In B. Engquist et al (eds) *Simulation and Visualization on the Grid.*
Springer-Verlag, Lecture Notes in Computational Science and Engineering,
No. 13, 2000, pp 221-233.

On Grid Partitioning for a High Performance Groundwater Simulation Software

Erik Elmroth*

NERSC, Lawrence Berkeley National Laboratory, University of California,
Berkeley, CA.

Abstract. The partitioning of a discretized computational domain is fundamental for solving many application problems on parallel high performance computer systems. This is also the key to work load balancing and minimizing communication overhead. This contribution focus on partitioning of real application problems for the parallel TOUGH2 software. TOUGH2 is a widely used reservoir simulator for solving subsurface flow related problems. It solves a set of coupled mass and energy balance equations using a finite volume method. The application problems considered are from the Yucca Mountain study for a potential repository of civilian nuclear waste. The quality of the partitions produced by three algorithms in the METIS package are analyzed. Performance results are presented for up to 512 processors on Cray T3E-900.

Keywords. Grid partitioning, load balance, communication overhead, groundwater flow.

1 Introduction

TOUGH2 is a widely used reservoir simulator for solving subsurface flow related problems such as nuclear waste geologic isolation, environmental remediation of soil and groundwater contamination, and geothermal reservoir engineering. It solves a set of coupled mass and energy balance equations using a finite volume method.

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 [9] for some examples). It 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. Scientists at Lawrence Berkeley National Laboratory are currently developing a 3D flow model of the Yucca Mountain site, involving computational grids of 10^5 to 10^6 grid blocks, and related coupled

---

* Present address: Department of Computing Science and High Performance Computing Center North (HPC2N), Umeå University, SE-901 86 Umeå, Sweden.

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. High performance capability of the TOUGH2 code is essential for these applications. A parallel version have recently been presented and shown to give very good parallel performance with parallel speedups up to 489 on 512 processors on Cray T3E-900 [2,3].

This contribution focus on the partitioning of the discretized computational geometry. The partitioning is fundamental for the parallel implementation. It is also the key to work load balancing and minimizing the communication overhead.

2 Parallel TOUGH2

During the initialization phase, input data specifying the problem is read, the discretized computational geometry is partitioned among the processors, and the initial data is distributed. This phase is partly serial, but the computations following are completely performed in parallel with communication required for exchanging subdomain border elements and convergence tests.

The time stepping, with dynamically adjusted step length, is performed as an outer loop. For each time step, a non-linear system of equations is solved using Newton's method. For each Newton iteration, a Jacobian matrix is generated by numeric differentiation and a linear system is solved using a preconditioned BICGSTAB implemented in Aztec [4].

2.1 Grid Partitioning and Grid Block Reordering

Given a computational geometry, space is discretized into many small volume blocks describing a computational mesh or grid. We will in the following consider the dual 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 [8].) 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.

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.

2.2 Partitioning Algorithms

We will analyze the quality of the partitionings produced by the algorithms METIS_PartGraphKway, METIS_PartGraphVKway, and METIS_PartGraphRecursive implemented in the METIS package [5–7]. The algorithms are here named *K-way*, *VK-way*, and *Recursive*, respectively. First, we give brief presentations of the algorithms, which all partitions an irregular graph $G = (V, E)$, where V is the set of vertices and E is the set of edges.

All three algorithms are based on a multi-level scheme. The graph is initially coarsened down to a few hundred vertices by grouping subsets of vertices into multi-nodes. The actual coarsening may be performed using a number of different algorithms, and here this is done using a Sorted Heavy Edge Matching algorithm in all three cases. The coarse graph is then partitioned into k parts. This is done by a traditional k-way algorithm for K-way and VK-way and by a recursive bisection algorithm for Recursive. The final step is the uncoarsening phase in which the multi-nodes are expanded until the original graph is obtained. The partitioning is refined during the uncoarsening phase, in order to take advantage of the additional degrees of freedom obtained by expanding the multi-nodes.

K-way and Recursive minimize the edge cut, i.e., the number of edges that straddle partitions (which is as an approximation of the communication volume). VK-way minimizes the true communication volume. All algorithms, however, are based on heuristics and non of them guarantees to find the global minimum of its object function.

We note that the edge cut is only an approximation of the true communication volume. Since several edges cut may correspond to the same element in the actual communication taking place, the edge cut overestimates the true communication volume. (We count each bidirectional edge cut as 2.) As a consequence, a partitioning that minimizes the edge cut may not minimize the true communication volume, as illustrated in the example in Figure 1.

The partitioning $\{A\ B\ C \mid D\ E\ F\}$ minimizes the edge cut with 6 cut edges, i.e., 3 bidirectional edges, whereas the true communication volume is 5. The total communication volume, is however only 4 for the partitioning $\{A\ B\ D \mid C\ E\ F\}$, even though the edge cut is 8 for that partitioning.

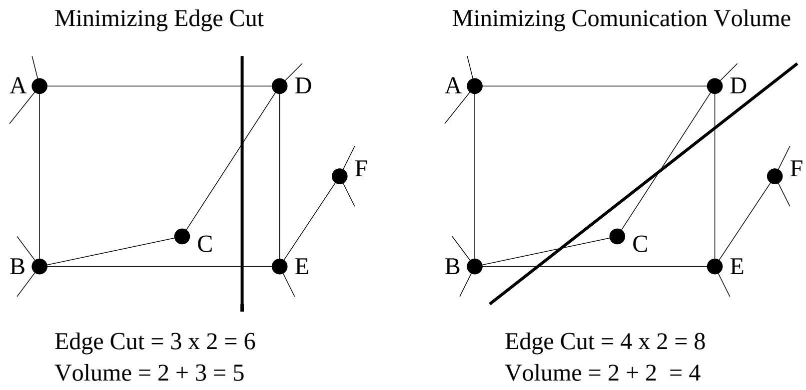


Fig. 1. An example illustrating that a partitioning that minimizes the edge cut does not necessarily minimize the communication volume.

3 Evaluation Criterias

The overall objective for a partitioning algorithm in this application is to divide the computational domain into subdomains for the available processors so that the computations efficiently can be performed in parallel.

So, what characterizes a partitioning that leads to efficient parallel computations? The final answer is the best trade-off when optimizing for a number of objectives, and which is the best compromise is not only problem and algorithm dependent but also dependent on the characteristics of the parallel computer system.

These issues include both load balancing and minimization of the total amount of overhead. Here we analyze computational load balance, communication volume and number of messages to be sent. For each of these, we investigate both the total (or average) load per processor and the difference between the maximum load at any processor and the average, as this may be the processor all other processors will be have to wait for during the computations. In summary, we focus on how well the algorithms perform in the following aspects:

- **Computation load balance:** Minimize the difference between the maximum number of elements for any processor and the average number of elements per processor.
- **Average (or total) communication volume:** Minimize the average number of external elements per processor.
- **Communication volume load balance:** Minimize the difference between the maximum number of external elements for any processor and the average number of external elements per processor.
- **Average (or total) number of messages:** Minimize the average number of neighboring processors per processor.

- **Load balance in number of messages:** Minimize the difference between the maximum number of neighbors for any processor and the average number of neighbors per processor.

The partitioning algorithms considered here are not explicitly designed to minimize all these criterias. They are however considered state-of-the-art algorithms for partitioning irregular graphs and our main interest is to find out how well they are doing in all these aspects.

3.1 Test Problems

Evaluation of the partitionings and the parallel performance of TOUGH2 using these partitionings have been performed for a 2D and a 3D real application problem arising in the Yucca Mountain nuclear waste site study. Results have been obtained for up to 512 processors of the Cray T3E-900 at NERSC, Lawrence Berkeley National Laboratory.

The linear systems have been solved using BICGSTAB with 3×3 Block Jacobi scaling and a domain decomposition based preconditioner with the ILUT incomplete LU factorization. Different levels of overlapping have been tried for this procedure, though all results presented are for non-overlapping tests, which in general have shown to give good performance.

The 2D problem consists of 17,584 blocks, 3 components per block and 43,815 connections between blocks, giving in total 52,752 equations. 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.

The 3D problem consists of 97,976 blocks, 3 components per block and 396,770 connections between blocks, giving in total 293,928 equations. The Jacobian matrix in the linear systems to be solved for each Newton step is of size 293,928 $\times$ 293,928 with 8,023,644 non-zero elements.

4 Grid Partitioning Analysis

Table 1 displays the maximum number of elements in any processor's partition for the three algorithms. We note that Recursive clearly outperforms the other two algorithms in terms of averaging the load for all tests from 2 to 128 processors on the 2D problem and 16 to 512 processors for the 3D problem. Only for 256 processors on the 2D problem is VK-way producing a slightly better partition in this aspect.

The difference between the maximum and average number of elements is illustrated in Figure 2. We also note that VK-way perform slightly better than K-way.

Table 2 shows that VK-way clearly outperforms the other two algorithms in minimizing the maximum and average communication volume in terms of number of external elements required per processor. This supports our discussion about differences between minimizing the edge cut and the true

Table 1. Maximum and average number of elements per processor.

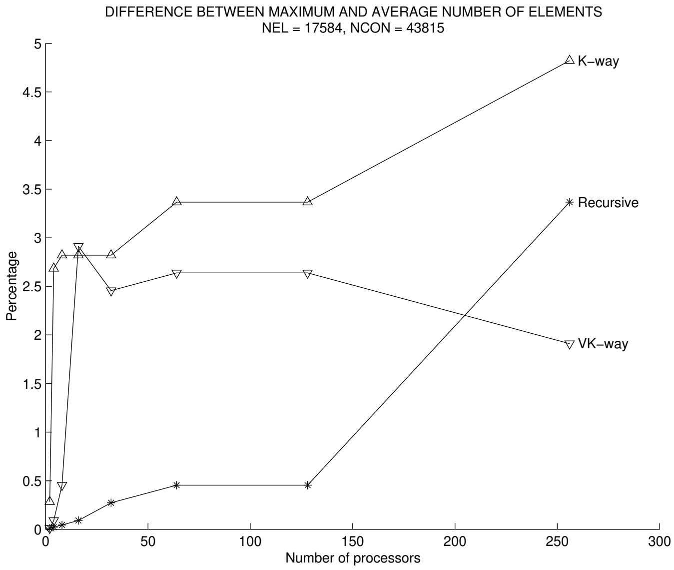
2D PROBLEM

# Processors	Maximum for any processor K-way	VK-way	Recursive	Average per processor All algorithms
2	8817	*8793*	*8793*	8792.00
4	4514	4400	*4397*	4396.00
8	2260	2208	*2199*	2198.00
16	1130	1131	*1100*	1099.00
32	565	563	*551*	549.50
64	284	282	*276*	274.75
128	142	141	*138*	137.38
256	72	*70*	71	68.69

3D PROBLEM

# Processors	Maximum for any processor K-way	VK-way	Recursive	Average per processor All algorithms
16	6308	6295	*6124*	6123.50
32	3154	3153	*3063*	3061.75
64	1579	1576	*1532*	1530.88
128	792	789	*767*	765.44
256	396	395	*384*	382.72
512	198	198	*193*	191.36

2D PROBLEM



3D PROBLEM

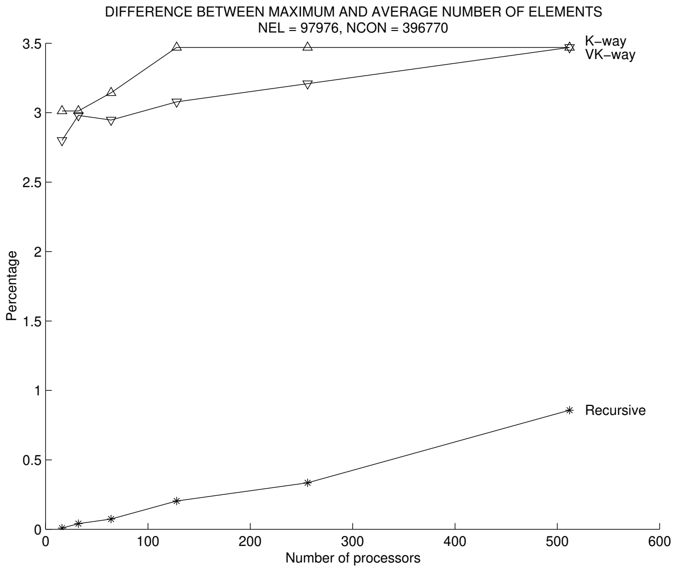


Fig. 2. Computation load imbalance. Difference between maximum and average number of elements per processor (in percentage).

communication volume. K-way appears to give slightly lower communication volume than Recursive.

Table 2. Maximum and average number of external elements per processor.

2D PROBLEM

	Maximum for any processor			*Average per processor*		
# Processors	*K-way*	*VK-way*	*Recursive*	*K-way*	*VK-way*	*Recursive*
2	134	*122*	150	134.00	*121.00*	149.00
4	266	*238*	254	175.25	*162.75*	191.00
8	212	*198*	220	173.75	*153.50*	171.00
16	198	*176*	206	146.88	*126.06*	142.75
32	150	*128*	158	110.62	*95.53*	113.28
64	110	*100*	120	83.20	*71.89*	85.58
128	80	*69*	84	61.02	*52.92*	62.83
256	64	*64*	74	44.65	*40.17*	45.30

3D PROBLEM

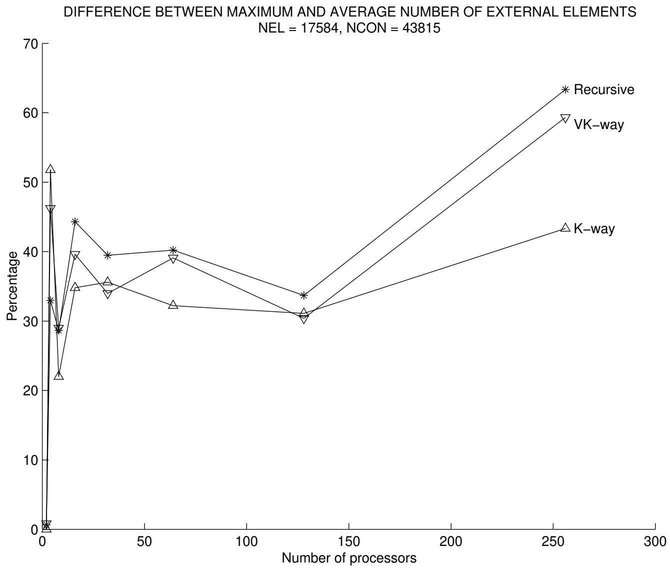
	Maximum for any processor			*Average per processor*		
# Processors	*K-way*	*VK-way*	*Recursive*	*K-way*	*VK-way*	*Recursive*
16	2172	*1762*	2299	1624.50	*1360.94*	1639.25
32	1536	*1265*	1468	1109.78	*949.19*	1106.12
64	1062	*908*	1129	768.69	*664.25*	770.27
128	746	*679*	736	533.48	*455.76*	526.59
256	529	*472*	521	355.00	*308.94*	352.21
512	340	*301*	374	232.89	*206.64*	234.12

The number of external elements required per processor decreases as the number of processors increases. So does, of course, also the number of elements per partition. However, the ratio of the number of external elements to the number of elements per partition increases. We note that the number of external elements required (206.64) actually is larger than the number of elements to be updated (191.36) by the average processor for the 3D problem on 512 processors.

We also observe a significant imbalance in the communication volume between different processors for all three algorithms. The difference in percentage between maximum and average in Figure 3 is between 30% and 65% for large number of processors for all algorithms on both problems.

We remark that the number of external elements do not only affect the communication volume, but also the computation load, as these elements also are used in computations. As the number of elements per partition becomes small and the imbalance between the number of external elements per processor becomes large, this actually becomes a significant factor for load balance.

2D PROBLEM



3D PROBLEM

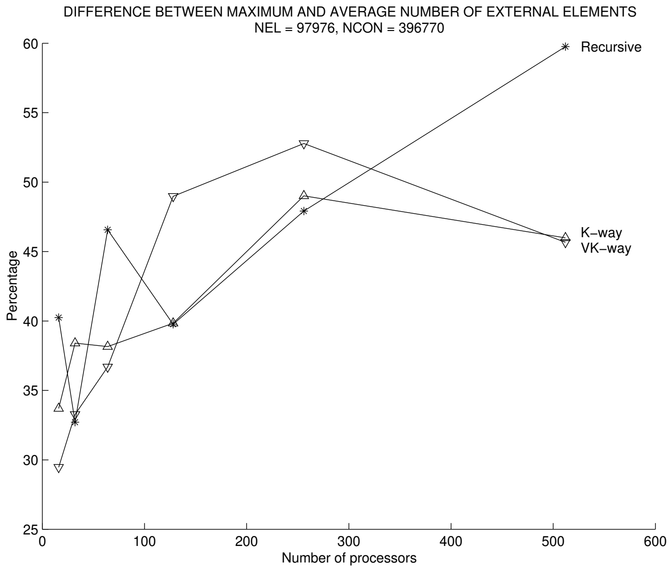


Fig. 3. Communication volume imbalance. Difference between maximum and average number of external elements per processor (in percentage).

We note that the imbalances caused by this is smaller for VK-way than for the other two algorithms, which then may show an advantage for VK-way on large number of processors.

Finally, the number of messages to be sent or received by each processor is proportional to the number of neighboring processors. Hence, the average and maximum number of neighbors per processor should be small. Table 3 shows that the three algorithms produce partitions where the average number of neighbors are roughly the same. The maximum number of neighbors for any processor is, however, slightly lower for K-way for large number of processors.

Table 3. Maximum and average number of neighbors per processor.

2D PROBLEM

	Maximum for any processor			*Average per processor*		
# Processors	*K-way*	*VK-way*	*Recursive*	*K-way*	*VK-way*	*Recursive*
2	*1*	*1*	*1*	*1.00*	*1.00*	*1.00*
4	*2*	*2*	3	*1.50*	*1.50*	2.00
8	*4*	5	*4*	*3.00*	*3.00*	3.25
16	*6*	*6*	*6*	4.00	4.00	*3.88*
32	*7*	*7*	*7*	4.50	4.44	*4.38*
64	*7*	*7*	8	4.94	4.91	*4.78*
128	*7*	8	8	*5.11*	5.19	5.22
256	*7*	9	9	*5.33*	5.36	5.40

3D PROBLEM

	Maximum for any processor			*Average per processor*		
# Processors	*K-way*	*VK-way*	*Recursive*	*K-way*	*VK-way*	*Recursive*
16	*9*	11	11	7.25	7.12	*7.00*
32	*12*	15	13	8.62	*8.31*	8.87
64	*14*	17	19	9.88	*9.47*	10.12
128	*15*	22	23	10.70	*10.58*	11.20
256	*18*	23	24	11.90	*11.65*	11.74
512	*19*	22	25	12.59	*12.21*	12.59

We conclude that none of the algorithms is superior in all aspects analyzed. Recursive gives most even distribution of elements, i.e., computation work load, VK-way gives least communication volume, and K-way gives the smallest number of messages for the "worst case" processor. How these differences affect the parallel performance in practice is analyzed in the following section.

5 Performance Analysis

The two test problems normally require simulations of $10^4 - 10^5$ simulated years. During such long simulations one would normally observe variations in number of time steps, Newton steps, and iterations in the linear solver required when the problem is solved on different number of processors. These issues are further discussed in [2]. In order to analyze the performance differences when using different partitioning algorithms we have therefore kept the simulation fixed to 1 time step, 1 Newton step and 8 iterations in the linear solver. This is indeed a very tiny simulation but still a valid instrument for analyzing the different partitioning as these are the steps that are repeated over and over again when performing longer simulations.

The execution times in seconds for both problems are presented in Table 4.

Table 4. Execution time for one time step, one Newton step and eight iterations in the linear solver.

2D PROBLEM

# Processors	*K-way*	*VK-way*	*Recursive*
2	11.787	11.799	*11.507*
4	5.188	5.103	*5.010*
8	2.441	2.423	*2.378*
16	1.179	1.187	*1.178*
32	0.618	0.613	*0.611*
64	0.357	0.352	*0.347*
128	0.260	0.261	*0.251*
256	0.244	*0.240*	0.255

3D PROBLEM

# Processors	*K-way*	*VK-way*	*Recursive*
16	15.234	16.384	*15.143*
32	5.709	5.991	*5.636*
64	2.541	2.589	*2.525*
128	1.325	*1.300*	1.307
256	0.910	0.886	*0.866*
512	0.834	*0.821*	0.881

The general trend is that Recursive shows best performance for small number of processors and that VK-way gets better as the number of processors increases. This follows naturally from the increasing importance of low communication volume as the number of processors becomes large and the fact that computation load balance is the most critical factor for small number of processors.

A closer look at the results reveals the following. For the 2D problem, the best result is always obtained for the partitioning that gives the best load balance, as in fact the VK-way algorithm actually gives better load balance than Recursive for 256 processors. For the 3D problem, Recursive gives better load balance for all tests, even though the difference becomes smaller for increasing number of processors. This together with increased importance of low communication volume allows VK-way to perform better than Recursive for 512 processors, slightly better on 128 processors and almost as good on 256 processors. The differences on 128 and 256 are to small to be significant but the trend as the number of processor increases is clear.

6 Conclusions

We conclude that the three partitioning algorithms clearly produce partitions that are beneficial from different aspects. Recursive perform the best work load distribution up to a large number of processors, VK-way is best in minimizing the communication volume, and K-way is slightly better than the other two in minimizing the number of messages to be sent during the computations. Which is the best choice depends on the characteristics of the computer system, e.g., communication bandwidth, start-up time for sending a message etc, and the characteristics of the problem, i.e., the number and size of messages to be sent.

For the TOUGH2 software and our two application problems from the Yucca Mountains study on up to 512 processors on Cray T3E we find Recursive to be the best choice up to around 128 processors and VK-way for very large number of processors. However, in these tests we never find very large differences in execution times using different partitioning algorithms, which show that this software and problem combination is not very sensitive to the differences provided by the three algorithms. Finally, we note that all three algorithms produce partitions with large imbalances in number of external elements per processor, i.e., large imbalances in communication volume.

7 Acknowledgement

We thank Chris Ding for helpful discussions and Yu-Shu Wu for providing the test problems.

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. G.S. Bodvarsson, T.M. Bandurraga, and Y.S. Wu. The site-scale unsaturated zone model of Yucca Mountain, Nevada, for the viability assessment. Yucca Mountain Characterization Project Report LBNL-40376, UC-814, Earth Sciences Division, Lawrence Berkeley National Laboratory, Berkeley, CA, 1997.
2. E. Elmroth, C. Ding, and Y.-S. Wu. High performance computations for large scale simulations of subsurface multiphase fluid and heat flow. Report, NERSC, Lawrence Berkeley National Laboratory, Berkeley, CA, 1999. *Submitted to The Journal of Supercomputing.*
3. E. Elmroth, C. Ding, Y.-S. Wu, and K. Pruess. A parallel implementation of the TOUGH2 software package for large scale multiphase fluid and heat flow simulations. In *Proceedings of Supercomputing '99*. ACM, 1999.
4. S.A. Hutchinson, L.V. Prevost, J.N. Shadid, C. Tong, and R.S. Tuminaro. Aztec user's guide. Version 2.0. Technical report, Massively Parallel Computing Research Center, Sandia National Laboratories, Albuquerque, NM, 1998.
5. G. Karypsis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. Report 95-035, Department of Computer Science, University of Minnesota, 1995, Revised 1998. To appear in *SIAM Journal of Scientific Computing*.
6. G. Karypsis and V. Kumar. Multilevel K-way partitioning scheme for irregular graphs. Report 95-064, Department of Computer Science, University of Minnesota, 1995, Revised 1998. To appear in *Journal of Parallel and Distributed Computing*.
7. G. Karypsis and V. Kumar. METIS. A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. Version 4.0. Technical report, Department of Computer Science, University of Minnesota, 1998.
8. K. Pruess. TOUGH2—a general-purpose numerical simulator for multiphase fluid and heat flow. Technical Report LBNL-29400, UC-251, Earth Sciences Division, Lawrence Berkeley National Laboratory, Berkeley, CA, 1991.
9. K. Pruess (editor). *Proceedings of the TOUGH Workshop '98*. Technical Report LBNL-41995, Conf-980559, Earth Sciences Division, Lawrence Berkeley National Laboratory, Berkeley, CA, 1998.