

Performance Evaluation of A Seismic Data Analysis Kernel on The KSR Multiprocessors¹

Weiming Gu

College of Computing, Georgia Institute of Technology
Atlanta, Georgia 30332-0280

Abstract

This paper presents the results of performance analysis of a seismic analysis kernel code on the KSR multiprocessors. The purpose of such analysis is to understand the performance behaviors of a class of applications on shared memory parallel machines. The g5 kernel code, commonly used in seismic analysis applications, is parallelized, and its computational and I/O performance is analyzed on a 32-node KSR-1 and a 64-node KSR-2.

1 Introduction

Current trends indicate that massively parallel processors (MPPs) will emerge as the mainstream supercomputers, with current machines including IBM's SP-2, Cray's T3D, Intel's Paragon, Thinking Machine Inc.'s CM5, Kendall Square Research's KSR-2, etc. Manufacturers of these machines have posted impressive performance numbers, some of which even exceed those of much more expensive vector supercomputers. However, these numbers often capture peak machine performance, and can not be achieved by most application programs. In addition, an application may perform very differently on different MPPs depending on their memory architectures and processor/memory interconnects[1].

The topic of this work is the investigation of the effective performance attainable for a specific class of application programs on shared memory supercomputers. Specifically, we are to investigate how seismic data analysis applications behave on the Kendall Square Research Inc.'s KSR multiprocessors. Toward this end, the computational kernel of seismic computation algorithms is parallelized and its performance is analyzed. The so called *g5 kernel* accounts for as much as 95% of total execution time of typical seismic analysis programs. As a result, the performance

¹This work was performed when the author was a summer intern at the Schlumberger Laboratory for Computer Science at Austin, Texas, during the Summer of 1993.

of the *g5 kernel* is a good predictor of the class of applications' performance.

The machine on which the *g5 kernel* code is parallelized and studied is a single ring 32-node KSR-1 machine and a dual-ring 64-node KSR-2 machine available at Georgia Institute of Technology. The KSR machines have an ALLCACHE memory architecture, in which all memory units on all processors are treated as caches at different levels. The machine is time shared, but processors can be dedicated to the execution of a specific application².

This paper is organized into four sections. In the next section, an overview of the KSR system is given, followed by discussions of the *g5 kernel* and its parallelization on KSR. In Section 3, the performance results of the *g5 kernel* and its data input/output on the KSR machines are presented and evaluated³. Conclusions are presented last.

2 The *g5 kernel* and its parallelization on KSR

Seismic data analysis is an important step in oil exploration. Data is first acquired from surface survey. It is then processed and interpreted in order to understand the subsurface structure of a region, and to determine whether there are oil deposits in the surveyed area[2, 3]. Such seismic data processing typically takes months on the fastest computing machines available today, and it involves terabytes of data. The majority of computational cost in seismic data processing is captured by the *g5 kernel*, which is described below after a brief overview of the KSR multiprocessor systems.

²Users cannot dedicate all processors available on a KSR machine to their jobs. The system reserves some processors for handling I/O and other system administration tasks.

³The technical report version (GIT-CC-94-43) of this paper contains additional results and analysis, not available in this paper due to constraints of space. The technical report is available online by FTP at <ftp.cc.gatech.edu>(130.207.9.11) in directory `pubcoctech.reports`.

Overview of the KSR multiprocessors. The KSR multiprocessors (both KSR-1 and KSR-2) are NUMA (non-uniform memory access) shared memory cache-only architectures with an interconnection network that consists of hierarchically interconnected rings. Each KSR node consists of a 64-bit RISC based processor, 32 MB of main memory, a higher performance 512 KB sub-cache (half of which for caching instructions and the other half for caching data), and a ring interface. Peak computational performance is 20 MFLOPS (million floating point operations) for each KSR-1 node and 40 MFLOPS for each KSR-2 node.

The KSR's ALLCACHE memory architecture, is a three-level hierarchy. At the top is each node's high performance cache, called *subcache* in ALLCACHE's terminology. At the second level is each node's main memory, termed *local cache*. The lowest level is the disk storage providing conventional virtual memory. ALLCACHE implements a *sequentially consistent* shared memory model. In this memory model, data moves to the point of reference on demand, and the unit of data movement is always a subpage of 128 bytes. Specifically, if a datum is requested by a processor and found in neither the processor's subcache nor its local cache, a request for the subpage that contains the datum is sent to other processors on the same ring as the requesting processor. If none of the processors on the same ring have the requested subpage, the request message is propagated to other rings. When the subpage is finally brought in to the requesting node, it is replicated in its local cache. A write request to a subpage goes through a similar process, with additional complexity that a write request invalidates all replicated copies of that subpage on other nodes' local caches and subcaches. Therefore, the cost of accessing a shared variable depends on where it is located. In general, a subcache memory access only takes 2 clock cycles, while it takes 10 clock cycles to access a word in a processor's own local cache. Accessing a subpage in other processor's memory takes 175 clock cycles if it is on the same ring as the requesting processor, and about 600 clock cycles if on other rings.

While the KSR machine's shared memory architecture is designed for ease of use by application programmers, programmers still need to carefully distribute both computation and data among participating processors in order to achieve high performance on this machine. The experiences and results from parallelizing the g5 kernel (presented next) demonstrate not only that such careful parallelization is necessary for high performance, but also that it is achievable.

The g5 kernel. The g5 kernel code is the compu-

tational core of many numerical analysis algorithms, such as convolution and finite element methods, commonly used in seismic analysis applications and in many other scientific computation programs. Typically, such algorithms operate on a two or three dimensional computation space, represented by a two or three dimensional array of discrete points. Given some initial values of these points, the algorithm iteratively computes new values for each point from its previous values and from the values of neighboring points. The algorithm terminates when either convergence or divergence is detected. Divergence indicates that the algorithm fails on the problem with the given initial values.

The g5 kernel's computation space is a two dimensional array of points layered on a plane. The specific computation used by the g5 kernel is described by the following formula,

$$\begin{aligned}
 N_{i,j} = & \\
 & c_1 * (P_{i-2,j-2} + P_{i-2,j+2} + P_{i+2,j-2} + P_{i+2,j+2}) + \\
 & c_2 * (P_{i,j-2} + P_{i,j+2} + P_{i-2,j} + P_{i+2,j}) + \\
 & c_3 * (P_{i-1,j-1} + P_{i-1,j+1} + P_{i+1,j-1} + P_{i+1,j+1}) + \\
 & c_4 * (P_{i,j-1} + P_{i,j+1} + P_{i-1,j} + P_{i+1,j}) + \\
 & c_5 * P_{i,j}
 \end{aligned}$$

where $P_{i,j}$ is the old value of the point at row i and column j from the previous iteration, and $N_{i,j}$ holds the new value to be calculated. Constants c_1 , c_2 , c_3 , c_4 , and c_5 are weights used in the calculations. For boundary points, the values of their neighbors outside the plane boundary are always assumed to be 0. All coefficients are real numbers and all point values are complex numbers (represented by two floating point words). From the formula, the calculation for each point per iteration requires 21 operations of complex and real number additions or multiplications, resulting in 42 actual floating point operations. For a problem size of an $1K \times 1K$ ($1,024 \times 1,024$) plane, the total computation required for each iteration of the g5 kernel code is 42 million floating point operations.

A straightforward implementation of the g5 kernel algorithm requires two arrays, one containing the old values from the previous iteration ('previous value array'), the other storing new values calculated from the current iteration ('new value array'). The roles of these arrays are interchanged between iterations. Namely, the 'new value array' becomes the 'previous value array' in the next iteration, and vice versa. To make the calculation of the boundary points the same as inside points, we extend the arrays by two rows and two columns outside each plane boundary, and set these extended array elements to 0. Thus there is no need for special treatment for boundary points during the computation.

Parallelization of the g5 kernel. Attainment of high performance on parallel machines require that programs exhibit balanced workloads and high data locality on all participating processors. Balanced workloads are not difficult to achieve for the g5 kernel code because the required computation for each point is always the same. The following three decompositions of the kernel's computation space result in balanced work load: (1) *row-based* parallelization divides the plane into blocks of rows and assigns each row block to one processor, (2) *column-based* parallelization divides the problem space into blocks of columns and assigns each column block to one processor, and (3) *grid-based* parallelization divides the plane into grids and distributes the grids among processors.

However, data locality on the KSR machines, and therefore, performance varies significantly among these three methods due to the machine's 'move-on-demand' shared memory policy. Specifically, the g5 implementation uses two arrays to store the old values from the previous iteration ('previous value array') and new values from the current iteration ('new value array'). After all points on the plane are computed, the distribution of the 'new value array' is exactly the same as the computational space distribution, because all replicated copies of array elements have been invalidated by storing new values into the array. During the next iteration of computation, however, the 'new value array' becomes the 'previous value array'. Therefore, some elements of the 'previous value array' must be accessed remotely. For a problem of size 256×256 computed on 16 processors, the column-based parallelization method requires each processor to access 128 remote subpages per iteration (Fortran uses column major array storage), while the row-based and grid-based (each processor is responsible for 16 grids of size 16×16) methods require each processor to access 512 and 704 subpages respectively, both of which are significantly higher than with the column-based approach.

If we increase the problem size to a $1K \times 1K$ plane (still running on 16 processors), the numbers of remote subpage accesses per iteration for the column-based, row-based, and 16×16 grid-based parallelization methods increase to 512, 2,048, and 11,264 respectively. Note that the number of remote subpage accesses for the column-based parallelization approach is determined by the number of rows in the computation space, while the number for the row-based approach is only affected by the number of columns in the computation space. The grid-based approach is affected by both factors, however. But if the grid size

increases as the problem size grows, the total number of remote subpage accesses drops drastically. For example, if the grid size increases to 256×256 for problem size $1K \times 1K$, each processor only needs to access 648 remote subpages, which is quite close to the column-based parallelization approach.

The access pattern of the 'new value array' is similar to that of the 'previous value array'. After each iteration, those elements of the 'previous value array' accessed by multiple processors are replicated on these processors' local caches. During the next iteration, the 'previous value array' becomes the 'new value array'. Writing new values into the 'new value array' has to invalidate those replicated subpages, a reversal of the process describe above. The column-based approach invalidates the least amount of subpages, while the small grid-based approach invalidates the most.

In summary, the three parallelization techniques, column-based, row-based, and grid-based decompositions, produce balanced work loads for the g5 kernel. However, column-based parallelization results in the best data locality, whereas small grid-based parallelization behaves the worst for small size grids.

3 Evaluation of the g5 kernel on KSR

In this section, the performance of the parallelized g5 kernel is evaluated by comparison of results obtained with the column-, row-, and grid-based methods of parallelization. The KSR-1 and KSR-2 machines are used in the evaluations. Last, the I/O throughput and performance of the g5 kernel are characterized and evaluated.

Computational performance and speedups on the KSR-1. Table 1 depicts the average ex-

| No. Proc. | 1 | 4 | 8 | 16 | 24 |
|------------------|-------|--------|--------|--------|--------|
| $1K \times 1K$ | 5.03 | 1.14 | 0.552 | 0.280 | 0.191 |
| 512×512 | 1.10 | 0.279 | 0.142 | 0.0736 | 0.0524 |
| 256×256 | 0.276 | 0.0723 | 0.0380 | 0.0206 | 0.0152 |

Table 1: Average execution time (in seconds) of each iteration of the g5 kernel (parallelized with the column-based approach).

ecution time of each iteration of the g5 kernel code on a 32-node KSR-1, and the resulted speedups are shown in Figure 1. Execution times and speedups are presented for three different problem sizes, $1K \times 1K$ ($1,024 \times 1,024$), 512×512 , and 256×256 . In the case of a problem size of an $1K \times 1K$ plane computing on 24 processors, the average computation time of each

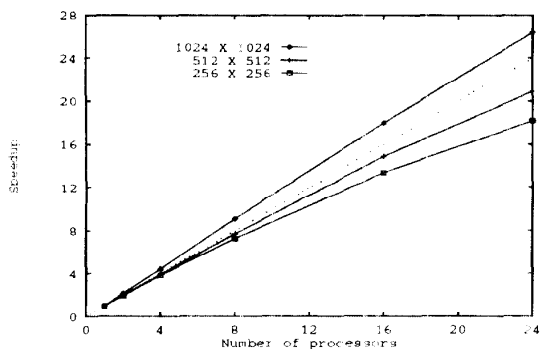


Figure 1: Speedups⁴ of the g5 kernel on the KSR-1.

iteration is less than 0.2 seconds. Therefore, the actual computational throughput of the application is at least 210 millions operations per second. Since the peak computational performance of 24 KSR-1 processors is 480 MFLOPS, the g5 kernel parallelized with the column-based technique realizes 44% of the machine's peak performance.

From the results in Table 1 and Figure 1, it is clear that the g5 kernel scales well both in problem size and in number of processors. Furthermore, as expected, speedup improves when problem sizes are increased.

Comparing the performance of different parallelization techniques. Section 2 describes several alternative parallelization techniques. Figure 2 depicts measurements (in terms of speedups) of the g5 kernel parallelized with these parallelization techniques.

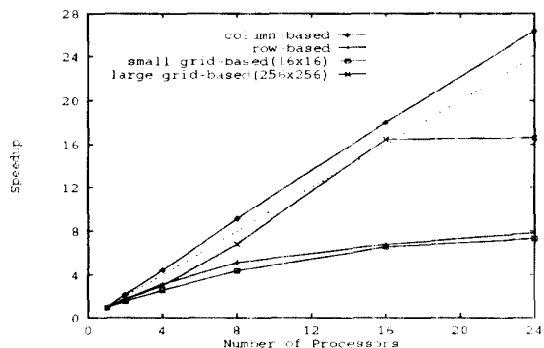


Figure 2: Speedups of the g5 kernel for different parallelization techniques on the KSR-1.

⁴Super-linear speedups are due to KSR-1's ALLCACHE memory architecture. When the g5 kernel runs on a large number of processors, it can load all its code and data into the fast sub-caches or local caches associated with these processors, while it can not when running on one processor.

niques: row-based, column-based, small grid-based (grid size 16×16), and large grid-based (grid size 256×256). The problem size for all the measurements is the same $1K \times 1K$ plane.

From the results shown in the figure, it is clear that column-based parallelization produces the best performance and speedups on any number of processors, while the small grid-based parallelization technique using small grids yields the worst performance. Row-based parallelization is generally close to the small grid-based approach. However, the large grid-based approach approximates the performance of the column-based approach, except on 24 processors. This anomaly is due to the fact that there are only 16 grids of size 256×256 for a problem size of $1K \times 1K$, and therefore at most 16 processors are needed.

We can conclude from these results that data locality determines the overall performance of the g5 kernel, especially on large numbers of processors. The column-based parallelization approach is superior in performance because it best preserves data locality. Performance differences due to data locality are significant. For example, when the g5 kernel runs on 24 processors, the best case speedup (column-base parallelization) is more than 3 times as much as the worst case (small grid-based parallelization).

Performance of the g5 kernel on the KSR-2.

The performance of the g5 kernel on the 64-node KSR-2 is shown in Figure 3. The figure shows the

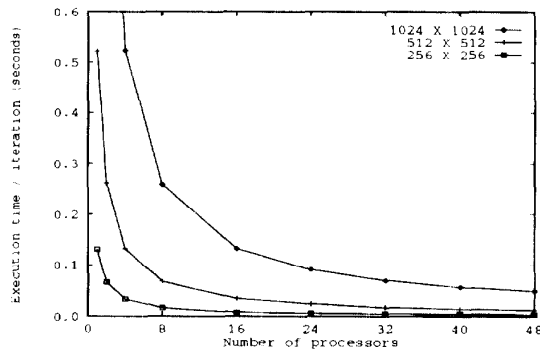


Figure 3: Average execution time of each iteration of the g5 kernel on a 64-node KSR-2 (with column-based parallelization).

average execution time of each iteration of the g5 kernel code with different problem sizes on different numbers of processors. By comparing the results on the KSR-2 (Figure 3) with the results on the KSR-1 (Table 1), it is clear that the execution speed of the g5 kernel on KSR-2 is about twice as fast as on KSR-1, re-

sulting from faster processor and interconnection network speed on the KSR-2. Column-based parallelization is again superior in performance to other techniques. Specifically, the execution time for problem size $1K \times 1K$ on 48 processors is less than 0.05 seconds, resulting in total computational throughput of 840 MFLOPS. While the peak computational performance of 48 KSR-2 processors is 1,920 MFLOPS, the g5 kernel can still realize 44% of KSR-2's peak performance.

From the comparison, we can also conclude that the g5 kernel scales very well except for small problem sizes on KSR-2 (speedup numbers on KSR-2 are not shown here due to constraints of space). It is also clear that running on two rings (the 64-node KSR-2 is interconnected by 2 rings) does not have any observable effects on the overall performance of the g5 kernel, thereby indicating good scalability for even larger machine sizes.

Input/Output performance on the KSR machines. Since typical seismic data processing applications involve large amounts of data, the I/O performance of the machine on which the applications are running can affect overall program performance. Only application-level I/O performance is measured in this paper. The KSR machine has I/O controllers attached to only some nodes, which implies that I/O performance may be different from node to node. Moreover, despite the availability of independent disks and I/O controllers, parallel I/O is not supported by the KSR machine's OS. As a result, the performance measurements presented here are attained from a simple program that reads from a file and writes to another. This program is run on different processors to obtain realistic averages. Table 2 shows the best and average times and the throughput of reading and writing files of size 8 MB on the KSR-2 installed at Georgia Tech.

| | Time (seconds) | | Throughput (MB/s) | |
|------------|----------------|------|-------------------|------|
| | Best | Avg. | Best | Avg. |
| write file | 2.94 | 5.67 | 2.72 | 1.41 |
| read file | 1.97 | 7.19 | 4.06 | 1.11 |

Table 2: I/O performance of the KSR-2: Best and average access times and throughput of reading and writing an 8 MB file.

From the results, it is clear that the I/O performance of the current KSR configuration is not sufficient for applications that require input and output of large amounts of data. However, there exist ways to improve I/O performance. For example, initial data

and final results can be compressed to reduce the total amounts of I/O, at some cost of extra computation. Support of parallel I/O can also improve the KSR's I/O performance.

4 Conclusions

The g5 kernel is a computational kernel extracted from seismic data processing applications. In this paper, the kernel is parallelized and its performance is evaluated on the KSR-1 and KSR-2 multiprocessors. Three approaches for parallelizing the g5 kernel are analyzed: column-based, row-based, and grid-based parallelizations. All three approaches result in well balanced decompositions, but differ significantly in data locality. In general, the column-based approach has the best data locality, while the small grid-based approach has the worst.

These results clearly indicate that data locality is one of the critical factors for attaining high performance for the g5 kernel. The best parallelized g5 kernel code achieves about 44% of both the KSR-1 and KSR-2 machines' peak computational performance. Similar conclusions can be drawn for the g5 kernel on any other NUMA shared memory machines.

The I/O performance of the KSR is not adequate for applications that require input and output of large amounts of data. Improvements of the performance of the I/O system, such as supporting parallel I/O, are required to make possible efficient execution of these applications.

Acknowledgment

Thanks to Dr. Peter T. Highnam of Schlumberger Laboratory for Computer Science for sponsoring this project and for many extremely helpful comments, and to Dr. Karsten Schwan of College of Computing, Georgia Tech for his help in writing this paper.

References

- [1] D. H. Bailey, E. Barszcz, L. Dagum, and H. D. Simon. NAS parallel benchmark results. In *Proceedings of Supercomputing'92*, pages 386-393, Minneapolis, Minnesota, November 1992.
- [2] D. Hale. Stable explicit depth extrapolation of seismic wavefields. In *60th Annual International Meeting and Exposition of the Society of Exploration Geophysicists*, pages 1301-1304, 1990.
- [3] P. T. Highnam and A. Pieprzak. Implementation of a fast, accurate 3-D migration on a massively parallel computer. In *61st Annual International Meeting and Exposition of the Society of Exploration Geophysicists*, pages 338-340, 1991.