A Parallel Implementation of the TOUGH2 Software Package for Large Scale Multiphase Fluid and Heat Flow Simulations

Erik Elmroth   Chris Ding   Yu-Shu Wu   Karsten Pruess
Lawrence Berkeley National Laboratory,
University of California, Berkeley, CA

Abstract

TOUGH2 is a widely used simulation package for solving groundwater flow related problems such as nuclear waste isolation, environmental remediation, and geothermal reservoir engineering. It solves a set of coupled mass and energy balance equations using a finite volume method. The parallel implementation first partitions the unstructured computational domain. For each time step, a set of coupled non-linear equations is solved with Newton iteration. In each Newton step, a Jacobian matrix is calculated and an ill-conditioned non-symmetric linear system is solved in parallel using a preconditioned iterative solver. Communication is required for convergence tests and data exchange across partitioning borders. A real problem with 17,584 blocks and 43,815 connections indicates good scalability properties. From 2 to 128 processors on Cray T3E, the solution time is reduced from 7984 to 126 seconds. Improved parallel performance is expected for larger problems with $10^5 - 10^6$ blocks in a Yucca Mountain nuclear waste site study.

Keywords. Ground water flow, grid partitioning, iterative linear solvers, preconditioners.

1 Introduction

Groundwater flow related problems touch many important areas in today's society, such as nuclear waste isolation, environmental remediation, geothermal reservoir engineering. Because of the complexity of geometry, composition, multiphase, and especially long time scales involved, numerical simulation play vital roles in the solutions of these problems.

This contribution describes the ongoing development of a parallel version of the widely used TOUGH2 software package [4, 5] for numerical simulation of flow and transport in porous and fractured media. The contribution includes our experiences from the software development, descriptions of algorithms and methods developed, and a presentation of the current status of the software, including parallel performance results on Cray T3E-900.

The serial version of TOUGH2 (Transport Of Unsaturated Groundwater and Heat version 2) is now being used by over 150 organizations in more than 20 countries (see [6] for some examples). The major application areas include geothermal reservoir simulation, environmental remediation, and nuclear waste isolation. TOUGH2 is one of the official codes used in the US Department of Energy's civilian nuclear waste management for the evaluation of the Yucca Mountain site as a repository for nuclear wastes. In this context arises the largest and most demanding applications for TOUGH2 so far. LBNL is currently in charge of developing a 3D flow model of the Yucca Mountain site, involving computational grids of 10^5 to 10^6 blocks, and several hundred thousand coupled equations of water and gas flow, heat transfer and radionuclide migration in subsurface [1]. Considerably larger and more difficult applications are anticipated in the near future, with the analysis of solute transport, with ever increasing demands on spatial resolution and a comprehensive description of complex geological, physical and chemical processes.

2 The TOUGH2 Simulation

The TOUGH2 simulation package solves mass and energy balance equations that describe fluid and heat flow in general multicomponent systems. The fundamental balance equations have the following form:

$$\frac{d}{dt}\int_V M^{(k)} dV = \int_S \mathbf{F}^{(k)} \cdot \mathbf{n} dS + \int_V q^{(k)} dV,$$

where the integration is over an arbitrary volume V, which is bounded by the surface S. Here $M^{(k)}$ denotes mass for the k-th component, (water, gas, heat, etc), $\mathbf{F}^{(k)}$ is the flow through the surface, and $q^{(k)}$ is source or sink inside V. This is a general form. All flow and mass parameters can be arbitrary non-linear functions of the primary thermodynamic variables, such as density, pressure, saturation, etc.

Given a computational geometry, space is discretized into many volume blocks. The integral on each block becomes a variable; this leads naturally to the finite volume method, resulting in the following ordinary differential equations:

$$\frac{dM_n^{(k)}}{dt} = \frac{1}{V_n}\sum_m A_{nm} F_{nm}^{(k)} + q_n^{(k)},$$

where V_n is the volume of the block n, and A_{nm} is the interface area bordering between blocks n, m and F_{nm} is the flow between them. Note that flow terms usually contain spatial derivatives, which are replaced by simple difference between variables defined on blocks n, m and divided by the distances between the block centers. See Figure 1 for an illustration.

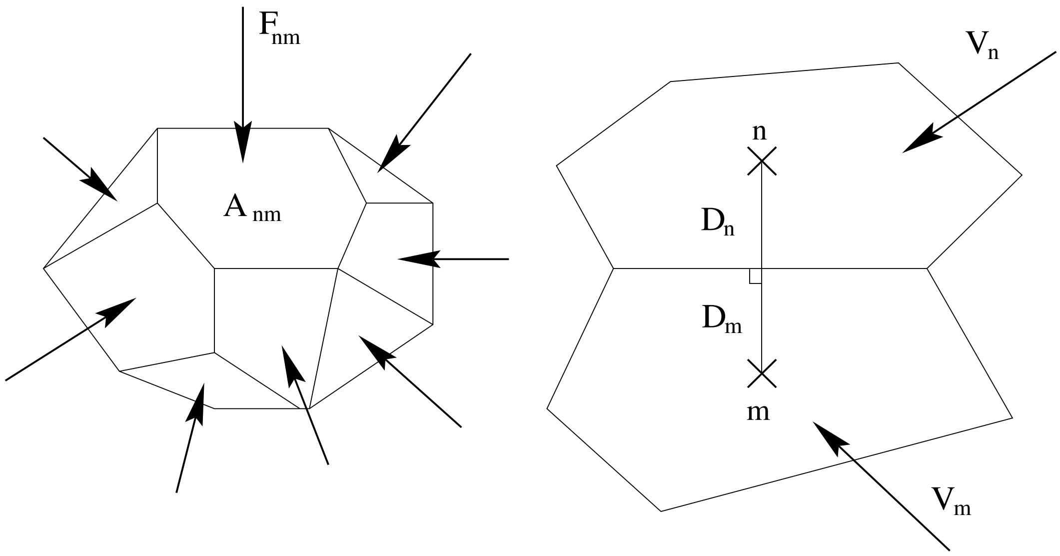

Figure 1: Space discretization and geometry data.

Time is discretized as a first order difference equation. Flow and source/sink terms on the right hand side are evaluated at $t + \Delta t$ for numerical stability for the multi-phase problems. This lead to coupled nonlinear algebraic equations.

3 Computational Procedure

The main solution procedures can be schematically outlined as the following loops:

```
Initialization and setup
Time step advance
     Newton iteration
          Calculate the Jacobian matrix
          Linear equation solver
     end do
end do
output
```

After reading data and setting up the problem, the time consuming parts are the three main loops (time step, Newton iteration, and the iterative linear solver). At each time step, the nonlinear discretized coupled algebraic equation is solved with the Newton method. Within each Newton iteration, the Jacobian matrix is first calculated by numerical differentiation. The implicit linear equation is then solved using a sparse linear solver with preconditioning. After several Newton iterations, the convergence is checked by a control parameter, which measures the residual in the Newton iterations. If the Newton step converges, the time will advance one more time step, and the process repeats until the pre-defined total time is reached.

If the Newton procedure does not converge after a preset max-Newton-iteration, the current time step is reduced (usually by half) and the Newton procedure is tried for the reduced time step. If converged, the time will advance; otherwise, time step is further reduced and another round of Newton iteration follows. These procedure are repeated until convergence in the Newton step is reached.

The system of linear equation is usually very ill-conditioned, and requires very robust solvers. The dynamically adjusted time step size is the key to overcome the combination of possible convergence problems for the Newton iteration and the linear solver. For this highly dynamic system, the trajectory is very sensitive to variations in the convergence parameters.

Computationally, the major part (about 65%) of the execution time is spent on solving the linear systems, and the second major part (about 30%) is the assembly of the Jacobian matrix.

4 Designing the Parallel Implementation

The aim of this work is to develop a parallel prototype of TOUGH2, and to demonstrate its ability to efficiently solve problems significantly larger than problems that have previously been solved using the serial version of the software. The problems should be larger both in the number of blocks and the number of equations per block. The target computer system for this prototype version of the parallel TOUGH2 is the 696 processor Cray T3E-900 at NERSC, Lawrence Berkeley National Laboratory.

In the following sections, we give an overview of the design of the parallel algorithm and its implementation.

4.1 Grid Partitioning and Grid Block Reordering

Given a finite domain as described in Section 2, we will in the following consider the dual mesh (or grid), obtained by representing each *block* (or volume element) by its centroid and by representing the interfaces between blocks by *connections*. (The nomenclature blocks and connections is used in consistency with the original TOUGH2 documentation [5].) The physical properties for blocks and their interfaces are represented by data associated with the blocks and connections, respectively.

In TOUGH2 the computational domain is defined by the set of all connections given as input data. From this information, an adjacency matrix is constructed, i.e., a matrix with a non-zero entry for each element (i, j) where there is a connection between blocks i and j. In the current implementation the value 1 is always used for non-zero elements, but different weights may be used. The adjacency matrix is stored in a compressed row format, called CSR format, which is a slight modification of the Harwell-Boeing format.

The actual partitioning of the grid into p almost equal-size parts is performed using a multilevel p-way partitioning algorithm, implemented in the METIS software package [3]. The partitioning algorithm is designed to minimize the number of edges cut.

After partitioning the grid on the processors, the blocks (or more specifically, the vector elements and matrix rows associated with the blocks) are reordered by each processor to a local ordering. The blocks for which a processor computes the results are denoted the *update* set of that processor. The update set can be further partitioned into the *internal* set and the *border* set. The border set consists of blocks with an edge to a block assigned to another processor and the internal set consists of all other blocks in the update set. Blocks not included in the update set but needed (read only) during the computations defines the *external* set.

Figure 2 illustrates how the blocks can be distributed over the processors. (The vertices of the graph represent blocks and the edges represent connections.) Table 1 shows how the blocks are classified in the update and the external sets and how the update sets are further divided in internal and border sets.

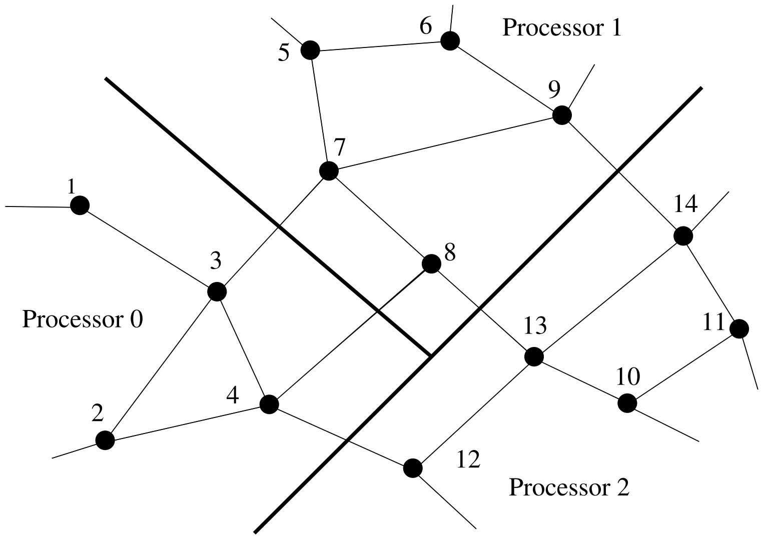

Figure 2: A grid partitioning on 3 processors.

In order to facilitate the communication of elements corresponding to border/external blocks, the local renumbering of the nodes is made in a particular way. All blocks in the update set precede the blocks in the external set, and in the update set, all internal blocks precede the border blocks. Finally, the external blocks are ordered internally with blocks assigned to a specific processor placed consecutively. One possible ordering is given as an example in Table 1.

The consecutive ordering of external blocks that reside on each processor makes it possible to receive data for these blocks into appropriate vectors without use of buffers and with no need for further reordering, provided that the sending processor has access to the ordering information. However, it is not possible in general to order the border blocks so that transformations can be avoided when sending, basically because some blocks in the border set may have to be sent to more than one processor.

4.2 Computing the Jacobian Matrix

In the parallel algorithm, each processor is responsible for computing the rows of the Jacobian matrix that correspond to blocks in the processor's *update* set. All derivatives are computed numerically.

The Jacobian matrix is stored in the Distributed Variable Block Row format (DVBR) [2]. All matrix blocks are

Table 1: Example of block distribution on the Update, Internal, Border, and External sets, using global indexing and an appropriate choice of local reordering.

	Update set		External set
	Internal	Border	
Processor 0	1, 2	3, 4	7, 8, 12
Local ordering	*1, 2*	*3, 4*	*5, 6, 7*
Processor 1	5, 6	7, 8, 9	3, 4, 13, 14
Local ordering	*1, 2*	*3, 4, 5*	*6, 7, 8, 9*
Processor 2	10, 11	12, 13, 14	4, 8, 9
Local ordering	*1, 2*	*3, 4, 5*	*6, 7, 8*

stored row wise, with the diagonal blocks stored first in each block row. The scalar elements of each matrix block are stored in column major order. The use of dense matrix blocks enable use of dense linear algebra software, e.g., optimized level 2 (and level 3) BLAS for subproblems. The DVBR format also allows a variable number of equations per block.

Computation of the elements in the Jacobian matrix is basically performed in two phases. The first phase consists of computations relating to individual blocks. At the beginning of this phase, each processor already holds the information necessary to perform these calculations. The second phase include all computations relating to interface quantities, i.e., calculations using variables corresponding to pairs of blocks. Before performing these computations, exchange of relevant variables is required. For a number of variables, each processor sends elements corresponding to *border* blocks to appropriate processors, and it receives elements corresponding to *external* blocks.

4.3 Solving the Linear Systems

The non-symmetric linear systems to be solved are generally very ill-conditioned and difficult to solve. Therefor the parallel implementation of TOUGH2 is made so that different iterative solvers and preconditioners easily can be tested. So far, most computational experiments have been made using the stabilized bi-conjugate gradient method (BICGSTAB), the squared conjugate gradient method (CGS), and GMRES in the Aztec software package [2], together with a number of different preconditioning techniques.

As an illustration of the difficulties arising in these linear systems, we would like to mention a very small problem from the Yucca Mountain simulations mentioned in the Introduction. This non-symmetric problem includes 45 blocks, 3 equations per block, and 64 connections. When solving the linear system, the Jacobian matrix is of size 135×135 with 1557 non-zero elements. For the first Jacobian generated, i.e., the matrix involved in the first linear system to be solved, the largest and smallest singular values are 2.48×10^{32} and 2.27×10^{-12}, respectively, giving the condition number 1.1×10^{44}

By applying block Jacobi scaling, where each block row is multiplied by the inverse of its 3×3 diagonal block, the condition number is significantly reduced. The scaling reduces the largest singular value to 7.69×10^{3} and the smallest is increased to 9.83×10^{-5}, altogether reducing the condition number to 7.8×10^{7}. This is, however, still an ill-conditioned problem. Therefor, a non-overlapping Additive Schwarz procedure with incomplete *LU* factorization is applied after the block Jacobi scaling. The Additive Schwarz procedure has shown to be absolutely vital for convergence on problems that are significantly larger.

5 Performance Analysis

The parallel algorithms have been implemented in Fortran using MPI communication primitives. The code has been verified correct against the serial code.

Performance tests for a real application problem have been performed. The problem consists of 17,584 blocks, 3 components per block giving 3 equations per block, and 43,815 connections between blocks. The Jacobian matrix in the linear systems to be solved for each Newton step is of size $52,752 \times 52,752$ with 946,926 non-zero elements. This application normally requires a simulation of 10^4 to 10^5 simulated years, requiring a significant execution time also with good parallel performance and a large number of processors. In order to investigate the parallel performance, we have therefore limited the simulation time to 10 years, which still is a significant simulation. A much shorter simulation will of course give the initialization phase unproportionally large impact on the performance figures. This phase is therefore excluded from the timings.

In this set of performance tests the linear systems have been solved using the BICGSTAB iterative solver with block Jacobi scaling followed by a non-overlapping Additive Schwarz procedure with incomplete *LU* factorization. The stopping criteria used for the linear solver is $\frac{\|r\|_2}{\|b\|_2} \leq 10^{-4}$, where r and b denote the residual and the right hand side, respectively.

Performance results have been obtained on the 696 processor Cray T3E-900 at NERSC, Lawrence Berkeley National Laboratory. Execution times in seconds and parallel speedup are presented for 2, 4, 8, 16, 32, 64, and 128 processors in Figure 3 and Figure 4, respectively.

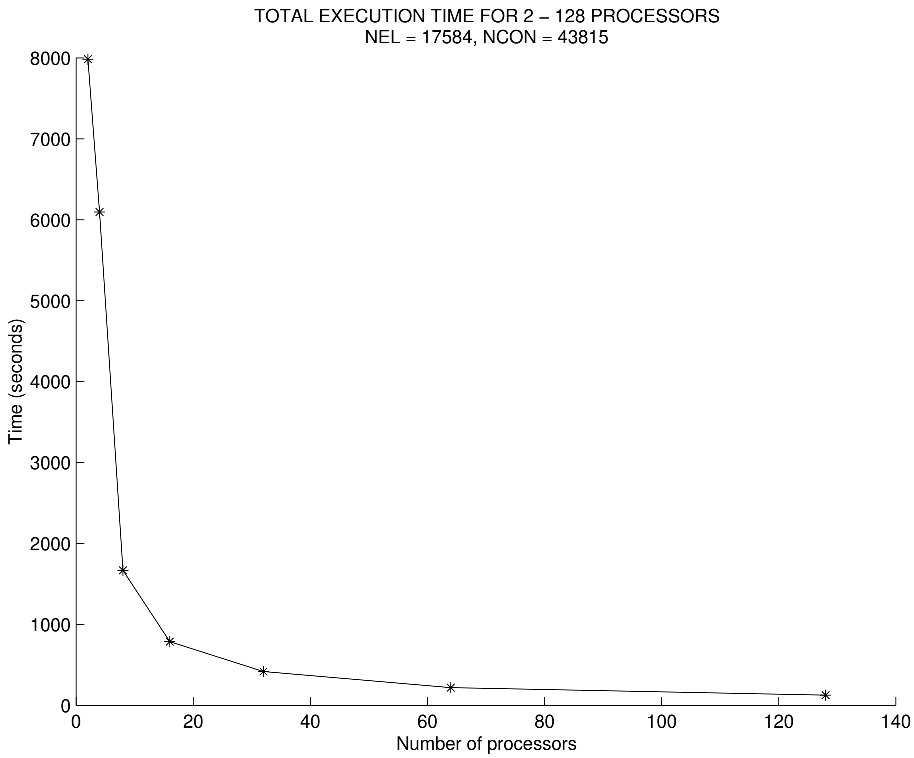

Figure 3: Execution time for 2, 4, 8, 16, 32, 64, and 128 processors on the Cray T3E-900.

Since the problem cannot be solved on one processor with the parallel code the speedup is normalized to be 2 on two processors, i.e., the speedup on p processors is calculated as $2T_2/T_p$, where T_2 and T_p denote the wall clock execution time on 2 and p processors, respectively.

The results clearly demonstrate good parallel performance. We actually observe super-linear speedup for some of the tests. We also observe a significantly lower speedup than expected on 4 processors. Both these phenomena can be explained by differences in the total amount of work performed, and in order to conduct this study, we give some further details in Table 2.

The table shows the average number of Newton iterations per time step, and the average number of iterations in

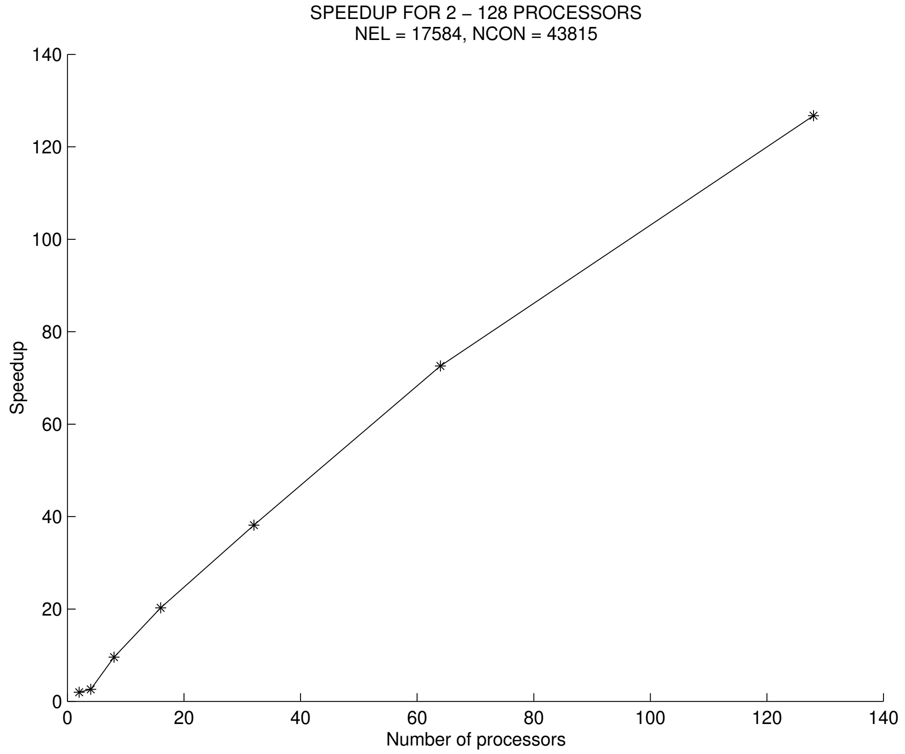

Figure 4: Parallel speedup for 2 to 128 processors.

Table 2: Summary of iterations counts and execution times for varying number of processors.

	2	4	8	16	32	64	128
#Time steps	104	129	104	104	104	104	94
#Newton iterations/Time step	6.54	7.98	6.43	6.35	6.37	6.30	6.60
Total #Newton iterations	680	1030	669	660	662	655	620
#Lin. solv. it./Newton step	14.36	23.09	16.34	16.42	17.37	18.23	19.18
#Lin. solv. it./Time step	93.9	184.4	105.1	104.2	114.6	114.8	126.5
Total #Lin. solv. iterations	9762	23781	10934	10838	11914	11943	11894
Time spent in Lin. solv. (secs)	5521	4224	1040	472	256	137	85
Time spent on other (secs)	2463	1873	629	317	162	83	41
Total execution time (secs)	7984	6097	1669	789	418	220	126

the linear solver per time step and per Newton step, as well as the total number of time steps, Newton iterations, and iterations in the linear solver. We recall that the linear system solve is the most time consuming operation and the computation of the Jacobian matrix is the second largest time consumer. Both of these operations are done once for each Newton step.

We note that some variations in the time discretization occur when the the problem is solved on different number of processors. Similar behavior have been observed, for example, when using different linear solvers in the serial version of TOUGH2. The variations in time discretization leads to variations both in the number of time steps needed and the number of Newton iterations required. Notably, the 4 processors execution requires 24% more time steps, 51% more Newton steps, and 144% more iterations in the linear solver compared to the execution on 2 processors. This increase of work fully explains the low speedup on 4 processors.

However, the figures in Table 2 do not explain the super-linear speedup observed. For example, the speedup on 8 processors is 19.6% larger than maximum expected (i.e., 9.57 vs. 8.00), but compared to two processors, the 8 processor execution requires only 1.6% less Newton iterations, and the total number of iterations in the linear solver is actually larger. When doubling the number of processors from 8 to 16, we see another factor of 2.20 in speedup, even though the reduction in number of Newton iterations and iterations in the linear solver is moderate (1.3% and 0.9%, respectively).

A breakup of the speedup in one part for the linear solver and one part for all other computations (mainly assembly of the Jacobian matrix) is presented in Figure 5. The figure shows that the super-linear behavior is a result of super-linear performance of the linear solver.

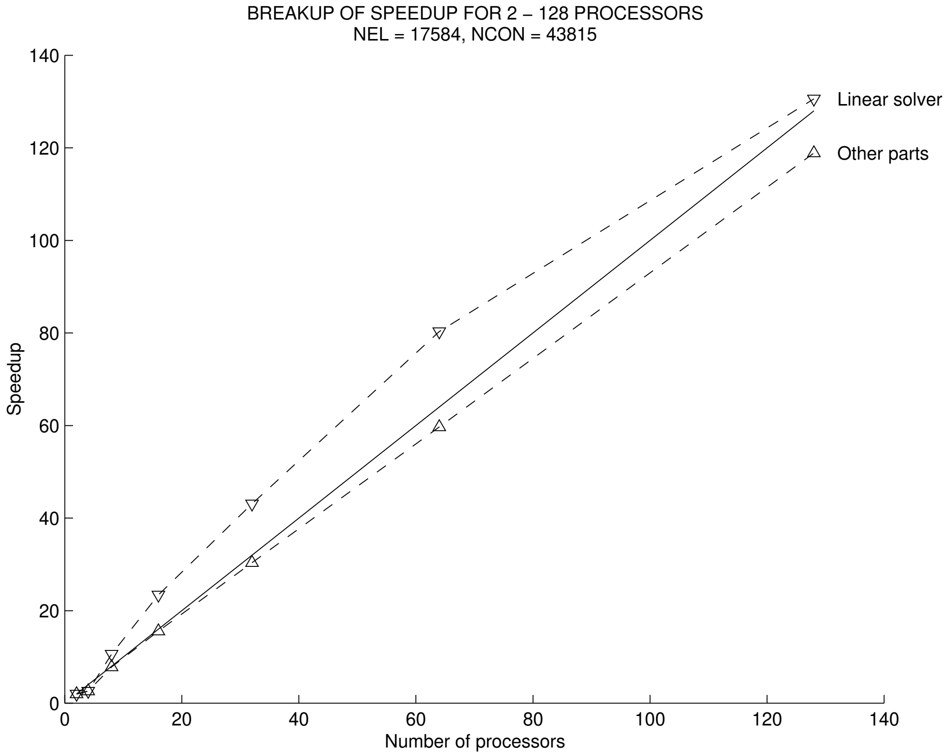

Figure 5: Breakup of speedup in one part for the linear solver (marked “∇”) and one part for all other computations (marked “$\triangle$”). The ideal speedup is defined by the straight line.

Note that the results presented are for the total time spent on these parts, i.e., a different number of linear systems to be solved or a difference in the number of iterations required to solve a linear system affect these numbers. The speedup of the “other parts” is close to p for all tests, and this is also an indication that this part of the computation may show good performance also for larger number of processors. The speedup of the linear solver is larger than what would normally be expected from 2 to 8 and from 8 to 16 processors. The speedup from 16 to 32, from 32 to 64, and from 64 to 128 processors is less than two.

The latter behavior is expected for several reasons. One is the obvious increased communication to computation ratio as the number of processor increases and each processor's subproblem becomes smaller. Another reason is the increased number of iterations due to less efficient preconditioning as the individual domains in the Additive Schwarz procedure becomes smaller. The best effect is expected when the whole matrix is used in the incomplete *LU* factorization, but in order to achieve good parallel performance, the size for the preconditioning operation on each processor is restricted to its local domain (but again, at the cost of decreased effect of the preconditioner). In average, the matrix used in the preconditioning by each processor is n/p, where n is the size of the whole (global) matrix and p is the number of processors.

Another effect of the decreased domains in the Additive Schwarz procedure is that the total amount of work to perform the incomplete *LU* factorizations becomes significantly smaller as the number of processors is increased. Figure 6 gives a further breakup of the speedup, now with the speedup for the linear solver separated into one part for the preconditioner and one for the other parts of the linear solver.

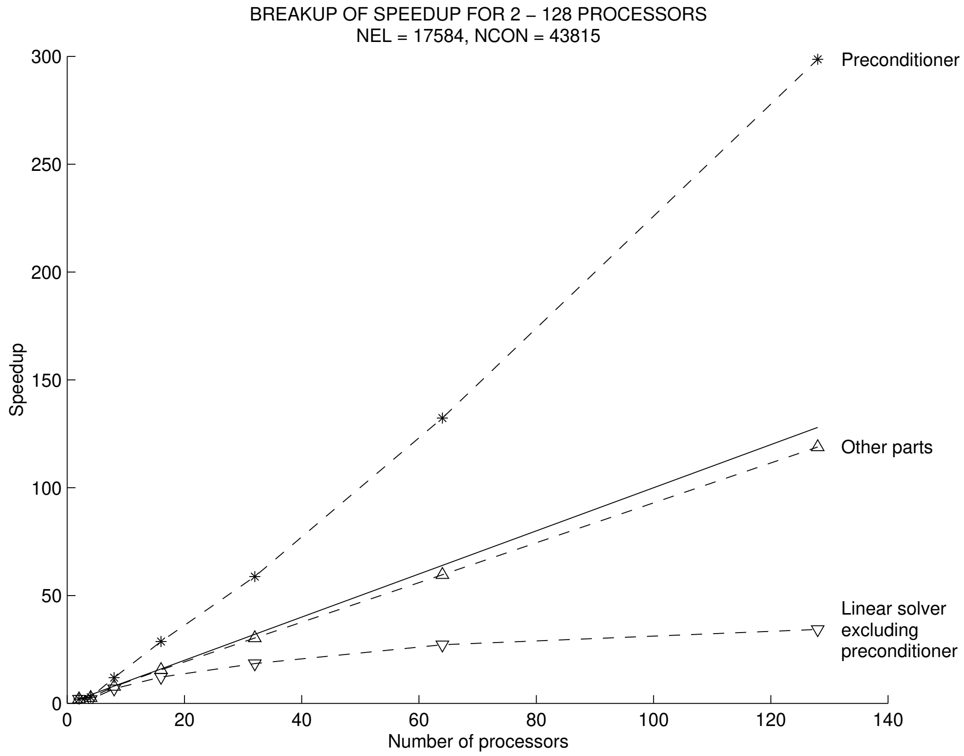

Figure 6: Breakup of speedup in one part for the linear solver excluding Additive Schwarz preconditioning procedure (marked "∇"), one part for the Additive Schwarz preconditioning procedure (marked "*") and one part for all other computations (marked "$\triangle$"). The ideal speedup is defined by the straight line.

The super-linear speedup for the preconditioner follows from the decreased domains for incomplete *LU* factorization in the Additive Schwarz procedure. As the number of processors increases, the amount of time spent on preconditioning become small compared to the time spent on iterations, whereas the relations is the opposite for 2 processors. Table 3 shows how much time is spent on preconditioning in percentage of the total time spent in the linear solver.

On the other hand, the effect of the preconditioner is better on large domains, illustrated by the increased number of iterations in the linear solver for increasing number of processors in Table 2, and here partially explaining the low speedup for the linear solver excluding time for preconditioning in Figure 6.

Table 3: Time spent on preconditioning in percentage of total time spent in linear solver.

#Processors	2	4	8	16	32	64	128
Percentage	83.3%	76.2%	73.9%	68.0%	61.0%	50.6%	36.4%

In all, these results illustrate the trade-off between the time spent on preconditioning and the effect of its results. With the objective to minimize the wall clock execution time, we note that smaller domains could be used on 2 and 8 processors. For completeness we also report that the execution time for the original serial code is 8245 seconds.

6 Conclusions

The performance results in Section 5 clearly demonstrate the implementation's ability to efficiently utilize up to 128 processors on the Cray T3E. The problems we are targeting in the near future are larger both in terms of number of blocks and number of equations per block. Moreover should the simulation time be significantly longer. With increased problem size we expect to be able to efficiently use an even larger number of processors, and longer simulations should not directly affect the parallel performance. However, we still have to investigate numerical issues, e.g., convergence properties, both for the linear solvers and the Newton iteration.

7 Acknowledgement

This work is supported by the Director, Office of Science, Office of Laboratory Policy and Infrastructure, of the U.S. Department of Energy under contract number DE-AC03-76SF00098. This research uses resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy.

References

[1] Bodvarsson, G.S., Bandurraga, T.M., and Wu, Y.S. (Eds.) *The Site-Scale Unsaturated Zone Model of Yucca Mountain, Nevada, for the Viability Assessment.* Yucca Mountain Characterization Project Report, LBNL-40376, UC-814, Lawrence Berkeley National Laboratory, Berkeley, CA, 1997.

[2] Hutchinson, S.A., Prevost, L.V., Shadid, J.N., Tong, C., and Tuminaro, R.S. *Aztec User's Guide.* Version 2.0, Massively Parallel Computing Research Center, Sandia National Laboratories, Albuquerque, NM, 1998.

[3] Karypsis, G. and Kumar, V. *METIS. A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices.* Version 4.0, University of Minnesota, Department of Computer Science, 1998.

[4] Pruess, K. *TOUGH User's Guide.* LBNL-29400, UC-251, Earth Sciences Division, Lawrence Berkeley National Laboratory, Berkeley, CA, 1987.

[5] Pruess, K. *TOUGH2—A General-Purpose Numerical Simulator for Multiphase Fluid and Heat Flow.* LBNL-29400, UC-251, Earth Sciences Division, Lawrence Berkeley National Laboratory, Berkeley, CA, 1991.

[6] Pruess, K. (Editor) *Proceedings of the TOUGH Workshop '98.* LBNL-41995, Conf-980559, Earth Sciences Division, Lawrence Berkeley National Laboratory, Berkeley, CA, 1998.