An Integrated Decomposition and Partitioning Approach for Irregular Block-Structured Applications

Jarmo Rantakokko

Dept. of Computer Science and Engineering, University of California, San Diego
9500 Gilman Dr, La Jolla, CA 92093-0114, USA
jarmo@cs.ucsd.edu

Abstract. We present an integrated domain decomposition and data partitioning approach for irregular block-structured methods in scientific computing. We have demonstrated the method for an application related to Ocean modeling. Our approach gives better results than other methods that we have found in the literature. We have compared the domain decomposition part with the Berger-Rigoutsos point clustering algorithm and the partitioning part with a multilevel method used in Metis, a bin packing method, and an inverse space filling curve algorithm. The partitions produced with our approach give lower run times, i.e. higher parallel efficiency, for our application than all the other partitions we have tried. Moreover, the integrated approach gives a possibility to further improve the parallel efficiency compared to doing the decomposition and partitioning in two separate steps.

1 Introduction

Irregular block decompositions are commonly used in scientific applications where partial differential equations are solved numerically. For example, in structured multiblock methods the computational grid is decomposed into blocks and the blocks are fitted around or within an object. The blocks may then be of different sizes and connected to each other in an irregular fashion. Similarly, in structured adaptive mesh refinement techniques we may have irregular regions with high error. The flagged points, i.e. the high error points, are clustered together and a new refined level of grids with an irregular block decomposition is created.

The problem we focus on in this paper originates from Ocean modeling and relates to both application domains discussed above. Here, we have an irregular geometry of water points but we still use a rectangular structured grid covering both land and water. The land points are then masked out in the computations. Still, the inactive points will consume both processor power and memory. It is then necessary to have an irregular block decomposition to cover the active points as efficiently as possible, minimizing the overheads associated with the inactive points. The strategy to use a structured grid and then mask out points to handle irregular boundaries is used, e.g. in HIROMB [12], MICOM [3],

and OCCAM[1] [5], but is not limited to ocean modeling applications. We have seen the same techniques used in oil/ground water flow simulations [15] and in electro-magnetics computations. It is a simple and yet efficient method to handle irregular geometries, especially if it is combined with an irregular block decomposition minimizing the fraction of inactive points in the blocks.

Our emphasis is to create an efficient partitioning and block decomposition method for the irregularly structured applications discussed above. The method should minimize the number of inactive points in the blocks, give a good load balance, a small number of communication dependencies, while keeping the total number of blocks small. This paper presents an integrated block decomposition and partitioning approach and compares it with other techniques found in the literature.

2 The partitioning approach

2.1 Overview

Our algorithm consist of three steps or phases. The idea is to first cluster the water points in "dense" blocks. We strive to get a block efficiency, i.e. the fraction of active points, above a given threshold. The next step is to distribute the blocks onto the processors with a small load imbalance ratio and a low number of inter-processor dependencies. The final step is to try to merge blocks on the same processor into larger rectangular blocks. The emphasis with the last step is to reduce the total number of blocks. There is a small cost associated with each block, e.g. starting up loops, calling functions, and updating the block boundaries. The two first steps will be explained in detail below.

2.2 The clustering algorithm

We want to cluster the active points in rectangular blocks with a high block efficiency. To begin with, we create a minimal bounding box surrounding all active points. We split this block recursively and create new bounding boxes around the resulting blocks until the block efficiency is above a given threshold.

In each recursion step, we choose only to split those blocks with a fraction of active points below a given threshold. This threshold can either be chosen as a fixed rate or a relative rate. Fixed rate means that we choose blocks with a fraction of active points below $X\%$ (X is a constant). Relative rate means that we choose blocks that has relatively many inactive points compared to the other blocks, i.e. we split only those blocks with the most inactive points. For example, we can choose to split all blocks with $Y\%$ more inactive points than in average.

In the splitting phase, we have a search window around the center of the block and do the split across the longest side where we cut through the least number of active points. In other words, we cut along the grid line, within the window, with the smallest signature S_i. The signature S_i for grid line i is defined

[1] OCCAM can also handle unstructured partition shapes.

as the sum of the active points along the line (surface in 3D), i.e. $S_i = \sum_j f_{ij}$ where $f_{ij} = 1$ for water points and $f_{ij} = 0$ for land points. See Figure 1. The purpose of the window is twofold: it limits the search space and avoids cuts into too thin blocks.

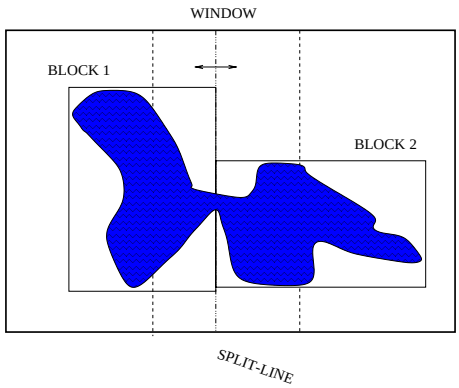


Fig. 1. We define a search window around the center and find the line that cuts through the least number of active points. Then, we set up minimal bounding boxes around the resulting blocks and repeat the algorithm recursively.

2.3 The distribution method

The second phase in our algorithm is to distribute the blocks onto the processors. We treat this in terms of a graph partitioning problem. The blocks constitute the vertices and the data dependencies between the blocks constitute the edges. We choose the vertex-weights proportional to the workload on the blocks and the edge-weights proportional to the communication volume between the corresponding blocks.

The Recursive Spectral Bisection method (RSB) [14] uses a heuristic to minimize the global edge-cut in the partitioning of the graph. The partitions can still be improved in the fine details with a local refinement method, e.g. Greedy, Kernighan-Lin [4]. We have implemented the RSB-method combined with a Greedy-like local refinement method [13]. In each bisection step we sort the vertices according to the Fiedler vector into two parts with approximately equal workloads. Then, we improve this assignment by swapping vertices between the two partitions using the local refinement method. The refinement method exploits gains computed from the edges in the graph. The gain of a vertex v to move from partition A to partition B, (A and B neighbors), is defined as

$$g(v, A, B) = \sum_{w \in B} E_w - \sum_{w \in A} I_w,$$

where E_w (external degree) is the edge-weight between vertex v in partition A and vertex w in partition B. I_w (internal degree) is the edge-weight between vertex v and vertex w, both in partition A.

If the blocks are large or we have a relatively small number of blocks compared to the number of processors, it will be difficult to balance the workload. Our remedy is to further split one block in each bisection step, if necessary. We can not just split any block. The decision which block to split is based on the block sizes and the gains computed above. We take the block that has the highest gain and is large enough to balance the workloads between the two partitions in the bisection step. Large gains give preference to blocks on the partition boundaries.

3 Numerical results

3.1 The applications

We have chosen applications which are related to Ocean modeling. We solve Poisson's equation within irregular geometries covered by the shape of Lake Superior[2], the Baltic Sea[3], and the World's Oceans[4]. Most of the runs are made for the Lake Superior case while the other geometries are only used to complement the results. These applications have a general interest in oceanography even though we solve a 2D equation with a constant forcing function. In Ocean modeling there is a part in the solver (the Barotropic part) where an elliptic equation is solved for all surface points.

We solve the 2D Poisson's Equation with a constant forcing function and with homogeneous Dirichlet boundary conditions.

$$\begin{cases} U_{xx} + U_{yy} = -1 & \text{in } \Omega \\ U = 0 & \text{at } \partial\Omega \end{cases}$$

The equation is discretized with the centered five point finite difference stencil and the corresponding system of linear equations (unsymmetric) is solved with a Krylov subspace method, the Conjugate-Gradient Squared method. Inside the loops, in the computations, we have a condition for each grid point checking if it is land or water. If it is a water point we apply the finite difference stencil otherwise we skip the computations for that grid point. The Baltic sea grid contains initially 25% water points, the Lake Superior grid contains 44%, and the World contains 66% water points.

3.2 Comparison with the Berger-Rigoutsos algorithm

Our first task is to evaluate if the block decomposition in the first step is efficient compared to other related work. To do this, we compare it with the Berger-Rigoutsos point clustering algorithm from structured adaptive mesh refinement

[2] The grid was created from boundary data provided by R. Banks, UCSD
[3] The data was provided by the Swedish Meteorological and Hydrological Institute
[4] The computational grid was extracted from an image provided by B. Huffaker, SDSC

applications [2]. This algorithm is very commonly used and can be considered as a standard benchmark in the area. Another possible candidate for a comparison could be the less complex algorithm by James Quirk [11], but it has no explicit criteria for optimizing the block efficiency.

The Berger-Rigoutsos algorithm is similar to ours, it is only the splitting criteria that differs. Here, the first criterion is to look for zero signatures $S_i = 0$ and make the cut along the most central of these. If all signatures are non-zero then a second criterion is to look for zero-crossings where the Laplacian of the signatures, $\Delta_i = S_{i+1} - 2S_i + S_{i-1}$, changes sign. The best zero-crossing is the one whose magnitude $Z_{i+1/2} = |\Delta_{i+1} - \Delta_i|$ is largest, and this index is chosen as the cutting line. In our algorithm we only have one cutting criterion. We find the smallest signature and make the cut at the corresponding grid line. Our algorithm does not include the arithmetic of computing a second derivative and thus is faster and simpler than the Berger-Rigoutsos method.

Unfortunately, the Berger-Rigoutsos algorithm is not well suited for our applications. Their algorithm is based on edge-detection techniques. The boundary (coast) line is too irregular and this produces only noise in the second derivative of the signature. The cutting lines become randomly placed and the algorithm does not maximize the fraction of active points in the resulting blocks. A modification is then to damp out the high frequency oscillations in the signature. This improves the block efficiency somewhat but increases the complexity of the algorithm considerably. Our clustering algorithm still gives a higher fraction of active points with the same number of blocks, or, for a given fraction of active points we get fewer blocks and a lower run-time.

We have made experiments running the solver on one processor with the initial decomposition produced either with our algorithm or with the Berger-Rigoutsos algorithm (see Figure 2). The differences in the run-times, without the decomposition times, are quite small as the overhead in processing a larger number of blocks (for a given fraction of active points) is small. Note, this overhead depends very much on the solver but also on the grid sizes and the memory hierarchy of the computer. In adaptive mesh refinement applications it is more important to reduce the number of "low-error" points in the blocks as all points are treated equally giving a high cost for these "non-necessary" points.

3.3 Parallel performance

Here, we also include the partitioning phase and compare the parallel run-times from the solver with data partitions produced with other techniques. The simplest possible data decomposition is a regular block-block decomposition of the entire domain, not reducing the number of inactive points. This will give a poor load balance as some blocks will cover more water points than others. A remedy is then to use the Recursive Coordinate Bisection method [1] to get blocks with a balanced workload. But, this introduces an irregular and imbalanced communication pattern increasing the total communication time.

Figure 3 shows the results from partitioning and running the Lake Superior application on IBM SP2 and ASCI Blue Pacific. We get similar results from the

Baltic Sea and the World's Oceans applications. For the Baltic Sea geometry the irregular decomposition gives an improvement of almost 50% (half the run-time) compared to the two other partitions and for the World geometry a $10-15\%$ improvement. The solver runs faster with our irregular domain decomposition. This is mostly due to the reduced number of inactive (land) points. Another observation is that the run-times using the irregular decomposition scale well with the number of processors.

The comparison above shows that reducing the number inactive points gives a better efficiency. To further explore the properties of our algorithm, we have made another set of experiments where we compare our distribution algorithm by replacing it in step 2 with other partitioning methods (step 1 and 3 are the same for all methods). First, we find the block decomposition that gives the best serial run-time (step 1) and then we distribute the blocks with Metis [7], a bin packing method [6], and an inverse space filling curve algorithm [9], respectively (step 2). Finally, we merge blocks within the same partitions (step 3). In Metis we use their recursive multilevel method *pmetis* with default settings. As a bin-packing method we use the *greedy* algorithm suggested in [6] but ignore all neighbor relations. The space filling curve we have chosen uses Morton ordering of the blocks with the center point as a reference. The load balancing is done with the greedy algorithm above but now following the ordering of the blocks.

Our distribution method outperforms all these other methods (see Figure 4). After step 1, we have 130 blocks to distribute to the processors. The granularity is too coarse and all the other methods fail to give a good load balance above 8 processors. We get an almost perfect load balance for all processor configurations due to the extra split in each recursion step. We can even improve our method by decreasing the number of blocks in the initial decomposition, i.e. allowing for a higher fraction of inactive points (*Our** in Figure 4), without sacrificing the load balance. This enhancement is not directly applicable to the other methods as it would be even more difficult to get a good load balance with a smaller number of blocks.

The problem with the other methods is that we will have some very large blocks where there are wide areas of water and this prohibits the algorithms from producing a well balanced workload. An obvious improvement is to split these larger blocks into smaller pieces. We have then included another splitting step in phase one that takes blocks larger than average size and splits them uniformly into four pieces. We have applied this for Metis (labeled *Metis** in Figure 4) and it improves the parallel efficiency considerably, but falls behind our method. This is due to that it makes more splits and introduces more blocks than our approach, which gives a minimal increase in the number of blocks in the second phase to maintain the load balance.

If we want to improve the parallel efficiency even more, as above with *Our**, by trading off the fraction of active points against the number of blocks, we have to search a 2-dimensional parameter space to find the optimal domain decomposition (x-number of splits to reduce the number of inactive points and y-number of splits to even out the block sizes). The merge step, phase 3 where we

Table 1. Parallel run-times IBM SP2, Lake Superior, 16 processors. The different columns represents the number of split steps increasing the fraction of active points. The rows corresponds to the number of steps to split large blocks, making the load balancing easier. The resulting distribution of the blocks is done with Metis. The best run-time corresponds to a parallel efficiency of 72%. The results show that it is difficult with this strategy to find the optimal partitioning, we have to search a 2-dimensional parameter space.

Better Load Balance ↓ / Better Block Efficiency →	5	6	7	8	9	10	11	12
0					3.294	2.706	1.677	1.500
1				1.300	0.770	0.783	**0.643**	0.773
2			1.363	0.754	0.649	**0.660**	0.653	**0.739**
3		0.864	0.732	0.661	**0.597**	0.668	0.701	0.835
4	0.910	0.823	0.646	0.666	0.604	0.681	0.715	0.813
5	0.751	0.678	**0.619**	0.682	0.649	0.739	0.854	
6	0.682	**0.662**	0.650	**0.625**	0.635	0.753		
7	0.711	0.725	0.623	0.740	0.762			
8	0.700	0.671	0.680	0.711				
9	**0.677**	0.682	0.719					
10	0.709	0.669						
11	0.732							

combine small blocks assigned to the same processor, becomes very important for this improvement strategy. The results would otherwise not be competitive at all, the number of blocks would then be too high.

As an example, we have applied this strategy distributing the resulting blocks with Metis onto 16 processors, see Table 1. The table shows that we have to do lots of runs to find the minimal running time, making the partitioning problem very hard. We have managed to increase the efficiency up to 72%, computed for the best run-time in the table. However, our method (*Our**) still gives the best result, with 78% efficiency. We get a better load balance with fewer blocks in our method.

4 Conclusions

We have here developed a new integrated block decomposition and partitioning method for irregularly structured problems arising in Ocean modeling. The approach is not only limited to Ocean modeling but is also suitable for other similar applications, e.g. structured adaptive mesh refinement applications solving time-independent elliptic equations. Here, the parallelism is within a level rather than between the levels, [8]. Partitioning the data of a grid level is then very similar to what we are doing here. It remains to see if our algorithm is competitive for dynamic problems as well. Some preliminary results show that phase 1, the regridding part, dominates the total partitioning and decomposition time. The partitioning time (phase 2) accounts only for a few percent of this.

We have compared our domain decomposition method with the Berger-Rigoutsos grid clustering algorithm and our method gives better results for the applications here. Our approach is simpler and faster but we still get a higher

block efficiency, i.e. a higher fraction of active points in the blocks. We have shown that is not necessary to introduce the complexity of the Berger-Rigoutsos algorithm to get comparable or even better results.

We have also compared the distribution method with other algorithms found in the literature, i.e. Metis, an inverse space filling curve, and a bin-packing method. Our method outperforms the other algorithms. We get a better load balance with fewer blocks and then less serial overhead in the solver, resulting in a shorter parallel run-time. In addition, the integrated decomposition and partitioning approach makes it possible[5] to even more increase the parallel efficiency by allowing for a higher fraction of inactive points but less blocks assigned to each processor as we use larger machine configurations.

It is possible to introduce the combined block-splitting and partitioning ideas with other methods than the recursive spectral bisection method. For example, the space filling curve and the bin-packing methods can easily be modified to include an extra step to split blocks to maintain the load balance. The space filling curve can locally be lengthen from splitting a block. A similar approach has been taken in [10]. For a bin-packing method we can take the highest bin, split one block, and redistribute the parts. Then, repeat this with the currently highest bin until we have a balanced workload. For a pure graph partitioning method, like Metis, this enhancement is not directly applicable. Splitting a node introduces new edges and this information is not available in the initial graph, the graph has to be rebuilt from the new block structure.

Finally, we like to mention that our irregular block decomposition and partitioning algorithm has successfully been used by the Swedish Meteorological and Hydrological Institute to partition the Baltic Sea [16, 17]. As a result, it has been possible to increase the grid resolution in the forecast model from 3 Nautic miles on a Cray C90 down to 1 Nautic mile on a Cray T3E and still solve the problem within the same time limits.

Acknowledgments

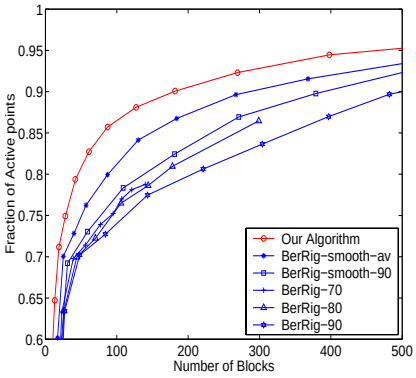
The work in this paper was financially supported by the *Swedish Foundation for International Cooperation in Research and Higher Education (STINT)* and the *US Department of Energy* by Lawrence Livermore National Laboratory under contract W07405-Eng-48, ONR contract N00014-93-1-0152. We would like to thank Scott Baden (UCSD) and Michael Thuné (Uppsala University) for advice on how to improve this paper.

References

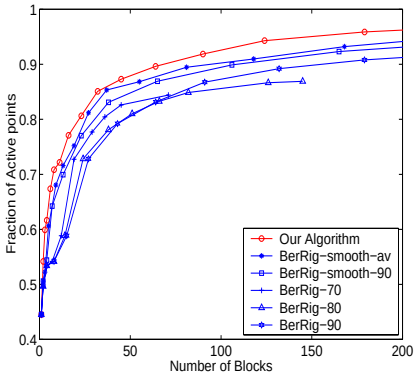
1. M.J. Berger, S. Bokhari, *A partitioning strategy for non-uniform problems on multiprocessors*, ICASE Report No. 85-55, NASA Langely Research Center, Hampton VA, 1985.

[5] In adaptive mesh refinement one may not want to change the initial mesh generation as this could affect the solution making it very difficult to ensure consistent behavior.

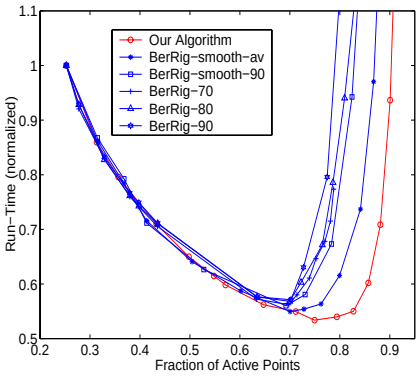
2. M.J. Berger, I. Rigoutsos, *An algorithm for point clustering and grid generation*, IEEE Transactions on Systems, Man and Cybernetics, 21(1991), pp. 1278-1286.
3. R. Bleck, S. Dean, M. O'Keefe, A. Sawdey, *A comparison of data-parallel and message-passing versions of the Miami Isopycnic Coordinate Ocean Model (MICOM)*, Parallel Computing, 21:1695-1720, 1995.
4. C.M. Fiduccia, R.M. Mattheyses, *A Linear-Time Heuristic for Improving Network Partitions*, in Proceedings of 19th IEEE Design Automation Conference, IEEE, pp. 175-181, 1982.
5. G.S. Gwilliam, *The OCCAM Global Ocean Model*, in Proceedings of the sixth ECMWF workshop on the use of parallel processors in meteorology, Reading, UK, November 21-25, 1994.
6. M.A. Iqbal, J.H. Saltz, S.H. Bohkari, *Performance tradeoffs in static and dynamic load balancing strategies*, Technical Report 86-13, ICASE, NASA Langley Research Center, Hampton, VA, 1986.
7. G. Karypis, V. Kumar, *Metis: Unstructured Graph Partitioning and Sparse Matrix Ordering System*, Technical Report, University of Minnesota, Department of Computer Science, Minneapolis, 1995.
8. S. Kohn, *A Parallel Software Infrastructure for Dynamic Block-Irregular Scientific Calculations*, UCSD CSE Dept. Tech. Rep. CS95-429, (Ph.D. Dissertation), Jun. 1995.
9. C. Ou, S. Ranka, *Parallel remapping algorithms for adaptive problems*, Technical Report, Center for Research on Parallel Computation, Rice University, 1994.
10. M. Parashar, J. Brown, *On Partitioning Dynamic Adaptive Grid Hierarchies*, Technical Report, Department of Computer Science, University of Texas, Austin, 1996.
11. J. Quirk, *A parallel adaptive grid algorithm for computational shock hydrodynamics*, Applied Numerical Mathematics, 20:427–453, Elsevier, 1996.
12. J. Rantakokko, *A framework for partitioning structured grids with inhomogeneous workload*, Parallel Algorithms and Applications, Vol 13, pp:135-151, 1998.
13. J. Rantakokko, *Data Partitioning Methods and Parallel Block-Oriented PDE Solvers*, Ph.D. thesis, Department of Scientific Computing, Uppsala University, Sweden, 1998.
14. H.D. Simon, *Partitioning of unstructured problems for parallel processing*, Computing Systems in Engineering, 2:135–148, 1991.
15. M.F. Wheeler, T. Arbogast, S. Bryant, J. Eaton, Q. Lu, M. Peszynska, *A Parallel Multiblock/Multidomain Approach for Reservoir Simulation*, in proceedings of the 1999 SPE Reservoir Simulation Symposium, Houston, Texas, 14-17 February, 1999.
16. T. Wilhelmsson, J. Schüle, J. Rantakokko, L. Funkquist, *Increasing Resolution and Forecast Length with a Parallel Ocean Model*, in proceedings of the Second EuroGOOS International Conference, Rome, Italy, March 11-13, 1999.
17. T. Wilhelmsson, J. Schüle, *Running an Operational Baltic Sea Model on the T3E*, in proceedings of the Fifth European SGI/Cray MPP Workshop, 1999.



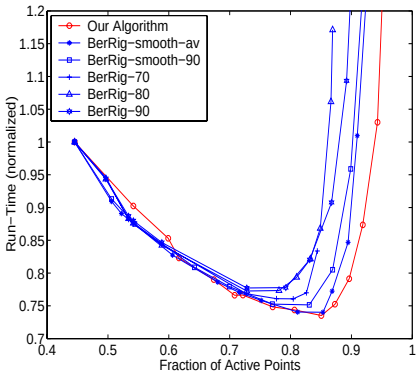
(a) Block decomposition, Baltic Sea



(b) Block decomposition, Lake Superior

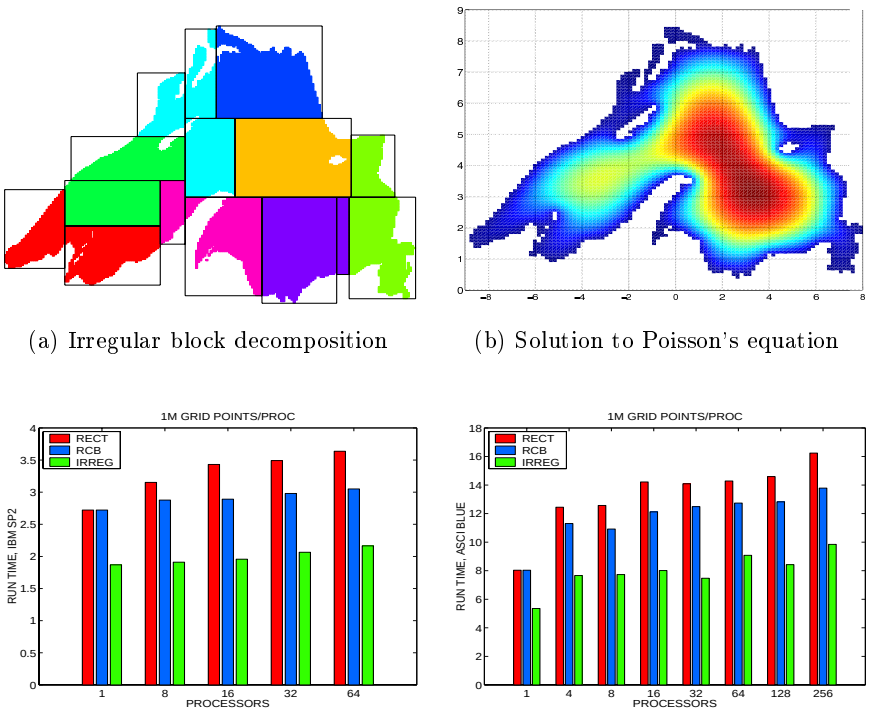


(c) Run times, Baltic Sea



(d) Run times, Lake Superior

Fig. 2. Comparison of the Berger-Rigoutsos algorithm with our algorithm. We have made five different runs with the Berger-Rigoutsos algorithm. Three runs where we only split blocks below a fixed fraction of active points (70%, 80%, and 90%) and two runs with the modified damped version (*smooth*), using a fixed threshold (90%) of active points and a relative threshold of active points (*av*). With the relative threshold we only split those blocks that has more than the average number of inactive points per block in each recursion step. This criterion is also used in our algorithm.



(a) Irregular block decomposition

(b) Solution to Poisson's equation

(c) Run times, IBM SP2

(d) Run times, ASCI Blue Pacific

Fig. 3. Parallel results from the Lake Superior application. (a) An example of an irregular block decomposition produced with our algorithm for 8 processors. The different colors indicate processor assignment. (b) Solution to Poisson's equation with constant forcing function and homogeneous Dirichlet boundary conditions. (c) Parallel run-times on IBM SP2 scaling the problem size linearly with the number of processors. Timings for a fixed number of iterations. We compare three different partitioning strategies, a regular block-block decomposition (*RECT*), the Recursive Coordinate bisection method (*RCB*), and our irregular block decomposition (*IRREG*). (d) The same as (c) but on ASCI Blue Pacific. The irregular block decomposition improves the performance considerably and scales well with the number of processors.

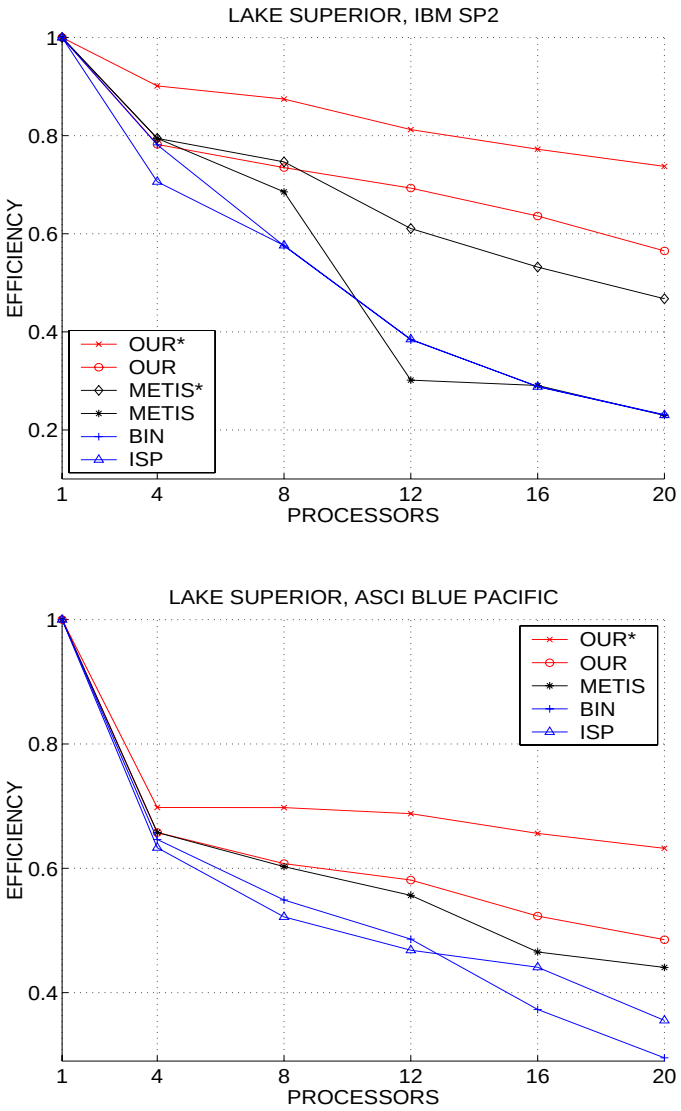


Fig. 4. Parallel Efficiency ($E_p = T_1/pT_p$ where T_1 = serial run time, T_p = parallel run time, and p = number of processors). We solve for the Lake Superior geometry with constant problem size 2000 × 2000 grid points. We take the block decomposition that gives the best run-time T_1 on a single processor and distribute the blocks with *Our* standard method, the *Metis* partitioning software, a *Bin* packing method disregarding neighbor relations, and an inverse space filling curve (*ISP*) method. We also have a second variant of our method, *Our**, where we trade-off the fraction of active points for a smaller number of blocks. The *Metis** method has an extra step to split up large blocks so that the algorithm can produce a better load balance.