Relationship between average and real memory behavior

Kay A. Robbins Steven Robbins

The University of Texas at San Antonio

Author Affiliation:
The authors are faculty members in the computer science program at the University of Texas at San Antonio
Mailing address:
Division of Mathematics, Computer Science, and Statistics
University of Texas at San Antonio
6900 North Loop 1604 West
San Antonio, TX 78249
Phone: 210-691-5543 Fax: 210-691-4439
Email:
krobbins@ringer.cs.utsa.edu
srobbins@ringer.cs.utsa.edu

Abstract —
The Cray Y-MP has a non-intrusive hardware performance monitor which accurately accumulates certain data about program performance. This paper examines the relationship between the averages obtained from the hardware performance monitor and actual memory behavior of the Perfect Club Benchmarks run on a single processor of an eight-processor Cray Y-MP. I/O and instruction buffer fetches are not considered. The vectorized programs show regular behavior characterized by dominant vector lengths and interburst times. The distribution of vector lengths is not well-predicted by hardware performance monitor averages. Scalar programs also exhibit some clumping of memory references but have less temporal regularity than the vectorized programs. While overall port utilization is surprisingly low, there is considerable cyclic variation, and all of the ports tend to experience their maximal loading at the same time. A simple probabilistic model is developed to allow estimation of port utilization from hardware performance monitor data. The results can be used as a guide for generating more realistic synthetic memory workloads and port utilization estimates for shared memory machines.

Key words: hardware performance monitor, benchmarks, Perfect Club, Cray Y-MP, memory utilization, shared memory.

1 Introduction

The Cray Y-MP family is unique in its provision of a built-in hardware performance monitor which is non-intrusive and user-accessible. The hardware performance monitor consists of a set of performance counters which can be turned on and off at runtime. Many tools for program profiling are built around this performance monitor. This study examines the time evolution and memory performance of real programs and the relationship of this performance to hardware performance monitor (HPM) data on a Cray Y-MP processor. The test suite used for the study is the Perfect Club Benchmarks[6]. This collection of 13 programs is drawn from a variety of scientific and engineering application areas. The programs range from highly vectorized (e.g. FLO52) to almost completely scalar (e.g. SPICE). The temporal behavior of these real programs is examined over entire executions on a single Cray Y-MP processor and interburst times are computed as a function of vector length. The distribution of interburst times is indicative of the expected peak loading during program execution.

Several performance studies based on HPM data have appeared in the literature. A hardware performance monitor study of the Perfect Club Benchmarks and other standard suites was presented in Berry, Cybenko, and Larson [1]. Williams, Myers and Koskela [14] assessed HPM data collected at two supercomputing sites over several months. They presented general distribution of workload characteristics over a large number of programs. Bradley et. al [3] developed SAD (Supercomputing Applications Database) and presented a detailed workload analysis of the workload at NCSA (National Center for Supercomputing Applications). They compared the performance of the SAD database with a number of standard benchmark suites and concluded that the Perfect Club Benchmarks most closely represented the performance distribution of the database. Gao and Larson [7] have analyzed the group 0 HPM data collected over a 13 month period at NCSA. They looked at the distribution of memory reference rates over the programs run during this period. These studies present HPM averages over entire executions.

A fine grain HPM study was performed by Malony, Larson, and Reed [9] who analyzed the Perfect Club suite using a subroutine-level tracing facility for the Cray X-MP and Cray 2. They used Cray compiler support for automatic instrumentation to dump and analyze hardware performance monitor data on exit and entry to subroutines. Subroutine-level tracing is a general, relatively low-cost technique for obtaining dynamic information about program behavior.

Vajapeyam, Sohi, and Hsu [13] decomposed programs into basic blocks and used the Cray tool `jumptrace` to obtain execution frequencies of the basic blocks. They also obtained hardware performance monitor data for the entire programs. The basic-block technique is also a relatively fast, low overhead method of obtaining information about program execution. The technique does not provide dynamic information about program execution. In addition, they did not profile library routines in the basic blocks, and since the vector length was not available, they used average vector lengths to estimate utilization.

Bradley and Larson [4] developed measures for internal program parallelism and applied an instruction-level simulator to some of the Perfect Club Benchmarks. They simulated a subset of the calling tree of the program in order to avoid excessive computation and used these results to obtain estimates for the program as a whole. They calculated the fraction of elapsed time in which there were one, two or three simultaneous memory operations for selected programs. They also measured the fraction of elapsed time in which one, two or three simultaneous floating point operations occurred.

The approach taken in this paper is complementary to the above studies. In order to obtain detailed temporal information such as interburst time distributions and stride distributions, a memory reference tracing facility in `sim` was used. `sim` is a Cray Y-MP simulator which provides exact, instruction-level timing of programs run on a single Cray Y-MP processor. `sim` takes a fully-linked object module as input and simulates program execution.

Unfortunately a program run under `sim` on a Cray Y-MP executes more than 1000 times slower than it would take to execute directly on a Cray Y-MP processor, and the memory reference trace files generated by `sim` are very large. The shorter traces were 200 to 500 megabytes in compressed format. The larger benchmarks had to be checkpointed and run in pieces to produce files that could fit on a single disk. The results of these runs provide detailed timing information and some new observations about the memory characteristics of programs.

The organization of the remainder of this paper is as follows. In the next section, a description of the measurement methodology is presented. In section 3 a brief overview of the Cray Y-MP memory architecture is presented, and in section 4 a summary of the hardware performance monitor results for the Perfect Club Benchmarks is given. Section 5 discusses the relationship between strides and memory conflicts, section 6 discusses vector length estimation, and section 7 examines the interburst times. Port utilization is discussed in section 8, and a model for port utilization is developed in section 9. Finally conclusions about the techniques used in this paper are discussed. A complete set of data is available in [11].

2 Methodology

`sim` is a Cray Y-MP simulator which provides exact, instruction-level timing of programs run on a single Cray Y-MP processor. `sim` does not take into account memory conflicts. The memory tracing facility for `sim`, which became available with release 6.0 of UNICOS, generates a file which has an entry containing the opcode, the time of issue, the starting address, the stride, and the vector length for each memory reference. The `sim` memory trace file was then used as input to a program which profiles port utilization, vector lengths, strides, and interburst times. The utilization results reported in this paper were based on averages over consecutive, nonoverlapping windows of 10,000 cycles. Since the maximum vector length is 64, a window size of 10,000 could include on the order of 100 vector instructions and was considered to be representative of local behavior. Larger window sizes show less structure, while smaller window sizes show wider variation in utilization behavior. The utilization program also kept track of interburst times as a function of vector length for each port, the interburst times for scalar reads and writes, and the distribution of strides.

Hardware performance monitor data was also gathered for the Perfect Club suite. The monitor consists of eight counters which can be set to nonintrusively accumulate occurrences of eight types of events. The four possible groups of eight events which can be monitored are summarized in Table 1. Group 0 contains general counts, group 1 counts various types of holds, group 2 summarizes memory events, and group 3 summarizes instruction distributions. Each group also includes a count of the total clock cycles of execution. Only one of the four groups (0, 1, 2, 3) of events can be captured at a time. Each program was run four times in order to accumulate a complete set of 32 hardware performance counts. The hardware performance monitor results were then interpreted in terms of the activity shown in the `sim` traces.

All of the `sim` runs were made in batch processing mode on a single processor of a Cray Y-MP model 8/864 with 512 megabytes of memory at the University of Texas Center for High Performance Computing (UTCHPC). The baseline version of the Perfect Club Benchmarks were compiled using version 4.0.4.4 of the Cray Fortran compiler under version 6.14 of Unicos. The programs were compiled using the same options as in [6]. Full vector optimization was turned on and double precision was turned off. No other optimizations were applied to the program code.

3 Cray memory architecture

The Cray Y-MP only supports transfers between memory and registers. There are no memory to memory instructions, and functional unit operands are held in registers. There are four types of memory instructions: scalar (A and S registers), scalar block (B and T registers), vector (V registers), and gather/scatter (V registers). Scalar memory instructions are single element reads or writes to scalar registers. Scalar block memory instructions transfer consecutive memory locations to or from the scalar block registers on consecutive cycles. Vector operations transfer between vector registers and memory locations which are a fixed distance apart (the stride). Up to 64 elements may be read or written in a single vector memory instruction. The Cray Y-MP also supports vector gather/scatter operations in which the addresses of the vector elements are contained in another vector register. The addresses are not required to satisfy any interrelationships. The hardware performance monitor counts B and T block, vector, and gather/scatter references as block references.

The Cray Y-MP processor has a separate instruction control unit which is capable of issuing a 16-bit instruction every cycle or a 32-bit or a 48-bit instruction every other cycle. Once an instruction has been issued to the appropriate functional unit or memory port, the instruction control unit proceeds with the next instruction provided the required registers and functional units are available. When an instruction is blocked from issuing, a hold occurs. The hold issue conditions are summarized in group 1 of the hardware performance monitor as listed in Table 1.

Each processor of the Cray Y-MP system has four ports (A, B, C, and D) and up to three instructions may be accessing memory simultaneously. Port D only handles I/O requests and instruction buffer fills and is not considered here. All writes are performed on port C. Scalar, vector, and B block reads are performed on port A. Vector and T block reads are performed on port B. Each port is capable of initiating a vector element memory request on each cycle provided there are no conflicts. Note that the term hold will be used to describe the blocking of an instruction issue, and the term conflict will be used to designate blocking of a port memory request.

The utilization program uses the Cray Y-MP port assignment strategy as now described. All scalar, block, and vector writes are performed on port C. Vector and block reads are performed on ports A and B. The default port for such reads is port B. Scalar reads are performed on port A. Scalar and vector operations are not allowed to proceed simultaneously. A summary of the memory operation times is presented in Table 2. The times are expressed in units of the processor cycle time (denoted by CP) which is six nanoseconds for the Cray Y-MP. Setup and shutdown times are counted in the port utilization. Scalar memory instructions use a port for five cycles, but consecutive scalar instructions can issue without delay. The impact of memory requests for a given processor is measured by the port utilization. Once a write request has been issued to memory from a port, the processor experiences no further delay. Read requests must return a value. The delay that a destination register experiences in waiting for the read to complete is called the latency. For writes the latency is the delay incurred until the source register becomes available.

The Cray Y-MP system has a memory interleaving scheme in which consecutive addresses are mapped to consecutive sections. The two least significant bits of the address give the section number, and the next three bits designate the subsection within the section. There are 32 subsections and each processor can access each subsection every five processor cycles. Thus only strides which are multiples of eight will give memory conflicts with references from the same vector operation. The subsections are further subdivided into banks which have a cycle time of five processor cycles. Different processors can access a subsection on consecutive cycles provided that the references are directed to different banks.

4 Hardware performance monitor data

The Perfect Club Benchmark suite was selected for this study because it is well-known and represents real applications from a variety of fields. Following [13] TRACK, SPICE, and QCD are classified as scalar, while BDNA, MG3D, FLO52, ARC2D, SPEC77, and MDG are classified as vectorized. The remaining benchmarks (TRFD, DYFESM, OCEAN and ADM) are assumed to be mixtures in [13] but are almost completely vectorized as far as memory references are concerned.

A summary of the hardware performance monitor measurements for the Perfect Club Benchmarks is given in Table 3. The programs are ordered by increasing length of execution. The longest one, MDG, runs for over 28 billion cycles.

The fraction of reads, denoted by F_r, is computed by $([2,6]-[2,5])/[2,6]$. The notation $[n,m]$ is used to designate the m-th counter in group n. F_r varied from .58 to .71 which is consistent with the commonly assumed 2/3 fraction of reads. The level of vectorization is commonly specified by the fraction of floating point operations which are due to vector instructions. This value can be obtained from the hardware performance monitor data as [3,7]/([0,3]+[0,4]+[0,5]). An alternative measure is the fraction of memory references which come from vector references. This information is not available directly from the hardware performance monitor. The fraction of block references shown in Table 3 includes scalar block references as well as vector references. It is computed from $([2,6]-[2,4])/[2,6]$. As expected, the two estimates are close except for SPICE, TRACK, and QCD. Each of these has a significant number of scalar block memory instructions.

5 Memory conflicts and strides

The hardware performance monitor provides a count of the number of memory conflicts as shown in Table 3. A memory conflict occurs when a memory request is blocked at a port on a given cycle. The exact number of conflicts depends on how other processes on the system are accessing memory during the run. The Cray Y-MP/8 at UTCHPC has an extremely high utilization, and during most runs the other seven processors were active running other jobs. Thus, there will be a certain number of ambient memory conflicts due to non-related jobs which are sharing the same memory system. The ambient conflict rate appears to be about 5%. This estimate is based on results from ARC2D which should have virtually no internal memory conflicts. 99.88% of the memory references in ARC2D are vector references. Most of ARC2D's references are stride 1 while the remaining are stride 2. ARC2D was run under a variety of loading conditions, and the average percentage of memory conflicts was computed to determine the ambient conflict rate.

Stride information is not available from the hardware performance monitor, but this information can be extracted from the `sim` traces. Table 4 shows the percentage of some selected strides for the Perfect Club Benchmarks. The strides have been taken modulo 32 since only strides which are multiples of eight (0, 8, 16, and 24) cause internal memory conflicts. The only benchmarks with a significant number of bad strides are applications which contain fast Fourier transforms and integral transforms. These benchmarks (TRFD, OCEAN, ADM and SPEC77) also have a relatively large percentage of memory conflicts.

In one of the benchmarks, SPEC77, there were more memory conflicts than actual memory references. The conflicts in SPEC77 can be explained in part by the presence of a large number of bad stride vectors. The fast Fourier transform algorithm which is part of SPEC77 contains a memory reference pattern of accessing memory locations which are powers of two apart for successively higher powers of two. In addition, SPEC77 had the largest number of gather/scatter operations. 61% of memory references for SPEC77 came from gather/scatters. MDG also had a significant number of gather/scatter operations, but this program did not experience many memory conflicts.

A value closely related to the conflict rate is the memory efficiency[2]. The efficiency, E is sometimes called the probability of acceptance[5]. E can be computed from the HPM data as follows:

$$E = \frac{\textit{Successful memory requests}}{\textit{Successful memory requests} + \textit{Memory conflicts}}$$

Since writes are considered to be completed once they leave the port, there is a direct correspondence between write performance and efficiency. For reads, the latency or time for the requested value to arrive at the processor is also a factor in program performance. A scalar read instruction with a single conflict will experience a (18-17)/17 $\times$100 = 5.88 % delay in having the value arrive at its destination. A vector read instruction with a vector length of 64 which experiences a conflict on each reference will have a delay of 64/(19+64)$\times$100 = 77% in having the final element arrive at its destination.

Memory designers are concerned with both the memory efficiency and the latency. As more processors are added in a shared memory system, the drop in efficiency due to interprocessor conflicts becomes an increasingly serious concern.

6 Vector length and level of vectorization

Table 5 gives vector length information for the Perfect Club Benchmarks. The hardware performance monitor provides the average vector length for floating point operations and the average vector length for integer operations. The average vector length for memory operations is not directly available from the hardware performance monitor data for the Cray Y-MP. An average memory vector length (AVL_1) can be estimated from the hardware performance monitor counts by dividing the number of vector element references by the number of vector memory instructions (($[2,6] - [2,4])/[3,5]$). The number of vector element references is estimated by taking the total memory references and subtracting the number of scalar references. AVL_1 should overestimate the average memory vector length since B and T block references are counted in the numerator of the calculation. The values in Table 5 show that AVL_1 is always greater than or equal to the actual average memory vector length computed from `sim`. The computation also assumes that the average vector length for reads and for writes is the same. The validity of the latter assumption was confirmed in the `sim` results.

In most cases the average vector length is between the floating point average vector length and the integer average vector length given by the hardware performance monitor. The exceptions are QCD, OCEAN, and MDG. A second method of computing the average memory vector length (AVL_2) is to use a weighted average of the floating point and integer vector lengths.

The average vector length for memory operations was computed from `sim` output and compared with the two estimates from the HPM data. In most cases method 2 gave a closer result. The discrepancy in method 1 can be explained by the presence of a significant block scalar references (SPICE, TRACK, QCD and ADM). The comparison of AVL_1 and AVL_2 values can be used to estimate the importance of block scalar references in the code. A second indication can be obtained by a comparison of the fraction of block references with the fraction of vector floating point references shown in Table 3. In the four cases in which there is a discrepancy (SPICE, TRACK, QCD, and ADM), there is a significant number of block scalar memory references.

Each Cray Y-MP vector register has 64 elements. The functional units take their input from registers, so long vectors are broken up into blocks of 64 elements. If a given vector is not divisible by 64, there will be one block which is smaller than 64. One would expect that programs containing large vectors or arrays would have a dominant vector length of 64. The three programs (FLO52, BDNA, and ARC2D) which fit this profile are highly vectorized as indicated by the percentage of floating point operations which are from vector instructions presented in the last column of Table 3. These programs also achieve the highest megaflop rates. The program with the largest number of length 64 vectors is ARC2D with 67% of the vectors being of length 64. Of the remaining programs, SPICE, TRACK, ADM and MG3D were dominated by very short vectors. DYFESM, QCD, SPEC77 and MDG were dominated by vectors of moderate length.

TRFD was the only program which did not appear to have a dominant vector length or a small set of dominant vector lengths. Its behavior will now be examined. TRFD is a molecular dynamics program contributed to the Perfect Club suite by IBM. TRFD spends a considerable amount of execution time in loops which have a very regular structure. The program evolves in phases as shown in Figure 1. There are five distinct phases (labeled A through E) separated by distinctive transition regions which are of relatively short duration.

The average vector length for TRFD is 22. However, the average is not indicative of the actual behavior of the program. Figure 2 shows that the program contains several dominant vector lengths (25, 30, 35, and 40). The graph plots those vector lengths that have a significant representation in each window of size 1,000,000 cycles. A thicker line indicates a stronger contribution.

Each phase of program is characterized by a dominant vector length. In region D, over 50% of the vector memory references have vector length 35 while the others are almost uniformly distributed between 1 and 34. A similar behavior is found in phase E. The dominant vector length in phase E is 40. Phases A, B, and C have dominant vector lengths of 20, 25, and 30 respectively.

The transition regions have a different structure. The region between phases D and E, for example, represents a range of about 100 million cycles and contains vector lengths which are approximately uniformly distributed between 3 and 35. However, if a smaller region is examined (e.g. 10 million cycles), a small number of vector lengths dominate.

A closer analysis of the code shows that TRFD spends most of its time in a single subroutine which is called with array sizes of 10×10, 15×15, 20×20, 25×25, 30×30, 35×35, and 40×40. Phase E corresponds to the call for 40×40 arrays. In the transition region, columns in upper and lower triangular 40×40 matrices are computed resulting in a distribution of vector lengths between one and the linear dimension of the base array.

7 Interburst times

Vectorized programs tend to spend a lot of execution time in loops in which one or two vectors are read, an operation is performed and the result is written. The statistical manifestation of this behavior is that the program has a few dominant interburst times. The interburst time is the time interval over which a port was idle immediately prior to the issue of a particular type of memory instruction. The distributions of interburst times are accumulated separately for vectors of each possible length and for scalars. The distribution of time intervals over which a port was idle after an instruction of a particular type released a port was also measured and found not to be significantly different from that based on the times prior. The interburst time provides an indication of the amount of computation between memory references. Programs with longer vectors tend to have longer interburst times. The interburst time is important in the generation of synthetic loads for testing memory efficiency [12]. A fairly realistic memory load can be generated by picking vector length, stride and interburst time from appropriate distributions.

Figure 3 shows the distribution of interburst times when the current operation was a vector operation with vector length 40. Vector length 40 operations dominate phase E of TRFD. Surprisingly, only a few interburst times are significant, but the times of significance vary among the ports. Port A has a single dominant interburst time (123 cycles), port B has three dominant interburst times (0 cycles, 8 cycles and 92 cycles), and port C has two dominant interburst times (0 cycles and 136 cycles).

8 Port utilization and port parallelism

In this section the activity of the individual ports is characterized in terms of port utilization and level of parallelism. The utilization is the fraction of cycles in which a port is making a memory request including setup and shutdown time. The utilizations for ports A, B, and C are denoted by A, B, and C respectively. The average port utilization for the Perfect Benchmarks is given in Table 6. The maximum average utilization for any port for any program is .437 for port B of ARC2D. Most average utilizations are below .3 with scalar programs having significantly lower utilizations than vectorized programs.

Low scalar utilization can be understood by considering the timing of a set of instructions which reads two values into scalar registers, multiplies them and writes the result to memory. Assuming no memory conflicts, the first scalar read instruction issues at time 1 and the read value arrives at the scalar destination register at time 18 . A scalar reference which is performed in isolation uses the port for five cycles. However, if another scalar reference attempts to issue while the port is being used for a scalar operation, it may proceed directly. The second scalar read issues at time 3 and occupies the port for five cycles. The second read value arrives at its destination register at time 20. The two values are multiplied (8 cycles) and then written from port C. The destination register is available immediately after the issue of the write. In this simple example port A had a utilization of $7/28 = .25$ and port B had a utilization of 0. No other instructions could issue until the read values arrived at time 20, since the floating point multiply holds issue. The write can be overlapped with the next set of reads giving a port C utilization of $5/28 = .18$. The maximum number of megaflops obtained from this scenario is one operation in 28 clock pulses or six megaflops. In order to obtain higher performance for scalar operations, memory fetches for subsequent instructions must be scheduled in advance of previous floating point functional unit instructions so that they won't be blocked from issue. Keeping additional scalar values in registers or allowing conditional issue would also increase performance.

Now consider the timing of a set of instructions to load two vector registers, perform a floating point multiply and store the result back to memory. An important factor which contributes to efficiency for vector operations on the Cray Y-MP is chaining. With chaining, a vector instruction can issue before the source registers are available. In the example, the floating point multiply instruction can issue immediately after the load requests and the setup for the floating point unit can be overlapped with the read latency. As the individual elements arrive from memory, they are directed to the floating point unit. The store operation can also be chained so that port C can accept the element results as they are being produced. The key is that chaining allows subsequent instructions to be issued and so much higher performance can be obtained without excessive compile-time analysis.

Consider the case where the vector length is 64. The first read instruction issues to port B at time 1 and the first read element value is returned at 21. The second read instruction issues at time 2 and the first read element value is returned at time 22. In the meantime, the floating point unit can set up (3 cycles) and prepare for chaining. The first product emerges from the floating point unit at time 32. The write instruction to port C can issue and 3 cycles of setup can be overlapped with the floating point calculations. The last element read value from port A is returned at 87 and the last value emerges from the floating point unit at 95. Since ports A and B can be immediately rescheduled for the next instructions, it is possible to obtain a port utilization of near 1 and a computational performance of 122 megaflops. Higher megaflop rates can be obtained if additional floating point operations can be performed before the results are written to memory. Since scalar and vector port usage are not allowed to overlap, interspersed scalar operations can degrade port utilization for vectors.

The level of parallelism as defined by Bradley and Larson [4] is the fraction of cycles in which one, two, or three memory results are produced simultaneously for a processor. For the purposes of analyzing and modeling loads from the viewpoint of the memory and interconnection network, however, the fraction of cycles in which one, two, or three ports are active is relevant. These fractions, the level of port parallelism, are denoted by $\bar{\rho_1}$, $\bar{\rho_2}$, and $\bar{\rho_3}$ respectively.

Let T be the total number of cycles in the program, and P be the total number of cycles of port activity. P can be calculated either from the utilization (A, B, C) or from the level of port parallelism ($\bar{\rho_1}$, $\bar{\rho_2}$, $\bar{\rho_3}$).

$$P = (A + B + C)T = (\bar{\rho_1} + 2\bar{\rho_2} + 3\bar{\rho_3})T$$

The maximum value of P/T is three. Table 6 shows the levels of port parallelism computed for the Perfect Benchmarks by analyzing `sim` output. The results show that fraction of cycles in which all three ports were busy was very low for all of the benchmarks. The highest level of parallelism was achieved by FLO5, and even in this case, all three ports were busy only about 7% of the time.

Scalar code results in asymmetric utilization of the ports, since scalar references can only be made on ports A and C. The maximum value of P/T for scalar code is two. Vector code utilizes the ports in a roughly equal manner, but port B always has a higher port utilization than port A because of the priority scheme used in assigning vector references. The 2/3 rule for the fraction of reads implies that the utilization of port C should be the average of the utilization of ports A and B. In general if A, B, and C represent the utilizations (fractions of cycles that the port is active) of their respective ports, the number of reads can be estimated as $(A+B)/(A+B+C)$ times the number of memory references. Table 6 gives the ratio of the number of reads predicted from this formula with the actual number of reads measured from hardware performance monitor data (R).

The variable memory behavior of scalar and vector code is quite different as indicated by a comparison of the port utilization for ARC2D and SPICE. ARC2D is a two-dimensional fluid dynamics program which was contributed to the Perfect Benchmark suite by NASA Ames. SPICE is a circuit-simulation program which was contributed to the Perfect Benchmark suite by the Center for Supercomputing Research and Development at the University of Illinois Champaign-Urbana. Figure 4 compares the utilizations as a function of time for ARC2D and SPICE. The average port activity over windows of 1,000,000 cycles is plotted for both programs. The activity for the individual ports is similar to the combined average behavior.

In order to describe the variability of memory activity over the execution of a program, the fraction of cycles in which each port was active was computed for windows of 10,000 cycles. Histograms of the fraction of cycles active in windows of 10,000 cycles were computed using a bin size of 0.01. A comparison of the histograms for SPICE and ARC2D is given in Figure 5. As expected, port B is virtually quiescent during the execution of SPICE, while ports A and

C are relatively more active. The utilization of port A is over twice the utilization of port C, but even port A has a small utilization. In contrast ARC2D has a much broader distribution of cycles active. Figure 5 shows that port B is the most active port indicating that memory activity is dominated by vector reads.

9 Modeling utilization from hardware performance monitor data

Since `sim` runs over a thousand times slower than the program it is simulating, it is not feasible to run `sim` on general programs in order to obtain the utilization or levels of port parallelism. A method of estimating port utilization directly from the hardware performance monitor data is now developed. Vector and scalar contributions can be considered separately because Cray hardware does not permit vector and scalar port activity to overlap. U and V in Table 6 are the ratios of the read and write port utilizations predicted from the model using hardware performance monitor data to the utilizations calculated from `sim` output. The method used to compute the port utilization from the hardware performance monitor data is described in the next two subsections.

9.1 Utilization due to vector operations

Port B only performs vector and block reads, while port A performs scalar, vector and block reads. It is impossible to obtain separate vector read utilization for ports A and B from the hardware performance monitor. A combined utilization due to vector and block reads can be estimated by adding the number of vector and block references to five times the number of vector load/store instructions and multiplying the result by the fraction of vector/block memory operations which are reads. This fraction is not directly obtainable from the hardware performace monitor, but it is assumed to be the same as the fraction of reads among all memory operations. To this combined utilization must be added the contribution by scalar reads to the utilization of port A as estimated by the scalar model in the next subsection.

The vector utilization of port C can be estimated by calculating the number of cycles during which port C was issuing vector and block writes using the same technique as for vector reads. This utilization is combined with the port utilization due to scalar writes as estimated by the scalar model to give a total estimated port utilization.

9.2 Utilization due to scalar operations

A model for the contribution of scalar memory operations to port utilization is now developed. A scalar reference which is performed in isolation uses the port for five cycles. However, if another scalar reference attempts to issue while the port is being used for a scalar operation, it may proceed directly. The only way that a scalar read can contribute less than five cycles to the port utilization is if another scalar read follows closely behind it. Scalar memory references take two cycles to issue if there are no holds. Thus, a scalar read which is immediately followed by another scalar read will contribute at least two cycles to the port utilization.

A probabilistic model for the contribution of scalar reads to scalar utilization of port A will now be developed. It is based on the assumption that scalar references are distributed uniformly in time throughout the program execution. Let:

$$\begin{aligned} T &= \text{total number of cycles for the program} \\ N_{sr} &= \text{number of scalar reads} \\ U_{sr} &= \text{port utilization from scalar reads} \end{aligned}$$

Since scalar references reserve the port between two and five cycles, U_{sr} is between $2N_{sr}/T$ and $5N_{sr}/T$. Let:

$$\begin{aligned} p_r &= \text{fraction of instructions that are scalar reads} \\ r &= \text{fraction of instructions which are two or three parcel} \\ & \text{(other than scalar reads and jumps)} \\ v &= \text{fraction of instructions which are one-parcel} \\ h &= \text{probability that an instruction holds issue on a given cycle} \\ k &= (1-h) \\ R_i &= \text{probability that a scalar read utilizes a port for } i \text{ cycles} \end{aligned}$$

Given that a scalar read issues, it will hold the port for only two cycles if it is immediately followed by another scalar read instruction which does not have a hold. Thus $R_2 = p_r k$. Also, every scalar read will occupy the port for at least two cycles so $R_0 = 0$ and $R_1 = 0$.

A scalar read will hold port A for exactly three cycles if it is followed by a one-parcel instruction which issues immediately (probability vk) followed by a scalar read which also issues immediately (probability R_2). It can also take three cycles if the scalar read is followed by another scalar read which has exactly one cycle of hold (hp_rk). Thus, $R_3 = p_rvk^2 + p_rhk = p_rk(vk + h)$.

There are five distinct possibilities for the reference occupying four cycles:

1. The scalar read can be followed by another scalar read which has exactly two holds (h^2p_rk).

2. It can be followed by a one-parcel instruction with one hold and then another scalar read with no holds ($vhkp_rk$).

3. The scalar reference can be followed by a one-parcel instruction with no holds and then another scalar read with one hold ($vkhp_rk$).

4. It can be followed by two one-parcel instructions with no holds and then another scalar read with no holds ($(vk)^2p_rk$).

5. It can be followed by a two parcel instruction with no holds and then another scalar read (rkp_rk).

Thus, $R_4 = p_rk(h^2 + 2vhk + v^2k^2 + rk)$. Finally $R_5 = 1 - R_2 - R_3 - R_4$ since a scalar read can occupy the port for no more than five cycles if there are no memory conflicts. The expected value of the utilization of port A due to scalar references is:

$$U_{sr} = \frac{N_{sr}}{T}\sum_{i=2}^{5} iR_i$$

A similar analysis and model can be applied to scalar writes for port C.

The above model for scalar references can be combined with vector references to allow an estimation of the port utilization directly from the data provided by the hardware performance monitor. The advantage of such an approach is that the model can be easily and inexpensively applied to HPM data for very long programs to estimate port utilization.

9.3 Evaluation of model parameters

In this section a method of estimating the model parameters from hardware performance monitor data is described. The value of T is part of the counts for all groups. The group 2 statistics give the total number of scalar memory references and the total number of reads. N_{sr} can be estimated if it assumed that the fraction of scalar memory references which are reads is the same as the overall fraction of memory reads. p_r is then computed as N_{sr} divided by the total number of instructions. The most frequently occurring two-parcel instructions are scalar memory operations and jumps. The jump instructions, in fact, delay issue for at least 5 cycles if the jump is taken so they will not be counted in r. The one-parcel instructions are estimated as the total number of instructions minus the scalar memory operations and jumps. The probability of a hold, h, is estimated as the total number of holds over the total number of cycles. Note: $p_r + r + v \neq 1$ because jumps are assumed to block issue for more than four cycles on the average and therefore don't affect the computation.

The number of vector reads is not explicitly available from hardware performance monitor data. However, it can be estimated by calculating the total fraction of reads and assuming that the vector reads make up the same fraction of the vector memory operations as reads do for memory references as a whole.

In order to estimate the port utilization due to vector operations, the average vector length and the number of vector reads and writes must be estimated. The average vector length can be estimated by dividing the total number of vector memory element operations by the total number of vector memory instructions. The number of vector reads can be estimated by multiplying the fraction of reads by the total number of vector memory operations. A similar estimation applies for writes. The vector memory operations and the block memory operations are lumped into a single count, but for most programs the count is dominated by the vector operations.

A comparison between the port utilization calculated from the hardware performance monitor data using the model and the port utilization derived from `sim` output is given in the last two columns of Table 6. U is the ratio of the sum of the A and B port utilizations predicted from the hardware performance monitor model divided by the sum of the A and B port utilizations measured from `sim` data. Similarly V is the hardware performance monitor model prediction of the port C utilization divided by the port C utilization measured from `sim` data.

The agreement is surprisingly good considering the simplicity of the model. `sim` ignores memory conflicts while the hardware performance monitor includes them. The value of T is longer in the HPM data due to this factor which lowers the prediction of utilization from the hardware performance monitor model. The model also relies on the assumption that the references are uniformly distributed. This assumption has no effect on the prediction of utilization due to vector operations, but it is an important consideration for scalar references. The model, which is based on a uniform distribution of references, predicts that a small number of scalar operations will take fewer than five cycles. In SPICE for example, the model predicts that 8% of the scalar reads and 3% of the scalar writes take two cycles. The utilization program for `sim` measured the actual number of cycles for scalar references. 28% of the scalar reads and 20% of the scalar writes for SPICE took two cycles. 61% of the scalar reads and 68% of the scalar writes took five cycles. This measurement implies that the model should overestimate scalar utilization for SPICE which is in fact the case. This result also indicates that the scalar overlap capability of the Cray Y-MP memory ports is an important performance factor for scalar programs. The main discrepency appears to be a result of the assumption of the uniformity of scalar references. The `sim` results show that scalar references tend to be clumped, and more scalar references have an interburst time of two than is predicted by the model.

10 Conclusions

The techniques used in this study are extremely costly in computer time, but they provide a basis for simple estimation of memory and port utilization directly from hardware performance monitor data. Although there is considerable variability in behavior across the Perfect Club benchmark suite, there are several general observations which can be made.

The analysis suggests that the low performance of scalar code on the Cray Y-MP is caused by scalar memory latency delays which prevent the issue of succeeding instructions. These delays could be reduced by introducing conditional issue hardware or by using compile-time techniques to preschedule reads. Since vector instructions allow chaining, issues are not prevented for vector instructions.

Vectorized programs which experience memory conflicts of greater than 15% are likely to contain a large number of bad strides. Portions of code with a dominant vector length also tend to have a small set of dominant interburst times. Although the average utilization for the typical program is not high, there are periods of high utilization of all ports. These periods of high utilization tend to occur at somewhat regular intervals corresponding to computationally intensive loops. Synthetic memory loads which consist of a continuous stream of requests forwarded at the maximal rate do not accurately reflect the behavior of real programs, at least for the Cray Y-MP architecture where values must be read into vector registers before they can be used by the functional units.

Synthetic memory models for scalar references should allow for the asymmetry of port utilization and should assume a distribution of time between references rather than a fixed probability of a reference occurring. A reasonable model for vector references assumes a relatively broad and fairly uniform distribution centered around the average utilization rate. The probabilistic model can also be used in conjunction with hardware performance monitor data from real programs to generate synthetic reference streams which are a combination of vector and scalar references representing the average behavior of specific programs.

One of the difficulties in using Cray Y-MP data for general program modeling is that the hardware performance monitor does not give direct counts for various instructions. The counters for the Cray C90 have been reorganized and somewhat remedy this problem. There are 32 simultaneous counters on the C90 and there is more of direct correspondence between the counters and various types of instructions, making evaluation of model parameters more direct.

Acknowledgments:

Greg Faanes of Cray Research provided help with `sim`. Charles Grassl of Cray Research provided useful advice on compiling the Perfect Club Benchmarks. The authors would also like to thank one of the anonymous referees for providing many useful suggestions for improving the paper. This work was partially supported by Cray Research, the University of Texas System for High Performance Computing, and the University of Texas at San Antonio Faculty Research Awards program.

References

[1] Berry, M., Cybenko, G., and Larson, J. 1991. Scientific benchmark characterization. *Parallel Computing*, 17, 1173–1194.

[2] Bailey, D. H. 1987. Vector computer memory bank contention. *IEEE Trans. on Computers*, C-36, 293–298.

[3] Bradley, D., Cybenko, G., Gao, H., Larson, J., Ahmad, F., Golab, J., and Straka, M. 1991. Supercomputer workload decomposition and analysis. *Proc. ACM Intl. Conf. on Supercomputing*, 458–467.

[4] Bradley, D. K., and Larson, J. L. 1993. A parallelism-based analytic approach to performance evaluation using applications programs. *IEEE Proc* to appear (August).

[5] Briggs, F. A., and Davidson, E. S. 1977. Organization of semiconductor memories for parallel-pipelined processors. *IEEE Trans. on Computers*, C-26, 162–169.

[6] Cybenko, G. 1991. Supercomputer performance trends and the Perfect Benchmarks. *Supercomputing Review*, Apr, 53–60.

[7] Gao, H., and Larson, J. L., 1993. A year's profile of academic supercomputer users using the Cray hardware performance monitor, CSRD Report 1300, 1993.

[8] Larson, J. L. 1991. Collecting and interpreting hpm performance data on the Cray Y-MP. *NCSA Datalink*, (Nov-Dec).

[9] Malony, A. D., Larson, J. L., and Reed, D. A. 1990. Tracing application program execution on the Cray X-MP and Cray 2. *Supercomputing '90*, 60–73.

[10] Robbins, K. A., and Robbins, S. 1991. Bus conflicts for logical memory banks on a Cray Y-MP type processor system. *Proc. Intl Conf. Parallel Proc.*, 21–24.

[11] Robbins, K. A., and Robbins, S. 1992. Experimental assessment of the Perfect Club Benchmarks on a Cray Y-MP. *UTSA Tech. Rep. UTSA-CS-92-102*.

[12] Robbins, K. A. and Robbins, S. Buffered banks in multiprocessor systems. *IEEE Trans on Computers* to appear.

[13] Vajapeyam, S., Sohi, G. S. and Hsu, W.-C. 1991. An empirical study of the Cray Y-MP processor using the Perfect Club Benchmarks. *18th Intl. Sym. Comp. Arch.*, 170–179.

[14] Williams, E., Myers, C. T., and Koskela, R. 1990. The characterization of two scientific workloads using the Cray X-MP performance monitor. *Supercomputing '90*, 142–152.

List of Tables

List of Figures

	Group			
	0	**1**	**2**	**3**
Counter	**General**	**Hold Issue CPs**	**Memory**	**Insts or Ops**
0	Insts	Semaphores	Ibuf fetches	Jump
1	Issue Holds	Shared regs	Fetch confs	Scalar FU/BT
2	Ibuf fetches	A regs/FUs	I/O refs	Scalar mem
3	FP adds	S regs/FUs	I/O confs	V int/log
4	FP mults	V regs	Scalar mem refs	V FP
5	FP recips	V FUs	Mem writes	V mem
6	Mem refs	Scalar mem	Mem refs	V int/log ops
7	I/O refs	Block mem	Mem confs	V FP ops

Table 1: Summary of hardware performance monitor counters. FP = floating point, CPs = clock pulses, FUs = functional units, Ibuf = instruction buffer, BT= B and T registers.

Instruction	**Issue**	**SU+SD**	**PU**	**Latency**
A read	2	0	5	18
S read	2	0	5	17
A write	2	0	5	0
S write	2	0	5	0
B read	1	5	$5+L$	$19+L$
T read	1	5	$5+L$	$19+L$
B write	1	6	$6+L$	$3+L$
T write	1	6	$6+L$	$3+L$
V read	1	5	$5+L$	$19+L$
V write	1	6	$6+L$	$3+L$
Gather	1	10	$10+L$	$24+L$
Scatter	1	10	$10+L$	$3+L$

Table 2: Summary of times for memory instructions on the Cray Y-MP. L denotes the vector or block length, SU is the port setup time, SD is the port shutdown time, and PU is the port utilization.

Program	Cycles	Mflps	Instructions Total	Instructions Mem	CPU Mem Refs	I/O Mem Refs	CPU Mem Confs	% Mem Confs	E	Frac (F_r) Mem Reads	Frac Block Refs	Frac Vect FP ops
FLO52	902	119	239	23	717	3.8	38	5	.95	.71	.99	.98
TRFD	1321	55	525	42	670	4.1	248	37	.73	.66	.98	1.00
SPICE	1363	7	327	88	101	.5	11	11	.90	.71	.15	.07
BDNA	1534	128	301	34	669	.4	42	6	.94	.59	.97	1.00
TRACK	1696	8	497	70	124	.04	6	5	.95	.66	.45	.08
DYFESM	2029	45	580	50	833	.2	49	6	.94	.58	.98	.95
ARC2D	2212	168	400	40	1937	2.3	106	5	.95	.71	.99	1.00
QCD	3295	13	1234	137	323	1.4	18	5	.95	.58	.59	.01
OCEAN	3482	73	960	216	2344	8.4	223	10	.91	.51	.93	.91
ADM	3916	23	1415	164	693	1.2	223	32	.76	.61	.84	.83
SPEC77	9010	37	2898	240	1689	52.3	2045	121	.45	.70	.94	.94
MG3D	20843	78	6057	749	10370	36.4	1601	15	.87	.58	.96	.98
MDG	28824	23	8521	259	2842	4.1	186	7	.93	62	62	.67

Table 3: Hardware performance monitor data for the Perfect Club Benchmarks. All counts are in millions.

	HPM		SIM								Algorithms								
		% Mem	% G/S	% Strides mod 32															
Program	mflps	confs	refs	0	1	2	8	16	24	Others	1	2	3	4	5	6	7	8	9
FLO52	119	5	0	0.0	81.6	18.4	0.0	0.0	0.0	0.0					x	x			
TRFD	55	37	0	0.0	67.4	0.0	15.1	0.0	0.0	15.2								x	
SPICE	7	11	0	0.0	99.1	0.1	0.0	0.0	0.0	0.7	x					x			
BDNA	128	6	0	0.0	90.4	0.0	2.2	0.0	0.0	7.4						x			
TRACK	8	5	0	0.0	96.3	0.1	1.1	0.0	0.0	2.6									x
DYFESM	45	6	1	0.4	87.0	0.0	0.0	0.0	0.0	11.7	x	x				x			
ARC2D	168	5	0	0.0	99.9	0.0	0.0	0.0	0.0	0.1	x			x					
QCD	13	5	0	0.0	99.9	0.0	0.0	0.0	0.0	0.1							x		
OCEAN	73	10	0	0.0	35.0	56.8	0.0	0.0	0.0	8.2			x						
ADM	23	32	1	5.7	46.1	33.0	0.0	5.1	0.0	10.1			x						
SPEC77	37	121	61	43.2	13.1	15.0	1.1	0.0	0.0	27.5			x	x					
MG3D	78	15	0	0.0	63.9	5.4	0.0	0.0	0.0	25.2			x			x			
MDG	23	7	41	0.0	98.5	0.0	0.0	0.0	0.0	1.5						x			

Table 4: Relationship of memory conflicts, strides, and algorithms for the Perfect Club Benchmarks. G/S = gather/scatter. The algorithms are: 1) Sparse linear systems solvers, 2) Nonlinear algebraic system solvers, 3) Fast Fourier transform, 4) Rapid elliptic problem solvers, 5) Multigrid schemes, 6) Ordinary differential equation solvers, 7) Monte Carlo schemes, 8) Integral transforms, 9) Convolution.

	HPM					SIM									
		Int	Flt	Mem	Mem	Flt	Mem	%	Most Frequent Mem VL						
Program	mflps	AVL	AVL	AVL_1	AVL_2	AVL	AVL	B/T	VL	%	VL	%	VL	%	% Rest
FLO52	119	45	40	41	41	40	41	0	64	41	32	16	16	16	27
TRFD	55	16	22	22	22	22	22	0	40	14	35	9	30	6	71
SPICE	7	9	2	6	2	2	3	9	1	35	2	35	3	27	3
BDNA	128	43	51	48	49	51	48	1	64	56	16	7	59	5	32
TRACK	8	11	5	22	6	5	7	31	4	65	16	17	8	9	9
DYFESM	45	34	18	27	22	18	26	2	16	36	45	20	64	12	32
ARC2D	168	55	53	52	53	51	51	0	64	67	23	21	31	10	2
QCD	13	21	18	45	19	18	22	30	18	87	64	10	12	2	1
OCEAN	73	44	44	49	44	44	49	0	64	61	31	17	1	9	13
ADM	23	16	7	10	8	7	8	20	1	49	14	12	7	11	28
SPEC77	37	15	12	13	13	12	12	3	12	26	16	20	15	19	35
MG3D	78	52	23	28	23	23	28	2	1	26	2	21	64	16	37
MDG	23	6	7	10	7	6	10	0	14	39	4	36	9	16	0

Table 5: Vector length information for the Perfect Club Benchmarks.

Program	Parallelism $\bar{\rho_1}$	$\bar{\rho_2}$	$\bar{\rho_3}$	Utilization A	B	C	P/T	read frac	R	U	V
FLO52	.320	.207	.074	.296	.391	.268	.956	.725	.99	.97	.96
TRFD	.148	.227	.053	.192	.304	.266	.762	.660	.99	.88	.87
SPICE	.238	.024	.000	.194	.011	.082	.287	.710	1.01	1.06	1.15
BDNA	.265	.096	.032	.140	.185	.226	.551	.586	1.01	.96	.99
TRACK	.213	.010	.000	.138	.022	.074	.234	.657	1.04	.94	1.09
DYFESM	.257	.079	.046	.157	.176	.222	.555	.577	1.04	.91	1.03
ARC2D	.346	.244	.056	.269	.437	.296	1.002	.708	1.00	.97	.97
QCD	.180	.031	.004	.136	.026	.092	.254	.580	1.10	.92	1.20
OCEAN	.393	.198	.032	.229	.205	.453	.887	.514	.95	1.09	1.00
ADM	.253	.051	.005	.130	.094	.146	.370	.607	1.00	.96	1.00
SPEC77	.200	.064	.013	.112	.138	.117	.367	.700	.97	.89	.86
MG3D	.313	.134	.034	.172	.221	.291	.684	.582	.99	.97	.97
MDG	.206	.021	.001	.104	.048	.098	.250	.616	.99	1.11	1.10

Table 6: The level of port parallelism and the port utilization for the Perfect Club Benchmarks.

Symbol	Meaning	Formula
T	total program cycles	available all groups
I	total instructions	$[0,0]$
AVL_1	average vector length method 1	$([2,6]-[2,4])/[3,5]$
AVL_2	average vector length method 2	$([3,6]+[3,7])/([3,3]+[3,4])$
F_w	fraction of write operations	$[2,5]/[2,6]$
F_r	fraction of read operations	$1-F_w$
S	number of scalar memory instructions	$[3,2]$
N_{sr}	number of scalar read instructions	$S \times F_r$
N_{sw}	number of scalar write instructions	$S \times F_w$
N_{vr}	number of vector read instructions	$[3,5] \times F_r$
N_{vw}	number of vector write instructions	$[3,5] \times F_w$
U_{vr}	port utilization due to vector reads	$(5N_{vr}+[1,7] \times F_r)/T$
U_{vw}	port utilization due to vector writes	$(6N_{vw}+[1,7] \times F_w)/T$
U_{sr}	port utilization due to scalar reads	$N_{sr}(\sum_{i=2}^{5} iR_i)/T$
U_{sw}	port utilization due to scalar writes	$N_{sw}(\sum_{i=2}^{5} iW_i)/T$
A	actual utilization of port A	from `sim`
B	actual utilization of port B	from `sim`
C	actual utilization of port C	from `sim`
P	total cycles of port activity	$(A+B+C)T$
R	ratio of predicted to actual reads	$\frac{A+B}{(A+B+C)F_r}$
U	ratio of predicted to actual port A and B util	$(U_{vr}+U_{sr})/(A+B)$
V	ratio of predicted to actual port C util	$(U_{vw}+U_{sw})/C$
p_r	prob inst is scalar read	N_{sr}/I
p_w	prob inst is scalar write	N_{sw}/I
h	prob hold on a given cycle	$[0,1]/T$
k	prob not hold on given cycle	$1-h$
v	prob inst is one-parcel	$(I-[3,0]-[3,2])/I$
d	prob one delay	$h+kv$
r	prob inst is two-parcel, not read or jump	p_w
w	prob inst is two-parcel, not write or jump	p_r
R_1	prob scalar read uses port for 1 cycles	0
R_2	prob scalar read uses port for 2 cycles	$p_r k$
R_3	prob scalar read uses port for 3 cycles	$R_1 d$
R_4	prob scalar read uses port for 4 cycles	$R_1(d^2+rk)$
R_5	prob scalar read uses port for 5 cycles	$1-R_1-R_2-R_3-R_4$
W_1	prob scalar read uses port for 1 cycles	0
W_2	prob scalar read uses port for 2 cycles	$p_w k$
W_3	prob scalar read uses port for 3 cycles	$W_1 d$
W_4	prob scalar read uses port for 4 cycles	$W_1(d^2+rk)$
W_5	prob scalar read uses port for 5 cycles	$1-W_1-W_2-W_3-W_4$

Table 7: Evaluation of model parameters from HPM data.

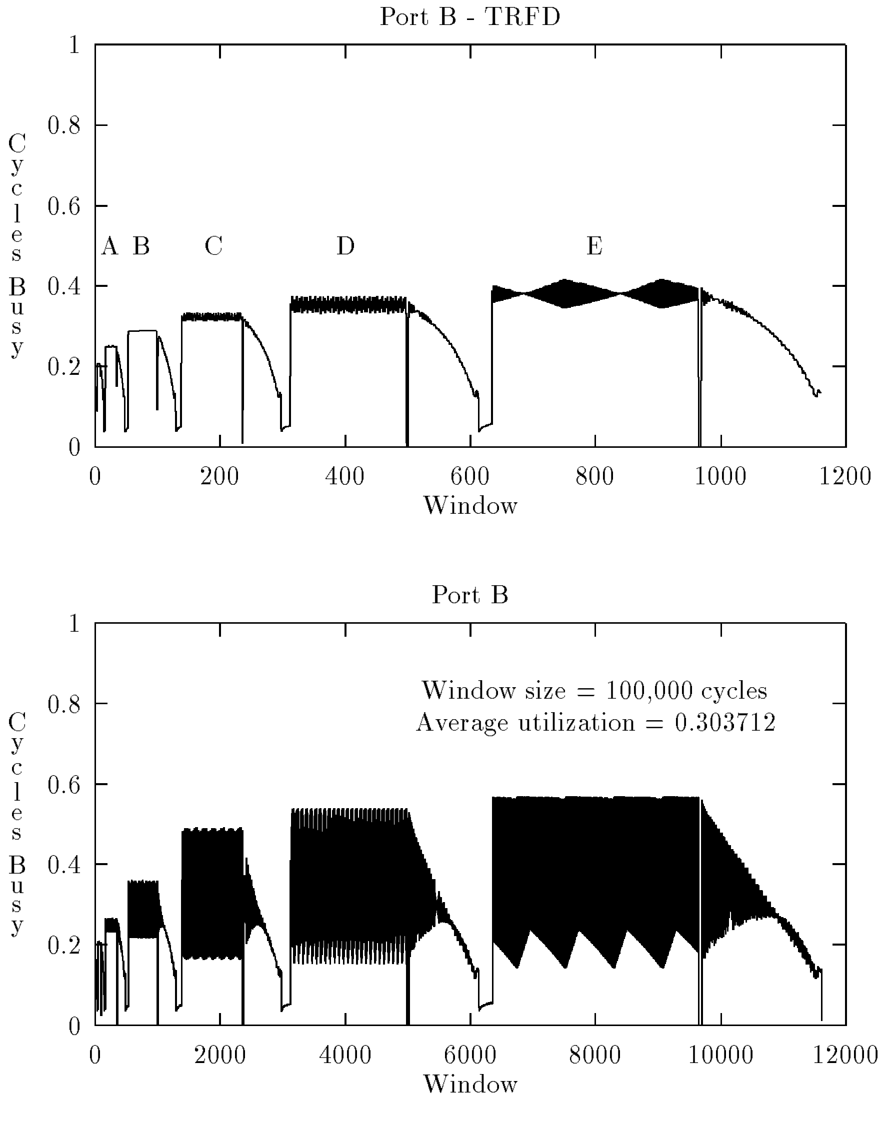


Figure 1: The utilization of port B for TRFD shows several phases separated by transition regions. In the upper graph each point represents the utilization averaged over a window of 1,000,000 cycles. When the utilization is averaged over windows of 100,000 cycles, additional structure is shown.

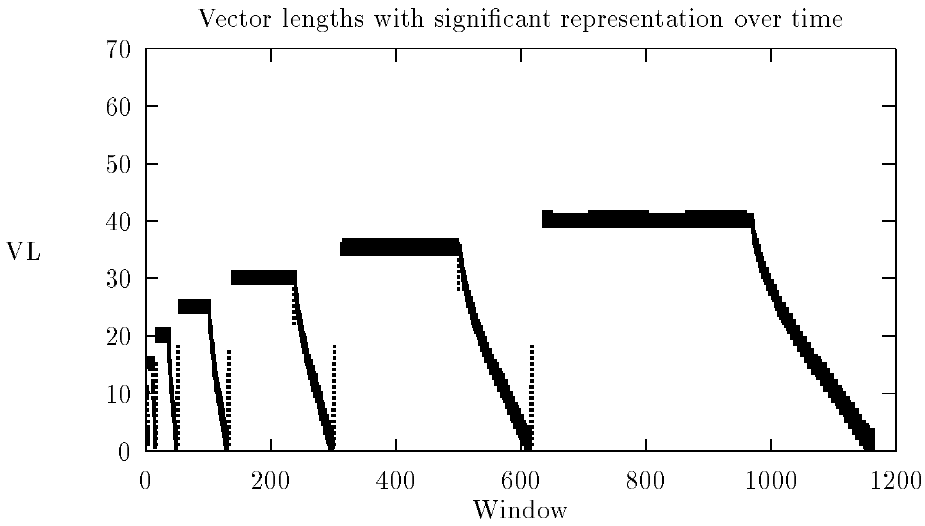


Figure 2: Distribution of vector lengths for TRFD.

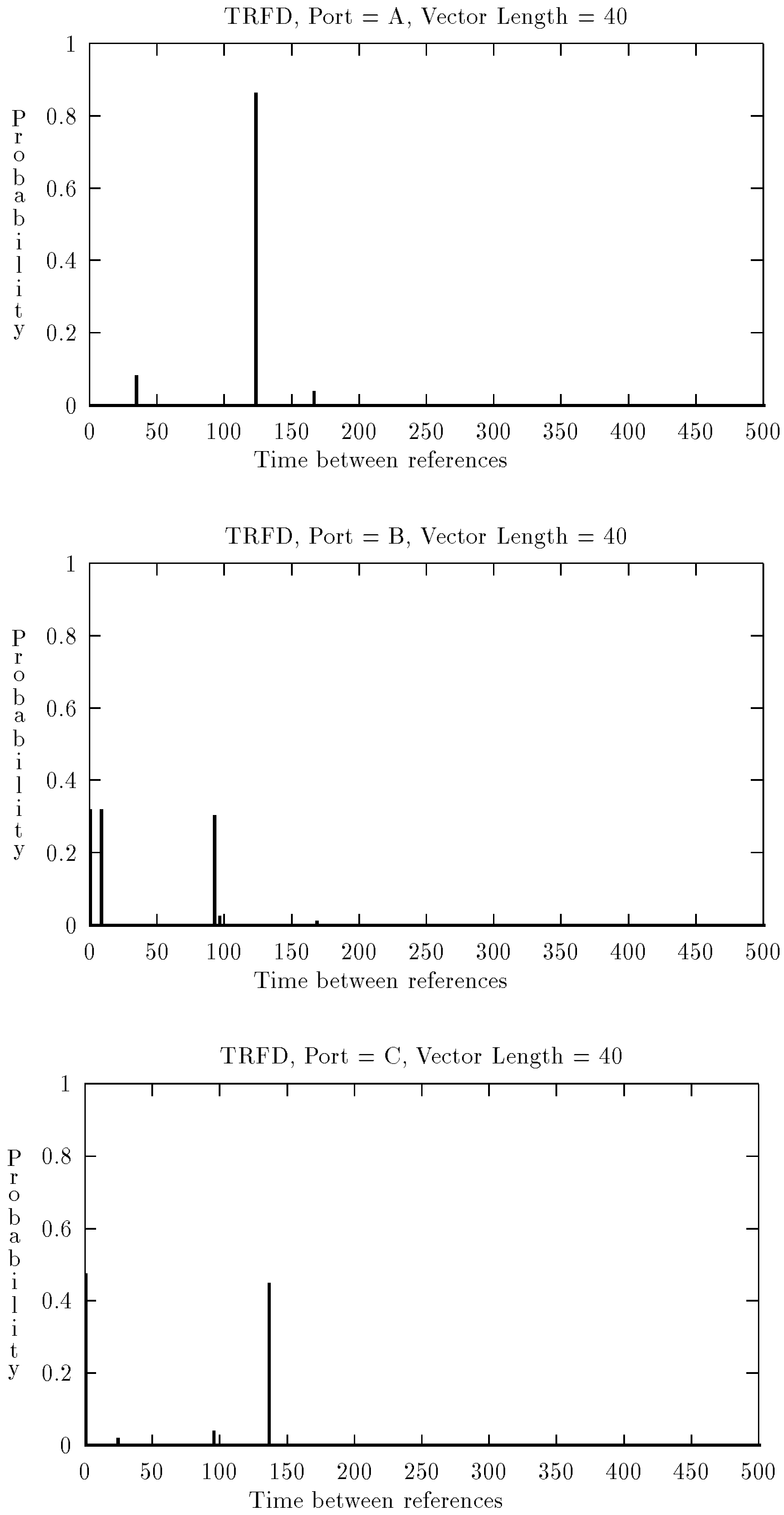
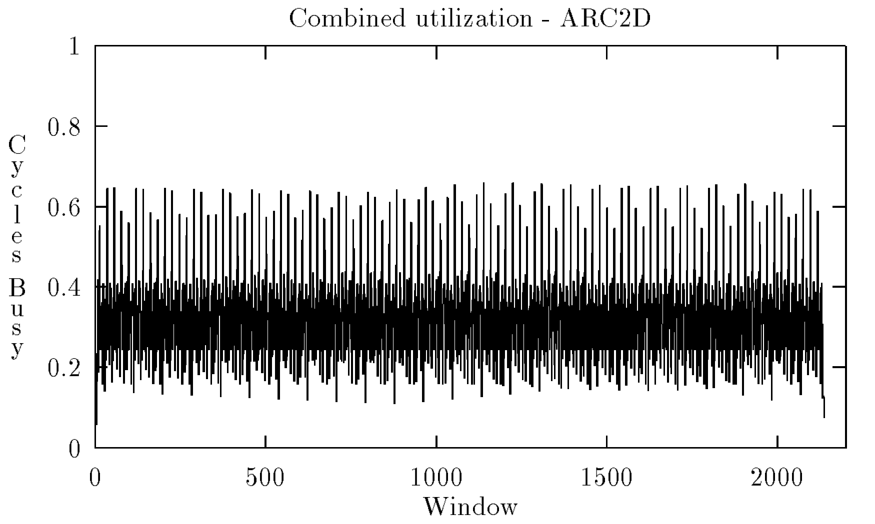


Figure 3: Distribution of interburst times for TRFD for vector length 40.



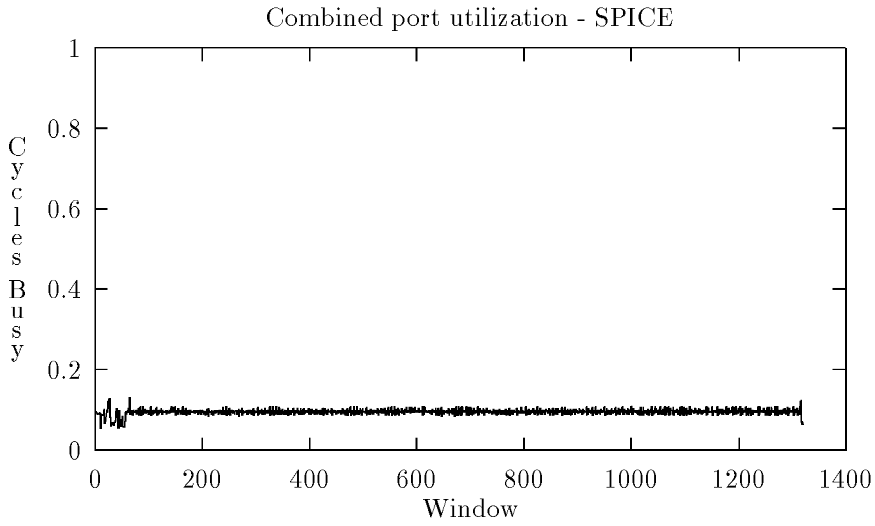


Figure 4: The average port utilization for ARC2D is compared with the average port utilization for SPICE. Windows of 1,000,000 cycles are used.

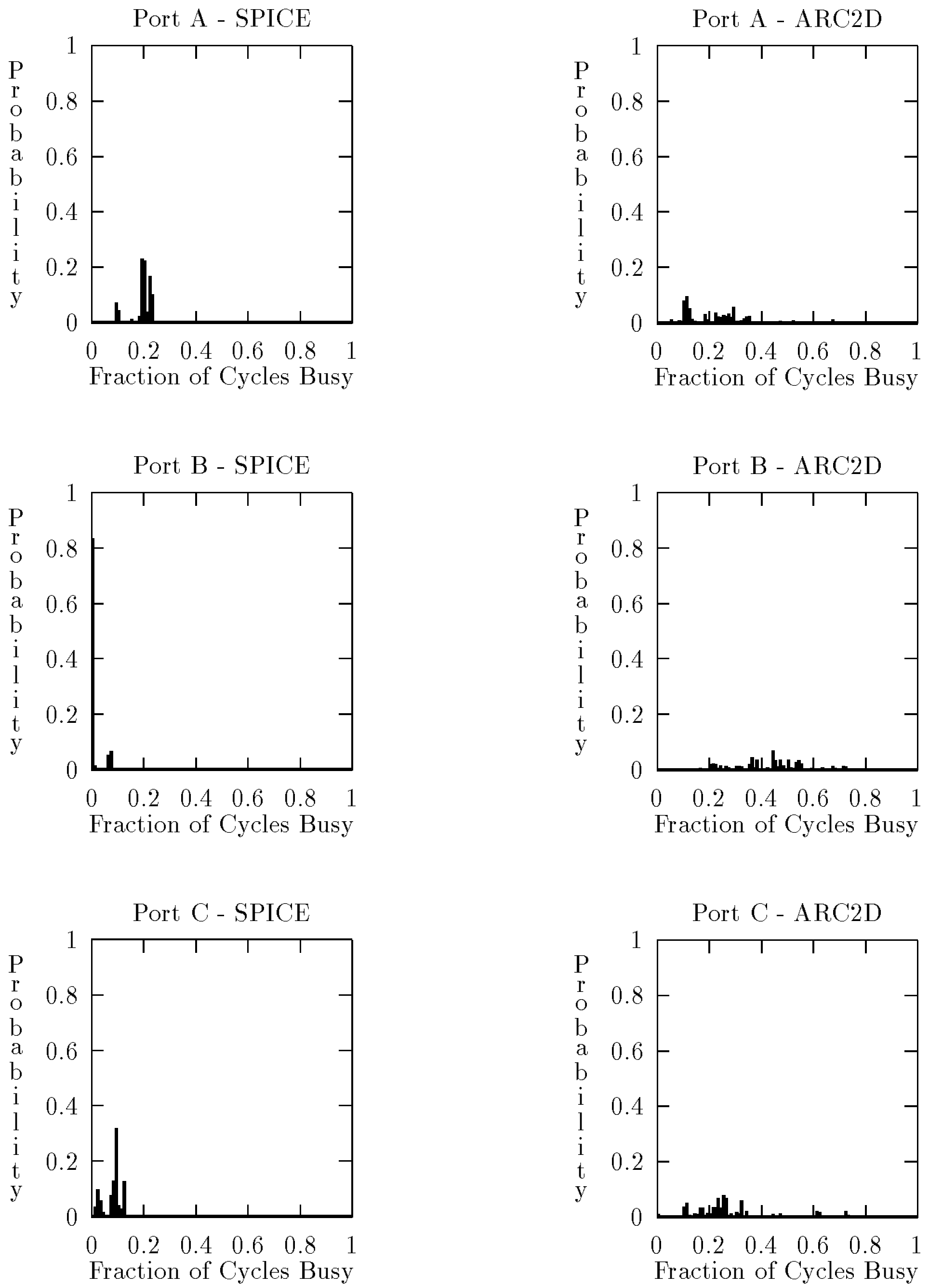


Figure 5: The distribution of fractions of active cycles in the partitions for SPICE and ARC2D. The partition size is 10,000 cycles.