# PERFORMANCE ATTRIBUTES FOR CODE AND WORKLOAD ANALYSIS ON CRAY X-MP AND Y-MP SYSTEMS


## George Delic

MARTIN MARIETTA TECHNICAL SERVICES, INC.
COMPUTATIONAL SCIENCE SERVICES
NATIONAL ENVIRONMENTAL SUPERCOMPUTING CENTER
135 WASHINGTON AVENUE
BAY CITY, MICHIGAN 48708



## Summary

A Performance Analysis Tools (PAT) report implementing hpm and perftrace software, installed under UNICOS on CRAY X-MP and Y-MP systems, was designed at the Ohio Supercomputer Center. The report presents a concise summary of code performance characteristics using 28 performance attributes grouped into four categories: operations and instructions, scalar to vector load balance, memory utilization, and input/output utilization. Demonstrated applications of this performance metric include code performance analysis in a case study, workload analysis for a 24-code sample, and performance comparison of CRAY X-MP and Y-MP processors. The analysis of the case study shows that a precise performance assessment was possible using this metric without the necessity of source code analysis. Measurements with the 24-code sample established realistic performance ranges in the metric and were also used to compare performance of CRAY X-MP and Y-MP models. Although the clock period on the Y-MP was reduced by 29% relative to the X-MP, increases of 52% were observed in instruction fetch rates and vector MFLOPS, with a corresponding rise of 39% in computational intensity.



*The International Journal of Supercomputer Applications,* Volume 7, No. 4, Winter 1993, pp. 304–336.



## Introduction

It was recognized early in the history of the Ohio Supercomputer Center (OSC) that there was a need to educate users on how to enhance performance characteristics of individual Fortran codes in a supercomputer environment. As part of the response to this need, on November 8, 1988, the Performance Analysis Tools (PAT) team was established at the OSC. Subsequently, the OSC established a policy that requires each research group using the supercomputer resource to report annually on code optimization activities. The aim was to build on experience and develop a robust and user-friendly interface to performance data of individual code execution in a supercomputer environment. As a result, in the fall of 1989, a software tool called PAT was installed on the CRAY Y-MP for direct implementation by the OSC user community, and usage has grown steadily. Since PAT's inception, users who have so far participated have all learned new things about codes which they often thought they had understood well, and one aim of this report is to explain the PAT experience.

The performance tools provided under UNICOS by Cray Research, Inc., were originally intended for the use of compiler developers and machine designers, not the end user; and at the beginning of the PAT project, the state of the associated Cray documentation was poor. Cray performance tools provide voluminous numerical data that are difficult to organize and decipher in terms of relevance to coding methodology. Furthermore, the amount of recorded performance data increases enormously as the granularity of the program module under study is refined from whole-code down to subroutine and loop level. Although Cray Research, Inc., has made significant progress in packaging and explaining this veritable mountain of performance data in the last few years, there is ample scope for further development. Therefore, the rationalization and compactification of performance data into a form more easily accessible to the average user is a high priority. In this respect, the PAT project designed software tools and report formats that allow the user to focus on those parts of the code in which activity takes place and thereby maximize the effort at enhancing vector performance. This is important because users new to vector and parallel architectures have the impression that large codes require considerable effort before significant performance enhancements can be achieved. However, after the performance analysis with PAT tools, data are concisely assembled and ordered in rank by subroutine program module, showing that typically a few subroutines account for 90% of the total CPU time. Once these critical subroutines have been identified, the user easily finds the work-intensive loops and is able to progress to more efficient performance by a sequence of loop modifications evaluated with PAT or other UNICOS performance tools. Although there are exceptions related to specific types of code, the experience over two years at OSC has been that users become sensitized to performance issues and respond by enhanced self-learning activity in Fortran coding practice.

The issue of supercomputer performance has received special attention in two important reports. The first of these, entitled "An Agenda for Improved Evaluation of Supercomputer Performance" (Infante, 1986), observed that "there is, at this time, no commonly accepted methodology for the evaluation of performance of supercomputer systems and no standard set of metrics for the representation of performance." This report calls for the "development and distribution of a coherent set of measurement criteria" and states the need to "collect and evaluate research results and experience in the area of supercomputer performance evaluation and attempt to translate it into a broadly usable collection of methods and measures." The report also discusses "quality of life," which implies ease of use and enhancement of the user's productivity in the supercomputer environment, as sharing equal merit with system performance. This need has not diminished with time because it is yet far from fulfilled. It is, in part, as a response to such recommendations that the PAT experience is described here.

The second report resulted from a workshop sponsored by the National Science Foundation (National Science Foundation, Division of Advanced Scientific Computing, 1989). Among other issues, it was noted there that the high-level mapping of an application to the architecture would remain the charge of the human

***"The project designed software tools and report formats that allow the user to focus on those parts of the code in which activity takes place and thereby maximize the effort at enhancing vector performance. This is important because users new to vector and parallel architectures have the impression that large codes require considerable effort before significant performance enhancements can be achieved."***

*"No applications code can be optimized without a precise and reproducible measure of its performance characteristics. Tracking changes in such performance measures, as the code is transformed, is the only means of assessing progress along the path toward well-optimized code."*

expert while only the low-level mapping would be the task of the compiler. For the high-level mapping, performance evaluation was considered critical to predicting the impact on a computation of changes in scale of either the application or the architecture onto which it is mapped. Both these reports make an important distinction between "application characterization" and "workload characterization." The latter relates to system behavior, workload throughput, and capacity planning. The former refers to the effectiveness of resource utilization by the application during the time that it is resident in the supercomputer. Different performance measures are generally required for these two categories, and in this report it is the measure of the application characteristics that is studied and not the system-wide performance. This distinction is possible here because of the way in which performance data are accumulated on CRAY X-MP and Y-MP systems.

The initial successes with the PAT project have led to developments in several directions. First has been the design of a human-machine interface to report individual Fortran code performance on the CRAY Y-MP in a meaningful and understandable format. Second is the accumulation of a statistically significant sample of real-world user code data for evaluation of performance metrics. Third is the exploration of models of code performance on a vector/parallel supercomputer environment. In a study of some of these developments, this report shows how a set of performance attributes (PAs) that measure code performance can be applied in a variety of ways to meet these diverse goals. The first section below defines the choice of PAs and discusses the PAT report format. Then the remainder of this study demonstrates the variety of ways in which the set of PAs may be used for individual applications code performance analysis as described for a case study, application workload analysis as described for a 24-code sample, and performance comparison of different generations of Cray vector processors. The case study and code sample are real-world Fortran code applications, which have been used because there is a consensus that applications packages provide the most accurate picture of system performance for actual problems (Infante, 1986; Berry et al., 1989; Berry, Cybenko, and Larson, 1991; Bradley et al., 1991; Delic, 1991). Conclusions are summarized in the final section, which also mentions an educational methodology addressing code structure and the relationship to performance.

## Performance Attributes and PAT Reports

### USER-FRIENDLY PERFORMANCE METRICS

No applications code can be optimized without a precise and reproducible measure of its performance characteristics. Tracking changes in such performance measures, as the code is transformed, is the only means of assessing progress along the path toward well-optimized vector code. Once on this path, the user needs guidelines as to what is good or bad performance, as determined by some performance metric, and what are achievable goals, as shown by acceptable values of the metric. The present study proposes a performance metric constituted of the PAs defined here. This metric with the associated report format, when presented to end users at the OSC, has motivated them to discover new paths to better performance through experimentation with optimization methods using the PAT tools described here.

Commonly used performance measures such as the floating-point (fp) operation rate or CPU execution time alone are insufficient as performance metrics because "supercomputer performance is too complex to be fully characterized by a single figure of merit" (Infante, 1986). Nevertheless, all supercomputer manufacturers quote the theoretical maximum fp operation rate achievable with their architecture as millions of fp operations per second (MFLOPS). This was introduced as a machine-independent performance measure for numerically intensive computation by McMahon at Lawrence Livermore National Laboratory in 1970 (McMahon, 1986). Because of its widespread use, MFLOPS may be used, for the sake of this argument, to analyze what is currently achieved at different supercomputer centers. For one processor of the CRAY X-MP the peak rate is 234 MFLOPS, and for one processor of the CRAY Y-MP it is 333 MFLOPS. However, the mean MFLOPS delivered by a sample of applications executing on a given architecture is typically a small fraction of this maximum. This has been confirmed, for example, by McMahon (1986), who reports a mean rate of 24.4 MFLOPS on a single-processor X-MP for one year (1985) of production time for applications at Lawrence Livermore National Laboratory. Other studies on the X-MP, over extended periods, report overall performance rates of 26 MFLOPS (West, 1989), 30 MFLOPS (Williams, Myers, and Koskela, 1990), and a (harmonic) mean of 28.3 MFLOPS (Bradley et al., 1991). The sample of 24 codes presented below showed a mean of 42 MFLOPS on a single-processor X-MP. Such measurements readily confirm the conclusion that "the gap between peak performance potential of a machine and that actually achieved is growing principally due to our inability to comprehend the system complexity" (National Science Foundation, Division of Advanced Scientific Computing, 1989).

Usually, the poor performance is due to a preponderance of scalar operations in the code and poor utilization of the vector architecture. A code is said to be well vectorized if it efficiently utilizes the vector architecture of a supercomputer. Conversely, a code is not well vectorized if it does not do so but, rather, performs the major part of the numerically intensive work in scalar mode. The requirement for good vector code provides the motivation for an educational methodology (G. Delic, unpublished report, 1993) that teaches the supercomputer user how to improve vector performance and also parallel performance on multiple CPUs. The combined effect of achieving even partial success in both objectives is a performance enhancement by a factor typically in the range of 10 to 100. In response to such challenges, the PAT tools have been developed to measure vector and scalar performance on a single processor of the CRAY X-MP or Y-MP. All performance measurements discussed here are for a single CPU because some UNICOS performance tools that are used do not function for multitasked (parallel) code. The PAT tools provide a quantitative assessment of performance for an individual application in a concise report showing whole-code performance and also performance of individual subroutines up to a cumulative total of approximately 90% of the execute CPU time.

The PAT report structure currently consists of three levels of machine-produced reports. The PAT Machine Summary (PMS) report, a machine-produced summary of one to three pages for the code under analysis, is based on the longer PAT Machine Report (PMR) from the next level down. The PMR report is a concatenation of filtered outputs from the following Cray UNICOS performance analysis tools: ja, jumptrace, hpm (groups 0 to 3), and perftrace (groups 0 to 3). A brief description of the Cray hardware performance monitor that is used by the hpm and perftrace UNICOS commands is given below. The last PAT report, PAT Machine Listing (PML), is a cft77 compiler listing with the loopmark option enabled for all subroutines, which account for approximately 90% of the CPU execution time of the code under analysis. Each of the PMR, PMS, and PML reports is invoked with a command of the same name, and man-pages have been installed explaining the commands and showing examples of usage.

The PML listing together with the PMS report has proved to be sufficient for the user to localize where code improvement activity is best directed and measure performance improvement quantitatively by comparison with the distribution of a sample of codes (given below). The PMS report gives a concise overview of code characteristics based on 28 PAs grouped as 12 for operations and instructions; 6 for scalar/vector load balance; 8 for memory utilization; and 2 for I/O utilization. The PAs for I/O show I/O references per CPU memory reference and the percentage of I/O references having conflicts. However, the hpm I/O memory reference statistic on which they are based includes any other simultaneous system activity. Therefore, the two PAs for I/O are not true indicators of I/O performance of the user code and will not be discussed further here. With the advent of UNICOS 6.0, a much more comprehensive assessment of I/O performance is obtainable through the utilization of the procstat and procrpt UNICOS tools (Cray Research, Inc., 1991a).

## THE HARDWARE PERFORMANCE MONITOR

The hardware performance monitor (hpm) on CRAY X-MP and Y-MP systems accumulates performance data while the user application executes and does not interfere with the events that are monitored. All performance events reported by the hpm are those monitored only while the CPU has entered user mode, and when it returns to the operating system the collection of performance data is stopped. The monitoring activity consists of incrementing each of the eight performance counters by the number and type of event recorded within each CP. At the end of the application execution, a report based on the accumulated data in the hpm counters is compiled in one of four groups. The first group (g0) reports fp operations, instructions issued or held, instruction buffer fetches, I/O, and CPU memory references. The second group (g1) reports on hold issue conditions sorted by hardware resource. These occur when instructions are held for execution, usually because a resource is unavailable. The third group (g2) reports on memory reference types and also gives measures of the extent of conflicts. The fourth and final group (g3) reports instructions categorized by type and also vector operations. In all four groups, values are reported as absolute magnitudes, percentages, or rates, but a separate execution of the code is required for each group. This is a hardware-related problem that is only resolved for the C-90, which reports all four groups in a single execution. However, on X and Y systems, the user may choose one group and invoke hpm (Larson, 1986) or perftrace (Larson and Lutz, 1985) only for that group of performance data.

The hpm tool does not add an overhead cost to the application but reports only whole-code performance, which is an average over all subroutines. To obtain performance at the level of a subroutine program module, or finer granularity, the perftrace tool is invoked. This UNICOS performance tool does generate a CPU overhead cost and also requires compilation with the flowtrace option enabled (Cray Research, Inc., 1991a). But no source modification is required if the smallest traced program modules are subroutines. If finer granularity is required, such as individual loop performance, then, under UNICOS 6.0, calls to the flowmark tool must be inserted into the source. The perftrace tool uses flowtrace to activate at subprogram (or flowmark) entry and exit points, where the hpm counters are read by the perftrace library routines. Estimates of overheads are subtracted from the measured statistics at entry and exit points because they include the cost of executing perftrace. This means that performance data reported by perftrace for subroutines requiring less than 50 μsec per call are unreliable. At the end of the execution, a report tool, perfview, may be applied to present any one of a number of performance reports using a data file generated by perftrace during execution. One such option is to choose a format (for each subroutine) exactly in the same form as the hpm report for the whole-code performance. A detailed description of the usage of these tools and examples of output may be found in the UNICOS Performance Utilities Reference Manual (Cray Research, Inc., 1989, 1991a), in which Appendix B presents a description of how to interpret performance statistics as reported by hpm. However, the output examples shown there are for UNICOS 5.1 on the X-MP. Consequently, minor differences should be noted in g0, g2, and g3 when hpm is used under UNICOS 6.0.

In the study of fine-grained loop performance on the CRAY Y-MP, an alternative approach has been to examine the performance of the instructions generated by the compiler dynamically as the code executes and allow the user-selected options for different reports. This is done by application of a CRAY Y-MP CPU simulator (Bradley and Larson, 1991) that "executes" instructions obtained from preprocessing of the Fortran source and the Cray Assembler Language file of the cft77 compilation. The simulation does not model activities such as memory conflicts, subroutine calls, or instruction buffer fetches but, nevertheless, gives good accuracy. The relationship between the CRAY X-MP architecture and instruction execution at the clock period level has been described in detail (Robbins and

***"All performance events reported by the hardware performance monitor are those monitored only while the CPU has entered user mode, and when it returns to the operating system the collection of performance data is stopped. The monitoring activity consists of incrementing each of eight performance counters by number and type."***

**Table 1**
**Performance Attributes for Operations and Instructions PA_OI_n**

| n | Description | hpm Location | Value[a] |
|---|---|---|---|
| 0 | Execution time in CPs converted to seconds | hpm g0, Col. 2, CP executing | CP executing |
| 1 | Total no. of MFLOPs on scalar and vector registers and FUs | hpm g0, Col. 1, Floating ops/CPU second | Total MFLOPS |
| 2 | No. of MFLOPS performed only on vector registers and FUs | hpm g3, Col. 1, Floating ops/CPU second | Vector MFLOPS |
| 3 | Total no. of instructions issued in millions per second | hpm g0, Col. 1, Million inst./sec (MIPS) | MIPS |
| 4 | Total no. of instruction buffer fetches in millions per second | hpm g0, Col. 1, Inst. buffer fetches/sec | Inst. buffer fetches/sec |
| 5 | Total no. of jump/special instructions in millions per second | hpm g3, Col. 2, actual inst. | No. of (000–017) jump/special instructions |
| 6 | Avg. vector length for vector fp operations | hpm g3, Col. 3, Vector fp avg. VL | Vector fp avg. VL |
| 7 | No. of CPs for which one or more hold issue conditions occurred in millions per second | hpm g0, Col. 2, CP holding issue | No. of CPs holding issue |
| 8 | Avg. no. of CPs between each instruction | hpm g0, Col. 1, Avg. clock periods/inst. | Avg. CPs per instruction |
| 9 | Avg. no. of CPs between each operation | hpm g0, Col. 2, Total no. in all counters | $\dfrac{\text{CP executing}}{\text{fp (adds + multiplies + reciprocals)}}$ |
| 10 | Blank NIP time as a fraction of the total | hpm g0, Col. 2, Total no. in all counters | $\dfrac{\text{CP executing} - \text{Inst.} - \text{CP holding issue}}{\text{CP executing}}$ |
| 11 | Total no. of vector integer and logical operations in millions per second | hpm g3, Col. 1, Vector integer & logical operations | [Y]: VILops |
| 12 | Avg. vector length for vector integer and logical operations | hpm g3, Col. 3, VIL avg. VL | [Y]: Vector integer & logical operations avg. VL |

[a] Brackets (where they appear) show the processor to which the value applies. FU = functional unit; NIP = next instruction parcel.

Robbins, 1990), providing a valuable resource for specialists. A more general survey of the architectural innovations is given by Chen et al. (1988).

## PERFORMANCE ATTRIBUTES FOR OPERATIONS AND INSTRUCTIONS (PA_OI_n)

The hpm data are easier to assimilate when grouped into the 12 key PAs for operations and instructions (PA_OI_n) listed in Table 1. For each PA, Table 1 shows the naming convention (n) used, a brief description, where the data originate from (hpm location), and the value. Features that are specific to CRAY X-MP or Y-MP models and are not shared by both are denoted by [X] or [Y], respectively. Obviously, some PAs are straightforward transcriptions from hpm (or perftrace), but two involve a calculation using the data from UNICOS performance tools as shown by an expression in the Value column of Table 1. For the user, the usefulness of this information is strongly dependent on how this information is presented, and on the PAT report these PAs for operations and instructions have been grouped into the three segments shown in Table 2, with each containing column ordering of the whole program and subroutines ranked in descending order of CPU time. The first segment shows timing information in three columns, then MFLOPS for scalar plus vector (PA_OI_1) and vector only (PA_OI_2) cases, and vector integer and logical operations, or VILops (PA_OI_11). The second segment shows MIPS (PA_OI_3), average vector lengths for fp (PA_OI_6) and VILops (PA_OI_12), and average CPs per fp operation (PA_OI_9) and instruction

(PA_OI_8). The third segment shows instruction buffer fetches (PA_OI_4), jump/special instructions (PA_OI_5), CP holding issue (PA_OI_7), and blank NIP time (PA_OI_10).

An assessment of the expected range of values for PAs for optimal and nonoptimal code on X-MP or Y-MP systems is discussed below and a case study is presented. However, some general comments on the relevance of these PAs to specific coding problems are appropriate to introduce the value of the less obvious ones in Tables 1 and 2. An example of a specific coding problem is that of determining subroutine calling overheads or code that is predominantly nonsequential for other reasons. One instance of a calling overhead oc-

**Table 2**
**Example of the PMS Report Section with Performance Attributes for Operations and Instructions**

| Routine | CPU Time (sec) | CPU Time (%) | CPU Time Accumulated (%) | Total MFLOPS | Vector MFLOPS | VILops |
|---|---|---|---|---|---|---|
| | PA_OI_0 | | | PA_OI_1 | PA_OI_2 | PA_OI_11 |
| Program | 76.64 | 100.00 | 0.00 | 146.901 | 145.129 | 19.737 |
| FOUR2 | 38.20 | 49.84 | 49.84 | 199.747 | 199.741 | 0.018 |
| FORR | 15.84 | 20.66 | 70.50 | 99.638 | 109.407 | 53.836 |
| ASSRHOP | 11.93 | 15.56 | 86.07 | 63.439 | 72.091 | 29.891 |
| POWER | 6.00 | 7.82 | 93.89 | 64.469 | 58.783 | 36.376 |

| Routine | MIPS | Avg. fp Vector Length | Avg. Vector Integer Logical Vector Length | Avg. CP per fp Operation | Avg. CP per Instruction |
|---|---|---|---|---|---|
| | PA_OI_3 | PA_OI_6 | PA_OI_12 | PA_OI_9 | PA_OI_8 |
| Program | 20.339 | 54.535 | 52.414 | 1.135 | 8.197 |
| FOUR2 | 15.984 | 53.885 | 42.773 | 0.835 | 10.431 |
| FORR | 14.178 | 70.038 | 53.927 | 1.673 | 11.759 |
| ASSRHOP | 27.419 | 41.000 | 49.454 | 2.628 | 6.080 |
| POWER | 43.642 | 51.211 | 51.136 | 2.586 | 3.820 |

| Routine | Instruction Buffer Fetches (million/sec) | Jump/Special Instructions (million/sec) | CP Holding Issue (million/sec) | Blank NIP Tiime (%) |
|---|---|---|---|---|
| | PA_OI_4 | PA_OI_5 | PA_OI_7 | PA_OI_10 |
| Program | 0.04931 | 1.0581 | 143.146 | 1.94% |
| FOUR2 | 0.00782 | 0.6488 | 149.728 | 0.61% |
| FORR | 0.14746 | 0.1500 | 148.474 | 2.44% |
| ASSRHOP | 0.00438 | 2.8360 | 129.581 | 5.83% |
| POWER | 0.00885 | 1.4529 | 100.856 | 13.33% |

curs when a large number of different routines are called from all sections of memory. In this case, the amount of nonsequential code is indicated in g3 by the number of jump/special instructions (PA_OI_5), which are those used to jump out of sequential execution. Loops executing within one to four instruction buffers are optimal, but nonsequential code (with many logic jumps out of instruction buffers) increases the number of instruction buffer fetches from memory, and this reduces computational efficiency (Cray Research, Inc., 1990). This condition will also give a large value ($>1 \times 10^6$) for the g0 counter of instruction buffer fetches per second (PA_OI_4). Efficient utilization of the instruction stack with smaller instruction parcels and predominantly sequential code is essential for good vector performance. When all instructions in the stack have been processed, more are fetched from memory (the acronym NIP denotes next instruction parcel), and this process is an order of magnitude slower than reading the stack. Thus, performance is degraded by excessively nonsequential code as occurs in subroutine calls or logical branches, or in loops requiring more instructions than fit the buffer, because this increases the number of times an instruction parcel is fetched. However, if extended memory addressing (EMA) is enabled, which is the default with UNICOS, then the g3 statistic can be high owing to inclusion of instructions used to load long addresses, although this is not the typical situation.

In a second instance of subroutine calling overhead, one routine that does little computation is called many times. The extent of this overhead may be determined by computing an estimate of blank NIP time (PA_OI_10). This is the number of CPs that are used to jump across instruction buffers, fetch words from memory to load the buffers, and process instructions of more than one parcel in length (Cray Research, Inc., 1990). When measured as a percentage of CP, executing values larger than 20% may represent mostly scalar code. Thus, blank NIP time is a measure of nonuser operations involved in the code execution, and large values are associated with inefficient utilization of computational resources by the user application.

## PERFORMANCE ATTRIBUTES FOR SCALAR TO VECTOR LOAD BALANCE (PA_SV_n)

The hpm data are best assimilated when grouped into the six key PAs for scalar to vector load balance defined in Table 3, which shows the naming convention (n) used, a brief description, where the data originate from (hpm location), and the value. Again, features that are specific to CRAY X-MP or Y-MP models are denoted by [X] or [Y], respectively. In Table 3, four of the six PAs require a calculation using the data from UNICOS performance tools as shown by an expression in the Value column. These PAs are displayed in one segment on the PMS report, as shown in Table 4, with each containing column ordering of the whole program and subroutines ranked in descending order of CPU time. The first three columns are the ratio of scalar to vector fp instructions (PA_SV_1), vector to scalar fp operations (PA_SV_2), and scalar to vector memory instructions (PA_SV_6). The last three columns show hold issue conditions with total number for vector registers (PA_SV_3) and functional units (PA_SV_4) and, finally, the ratio of scalar to vector (PA_SV_5).

These PAs are critical in assessing the scalar to vector load balance in an application code by calculating ratios measuring this balance. Effective utilization of the Cray architecture requires operational work to be performed predominantly in vector mode on fp vector registers and functional units. Codes that do not have such characteristics are said to be poorly vectorized. It may happen that some codes are not candidates for good vector performance either by choice of the computational algorithm or minimal amounts of either integer or fp work. Thus, it is important to first determine the nature of the computational problem and its suitability for a vector architecture and only then determine the relative amount of computational work performed in vector mode.

Hold issue conditions are a normal occurrence in Cray architectures and increase dramatically when there is enhanced simultaneous instruction processing. They are also useful as an indirect measure of use of registers and functional units (FUs) from the ratio of scalar to vector activity. This is measured by the sum of

## Table 3

**Performance Attributes for Scalar to Vector Load Balance PA_SV_n**

| n | Description | hpm Location | Value[a] |
|---|---|---|---|
| 1 | Ratio of scalar to vector fp instructions | hpm g3, Col. 2, actual inst. on both counters | [X]: $\frac{\text{(020–137) scalar}}{\text{(160–174) vector fp}}$ <br> [Y]: $\frac{\text{(020–077) scalar FU}}{\text{(160–174) vector fp}}$ |
| 2 | Ratio of vector to scalar fp operations | hpm g3,g0 Col. 2, g3—actual ops. g0—total no. of all counters | $\frac{\text{g3 vector fp}}{\text{g0 fp (adds + multiplies + reciprocals)} - \text{g3 vector fp}}$ |
| 3 | No. of hold issues on vector registers in millions per second | hpm g1, Col. 2, actual no. of CPs | V-registers |
| 4 | No. of hold issues on vector FUs in millions per second | hpm g1, Col. 2, actual no. of CPs | Vector FUs |
| 5 | Ratio of scalar to vector hold issues | hpm g1, Col. 2, actual no. of CPs in all counters | $\frac{\text{A/FUs + S-registers/FUs}}{\text{V-registers + vector FUs}}$ |
| 6 | Ratio of scalar to vector memory instructions | hpm g3, Col. 2, actual inst. on both counters | [Y]: $\frac{\text{(100–137) scalar memory}}{\text{(176–177) vector load \& store}}$ |

[a] Brackets (where they appear) show the processor to which the value applies. FU = functional unit.

## Table 4

**Example of the PMS Report Section with Performance Attributes for Scalar to Vector Load Balance**

| Routine | Scalar to Vector fp Inst. | Vector to Scalar fp Operations | Scalar to Vector Memory Inst. | Hold Issues: Vector Regs. (million/sec) | Hold Issues: Vector FUs (million/sec) | Hold Issues: Scalar to Vector |
|---|---|---|---|---|---|---|
| | PA_SV_1 | PA_SV_2[a] | PA_SV_6 | PA_SV_3 | PA_SV_4 | PA_SV_5 |
| Program | 4.3703 | 81.9100 | 0.5020 | 65.795 | 74.505 | 0.0819 |
| FOUR2 | 2.3420 | 36228.0233 | 0.0293 | 84.871 | 98.851 | 0.0262 |
| FORR | 5.3858 | −11.2000 | 0.0129 | 63.814 | 55.936 | 0.0510 |
| ASSRHOP | 10.2765 | −8.3326 | 0.0007 | 39.907 | 49.346 | 0.0882 |
| POWER | 20.1320 | 10.3376 | 61.8343 | 10.425 | 13.441 | 2.8397 |

[a] For the CRAY Y-MP, this PA can show negative values in some cases (see discussion in text), and therefore the absolute value should be used.

**Table 5**
**Performance Attributes for Memory Utilization PA_MU_n**

| n | Description | hpm Location | Value[a] |
|---|---|---|---|
| 1 | Avg. no. of memory accesses per fp operation | hpm g0, Col. 2, Total no. in all counters | $\frac{\text{CPU references}}{\text{fp (adds + multiplies + reciprocals)}}$ |
| 2 | Avg. no. of scalar memory accesses per block memory access | hpm g2, Col. 2, actual refs. in both counters | $\frac{\text{Scalar memory refs.}}{\text{Block memory refs.}}$ |
| 3 | Fraction (%) of ([X]) block or ([Y]) CPU memory references resulting in conflicts | hpm g2, Col. 1, % having conflict | [X]: Block memory refs. (% having conflicts)<br>[Y]: CPU memory refs. (% having conflicts) |
| 4 | Ratio of hold issue conditions on scalar to block memory references | hpm g1, Col. 2, actual no. of CPs in both counters | $\frac{\text{Waiting on scalar memory references}}{\text{Waiting on block memory references}}$ |
| 5 | No. of hold issue conditions on block memory references (million/sec) | hpm g1, Col. 2, actual no. of CPs | Waiting on block memory references |
| 6 | Ratio of hold issue conditions on vector operations to block memory references | hpm g1, Col. 2, actual no. of CPs in all counters | $\frac{\text{Waiting on V-registers and vector FUs}}{\text{Waiting on block memory references}}$ |
| 7 | Ratio of CPU memory to block memory refs. | hpm g2, Col. 2, actual refs. in all counters | [X]: $\frac{\text{Vector + scalar memory references}}{\text{Block memory references}}$<br>[Y]: $\frac{\text{CPU memory references}}{\text{Block memory references}}$ |
| 8 | Avg. vector length on vector memory access | hpm g3, Col. 3, Vector memory avg. VL | [X]: Vector memory avg. vector length |
| | Ratio of vector fp to vector memory instructions | hpm g3, Col. 2, actual inst. on both counters | [Y]: $\frac{\text{(160–174) vector fp}}{\text{(176–177) vector load \& store}}$ |

[a] Brackets (where they appear) show the processor to which the value applies.

address (A) and scalar (S) entries in g1 compared with the sum of vector (V) registers and vector FUs. Because enhanced instruction overlap is the goal in supercomputing (Patterson and Hennessy, 1990; Robbins and Robbins, 1990), evidence of chaining of floating-point FUs is important. Each CPU contains computational units consisting of registers and FUs made up of three types for vector, scalar, and address operations with seven independent FUs in the CRAY X-MP and Y-MP. Chaining of FUs yields overlap of operations and therefore enhances performance, but chains of three or more FUs are not easy to construct. This evidence is easily found by inspection of hold issues in hpm g1, shown as PA_SV_3 and PA_SV_4 in Tables 3 and 4. The presence of effective chaining is indicated by large but nearly equal values for hold issue conditions on both vector registers and FUs (PA_SV_3,4). Conversely, if both of these are small, then chaining is inhibited even if the value for vector MFLOPS reported in g3 is significant. The meaning of "large" and "small" in this context is discussed below. However, it was observed that in changing from cft77 3.0 to version 4.0, the balance between the two vector counters changes substantially for the same code without a corresponding change in the MFLOPS rate.

A direct measure of the scalar to vector load balance is comparison of either instructions (PA_SV_1) or operations (PA_SV_2). But the preferred method of assessing vector performance is by comparing operations. However, the number of vector fp instructions in g3 may also include vector logical instructions. This is because the additional vector logical (AVL) unit allows for simultaneous execution of two vector logical operations, but in doing so it uses the fp multiply FU. This usage is included in the g0 and g3 (X-MP) or g3 (Y-MP) fp statistics. In such cases, the g0 MFLOP rate is incorrect, and the g3 average vector length can be larger than 64.

## PERFORMANCE ATTRIBUTES FOR MEMORY UTILIZATION (PA_MU_n)

The hpm data are best assimilated when grouped into eight key PAs for memory utilization as shown in Table 5, which is similar to Tables 1 and 3 with features specific to CRAY X-MP or Y-MP models denoted by [X] or [Y], respectively. In Table 5, six of the eight PAs require a calculation using the data from UNICOS performance tools as shown by an expression in the Value column. In Table 6, these PAs have been grouped into two segments with each containing column ordering of whole-program routines and subroutines ranked in descending order of CPU time. The first segment shows memory references per fp operation (PA_MU_1), the ratio of scalar to block memory references (PA_MU2), memory conflicts (PA_MU_3), and the ratio of vector fp instructions to vector memory instructions (PA_MU_8). In the second segment, the last column shows the ratio of CPU memory references to block memory references (PA_MU_7), and the first three columns show hold issues as a ratio of scalar to block memory (PA_MU_4), vector operations to block memory references (PA_MU_6), and total number of block memory references (PA_MU_5).

Although it is essential to establish the load balance

**Table 6**
**Example of the PMS Report Section with Performance Attributes for Memory Utilization**

| Routine | Memory Refs. per FLOP | Scalar to Block Memory Refs. | CPU Memory Refs. % Conflict | Vector fp Inst. to Vector Load & Store Inst. |
|---|---|---|---|---|
| | PA_MU_1 | PA_MU_2 | PA_MU_3 | PA_MU_8 |
| Program | 0.9497 | 0.0103 | 9.48% | 0.9453 |
| FOUR2 | 0.7749 | 0.0005 | 6.18% | 1.2950 |
| FORR | 1.9242 | 0.0002 | 17.94% | 0.5202 |
| ASSRHOP | 1.0810 | 0.0000 | 13.52% | 0.4238 |
| POWER | 0.4803 | 1.2091 | 42.80% | 4.1880 |

| | Hold Issues | | | |
|---|---|---|---|---|
| Routine | Scalar to Block Memory Refs. | Block Memory Refs. (million/sec) | Vector Operations to Block Memory Refs. | CPU Memory to Block Memory Refs. |
| | PA_MU_4 | PA_MU_5 | PA_MU_6 | PA_MU_7 |
| Program | 0.0257 | 21.512 | 6.5219 | 1.0103 |
| FOUR2 | 0.0021 | 12.451 | 14.7555 | 1.0005 |
| FORR | 0.0062 | 48.740 | 2.4569 | 1.0002 |
| ASSRHOP | 0.0000 | 28.020 | 3.1853 | 1.0000 |
| POWER | 1.7209 | 1.120 | 21.3034 | 2.2091 |

between scalar and vector operations, it is also necessary to determine the load balance between memory and CPU activity on both scalar and vector operations. The time required to extract user data from memory and the frequency of memory traffic can have a dramatic impact on code performance, both in the actual CPU execution time and in the peak sustained MFLOPS rate. A variety of memory wait states or conflict situations may occur during code execution that interfere with chaining and vector performance. As one example, block memory references without conflict are performed in one CP but memory conflicts induce hold-issue conditions of more than one CP. In a memory access, such as a load operation, a CPU is assigned an access priority for a subsection, and a bank select code assigns a CPU to a bank within the same subsection. Then, for a period of 5 CPs, known as the bank busy time, no other access to the same bank is possible. Clearly, two consequent memory instructions that access the same bank in two consecutive clock periods are in conflict. The cause of bank conflicts is related to allocation of memory for arrays where consecutive array elements are stored in consecutive banks and consecutive banks are stored in consecutive sections. On the CRAY Y-MP at the OSC there are 64 million words of memory arranged in 256 banks with 64 banks in a section. If the loop stride through the array is not one, but a power of 2 in the range 16 to 256, a bank conflict occurs, and the consequent variations in memory access times result in slower chaining performance. Bank conflicts may often be reduced by simple changes in the Fortran constructs (Levesque, 1989). Memory conflicts are computed in hpm g2 as the ratio of total conflicts to total memory references to indicate the average conflict rate on memory references over the execution. This ratio is shown on the X-MP as a percentage for scalar, I/O, block and vector memory, and on the Y-MP for I/O and CPU memory references. On the X-MP, vector memory references (load or store to vector registers) and block memory references (load or store to vector and B/T registers) are distinguished but may often be considered as virtually synonymous (Cray Research, Inc., 1991). On the Y-MP, these two types of memory references are not distinguished, and under UNICOS 6.0, average conflicts per reference are reported and not the percentage.

## Code Analysis with the PAT Report: A Case Study

To date, there have been some 1,500 invocations of the PAT tools at the OSC on the CRAY Y-MP under two different versions of the cft77 compiler and UNICOS operating system. No one selected case could be considered typical as each code has its own specific problems. Nevertheless, results for one specific code will be discussed here to demonstrate what can be learned from a PAT report. The code chosen for this purpose is used in a three-dimensional $N$-body model for the gravitational evolution of structure in the expanding universe (Villumsen, 1989). This code is based on a standard particle-mesh methodology and solves Poisson's equation at successive time steps on a cubical grid with fast Fourier transform (FFT) convolution techniques. The equations of motion are integrated with a time-centered leapfrog scheme. A multigrid method is applied to obtain high resolution in a small spatial region. Typically, a calculation may have $256^3$ grid points with $128^3$ particles, and the FFT computation time scales as $N^3\log(N)$ where $N^3$ is the number of grid points. The number of other operations scales linearly with the number of particles.

The PAT report results for this code compiled with cft77 3.1.1 and executed under UNICOS 5.0 on the Y-MP are shown in Tables 2, 4, and 6 corresponding to the respective categories of the last three sections. The value of PAs averaged over all subroutines is given by the entry "program" in all tables, and Table 2 shows that four subroutines, FOUR2, FORR, ASSRHOP, and POWER, account for 94% of the accumulated CPU time. The first segment of Table 2 shows that the vector MFLOPS rate drops from 199.7 to 58.8 over the four routines but that VILops for the last three range from 53.8 to 29.9 with 0 for FOUR2. Therefore, vector performance in ASSRHOP and POWER is significantly less than that in FOUR2 or FORR. On the other hand, FORR, ASSRHOP, and POWER account for nearly half of the CPU time, so, for example, a doubling of the

MFLOP rate in these three routines could reduce the overall CPU time by 25%. The second segment of Table 2 shows that while the average vector lengths for fp and VIL operations are good in all cases, the MIPS rate doubles for ASSRHOP and rises to 43.6 for POWER. Despite a similar MIPS rate, FORR requires twice the number of CPs per FLOP that FOUR2 does while the last two routines require three times as much. The average CP per instruction (or time between instructions) decreases as the MIPS rate increases. At this point, despite good average vector lengths, several questions arise. Why does FORR need double the number of CPs per operation compared with FOUR2? Why are ASSRHOP and POWER such relatively poor performers? Partial answers to these questions may be found in the third segment of Table 2, where a clear and correlated trend can be seen in the jump/special instructions and blank NIP time. Both of these attributes give small values for FOUR2 and FORR, but note that blank NIP time for FORR is four times larger. By contrast, ASSRHOP shows $2.8 \times 10^6$/sec and 5.8% for the respective PAs and POWER shows $1.45 \times 10^6$/sec and 13.3%. The two routines ASSRHOP and POWER clearly show large amounts of nonsequential code with associated increased instruction parcel processing and a consequent loss of performance for user data operations. Instruction buffer fetch rates are small to negligible, and the large number of CPs during which one or more hold issues occurred should be noted.

Table 4 shows why FOUR2 is the best vector fp performer: the ratio of vector to scalar operations is large, and the ratio of scalar to vector instructions is small in contrast to the other three routines. The negative sign for PA_SV_2 should be ignored (see the discussion in the next section). The largest number of scalar instructions is in POWER, which is also accessing predominantly scalar memory. The hold issues on vector registers and FUs show that chaining is very effective in FOUR2, less effective in FORR and ASSRHOP, and not effective in POWER. The ratio of scalar to vector hold issues confirms the predominantly scalar character of POWER. An intermediate conclusion is that good chaining on vector processing elements leads to high vector MFLOPS rates in routine FOUR2. The

***"The PAT performance attributes can be used not only to investigate the path to improved performance for one code, but also to investigate the spectrum of performance achieved with a sample of codes. Such an analysis is required to establish basic statistical information on realistic and achievable values for minimum, maximum, and mean values for each performance attribute."***

*"During code execution, various types of instructions are issued for operations that are not directly related to user data manipulation. Therefore, it seems appropriate to refer to these collectively as an execution penalty because the user data receive no direct arithmetic operational benefits from them."*

converse is the case with an increased scalar instruction rate for operational work and memory activity in routine POWER. However, in the latter case, as noted for segment three of Table 2, a significant amount of the scalar work is overhead in instruction processing.

Table 6 shows the memory activity in detail. The first column of segment 1 shows memory references per fp operation, and the value of 0.77 indicates that FOUR2 is CPU bound, while a value of 1.9 shows FORR is memory bound. The load balance of scalar to vector memory is measured in the second column, which clearly shows that with the exception of POWER the routines are accessing vector memory almost exclusively. The third column shows an increasing incidence of memory conflicts with the largest value of 42% for POWER, which is a cause for concern. The last column shows the number of vector instructions issued per vector memory reference, and a value of 1.3 confirms the CPU-intensive character of FOUR2. The two routines FORR and ASSRHOP issue twice as many memory instructions as fp instructions, while POWER appears to be CPU intensive on vector instructions, but (from Table 4) is known to be dominated by scalar work. Segment 2 of Table 6 shows (in the last column) the ratio of CPU (scalar plus block) memory to block memory references, which should be unity for good vector performing routines because scalar memory activity is negligible compared with that for vector memory. Here the scalar memory characteristics of POWER are again confirmed with a value of 2.2 for this PA. The first three columns of segment 2 show hold issues with a negligible value for the ratio of scalar to block for all routines except POWER. The second column shows that the actual number of block memory references per second is highest for FORR, as expected (from PA_MU_1), and lowest for POWER. The third column shows the number of vector operations for each block memory reference, and a value of 14.8 confirms the (vector) CPU bound character of FOUR2 and the deceptive value of 21.3 for POWER. Although it is not shown here, POWER has a large proportion of I/O activity in relation to the other routines and possible I/O memory conflicts in FORR.

As a result of this report, a summary can now be made of the character of each routine. The first ranking subroutine, FOUR2, performs fp operations almost entirely in vector mode with good chaining properties in evidence. Nevertheless, with a measured value of 0.835 CP per fp operation (PA_OI_9, Table 2), there is still some scope for improvement toward the theoretical limit of 0.5. The reason seems to be that the average fp vector length of 54 (PA_OI_6, Table 2) falls short of the ideal value of 64. Pipelining and chaining would improve if this gap could be closed. However, it would be unusual to see the routine achieve more than approximately 275 MFLOPS, which is equivalent to a reduction of about 20% in total CPU time, and this may not be easy to accomplish.

The second ranking subroutine, FORR, delivers half the vector fp MFLOPS of FOUR2 despite fp and VIL vector lengths of 70 and 54, respectively. The fp vector length exceeding 64 results from the additional vector logic (AVL) being enabled as discussed above. The number of instruction buffer fetches is largest for this routine (PA_OI_4, Table 2), which is part of the reason for loss in performance, but the measured value is not large enough to show a serious problem. Similar comments would apply to blank NIP time (PA_OI_10, Table 2). It could be concluded that FORR has substantial nonsequential code and subroutine calling overhead. An algorithmic change to remedy this could enhance performance. Nevertheless, some chaining is in evidence (PA_SV_3, 4, Table 4), even if the load balance of vector to scalar fp operations (PA_SV_2, Table 4) is not as favorable as it is in FOUR2. A significant part of this vector work is, of course, VILops (PA_OI_11, Table 2). Another feature of FORR is the existence of two memory references per fp operation (PA_MU_1, Table 6), which suggests that memory conflicts (PA_MU_3, Table 6), although not excessive, could be a contributing factor in degraded chaining performance when compared with that of FOUR2.

The third ranking routine, ASSRHOP, has the lowest fp vector length (PA_OI_6, Table 2) with the highest values for CP per fp operation (PA_OI_9, Table 2) and jump/special instructions (PA_OI_5, Table 2). This large value of $2.8 \times 10^6$/sec for the last mentioned PA is the cause of inhibited vector performance even though memory access is almost exclusively to block memory (PA_MU_2, 7, Table 6). The problem is confirmed by the significant value of 5.83% for the blank NIP time (PA_OI_10, Table 2).

The lowest ranking routine POWER measured the lowest value of vector MFLOPS (despite a reasonable vector length) but had a value of CP per fp operation similar to that of ASSRHOP (PA_OI_9, Table 2). But POWER has more than double the blank NIP time (PA_OI_10, Table 2) of ASSRHOP and also a large value for jump/special instructions (PA_OI_5, Table 2). So subroutine calling overhead and instruction parcel processing are factors contributing to poor performance in POWER. The number of scalar instructions is largest for this routine (PA_SV_1, Table 4), and memory instructions are mostly for scalar memory access (PA_SV_6, Table 4, PA_MU_7, Table 6). Furthermore, the memory conflict problem is the most serious for this routine (PA_MU_3, Table 6).

The analysis of the PAT report shown in Tables 2, 4, and 6 suggests that effort is best expended in improving the performance of the last three ranked routines because there is clearer evidence for the problems that inhibit performance and there is more scope for performance improvement than in the top ranking routine. This conclusion is reached without reference to the source code. The decoupling of performance and source code analysis is also justified in part by the fact that "sophisticated compiler transformations render accurate source-level analysis impossible" (Bradley and Larson, 1991). However, to complete the discussion of this case study, it is appropriate to explain the nature of the coding problems associated with each of these four routines as implemented by the user.

Subroutine FOUR2 performs a three-dimensional FFT using the Cray library routines RFFTMLT and CFFTMLT. It computes $N$ one-dimensional transforms and takes advantage of vectorization by computing these transforms simultaneously. The only way to further improve performance is to increase $N$, which would, however, also raise the memory requirements. The subroutine FORR solves for the grid cell position of each particle and computes the potential and forces at this position by interpolation with the grid of values at the corners of the cell. Each particle requires a gather operation of 36 values of the potential fetched from memory, and this is why the routine is memory bound. Subroutine FORR is largely vectorized, but the performance is sensitive to memory conflicts and enhanced instruction fetches. Subroutine ASSRHOP computes the discretized density field on the cubical grid by first finding the grid cell location of each particle. Then at each corner of the grid cell, a fraction of the particle mass is added in proportion to the particle distance. This requires a scatter loop over the eight corners of each grid cell and results in less than optimal vector lengths and lost performance. Subroutine POWER computes the power spectral density of the density field in three dimensions. This requires a scatter operation to compute the wavenumber bin for each wave. The scalar and vector operations have been placed into separate loops, and the significant amount of scalar work reduces the overall fp performance with the scalar scatter loop resulting in enhanced jump/special instructions and memory conflicts. In this respect, the impact of indirect addressing on overall performance and improvements possible from reorganization within algorithms are described by Ginsberg (1989).

## Workload Analysis with PAs

The PAT performance attributes can be used not only to investigate the path to improved performance for one code, but also to investigate the spectrum of performance achieved with a sample of codes. Such an analysis is required to establish basic statistical information on realistic and achievable values for minimum, maximum, and mean values for each PA. The purpose of this section is to show results of such an analysis when a 24-code sample was executed on both the CRAY X-MP and the Y-MP with the same compiler and operating system. The Fortran codes applied in this survey sample were written or used by the OSC user community, and performance attributes for each code on both processors were computed and compared. The methodology applied in the analysis of performance results is that typical of any experimental science: statistical sampling of the population, measurement of a group of parameters, and after-the-event analysis of statistical significance and correlation between parameters. In this respect, the volume of performance data accumulated by Cray UNICOS performance software can be drastically reduced to a few dominant factors. A description of such a detailed statistical analysis is the subject of a separate report with a larger sample of codes (G. Delic and R. I. Haller, work in progress, 1993). In this section, this comparison is presented in tabular and graphical form. This allows the study of trends in the sample and a quantitative comparison of two major supercomputer models. The results of the 24-code survey sample have been collected in Figures 1 to 21 and Tables 7, 8, and 9. The figures show plots of the PAs listed in Tables 1, 3, and 5 but grouped in a different order that should emphasize relationships between them. It is important to note that all figures correspond to the codes ranked in decreasing vector MFLOPS (from left to right) on both processors, and for each code the two values shown are for the Y-MP (solid bar) and X-MP (shaded bar), respectively. In the case of Figures 1 to 18, the ranking corresponds to that of Figure 4 where the hierarchy is on decreasing X-MP and Y-MP vector MFLOPS, respectively. This means that not all adjacent bars in Figures 1 to 18 correspond to the same code (although most do). For Figures 19 and 20, the ranking corresponds to that of Figure 21, which ranks both X-MP and Y-MP results on decreasing Y-MP vector MFLOPS, and adjacent bars do correspond to the same code. For convenience, mention is made of the left-hand side of this ordered sample as the upper half and the right-hand side as the lower half.

Figures 1 to 3 show the first group of PAs consisting of total MFLOPS (PA_0I_1), MIPS (PA_OI_3), and average CP per instruction (PA_OI_8). Since the sample is ranked on vector MFLOPS, total (scalar plus vector) MFLOPS shows a decline from left to right, with mean values shown in Table 7 as 42.4 (X) and 61.8 (Y) MFLOPS. What is remarkable here is the difference in the maximum value, which improves by 52% in moving from 111.2 MFLOPS on the X-MP to 169.8 MFLOPS on the Y-MP. Approximately half of the sample is 20 MFLOPS or less on both processors, and those codes delivering more than 40 MFLOPS experience an enhanced performance on the Y-MP compared with the X-MP. This improvement is greater than could reasonably be expected from the 30% change in clock speed from 8.5 to 6.0 nsec. Possible reasons for this improvement are investigated in the next section. The MIPS rate shown in Figure 2 has an inverse dependence on MFLOPS and has increased values for predominantly scalar code. A comparison of X and Y processors shows that the mean has moved from 29.1 to 39.2, an increase of 35%, which is consistent with the 30% improvement in clock speed. Figure 3 shows the decreasing trend in the CP per instruction with increasing scalar character of a code. Codes that show large MFLOPS also show large values for PA_OI_8 because this corresponds to a lower instruction issue rate or, equivalently, more operations per instruction. It is interesting to observe that the X-MP and Y-MP report similar values for the lower half of the sample (mostly scalar code), but the Y-MP reports smaller values, in general, for the upper half of the sample. From Table 7, the respective mean values are seen to be 5.7 (X) and 5.0 (Y).

Figures 4 to 6 show another group of PAs consisting of vector MFLOPS (PA_OI_2), average fp vector length (PA_OI_6), and average CP per fp operation (PA_OI_9). As noted above, the ranking on vector MFLOPS clearly shows the spectrum of performance from codes in the upper half of the sample, which deliver high vector fp performance, and those in the lower half of the sample, which are predominantly scalar. Remarks similar to those for PA_OI_1 also apply here. The average fp vector length confirms that an ideal value of 64 is a necessity for peak vector MFLOPS performance but is not sufficient to guarantee this because there may be other reasons why such performance is inhibited. On the other hand, a short vector length will guarantee poor vector performance. The average vector length is computed by dividing the total number of vector fp operations by the total number of vector fp instructions. Values for a vector length greater than 64 are the result of the additional vector logical (AVL) unit being enabled. Evidently, the vector length does not depend on the machine, because both X and Y processors have vector registers 64 elements long. However,

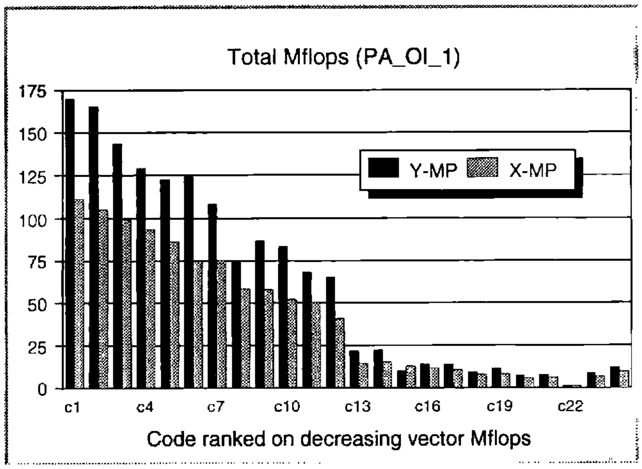


**Fig. 1 Total MFLOPS (PA_OI_1) for CRAY X-MP (shaded bars) and Y-MP (solid bars) supercomputers with a 24-code sample sorted on decreasing vector MFLOPS as in Fig. 4**

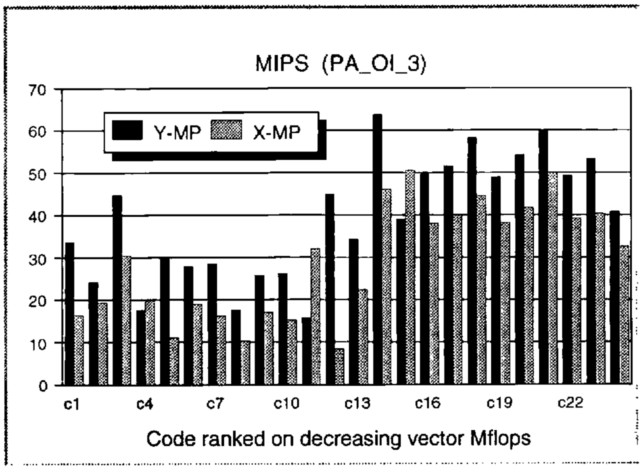


**Fig. 2 MIPS rate (PA_OI_3) for CRAY X-MP (shaded bars) and Y-MP (solid bars) supercomputers with a 24-code sample sorted on decreasing vector MFLOPS as in Fig. 4**

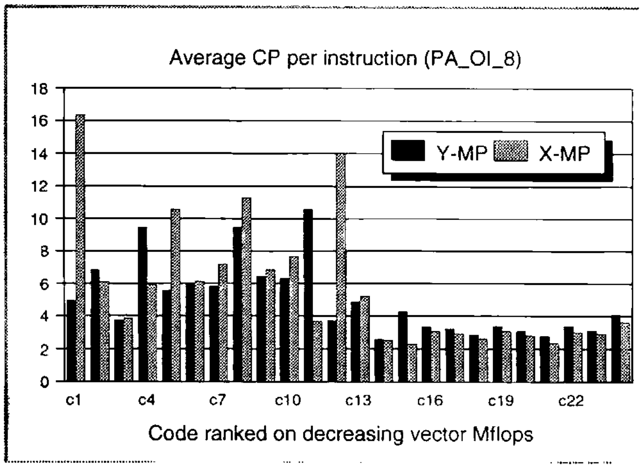


**Fig. 3 Average CP per instruction (PA_OI_8) for CRAY X-MP (shaded bars) and Y-MP (solid bars) supercomputers with a 24-code sample sorted on decreasing vector MFLOPS as in Fig. 4**

Figure 5 does show that some adjacent pairs of codes have swapped location when ranked strictly on vector MFLOPS on both the X and Y processors. Figure 6 and Table 7 show that peak vector fp performing codes in this sample achieve values on the order of 1 CP per FLOP, while scalar code can have extremes as high as 100 CP per FLOP. Such extremes occur in codes with little fp work and many logical branches. The theoretical ideal value of PA_OI_9 for the X-MP and Y-MP architecture is 0.5 CP per FLOP, or two operations per CP. However, when comparing the X-MP with the Y-MP, note that while similar values of PA_OI_9 are reported for the upper half of the sample (higher MFLOPS), considerably larger values are measured for the Y-MP on the lower half of the sample. Thus, the move in the mean from 11 (X) to 13.7 (Y) measures how much slower the Y-MP is compared with the X-MP for scalar work on a CP count.

Figures 7 to 9 show the next group of PAs consisting of instruction buffer fetches in millions per second

**Table 7**
**Statistics of the 24-Code Sample on CRAY X-MP and Y-MP Processors with Performance Attributes for Operations and Instructions**

| PA | CPU | Minimum | Maximum | Mean | Standard Deviation | Skewness |
|---|---|---|---|---|---|---|
| PA_OI_0 | X | 0.55 | 408.1 | 70.44 | 101.8 | 2.250 |
| | Y | 0.35 | 326.2 | 52.23 | 80.20 | 2.383 |
| PA_OI_1 | X | 1.12 | 111.2 | 42.39 | 36.82 | 0.5039 |
| | Y | 1.15 | 169.8 | 61.84 | 56.0 | 0.5369 |
| PA_OI_2 | X | 0.0 | 111.9 | 39.44 | 38.71 | 0.4607 |
| | Y | 0.0 | 168.8 | 59.84 | 58.89 | 0.4459 |
| PA_OI_3 | X | 8.34 | 50.58 | 29.14 | 13.25 | 0.0267 |
| | Y | 15.79 | 63.78 | 39.20 | 14.11 | −0.0341 |
| PA_OI_4 | X | 0.0087 | 1.127 | 0.2897 | 0.2824 | 1.223 |
| | Y | 0.0155 | 1.562 | 0.4368 | 0.3889 | 1.186 |
| PA_OI_5 | X | 1.303 | 6.844 | 3.911 | 1.497 | 0.2000 |
| | Y | 1.291 | 6.952 | 3.267 | 1.646 | 0.7770 |
| PA_OI_6 | X | 0.0 | 74.15 | 38.48 | 25.31 | −0.1779 |
| | Y | 0.0 | 74.16 | 38.48 | 25.31 | −0.1779 |
| PA_OI_7 | X | 37.51 | 107.1 | 73.93 | 21.28 | −0.0390 |
| | Y | 48.94 | 146.3 | 103.1 | 27.58 | −0.1265 |
| PA_OI_8 | X | 2.31 | 16.35 | 5.672 | 3.786 | 1.406 |
| | Y | 2.614 | 10.56 | 4.995 | 2.218 | 1.116 |
| PA_OI_9 | X | 1.06 | 100.4 | 10.99 | 19.65 | 3.879 |
| | Y | 0.982 | 144.9 | 13.66 | 28.30 | 4.116 |
| PA_OI_10 | X | 2.23 | 33.06 | 12.37 | 7.849 | 0.7687 |
| | Y | 2.76 | 47.24 | 14.65 | 10.33 | 1.437 |
| PA_OI_11 | Y | 0.004 | 134.4 | 28.38 | 38.05 | 1.693 |
| PA_OI_12 | Y | 3.5 | 63.39 | 34.34 | 20.86 | −0.0801 |

(PA__OI__4), jump/special instructions (PA__OI__5), and percentage of blank NIP time (PA__OI__10). These three attributes, which were explained above, are related because they measure overhead generated by the code during execution. Experience has shown that they are reliable and sensitive indicators of performance loss due to coding inefficiency. This was confirmed in the case study described above. Here, for the 24-code sample, PA__OI__4 shows values less than 0.5 $\times$ $10^6$/sec on the Y-MP for the upper half of the sample (high vector MFLOPS) and values larger than this for the lower half (scalar code). Values as large as 1 $\times$ $10^6$/sec are clear indicators of scalar performance. From Table 7, the mean value on the Y-MP is 0.437, which is 50% larger than that for the X-MP reported at 0.29 $\times$ $10^6$/sec. A similar inspection of PA__OI__5 in Figure 8 shows the same trend with values generally less than 3 $\times$ $10^6$/sec for predominantly vector code and values larger than this for scalar code. Here the mean value on the Y-MP of 3.27 is less than the mean of 3.91 reported for the X-MP, although the maximal and minimal values are similar, as is seen in Table 7. In Figure 9, the blank NIP time (PA__OI__10) is less than 10% for code

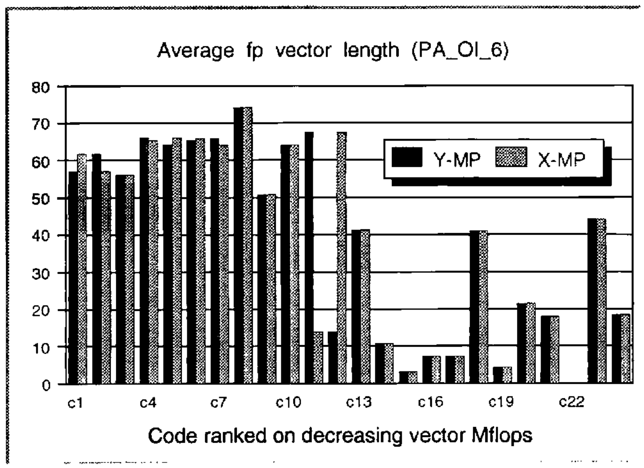


**Fig. 5 Average floating-point vector length (PA__OI__6) for CRAY X-MP (shaded bars) and Y-MP (solid bars) supercomputers with a 24-code sample sorted on decreasing vector MFLOPS as in Fig. 4**

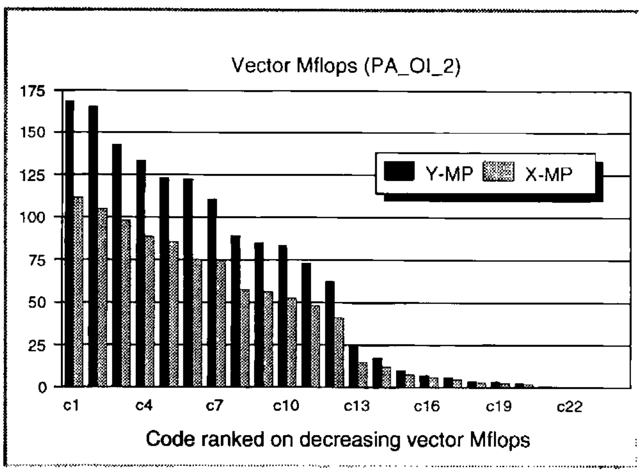


**Fig. 4 Vector MFLOPS (PA__OI__2) for CRAY X-MP (shaded bars) and Y-MP (solid bars) supercomputers with a 24-code sample that has been sorted from left to right on decreasing X-MP and Y-MP vector MFLOPS, respectively** This means that not all adjacent bars correspond to the same code (see also the discussion of Fig. 5 in the text).

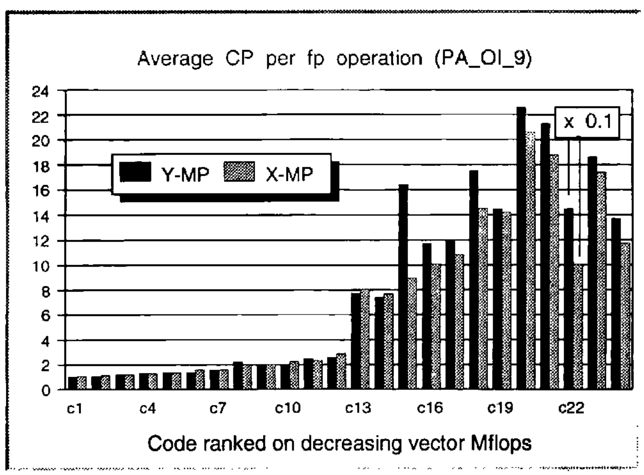


**Fig. 6 Average CP per fp operation (PA__OI__9) for CRAY X-MP (shaded bars) and Y-MP (solid bars) supercomputers with a 24-code sample sorted on decreasing vector MFLOPS as in Fig. 4**

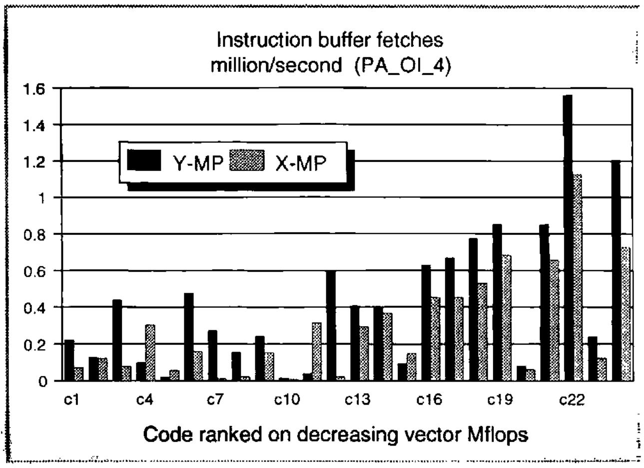


**Fig. 7 Instruction buffer fetches in millions per second (PA_OI_4) for CRAY X-MP (shaded bars) and Y-MP (solid bars) supercomputers with a 24-code sample sorted on decreasing vector MFLOPS as in Fig. 4**

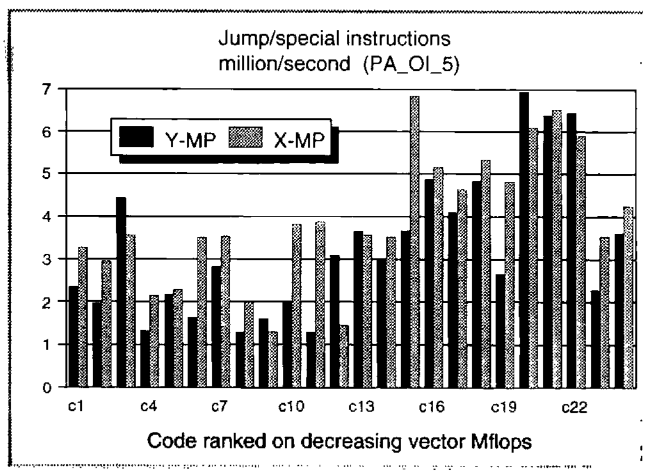


**Fig. 8 Jump/special instructions in millions per second (PA_OI_5) for CRAY X-MP (shaded bars) and Y-MP (solid bars) supercomputers with a 24-code sample sorted on decreasing vector MFLOPS as in Fig. 4**

with good vector performance but larger than this for scalar code, and in the former case, values for the Y-MP are generally a little larger. For PA_OI_10, the values reported for X and Y models can be very different for some codes, which accounts for the differences shown in Table 7. In summary, the evidence suggests that all three PAs shown in Figures 7, 8, and 9 are correlated.

Other PAs (not shown here) are CP holding issue in millions per second (PA_OI_7), vector integer logical operations, or VILops (PA_OI_11), and the average vector integer logical vector length (PA_OI_12). The last two are reported only for the Y-MP because of changes in the hpm report format. Values of PA_OI_7 on the Y-MP are substantially larger than those on the X-MP, and Table 7 shows mean values of $74 \times 10^6$/sec (X) versus $103 \times 10^6$/sec (Y). The last two attributes in this group, PA_OI_11 and PA_OI_12 (reported only for the Y-MP), show that good vector fp performance does not exclude substantial integer or logical work.

The next group of PAs, shown in Figures 10, 11, and 12, respectively, are for the scalar to vector load balance of fp instructions (PA_SV_1), hold issues (PA_SV_5), and memory instructions (PA_SV_6). This group emphasizes ratios of instructions, and since those for vector work are in the denominator, code with good vector MFLOPS performance should show small values relative to predominantly scalar code. This is indeed the case, and PA_SV_1 in Figure 10 shows that the upper half of the sample of good vector code reports a value of 10 (or less), but this increases rapidly on the logarithmic scale as scalar work predominates. If there are no vector instructions issued, the ratio is infinite and the PAT tools then insert a zero as happens in one case for the Y-MP. Therefore, the statistics for PA_SV_1 in Table 8 are biased in this case. The ratio of hold issues (PA_SV_5) shown in Figure 11 follows the same trend starting from a value close to 0.1 for good vector code and increasing to approximately $10^5$. For the upper half of the sample, there are significant differences between X and Y performance, and the respective mean values are 4,750 (X) and 673 (Y). Figure 12 shows (PA_SV_6) values in the range 0.01 to 1 for the upper half of the sample and values larger than 1 for scalar code.

Important measures of scalar to vector load balance are shown in Figures 13, 14, and 15, respectively, for vector to scalar fp operations (PA_SV_2), hold issues on vector registers (PA_SV_3), and hold issues on vector FUs (PA_SV_4), with the last two in millions per second. All three PAs show a dramatic decline with increasing scalar activity in the code. Figure 13 shows PA_SV_2 on a logarithmic scale, with very large values for code with peak vector MFLOPS and very small values for predominantly scalar code. There are apparent differences between the X-MP and Y-MP that require some explanation. From Table 3, it is evident that in calculating PA_SV_2 the scalar operations are computed in the denominator as the difference of the hpm g0 FLOPS and g3 vector FLOPS. This denominator gives only positive values for hpm data on the X-MP, but some codes report negative values on the Y-MP. Clearly, g3 vector FLOPS on the Y-MP contain more than the user code's numerical computation, for reasons such as those discussed above. Another reason for such distorted values is the possibility that the g3 fp counter includes the effects of EMA implementation as already discussed. Since the problem of negative values of PA_SV_2 occurs only in codes with large amounts of vector work, it is reasonable to use the absolute value, which is the quantity shown in Figure 13. The trends displayed for PA_SV_2 seem to justify this choice. For PA_SV_2, Table 8 shows statistics for both cases (without and with the absolute value). The actual absolute amount of vector work is best measured by inspecting hold issues on vector registers and FUs, as shown in Figures 14 and 15, respectively. If these PAs report large values, this shows many vector instructions waiting for execution, and the converse shows mainly scalar activity. Table 8 shows that mean values are $15 \times 10^6$/sec (X) and $19 \times 10^6$/sec (Y) for PA_SV_3 and $24 \times 10^6$/sec (X) and $29 \times 10^6$/sec (Y) for PA_SV_4, respectively. There is a clear increase in vector resource utilization on the Y-MP compared with the X-MP. Chaining is in evidence if both of these PAs show large values (larger than $50 \times 10^6$/sec, for example), which is not the

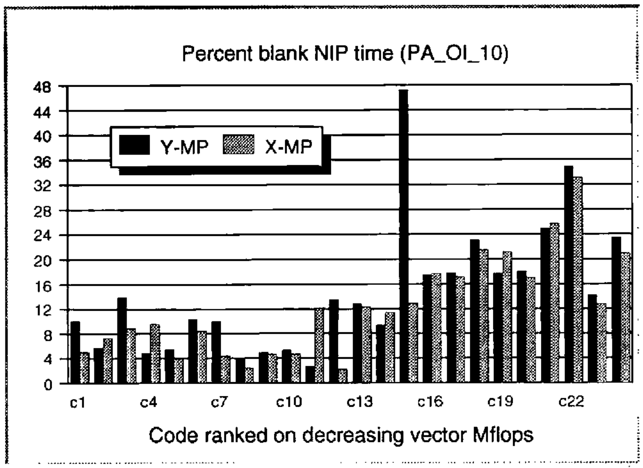


**Fig. 9 Percentage of blank NIP time (PA_OI_10) for CRAY X-MP (shaded bars) and Y-MP (solid bars) supercomputers with a 24-code sample sorted on decreasing vector MFLOPS as in Fig. 4**

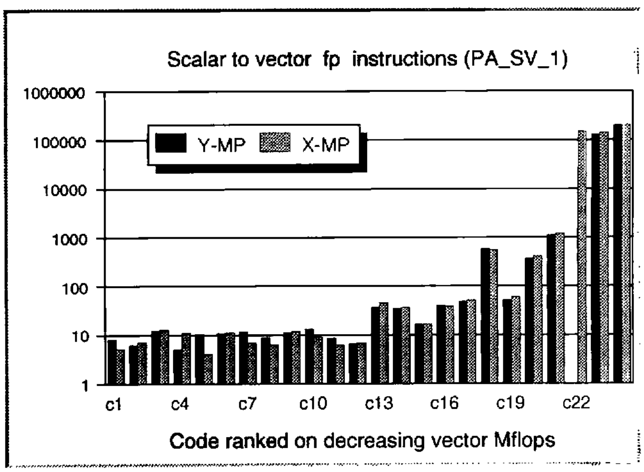


**Fig. 10 The ratio of scalar to vector fp instructions (PA_SV_1) on a logarithmic scale for CRAY X-MP (shaded bars) and Y-MP (solid bars) supercomputers with a 24-code sample sorted on decreasing vector MFLOPS as in Fig. 4** Note that the scalar instruction set included is different for the X-MP and Y-MP (see Table 3).

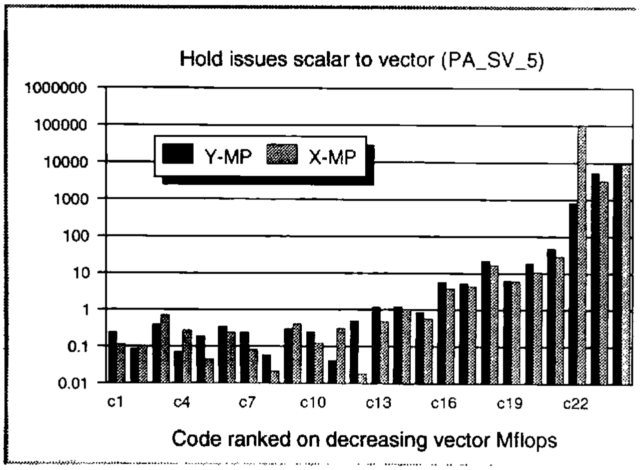


**Fig. 11 The ratio of scalar to vector hold issues (PA_SV_5) on a logarithmic scale for CRAY X-MP (shaded bars) and Y-MP (solid bars) supercomputers with a 24-code sample sorted on decreasing vector MFLOPS as in Fig. 4**

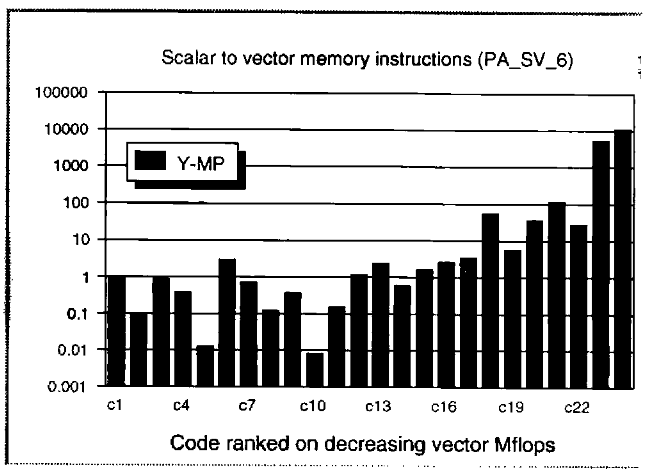


**Fig. 12 The ratio of scalar to vector memory instructions (PA_SV_6) on a logarithmic scale for the CRAY Y-MP supercomputer with a 24-code sample sorted on decreasing vector MFLOPS as in Fig. 4**

case in this sample. Experience with other codes has shown that to exceed 300 MFLOPS on the Y-MP, values of PA_SV_3 and PA_SV_4 must be in excess of 120 $\times$ $10^6$/sec (as measured with cft77 4.0).

The first group of memory utilization PAs (not shown in the figures) are for memory references per FLOP (PA_MU_1), the ratio of hold issues for vector operations to those for block memory references (PA_MU_6), and the ratio of CPU (vector plus scalar) memory references to block memory references (PA_MU_7). Table 8 shows that PA_MU_1 ranges between a minimum of 0.41 to maxima of 7.6 (X) and 9.7 (Y), respectively, while mean values tend to be similar. The codes delivering peak vector MFLOPS tend to be CPU bound with values less than 2.5 while codes with larger values tend to show reduced MFLOPS because of the increased sensitivity to memory contentions. Small values of PA_MU_6 for a well-vectorized code could signal a bottleneck on vector memory. However, a large value for a code that delivers a low MFLOPS rate is spurious because there is so little vector work in this case. From Table 8, both the peak values of 260 (X) and 371 (Y) and the mean values of 15.2 (X) and 21.2 (Y) are considerably larger for the Y-MP. The value of PA_MU_7 for codes in the upper half of the sample is unity, showing that vector memory activity dominates over scalar memory activity.

Figures 16, 17, and 18 show, respectively, the ratio of scalar to block memory references (PA_MU_2), the hold issues ratio of scalar to block memory references (PA_MU_4), and the number of hold issues for block memory references (PA_MU_5). These three PAs clearly indicate the balance between scalar and vector memory utilization. In particular, small values of the order of 0.01 (or less) for PA_MU_2 show the dominance of vector memory activity, which is partially confirmed by PA_MU_4 and is evident in Figure 18 for PA_MU_5. Codes that deliver both high vector MFLOPS (PA_OI_2) and large values of PA_MU_5 show no memory bottlenecks and may have good chaining properties (PA_SV_3,4). For PA_MU_2, Table 9 shows a small difference between processors with mean values of 0.296 (X) and 0.270 (Y), respectively. There appear to be large differences between X and Y

**Table 8**

**Statistics of the 24-Code Sample on CRAY X-MP and Y-MP Processors with Performance Attributes for Scalar to Vector Load Balance**

| PA | CPU | Minimum | Maximum | Mean | Standard Deviation | Skewness |
|---|---|---|---|---|---|---|
| PA_SV_1[a] | X | 4.115 | 207949.9 | 20935.0 | 56044.4 | 2.4319 |
| | Y | 0.0 | 200160.8 | 13978.7 | 47013.2 | 3.239 |
| PA_SV_2[b] | X | 0.0 | 7902.9 | 896.0 | 1877.8 | 2.436 |
| | Y | −2285.5 | 161.1 | −106.6 | 466.4 | −4.230 |
| | \|Y\| | 0.0 | 2285.5 | 146.8 | 455.3 | 4.302 |
| PA_SV_3 | X | 0.0 | 45.0 | 15.14 | 13.82 | 0.5504 |
| | Y | 0.0 | 59.11 | 19.18 | 17.84 | 0.6416 |
| PA_SV_4 | X | 0.0 | 81.65 | 23.95 | 24.05 | 0.9517 |
| | Y | 0.0 | 95.24 | 29.16 | 28.83 | 0.8328 |
| PA_SV_5 | X | 0.018 | 101297.2 | 4749.8 | 20226.6 | 4.520 |
| | Y | 0.042 | 9847.8 | 673.1 | 2193.6 | 3.429 |
| PA_SV_6 | Y | 0.0 | 11384.7 | 717.5 | 2485.6 | 3.5820 |

[a] This PA measures different quantities on X and Y processors (see Table 3).

[b] For the Y-MP, this PA can show negative values in some cases (see discussion in text). Therefore, the line denoted |Y| shows statistics obtained for the absolute value.

processors for PA_MU_4, with mean values of 0.098 (X) and 0.209 (Y), respectively. This is explained in part by the difference in hold issues for block memory references (PA_MU_5) shown in Figure 18 and Table 9. Negligible values for PA_MU_5 in the lower half of the sample clearly show little vector work, and this would also result in large values for PA_MU_2 and PA_MU_4.

Not shown here are memory reference conflicts as a percentage (PA_MU_3), the ratio of vector fp instructions to vector memory instructions on the Y-MP (PA_MU_8), and vector memory average vector length on the X-MP (PA_MU_8). The switch in the choice of attribute for PA_MU_8 is due to the change of recorded data in hpm from X to Y processors. Also, from Table 5, it is seen that PA_MU_3 reports only block memory reference conflicts on the X-MP and both scalar and block memory reference conflicts on the Y-MP. In either case, for codes reporting high vector MFLOPS, a memory bottleneck is generally indicated

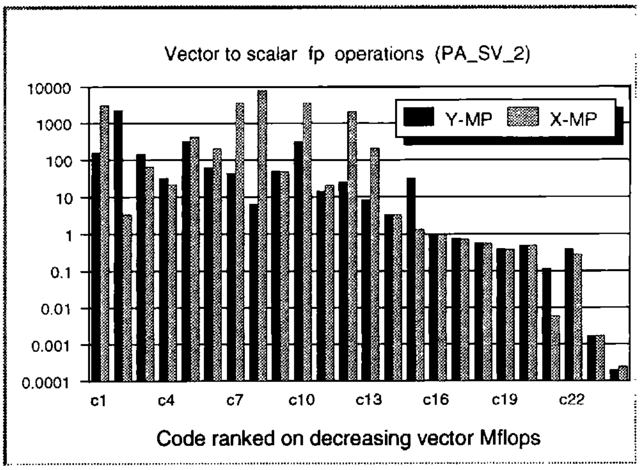


**Fig. 13 The ratio of vector to scalar fp operations (PA_SV_2) on a logarithmic scale for CRAY X-MP (shaded bars) and Y-MP (solid bars) supercomputers with a 24-code sample sorted on decreasing vector MFLOPS as in Fig. 4** Note that for the Y-MP, the absolute value is shown since negative values are possible on this processor (see the discussion in text).

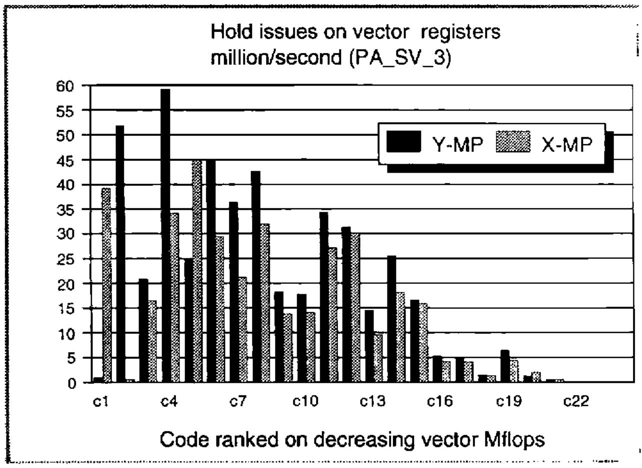


**Fig. 14 Hold issues on vector registers in millions per second (PA_SV_3) for CRAY X-MP (shaded bars) and Y-MP (solid bars) supercomputers with a 24-code sample sorted on decreasing vector MFLOPS as in Fig. 4**

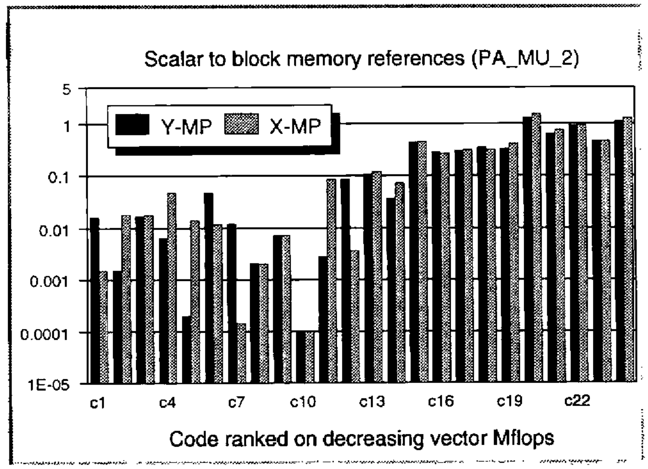


**Fig. 16 The ratio of scalar to block memory references (PA_MU_2) on a logarithmic scale for CRAY X-MP (shaded bars) and Y-MP (solid bars) supercomputers with a 24-code sample sorted on decreasing vector MFLOPS as in Fig. 4**

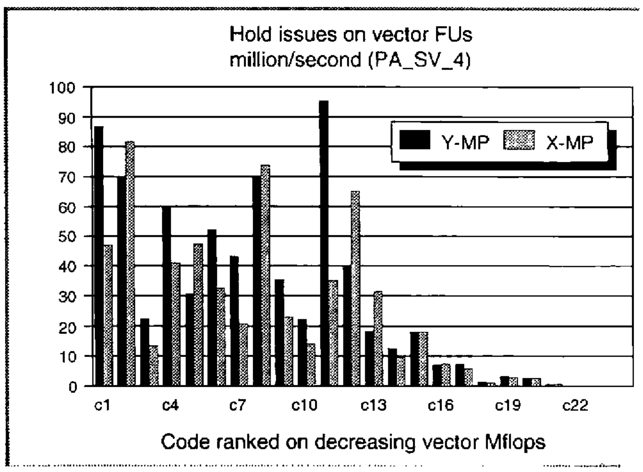


**Fig. 15 Hold issues on vector functional units in millions per second (PA_SV_4) for CRAY X-MP (shaded bars) and Y-MP (solid bars) supercomputers with a 24-code sample sorted on decreasing vector MFLOPS as in Fig. 4**

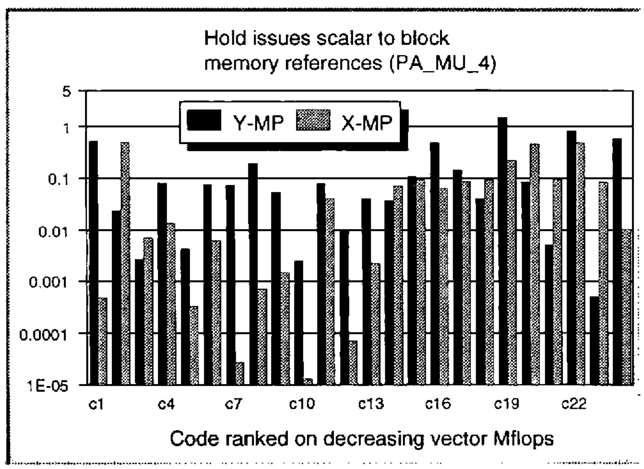


**Fig. 17 The ratio of hold issues for scalar to block memory references (PA_MU_4) on a logarithmic scale for CRAY X-MP (shaded bars) and Y-MP (solid bars) supercomputers with a 24-code sample sorted on decreasing vector MFLOPS as in Fig. 4**

by values larger than 10%. Exactly how sensitive a code is to such conflicts depends on how compute intensive it is (PA_SV_2,5). Nevertheless, for the upper half of the sample on the Y-MP, PA_MU_2, 4 tend to shadow PA_MU_3, which suggests that increased block memory activity corresponds to reduced conflicts. For those codes that are dominated by block memory activity, a striking feature of PA_MU_3 is the drastic reduction in memory bank conflicts on the Y-MP compared with the X-MP, and Table 9 shows a reduction in the mean value by a factor of 4. The X-MP has a 32-bank, 8-million-word shared memory while the Y-MP has a 256-bank, 64-million-word memory. This change is also reflected in the differences between X-MP and Y-MP values seen in Figures 17 and 18. For the Y-MP, to measure if a code is compute intensive or memory intensive only on vector characteristics, PA_MU_8 shows the ratio of instructions for fp to vector memory activity. Therefore, a large value for a code that is predominantly scalar is of little significance. Finally, for the X-MP, PA_MU_8 shows vector lengths close to the ideal of 64 for codes delivering good vector MFLOPS performance.

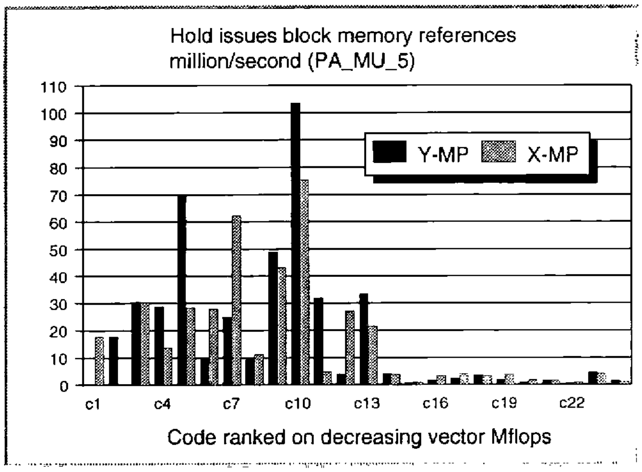


**Fig. 18 Hold issues for block memory references in millions per second (PA_MU_5) for CRAY X-MP (shaded bars) and Y-MP (solid bars) supercomputers with a 24-code sample sorted on decreasing vector MFLOPS as in Fig. 4**

## Performance Comparison Using PAs

This section presents a performance comparison for a single processor of the CRAY X-MP and Y-MP models in three key areas, namely, computational activity, scalar to vector load balance, and execution penalty, which are each introduced in separate sections. The comparison is based on the 24-code sample discussed above with the same compiler (cft77 3.1.1) and operating system (UNICOS 5.0). Results of Y-MP performance with a much larger sample is the subject of another study (G. Delic and R. I. Haller, work in progress, 1993).

### COMPUTATIONAL ACTIVITY

Under this title are collected those attributes from the four groups of PAs in Tables 1, 3, and 5 that show a mixture of computational activity on different hardware resources. The performance change in PAs on the CRAY Y-MP relative to the X-MP is computed for the sample mean to avoid specific effects of any one code.

From Tables 7, 8, and 9, examples of attributes that have increased sharply, and the corresponding percentage change in the sample mean value, are the MIPS rate (PA_OI_3) with 35%, CPs holding issue (PA_OI_7) with 39%, and total MFLOPS (PA_OI_1) with 46%. On the other hand, the average time between instructions (PA_OI_8) and the jump/special group of instructions (PA_OI_5) are reduced in the mean by 12% and 4%, respectively. While the 29% improvement in clock speed of the Y-MP could imply better scalar fp performance, those codes in the upper half of the sample, which show good vector performance on the X-MP, also gain the most improvement in moving to the Y-MP. This is reflected in vector MFLOPS (PA_OI_2) with improvements in both the mean and peak values in excess of 50%. Table 8 shows that this is indicated by the increased number of hold issues on vector registers (PA_SV_3) and FUs (PA_SV_4) with increases in the mean of 27% and 22%, respectively. Furthermore, from Table 8, the ratio of hold issues on vector operations to those for block memory references (PA_MU_6) shows an increase in mean of 39%. This is so despite the fact that the sample mean for the average number of memory references per FLOP (PA_MU_1) increases by only 6% for the Y-MP when compared with the X-MP. Taken together, these are clear indications that, in the mean, good vector performing codes become 39% more compute intensive, or CPU bound, on the Y-MP compared with the X-MP and fp operation rates increase by a factor that is close to twice the clock speed improvement.

## SCALAR TO VECTOR LOAD BALANCE

The study of scalar to vector load balance is divided into two components, the first being fp computation and the second memory utilization. For fp computation, Figure 6 and Table 7 show that the lower half of the sample, which corresponds to predominantly scalar code, accounts for the significant increase of 25% in the sample mean of average CPs per FLOP (PA_OI_9). This shows that scalar processing on the CRAY Y-MP costs more CPs per operation, although in view of the 29% shorter CP, this need not imply that the Y-MP is slower than the X-MP for such operations. On the other hand, as seen in Figure 10, the ratio of scalar to vector fp instructions (PA_SV_1) is not significantly different for X and Y processors. Nevertheless, there is an important shift in the scalar to vector computational load balance. In the previous section increases greater than 20% were noted for hold issues on vector registers and FUs (PA_SV_3,4). However, the ratio of scalar to vector instructions (PA_SV_1), shown in Figure 10, have similar values for both processors (with one exception). Similarly, despite the scatter, Figure 11 shows that the ratio of scalar to vector hold issues (PA_SV_5) has similar means on X and Y processors (again with one exception). While these results could suggest that scalar and vector instruction rates have increased in the same proportion, it is of course the relative increase in the vector operation rate that contributes to a vector MFLOPS rate (PA_OI_2) improvement of 51%.

In the case of scalar to vector load balance on memory utilization, hold issues on block memory references (PA_MU_5) show an increased mean of 11.6% (Table 9) for the Y-MP over the X-MP. This appears to be more than compensated by a 110% increase in the mean for the hold issues ratio of scalar memory to block memory references (PA_MU_4). However, the ratio of actual scalar to block memory references (PA_MU_2) shows a decrease in mean of 10%. The latter result demonstrates an increase in block memory activity for the Y-MP (largely for the upper half of the sample). On both X and Y processors, vector memory activity dominates for codes in the upper half of the sample because PA_MU_7 is unity there.

## EXECUTION PENALTY

During code execution, various types of instructions are issued for operations that are not directly related to user data manipulation. Examples are those for instruction fetches from memory, subscript calculations, EMA and AVL processing, and so on. Therefore, it seems appropriate to refer to these collectively as an execution penalty because the user data receive no direct arithmetic operational benefits from them. The execution is said to suffer a penalty because there is still an associated cost in CPU time to the user application for exe-

**Table 9**
**Statistics of the 24-Code Sample on CRAY X-MP and Y-MP Processors with Performance Attributes for Memory Utilization**

| PA | CPU | Minimum | Maximum | Mean | Standard Deviation | Skewness |
|---|---|---|---|---|---|---|
| PA_MU_1 | X | 0.4126 | 7.602 | 1.585 | 1.422 | 3.203 |
| | Y | 0.4246 | 9.740 | 1.700 | 1.805 | 3.676 |
| PA_MU_2 | X | 0.0 | 1.548 | 0.2963 | 0.4192 | 1.683 |
| | Y | 0.0 | 1.306 | 0.2701 | 0.3675 | 1.503 |
| PA_MU_3[a] | X | 7.255 | 372.6 | 67.86 | 71.69 | 3.259 |
| | Y | 2.82 | 70.51 | 15.58 | 17.57 | 1.911 |
| PA_MU_4 | X | 0.0 | 0.5053 | 0.09819 | 0.1551 | 1.847 |
| | Y | 0.0 | 1.494 | 0.2088 | 0.3453 | 2.415 |
| PA_MU_5 | X | 0.3166 | 75.43 | 16.32 | 19.84 | 1.575 |
| | Y | 0.237 | 103.49 | 18.19 | 25.13 | 1.947 |
| PA_MU_6 | X | 0.0 | 259.8 | 15.23 | 51.61 | 4.426 |
| | Y | 0.0 | 370.9 | 21.20 | 73.43 | 4.487 |
| PA_MU_7[a] | X | 0.4643 | 2.425 | 1.032 | 0.3434 | 2.434 |
| | Y | 1.0 | 2.306 | 1.270 | 0.3685 | 1.503 |
| PA_MU_8[a] | X | 3.49 | 63.92 | 35.90 | 23.11 | −0.0596 |
| | Y | 0.0 | 2.361 | 0.8955 | 0.5214 | 0.8949 |

[a] This PA measures different quantities on X and Y processors (see Table 5).

cution. Four PAs contribute to this execution penalty, and when compared with the CRAY X-MP, two of them have (in the mean) significantly larger values on the Y-MP while two have smaller values. Table 7 shows mean value increases for instruction buffer fetches (PA_OI_4) and blank NIP time (PA_OI_10) of 52% and 19%, respectively. In the case of PA_OI_10, with one exception, most of the increase is in the upper half of the sample. However, one PA that is reduced in value is that for the number of jump/special instructions (PA_OI_5), with a mean value reduction of 4%, suggesting more efficient processing of nonsequential code on the Y-MP. Finally, memory conflicts (PA_MU_3) appear to be reduced, but Table 5 shows that this attribute measures different quantities on X-MP and Y-MP processors. However, for the upper half of the sample (higher vector MFLOP code), this PA is measuring memory conflicts on block memory references, and these are reduced by a factor of 4 on the Y-MP compared with the X-MP.

## SUMMARY OF PERFORMANCE COMPARISON

The above analysis with the PAs of Tables 1, 3, and 5, when applied in the comparison of CRAY X-MP and Y-MP with the 24-code sample, shows the following general features:

The duration of the CP decreases by 29%.
Instruction issue rates increase by 35%.
Instruction fetch rates increase by 52%.
Instruction parcel processing rates increase by 19%.
CPs per instruction increase by 12%.
CPs per FLOP increase by 25%.
CPs holding issue increase by 39%.
Vector MFLOPS increase by 52%.
Hold issues on vector registers and FUs increase by more than 22%.
The hold issues ratio of vector operations to block memory references increases by 39%.
Block memory activity increases by 10%.
Memory conflicts decrease by a factor of 4.

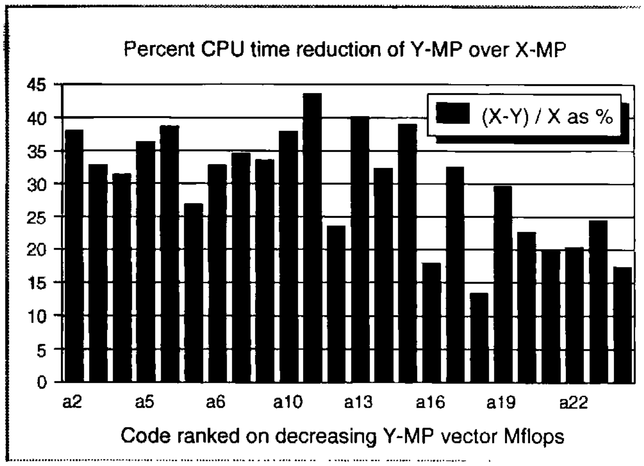


**Fig. 19 The percentage reduction in CPU execution time in moving each code in the sample to the CRAY Y-MP supercomputer measured relative to the X-MP** Here the 24-code sample is sorted on decreasing Y-MP vector MFLOPS as in Fig. 21.

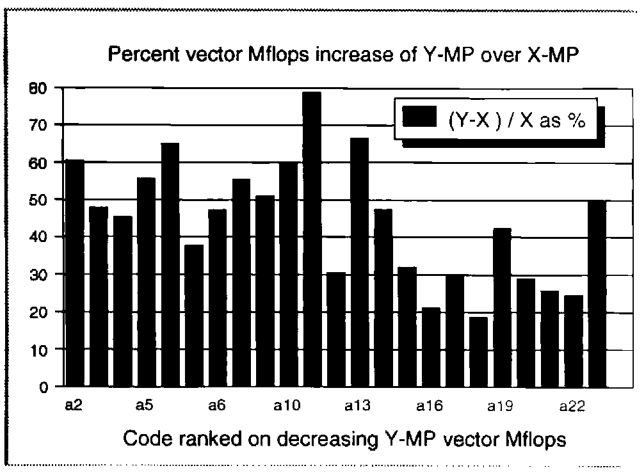


**Fig. 20 The percentage increase in vector MFLOPS in moving each code in the sample to the CRAY Y-MP supercomputer measured relative to the X-MP** Here the 24-code sample is sorted on decreasing Y-MP vector MFLOPS as in Fig. 21.

These results show several important improvements in processing on the Y-MP over the X-MP, which are the consequences of a combination of several factors: the 29% reduced CP, a 10% increase in block memory activity, substantially fewer memory conflicts, and the 39% increase in the number of hold issues for vector FLOPS compared with hold issues for block memory references. The consequent 52% enhancement of vector fp operation rates shows that compute-intensive applications become more CPU bound on the Y-MP than on the X-MP. This analysis shows that insight into the performance differences of X-MP and Y-MP processors comes from measuring changes in PAs for the 24-code sample.

It is appropriate at this point to explain some of the architectural differences in X-MP and Y-MP models that make possible such dramatic changes in performance. In the case of the shared memory, the number of memory banks increased from 32 (X) to 256 (Y), and the Y-MP has a bipolar memory, which allows simultaneous read and write operations. The impact on performance of these (and other) differences in memory architecture is discussed elsewhere (Faanes and Schwarzmeier, 1989). The X and Y models also differ in the address modes in that address registers and functional units have 24-bit words on the X-MP and 32-bit words on the Y-MP with an X-mode addressing option. Both the X-MP and Y-MP CPUs have four instruction buffers, each of which contains 128 consecutive 16-bit instruction parcels for a total of 512. However, for the X-MP, instruction processing is limited to one or two instruction parcels only, whereas on the Y-MP the instruction set is expanded to include three parcel instructions. More detail is to be found in the relevant vendor documentation (Cray Research, Inc., 1990, 1991b).

The combined effect of these architectural improvements for the individual codes in the sample is best appreciated when the percentage reduction in CPU execution time shown in Figure 19 is compared with the percentage increase in vector MFLOPS shown in Figure 20. To see how the results of Figure 20 are obtained, the results of Figure 4 for vector MFLOPS are shown again in Figure 21, but in the latter case both X-MP and Y-MP results are sorted on decreasing Y-MP vector MFLOPS. As a consequence, in Figure 21 for all cases, adjacent bars for Y-MP (solid) and X-MP (shaded) represent the same code. Comparison of Figures 19 and 20 shows that the improvement in CPU time is directly proportional to the improvement in vector MFLOPS. Overall improvements in vector MFLOPS range from a minimum of 19% to a maximum of 79%, and the corresponding improvements in CPU time range from 13% to 44%. Codes in the lower half of the sample show improvements in execution time, which have a mean on the order of 26%. Codes in the upper half of the sample show a mean improvement in CPU time of 34%. A simple linear (least squares) regression gives the good approximation $time\_\% = 0.39 \times MFLOPS\_\% + 13.4$, where $time\_\%$ and $MFLOPS\_\%$ represent the variables shown in Figures 19 and 20, respectively. It is remarkable to observe that 5 of the 24 codes in the sample recorded a gain of 60%, or more, in vector MFLOPS and a corresponding decrease in CPU execution time of 38%, or more. The code with the largest improvement increased the vector MFLOPS rate by 79% and decreased execution time by 44%.

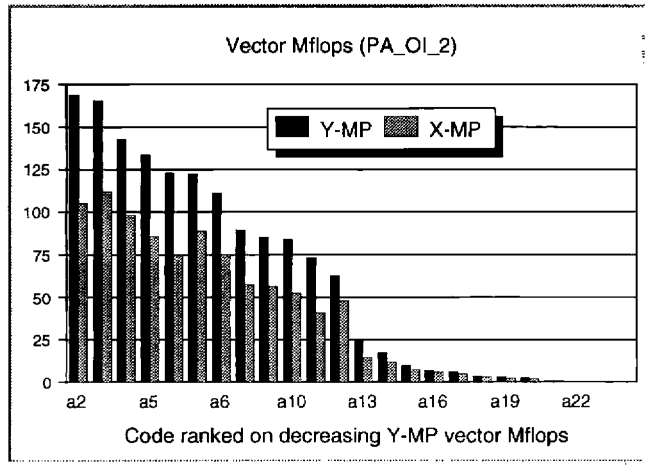


**Fig. 21 The 24-code sample sorted on decreasing Y-MP vector MFLOPS (solid bars), which means that the order of the sample on the X-MP (shaded bars) is a little different from that in Fig. 4**

## Conclusion

In the mid-1980s, a selection of the new generation of supercomputers arrived at several academic sites across the United States supported in part by a new initiative of the National Science Foundation. This development raised the reasonable expectation that the scale of computational activities by academic researchers would be extended by some orders of magnitude. Such expectations were based on the known theoretical peak performance limits for numerical computation advertised by hardware vendors. The wider availability of such powerful computational resources meant that many researchers in the academic environment were exposed to supercomputing for the first time and usually without prior knowledge (or understanding) of the special requirements of these architectures for achieving peak performance. In the early 1990s, the user population at several of these national supercomputing sites is predominantly performing tasks at 10% to 18% of the achievable maximum peak performance of these advanced architectures. There can be only two possible reasons for this circumstance: either the majority of user applications on modern supercomputers are unsuited for this type of architecture, or the users are ignorant of performance metrics and of how to benefit from them. Both problems can be alleviated substantially by a planned research and development effort centered on performance issues for user applications coupled with an educational methodology. Such activities have been repeatedly called for (Infante, 1986; National Science Foundation, Division of Advanced Scientific Computing, 1989) but have yet to be widely implemented. Furthermore, performance data measurement and collection is a prerequisite to formulating policy issues on governance of supercomputer resource allocation and usage.

The experience at OSC during a three-year period was an important learning exercise in this respect. Performance Analysis Tools were designed and integrated into the resource allocation process from the earliest years with the support of the user community and the OSC management. Several important benefits to the OSC user community flowed from this development. The first was a linkage between resource allocation and application performance. The second was development of specialized performance analysis tools (PAT) enabling evaluation, compactification, and simplification of the voluminous numerical performance data reported by Cray performance monitors. The third benefit was utilization of machine-produced reports for rapid assessment of the interaction between code and architecture without detailed source code analysis. Instrumental in the success of the project was the implementation of an educational methodology, including a training program and a tutorial analysis of performance reports through a consultation service. The same methodology is now implemented at the National Environmental Supercomputing Center.

The most important achievement of the PAT project at OSC and the U.S. Environmental Protection Agency is that the user community has been sensitized to performance issues when utilizing the CRAY Y-MP. It is now common for users to conduct experimental tests of coding modifications using either PAT or UNICOS performance tools. It has been a rewarding experience to see the determination with which some users strive toward the goal of peak performance, and their satisfaction at the reward of reduced CPU time cost. To foster this activity, an incentive scheme was instituted at the OSC in the form of an award of resource units on the CRAY Y-MP for the user who demonstrated the greatest degree of performance improvement in an application. One graduate of this author's educational methodology was subsequently the recipient of the 1991 OSC Optimization Award (Elster, 1992).

The PAs described in this report have been instrumental in these successes because they helped to establish the empirical facts through the accumulation of raw data on performance for a code sample. With the PAT tools, a quantitative assessment of applications performance is possible with regard to where a particular code sits both with respect to the distribution of performance for a given sample and the theoretical maximum for either the CRAY X-MP or Y-MP. Specific features of interest described here were utilization of registers, functional units, and memory, all sorted into scalar and vector activity. To measure such features, a performance metric was defined in terms of 28 PAs that were assembled from numerical data reported by the hardware performance monitor of CRAY X-MP and Y-MP systems. It was emphasized that the performance metric chosen is characterized by a group of PAs and not by any single one, although there is a natural bias toward tracking numerically intensive floating-point computation. A full description of the PAs was given, and a list of the definitions was given in Tables 1, 3, and 5. This report demonstrated the utility of these PAs in three ways: assessment of code performance, workload analysis, and a comparison of X-MP and Y-MP performance.

In assessing performance for an individual code, the discussion of a case study showed that this choice of PAs established performance problem areas without the need of source code inspection. On the basis of this analysis, recommendations were made as to where effort is best expended in improving the code or algorithm. Such guidelines are important for the individual applications programmer who determines how much effort is required in loop modification as opposed to a major redesign of the code or even modification of the algorithm.

For workload analysis, the same performance metric was applied to a sample of applications executing on CRAY X-MP and Y-MP processors. Extensive tables and figures showed results of PAs for a 24-code sample to establish the spectrum of realistically achievable performance of a workload with values for the minimum, maximum, and mean of each PA.

To extract general performance differences between X and Y processors, group characteristics of the 24-code sample were sorted into three basic categories: computational activity, scalar to vector load balance, and execution penalty. A careful comparison of measured results for the corresponding groups of PAs was beneficial in understanding how vector floating-point performance of the Y-MP could improve by 52% compared with the X-MP while the clock period was reduced by only 29%. As a general result, the sample of codes was found to become more CPU bound on the Y-MP compared with the X-MP. A performance comparison for this sample on X and Y processors showed that overall improvements in vector MFLOPS for individual codes ranged from 19% to 79%, with a corresponding reduction in CPU time in the range of 13% to 44%.

In summary, the PAs used here provide a useful metric for individual code performance measurement, workload analysis, or machine comparison. The trends seen in the 24-code sample studied here are strongly suggestive for future development. In the science of performance analysis, as in any other, laws exist and postulates can be tested. However, a much larger sample must be studied for distinctive patterns to emerge or for signals of some fundamental empirical laws to be observed. Such a study requires a more detailed statistical analysis (G. Delic and R. I. Haller, work in progress, 1993) of population samples for user applications on the Y-MP performed over different time intervals and for different sample sizes. The indications are that the performance analysis of codes executing on supercomputers is a nascent science worthy of deeper investigation with the traditional scientific methods.


## ACKNOWLEDGMENT

The author gratefully acknowledges the support shown by the Ohio Supercomputer Center for the development of Performance Analysis Tools during his tenure as a staff member, when the work described here was performed. Alvin Stutz (Associate Director, User Services) and Professors Mike Lee (Kent State University) and Comer Duncan (Bowling Green State University), both of the Allocations Committee, have all shown a strong interest in the growth of PAT. The author is grateful to Professor Jens Villumsen (The Ohio State University) for permission to include the results of his code as the case study presented here. Hundreds of OSC users (including those of the 24-code sample) have helped this author come to a better understanding of performance issues with their many questions. Thanks are also due to Peter A. Rigsby and Robert Swanson of the Program Analysis Tools User Interface Group at Cray Research, Inc., for helpful discussions.


## BIOGRAPHY

*George Delic* holds a bachelor of science degree (with honours) from the University of New South Wales, Sydney, and in 1971 received a Ph.D. in theoretical physics from the Institute of Advanced Studies, Australian National University, Canberra. He has more than 40 journal publications in his research interest areas, which include scattering theory, computational physics, computational mathematics, numerical analysis, and supercomputer performance. Dr. Delic's career spans nine years as a research scientist in Germany and the United States and nine years as a faculty member in the Department of Physics at the University of the Witwatersrand, Johannesburg, South Africa. In 1988 he joined the Ohio Supercomputer Center in Columbus, as a senior supercomputer specialist. In 1992 Dr. Delic joined Sterling Federal Systems as principal analyst and moved to Research Triangle Park in North Carolina, where he participated in development of a user support service for the Environmental Protection Agency supercomputer user community. He is currently with Martin Marietta Technical Services, Inc., as Manager of Computational Science Services at the National Environmental Supercomputing Center in Bay City, Michigan.

## SUBJECT AREA EDITOR

Jack Dongarra

## REFERENCES


Berry, M., Chen, D., Koss, P., et al. 1989. The Perfect Club benchmarks: effective performance evaluation of supercomputers. *Internat. J. Supercomput. Appl.* 3(3):5–40.

Berry, M., Cybenko, G., and Larson, J. 1991. Scientific benchmark characterizations. *Parallel Comput.* 17:1173–1194.

Bradley, D., Cybenko, G., Gao, H., Ahmad, F., Golab, J., Larson, J., and Straka, M. 1991. Supercomputer workload decomposition and analysis. In *Proceedings of the ACM international conference on supercomputing.* New York: ACM Press, pp. 458–467.

Bradley, D. K., and Larson, J. L. 1991. Fine-grained measurement of loop performance on the CRAY Y-MP. In *Fifth SIAM conference on parallel processing for scientific applications,* edited by D. C. Sorenson. Philadelphia: SIAM.

Chen, S., Hsiung, C., Larson, J., and Somdahl, E. 1988. CRAY X-MP: a multiprocessor supercomputer. In *High speed and large-scale computing: a panoramic view,* edited by M. Ginsberg. Warrendale, Pennsylvania: Society of Automotive Engineers, pp. 57–70.

Cray Research, Inc. 1989. *UNICOS performance utilities reference manual,* SR-2040 version B (UNICOS 5.0).

Cray Research, Inc. 1990. *CRAY Y-MP system programmer reference manual,* CSM-2040-0A0.

Cray Research, Inc. Software Training. 1990. *CRAY Y-MP system architecture for applications programmers,* TR-YSAAP.

Cray Research, Inc. 1991a. *UNICOS performance utilities reference manual,* SR-2040 version 6.0.

Cray Research, Inc. 1991b. *CRAY Y-MP E series functional description manual,* HR-04016-0A.

Delic, G. 1991. Performance analysis of a 24 code sample on CRAY X/Y-MP systems at the Ohio Supercomputer Center. In *Fifth SIAM conference on parallel processing for scientific applications,* edited by D. C. Sorenson. Philadelphia: SIAM.

Elster, C. 1992. Microscopic calculations of nuclear scattering. *Ohio Supercomputer Center Visions* Fall:11–15.

Faanes, G. J., and Schwarzmeier, J. L. 1989. Comparing the performance of CRAY Y-MP and CRAY X-MP computer systems. *Cray Channels,* Winter:26–30.

Ginsberg, M. 1989. Computational environment influences on the performance of crashworthiness programs. In ASME AMD, vol 106. *Crashworthiness and occupant protection in transportation systems,* edited by T. B. Kahlil and A. I. King. pp. 11–21.

Infante, E. F. 1986. An agenda for improved evaluation of supercomputer performance, a report prepared by the Committee on Supercomputer Performance and Development. Washington, D.C.: National Academy Press.

Larson, J. 1986. CRAY X-MP hardware performance monitor. *Cray Channels,* Winter:18–19.

Larson, J., and Lutz, R. 1985. Perftrace user guide. Technical Report. Cray Research, Inc.

Levesque, J. M. and Williamson, J. W. 1989. *A guidebook to Fortran on supercomputers.* New York: Academic Press.

McMahon, F. H. 1986. The Livermore Fortran kernels: a computer test of the numerical performance range, UCRL-53745. Livermore, California: Lawrence Livermore National Laboratory.

National Science Foundation, Division of Advanced Scientific Computing. 1989. Report of the Workshop on Scientific Computing Performance Analysis, August 29–31, Boulder, Colorado.

Patterson, D. A., and Hennessy, J. L. 1990. *Computer architecture: a quantitative approach.* San Mateo, California: Morgan Kaufmann Publishers.

Robbins, K. A., and Robbins, S. 1990. *The CRAY X-MP/Model 24: a case study in pipelined architecture and vector processing.* New York: Springer-Verlag.

Villumsen, J. V. 1989. A new hierarchical particle-mesh code for very large cosmological N-body simulations. *Ap. J. Suppl.* 71: 407–431.

West, E. 1989. X-MP system wide performance analysis. *Center for Large Scale Computation News* (Ontario) 3(1):15–16.

Williams, E., Myers, C. T., and Koskela, R. 1990. The characterization of two scientific workloads using the CRAY X-MP performance monitor. In *Proceedings of Supercomputing '90.* Los Alamitos, California: IEEE Computer Society Press, pp. 142–152.