Optimization and Evaluation of Hartree-Fock Application's I/O with PASSION

Meenakshi A. Kandaswamy
EECS Department
Syracuse University
Syracuse
NY 13244
(meena@ece.nwu.edu)
(http://www.ece.nwu.edu/~meena)

Mahmut T. Kandemir
EECS Department
Syracuse University
Syracuse
NY 13244
(mtk@ece.nwu.edu)

Alok N. Choudhary
ECE Department
Northwestern University
Evanston
IL 60208
(choudhar@ece.nwu.edu)
(http://www.ece.nwu.edu/~choudhar)

David E. Bernholdt
Northeast Parallel Architectures Center
Syracuse University
Syracuse
NY 13244
(bernhold@npac.syr.edu)
(http://www.npac.syr.edu/users/bernhold/homepage)

Abstract:

Parallel machines are an important part of the scientific application developer's tool box and the processing demands placed on these machines are rapidly increasing. Many scientific applications tend to perform high volume data storage, data retrieval and data processing, which demands high performance from the I/O subsystem. In this paper, we conduct an experimental study of the I/O performed by the Hartree-Fock (HF) method, as implemented using a fully distributed data approach in the NWChem parallel computational chemistry package. We use PASSION, a parallel and scalable I/O library to improve the I/O performance of the application and present extensive experimental results. The effects of both application-related factors and system-related factors on the application's I/O performance are studied in detail. We rank the optimizations based on the significance and impact on the performance of HF's I/O phase as: I. efficient interface to the file system, II. prefetching, and III. buffering. The results show that within the limits of our experimental framework, application-related factors are more effective on the overall I/O behavior of this application. We obtained up to 95% improvement in I/O time and 43% improvement in the overall application performance with the optimizations.

Keywords:

Proceedings of the ACM/IEEE SC97 Conference (SC'97)
0-89791-985-8/97 $ 17.00 © 1997 IEEE

IEEE COMPUTER SOCIETY

Computational chemistry, Parallel Application, High-performance I/O, Run-time System, Prefetching, File System interface.

1 Introduction

Parallel machines are an important part of the scientific application developer's tool box and the computational and processing demands placed on these machines are rapidly increasing. Many scientific applications tend to perform high volume data storage, data retrieval and data processing, which demands high performance from the I/O subsystem. However, the advancements of I/O systems are much behind compared to processor development and hence resulting in a widening gap between the two subsystems and as a result I/O-intensive high-performance applications do not perform well. In this paper, we conduct a detailed experimental study of the I/O performed by the Hartree-Fock (HF) method [11], as implemented using a fully distributed data approach in the NWChem parallel computational chemistry package [9]. The Hartree-Fock method describes the electronic structure of a molecule (the wavefunction), and is widely used in quantum chemical calculations. HF I/O phases contribute up to 62.34% of the total execution time in some of our application inputs and hence make a very compelling case for the need of I/O optimizations and analysis of its I/O phase. For some application inputs discussed in this paper, up to 1.9GB of data are read/ written per application phase and this warrants optimizations to the I/O phase to efficiently utilize the existing I/O system.

We use PASSION [17], [7], [8], [13], a parallel and scalable I/O library to implement the application's I/O. Furthermore, we modify the application to use PASSION's optimizations such as prefetching. We classify the factors that affect the I/O performance of the application into two categories, namely, application-related factors and system-related factors and investigate the impact of each category on the I/O behavior of HF.

In this paper, we 1) present extensive experimental results related to the I/O behavior of HF which is an I/O-intensive application using PASSION library, 2) explain the effect of application and system-related factors on HF and 3) rank the optimizations based on the significance and impact on the performance of HF's I/O phase as: I. efficient interface to the file system, II. prefetching optimization, and III. buffering. The results show that within the limits of our experimental parameters, application-related factors are more effective on the overall I/O behavior of this application. We obtained up to 95% improvement in I/O time and 43% improvement in the overall application performance.

2 NWChem and the Hartree-Fock Application

The NWChem computational chemistry package has been developed, mainly at the Pacific Northwest National Laboratory, with the goal of expanding the time and length scales and improving the accuracy of chemistry simulations by taking advantage of massively parallel processors (MPPs) [5], [6].

The Hartree-Fock method is one of the simplest approaches which include quantum mechanical effects, and is based on self-consistent field (SCF) theory [11]. At the heart of the Hartree-Fock method is the construction of the Fock matrix *F* according to the formula

$$F_{pq} = h_{pq} + \sum_{r=1}^{N} \sum_{s=1}^{N} D_{rs}[(pq|rs) - \frac{1}{2}(pr|qs)] \qquad (1)$$

where N is the dimension of the basis set, *h* are one-electron Hamiltonian matrix elements, *D* is the one-particle density matrix, and (*pq*|*rs*) are two-electron integrals. The iterative procedure starts with an initial guess for the density matrix, *D*, which is then used to construct a Fock matrix, which in turn is used to improve the density matrix. The one- and two-electron integrals, *h* and (*pq*|*rs*) depend on the positions of the atoms in the molecules, and on the specific choice of basis set. Their values do not change during the iterative procedure. The two-electron integrals are quite numerous, formally $O(N^4)$.

In a disk-based implementation, the integrals are computed on the first iteration and written to disk, then read from disk rather than being recomputed for each subsequent iteration. In NWChem each node writes a private file of the integrals it evaluated during first construction of the Fock matrix. Figure 1 shows a sketch of the I/O activity observed from a single compute node when the disk-based version of HF is used. The HF application code consists of about 16,500 lines of

0-89791-985-8/97 $ 17.00 © 1997 IEEE
IEEE COMPUTER SOCIETY

FORTRAN code and about 225,000 lines when supporting libraries are included.

3 Hardware and Software Details

3.1 Hardware Platform

We use the 512-node Intel Paragon with i860 processors at Caltech for all the experiments discussed in this paper. The Parallel File System (PFS) is part of the OSF.R1.4.3 release of the operating system. There are two different parallel file system configurations that we use, namely a 12 I/O node X 2 GB partition on original Maxtor RAID 3 level disks and a 16 I/O node X 4 GB partition on individual Seagate disks. In both the partitions, the stripe factor is equal to the number of I/O nodes. The default striping unit size of both the I/O partitions is 64 KB.

3.2 PASSION Library

PASSION [17], [7], [8], [13], [3] is a parallel run-time system which offers a high-level interface to the underlying parallel I/O subsystem of a parallel computer. PASSION calls can be used from both in-core and out-of-core programs for I/O support. In addition, it offers several optimizations such as data prefetching, data sieving, data reuse etc. PASSION calls can be issued directly from a parallel program through a C or Fortran interface and they can use the underlying native parallel file system calls. PASSION is supported on a variety of platforms, including Intel Paragon, Touchstone Delta, iPSC/860 and IBM SP-2. There are two abstract storage models supported by PASSION: Local Placement Model (LPM) and Global Placement Model (GPM). In the LPM model, each processor stores data on a virtual local disk and only that processor has access to that disk. Data sharing between processors is done by means of communication. The data distribution amongst the processors can be seen at the file-level itself. We should emphasize that the LPM model fits the I/O model used in HF, so we use LPM in our experiments. Also, we use Pablo [15] I/O library to trace the I/O activity of HF both qualitatively and quantitatively.

3.3 Default Configuration for the Experiments

The default striping configuration for our experiments is fixed at : number of processors = 4; striping unit = 64KB; and striping factor = 12; and the default PFS partition we use is 12 I/O node X 2 GB partition on original Maxtor RAID disks. The various HF code implementations we use are 1) Original--the original code with Fortran I/O from Pacific Northwest Laboratory, 2) PASSION--a modified version which uses PASSION read/write calls, and 3) Prefetch--a modified version which uses PASSION Prefetch calls.

4 Overview of I/O Behavior of HF

As noted above, it is possible either to recompute the integrals for each iteration of the HF procedure, or to use disk storage to avoid the need to recompute. We will refer to the disk-based (Original) version as DISK, and the (Original) recomputing implementation as COMP.

In Table 1, we present the best sequential execution times obtained from several runs of both the COMP version and the DISK version for several input sizes in the range of $\mathcal{N}$=66 to $\mathcal{N}$=134. Note that factors such as the nature of the molecule and the chosen basis set, may result in substantial variations in the computational cost of the integrals and the number of iterations required to converge the HF solution, so that larger $\mathcal{N}$ does not strictly imply a more expensive calculation. We observe that the DISK version has the best sequential time for all cases except one case. The speedups presented in Figure 2 for these problem sizes are calculated over the best sequential execution times from Table 1. From the curves, we can conclude that the disk based version of HF is preferable to the version which recomputes the integrals. This result emphasizes the importance of I/O for HF.

IEEE COMPUTER SOCIETY

The I/O activity of the HF application consists of two main phases:

- Write phase (performed only once), where the integrals are calculated and packed in a memory buffer and written to disk after the buffer is full;

- Read phase (performed several times), integral values are read from disk to a memory buffer to compute the Fock matrix. The integral file is resident on the disks and striped across various I/O nodes.

The other small I/O accesses are to the input file and a run-time database file used for check pointing some values, but these contribute to an insignificant part of the file activity. We classify the three representative runs as SMALL ($\mathcal{N}$ = 108), MEDIUM ($\mathcal{N}$ = 140) and LARGE ($\mathcal{N}$ = 285) input sizes, based on the number of basis functions. We present I/O summary tables for the various experiments and these capture the various I/O operation counts, the time taken, volume of data read or written and percentages of I/O and execution times for different operations. Note that this includes the I/O activity performed by all the processors that are executing the application. Along with this, we present tables that give the distribution of the request sizes that are issued from the application. Also we present graphs that show quantities related to the read and write requests such as durations of each request, size etc., across the entire execution time of the application.

In Table 2, we summarize the results of the SMALL input with the various I/O measurements. Firstly, we find reads to be the dominant operation in operation count, I/O time and volume of data. This is because the reads are repeated several times and contribute to the bulk of the I/O time. For the SMALL case, 93.76% of the I/O time is read operations and that is 39.28% of the total execution time. This is followed by writes which are done to write the integrals to files in the write phase and they contribute to only 4.91% of the total I/O time and 2.06% of the total execution time. All the other operations such as open, close, seek and flush take less than 2% of the total I/O time. We clearly see that, for this input the HF code is I/O intensive and specifically read operation-intensive. In Table 3, we present the distribution of the read and write request sizes. The small reads and writes in the range of (< 4 K) corresponds to the initial reads to the input file and writes to the run-time database file for check pointing some calculations. The larger writes and reads (64K <= Sz < 256K) correspond to the integral file writes and reads. We used the the default buffer size set in the program of 8192 8-byte elements.

In Figure 3, we present the durations of the read and write operations performed in SMALL, over the entire execution time of the program. We can clearly identify the write phase of integral values (performed just once) followed by the read phase (performed many times). Also the small reads at the very beginning, where the small input files are read and some scarce small writes to the run-time database file sprinkled about. We see that the average duration of read operations is 0.1 second and the average write operations is 0.03 second. Figure 4, on the other hand presents the read and write sizes of SMALL which shows the read and write request sizes issued from the application.

Tables 4 and 5 show the I/O summary of MEDIUM and the read and write size distribution respectively. We see that the I/O time corresponds to 62.34% of the total execution time. Reads and writes contribute to 59.01% and 3.31% of the total execution time. The trend of the operation distribution is maintained as in SMALL. Figure 5 gives the read and write operation durations across the entire execution time of the program. The average read operation duration is 0.12 second and the average write duration is 0.087 second. Tables 6 and 7 and Figure 6 present the I/O information for LARGE. We note that the I/O behavior of the application is similar for all inputs.

5 I/O Optimizations

There are several factors that can affect the I/O performance of an application. Specifically, we divide them into two categories, namely application related parameters and system related parameters. Application related parameters are those that can be altered from the application by the user. The system related parameters are those that are dependent on the parallel machine on which the application is executed and its configuration on which the user has limited control and/or is generally reluctant to change. The application related parameters are interface to the parallel file system, prefetching, and buffering, whereas the system related parameters are number of processors, number of I/O nodes and stripe factor, and size of striping unit.

5.1 Effect of Application Related Parameters

We study the effect of each one of the application-related parameters on the I/O performance of HF. Recall that all the

0-89791-985-8/97 $ 17.00 © 1997 IEEE

experiments are performed on the default configuration unless otherwise specified.

5.1.1 Software Interface to File System

The interface to the file system plays an important role as it adds a layer between the application and the I/O subsystem and hence can either further increase the I/O latency if not well implemented or play a role in reducing the I/O time observed by the application by performing optimizations. PASSION provides a convenient high-level interface for the user. PASSION library is implemented in C with wrappers so that it can be called both from Fortran and C and can use the underlying parallel file system efficiently. Recall that original HF implementation uses Fortran I/O calls.

Table 8 and 9 give the I/O summary of the PASSION version of HF and the read and write size distribution for the SMALL problem. The ratio among the operations and the operation counts (except that of seek) have remained almost the same as the Original case. However, the I/O time now constitutes only 27% as opposed to the 41.90% of the Original case. Out of this 14.9% reduction, almost 14% is due to the reduction in the read times. We note that the number and the order of the I/O calls remain the same. The mere change to the library which uses C calls and a better interface to the file system have brought up this significant reduction. The PASSION library does not have any knowledge of where the file pointer is from a previous I/O call and so a fresh seek has to be performed for every call. Hence, the increase in the number of seeks. We show the results of PASSION implementation in Figure 7 for the SMALL input. We see that the average read request duration has reduced by one order of magnitude (0.05 second) and the average write request duration reduced to 0.01 second from 0.03 second. When the average request durations reduce, the total execution time also reduces as can also be observed from Figure 7. We present the results for the PASSION version of MEDIUM input in Table 10. We see that there is a reduction in the total I/O time from 62.34% in the Original version to 43.81% in the PASSION version. From Figure 8, we see that the average read and write durations are 0.05 second and 0.06 second, respectively. In Table 11, we show the I/O summary of the PASSION version for LARGE and in Figure 9 we see the read and write duration distributions. The total I/O time for LARGE has reduced from 54.96% to 39.56% due to PASSION.

In summary, we observe that a more efficient software interface to the file system has resulted in a significant reduction (23%-28%) in the total execution time for all the input sizes.

5.1.2 Prefetching

A typical application of prefetching is shown in Figure 10. Prefetching has been shown to hide the I/O latency significantly in [1], [14]. HF is iterative in nature with computation and I/O part of each iteration and hence lends itself well to pipelined prefetch accesses.

Tables 12 and 13 present the I/O summary and distribution of I/O request sizes of the prefetching version of HF for SMALL. We use the PASSION prefetch routines to prefetch the group of integrals required for the next iteration in the current iteration of the program. Unlike the case of the software interface, I/O time is not reduced but is effectively hidden by overlapping I/O with computation. A majority of the reads are asynchronous reads, which correspond to the prefetch operations. The total read time contributed to 93.23% of the I/O time in the PASSION case and now contributes to a much smaller fraction (36.83%) of the total I/O time. We see that the I/O time is reduced very significantly (from 785.72 seconds to 95.20 seconds) in the entire program. We note the fact that the computation time is sufficient to hide or overlap only some percentage of the the time spent on I/O. Due to this, when computation is completed for the current iteration, the prefetch request for the data needed in the next iteration is not complete and the execution of the program is stalled till the prefetch is completed. (at the `wait()` command in Figure 10)

We also note that prefetching did not produce results as we expected. PASSION uses the file system's non-blocking or asynchronous reads for prefetching and because of this it has to translate a single request to a logically contiguous chunk of data access into multiple requests to physically contiguous chunks of data accesses. This book-keeping contributes to a big percentage of the prefetching overhead. Posting of individual requests also adds to the overhead as each request needs to obtain a token to be entered in the queue of asynchronous requests to a given file. Copying data from the prefetch buffer to the application buffer also contributes to the prefetch overhead. (see Figure 10).

Figure 11 shows the I/O activity of SMALL with prefetching across the execution time. Table 14 and Figure 12 give the I/O summary and read and write duration activity of MEDIUM respectively and those for LARGE are given in Table 15 and and Figure 13. Figure 14 summarizes the read and write durations of SMALL and MEDIUM and we see that there is approximately a 50% reduction in all the cases except one case. These explain the reduction in I/O time.

Figure 15 summarizes the execution times of the Original version, PASSION version and Prefetch version for our inputs. We see that there is a significant reduction in the execution time due to the change of the file system interface. We find a moderate reduction in the execution time going from the PASSION version to the Prefetch version. PASSION produces a 23%, 28% and 23% reduction in total time for SMALL, MEDIUM and LARGE respectively and 51%, 43% and 44% reduction in I/O time for SMALL, MEDIUM and LARGE respectively. Prefetch produces a 32%, 43% and 39% reduction in execution times for SMALL, MEDIUM and LARGE respectively and 94%, 94% and 95% reduction in I/O time for SMALL, MEDIUM and LARGE, respectively.

5.1.3 Buffering

In this experiment, we modify the available memory (buffer) to the integral calculations (also called ''slab'' in PASSION). Recall, from earlier discussion on the DISK version of HF, that both the write and read operations to the integral files are performed through a memory buffer. When integrals are computed, a buffer of a certain size is filled up and then written to the disk. We observe from Table 16 that the total and I/O times decrease with the increase in the memory buffer size available for the application. The default buffer size of HF is set to 8192 doubles or 64K bytes. A larger memory buffer enables more integrals to be stored on memory. As an example, when we move from 64K to 256K, there is a 8%, 27% and 50% reduction in the I/O times for Original, PASSION and Prefetch, respectively.

5.2 Effect of System Parameters

In this section, we study the effect of system parameters on the application which are dependent on the machine configuration available.

5.2.1 Number of Processors

We study the scalability of the application both overall as well as the I/O part of the application. All the three problem sets are run on 4, 16 and 32 processors. The speedups shown in Figure 16 with respect to the four processor Original case. The I/O scalability improves when moving from the Original version to the PASSION version and it reflects in the total scalability of the program. The increase in the speedups when moving from the PASSION version to the Prefetch version is significant. In fact, the I/O speedups are super-linear in the case of the prefetching version and this can be explained based on the fact that the algorithm has been somewhat changed due to the introduction of prefetching calls. We see that the PASSION version and the Prefetch version scale better compared to the Original version. To gain more insight into the scalability of the application, we also conducted experiments with many other input sizes. Figure 17 shows the generic I/O speedup curves for a typical input for HF. The I/O speedup of PASSION and Prefetch are generally always better than that of the Original. Up to the point P_0, I/O scales well for all the versions with the Prefetch version scaling the best, because of the parallel access to the I/O nodes. Beyond P_0 however, the contention in the I/O nodes dominates and speedups degrade for all the version. The real value of P_0 depends on the problem size and number of I/O nodes.

5.2.2 Stripe Factor

Table 17 shows that there is a reduction in the average time to service a read or write request when the stripe factor increases to a value of 16 from a value of 12. We see that this reduction is reflected in the total execution times and I/O times (Table 18). This is an expected result, because when the number of compute nodes remains a constant and the number of I/O nodes is increased, the average number of requests received by an I/O node (and consequently the contention in the I/O subsystem) decreases.

5.2.3 Size of Striping Unit

Table 19 presents the execution times and I/O times for the striping units of 32K, 64K and 128K. We see that the effect of striping unit size is minimal and unpredictable. In HF, each processor accesses a unique file that is striped across the various I/O nodes and there will be interfering requests to I/O nodes based on the position at which striping is started. We observe from Table 19 that, for the Original case, 128K is the best striping unit size among our experimental values whereas for the PASSION and Prefetch cases 64K is the best.

0-89791-985-8/97 $ 17.00 © 1997 IEEE

6 Relative Significance of Optimizations

In this section, we present an incremental evaluation of the I/O optimizations performed in this paper. The optimizations performed fell into application-related and system-related categories. We will consider the SMALL only as the results were similar in the cases of MEDIUM and LARGE.

We represent each combination in the Figure 18 with a five-tuple of (V,P,M,Su,Sf), where V is the version used (O - Original, P - PASSION, F - Prefetch); P is the number of processors; M is the buffer size (in KB); Su is the stripe unit size (in KB); and Sf is the stripe factor; V and M being the application-related factors and P, Su and Sf are the system-related factors. Note that (O,4,64,64,12) corresponds to the default configuration. In Figure 18 we present the percentage reductions with respect to original execution and original I/O times.

We see that just by changing the Fortran I/O calls to PASSION calls, we get a reduction of 23.24% in total execution time and 50.52% in I/O time and the reduction in I/O time is fully reflected (in terms of seconds) in the execution time. Prefetching version additionally reduces execution time and I/O time by 8.73% and by 43.48% respectively. The reduction in the I/O time is not fully reflected in the reduction in execution time, in the case of prefetching. When the number of processors is increased, the computation performed per processor also scales down, but the I/O contention increases as the number of I/O nodes still remains the same. When the number of processors is increased to 32, we see that there is a reduction in I/O time and also increased contention in the I/O nodes. There is an additional reduction of 44.03% and 4.4% in the total time I/O time respectively. However, we speculate that increasing the number of processors to large values will increase the contention on the I/O subsystem. When the buffer size of the application is increased there is a further reduction of an additional 1% and 0.6% in execution time and I/O time respectively. When the stripe unit is increased to 128KB, we see that there is an additional decrease of 1% and 0.3% in execution time and I/O time respectively. When the stripe factor is increased to 16, we find that there is no reduction in execution time and 0.5% additional decrease in I/O time. To summarize, we argue that the application-related factors are more important for the overall performance of HF on Intel Paragon. From these observations, within our experimental parameters, we can order the parameters as efficient interface, prefetching, buffering, number of processors, striping factor and striping unit size, according to their impact on the I/O performance of this application.

7 Related Work and Conclusions

There are several works on I/O characterization of large applications. Three I/O-intensive applications from the Scalable I/O Initiative Application Suite are studied in [4]. An I/O-intensive three-dimensional parallel application code is used to evaluate the I/O performances of the IBM SP and Intel Paragon in [12]. They found IBM SP to be faster with read operations and Paragon for writes. del Rosario and Choudhary [16] discuss several grand-challenge problems and the demands that they place on the I/O performance of parallel machines. Significant amount of work has been done in the areas of out-of-core compilation [2], [3], [14]. In [10], Smirni *et.al*, discuss two different algorithms of the Hartree-Fock SCF method, namely, MESSKIT and NWChem. They too point out the importance of an efficient interface. In comparison, we evaluated the NWChem both from application and system related parameters point of view. We modified NWChem to work with PASSION library and experimented with prefetching.

In this paper we studied an I/O-intensive application, namely, the disk-based version of HF in detail in its original form. HF has two phases, namely the write phase where the integral calculations are computed and written to disk through an application buffer and several read phases where the integrals are read from disk. Together they contribute to a significant part of the total execution time. We used PASSION, a parallel and scalable run-time I/O library to reduce the I/O time significantly. We studied the I/O behavior by experimenting with both application-related and system-related parameters. We found out that the application-related parameters play a major role in determining the I/O performance. We ordered the parameters according to the contribution they made to I/O behavior as efficient interface, prefetching and buffering.

References

1 M. Arunachalam, A. Choudhary and B. Rullman. **Implementation and evaluation of prefetching in the Intel Paragon parallel file system**, In Proceedings of International Parallel Processing Symposium, April 1996.

2 R. Bordawekar. **Techniques for Compiling I/O Intensive Parallel Programs**, Ph.D. Thesis, Dept. of Electrical and Computer Eng., Syracuse University, April 1996.
3 R. Bordawekar, J. M. del Rosario and A. Choudhary. **Design and implementation of primitives for Parallel I/O**, In Proceedings of Supercomputing'93, November 1993.
4 P. E. Crandall, R. A. Aydt, A. A. Chien and D. A. Reed. **Input/output characteristics of scalable parallel applications**, In Proceedings of Supercomputing'95.
5 M. F. Guest, E. Aprà, D. E. Bernholdt, H. A. Früchtl, R. J. Harriso n, R. A. Kendall, R. A. Kutteh, X. Long, J. B. Nicholas, J. A. Nichols, H. L. Taylor, A. T. Wong, G. I. Fann, R. J. Littlefield, and J. Nieplocha. **High performance computational chemistry: NWChem and fully distributed parallel algorithms**, In High Performance Computing: Technology, Methods, and Applications, volume 10 of Advances in Parallel Computing, pages 395-427. Elsevier, Amsterdam, 1995.
6 M. F. Guest, E. Aprà, D. E. Bernholdt, H. A. Früchtl, R. J. Harriso n, R. A. Kendall, R. A. Kutteh, X. Long, J. B. Nicholas, J. A. Nichols, H. L. Taylor, A. T. Wong, G. I. Fann, R. J. Littlefield, and J. Nieplocha. **Advances in parallel distributed data software; computational chemistry and NWChem**, In Applied Parallel Computing. Computations in Physics, Chemistry and Engineering Science, volume 1041 of Lecture Notes in Computer Science, Springer, Heidelberg, 1996.
7 A. Choudhary, R. Bordawekar, S. More, K. Sivaram and R. Thakur. **PASSION run-time library for the Intel Paragon**, In Proceedings of the Intel Supercomputer User's Group Conference, June 1995.
8 A. Choudhary, R. Bordawekar, M. Harry, R. Krishnaiyer, R. Ponnusamy, T. Singh and R. Thakur. **PASSION: Parallel and scalable software for input-output**, NPAC technical report SCCS-636, Sept. 1994.
9 **NWChem, a computational chemistry package for parallel computers, version 1.1, 1995.** High Performance Computational Chemistry Group, Pacific Northwest Laboratory, Richland, Washington 99352, USA.
10 E. Smirni, C. L. Elford, A. J. Laevry, D. A. Reed and A. A. Chien. **Algorithmic influences on I/O access patterns and parallel file system performance**, Technical Report, Pablo Group, UIUC.
11 A. Szabo and N. S. Ostlund. **Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory**. McGraw-Hill, New York, revised first edition, 1989.
12 R. Thakur, W. Gropp, and E. Lusk. **An Experimental Evaluation of the Parallel I/O Systems of the IBM SP and Intel Paragon Using a Production Application**, In Proc. of the 3rd Int'l Conf. of the Austrian Center for Parallel Computation (ACPC) with special emphasis on Parallel Databases and Parallel I/O, September 1996. Lecture Notes in Computer Science 1127, Springer-Verlag, pp. 24-35.
13 R. Thakur, R. Bordawekar and A. Choudhary, R. Ponnusamy and T. Singh. **PASSION run-time library for parallel I/O**, In Proceedings of the Scalable Parallel Libraries Conference, October 1994.
14 Todd C. Mowry, Angela K. Demke and Orran Krieger. **Automatic compiler-inserted I/O prefetching for out-of-core applications**, In Second symposium on operating systems design and implementations (OSDI'96), October 28-31, 1996.
15 D. A. Reed, R. A. Aydt, R. J. Noe, P. C. Roth, K. A. Shields, B. W. Schwartz and L. F. Tavera. **Scalable performance analysis: The Pablo performance analysis environment**, In Proceedings of the Scalable parallel libraries conference, A. Skjellum, Ed. IEEE Computer Society, 1993, pp.104-113.
16 J. M. del Rosario and A. Choudhary, **High Performance I/O for Parallel Computers: Problems and Prospects**, In IEEE Computer, March 1994.
17 R. Thakur, A. Choudhary, R. Bordawekar, S. More and S. Kuditipudi, **PASSION: optimized I/O for parallel applications**, In Computer, IEEE Computer Society, June 1996.

8 Acknowledgments

A modified version of NWChem Version 1.1, as developed and distributed by Pacific Northwest National Laboratory, P. O. Box 999, Richland, Washington 99352, USA, and funded by the U. S. Department of Energy, was used to obtain some of these results. We would like to thank Fatih Sevilgen for the initial implementation of HF with PASSION. We also thank Evgenia Smirni for her help in using the Pablo instrumentation library.

NSF Young Investigator Award CCR-9357840, NSF CCR-9509143 and in part by the Scalable I/O Initiative, contract number DABT63-94-C-0049 from Defense Advanced Research Projects Agency(DARPA) administered by US Army at Fort

Huachuca. Dr. D.E.Bernholdt was supported by the Alex G. Nason Fellowship at Syracuse University.

Problem Size N	Best Seq. time (seconds)	Version
66	101.8	DISK
75	433.3	DISK
91	855.0	DISK
108	3335.6	DISK
119	4984.9	COMP
134	2915.0	DISK

Table 1: Best sequential Execution times

Operation	Operation Count	I/O Time (Seconds)	I/O Volume (Bytes)	Percentage of I/O time	Percentage of Execution time
Open	19	3.13		0.20	0.08
Read	14,521	1489.07	909,301,536	93.76	39.28
Seek	1,018	17.0		1.07	0.45
Write	2,442	78.01	57,477,540	4.91	2.06
Flush	50	0.44		0.03	0.01
Close	14	0.52		0.03	0.01
All I/O	18,064	1,588.17	966,779,076	100.0	41.9

Table 2: I/O Summary of the Original version of SMALL: 4 processors

Operation	Size < 4K	4K < Size < 64K	64K < Size < 256K	256K <= Size
Read	646	3	13,872	0
Write	1,572	3	867	0

Table 3: Read and Write Size distribution of the Original version of SMALL : 4 processors

Operation	Operation Count	I/O Time (Seconds)	I/O Volume (Bytes)	Percentage of I/O time	Percentage of Execution time
Open	19	3.10		0.01	0.01
Read	258,636	28,937.03	16,914,356,715	94.66	59.01
Seek	903	6.36		0.02	0.01
Write	18,865	1,623.27	1,128,649,105	5.31	3.31
Flush	43	0.16		0.00	0.00
Close	14	0.40		0.00	0.00
All I/O	278,480	30,570.31	18,043,005,820	100.00	62.34

Table 4: I/O Summary of the Original version of MEDIUM: 4 processors

Operation	Size < 4K	4K < Size < 64K	64K < Size < 256K	256K <= Size
Read	573	3	258,060	0
Write	1,658	3	17,204	0

Table 5: Read and Write Size distribution of the Original version of MEDIUM : 4 processors

Operation	Operation Count	I/O Time (Seconds)	I/O Volume (Bytes)	Percentage of I/O time	Percentage of Execution time
Open	19	1.97		0.00	0.00
Read	566,315	60,284.31	37,077,005,416	95.56	51.66
Seek	994	8.01		0.01	0.01
Write	40,331	2,792.11	2,475,802,305	4.43	2.39
Flush	49	0.25		0.00	0.00
Close	14	0.46		0.00	0.00
All I/O	607,722	63,087.11	39,552,807,721	100.00	54.06

Table 6: I/O Summary of the Original version of LARGE: 4 processors

Operation	Size < 4K	4K < Size < 64K	64K < Size < 256K	256K <= Size
Read	632	3	565,680	0
Write	2,616	3	37,712	0

Table 7: Read and Write Size distribution of the Original version of LARGE : 4 processors

IEEE COMPUTER SOCIETY

Operation	Operation Count	I/O Time (Seconds)	I/O Volume (Bytes)	Percentage of I/O time	Percentage of Execution time
Open	19	0.67		0.08	0.02
Read	14,521	732.49	909,301,536	93.23	25.17
Seek	15,693	14.16		1.8	0.49
Write	2,446	37.39	57,477,556	4.81	1.3
Flush	50	0.17		0.02	0.01
Close	14	0.44		0.06	0.02
All I/O	32,743	785.72	966,779,092	100.0	27.0

Table 8: I/O Summary of PASSION version of SMALL: 4 processors

Operation	Size < 4K	4K <= Size < 64K	64K <= Size < 256K	256K <= Size
Read	646	3	13,872	0
Write	1,576	3	867	0

Table 9: Read and Write Size distribution of PASSION version of SMALL : 4 processors

Operation	Operation Count	I/O Time (Seconds)	I/O Volume (Bytes)	Percentage of I/O time	Percentage of Execution time
Open	19	0.88		0.01	0.00
Read	258,621	13,843.16	16,913,373,555	92.20	40.39
Seek	276,091	119.48		0.80	0.35
Write	18,868	1,049.35	1,128,583,577	6.99	3.06
Flush	43	0.20		0.00	0.00
Close	14	0.45		0.00	0.00
All I/O	553,656	15,013.51	18,041,957,132	100.00	43.81

Table 10: I/O Summary of PASSION version of MEDIUM : 4 processors

Operation	Operation Count	I/O Time (Seconds)	I/O Volume (Bytes)	Percentage of I/O time	Percentage of Execution time
Open	19	0.65		0.00	0.00
Read	566,330	33,805.21	37,077,988,576	95.38	37.73
Seek	604,342	256.56		0.72	0.29
Write	40,336	1,380.79	2,475,867,865	3.90	1.54
Flush	49	0.15		0.00	0.00
Close	14	0.37		0.00	0.00
All I/O	1,211,090	35,443.72	39,553,856,441	100.00	39.56

Table 11: I/O Summary of PASSION version of LARGE : 4 processors

Operation	Operation Count	I/O Time (Seconds)	I/O Volume (Bytes)	Percentage of I/O time	Percentage of Execution time
Open	19	0.64		0.67	0.02
Read	649	3.17	75,168	3.32	0.12
Async Read	13,936	35.07	913,421,184	36.83	1.36
Seek	15,757	13.20		13.87	0.51
Write	2,446	38.58	57,477,556	40.52	1.5
Flush	50	0.16		0.17	0.01
Close	14	4.39		4.61	0.17
All I/O	32,871	95.20	970.973,908	100	3.69

Table 12: I/O Summary of Prefetch version of SMALL : 4 processors

Operation	Size< 4K	4K <= Size < 64K	64K <= Size < 256K	256K <= Size
Read	646	3	0	0
Asynch Read	0	0	13,936	0
Write	1,576	3	867	0

Table 13: Read and Write Size distribution of Prefetch version of SMALL: 4 processors

IEEE COMPUTER SOCIETY

Operation	Operation Count	I/O Time (Seconds)	I/O Volume (Bytes)	Percentage of I/O time	Percentage of Execution time
Open	19	0.81		0.05	0.00
Read	576	2.85	72,075	0.18	0.01
Async Read	258,135	609.93	16,919,200,440	37.86	2.23
Seek	276,183	117.49		7.29	0.43
Write	18,870	876.19	1,128,714,665	54.39	3.20
Flush	43	0.13		0.01	0.00
Close	14	3.48		0.22	0.01
All I/O	553,840	1,610.89	18,047,987,180	100.00	5.89

Table 14: I/O Summary of Prefetch version of MEDIUM : 4 processors

Operation	Operation Count	I/O Time (Seconds)	I/O Volume (Bytes)	Percentage of I/O time	Percentage of Execution time
Open	19	1.29		0.04	0.00
Read	635	3.19	75,496	0.11	0.00
Async Read	565,755	1,342.66	37,081,845,720	44.41	1.63
Seek	604,402	253.79		8.39	0.31
Write	40,336	1,419.09	2,475,867,865	46.93	1.72
Flush	49	0.16		0.01	0.00
Close	14	3.39		0.11	0.00
All I/O	1,211,210	3,023.58	39,557,789,081	100.00	3.67

Table 15: I/O Summary of Prefetch version of LARGE : 4 processors

Buffer Size	Original		PASSION		Prefetch	
(Bytes)	total time (seconds)	I/O time (seconds)	total time (seconds)	I/O time (seconds)	total time (seconds)	I/O time (seconds)
64K	947.69	397.05	727.40	196.43	644.68	23.8
128K	903.23	365.57	722.90	186.67	611.31	16.65
256K	901.85	364.69	682.98	141.68	607.85	11.82

Table 16: Execution(right) and I/O times(left) for different buffer sizes of SMALL

Striping factor	Original I/O	PASSION I/O	Prefetch	Striping factor	Original I/O	PASSION I/O	Prefetch
12	0.01	0.05	0.004	12	0.03	0.01	0.01
16	0.053	0.0216	0.006	16	0.024	0.006	0.014

Table 17: Average Read (left) and Write (right) times of SMALL

Striping factor	Original I/O	PASSION I/O	Prefetch	Striping factor	Original I/O	PASSION I/O	Prefetch
12	947.69	727.40	644.68	12	397.05	196.43	23.8
16	745.44	621.29	643.18	16	211.3	88.3	30.19

Table 18: Execution times (left) and I/O times (right) of SMALL : Varying stripe factors

Striping Unit	Original I/O	PASSION I/O	Prefetch	Striping Unit	Original I/O	PASSION I/O	Prefetch
32K	919.67	728.10	647.45	32K	391.43	188.44	25.53
64K	947.69	727.40	644.68	64K	397.05	196.43	23.8
128K	897.11	749.91	650.19	128K	370.36	212.34	26.58

Table 19: Execution times (left) and I/O times (right) of SMALL : Varying stripe units

```
COMPUTE integrals
WRITE integrals into file
LOOP until converges
READ integrals from file
  do some computation
end LOOP
```

Figure 1: Sketch of I/O Activity in HF

IEEE COMPUTER SOCIETY

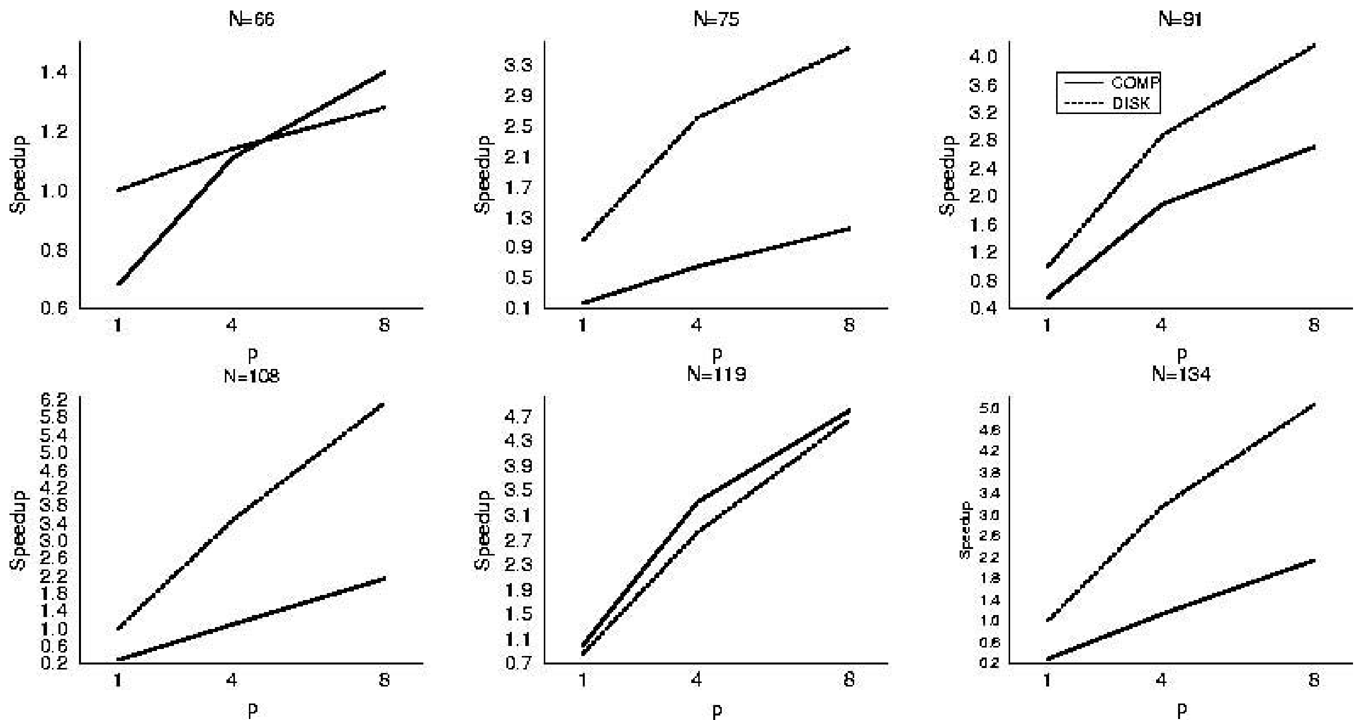


Figure 2: Hartree-Fock Speedups for COMP Vs. DISK versions (A) $\mathcal{N}$=66. (B) $\mathcal{N}$=75. (C) $\mathcal{N}$=91. (D) $\mathcal{N}$ =108. (E) $\mathcal{N}$=119. (F) $\mathcal{N}$=134. (p - number of processors)

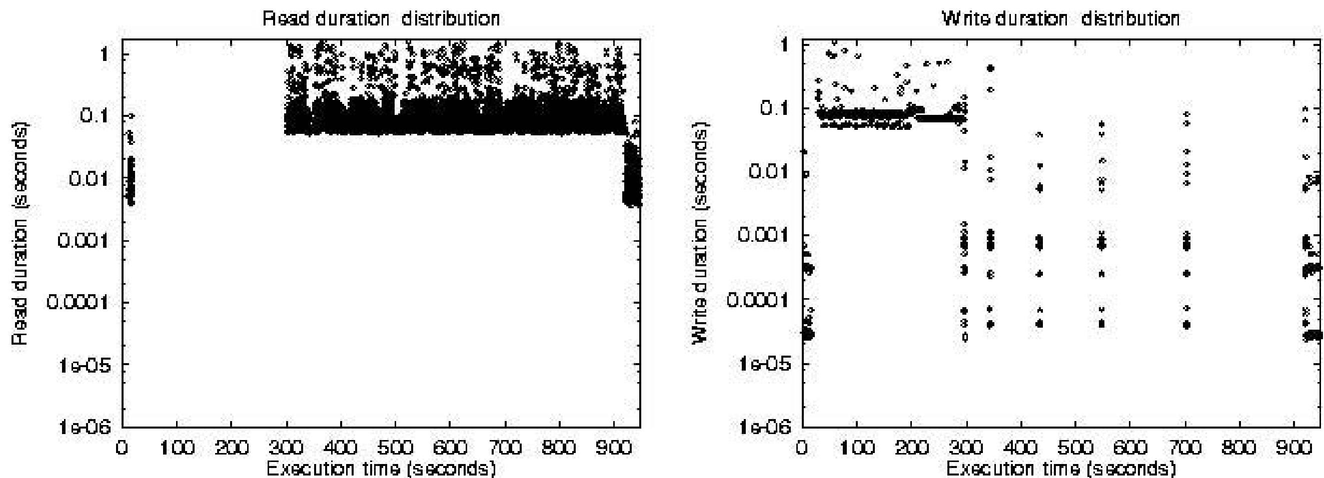


Figure 3: Read and Write Operation Durations of the Original version of SMALL.

IEEE COMPUTER SOCIETY

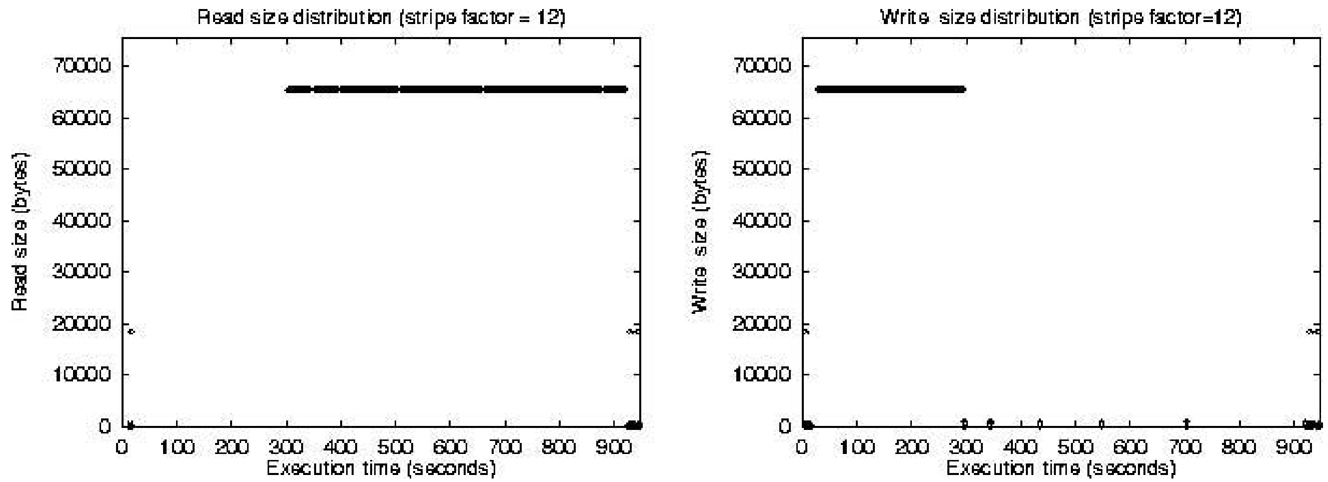


Figure 4: Read and Write Sizes on SMALL.

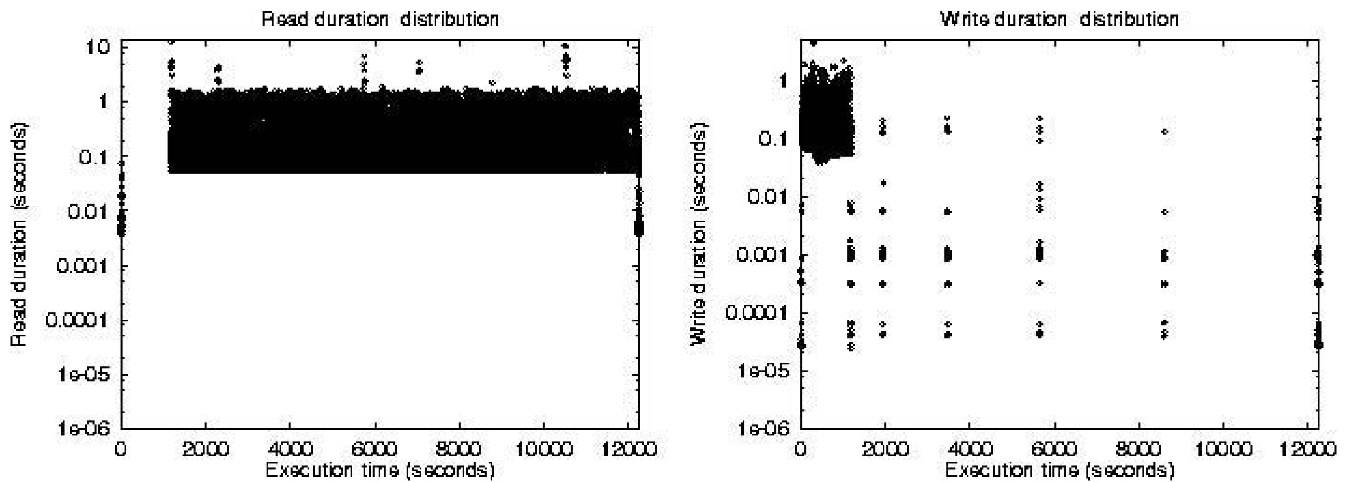


Figure 5: Read and Write Operation Durations of the Original version of MEDIUM

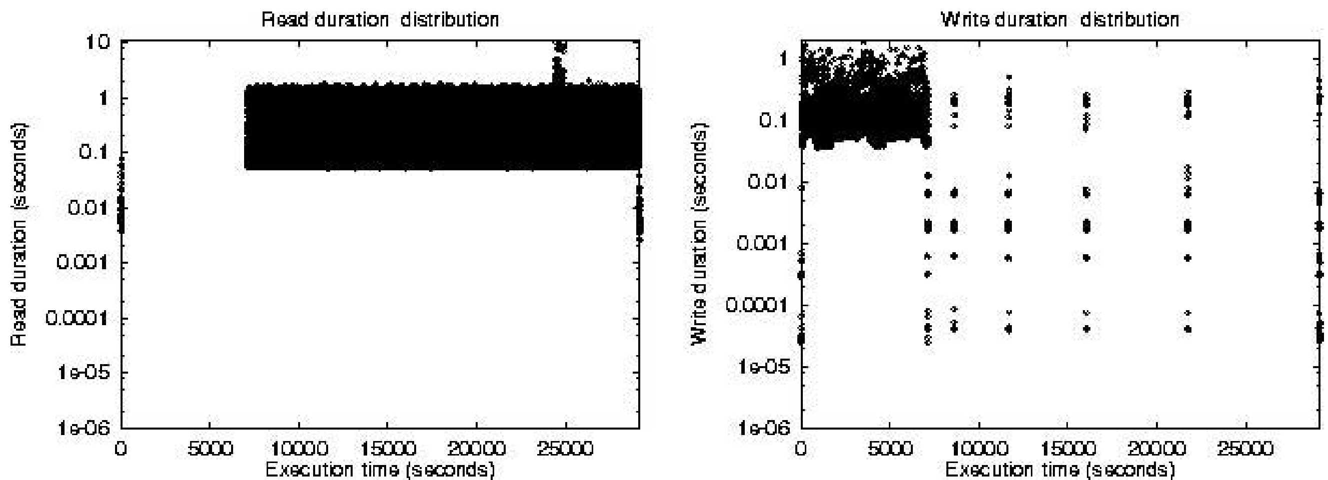
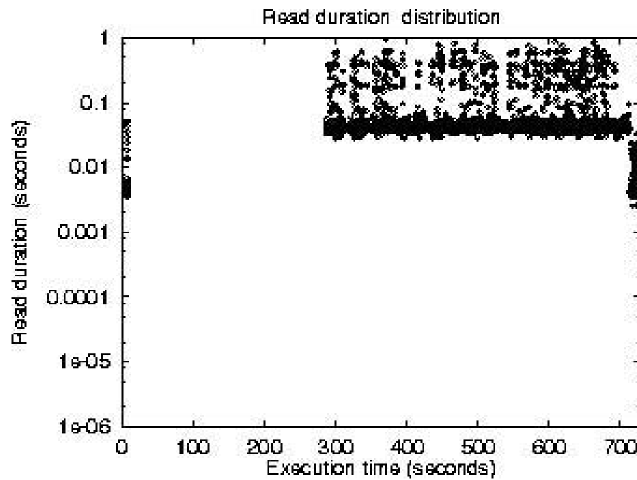


Figure 6: Read and Write Operation Durations of the Original version of LARGE

IEEE COMPUTER SOCIETY



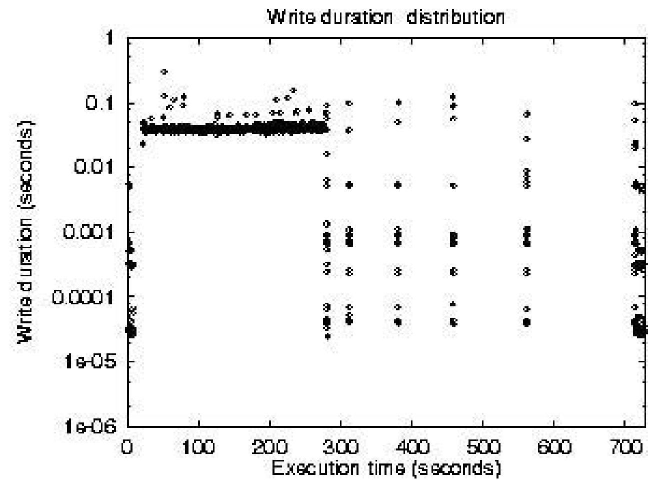
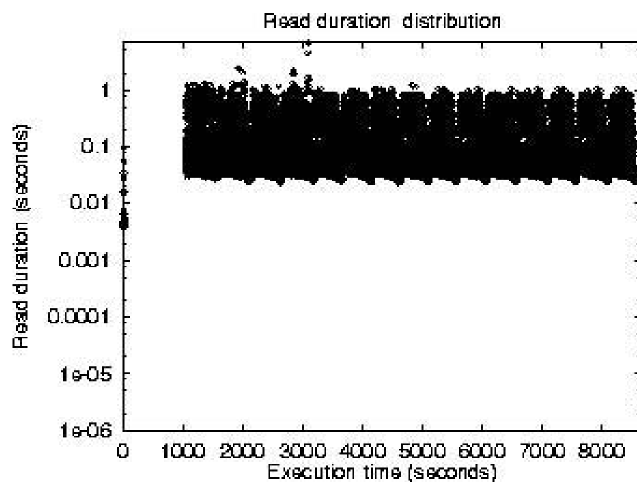


Figure 7: Read and Write Operation Durations of PASSION version of `SMALL`



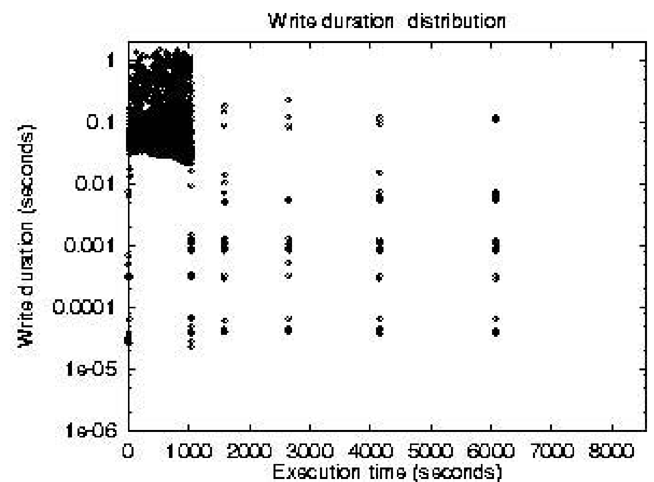
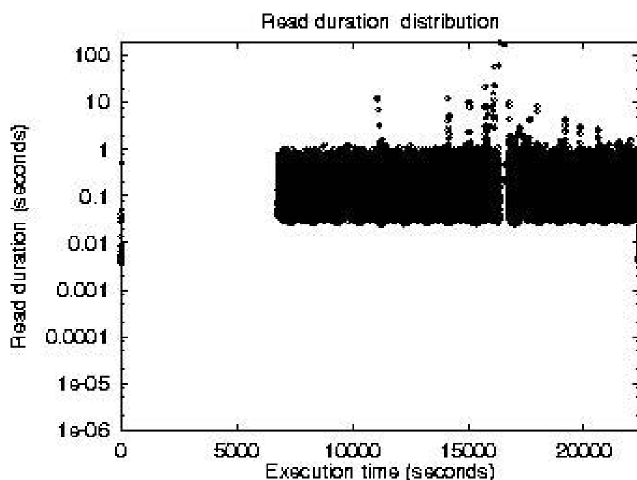


Figure 8: Read and Write Operation Durations of PASSION version of `MEDIUM`



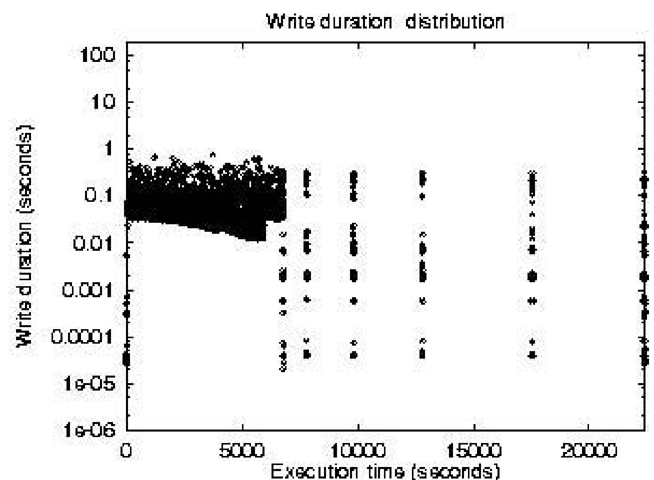


Figure 9: Read and Write Operation Durations of PASSION version of `LARGE`

IEEE COMPUTER SOCIETY

```
iteration: if (current-record == 1) then prefetch(buffer1)
                    wait(buffer1)
                    buffer2 = buffer1
                    increment current-record
                      prefetch(buffer1)
                         -
                         -
                         -
                      process buffer2
                         -
                         -
                         -
                    goto iteration
```

Figure 10: General Outline of Prefetching

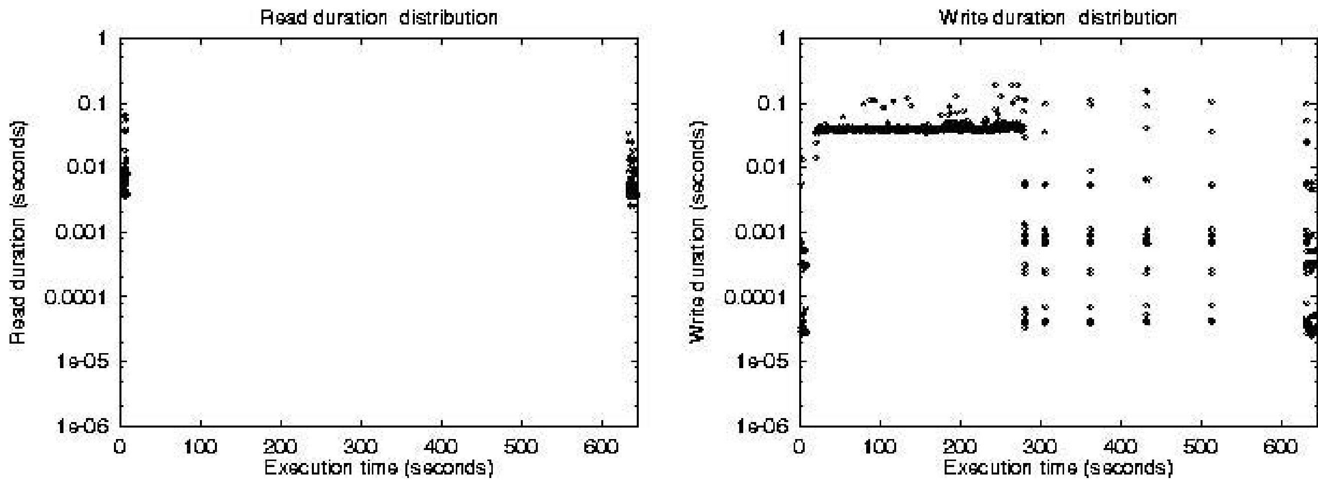


Figure 11: Read and Write Operation Durations of Prefetch version of SMALL

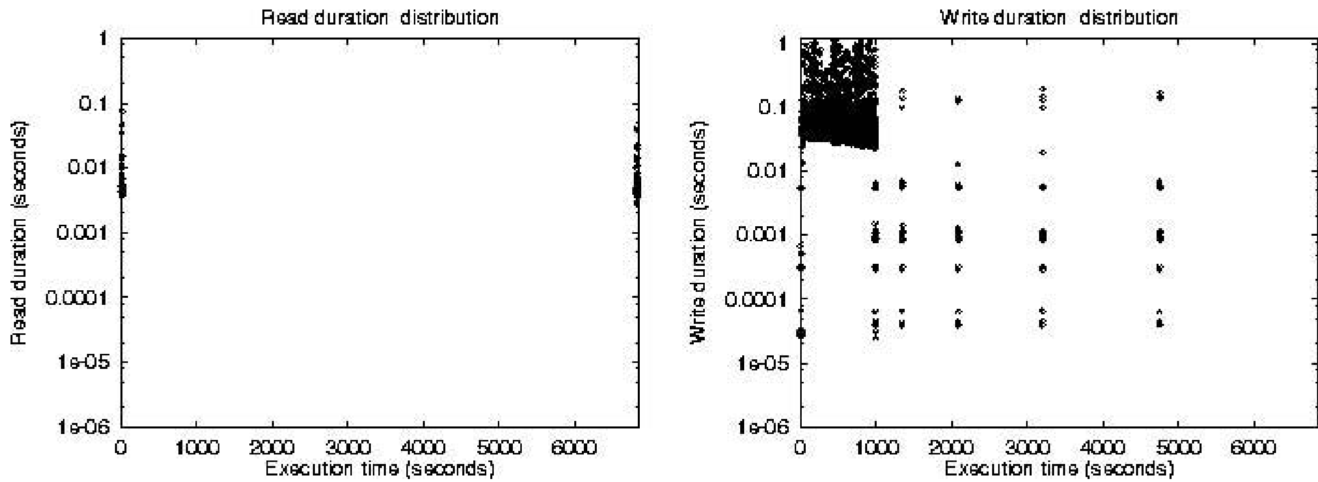


Figure 12: Read and Write Operation Durations of Prefetch version of MEDIUM

IEEE COMPUTER SOCIETY

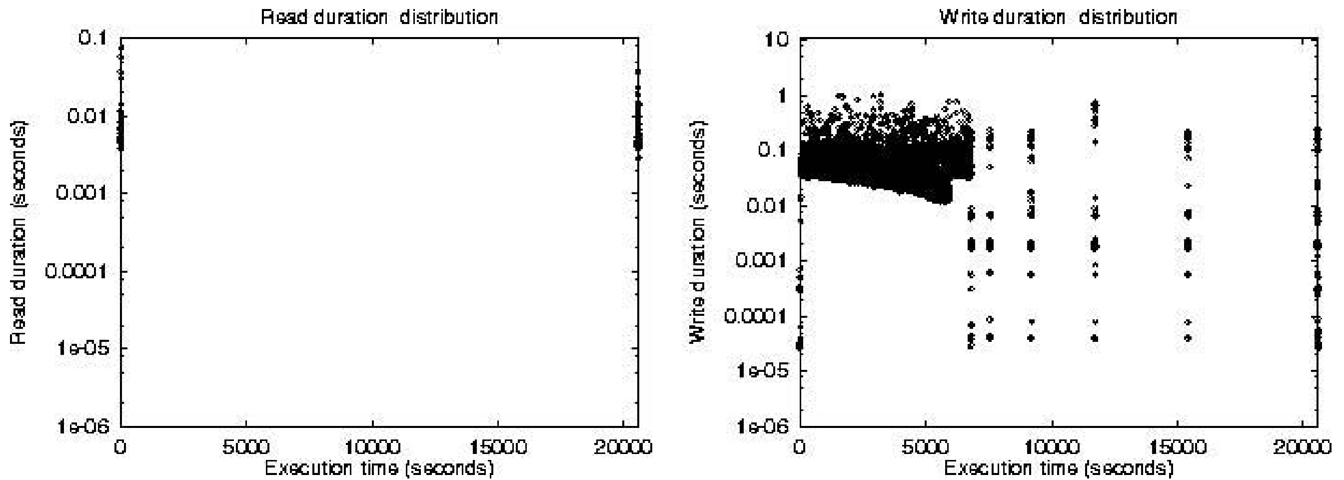


Figure 13: Read and Write Operation Durations of Prefetch version of LARGE

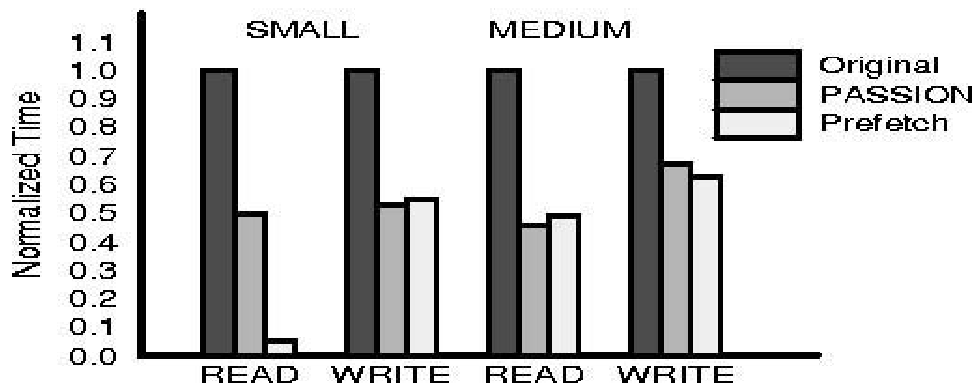


Figure 14: Normalized Average Read and Write Durations for SMALL and MEDIUM

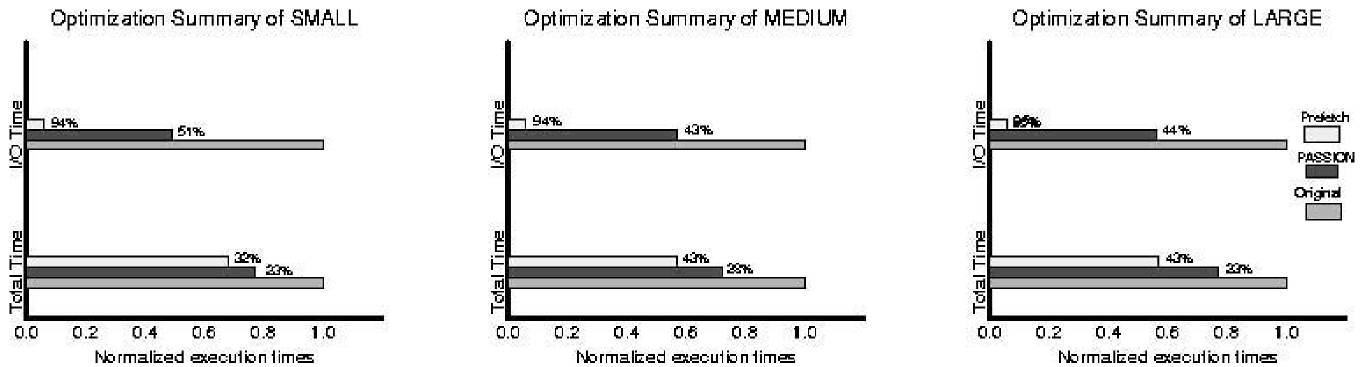


Figure 15: Performance Summary of PASSION and Prefetch for SMALL, MEDIUM and LARGE

IEEE COMPUTER SOCIETY

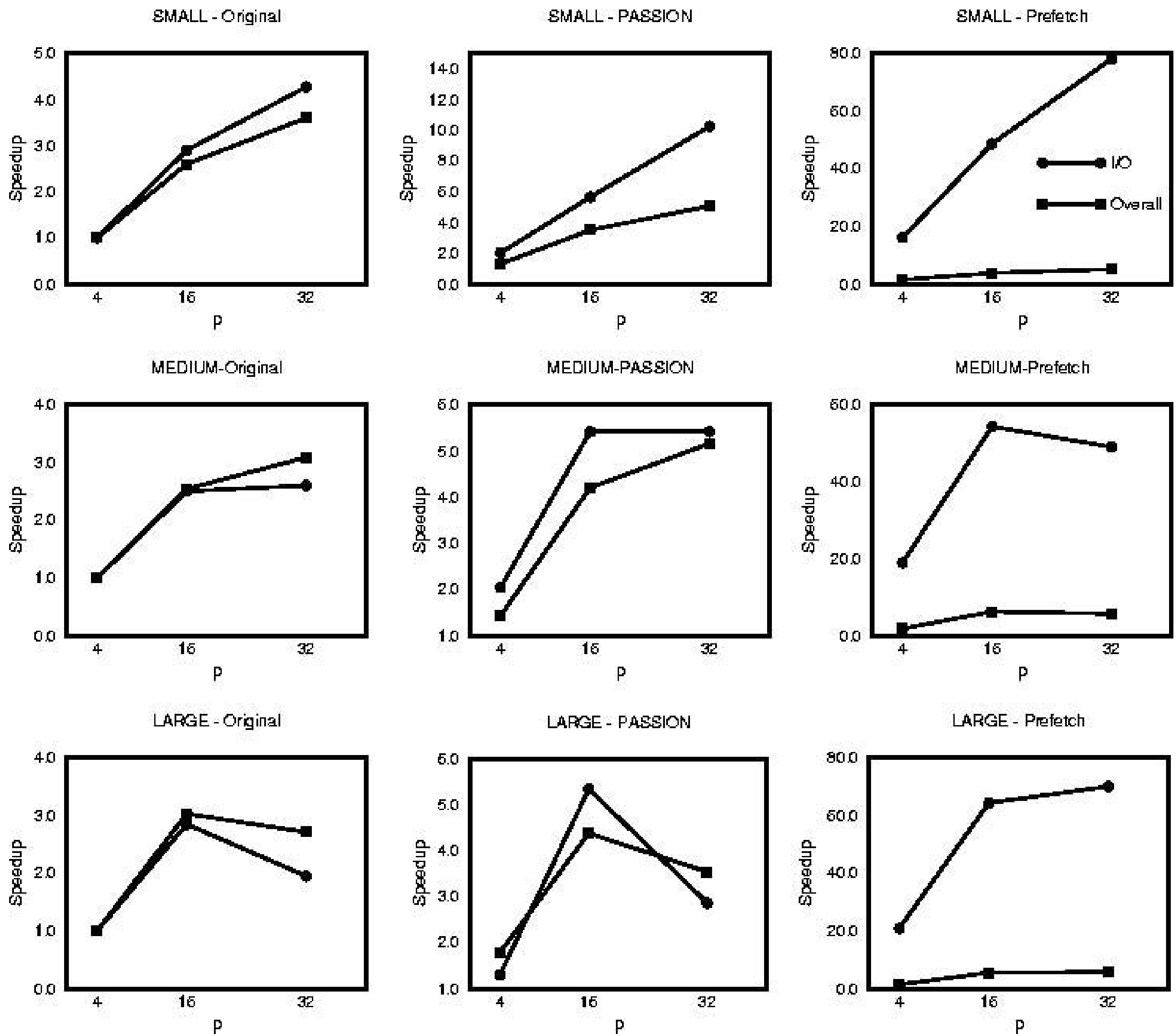


Figure 16: Total and I/O speedups of the three versions for SMALL, MEDIUM and LARGE (p - number of processors)

0-89791-985-8/97 $ 17.00 © 1997 IEEE
IEEE COMPUTER SOCIETY

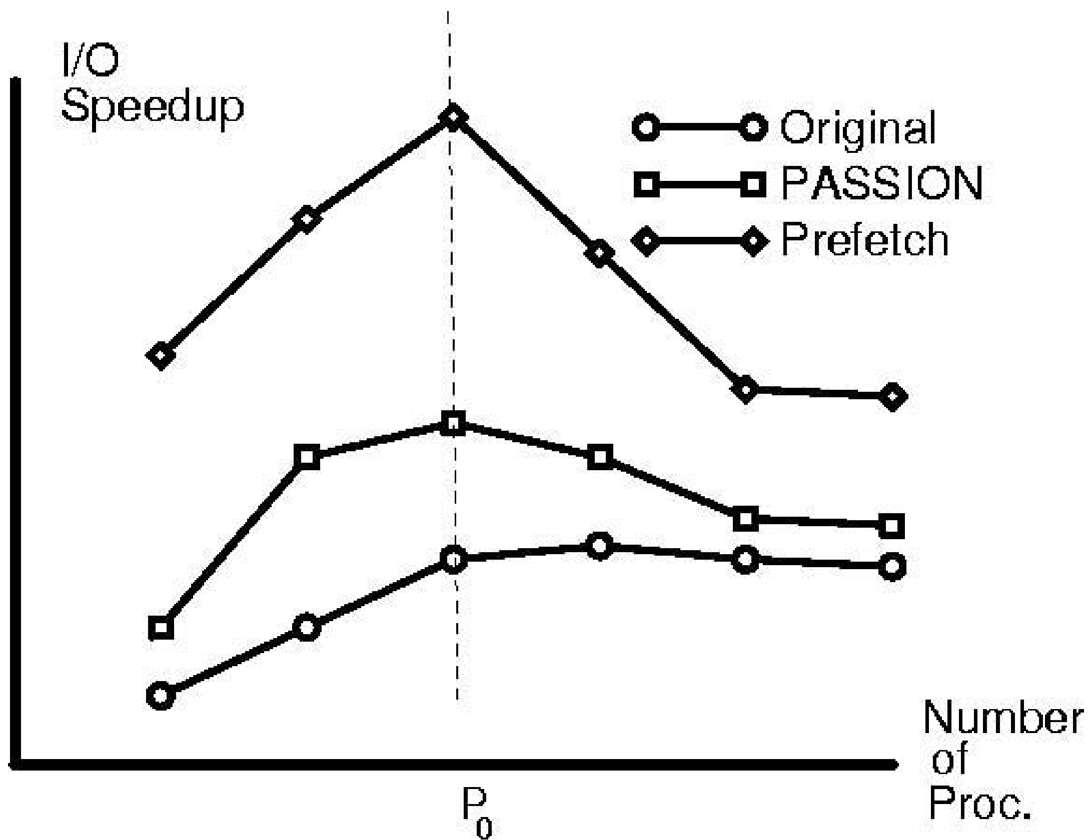


Figure 17: I/O speedup curves for Original, PASSION and Prefetch

IEEE COMPUTER SOCIETY

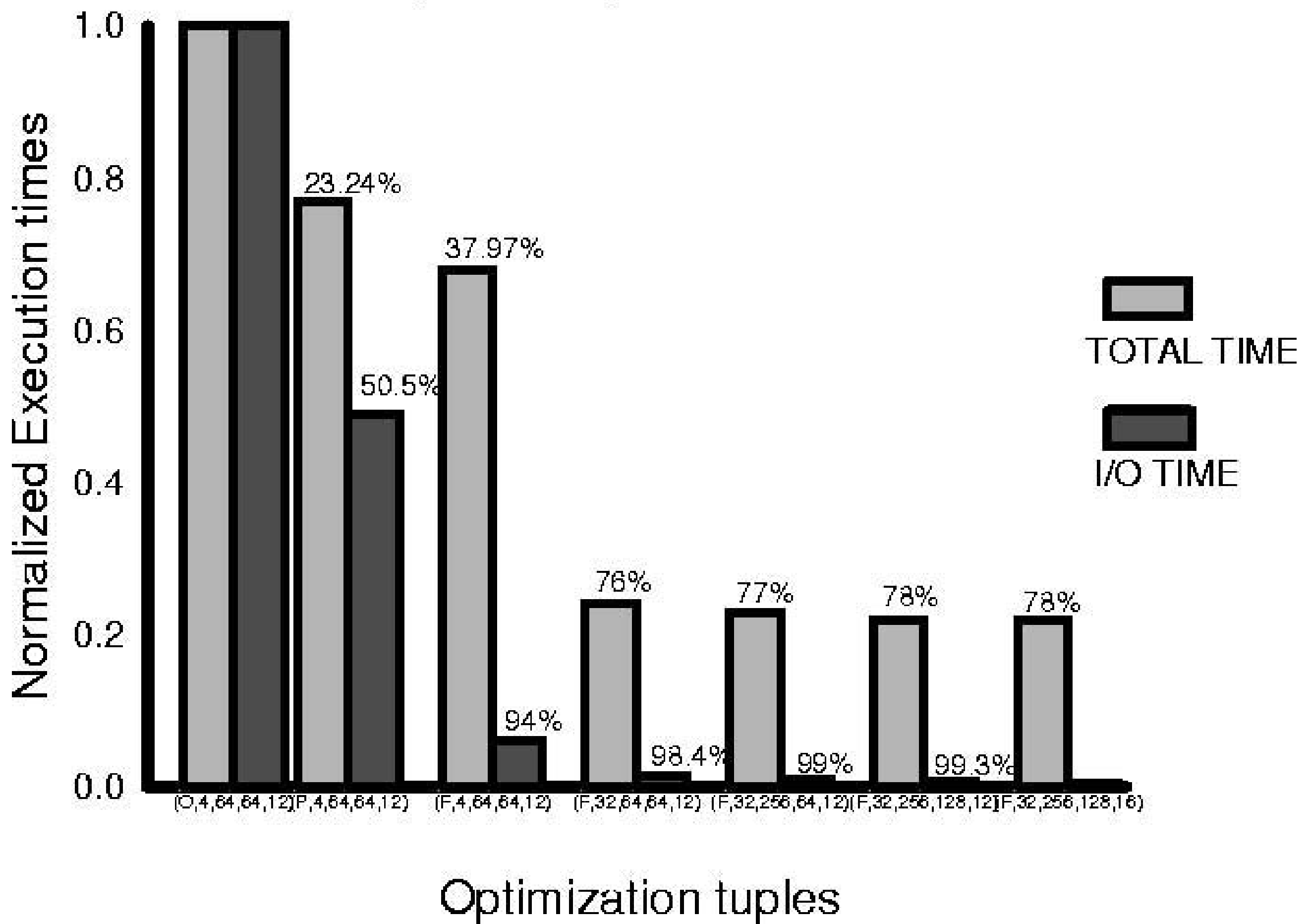


Figure 18: Impact of Optimizations on SMALL

...(PFS)

PFS performs *striping*, that is partitioning of data into equal-sized chunks, each of which is interleaved onto a fixed number of storage areas in a round-robin fashion. *Stripe unit* is the unit of data interleaving and *stripe* is a group of logically contiguous stripe units that are stored in separate I/O nodes. The number of the stripe units in a stripe is called *stripe factor*.

Meenakshi A. Kandaswamy graduated with B.E. (Honors) in Computer Science and Engineering from Regional Engineering College, Trichy, India in 1989 and with M.S. in Computer Science from Syracuse University in 1995. She is currently a Ph.D. candidate in the Department of Electrical Engineering and Computer Science at Syracuse University. She was also a summer intern in the areas of parallel compilers and parallel I/O during the summers of 1993 and 1994 respectively at Intel Supercomputer Systems Division, Beaverton, Oregon. Her research interests include high-performance I/O, parallel applications, multiprocessor file systems and memory hierarchies.

Mahmut T. Kandemir received his B.S. in 1988 and his M.S. in 1992, both in Computer Engineering from Technical University of Istanbul, Turkey. He is currently a Ph.D. candidate in the department of Electrical Engineering and Computer Science at Syracuse University. His research interests are parallelizing compilers and optimization of out-of-core computations.

Alok N. Choudhary received his Ph.D. from University of Illinois, Urbana-Champaign, in Electrical and Computer Engineering, in 1989, M.S. from University of Massachusetts, Amherst, in 1986 and B.E.(Hons.) from Birla Institute of Technology and Science, Pilani, India in 1982. He has been an associate professor in the Electrical and Computer Engineering Department at Northwestern University since September, 1996. From 1993 to 1996 he was an associate professor in the ECE department at Syracuse University and from 1989 to 1993 he was an assistant professor in the same

IEEE COMPUTER SOCIETY

department. He received the National Science Foundation's Young Investigator Award in 1993 (1993-1999). He has also received an IEEE Engineering Foundation award, an IBM Faculty Development award and an Intel Research Council award. He has published extensively in various journals and conferences in the above areas. He has also written a book and several book chapters on the above topics. He is an editor of the Journal of Parallel and Distributed Computing and has served as a guest editor for IEEE Computer and IEEE Parallel and Distributed Technology. He is a member of the IEEE Computer Society and the ACM. He has also been a visiting researcher at Intel and IBM.

David E. Bernholdt received his BS from the University of Illinois at Urbana-Champaign (1986) and his PhD from the University of Florida (1993), both in Chemistry. He was a postdoctoral fellow at the Pacific Northwest National Laboratory (PNNL), where he was part of the development team for the NWChem parallel computational chemistry package. In 1995, he was awarded the Alex G. Nason Fellowship at the Northeast Parallel Architectures Center (NPAC) at Syracuse University, where he presently holds the positions of Research Scientist in NPAC and Research Assistant Professor of Chemistry. Bernholdt's research emphasizes the efficient use of massively parallel processors (MPPs) in chemistry and the development of new computational methods which provide better performance for the large molecular simulations which MPPs make possible. He also continues his affiliation with PNNL as a member of the NWChem development team.

0-89791-985-8/97 $ 17.00 © 1997 IEEE

IEEE COMPUTER SOCIETY