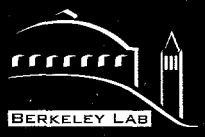
LBNL-40793
UC-405



ERNEST ORLANDO LAWRENCE
BERKELEY NATIONAL LABORATORY

CONF-971138--

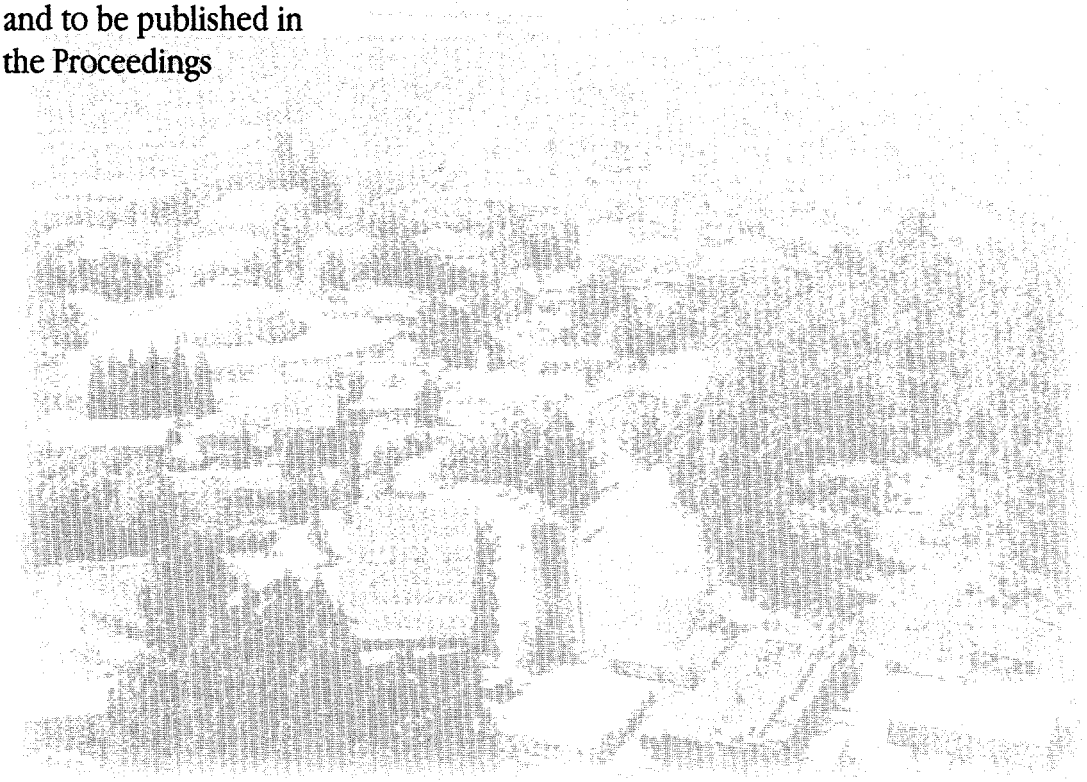
P-HARP: A Parallel Dynamic Spectral Partitioner

Andrew Sohn, Rupak Biswas,
and Horst D. Simon

Computing Sciences Directorate

May 1997

To be presented at
Supercomputing '97,
San Jose, CA,
November 15–21, 1997,
and to be published in
the Proceedings



LBNL-40793
UC-405

P-HARP: A Parallel Dynamic Spectral Partitioner

Andrew Sohn[1], Rupak Biswas[2], and Horst D. Simon[3]

[1]Dept. of Computer and Information Science
New Jersey Institute of Technology
Newark, NJ 07102

[2]MRJ Technology Solutions, MS T27A-1
NASA Ames Research Center
Moffett Field, CA 94035

[3]Computing Sciences Directorate
Ernest Orlando Lawrence Berkeley National Laboratory
University of California
Berkeley, CA 94720

May 1997

This work was supported by the Director, Office of Computational and Technology Research, Division of Mathematical, Information, and Computational Sciences, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098, and in part by the NASA JOVE Program, USRA RIACS, and MRJ Technology Solutions.

P-HARP: A Parallel Dynamic Spectral Partitioner

(Extended Abstract)

Andrew Sohn[1], Rupak Biswas[2], Horst D. Simon[3]

Submitted to Supercomputing'97

May 16, 1997

Abstract

Partitioning unstructured graphs is central to the parallel solution of problems in computational science and engineering. We have introduced earlier the sequential version of an inertial spectral partitioner called HARP which maintains the quality of recursive spectral bisection (RSB) while forming the partitions an order of magnitude faster than RSB. The serial HARP is known to be the fastest spectral partitioner to date, three to four times faster than similar partitioners on a variety of meshes. This paper presents a parallel version of HARP, called P-HARP. Two types of parallelism have been exploited: loop level parallelism and recursive parallelism. P-HARP has been implemented in MPI on the SGI/Cray T3E and the IBM SP2. Experimental results demonstrate that P-HARP can partition a mesh of over 100,000 vertices into 256 partitions in 0.25 seconds on a 64-processor T3E. Experimental results further show that P-HARP can give nearly a 20-fold speedup on 64 processors. These results indicate that graph partitioning is no longer a major bottleneck that hinders the advancement of computational science and engineering for dynamically-changing real-world applications.

---

1. Dept. of Computer and Information Science, New Jersey Institute of Technology, Newark, NJ 07102; sohn@cis.njit.edu; http://www.cs.njit.edu/sohn; This work is supported in part by the NASA JOVE Program, USRA RIACS, and MRJ Technology Solutions.

2. MRJ Technology Solutions, MS T27A-1, NASA Ames Research Center, Moffett Field, CA 94035; rbiswas@nas.nasa.gov; (415) 604-4411, Fax: (415) 604-3957; This work is supported by NASA under Contract Number NAS 2-14303.

3. NERSC, MS 50B-4230, Lawrence Berkeley National Laboratory, Berkeley, CA 94720; simon@nersc.gov. This work was supported by the Director, Office of Computational and Technology Research, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098

1 Introduction

One of the most difficult problems to implement on distributed-memory parallel machines is a problem with a dynamically-changing data structure. This situation is typical for applications which involve grid adaptation, variable-order methods, or multizonal grid technologies. Such problems require repeated load balancing to equidistribute the computational work among the processors, while simultaneously reducing the runtime interprocessor communication. An important aspect of dynamically load-balancing such numerical simulations is the partitioning of the underlying computational mesh. Significant progress has been made over the years in partitioning algorithms for static grids. The ultimate goal for partitioning static grids is to reduce the cutsize of the partitions; however, for dynamically-changing grids, partitioners need to be fast as well. This is important because frequent load balancing is required for adaptive calculations. Some excellent serial partitioning algorithms are now available [5,6,7,8,10,11,13]. Only a few of them have been parallelized to date [1,3,4,9,14], and some preliminary results reported.

In [11], we introduced the sequential version of a new inertial spectral partitioner called HARP. We showed how a particularly successful approach for graph partitioning based on spectral algorithms can be extended to handle the dynamic case. Our goal was to combine the overall effectiveness of the spectral partitioners in terms of reducing the cutsize of the partition, with the speed of recursive inertial bisection. Special techniques were reported that used the dynamic character of the calculation to produce a fast repartitioning of the grid.

This paper describes the algorithmic modifications that have been made for a distributed-memory implementation of HARP[11]. The parallel version of HARP, called P-HARP, is described in Section 2. Some performance results for the sequential HARP code is given in Section 3 for completeness; additional results are given in [11]. Section 4 gives a detailed performance report of P-HARP. The paper concludes with a summary in Section 5.

2 P-HARP Algorithm

The idea behind HARP is to combine the proven *high quality* of recursive spectral bisection and the *fast* computational feature of recursive inertial bisection. The relationships between the number of eigenvectors used and the partition quality as well as the partitioning times have been investigated experimentally. Details and motivations of the original HARP algorithm are described in [11]. The following pseudocode briefly outlines the serial version of HARP:

```
for (i=0; i<log(npart); i++) {          /* npart = total # of partitions */
  for (j=0; j<2^i; j++) {
  1   Find an inertial center of the unpartitioned vertices
  2   Construct an inertial matrix using the inertial vector
  3   Find the eigenvectors of the inertial matrix
  4   Project the vertex coordinates on the dominant inertial direction (eigenvector 0)
  5   Sort the projected coordinates
  6   Divide the unpartitioned vertices into two sets according to the sorted values
  }
}
```

Suppose that the original eigenvectors of the graph are evec[V][N], where N is the number of eigenvectors used and V is the number of vertices. The partitioning algorithm consists of five major steps: inertia, eigen, project, sort, and split. The first step, called inertia, finds the inertial center, center[N], of the unpartitioned vertices in Line 1, fol-

lowed by the computation of an inertial matrix inertia[N][N] in Line 2. Inertial center, center[N], needs N components each of which bears the inertial distance between the vertices and the center. Inertia[N][N] indicates how far the N inertial vectors are away from each other. The second step, called eigen, computes the eigenvectors of the inertial matrix in Line 3. Two EISPACK routines: TRED2 and TQLI are used to compute the eigenvectors. TRED2 reduces a real symmetric matrix to a symmetric tridiagonal matrix using and accumulating orthogonal similarity transformations. TQLI finds the eigenvalues and eigenvectors of a symmetric tridiagonal matrix by the QL method. The third step in Line 4, called project, projects the original eigenvector of each vertex on the dominant direction of the inertial eigenspace. The fourth step, called sort, sorts the vertices based on the inertial value of their coordinates in Line 5. 32-bit and 64-bit floating-point radix sort are used for the SP2 and the T3E, respectively. A radix of eight bits (bucket size of 256) is used in the implementation. The last step in Line 6, called split, divides the sorted vertices into two halves. If an odd numbers of partitions are desired, the vertices can be divided accordingly.

A parallel version of HARP, called P-HARP, has been designed and implemented on two distributed-memory parallel machines: SP2 and T3E. Parallelization of HARP "appears" to be simple and straightforward because HARP is recursive. This recursive algorithmic nature can provide a large amount of natural parallelism which can be potentially utilized by parallel machines. A key misconception of recursion, however, is that it "appears" to be naturally parallel. In reality, it is not highly parallel until each processor is adequately load balanced.

Given P processors and S sets (or partitions), HARP runs log S iterations. In a naive implementation, there is only one task to perform in the first iteration, hence only one processor is busy. This gives a poor processor utilization of $1/P$. At the end of the first iteration, the mesh is divided into two partitions. Each partition is assigned to a processor. Therefore, in the second iteration, two processors independently perform mesh partitioning on their respective submesh. The second iteration thus keeps only two processors busy, resulting in a processor utilization of only $2/P$. The recursion thus needs log P iterations before all processors have some work to perform. What is more problematic by relying on this type of task-level recursive parallelism is:

- the amount of work that has to be performed in the first iteration is the most among all iterations since the original mesh is partitioned into half, and
- only one processor performs this most time-consuming iteration.

It is therefore imperative that the recursive algorithm needs parallelization other than at the task level.

P-HARP exploits parallelism both at the task-level recursion and the function-level loop. When the number of current partitions is smaller than the number of processors P, function level parallelism is exploited to improve performance. When the number of subdomains reaches P, the partitioner turns to task-level recursive parallelism where all processors operate independently of one another. Figure 1 illustrates the operation of P-HARP when eight processors are used. Three of the five modules of P-HARP have been parallelized to date. In iteration 0, all eight processors work together to find the inertial center of the unpartitioned vertices. This step is expensive since it involves all the unpartitioned vertices and their original eigenvectors. Each processor finds an inertial matrix of V/P vertices. At the end of the computation, the P inertial matrices are reduced to a single inertial matrix, that is then broadcast before the next step.

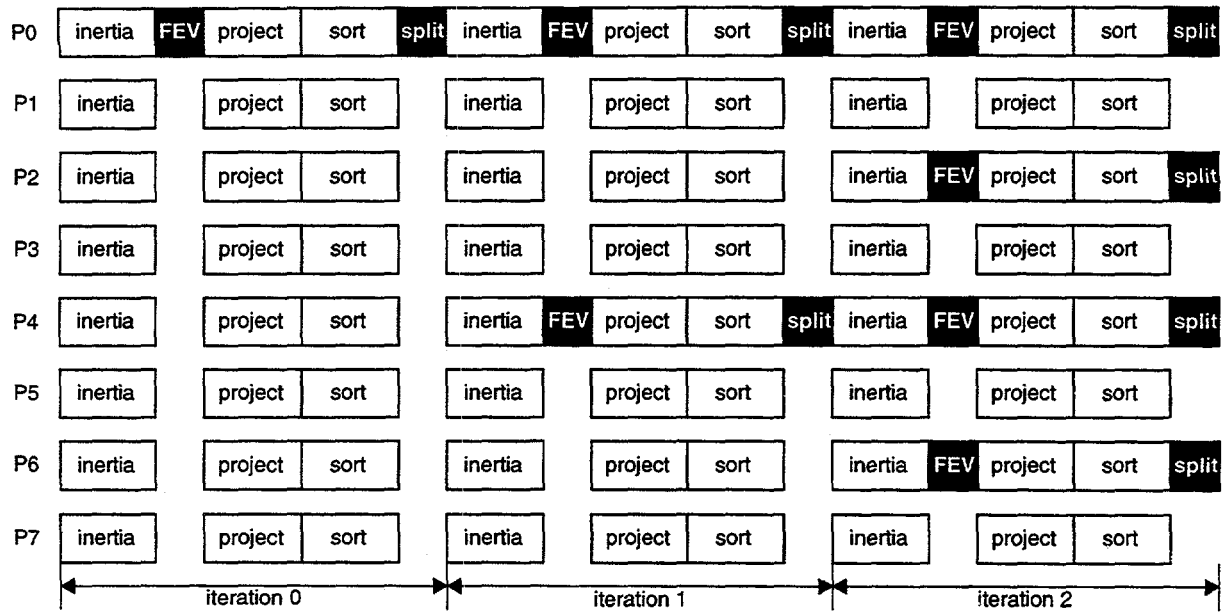


Figure 1: P-HARP running on eight processors. The figure is not drawn to scale in time.

The second step of finding the eigenvectors of the inertial matrix is relatively trivial even for large meshes and is therefore not parallelized. The third step, where the eigen coordinates of the unpartitioned vertices are projected on the major inertial direction is somewhat expensive and has been parallelized. Each processor finds the projected coordinates of V/P vertices. Unlike the first step where P inertial matrices are reduced to a single matrix, here P vectors each of size of V/P are gathered in one processor to form a vector of size V, followed again by broadcasting.

Sorting is the most time-consuming step in HARP. Bitonic sorting is used to parallelize the sort step in P-HARP. In particular, a balanced bitonic sorting is used in the implementation, which always sorts the elements in an ascending order. Bitonic sorting, introduced by Batcher [2], consists of two steps: *local sort* and *merge*. Given P processors and n elements, each processor holds n/P elements. In the local sort step, each processor sorts its elements in either ascending or descending order depending on the second bit of the processor number. The merge step consists of $O(\log^2 P)$ steps. In each merge step, elements are sorted across processors in a pair. As iterations progress, the distance between the pair of processors widens. The last iteration will sort elements on two processors with the distance of $P/2$. While the original bitonic sorting may not be the fastest method on parallel machines, it was selected because of the regularity in communication and the simplicity in implementation. The final step, where the unpartitioned vertices are divided into two sets, requires a negligible amount of time and is thus not parallelized.

3 Performance of Sequential HARP

To verify the performance of HARP, we have performed a substantial number of experiments over the last few years. The Cray T3E installed at NERSC and the IBM SP2 installed at NASA Ames Research Center are used for experimentation. Seven different two- and three-dimensional test meshes are used in this study. They varied in size from 1200 vertices to more than 100,000 vertices. Table 1 shows the characteristics of the test meshes.

	SPIRAL	LABARRE	STRUT	BARTH5	HSCTL	MACH95	FORD2
type, 2D or 3D	2D	2D	3D	2D	3D	3D	3D
# of vertices V	1200	7959	14,504	30,269	31,736	60,968	100,196
# of edges E	3191	22,936	57,387	44,929	142,776	118,527	222,246

Table 1: Characteristics of the seven test meshes used.

SPIRAL is a very small toy grid which is a long chain geometrically arranged in a spiral. This mesh has no computational significance other than to serve as a difficult test case for partitioners. STRUT is a three-dimensional mesh used in civil engineering problems for structural analysis. BARTH5 is a dual graph for a four-element airfoil. HSCTL is a three-dimensional mesh for a high-speed civil transport configuration. MACH95 is a tetrahedral mesh around a helicopter rotor blade. FORD2 is a surface mesh of a model Ford car.

The sequential HARP results are compared with the MeTiS2.0 multilevel partitioner. All HARP results in this section are based on 10 eigenvectors, and are denoted as HARPα. Two parameters are usually used to characterize the performance of graph partitioning algorithms: the number of cut edges C, and the partitioning time T. Tables 2 and 3 show the absolute numbers of edge cuts and the execution times on a single-processor SP2 for all seven test meshes.

# of sets	SPIRAL		LABARRE		STRUT		BARTH5		HSCTL		MACH95		FORD2	
	HARPα	MeTiS2	HARPα	MeTiS2	HARPα	MeTiS2	HARPα	MeTiS2	HARPα	MeTiS2	HARPα	MeTiS2	HARPα	MeTiS2
2	9	9	169	144	82	82	109	86	1484	576	817	815	324	379
4	29	29	423	325	539	528	296	201	1958	1322	1657	1623	911	817
8	67	65	759	530	1027	1005	513	381	3180	2393	3731	3161	1826	1303
16	151	145	1150	864	1970	1939	855	588	5770	4371	5687	4600	3062	2146
32	301	290	1775	1381	3757	3261	1315	985	9652	6970	8664	6128	4732	3203
64	623	589	2667	2132	6879	4947	2012	1561	15896	10306	11557	8467	7561	4928
128	1234	985	4093	3227	8723	7287	3186	2427	22454	15102	15001	10981	11318	7616
256	2156	1526	6140	4806	13263	10551	4954	3672	34980	21857	20954	13966	17425	11332

Table 2: Comparison of the number of cut edges for varying number of partitions.

# of sets	SPIRAL		LABARRE		STRUT		BARTH5		HSCTL		MACH95		FORD2	
	HARPα	MeTiS2	HARPα	MeTiS2	HARPα	MeTiS2	HARPα	MeTiS2	HARPα	MeTiS2	HARPα	MeTiS2	HARPα	MeTiS2
2	0.011	0.02	0.039	0.10	0.066	0.19	0.133	0.28	0.140	0.48	0.264	0.79	0.431	1.18
4	0.012	0.03	0.068	0.22	0.119	0.42	0.249	0.60	0.263	1.00	0.508	1.62	0.831	2.40
8	0.018	0.05	0.103	0.33	0.180	0.65	0.375	0.88	0.394	1.84	0.759	2.42	1.246	3.59
16	0.027	0.11	0.141	0.50	0.243	0.92	0.501	1.21	0.527	2.24	1.014	3.17	1.658	4.78
32	0.040	0.14	0.184	0.70	0.311	1.22	0.635	1.59	0.665	2.93	1.275	4.29	2.090	5.92
64	0.062	0.21	0.236	0.90	0.388	1.65	0.776	2.08	0.813	3.76	1.541	5.46	2.503	7.50
128	0.099	0.28	0.307	1.18	0.485	2.17	0.935	2.70	0.978	4.90	1.832	6.77	2.947	9.23
256	0.169	0.45	0.414	1.56	0.614	2.87	1.132	3.29	1.180	5.97	2.163	8.23	3.427	11.35

Table 3: Comparison of the execution times in seconds on a single-processor SP2.

Figure 2 plots the ratio of HARPα to MeTiS2.0. Figure 2(a) shows that HARPα gives partitions that are of poorer quality than MeTiS2.0. We find that the maximum overall difference is about 60%. It should be noted however that the HARPα results are based on 10 eigenvectors. Increasing the number of eigenvectors will improve partition quality at the expense of increased execution time.

The results shown in Fig. 2(b) indicate that HARPα is, on the average, three times faster than MeTiS2.0. This is precisely the purpose of developing HARP. Since dynamically-changing computations require rapid runtime mesh repartitioning, this fast algorithm is perfectly suitable for our purposes. The fact that the partition quality is somewhat

poor is not a major concern when dealing with adaptive computations. Since adaption and repartitioning has to be performed fairly frequently, it is more important to decrease the partitioning time than reducing the number of cut edges.

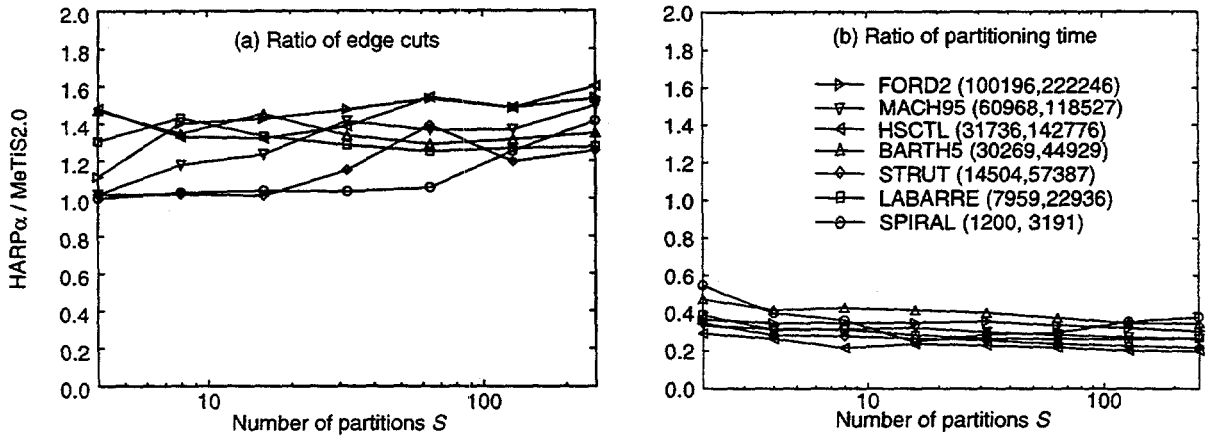


Figure 2: Comparison between HARPα and Metis2.0 on SP2 in terms of edge cuts and execution time.

4 Performance of P-HARP

4.1 Distribution of execution times

Figures 3 and 4 list the distribution of the five individual steps on T3E. Figure 3 shows the time distribution for the MACH95 mesh with 1 and 64 processors. It is clear from the sequential results of Fig. 3(a) that inertia computation and sorting are the major steps. It should be noted that sorting takes approximately half the total execution time. We find almost an identical pattern for the larger FORD2 mesh in Fig. 4. The two steps of eigen solver and split which are not considered for parallelization indeed show negligible computation time. Parallel results on 64-processor T3E are also very similar for the two meshes.

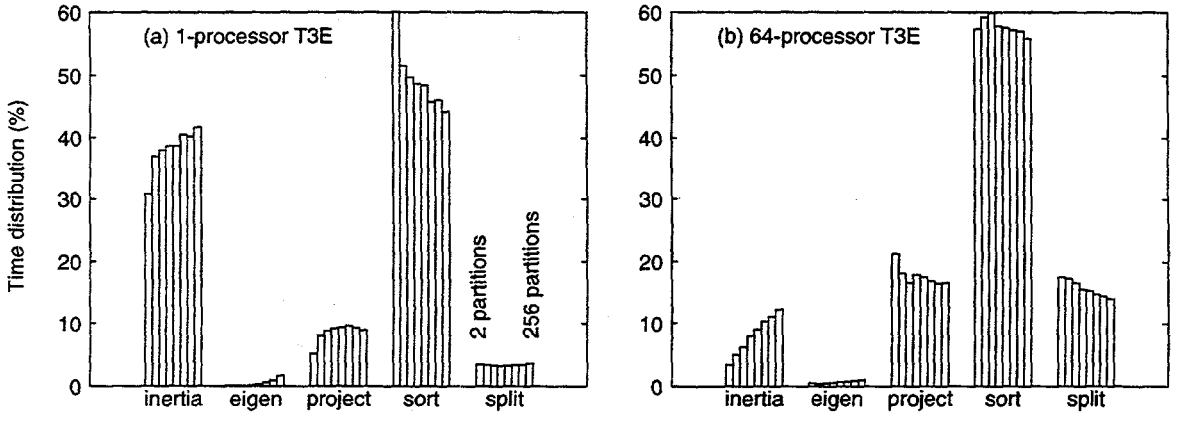


Figure 2: Relative time distribution of MACH95(60968,118527) on T3E before and after parallelization.

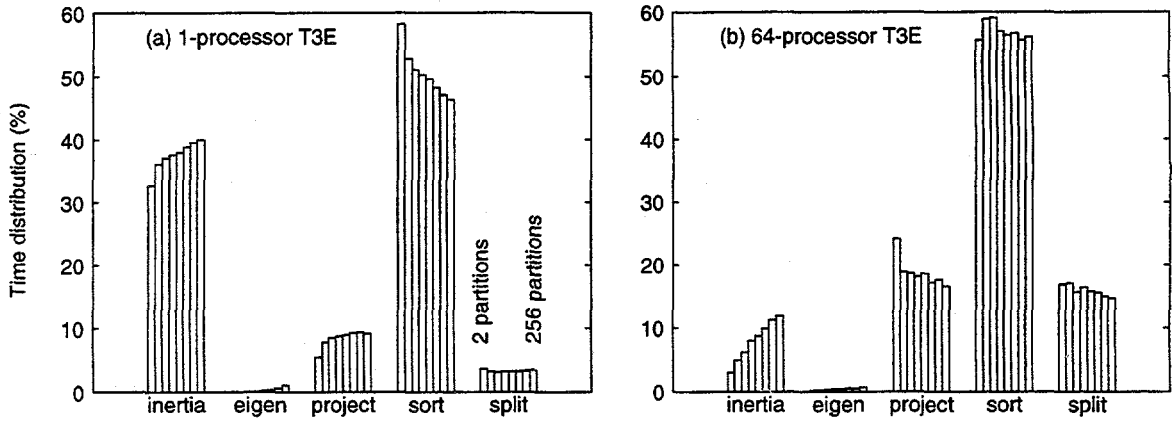


Figure 3: Relative time distribution of FORD2(100196,222246) on T3E before and after parallelization.

(*** ANDREW: I think this section needs some work. What does "to effectively parallelize" mean? If each component is 20% of the total time, is that good? I think good parallelization means each component shows good speedup. Later in the paper, you show 20X speedup for the total time. I think we should show the speedup of each component. Components that show poor speedup have a problem. Percentage of the total time is not the right measure.)

As seen from the plots, the inertia step has been effectively parallelized. In the sequential version, inertia computation required nearly 40% of the total execution time. However, the parallel version essentially eliminated this bottleneck by reducing the time to approximately 10% of the total execution time. It is interesting to observe that the relative time for the project steps has 'effectively' risen to almost 20% from the 10% of the sequential version. Two reasons contributed to this relative rise, which are the computational complexity and communication complexity.

First, the inertia step has *three* nested loops while project has *two* nested loops. The amount of reduction in computation time for three nested loops can be a lot more significant than the one for two nested loops. Second, the communication complexity for inertia is substantially lower than the one for project. The inertia step requires a reduction and broadcasting of a NxN inertial matrix. The project step, on the other hand, needs gathering and broadcasting of V/P vertices. The communication complexity of inertia is fixed to the dimension of the inertial matrix while that of project is variable depending on the size of the mesh. This is precisely why the relative project times for MACH95 shown in Fig. 3(b) is slightly lower than the one for FORD2 shown in Fig. 4(b).

The relative time for split has also risen to approximately 15% from 5%. Again, the reasons are that the step has only a *single* loop and hence is not parallelized. It is natural to see its relative time rising.

The most disturbing result, despite its parallelization, is the sorting step. The relative time of the parallel sort has increased slightly compared to the sequential version. While the overall computation time has been reduced, it clearly indicates that the sorting step has been less effective than other steps. There are several reasons for this increase. Sorting, in general, is highly communication intensive. The typical ratio of communication-to-computation is about 2. Each step again involves some barrier synchronization. Parallel bitonic sorting has $O(\log^2 P)$ complexity. For P=64, it needs 21 (= 1 + 2 + 3 + 4 + 5 + 6) iterations, where each iteration involves all the processors sending $2V/P$ elements (keys and vertices) to each other in pairs. Although the performance of parallel sorting is modest, we have identified the problems and solutions to them can be addressed in the immediate future. One immediate solution is to reduce the communication time by overlapping computation and communication, as reported in [12]. Other solutions to the com-

munication reduction include eliminating $\log^2 P$ barrier synchronizations. We do not foresee any major bottleneck in solving these high communication times.

4.2 Effects of parallelization

Table 4 shows the execution time for MACH95 and FORD2 on T3E with 1 to 64 processors. These results are plotted in Fig. 5. The corresponding SP2 results are shown in Table 5 and plotted in Fig. 6. P-HARP can partition a mesh of over 100,000 vertices into 256 partitions in a quarter of a second on a 64-processor T3E.

# of processors	MACH95								FORD2							
	2	4	8	16	32	64	128	256	2	4	8	16	32	64	128	256
1	0.329	0.681	1.054	1.418	1.794	2.076	2.442	2.797	0.508	1.111	1.710	2.310	2.901	3.440	3.956	4.482
2	0.188	0.373	0.563	0.745	0.917	1.077	1.235	1.411	0.309	0.614	0.909	1.212	1.513	1.789	2.043	2.310
4	0.124	0.232	0.324	0.413	0.499	0.577	0.656	0.743	0.198	0.374	0.523	0.677	0.823	0.962	1.088	1.223
8	0.089	0.155	0.214	0.258	0.299	0.338	0.380	0.422	0.147	0.257	0.349	0.421	0.493	0.558	0.620	0.686
16	0.071	0.119	0.151	0.177	0.200	0.219	0.237	0.261	0.123	0.197	0.254	0.296	0.332	0.360	0.395	0.425
32	0.066	0.105	0.128	0.141	0.156	0.167	0.176	0.186	0.106	0.166	0.206	0.230	0.251	0.265	0.281	0.297
64	0.065	0.099	0.119	0.135	0.142	0.149	0.154	0.157	0.106	0.160	0.204	0.209	0.223	0.231	0.244	0.248

Table 4: Partitioning times on Cray T3E.

# of processors	MACH95								FORD2							
	2	4	8	16	32	64	128	256	2	4	8	16	32	64	128	256
1	0.264	0.508	0.759	1.014	1.275	1.541	1.832	2.163	0.431	0.831	1.246	1.658	2.090	2.503	2.947	3.427
2	0.178	0.294	0.426	0.548	0.677	0.812	0.956	1.123	0.287	0.481	0.688	0.895	1.104	1.318	1.539	1.783
4	0.141	0.208	0.269	0.336	0.400	0.467	0.538	0.622	0.211	0.341	0.440	0.545	0.651	0.757	0.869	0.987
8	0.111	0.165	0.211	0.242	0.275	0.308	0.350	0.388	0.168	0.267	0.331	0.384	0.436	0.490	0.545	0.604
16	0.117	0.174	0.214	0.242	0.254	0.271	0.289	0.310	0.179	0.276	0.336	0.372	0.397	0.422	0.452	0.482
32	0.162	0.139	0.173	0.193	0.208	0.217	0.224	0.234	0.207	0.209	0.260	0.289	0.307	0.322	0.342	0.346
64	0.234	0.139	0.226	0.194	0.209	0.258	0.236	0.259	0.263	0.225	0.261	0.278	0.301	0.340	0.313	0.333

Table 5: Partitioning times on IBM SP2.

Figures 5 and 6 are plotted to illustrate the effectiveness of P-HARP. Figure 5(a) demonstrates that the execution time increases much less than linearly as the number of partitions increases.

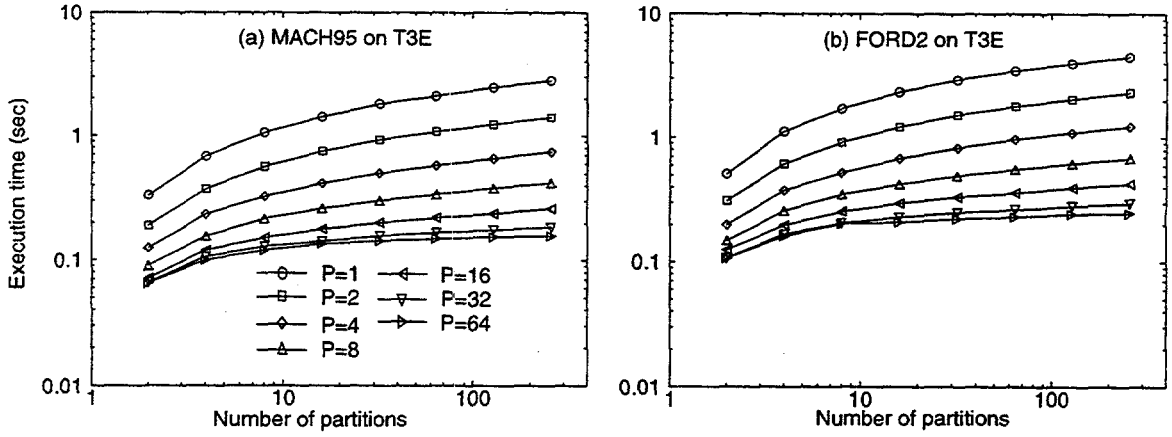


Figure 4: The relation between partitions and the number of processors on T3E. The plots are in log scale.

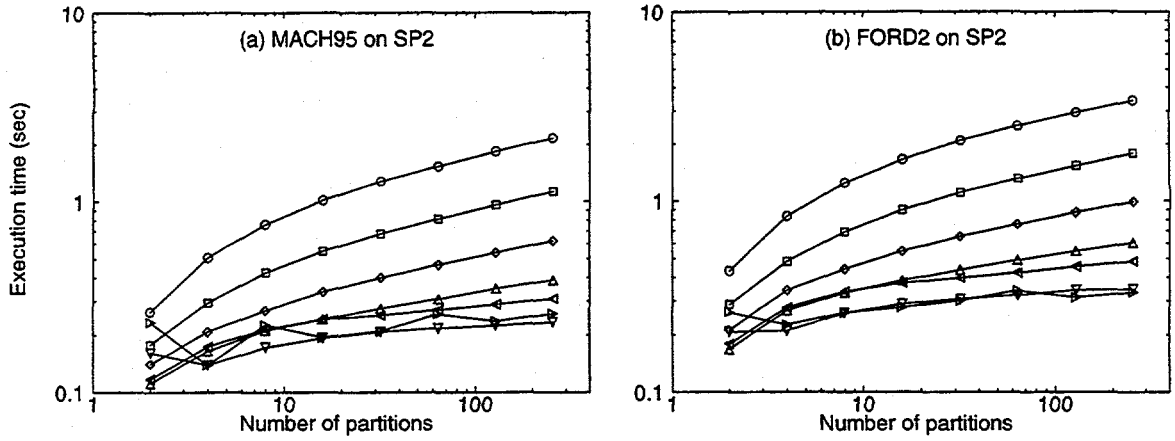


Figure 5: The relation between the number of partitions and the number of processors on SP2.

The T3E results clearly show that the total execution time is not proportional to the number of partitions. For example, consider Fig. 5(a). The bottom curve which shows 64 processors indicates that as the number of partitions is increased from 2 to 256, the execution time has risen only slightly to more than double: 0.065 sec to 0.157 sec. This small increase indicates that even a larger number of partitions will not require substantial computation time. The execution time for FORD2 further confirms this behavior, where the T3E execution time for P=64 has risen to 0.248 sec from 0.106 sec for P=2. A similar pattern is seen for SP2 in Fig. 6. Figure 7 summarizes the results that show negligible increment for a large number of partitions for five of our seven test meshes when using 64 processors.

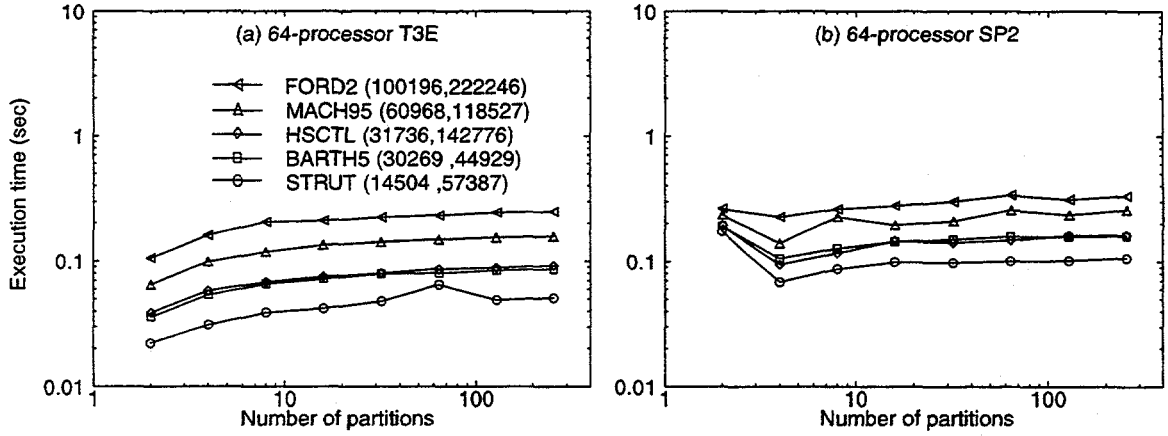


Figure 6: Execution times for 2 to 256 partitions on 64 processors.

4.3 Scalability of P-HARP

Figure 8 shows the parallel speedup of P-HARP. A mere 10-fold speedup is obtained on 64 processors of the SP2. However, the T3E shows almost a 20-fold speedup. The reasons that the T3E gives better scalability than the SP2 are a faster network, a better communication optimization, and an one-sided communication paradigm. The SP2 is a pure message-passing machine while the T3E is a "shared-memory" machine. Message-passing constructs on the SP2 are used at runtime as they are specified in the program. However, the T3E has the capability of shared-memory programming, where two-sided message-passing constructs can be converted to a single one-sided communication. A pair of

MPI_Send and MPI_Recv on the SP2 can be translated to a single PUT or GET construct, where PUT can write data to a remote memory location while GET can read data from a remote memory location. While MPI_Send and MPI_Recv are two-way communication which requires synchronization between the two processors, PUT and GET do not have such a strict synchronization requirement. The only constraint for one-sided communication constructs is that data read/write must be done properly so that program semantics are not violated. It is not clear how well the two-sided communication constructs are translated to one-sided constructs on the T3E. Such a study on compiler behavior/optimization is beyond the scope of this work.

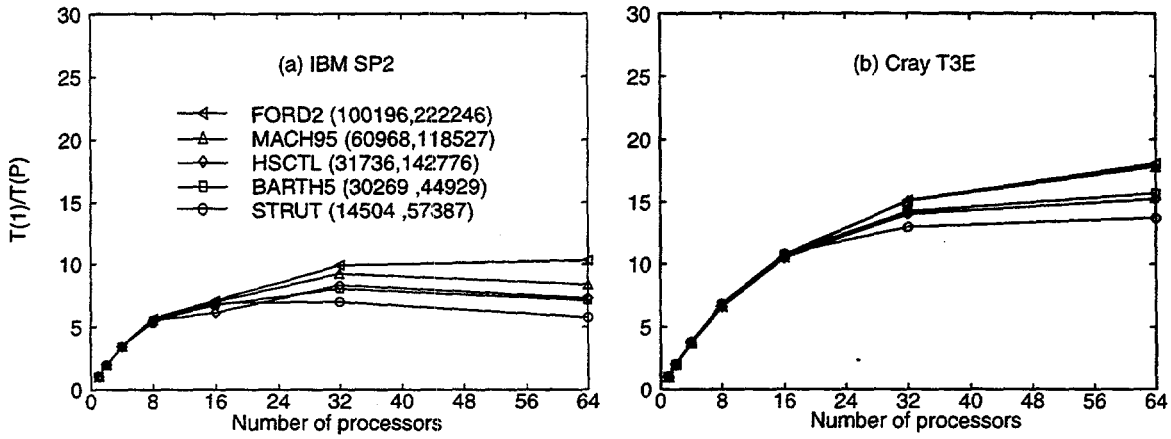


Figure 7: Speedup of P-HARP.

Another reason worth mentioning is that even though both the SP2 and the T3E are capable of overlapping computation and communication, they have different hardware support. The SP2 provides an i860 communication co-processor and a 4KB communication buffer for each Power2 computation processor. The T3E provides 512 external registers (E-registers) along with a stream buffer to take some burden off the computation processor. The SP2 communication co-processor runs at 40 MHz while the T3E communication logic runs at 75 MHz. This doubling of the communication clock along with twice the communication buffer size could effectively give twice the overall communication performance. These communication issues will be addressed in a future version of P-HARP as they are beyond the scope of this report.

The 20-fold speedup is modest in spite of the efforts of exploiting parallelism both at the loop level and at the task level. The main reason for this modest scalability is due to the fact that the current version of bitonic sorting is not crafted for performance. The current implementation is our first attempt to parallelizing the sorting step. Bitonic sorting provides a clean and regular communication pattern compared to other parallel sorting such as radix sort and sample sort. This regularity in communication can be effectively utilized to improve the performance of P-HARP. Several solutions are currently being undertaken, including the elimination of barrier synchronization, and the overlapping of computation with communication. One can argue that performance will not drastically improve since not all the steps are parallelized. In other words, linear speedup will be difficult to obtain because of the Amdahl's law which states that those portions that are not parallelized will eventually limit the overall performance. However, we do not foresee that to be a bottleneck. The overall times for the two steps of eigen solver and split are indeed negligible.

5 Summary

Computational science and engineering problems that utilize dynamic remeshing require runtime mesh partitioning when implemented on distributed-memory multiprocessors. We have presented in this paper a parallel version of the dynamic inertial spectral partitioner HARP. The parallel version, called P-HARP, can quickly partition realistically-sized meshes on a large number of processors while maintaining the partition quality of recursive spectral bisection. To demonstrate the effectiveness of P-HARP, we have selected seven two- and three-dimensional meshes, with the largest containing over 100,000 vertices.

P-HARP exploits parallelism in task-level recursion and function-level loop to overcome the misconception that recursion is highly parallel. When the number of current partitions is smaller than the number of processors P, function-level parallelism is exploited to improve the performance. When the number of subdomains reaches P, the partitioner turns to task-level recursive parallelism where all the processors operate independently of one another. Sequential HARP consists of five major steps: inertia computation, eigenvector solution, projection, sorting, and splitting. The three most time-consuming steps of inertia computation, projection, and sorting have been parallelized. The execution times of the eigen solver and the splitting phase are negligible, hence not parallelized.

P-HARP has been implemented in Message Passing Interface on two distributed-memory multiprocessors: the Cray T3E installed at NERSC of Lawrence Berkeley and the IBM SP2 installed at NASA Ames. Experiments have been performed on up to 64 processors of the two machines. Our largest test mesh with 100,196 vertices can now be partitioned into 256 sets in less than a quarter of second. Results have shown that the total execution time is significantly less than proportional to the number of partitions. As the number of partitions has increased from 2 to 256, the execution times on the 64-processor T3E have risen slightly more than twice, i.e., 0.106 sec to 0.248 sec. A similar pattern has been observed for other meshes and the SP2. All the meshes show that the partitioning times are relatively constant in spite of the increase in the number of partitions when using a large number of processors.

The T3E has given nearly a 20-fold speedup while the SP2 has yielded 10-fold speedup on 64 processors. The reasons that the T3E has given twice the speedup of the SP2 are a faster network and a different communication paradigm. The T3E communication logic runs at 75 MHz while the SP2's i860 communication co-processor runs at 40 MHz. The speedup of 20-fold is modest compared to the efforts expended on parallelization. However, we do not foresee a major problem improving the performance in the immediate future. The results reported in this paper are the first attempt to fully parallelize HARP. We were able to integrate bitonic sorting into HARP. The current implementation has several places where performance can be further improved, including overlapping computation with communication for bitonic sorting and eliminating barrier synchronizations. In the near future, we will address the scalability issue of P-HARP. These parallel partitioning times have indicated that graph partitioning can now be truly embedded in dynamically-changing real-world applications.

References

1. S.T. Barnard and H.D. Simon, "PMRSB: Parallel multilevel recursive spectral bisection," In: *Supercomputing '95*, San Diego, CA, 1995.
2. K. Batcher, "Sorting networks and their applications," in: *Proc. AFIPS Spring Joint Computer Conference*, Reston, VA, 1968, pp. 307-314.
3. P. Diniz, S. Plimpton, B. Hendrickson, and R. Leland, "Parallel algorithms for dynamically partitioning unstructured grids," in: *Proc. 7th SIAM Conference Parallel Processing for Scientific Computing*, San Francisco, CA, 1995.
4. R. Van Driessche and D. Roose, "An improved spectral bisection algorithm and its application to dynamic load balancing," *Parallel Computing*, **21** (1995) 29-48.

5. C. Farhat, "A simple and efficient automatic FEM domain decomposer," *Computers and Structures*, **28** (1988) 579-602.
6. B. Hendrickson and R. Leland, "A multilevel algorithm for partitioning graphs," Report SAND93-1301, Sandia National Laboratories, Albuquerque, NM, 1993.
7. B. Hendrickson and R. Leland, "Multidimensional spectral load balancing," Report SAND93-0074, Sandia National Laboratories, Albuquerque, NM, 1993.
8. G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," Report 95-035, University of Minnesota, Minneapolis, MN, 1995.
9. G. Karypis and V. Kumar, "A coarse-grained parallel formulation of multilevel k-way graph partitioning algorithm," in: *Proc. 8th SIAM Conference on Parallel Processing for Scientific Computing*, Minneapolis, MN, 1997.
10. H.D. Simon, "Partitioning of unstructured problems for parallel processing," *Computing Systems in Engineering*, **2** (1991) 135-148.
11. H.D. Simon, A. Sohn, and R. Biswas, "HARP: A fast spectral partitioner," in: *Proc. 9th ACM Symposium on Parallel Algorithms and Architectures*, Newport, RI, 1997, to appear.
12. A. Sohn and R. Biswas, "Communication studies of DMP and SMP machines," NAS Technical Report NAS-97-005, NASA Ames Research Center, Moffett Field, CA, 1997.
13. C. Walshaw, M. Cross, and M.G. Everett, "A localised algorithm for optimising unstructured mesh partitions," *Intl. J. Supercomputer Appl.*, **9** (1995) 280-295.
14. C. Walshaw, M. Cross, and M.G. Everett, "Dynamic load-balancing for parallel adaptive unstructured meshes," in: *Proc. 8th SIAM Conference on Parallel Processing for Scientific Computing*, Minneapolis, MN, 1997.