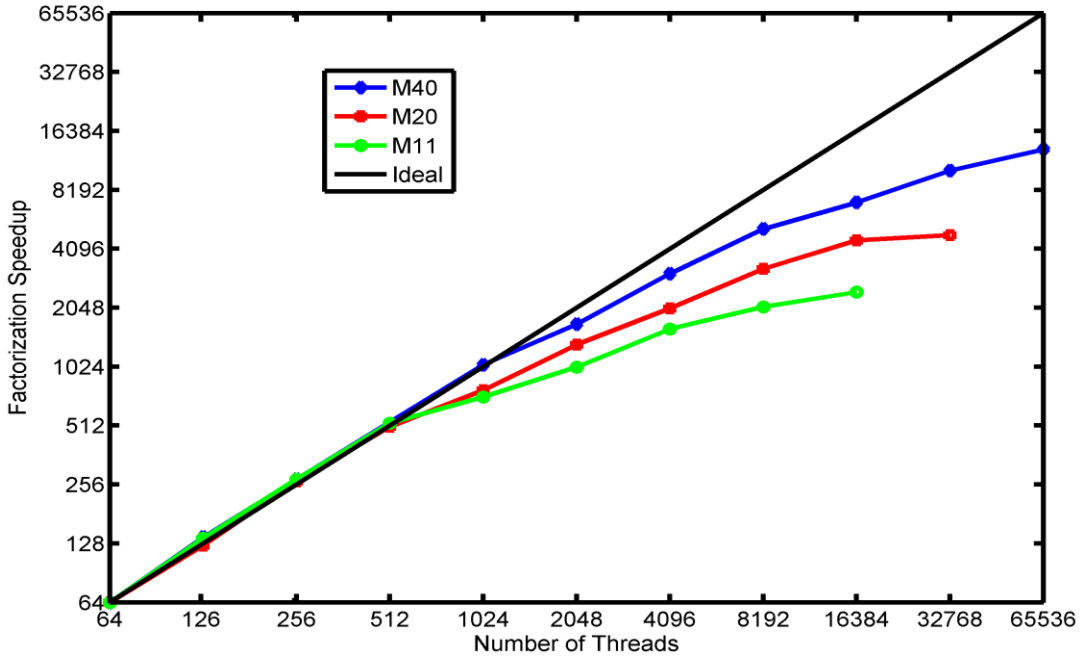

Figure 5 Parallel speedup

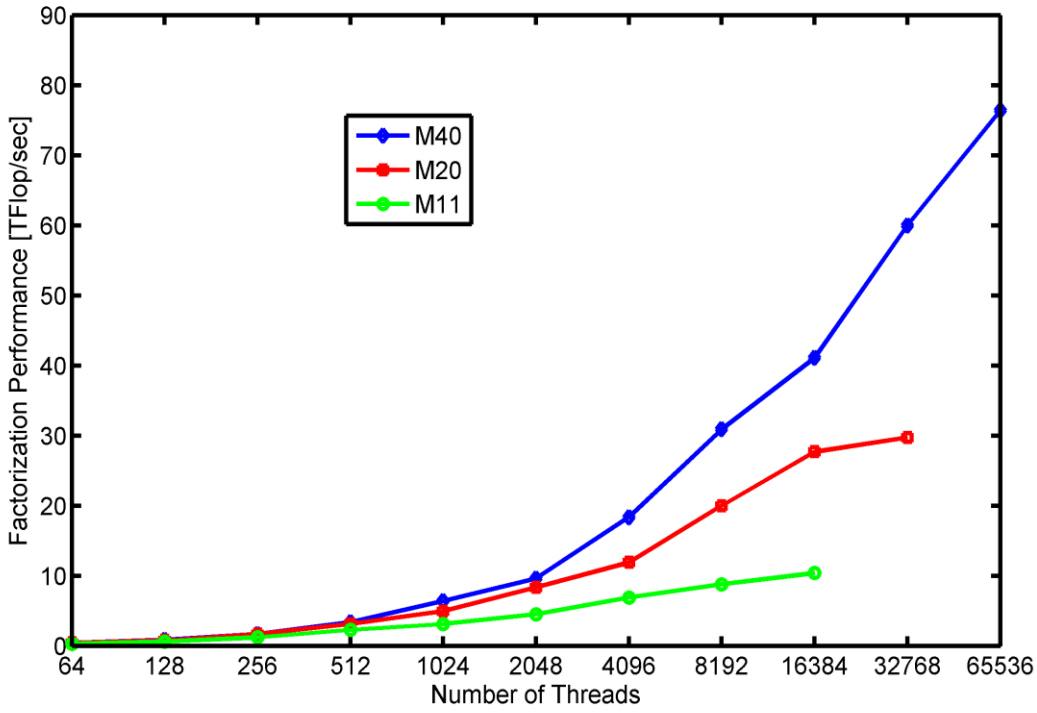

Figure 6. Factorization Performance

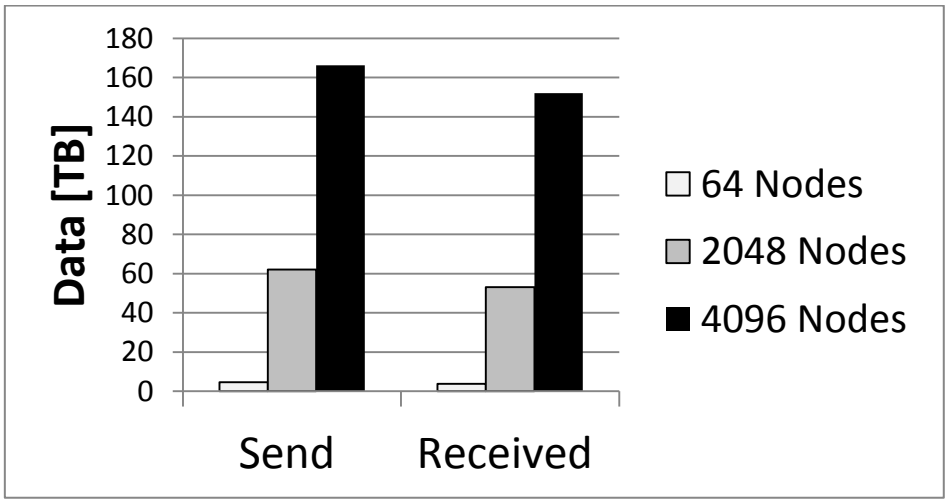

Figure 7. Total Data Sent and Received from Interconnect for M40

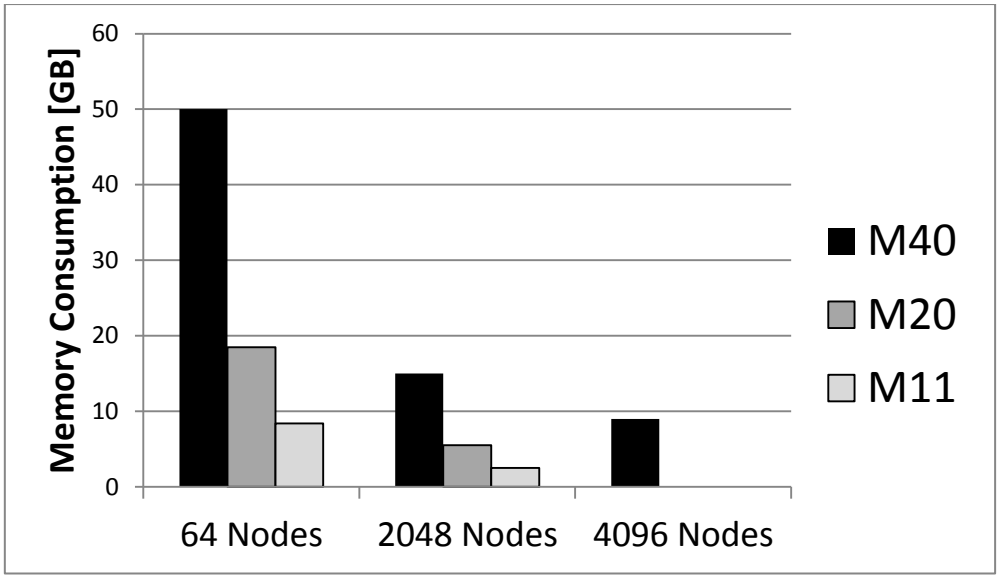

Figure 8. Peak memory consumption per MPI rank

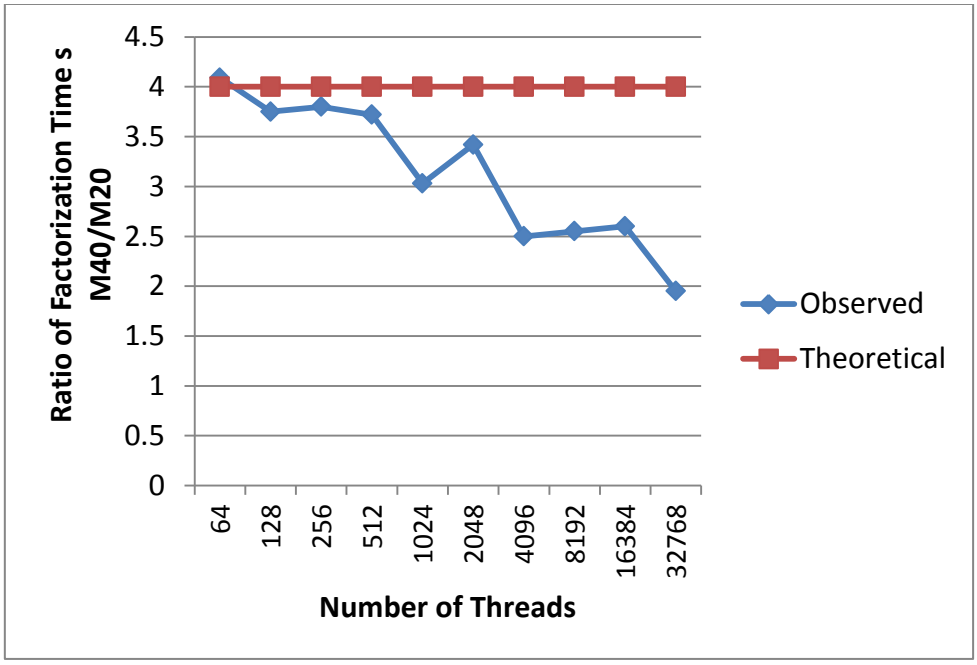

Figure 9. Observed Reduction of Computational Complexity for Parallel Sparse Factorization

5. Conclusions

The factorization performance of the highly scalable direct sparse linear solver WSMP is evaluated on the sustained peta-scale HPC system of Blue Waters. The test systems are extracted via a com-

mercial finite element code from practical solid mechanics engineering problems that are discretized with 3D unstructured meshes.

WSMP has shown enough scalability and robustness to solve more than 40 million equations on more than 65,000 of cores with 76.4 TFlops of sustained performance. Its hybrid MPI/Pthreads implementation can take full advantage of large amounts of distributed memory, modern multicore processors, and low latencies and increased bandwidth of leading interconnect network technologies. As the communication surpasses computation with an increased number of MPI domains and nodes, the overall performance of WSMP remains high. A relatively inexpensive symbolic factorization phase computes the static structure of the factors to enable the subsequent numerical factorization to proceed with efficient use of floating point and integer operations. The multifrontal technique ensures that most floating point computation is performed by cache friendly level 2 and level 3 basic linear algebra subprograms (BLAS) [28].

Since iterative solvers are not robust enough to handle a variety of problems in a general purpose finite element program, hybrid multifrontal direct solvers are already default solvers in most of commercial general purpose FEA codes, and their usage will increase as the size, speed, and availability of multi/many core CPUs, memory, and interconnect network grows in HPC. Analysis of system data from the past decade of the top HPC systems [29] indicate that the scale of computing used in the cutting edge HPC systems, such as Blue Waters, is approximately 5 years ahead of that used by the leading industrial adopter. Thus, this exciting technological advance will pave the way to higher fidelity and complexity simulation studies in many fields of engineering.

In our future work, we plan to solve larger than M40 FEA problem sizes on Blue Waters with WSMP as a standalone solver or implemented in open source FEA codes such as FEAP [30], or Alya [31]. In recent years, substantial efforts were undertaken to adapt linear sparse solvers for evolving GPU systems. Work is also underway to port WSMP to the GPU-accelerated Cray XK7 nodes of Blue Waters.

Acknowledgments

The authors would like to thank the Private Sector Program and the Blue Waters sustained-petascale computing project at the National Center for Supercomputing Applications (NCSA). Blue Waters is supported by the National Science Foundation (award numbers OCI 07-25070 and ACI-1238993) and the state of Illinois.

References

[1] P. Mösta, C. D. Ott, D. Radice, L. F. Roberts, E. Schnetter, R. Haas, A large-scale dynamo and magnetoturbulence in rapidly rotating core-collapse supernovae, Nature, 528 (2015), pp. 376-379.

[2] G. Zhao, J. R. Perilla, E. L. Yufenyuy, X. Meng, B. Chen, J. Ning, J. Ahn, A. M. Gronenborn, K. Schulten, C. Aiken, and P. Zhang, Mature HIV-1 capsid structure by cryo-electron microscopy and all-atom molecular dynamics. Nature, 497(2013), pp. 643-646.

[3] M. Winkel, R. Speck, H. Hübner, L. Arnold, R. Krause, P. Gibbon, A massively parallel, multi-disciplinary Barnes–Hut tree code for extreme-scale N-body simulations, Computer physics communications 183(2015), pp. 880-889.

[4] Y. Lee, P. C. Diniz, M. W. Hall, R. Lucas, Empirical Optimization for a Sparse Linear Solver: A Case Study, International Journal of Parallel Programming, 33(2005), pp. 165-181.

[5] C. O. Zienkiewicz CO, L. T. Taylor,The Finite Element for Solid and Structural Mechanics, 6th ed., Butterwoth-Heinemann Elesevier, 2005

[6] S. Koric and B.G. Thomas, Efficient thermo-mechanical model for solidification processes, International Journal for Numerical Methods in Engineering 66(2006) pp. 1955-1989.

[7] J. Dongarra, I. Duff,D. Sorensenand H. Van Der Vorst, Numerical Linear Algebra for High-Performance Computers,Society for Industrial and Applied Mathematics, Philadelphia, 1998.

[8] S. Yousef ,Iterative Methods for Sparse Linear Systems, 2nd ed. Philadelphia: Society for Industrial and Applied Mathematics, 2003

[9] T. A. Davis, Direct Methods for Sparse Linear Systems (Fundamentals of Algorithms),Society for Industrial and Applied Mathematics, Philadelphia, 2006.

[10] S. A. Kilic,F. Saied F, A. Sameh, Efficient iterative solvers for structural dynamics problems Computers and Structures 82(2004)pp. 2363-2375.

[11] I. S. Duff and J. K. Reid, The multifrontal solution of indefinite sparse symmetric linear equations. ACM Transactions on Mathematical Software 9(1983) pp. 302–325.

[12] P.R. Amestoy, I.S. Duff, J.-Y. L'Excellent, Multifrontal parallel distributed symmetric and unsymmetric solvers Comput. Methods Appl. Mech. Engrg. 184 (2000), pp. 501-520

[13] N. I. M Gould,J.A. Scott,Y. Hu,A Numerical Evaluation of Sparse Direct Solvers for the Solution of Large Sparse Symmetric Linear Systems of Equations, ACM Trans. Math. Softw., 33(2006) pp.1-32.

[14] A. Gupta, S. Koric, T. George, Sparse Linear Solvers on Massively Parallel Machines, in: ACM/IEEE Conference on High Performance Computing SC 2009, Portland, Oregon, USA November 2009.

[15] M. Wozniak, M. Paszynski, D. Pardo, L. Dalcin, V. M. Calo, Computational cost of isogeometric multi-frontal solvers on parallel distributed memory machines, Computer Methods in Applied Mechanics and Engineering **284(2015)** pp. 971-987.

[16] S. Koric,Q. Lu, and E. Guleryuz, Evaluation of massively parallel linear sparse solvers on unstructured finite element meshes, Computers and Structures 141(2014) pp. 19-25.

[17] A. Gupta, G. Karypis, V. Kumar, Highly scalable parallel algorithms for sparse matrix factorization , IEEE Transactions on Parallel and Distributed Systems. 8 (1994) pp. 502-520

[18] A. Gupta. WSMP: Watson sparse matrix package (Part-I: direct solution of symmetric sparse systems). IBM TJ Watson Research Center, Yorktown Heights, NY, 2015.

[19] C. Farhat, F-X. Roux, An unconventional domain decomposition method for an efficient parallel solution of large-scale finite element systems. SIAM J. Sci. Stat. Comput. 13(1992) pp. 379–396

[20] C. Farhat, K. Pierson, M. Lesoinne, The second generation FETI methods and their application to the parallel solution of large-scale linear and geometrically non-linear structural analysis problems, Comput. Meth. Appl. Mech. Eng. 184 (2000) pp. 333-374

[21] C. Farhat, M. Lesoinne, P. LeTallec, K. Pierson, and D. Rixen, *FETI-DP:* a dual-primal unified FETI method. I. A faster alternative to the two-level FETI method. Internat. J. Numer. Methods Engrg. 50 (2001) pp. 1523-1544

[22] A. Klawonn and O. Rheinbach, Highly scalable parallel domain decomposition methods with an application to biomechanics. ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik 90(2010) pp. 5–32.

[23] W. Joseph, H. Liu,The multi frontal method for sparse matrix solution: Theory and practice, *SIAM Review*,34(1992) pp. 82–109.

[24] K. Polgar, M. Viceconti, JJ. Connor, JJ., A comparison between automatically generated linear and parabolic tetrahedra when used to mesh a human femur**.** Journal of Engineering in Medicine. 215(2001) pp. 85-94.

[25] O. Cifuentes ,A. Kalbag, Performance study of tetrahedral and hexahedral elements in 3-D finite element structural analysis, Finite Elements in Analysis and Design. 12(1992)pp. 313-318..

[26] NX Nastran User's Manual, Version 8.0, Siemens PLM Software, Plano, TX, 2012.

[27] Blue Waters. Sustained Petascale Computing, NCSA, University of Illinois http://www.ncsa.illinois.edu/BlueWaters 2015.

[28] J. Dongarra, J. Ducroz,I. Duff and S. Hammarling,. A set of level 3 basic linear algebra subprograms, ACM Trans. Math. Softw. 16(1990)pp. 1–17.

[29] J. P. Wolf, D. L. Crawford, A Livermore Perspective on the Value of Industrial Use of HPC at National Laboratories, in: A. Osseyran, M. Giles editors. Industrial Applications of High-Performance Computing, CRC Press, 2015, pp. 205-222

[30] FEAP-A Finite Element Analysis Program, Berkley, http://www.ce.berkeley.edu/projects/feap 2015

[31] Alya System, Barcelona Supercomputing Center, http://www.bsc.es/alya 2015