DOI 10.12694/scpe.v16i2.1087
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 16, Number 2, pp. 153–162. http://www.scpe.org

A SCALABILITY STUDY USING SUPERCOMPUTERS FOR HUGE FINITE ELEMENT VARIABLY SATURATED FLOW SIMULATIONS*

FRED T. TRACY† THOMAS C. OPPE‡ WILLIAM A. WARD§ AND MAUREEN K. CORCORAN¶

Abstract. This paper describes the challenges and scalability results of running a large finite element model of variably saturated flow in a three-dimensional (3-D) levee on a large high performance, parallel computer using a mesh with more than a billion nodes and two billion elements. MPI (Message Passing Interface) was used for the parallelization. The original finite element model consisted of 3,017,367 nodes and 5,836,072 3-D prism elements. The model exhibited three characteristics which made the problem difficult to solve. First, the different soil layers had soil properties that differed by several orders of magnitude. Secondly, there existed a 5 ft × 6 ft × 6 ft region at the toe of the levee where the mesh was refined using 1 in × 1 in × 1 in 3-D prism elements having randomly generated soil properties. Thirdly, variably saturated flow in levees is governed by the highly nonlinear Richards' equation.

A utility program was written to increase the size of the original problem by an arbitrarily large factor by replicating the original mesh in the y direction. A factor of two, for instance, would exactly double the number of elements and double the number of nodes less the interface nodes connecting the two pieces. The original data set was run using 32, 64, 96, and 256 MPI processes (one core per process was used throughout this study) with time to solution taken for each of these process counts. The data set was then magnified by 2 and runs for 64, 128, 192, and 512 processes were made with time to solution again recorded. This procedure was repeated for different numbers of processes and magnification values. The largest data set was generated from a magnification of 350 yielding a mesh of 1,044,246,303 nodes and 2,042,625,200 3-D prism elements. The Cray XE6 and Cray XC30 computers were used in this study. A tabulation of results is presented and analyzed, as well as the significant challenges that occurred in scaling up the problem size. Weak and strong scalability results are also presented in this paper.

Key words: high performance computing, finite element method, variably saturated seepage flow modeling

AMS subject classifications. 35J66, 65Y05, 76S05

1. Introduction.

The maturing of large parallel supercomputers makes it possible to solve problems never before attempted. It is a trivial matter to ask for 100,000 or more cores in a batch submission script, but doing so without experimenting with gradually scaled-up problem sizes often leads to unexpected parallel inefficiencies, waste of resources, or even outright failure. This paper describes the challenges presented and the efforts made to run ever larger finite element groundwater models using up to a billion nodes and two billion 3-D prism elements. Weak and strong scaling were used to measure the success of the effort.

The problem chosen is a finite element simulation of variably saturated flow in porous media [1] where there are significant heterogeneities in the soil and difficult nonlinearities in the governing partial differential equation of Richards' equation [2]. Large systems of linear, simultaneous equations must also be solved hundreds of times using solvers such as the conjugate gradient method [3]. The matrices are often stored in a sparse matrix format leading to matrix-vector operations that involve irregular data patterns and indirect addressing. Clearly, the chosen problem is a difficult real-world problem to tackle.

2. Description of the problem.

The problem consists of steady-state flow through a levee as shown in Fig. 2.1 and idealized in Fig. 2.2 where there are several soil layers with soil properties differing by 5 to 6 orders of magnitude (e.g., from clay with a low hydraulic conductivity to sand with a high hydraulic conductivity). Fig. 2.3 shows a portion of the 3-D mesh of the levee system before a tree with its root system was added at the toe. More details are given in [4]. To model the tree root at the toe of the levee, a 5 ft × 6 ft × 6 ft heterogeneous zone was added (see Fig. 2.4) in which the mesh was refined using 1 in × 1 in × 1 in 3-D prism elements. To simulate heterogeneities, a randomly generated hydraulic conductivity was assigned to each element in this zone. The resulting mesh consisted of 3,017,367 nodes and 5,836,072 3-D prism elements.

*This work was supported in part by a grant of computer time from the Department of Defense High Performance Computing Modernization Program (HPCMP).

†Information Technology Laboratory (ITL), Engineering Research and Development Center (ERDC), Vicksburg, MS, USA.
‡ITL, ERDC, Vicksburg, MS, USA.
§HPCMP, ITL, ERDC, Vicksburg, MS, USA.
¶Geotechnical and Structures Laboratory, ERDC, Vicksburg, MS, USA.

FIG. 2.1. *River side of a levee with trees.*

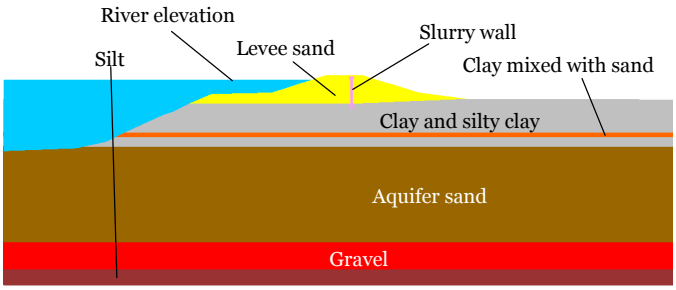

FIG. 2.2. *Cross section of a levee with material types and elevation of the river.*

3. Computational challenges. The many computational challenges inherent in this application were as follows:

- *Complicated geometry.* The geometry of the levee system contained several regions having different soil properties.
- *Wide-ranging material property values.* The soils in the levee ranged from sand to silt to clay, and the associated material properties such as hydraulic conductivity spanned several orders of magnitude [5].
- *Nonlinear system of equations to solve.* The system of equations resulting from the finite element discretization of the levee seepage flow problem was nonlinear because the flow was partially saturated and partially unsaturated. This required the repeated solution of a simultaneous, linear system of equations resulting from either a Picard or Newton linearization in conjunction with line search algorithms [6], [7], and [8]. As the number of nodes and elements increased, this computation became increasingly difficult.
- *Ill-conditioned system of linear equations.* When the soil is partially saturated, material properties such as relative hydraulic conductivity and moisture content become several orders of magnitude smaller as the soil becomes less and less moist. This adds additional stress to solving the linear system of equations at each nonlinear step. A study of preconditioners and solvers for the variably saturated problem is given in [9].
- *Sparse matrix format.* The coefficients of the matrices used in solving the linear system of equations at each nonlinear iteration were stored in sparse matrix format with indirect addressing. This irregular pattern of data decreased the efficiency of the computer for such things as vectorization of the matrix-vector calculations.

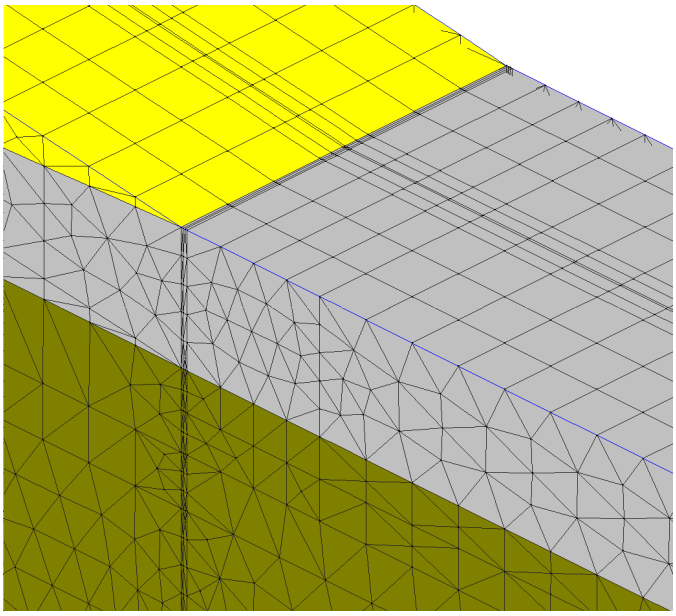

FIG. 2.3. *Portion of the 3-D mesh before the root zone was added.*

4. High performance parallel computing. A 3-D seepage/ground-water finite element program was parallelized using MPI [10] on Garnet, the Cray XE6 at ERDC [11], and on Lightning, the Cray XC30 at the Air Force Research Laboratory, Aberdeen, MD, USA [12]. Garnet consists of 4,716 dual-socket compute nodes with each socket populated with a 2.5 GHz 16-core AMD 6200 Opteron (Interlagos) processor. Each node has 64 GB memory (60 GB user-accessible) or an average of 1.875 GB memory per core. The interconnect type is Cray Gemini in a 3-D torus topology. Garnet is rated at 1.5 peak PFLOPS or 10 GFLOPS per core. Garnet has a large Lustre file system that is tuned for parallel I/O. Lightning consists of 2,360 dual-socket compute nodes with each socket populated with a 2.7 GHz 12-core Intel Xeon E5-2697v2 (Ivy Bridge) processor. Each node has 64 GB memory (63 GB user-accessible) or an average of 2.625 GB memory per core. The interconnect type is Cray Aries in a Dragonfly topology. Lightning is rated at 1.2 peak PFLOPS or 21.6 GFLOPS per core. Lightning has a large Lustre file system that is tuned for parallel I/O.

The parallelization of the 3-D seepage/groundwater program was broken into four separate parts, and each part was a stand-alone FORTRAN or C computer program. One MPI process was placed on each core of a compute node. The four programs will now be described.

- *Partitioner.* The mesh is partitioned using the Parallel Graph Partitioning and Fill-reducing Matrix Ordering program, ParMETIS [13], to divide the mesh into approximately equal pieces among the processes. This is illustrated in Fig. 4.1 where three parallel processes have the nodes of the mesh divided among them. The output of the first version of ParMETIS used was a single file containing a line for each finite element node stating to which process the given node belongs. As the problem size grew, the version of ParMETIS that distributes the output file over the processes in a block partition style was used. As seen in Fig. 4.1, some elements contained nodes belonging to different processes. Each element is assigned to that process that owns the first node of the element. Nodes of an element belonging to a particular process that are not owned by that process are called ghost nodes. Also some elements have nodes belonging to a particular process but that element does not belong to that process. These are called ghost elements, and they create even more ghost nodes.
- *Preparer.* The next piece of the solution creates a file for each process that contains the owned and ghost nodes and elements for that process using local numbers (see Sect. 5.2.2), a subset of the boundary condition file and initial condition file belonging to that process, a list of nodes where data are to be sent to every other process, a list of nodes where data are to be received from every other process, and

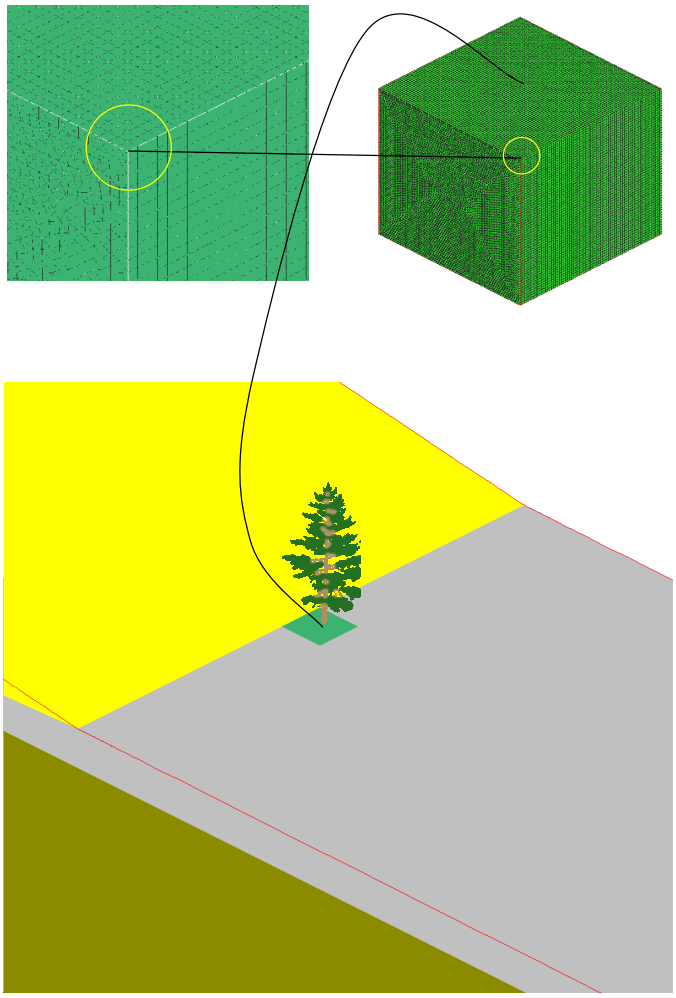

FIG. 2.4. *Heterogeneous zone representing the roots of a tree.*

global node and element numbers for the owned and ghost nodes and elements on that process.

- *Finite element program.* This part does the finite element computations with resulting output files containing results for each owned node of that process. This is the primary focus of the time-to-solution results presented later in this paper.
- *Postprocessor.* This final part is a postprocessor program that combines all data from each process into the final output files.

5. Challenges.

5.1. Partitioner. This original version of this program (written in C) simply reads in the geometry data file and calls one of the ParMETIS programs to produce a single file with a line of data for each node that gives the process that owns the node. That technique was adequate for the original problem with 3,017,367 nodes and smaller versions of the bigger meshes. For example, when the original data set is doubled ($m = 2$), the mesh has 6,000,831 nodes, and the partitioner program took only 6.3 seconds to run. At this size of problem, it was possible to use global arrays to store the mesh. However, as the problem size grew, the geometry files also continued to grow, and the time for reading this file into the program continued to grow as well.

5.1.1. No global arrays. The finite element mesh data needed for the partitioner took 710 Mbytes of memory for the original problem. However, magnifying the problem to $m = 100$, for instance, required 68

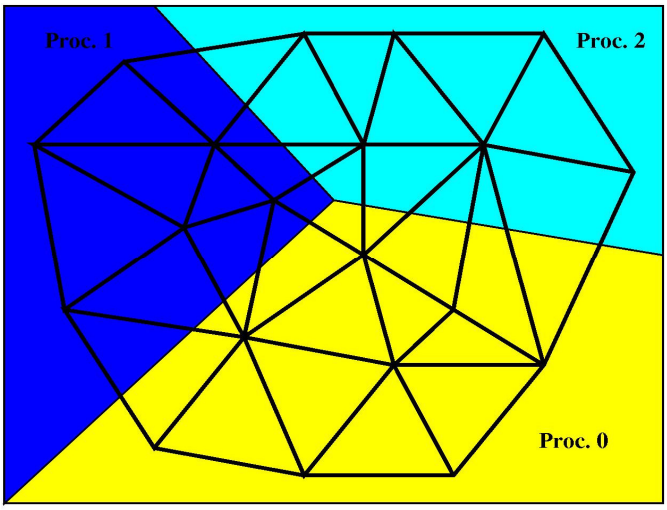

FIG. 4.1. *An example of dividing a finite element mesh into several partitions.*

Gbytes of memory to store this same data. This clearly illustrated why no global arrays can be allowed in any part of this application. To completely avoid this use of global arrays makes the process of parallelization much more difficult. It was decided to first read the mesh data and store it in the block partition format. Then the adjacency table routine [14] had to be modified to use this partitioned data. Finally, a different version of ParMETIS was called and the output file written to all the processes in block partition format.

5.1.2. No ASCII data file containing the entire mesh. The original version of this application had as one of its ASCII input files a file containing the nodes and elements. A node line began with the characters, GN, and an element line began with GE. Further, these type lines could be mixed in any order, requiring the reading and processing of one line at a time. As the problem size grew, the time to process the large geometry data files became prohibitive. For example, at $m = 2$, the entire partitioner took 6.3 sec on Garnet and for $m = 4$, the time was 11.8 sec on Garnet. However, at $m = 100$, the time to read this input file exceeded two hours.

Different solutions were suggested including rewriting the ASCII file as a binary fire. A better solution was to modify the serial program that generated the geometry file in the first place. Instead of writing one huge file, the new version of the serial program wrote node and element data for each process. Further, one file for nodes and one file for elements were written in a simpler format. In this new way, to create the data files for each process for $m = 100$ took 2544.6 sec on Garnet. However, since 3,200 processes were used for the seepage/groundwater program run, this is still less than one second per process. In the case of $m = 100$ in which the grid data could not be read into the partitioner program in two hours using the previous approach, the new approach required only 1.9 seconds on Garnet for the same task.

5.2. Preparation program. When this research project began, it was thought that the parallelization of the actual finite element program would present the greatest challenge. However, because of the restriction of no global arrays, the preparation program was by far the most troublesome and could be improved further from the modifications that were made. Given below is a summary of the challenges faced for the preparation program.

5.2.1. No ASCII data file containing the entire mesh. The same problem existed with the preparation program as with the partitioner in that the time reading the very large geometry file became overwhelming. The same technique of reading the individual node and element files for each process worked well here, too.

5.2.2. Global, local, and ghost nodes and elements. Each node has an original global node number assigned to it, and it is the same everywhere. When the nodes are first and temporarily distributed evenly in

the block partition style to the different processes, each node then has a local node number stating where it is placed on its process. Now ParMETIS dictates that a given node actually belongs to a different process than, in general, it now resides, giving rise to a second local node number. Also, elements are done exactly like the nodes; read in block partition format and then moved to their proper final process. As with the nodes, each element has two different local node numbers. As can be seen, managing all this data is very tedious.

5.2.3. Output files. The finite element program needs various files such that there is one for each process and all data use the final local node and element numbers. These files will now be briefly described.

- *Mesh file.* The list of local (owned plus ghost) nodes and elements are placed in this file. Before writing, the ghost nodes are sorted by increasing process number where the given node resides. For example, suppose for a small problem, process 0 has 10 owned nodes and 2 ghost nodes from process 1, 3 ghost nodes from process 4, and 1 ghost node from process 14. The ghost node numbers are assigned local node numbers 11 and 12 for the process 1 ghost nodes, 13, 14, and 15 for the process 4 ghost nodes, and 16 for the process 14 ghost node. Getting all these data properly numbered and sorted without any arrays containing the entire mesh is remarkably challenging.
- *Initial condition file.* The original initial condition file contains a value of total head for each node to give the finite element program a place to start for iterating to a solution. A value for all the local nodes and elements is now needed for each process.
- *Boundary condition file.* The original file contains parameter data that is simply reproduced for the file for each process. However, boundary conditions are also applied to individual nodes with one example being the total head resulting from the elevation of a river. The original data has for this type of boundary condition the global node number and the total head associated with it. An initial way of handling such a boundary condition was to send the global node number to all the processes and then have each process search through the global node numbers associated with each local node to see if the node is on that process. If the node is on that process, that boundary condition line is written to that process' boundary condition file with the global node number being replaced with its local node number. As the problem size increased, so did the number of such boundary condition nodes to consider. This simple search had to be replaced with a hash formulation [15]. For $m = 100$, the search time using the hash table was 7.5 sec on Garnet.
- *Combination of global node numbers, global element numbers, and ghost node data.* A file for each process containing (1) the global node number for each local node, (2) the global element number for each local element, (3) the number of values and starting memory location for data received from the other processes, and (4) the number and values of the local node numbers where data are to be sent to all the other processes.

5.3. Finite element program. This is the major piece of the seepage/ground-water suite of programs that was tested and timed as it is the primary computational engine. It was initially thought that this would cause the greatest difficulty in running huge problems with large numbers of processes. However, there was only one place where global arrays were being used, so only this algorithm had to be revised. This was the ghost node update portion of the conjugate gradient solver where the data created by the preparer program was stored more efficiently.

6. Results and Analysis.

6.1. Results. Tables 6.1–6.12 give the time to solution for the finite element program on the Cray XE6 and XC30 for different problem sizes (m values) and process counts. The PGI [16] compiler with the compiler option "-fastsse" was used in the computer runs. The original problem was run with 32 MPI processes. The "weak" scaling or speedup in the tables refers to increasing the number of MPI processes the same amount as the problem size is increased. Thus, in Table 6.2, $\frac{T_{m=1,p=32}}{T_{m=2,p=64}}$ means divide the time obtained from running the $m = 1$ problem with 32 MPI processes by the time it took to run the $m = 2$ problem with 64 MPI processes. As another example, when $m = 100$, the number of processes used was 3200 for the weak scaling. The ideal result of the weak scaling ratios is always 1.

"Strong" scaling in the paper refers to keeping the problem size the same while increasing the number of MPI processes and computing the ratio of the original and new running times. For example, $\frac{T_{m=1,p=32}}{T_{m=1,p=256}}$ in Table

TABLE 6.1
Time (sec) for the Cray XE6 and XC30 for $m = 1, nodes = 3017367,$ and $elements = 5836072$.

Processes (p)	Cray	Time (T)	Strong Scaling	Ideal	Weak Scaling	Ideal
	XE6	2195.9				
32	XC30	902.1				
	XE6	2195.8				
..	XC30	899.6				
	XE6	2199.7				
..	XC30	905.8				
	XE6	Avg: 2197.1				
..	XC30	Avg: 902.7			NA	NA
	XE6	1082.7				
64	XC30	458.0				
	XE6	1083.9				
..	XC30	461.3				
	XE6	1079.1				
..	XC30	458.7				
	XE6	Avg: 1081.9	$= 2.03$			
..	XC30	Avg: 459.3	$\frac{T_{m=1,p=32}}{T_{m=1,p=64}} = 1.97$	2		
	XE6	742.9				
96	XC30	320.5				
	XE6	739.1				
..	XC30	318.9				
	XE6	737.2				
..	XC30	313.7				
	XE6	Avg: 739.7	$= 2.98$			
..	XC30	Avg: 317.7	$\frac{T_{m=1,p=32}}{T_{m=1,p=96}} = 2.84$	3		
	XE6	353.1				
256	XC30	128.9				
	XE6	332.1				
..	XC30	122.8				
	XE6	330.1				
..	XC30	127.9				
	XE6	Avg: 338.4	$= 6.49$			
..	XC30	Avg: 126.5	$\frac{T_{m=1,p=32}}{T_{m=1,p=256}} = 7.14$	8		

6.1 refers to dividing the time it took to run the problem when $m = 1$ using 32 MPI processes by the time it took to run the same problem with 256 MPI processes. The ideal scaling or speedup value is always the ratio of the new number of MPI processes and the original number of MPI processes. In the above example, the ideal speedup is 8.

To test strong scaling, the process count was made 2, 3, and sometimes 8 times that of the original process count for given values of m. Values of $m = 2$, 3, 4, 5, 10, 20, 30, 100, 200, 300, and 350 were evaluated. The largest data set where $m = 350$ contains 1,044,246,303 nodes and 2,042,625,200 elements.

Fig. 6.1 shows the results of the weak scaling. It is important to note that between $m = 1$ and $m = 2$, the number of elements exactly doubles, but the number of nodes less than doubles (slightly) due to interface nodes. Thus the amount of computation per process may be slightly less in the $m = 2$ case for double the number of processes. Thus, weak scaling ratios may be slightly higher than they would be if a perfect doubling of the nodes was also done. Nevertheless, the ideal weak scaling is approximately 1. Figs. 6.2, 6.3, and 6.4 show plots of the strong scaling results. Table 6.13 shows a ratio of the running times for the XE6 and XC30 for different values of m. For multiple runs on the same machine using the same value of m, the elapsed times were averaged.

6.1.1. Analysis. The weak scaling was excellent throughout all the problem sizes and both computers with the XC30 results being better at the larger problem sizes. The strong speedup results were good on the XE6 until $m = 200$ and the process count was increased 3 times such that the process count was 19,200. Also, the XE6 did poorly at $m = 300$ when the process count was doubled. The XC30 consistently outperformed the

TABLE 6.2
Time (sec) for the Cray XE6 and XC30 for $m = 2$, nodes = 6000831, and elements = 11672144.

Processes (p)	Cray	Time (T)	Strong Scaling	Ideal	Weak Scaling	Ideal
	XE6	2259.2				
64	XC30	897.9				
	XE6	2196.8				
..	XC30	908.3				
	XE6	2202.5				
..	XC30	904.1				
	XE6	Avg: 2219.5			$= 0.99$	
..	XC30	Avg: 901.8			$\frac{T_{m=1,p=32}}{T_{m=2,p=64}} = 1.00$	1
	XE6	1097.4				
128	XC30	468.4				
	XE6	1093.5				
..	XC30	460.5				
	XE6	1103.1				
..	XC30	460.1				
	XE6	Avg: 1098.0	$= 2.02$			
..	XC30	Avg: 463.0	$\frac{T_{m=2,p=64}}{T_{m=2,p=128}} = 1.95$	2		
	XE6	743.1				
192	XC30	312.9				
	XE6	748.5				
..	XC30	316.4				
	XE6	756.2				
..	XC30	314.7				
	XE6	Avg: 749.3	$= 2.96$			
..	XC30	Avg: 314.7	$\frac{T_{m=2,p=64}}{T_{m=2,p=192}} = 2.85$	3		
	XE6	360.8				
512	XC30	130.4				
	XE6	348.5				
..	XC30	130.2				
	XE6	349.7				
..	XC30	130.5				
	XE6	Avg: 353.0	$= 6.29$			
..	XC30	Avg: 133.7	$\frac{T_{m=2,p=64}}{T_{m=2,p=512}} = 6.72$	8		

XE6 by a factor of approximately 2.5.

7. Consistency check. A system of simultaneous, linear equations is solved each time one of 300 nonlinear iterations is done with the number of unknowns equal to the number of node points. Some parallel programs struggle with getting the same answers when the number of processes is increased. This test is much more difficult in that the size of the problem and the number of processes are increased as m is increased. Table 7.1 gives pressure head results for the first few nodes and the last few nodes of the respective meshes for $m = 1$ and $m = 350$. Because of symmetry, these results should match. These numbers, in fact, match very well.

8. Conclusions. The following conclusions are made from this study:

- It is remarkable that even at over a billion nodes and two billion 3-D elements, the Cray XE6 and Cray XC30 can still be productively used as indicated by the speedup computations when comparing the times to solution of the original problem with data sets up to 350 times larger.
- The weak speedup values were excellent and about the same on both computers.
- The strong speedup capability began to fail for the XE6 at higher values of m. The XC30 values remained good for all problems that were run.
- The XC30 performed approximately 2.5 times better than the XE6.
- It is well known that reading ASCII files can be inefficient. For very large data sets, the problem becomes acute.
- For large problems, the use of global arrays in the source code becomes infeasible.

TABLE 6.3
Time (sec) for the Cray XE6 and XC30 for $m = 3, nodes = 8984295, and\ elements = 17508216$.

Processes (p)	Cray	Time (T)	Strong Scaling	Ideal	Weak Scaling	Ideal
	XE6	2215.1				
96	XC30	906.1				
	XE6	2200.4				
..	XC30	898.0				
	XE6	2195.3				
..	XC30	901.8				
	XE6	Avg: 2203.6			$= 1.00$	
..	XC30	Avg: 902.0			$\frac{T_{m=1,p=32}}{T_{m=3,p=96}} = 1.00$	1
	XE6	1097.3				
192	XC30	462.0				
	XE6	1126.4				
..	XC30	462.6				
	XE6	1117.2				
..	XC30	460.6				
	XE6	Avg: 1113.6	$= 1.98$			
..	XC30	Avg: 461.7	$\frac{T_{m=3,p=96}}{T_{m=3,p=192}} = 1.95$	2		
	XE6	751.0				
288	XC30	312.6				
	XE6	747.4				
..	XC30	313.7				
	XE6	750.2				
..	XC30	314.8				
	XE6	Avg: 749.5	$= 2.94$			
..	XC30	Avg: 313.7	$\frac{T_{m=3,p=96}}{T_{m=3,p=288}} = 2.89$	3		
	XE6	313.7				
768	XC30	141.6				
	XE6	313.0				
..	XC30	132.5				
	XE6	310.5				
..	XC30	130.2				
	XE6	Avg: 312.2	$= 7.06$			
..	XC30	Avg: 134.8	$\frac{T_{m=3,p=96}}{T_{m=3,p=768}} = 6.69$	8		

- Preparing the different data files for the finite element program requires very tedious bookkeeping when using the traditional MPI paradigm of accumulating large amounts of data before sending them.
- Large nonlinear implicit algorithms for 3-D finite element variably saturated flow calculations can be performed with excellent scalability when increasing the problem size to ever larger numbers of elements.

REFERENCES

[1] J. ISTOK, *Groundwater modeling by the finite element method*, AGU, 1989.
[2] L. A. RICHARDS, *Capillary conduction of liquids through porous mediums*, J. of Physics, 1 (1931), pp. 318-333.
[3] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia, 2003.
[4] M. CORCORAN, J. PETERS, J. DUNBAR, J. LLOPIS, F. TRACY, J. WIBOWO, J. SIMMS, C. KEES, S. MCKAY, J. FISCHENICH, M. FARTHING, M. GLYNN, B. ROBBINS, R. STRANGE, M. SCHULTZ, J. CLARKE, T. BERRY, C. LITTLE, AND L. LEE, *Initial research into the effects of woody vegetation on levees, volume I of IV: project overview, volume II of IV: field data collection, volume III of IV: numerical model simulation, and volume IV of IV: summary of results and conclusions*, U.S. Army Engineer Research and Development Center, Vicksburg, MS, 2011.
[5] A. W. WARRICK, *Soil Water Dynamics*, Oxford University Press, 2003.
[6] S. MEHL, *Use of Picard and Newton iteration for solving nonlinear ground water flow equations*, Ground Water, 44 (2006), pp. 583-594.
[7] F. T. TRACY, *Testing line search techniques for finite element discretizations for unsaturated flow*, Proc. of the 9th Int. Conf. in Computational Science, Baton Rouge, LA, May 25-27, 2009.
[8] C. T. KELLEY, *Solving nonlinear equations with Newton's method*, SIAM, 2003.
[9] H. V. NGUYEN, J. C. CHENG, AND R. S. MAIER, *Study of parallel linear solvers for three-dimensional subsurface flow problems*,

TABLE 6.4
Time (sec) for the Cray XE6 and XC30 for $m = 4, nodes = 11967759, and\ elements = 23344288$.

Processes (p)	Cray	Time (T)	Strong Scaling	Ideal	Weak Scaling	Ideal
	XE6	2229.6				
128	XC30	905.1				
	XE6	2224.3				
..	XC30	910.0				
	XE6	2225.2				
..	XC30	914.6				
	XE6	Avg: 2226.4			$= 0.99$	
..	XC30	Avg: 909.9			$\frac{T_{m=1,p=32}}{T_{m=4,p=128}} = 0.99$	1
	XE6	1102.4				
256	XC30	474.6				
	XE6	1102.6				
..	XC30	471.9				
	XE6	1106.0				
..	XC30	467.6				
	XE6	Avg: 1103.7	$= 2.02$			
..	XC30	Avg: 471.4	$\frac{T_{m=4,p=128}}{T_{m=4,p=256}} = 1.93$	2		
	XE6	757.3				
384	XC30	332.9				
	XE6	757.2				
..	XC30	327.4				
	XE6	753.7				
..	XC30	316.1				
	XE6	Avg: 756.1	$= 2.94$			
..	XC30	Avg: 325.5	$\frac{T_{m=4,p=128}}{T_{m=4,p=384}} = 2.80$	3		
	XE6	312.0				
1024	XC30	143.8				
	XE6	311.8				
..	XC30	137.3				
	XE6	326.2				
..	XC30	140.4				
	XE6	Avg: 316.7	$= 7.03$			
..	XC30	Avg: 140.5	$\frac{T_{m=4,p=128}}{T_{m=4,p=1024}} = 6.48$	8		

Proc. of the 9th Int. Conf. in Computational Science, Baton Rouge, LA, May 25-27, 2009.

[10] MESSAGE PASSING INTERFACE FORUM, *MPI: A Message-Passing Interface Standard, Version 3.0*, http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf, 2012.

[11] ERDC DSRC, http://www.erdc.hpc.mil/hardware/index.html, Department of Defense Supercomputing Resource Center, Vicksburg, MS, 2014.

[12] AFRL DSRC, http://www.afrl.hpc.mil/index.html, Department of Defense Supercomputing Resource Center, Aberdeen, MD, USA 2014.

[13] G. KARYPIS, *ParMETIS - Parallel Graph Partitioning and Fill-reducing Matrix Ordering*, http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview, 2014.

[14] D. EPPSTEIN, *Lecture notes: graph algorithms*, http://www.ics.uci.edu/eppstein/161/960201.html, ICS, 161 (1996).

[15] T. H. CORMEN, C. E. LEISERSON, R. L. RIVEST, AND C. STEIN, *Introduction to Algorithms, 3rd ed.*, Massachusetts Institute of Technology, (2009), pp. 253-280.

[16] PORTLAND GROUP, INC., http://www.pgroup.com, 2014.

Edited by: Dana Petcu
Received: Dec 1, 2014
Accepted: Mar 19, 2015

TABLE 6.5
Time (sec) for the Cray XE6 and XC30 for $m = 5$, nodes $= 14951223$, and elements $= 29180360$.

Processes (p)	Cray	Time (T)	Strong Scaling	Ideal	Weak Scaling	Ideal
	XE6	2230.5				
160	XC30	910.6				
	XE6	2222.8				
..	XC30	911.34				
	XE6	2222.8				
..	XC30	911.2				
	XE6	Avg: 2225.4			$= 0.99$	
..	XC30	Avg: 911.0			$\frac{T_{m=1,p=32}}{T_{m=5,p=160}} = 1.00$	1
	XE6	1105.6				
320	XC30	463.7				
	XE6	1105.4				
..	XC30	464.6				
	XE6	1105.3				
..	XC30	463.7				
	XE6	Avg: 1105.4	$= 2.01$			
..	XC30	Avg: 464.0	$\frac{T_{m=5,p=160}}{T_{m=5,p=320}} = 1.96$	2		
	XE6	762.5				
480	XC30	314.9				
	XE6	770.9				
..	XC30	317.4				
	XE6	761.8				
..	XC30	317.4				
	XE6	Avg: 765.1	$= 2.94$			
..	XC30	Avg: 316.6	$\frac{T_{m=5,p=160}}{T_{m=5,p=480}} = 2.88$	3		
	XE6	333.2				
1280	XC30	130.4				
	XE6	335.6				
..	XC30	133.5				
	XE6	354.5				
..	XC30	132.9				
	XE6	Avg: 341.1	$= 6.52$			
..	XC30	Avg: 132.3	$\frac{T_{m=5,p=160}}{T_{m=5,p=1280}} = 6.89$	8		

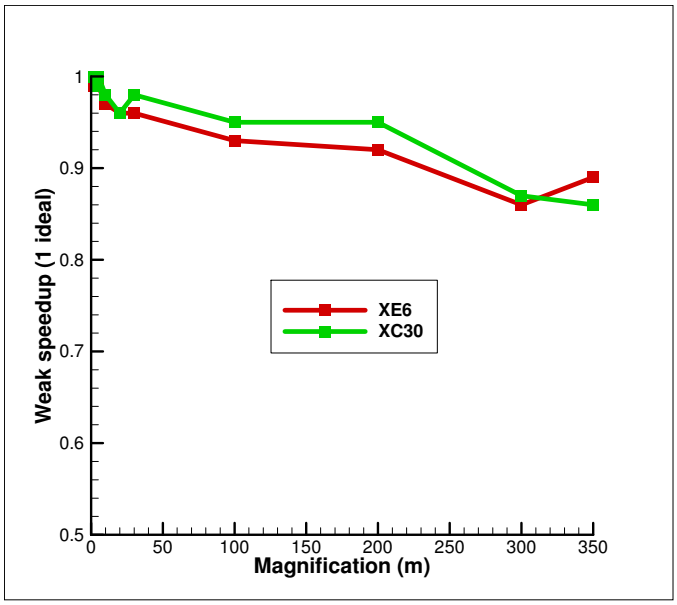

FIG. 6.1. *Weak speedup for different m values.*

TABLE 6.6
Time (sec) for the Cray XE6 and XC30 for $m = 10$, *nodes* $= 29868543$, *and elements* $= 58360720$.

Processes (p)	Cray	Time (T)	Strong Scaling	Ideal	Weak Scaling	Ideal
	XE6	2260.8				
320	XC30	916.4				
	XE6	2264.3				
..	XC30	922.1				
	XE6	2268.6				
..	XC30	914.8				
	XE6	Avg: 2264.6			$= 0.97$	
..	XC30	Avg: 917.8			$\frac{T_{m=1,p=32}}{T_{m=10,p=320}} = 0.98$	1
	XE6	1138.8				
640	XC30	468.9				
	XE6	1143.4				
..	XC30	467.2				
	XE6	1137.2				
..	XC30	468.7				
	XE6	Avg: 1139.8	$= 1.99$			
..	XC30	Avg: 468.3	$\frac{T_{m=10,p=320}}{T_{m=10,p=640}} = 1.96$	2		
	XE6	787.0				
960	XC30	331.1				
	XE6	774.6				
..	XC30	335.9				
	XE6	801.0				
..	XC30	347.1				
	XE6	Avg: 787.5	$= 2.88$			
..	XC30	Avg: 338.0	$\frac{T_{m=10,p=320}}{T_{m=10,p=960}} = 2.72$	3		
	XE6	447.3				
2560	XC30	173.4				
	XE6	404.7				
..	XC30	156.2				
	XE6	350.3				
..	XC30	171.6				
	XE6	Avg: 400.8	$= 5.65$			
..	XC30	Avg: 167.1	$\frac{T_{m=10,p=320}}{T_{m=10,p=2560}} = 5.47$	8		

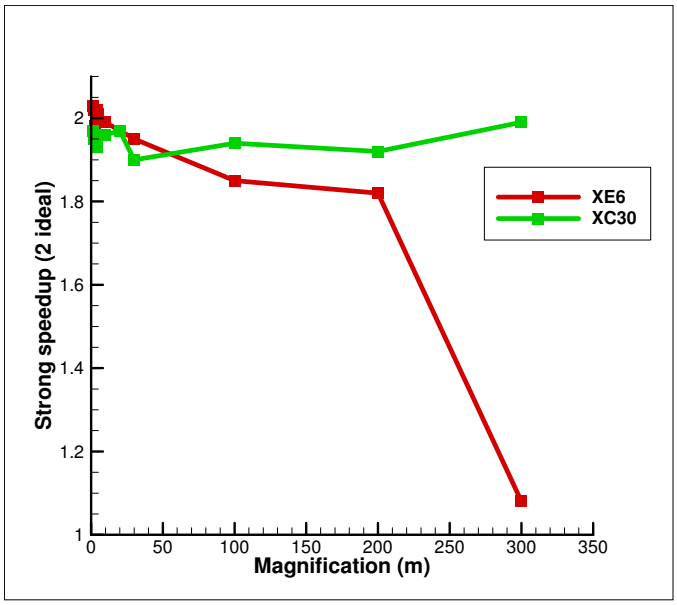

FIG. 6.2. *Strong speedup for twice the original number of processes for different* m *values.*

TABLE 6.7
Time (sec) for the Cray XE6 and XC30 for $m = 20$, *nodes* $= 59703183$, *and elements* $= 116721440$.

Processes (p)	Cray	Time (T)	Strong Scaling	Ideal	Weak Scaling	Ideal
640	XE6 XC30	2276.2 918.5				
..	XE6 XC30	2287.2 968.4				
..	XE6 XC30	2277.4 920.0				
..	XE6 XC30	Avg: 2280.3 Avg: 902.7			$= 0.96$ $\frac{T_{m=1,p=32}}{T_{m=20,p=640}} = 0.96$	1
1280	XE6 XC30	1152.4 477.4				
..	XE6 XC30	1171.7 476.5				
..	XE6 XC30	1153.0 472.6				
..	XE6 XC30	Avg: 1159.0 Avg: 475.5	$= 1.97$ $\frac{T_{m=20,p=640}}{T_{m=20,p=1280}} = 1.97$	2		
1920	XE6 XC30	805.5 323.2				
..	XE6 XC30	812.3 329.3				
..	XE6 XC30	805.8 324.1				
..	XE6 XC30	Avg: 807.9 Avg: 325.5	$= 2.82$ $\frac{T_{m=20,p=640}}{T_{m=20,p=1920}} = 2.87$	3		
5120	XE6 XC30	410.8 141.1				
..	XE6 XC30	402.2 140.1				
..	XE6 XC30	408.7 140.9				
..	XE6 XC30	Avg: 407.2 Avg: 140.7	$= 5.59$ $\frac{T_{m=20,p=640}}{T_{m=20,p=5120}} = 6.65$	8		

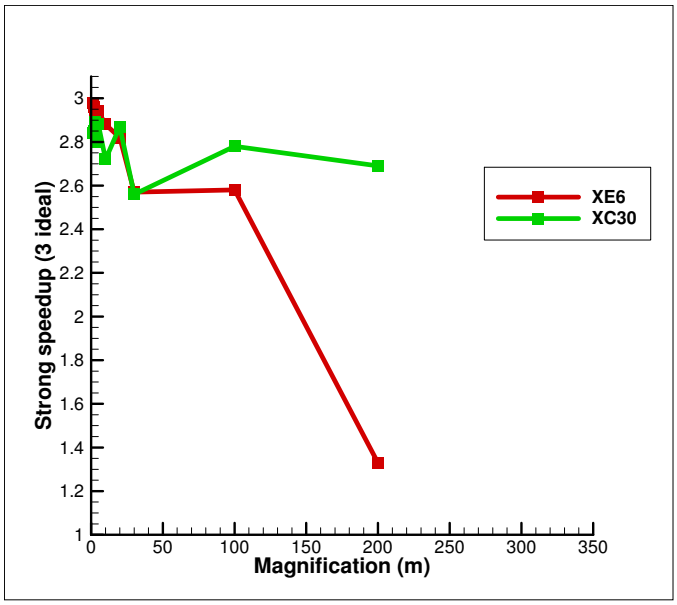

FIG. 6.3. *Strong speedup for three times the original number of processes for different* m *values.*

TABLE 6.8
Time (sec) for the Cray XE6 and XC30 for $m = 30$, *nodes* $= 89537823$, *and elements* $= 175082160$.

Processes (p)	Cray	Time (T)	Strong Scaling	Ideal	Weak Scaling	Ideal
	XE6	2276.5				
960	XC30	924.1				
	XE6	2278.5				
..	XC30	922.7				
	XE6	2277.9				
..	XC30	923.7				
	XE6	Avg: 2277.6			$= 0.96$	
..	XC30	Avg: 923.5			$\frac{T_{m=1,p=32}}{T_{m=30,p=960}} = 0.98$	1
	XE6	1172.5				
1920	XC30	473.0				
	XE6	1165.8				
..	XC30	473.4				
	XE6	1159.7				
..	XC30	514.6				
	XE6	Avg: 1166.0	$= 1.95$			
..	XC30	Avg: 487.0	$\frac{T_{m=30,p=960}}{T_{m=30,p=1920}} = 1.90$	2		
	XE6	889.7				
2880	XC30	348.8				
	XE6	895.0				
..	XC30	364.8				
	XE6	872.4				
..	XC30	370.0				
	XE6	Avg: 885.7	$= 2.57$			
..	XC30	Avg: 361.2	$\frac{T_{m=30,p=960}}{T_{m=30,p=2880}} = 2.56$	3		
	XE6	398.7				
7680	XC30	139.9				
	XE6	429.1				
..	XC30	139.4				
	XE6	447.1				
..	XC30	138.2				
	XE6	Avg: 425.0	$= 5.36$			
..	XC30	Avg: 139.2	$\frac{T_{m=30,p=960}}{T_{m=30,p=7680}} = 6.72$	8		

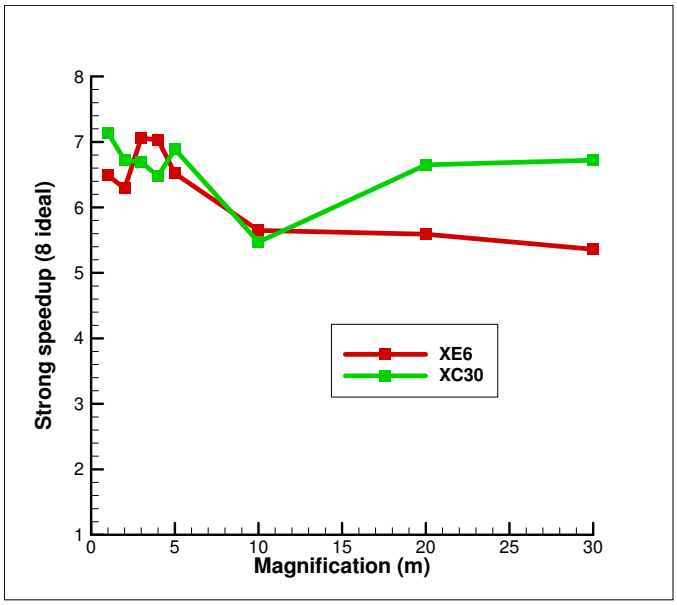

FIG. 6.4. *Strong speedup for eight times the original number of processes for different* m *values.*

TABLE 6.9
Time (sec) for the Cray XE6 and XC30 for $m = 100, nodes = 298380303, and\ elements = 583607200$.

Processes (p)	Cray	Time (T)	Strong Scaling	Ideal	Weak Scaling	Ideal
	XE6	2369.4				
3200	XC30	969.6				
	XE6	2358.4				
..	XC30	934.2				
	XE6	2359.9				
..	XC30	961.3				
	XE6	Avg: 2362.6			$= 0.93$	
..	XC30	Avg: 955.0			$\frac{T_{m=1,p=32}}{T_{m=100,p=3200}} = 0.95$	1
	XE6	1228.4				
6400	XC30	479.2				
	XE6	1357.0				
..	XC30	499.5				
	XE6	1250.3				
..	XC30	498.7				
	XE6	Avg: 1278.6	$= 1.85$			
..	XC30	Avg: 492.5	$\frac{T_{m=100,p=3200}}{T_{m=100,p=6400}} = 1.94$	2		
	XE6	899.8				
9600	XC30	339.2				
	XE6	897.4				
..	XC30	352.4				
	XE6	952.1				
..	XC30	354.1				
	XE6	Avg: 916.4	$= 2.58$			
..	XC30	Avg: 348.6	$\frac{T_{m=100,p=3200}}{T_{m=100,p=9600}} = 2.78$	3		

TABLE 6.10
Time (sec) for the Cray XE6 and XC30 for $m = 200, nodes = 596726703, and\ elements = 1167214400$.

Processes (p)	Cray	Time (T)	Strong Scaling	Ideal	Weak Scaling	Ideal
	XE6	2394.4				
6400	XC30	947.3				
	XE6	2391.1				
..	XC30	943.6				
	XE6	2392.4				
..	XC30	945.3				
	XE6	Avg: 2392.6			$= 0.92$	
..	XC30	Avg: 945.4			$\frac{T_{m=1,p=32}}{T_{m=200,p=6400}} = 0.95$	1
	XE6	1300.0				
12800	XC30	494.8				
	XE6	1292.8				
..	XC30	493.0				
	XE6	1340.4				
..	XC30	488.3				
	XE6	Avg: 1311.1	$= 1.82$			
..	XC30	Avg: 492.0	$\frac{T_{m=200,p=6400}}{T_{m=200,p=12800}} = 1.92$	2		
	XE6	1783.5				
19200	XC30	351.2				
	XE6	1782.0				
..	XC30	350.3				
	XE6	1839.1				
..	XC30	352.6				
	XE6	Avg: 1801.5	$= 1.33$			
..	XC30	Avg: 351.4	$\frac{T_{m=200,p=6400}}{T_{m=200,p=19200}} = 2.69$	3		

TABLE 6.11
Time (sec) for the Cray XE6 and XC30 for $m = 300, nodes = 895073103, and\ elements = 1750821600$.

Processes (p)	Cray	Time (T)	Strong Scaling	Ideal	Weak Scaling	Ideal
	XE6	2671.7				
9600	XC30	1039.2				
	XE6	2442.3				
..	XC30	1041.0				
	XE6	2566.9				
..	XC30	1034.9				
	XE6	Avg: 2560.3			$= 0.86$	
..	XC30	Avg: 1038.4			$\frac{T_{m=1,p=32}}{T_{m=300,p=9600}} = 0.87$	1
	XE6	2351.4				
19200	XC30	512.3				
	XE6	2390.8				
..	XC30	517.1				
	XE6	2392.9				
..	XC30	527.2				
	XE6	Avg: 2378.4	$= 1.08$			
..	XC30	Avg: 522.2	$\frac{T_{m=300,p=9600}}{T_{m=300,p=19200}} = 1.99$	2		

TABLE 6.12
Time (sec) for the Cray XE6 and XC30 for $m = 350, nodes = 1044246303, and\ elements = 2042625200$.

Processes (p)	Cray	Time (T)	Strong Scaling	Ideal	Weak Scaling	Ideal
	XE6	2464.6				
11200	XC30	1041.9				
	XE6	2510.2				
..	XC30	1048.6				
	XE6	2424.8				
..	XC30	1043.1				
	XE6	Avg: 2466.5			$= 0.89$	
..	XC30	Avg: 1044.5			$\frac{T_{m=1,p=32}}{T_{m=350,p=11200}} = 0.86$	1

TABLE 6.13
Ratio of running times for the XE6 and XC30 for values of m.

m	1	2	3	4	5	10
Ratio	2.45	2.46	2.39	2.34	2.45	2.41
m	20	30	100	200	300	350
Ratio	2.58	2.59	2.57	3.44	3.51	2.36

TABLE 7.1
Consistency check comparing values of pressure head from the original mesh and the mesh for $m = 350$ *for the first few nodes and last few nodes of each mesh.*

Node	$m = 1$	Node	$m = 350$
1	129.00000	1	129.00000
2	128.99678	2	128.99678
3	128.99345	3	128.99345
4	119.00000	4	119.00000
5	118.99735	5	118.99735
6	124.23808	6	124.23808
7	118.99464	7	118.99464
8	123.54520	8	123.54520
9	128.98993	9	128.98993
10	110.50000	10	110.50000
6000826	0.00000	1044246298	0.00000
6000827	0.041718481	1044246299	0.041718484
6000828	3.0365778	1044246300	3.0365778
6000829	0.036494874	1044246301	0.036494874
6000830	0.00000	1044246302	0.00000
6000831	0.00000	1044246303	0.00000