Performance Analysis of Four SIMD Machines

Rod Fatoohi*

Report RNR-92-034, November 1992

Abstract

This paper presents the results of an experiment to study the performance of four SIMD machines. The objectives of this study are to analyze the cost of regular communication on several SIMD machines and study its impact on the performance of two kernels. The machines are: a 32k processor CM2, a 16k processor MPP, a 16k processor MasPar MP-1, and a 4k processor DAP 610C. Regular communication is exemplified, in this study, by the shift operation where all elements of an array are shifted some number of positions along an array dimension. The cost of shift operations on the four machines is measured and analyzed for several two-dimensional arrays. Also, the ratio of compute speed to shift speed is computed for each machine. The low performance rates for basic operations on the CM2 are further investigated using several loops. In addition, performance rates and the communication costs for two kernels on the four machines are measured and analyzed. The study shows that shift cost varies significantly from one machine to another, and depends on several factors including network topology, communication bandwidth, and the compilation partitioning scheme. Results also show that the communication overhead is quite significant on some of these machines even for nearest neighbor communication. Finally, results from this study are useful for obtaining a rough estimate of the communication overhead for many algorithms on these machines.

*Computer Scientist. NAS Applied Research Branch, NASA Ames, MS T045-1, Moffett Field, CA 94035. The author is an employee of Computer Sciences Co. This work was funded through NASA Contract NAS 2-12961. A revised version of this report appeared in the Proceedings of the 7th ACM Int. Conf. on Supercomputing, (Tokyo, July 1993), ACM Press, pp. 117 – 126.

1 Introduction

The performance of machines with many small processors has varied significantly between one application and another. Some of them achieve a reasonable percentage of their peak rates on a variety of applications while others do that only for very specific well written codes. The results of NAS Parallel Benchmark [2] show these trends even for kernels. It is well known that the performance of these machines is dependent on several factors including the communication network, the computational algorithm, and the software environment, including the programming language used.

The performance of Single Instruction Multiple Data (SIMD) machines has been investigated by several researchers. In particular, the performance of the CM2 has attracted many researchers. Early work by Levit [9] using the fieldwise model gives a good understanding of the communication cost on the CM2. Pozo [11] also used the fieldwise model to develop performance models for communication and computation on the CM2. Recently, Chang [4] developed a timing model for both the communication time and the arithmetic time using the slicewise model. On the other SIMD machines, the work by Fatoohi and Grosch [7] is one of the few to analyze the performance of the MPP. As far as a comparison between several SIMD machines, there has been no real attempt to address this issue except for some performance comparison between two machines at the application level.

This study is an attempt to understand the impact of regular communication on the performance of four SIMD machines using high level programming languages. The machines chosen for this study are: a 32k processor CM2 at NASA Ames Research Center, a 16k processor MPP at NASA Goddard Flight Center, a 16k processor MasPar MP-1 at MasPar facilities in Sunnyvale, California, and a 4k processor DAP 610C at AMT facilities in Irvine, California. Regular communication is exemplified in this study by the shift operation where all elements of an array are shifted some number of positions along an array dimension. First, the cost of shift operations for several array sizes is measured. Then, performance models are developed for the different machines. The ratio of compute speed to shift speed is also computed for each machine using different array sizes. In addition, the computation cost for several simple loops on the CM2 is investigated.

Two numerical algorithms are also considered in this study. The algorithms are a four color cell relaxation scheme for the solution of the Cauchy-Riemann equations and an ADI method for the solution of the diffusion equation. These algorithms were first implemented on the MPP; see [7, 8] for details. Recently, they have been implemented on the CM2, MP-1, and 610C for several domain sizes. Some performance comparison and analysis are also given.

2 Architectures

Some of the main features of the four SIMD machines are listed in Table 1; for more details see [1, 3, 10, 12]. The table lists features for the machine configurations tested. These features show that the CM2 is different from the other machines especially in the floating point implementation and network topology.

Table 1: Machine characteristics

Feature	CM2	MPP	MP-1	610C
No. of processors	32k	16k	16k	4k
Clock rate (Mhz)	7	10	12.5	10
Memory per processor (kbit)	1024	1	128	64
Peak 32-bit + rate (Mflops)	7168	430	1721	560
Peak 32-bit $\times$ rate (Mflops)	7168	216	1061	472
Floating point implementation	64-bit	1-bit	4-bit	8-bit
Connectivity	Hypercube	4-NN	8-NN	4-NN
Front end machine	Sun 4/490	Vax 780	Vax 3520	Sun 3/50
Programming language	Fortran	Pascal	Fortran	Fortran

NN: Nearest Neighbor

The CM2 at NAS has 32k 1-bit serial processors and 1024 processing elements (PEs). Each PE is a vertex of a ten dimensional hypercube, and has a 64-bit Weitek floating point unit (FPU) and 4 MBytes of memory. Each FPU can perform two floating point operations, add and multiply, every cycle with a peak rate of 14.34 Gflops for 32k processors.

The MPP performs floating point operations in a single bit fashion while the MP-1 does these operations in a four bit fashion. The DAP 610C has 8-bit coprocessors to do floating point operations. This means that these three machines require many cycles to perform floating point operations, and cannot do more than one of these operations at a given time.

The processors of the MPP and DAP 610C are connected by a two-dimensional grid where each processor can communicate directly with its four nearest neighbors. The processors of the MP-1 are connected by the X-Net where each processor can communicate directly with its eight nearest neighbors. The processors at the edges for these three machines are connected to the processors at the opposite edges. The FPUs of the CM2 are connected in a hypercube topology in which an N-dimensional grid is embedded using Gray encoding addresses.

The Connection Machine Fortran (CMF) compiler has two execution models: fieldwise and slicewise. The fieldwise model uses the single bit processors along with the FPUs to do floating point operations. The slicewise model uses only the FPUs as the basic processing units. In this work, only the slicewise model is considered.

3 Shift cost

3.1 Measurement conditions

The cost of shift operations on the four machines is measured for several array sizes using different machine configurations. In a shift operation all elements of an array are shifted some number of positions along one of the array dimensions. In this work, only square arrays with power-of-two dimensions are considered. Also, only shifts that are a power-of-two are considered. Each shift operation is performed several hundred times during dedicated time and the average time is reported. Also, the reported results represent the average times of two shifts along two different directions since, for some machines, shifting along one dimension is faster than shifting along other dimensions.

There are two types of shift operations on these machines: circular shift, where data shifted out of the processor array are inserted at the opposite edge, and end-off shift, where data shifted out is lost and zeros are shifted in at the other edge. In this study only the fastest way to shift a power-of-two two-dimensional array on each machine is considered. Based on some experiments, the fastest functions to do shift on the corresponding machines are: *cshift* (circular shift) on the CM2, *shift* (end-off shift) on the MPP, *eoshift* (end-off shift) on the MP-1, and *shxp* (end-off shift) on the 610C.

The following compilers are used to generate the timing results: the CMF compiler release 1.1 operating in the slicewise model for the CM2, the MPP Pascal compiler for the MPP, the MasPar Fortran (MPFortran) compiler release 2.0 for the MP-1, and the DAP Fortran-Plus compiler version 2 for the 610C.

3.2 Timing results

Measurement results on the 8k and 32k processors of the CM2, 16k processors of the MPP, 4k and 16k processors of the Maspar MP-1, and 4k processors of the DAP 610C are given in Figures 1 through 6. (The solid lines in these figures represent the measured values while the dotted lines represent the estimated values which will be discussed later.) The results are for several array sizes, ranging from 64×64 up to 512×512, shifted by power-of-two steps, ranging from two up to 64 steps. However, some array sizes are not considered for all machines. For example, only one array size is implemented on the MPP where the array size has to match the machine size.

The results show that, for a given array, the shift cost increases linearly with the number of steps shifted on the MPP, MP-1 and 610C. However, on the CM2 this cost increases nonlinearly up to a certain number of steps, which depends on the array size and machine size, and then remains constant. These results will be explained further in the next section.

Each array size is compared across machines in Figures 7 through 10. These figures show that the cost of shift operations varies significantly from one machine to another. In some cases, this variation is more than an order of magnitude between two machines. The results for one array size (128×128) show that shift is most efficient on the MPP. All results show that shift is overall least efficient on the 8k processor CM2 and overall most efficient on the 16k processor MP-1. The results also show that shift on the 610C is relatively less expensive (compared to other machines) for small arrays than for large arrays.

3.3 Performance models

To better understand the behavior of shift operations on the four machines, performance models will be developed. The shift cost of the CM2 behaves differently from that of the MPP, MP-1 and 610C; as shown in Figures 1 through 6. Therefore, the shift model of the CM2 is different from that of the other machines.

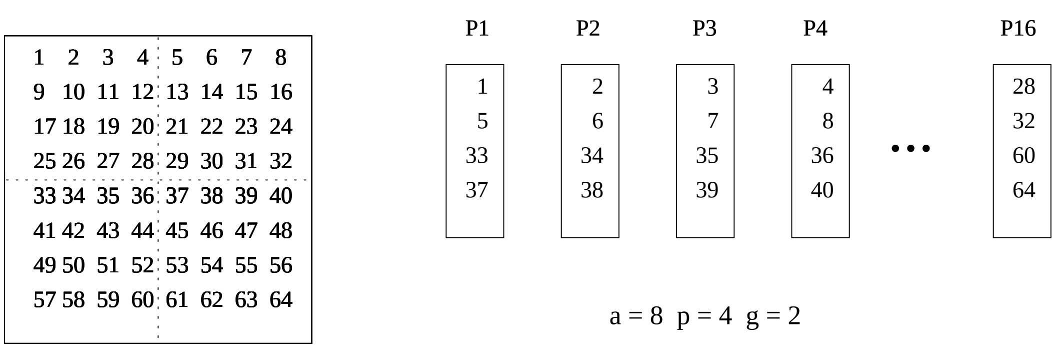
The performance models developed are for shifting a two-dimensional array of size $A : (a \times a)$ elements a distance of d steps on a $P : (p_1 \times p_2)$ processor machine so that each processor has $G : (g_1 \times g_2)$ elements of the array, where $g_1 = a/p_1$ and $g_2 = a/p_2$. For simplicity, it is assumed that $p_1 = p_2 = p$ so that $g_1 = g_2 = g$ even though these models can be easily generalized for a rectangular array of processors.

The first model is a simple linear model for $g = 1$; i.e., the number of array elements is equal to the number of processors. This model is applicable to shift cost on the MPP, MP-1, and 610C. The shift cost on a SIMD machine with nearest neighbor connectivity, T_{nn}, is modeled by the following equation for $g = 1$

$$T_{nn} = d\, t_{offp} + C_{offp}, \qquad g = 1, \tag{1}$$

where t_{offp} is the time to move a 32 bit word to a neighboring processor and C_{offp} is the startup time (or latency) in moving data between processors. Since this is a linear function, the value of C_{offp} and t_{offp} can be easily obtained from two measurements using different distances.

If the number of the array elements is larger than the number of the processors used ($g > 1$), then the array has to be partitioned between the processors. Partitioning can be done either by the user or the compiler. Except for the MPP Pascal compiler on the MPP, all other compilers partition arrays automatically. The partitioning approach that the MP-1 and 610C compilers are currently using is called "cut-and-stack" where the array is partitioned into g^2 $(= A/P)$ subarrays and these subarrays are run on the machine sequentially. This approach requires an internal data movement between subarrays for data at the edges. The following example shows an array of 64 elements which has been mapped onto a machine with 16 processors using the cut-and-stack approach. The number of subarrays is therefore four.



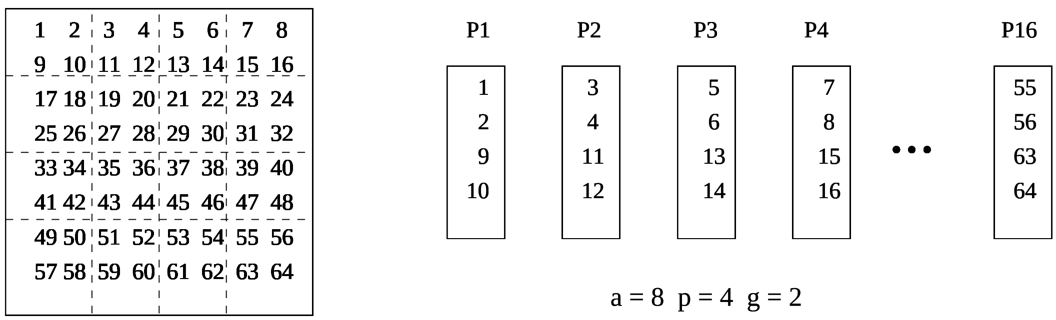
Cut and Stack

The shift cost on a SIMD machine with nearest neighbor connectivity for $g > 1$ is modeled by

$$T_{nn} = g^2(d\, t_{offp} + C_{offp}) + g^2 t_{onp} + C_{onp}, \qquad g > 1, \tag{2}$$

where t_{onp} is the time to move a 32 bit word locally (copying data) and C_{onp} is the startup time (or latency) in moving data locally. The first term in Eq. (2) represents the external data movement cost between processors for g^2 subarrays while the other terms represent the internal data movement cost within processors.

The CM2 shift model is different from the previous model since the communication network as well as the CMF partitioning scheme are different from those of the other machines. The CM2 communication network is based on an N-dimensional grid embedded on a hypercube using Gray encoding addresses. The CMF partitioning approach is called "hierarchical" where arrays are partitioned into blocks of g^2 ($= A/P$) elements and each block is assigned to a PE. The above example is also used to demonstrate the hierarchical approach as follows.



Hierarchical

There are several performance models for shift operations on the CM2. Among these are the timing models by Levit [9] and Pozo [11] for the fieldwise model and the timing model by Chang [4] for the slicewise model. The following model is based on Chang's model where the timing parameters are computed using a different set of measurements. In this model P represents the number of PEs, rather than the number of 1-bit serial processors, and each PE has g^2 elements.

The CM2 shift time depends on the relationship between the number of steps shifted (d) and the number of PE elements in every dimension (g). For $d < g$, shift time has two components: ON-PE and OFF-PE. The ON-PE shift results from moving data within a PE while the OFF-PE shift results from moving data between PEs. Shifting a 2-D array with distance d along the first dimension causes moving d columns, each of size g, to the neighboring PE while $g - d$ columns, also of size g, move internally within the PE. Shifting along the second dimension has a similar behavior. This means that the shift cost for the CM2, T_{cm}, is modeled by

$$T_{cm} = g\,(g - d)\,t_{onp} + C_{onp} + g\,d\,t_{offp} + C_{offp}, \qquad d < g, \tag{3}$$

where t_{offp}, C_{offp}, t_{onp}, and C_{onp} are defined as in Eqs. (1) and (2) except that processors here mean PEs. The first two terms of Eq. (3) represent the ON-PE communication while the next two terms represent the OFF-PE communication.

If the number of steps shifted along one dimension is equal to the number of PE elements along that dimension ($d = g$), then all PE elements of the array (g^2) are

Table 2: Measured cost of communication parameters for a 32-bit word (in μsec).

Parameter	CM2	MPP	MP-1	610C
External communication time (t_{offp})	9.0	3.2	2.7	3.2
External communication latency (C_{offp})	20.0	13.4	41.9	7.2
Internal communication time (t_{onp})	0.7	-	5.6	9.0
Internal communication latency (C_{onp})	35.0	-	59.1	18.0

moved to the neighboring PE; i.e., there is no ON-PE communication. This means that

$$T_{cm} = g^2\, t_{offp} + C_{offp}, \qquad d = g. \tag{4}$$

Finally, if d is a power of two and larger than g, then the CM shift time is twice as much as for $g = d$ since the longest distance between any two PE addresses is always two. This is due to the Gray-code ordering of the OFF-PE bits of grid addresses. This means that

$$T_{cm} = 2\,[g^2\, t_{offp} + C_{offp}], \qquad d > g. \tag{5}$$

The value of the parameters t_{offp}, C_{offp}, t_{onp}, and C_{onp} for different machines can be obtained from the measured results given in Figures 1 through 6. For the MPP, MP-1, and 610C, the parameters t_{offp} and C_{offp} are obtained by applying Eq. (1) to the timing results for $g = 1$. Then the parameters t_{onp} and C_{onp} for the MP-1 and 610C are approximated using Eq. (2) and the timing results for $g > 1$. For the CM2, the parameters t_{offp} and C_{offp} are computed by applying Eq. (4) to the timing results for $d = g$. Then the parameters t_{onp} and C_{onp} are approximated using Eq. (3) and the timing results for $d < g$. The measured values of these parameters for the four machines are given in Table 2. These values show that the ratio of external to internal communication times (t_{offp}/t_{onp}) is higher for the CM2 than for the other machines, where it is less than one for the MP-1 and 610C. The results for the MP-1 show that the latencies (C_{offp} and C_{onp}) are relatively high which mainly due to software overhead.

The shift time for the four machines can be estimated using the models (Eqs. (1) through (5)) and the measured timing parameters (Table 2). The dotted lines in Figures 1 through 6 represent the estimated shift times for these machines. The differences between the measured and estimated results are within 10% for all cases and within 5% for most cases.

The performance models show that the hierarchical approach, which is used by the CMF compiler in partitioning large arrays, is more efficient than the cut-and-stack approach, which is used by the MP-1 and 610C compilers. This can be explained by considering the case of quadrupling the array size; i.e., g is doubled. For $d < g$ on the CM2, Eq. (3) shows that the OFF-PE communication time will be at most doubled, since the parameter C_{offp} will not change. For the ON-PE communication time, the first term will increase by a factor of more than four while the parameter C_{onp} will remain unchanged. Overall, the CM2 shift cost will increase by a factor of less than four if the array size is quadrupled. For the MP-1 and 610C, if the array size is quadrupled, then Eq. (2) shows that shift time will increase by a factor of more than four because of the parameter t_{onp}.

The models also show that with hierarchical there is no internal communication for long communication ($d \geq g$) while with cut-and-stack internal communication can not be ignored. In summary, the hierarchical performance improves for large arrays while the cut-and-stack performance degrades for large arrays. The two different approaches cause the 610C to outperform the 32k processor CM2 for small arrays (Figure 7) while the CM2 outperformed the 610C for large arrays (Figure 10).

Several experiments have shown that for $d < g$ on the CM2 shifting along the second dimension is more expensive than along the first dimension. These experiments also indicate that the difference between the two shifts is a function of g. Pozo [11] attributed this difference to the cyclic characteristic of the Gray-code ordering of the OFF-PE bits of grid addresses. Since the estimated results do not take into account this overhead and the measured results are based on the average of two shifts along two directions, the differences between the measured and estimated results can partially be attributed to the difference between the two shifts.

Another interesting observation is that for a small g, T_{cm} can be smaller using Eq. (4) than using Eq. (3) even though d is larger in Eq. (4) than in Eq. (3). This case is clearly shown in Figure 7 for the 64×64 array on the 32k processor CM2 where there is a dip at $d = 2$. This happens mainly because the latency times, C_{onp} and C_{offp}, dominate the shift time for a small g; see Eq. (3).

The communication bandwidth between adjacent processors (or PEs) can be obtained from the measured value of t_{offp} (Table 2). The measured communication bandwidth is therefore 0.44 MBytes/sec for the CM2, 1.25 MBytes/sec for the MPP, 1.48 MBytes/sec for the MasPar MP-1, and 1.25 MBytes/sec for the DAP 610C. These measured values are for a unidirectional data transfer. These results show that the CM2 lags the other machines in communication bandwidth.

4 Compute to shift speed

For scientific programming shift operations are not useful without arithmetic operations. One interesting measure is the ratio of the floating point speed to the communication speed. This ratio is called α [9]. It is computed by taking the ratio of the shift time to the floating point time. The floating point time is measured for each array size by taking the average time of adding and multiplying two arrays of 32-bit elements. Except for the CM2, the time to do addition is not the same as for multiplication since these machines do not have 32-bit floating point units.

Table 3 lists the Mflops rates obtained from the floating point times. The results for the CM2 and 610C show performance improves as the array size increases from 64×64 to 512×512. A factor of 2.9 improvement is observed for the 32k processor CM2 while a factor of 1.7 improvement is observed for the 610C. (The CM2 results will be discussed further in the next section.) On the other hand, the MP-1 achieves its best rate when the array size matches the machine size. For larger arrays, the MP-1 shows a drop and then an improvement in performance.

Figures 11 through 16 show the measured value of α for different array sizes and

Table 3: Average performance (in Mflops) of 32-bit floating point operations (add and multiply).

array	CM2		MPP	MP-1		610C
size	(8k)	(32k)	(16k)	(4k)	(16k)	(4k)
64×64	169	303	-	117	-	196
128×128	220	677	185	109	464	295
256×256	222	880	-	-	354	324
512×512	220	889	-	-	413	332

machine configurations. The shift speed is obtained from Figures 1 through 6 while the compute speed is obtained from Table 3. The results for the 8k processor CM2 show that, in general, α is higher for small arrays than for large arrays. On the other hand, the results for the 610C show that α increases as the array size increases. This is because for oversized arrays on the 610C the shift speed decreases (Eq. (2) and Figure 6) while the compute speed improves (Table 3).

The results, given in Figures 11 through 16, also show that the ratio of computation to communication is much higher on the CM2 than on the MPP while the MP-1 and 610C fall in between. In some cases, communication on the CM2 is as much as 15 times slower than computation. Even short distance communication can cost as much as five times the computation on the CM2. While on the other machines, this type of communication costs less than twice the computation.

5 Computation cost on the CM2

The average performance rates of basic operations, given in Table 3, differ from the peak rates, given in Table 1. Basic operations achieve 57%, 33%, and 64% of the peak rates of the MPP, MP-1, and 610C, respectively, while they achieve only 6% of that of the CM2. The low percentage rate on the CM2 needs further explanation. This is accomplished by first giving more details about the FPUs of the CM2 and then by developing a simple performance model for floating point operations using the slicewise model.

Each FPU of the CM2 is a Weitek wtl 3164 microprocessor. This microprocessor has add and multiply pipes, which can be chained to get multiply/add operations. Also, it has several vector registers, each of size four, so that all vector operations are based on this size. The data path between the PE memory and vector registers is 32-bit wide. The floating point pipes as well as the load pipe can produce a result every cycle after a latency of two cycles while the store pipe can produce a result every other cycle after a latency of four cycles. All these features have a great impact on the performance of the machine.

There are several studies about the performance of the floating point operations on the CM2. Levit [9] measured the fieldwise model performance using the nine loops given in [5]. Pozo [11] also measured the fieldwise model performance using basic and triad operations. Chang [4] measured the slicewise model performance using basic, triad and some finite difference floating point operations. In addition, he developed a simple performance model for these operations where the parameters of his model are computed from the measured results using a least square fit. The following model is similar to Chang's model except that the parameters are computed using the Processing Element Assembler Code (PEAC) language [13]; i.e., it is based on the number of cycles required to perform each operation.

The following performance model is for doing floating point operations within a loop using arrays of size $A : (a \times a)$ elements each on a $P : (p \times p)$ PE machine so that each PE has $G : (g \times g)$ elements of each array. The computation time of the loop on the CM2 is modeled by

$$T_{comp} = (g^2/4)t_{comp} + C_{comp}, \tag{6}$$

where t_{comp} is the time to compute the loop on each PE using vectors of length four and C_{comp} is the startup time for the CM to receive addresses and data from the frond end for the loop. The value of t_{comp} includes time to load and store operands to memory. The value of t_{comp} and C_{comp} can be obtained using the PEAC cycle count [13].

The processing rate, R_p, of a loop is computed by

$$R_p = N_f \ A/T_{comp}, \tag{7}$$

where N_f is the number of flops in the loop.

The performance model is used to estimate the cost of adding (or multiplying) two arrays of 32-bit floats on the CM2. (This operation is used to compute α in the previous section.) The compute time for this operation is obtained from [13]:

$$t_{comp} = [t_{lod} + t_{add} + t_{str} + t_{jnz}]/R_c, \tag{8}$$

where t_{lod} is the vector load time using 32-bit floats (= 6 cycles), t_{add} is the vector add time using 32-bit floats (= 6 cycles), t_{str} is the vector store time using 32-bit floats (= 12 cycles), t_{jnz} is the decrement and jump time (= 5 cycles), and R_c is the clock rate (= 7 MHz for the machine at NAS). The value of t_{comp} for this operation is therefore equal to 4.14 (29/7) μseconds. Similarly, the value of C_{comp} for this operation is equal to 3.29 (23/7) μseconds.

The processing rates, both measured and estimated, for adding two arrays of different sizes are given in Table 4. The results show the performance improves with the increase in the domain size. The results also show that the model predicts the execution time within 10% for the largest array while it is off by up to 40% for the smallest domain. This shows that the model, which is based on the PEAC cycle count, may have underestimated the startup cost of this loop.

The performance rates, as given in Table 4, show that the 1024 PE (32k processor) machine achieves less than one Gflops for this basic operation, using both measured and estimated values. As shown in the model, this operation requires 29 cycles (ignoring the startup time) to produce four results with an efficiency of less than 14%. However, if the loop has more operations, performance can improve.

Table 5 shows the performance, both measured and estimated, of the 512×512 array on the 8k processor CM2 for the nine loops given in [5]. The values of t_{comp} and

Table 4: Performance (in Mflops) of a 32-bit floating point operation (add or multiply) on the CM2.

array size	256 PEs		1024 PEs	
	Measured	Estimated	Measured	Estimated
64×64	169	206	303	551
128×128	220	235	677	825
256×256	222	244	880	942
512×512	220	246	889	977

Table 5: Performance on 256 PEs (8k procs.) CM2 using 32-bit arithmetic, where a, b, c, d, e are arrays of size 512×512 and $\alpha, \beta, \gamma, \delta$ are scalars.

Loop No.	Operation	Flop Count	t_{comp} (μsec)	C_{comp} (μsec)	Measured (Mflops)	Estimated (Mflops)
1	$a = \alpha b$	1	3.29	4.43	284	310
2	$a = bc$	1	4.14	3.29	220	246
3	$a = \alpha(b + c)$	2	5.00	4.86	368	408
4	$a = b(c + d)$	2	5.00	3.71	359	408
5	$a = \alpha b + \beta c$	3	4.43	6.43	617	690
6	$a = \alpha b + cd$	3	5.29	5.29	544	579
7	$a = bc + de$	3	6.14	4.14	434	499
8	$a = \alpha b + \beta c + \gamma d$	5	5.57	8.43	812	914
9	$a = \alpha b + \beta c + \gamma d + \delta e$	7	6.71	10.43	932	1061

C_{comp} are computed using the PEAC cycle count. The estimated performance rates are computed using Eqs. (6) and (7). For all these loops, the differences between the estimated and measured values are within 13%. These results show that performance varies significantly between one loop and another. For example, the compute time (t_{comp}) for loop 9 is only 62% more than that of loop 2 even though loop 9 does seven flops while loop 2 does only one flop. This translates to a performance difference of a factor of more than four between these two loops. The results also show that a reasonable percentage of the peak rate of the CM2 can only be achieved for very special loops.

In summary, these results show that the performance of the machine is quite sensitive to the complexity of the loop. Loops with many arithmetic operations can achieve reasonable performance on the CM2 since data reuse as well as chaining multiply and add operations are more likely to happen in these loops.

6 Performance of two algorithms

6.1 Algorithms

The previous sections concentrated only on shift and basic operations. In order to have a better understanding of the impact of regular communication on the performance of the four machines, two numerical algorithms are also considered. These algorithms are a four color cell relaxation scheme for the solution of the Cauchy-Riemann equations and an ADI method for the solution of the diffusion equation. These algorithms are chosen since they depict two different communication patterns and can be parts of more complex problems. The two solution procedures, which are summarized in this section, are not unique but they are used on all four machines. Some machines, such as the CM2, have a library routine to solve tridiagonal equations. This routine has been tested and found to be faster than this implementation for large domains. Therefore, the following solution procedures represent *only* one way to solve these problems on all machines *rather* than the best way on each machine.

The Cauchy-Riemann equations are solved using a four color reordering scheme [8]. This is done by dividing the domain into four sets of disjoint cells so that the cells of each set are processed simultaneously. The relaxation process is performed for each set *color* separately, that is, four times to complete a sweep. The maximum residual is then computed and tested against the convergence tolerance. The whole process is repeated until the iteration converges. This algorithm maps well onto a SIMD architecture since it can be implemented almost entirely with matrix operations and a few scalar operations. In addition, nearly all data transfer operations are between nearest neighbors.

The diffusion equation is solved using an ADI method which requires solving two sets of tridiagonal equations [7]. These equations are solved by the cyclic elimination algorithm. The ADI method is reasonably efficient on a SIMD machine since it requires mostly matrix operations with a few scalar operations. However, it requires long distance communication since at the end of each level of the elimination process the distance shifted is doubled. Also, cyclic elimination is not the most efficient algorithm to solve tridiagonal equations; Gaussian elimination is more efficient [7].

The numbers of floating point operations and shift operations are computed for each kernel. Here division is considered as a single operation. The ratio of compute operations to shift operations is about two for the first kernel, which is called RELAX, where all shift operations are to nearest neighbors. This ratio is about 2.2 for the second kernel, which is called ADI, provided that communication is ranging between one step and $a/2$ steps in power-of-two steps where a is the array dimension.

6.2 Results

The two algorithms have been implemented on the four machines using different grid sizes and machine configurations. Early results on the CM2 (using CMF 1.0 Fieldwise and Slicewise), the MPP, the MP-1 (using MPFortran 1.0), and the DAP 510 and 510C were reported in [6]. Also, performance comparisons with vector machines (Cray

Table 6: Results on the CM2 with SUN 4/490 and CMF 1.1 Slicewise.

Domain size	8k		32k	
	time (msec)	rate (Mflops)	time (msec)	rate (Mflops)
RELAX				
64×64	11.8	49	8.2	70
128×128	26.5	89	12.5	187
256×256	77.4	122	28.1	335
512×512	266.0	142	81.4	465
ADI				
64×64	23.4	28	12.6	53
128×128	77.6	40	28.0	112
256×256	298.5	48	92.4	155
512×512	1207.8	53	364.6	176

Table 7: Results on the MPP (16k processors)

Domain size	RELAX		ADI	
	time ($m$sec)	rate (Mflops)	time ($m$sec)	rate (Mflops)
128×128	13.6	172	23.7	132

Y-MP, Cray-2, and NEC SX-2) are given in [6]. In this section, an updated set of results as well as the DAP 610C results are given.

Tables 6 through 9 list the results on the CM2 (8k and 32k processors using CMF 1.1), the MPP (16k processors using MPP Pascal), the MasPar MP-1 (4k and 16k processors using MPFortran 2.0), and DAP 610C (4k processors using DAP Fortran-Plus). All timing results are for a single iteration of the relaxation scheme and one time step for the ADI method. These results represent the best case times from many runs. The processing rates are computed by counting the number of additions, multiplications, and divisions in the source codes. All the results are in single precision (32 bits).

The CM2 results (Table 6) show a significant performance improvement (up to a factor of 6.6) as the domain size increases. Performance improves because both communication and computation on the CM2 improve as the array size increases; see Figures 1 and 2 and Table 3. The 610C results (Table 9) also show performance improvement as the domain size increases but by a smaller factor (less than 30%). This is because on the 610C the computation speed improves while the communication speed degrades for larger arrays. The MP-1 results (Table 8) show best rates when the domain size matches the machine size, which also agree with the timing results given in Figures 4 and 5 and Table 3. (A speedup of more than four was observed for the

Table 8: Results on the MasPar MP-1 using MPFortran 2.0 compiler.

Domain size	4k		16k	
	time (msec)	rate (Mflops)	time (msec)	rate (Mflops)
RELAX				
64×64	7.0	82	-	-
128×128	37.8	62	7.1	331
256×256	144.0	65	37.5	257
512×512	557.5	67	144.7	262
ADI				
64×64	8.6	77	-	-
128×128	72.2	43	11.6	268
256×256	282.7	51	90.3	158
512×512	1176.6	55	349.6	184

Table 9: Results on the DAP 610C (4k processors)

Domain size	RELAX		ADI	
	time (msec)	rate (Mflops)	time (msec)	rate (Mflops)
64×64	3.7	156	6.6	102
128×128	14.7	159	29.5	106
256×256	54.5	173	118.3	121
512×512	213.3	178	495.0	130

Table 10: Relative performance of the two algorithms to 8k processor CM2

Domain size	CM2 (8k)	CM2 (32k)	MPP (16k)	MasPar MP-1 (4k)	MasPar MP-1 (16k)	DAP 610C (4k)
RELAX						
64×64	1.00	1.44	-	1.69	-	3.19
128×128	1.00	2.12	1.95	0.70	3.73	1.80
256×256	1.00	2.75	-	0.54	2.06	1.42
512×512	1.00	3.27	-	0.48	1.84	1.25
ADI						
64×64	1.00	1.86	-	2.72	-	3.55
128×128	1.00	2.77	3.27	1.07	6.69	2.63
256×256	1.00	3.23	-	1.06	3.31	2.52
512×512	1.00	3.31	-	1.03	3.45	2.44

two algorithms using the 128×128 domain when the number of processors increased from 4k to 16k.) Increasing the domain size on the MP-1 causes a performance degradation, but then the performance improves for larger domains.

The performance of the two algorithms on the four machines is compared to the performance of the 8k processors of the CM2. Table 10 shows the relative performance rates of the machines. These rates show that the MPP, the 16k processor MP-1 and the 610C outperform the 8k processor CM2 in all cases considered. In some cases, these machines even outperform the 32k processor CM2. The 4k processor MP-1 outperforms the 8k processor CM2 for ADI and for the small domain of RELAX. Also, these results show that these machines are competitive in solving these problems.

6.3 Communication cost

The communication cost of the two algorithms on the four machines is estimated by using the measured shift costs, as given in Section 3. These shift costs are multiplied by the number of shift operations in each code, using the source codes, to estimate the communication cost. Table 11 shows the estimated communication costs relative to the total measured times for the two codes. The results show that, except for the oversized problems on the MP-1, communication costs more in ADI than in RELAX since ADI requires long communication while RELAX requires only nearest neighbor communication. The results also show that communication is least expensive on the MPP. While on the other machines, the communication cost is quite significant even for nearest neighbor communication.

The CM2 results, as given in Table 11, show that the communication cost depends on the domain size where it is less significant for large domains. The same conclusion can be drawn from α results and the ratio of compute to shift operations for the two kernels. The value of α for nearest neighbor communication on the 8k processor CM2, as given in Figure 11, is ranging between four for the 64×64 array and less than one for the 512×512 array, and slightly above two for the 128×128 array. Since

Table 11: Percentage (%) of communication cost to total cost for the two codes.

Domain size	CM2 (8k)	CM2 (32k)	MPP (16k)	MP-1 (4k)	MP-1 (16k)	610C (4k)
RELAX						
64×64	64	67	-	47	-	20
128×128	49	61	8	54	46	39
256×256	38	47	-	-	51	37
512×512	30	36	-	-	46	37
ADI						
64×64	80	80	-	61	-	47
128×128	78	83	25	40	67	57
256×256	76	81	-	-	47	55
512×512	74	77	-	-	44	53

the ratio of compute to shift operations for RELAX is two, then the communication cost should be more than 50% for the 64×64 array, less than 50% for the 512×512 array, and about 50% for the 128×128 array. The results of Table 11 confirm that. The same analysis can be obtained for the 32k processor results as well as for the ADI results.

The 610C results (Table 11) show that the communication cost is less for domain sizes which match the machine size than for oversized problems. This observation agrees with the results obtained for α (Figure 16). For nearest neighbor communication, α is about 0.5 for the 64×64 array and about 1.5 for larger arrays. This means that for routines that have twice as many arithmetic operations as shift operations, such as RELAX, the communication cost will be less than 50% and reasonably small for domain sizes that match the machine size. As given in Table 11, the communication cost is 20% for the 64×64 problem and less than 40% for the other domains.

The MPP results (Table 11) show a low communication overhead, which agrees with the α results (Figure 13). The MP-1 results (Table 11) show that the communication cost is about 50% for RELAX and ranging between 40% and 67% for ADI. The value of α for nearest neighbor communication on the MP-1 (Figures 14 and 15) is between one and two. Some of the MP-1 results are not quite justified and seem to depend on factors like compiler partitioning and mapping techniques.

The communication costs on these machines have also been estimated using other techniques. The TMC Prism programming environment, version 1.0, is also used to estimate the communication cost on the 8k processor CM2. The differences between the two techniques (using the measured shift costs and Prism) are within 1% for most cases and within 4% for the two smallest domains for RELAX. This means that the CM2 results are reasonably accurate. The MPP results are verified by successfully estimating the total cost which includes the computation cost; see [7] and [8].

Another technique to estimate the communication cost is by removing all shift operations from the codes and rerunning them. (Obviously, the numerical results are not correct.) The differences in execution time between the no communication runs and the *actual* runs is considered as the communication cost. This technique is used on both the MP-1 and 610C. The results using this technique agree with the results given in Table 11 in some cases, while they are off by more than 10% in other cases. The problem with this technique is that compilers might do different optimizations using two different codes (one with communication and the other without communication).

7 Concluding remarks

This study analyzed the cost of regular communication on four SIMD machines and how this type of communication affects the performance of two kernels. This was accomplished by developing performance models for shift operations on these machines. These models showed that the performance of shift operations depends on many factors including the connectivity, the communication bandwidth between processors, and the partitioning scheme that the compiler uses to map oversized arrays onto these machines. The performance models predicted the shift cost within about 10% of the measured values. The study showed that shift cost varies from one machine to another by up to an order of magnitude depending on the array size and machine configuration.

The results of performing basic operations on the CM2 showed that the machine achieves only a small percentage of its peak performance with these operations. The low percentage rate was investigated further by developing a simple performance model and by studying the performance of nine different loops. The results showed that the CM2 can achieve a reasonable percentage of its peak rate only for very specific loops where there is data reuse and many arithmetic operations.

The results of two kernels showed that the performance of SIMD machines has not improved very much since the introduction of the MPP (more than a decade ago). The CM2, MP-1 and 610C have much more memory than the MPP, and have compilers that can partition oversized arrays, unlike the MPP. However, these machines have comparable performance rates to the MPP. This might change with the introduction of the CM5 and MP-2.

One interesting point about the implementation of the two kernels is the portability between the four machines. The two codes needed only small modifications to run on these machines, with no changes to the numerical algorithm. In some cases, such as on the CM2 and MP-1, the same code basically ran on both machines. This shows that on one class of parallel architectures, codes can easily be moved from one machine to another.

One interesting question is "which machine has the *best* compute to shift speed ratio (called α) for a certain type of application?" This question is hard to answer since applications have different computation and communication requirements which depend on the numerical algorithm used and the mapping of the algorithm onto the architecture. Some applications have more data locality than others, which translates to a better compute to communicate ratio.

If the objective is to have no more than 50% communication overhead, then for the two kernels considered here (which have roughly twice as many floating point operations as shift operations) the MP-1 and 610C seem to have the desirable α while the CM2 has a high α and the MPP has a low α. Obviously, a low α does not hurt except it might be considered "overkill". For other applications where there are more arithmetic operations for each shift operation, a high α may not be harmful.

The study showed that the ratio of the compute speed to shift speed can be estimated for several SIMD machines. This is the architecture side of the problem of mapping algorithms onto architectures. From the algorithm side, the ratio of compute to shift operations can also be computed. From the two ratios (architecture and algorithm), a rough estimate of the communication cost can be obtained.

This work can be extended to estimate the communication cost for more complex problems once the ratio of compute to shift operations is computed. Using the computed value of α from this study, the communication cost can be estimated even before implementing these applications on parallel machines. Such a study requires a full understanding of the communication and computation requirements of different numerical algorithms used to solve complex problems.

This study is only the first step in understanding the impact of the communication overhead on the performance of parallel machines. Many features of these architectures were not exploited. For example, the ability of some of these machines, such as the CM2, to communicate with more than one neighbor simultaneously was not exploited even though this type of communication requires a reordering of the computation and calling special routines on the CM2. The impact of the X-Net on the performance of the MP-1 is not fully understood even though it seems to be minimal. The performance of irregular communication, which is sometimes referred to as the router communication, was not considered. Neither was the impact of front end machine nor the performance of arrays with non-power-of-two dimensions. Also, this study concentrated only on single precision (32 bit) arithmetic even though a similar procedure can be used for double precision. All these issues can have a great impact on the performance of any machine.

8 Acknowledgment

I would like to thank Kenneth Jacobsen of MasPar and Rex Thanakij of AMT for their assistance in running codes on the MasPar MP-1 and DAP 610C. Also, I am thankful to Mark Kremenetsky of TMC for discussions about the CM2.

References

[1] Active Memory Technology, *DAP Series Technical Overview*, 1988.

[2] D. H. Bailey, E. Barszcz, L. Dagum, and H. D. Simon, *NAS Parallel Benchmark Results*, in Proceedings of Supercomputing '92, (Minneapolis, November 1992), IEEE Computer Society Press, pp. 386 – 393.

[3] K. E. Batcher, *Design of a Massively Parallel Processor*, IEEE Trans. Computers, Vol. C-29, 1980, pp. 836 – 840.

[4] Y. Chang, *Comparison of Finite Difference and the Pseudo-Spectral Approximations for Hyperbolic Equations and Implementation Analysis on Parallel Computer CM-2*, CAM Report 92-02 (Ph.D. thesis), Dept. of Mathematics, Univ. of California at Los Angeles, January 1992.

[5] R. A. Fatoohi, *Vector Performance Analysis of Three Supercomputers: Cray-2, Cray Y-MP, and ETA10-Q*, in Proceedings of Supercomputing '89, (Reno, November 1989), ACM Press, pp. 779 – 788.

[6] R. A. Fatoohi, *Performance Comparison of Several SIMD Machines*, in Proceedings of the fifth SIAM conference on parallel processing for scientific computing, (Houston, March 1991), SIAM, pp. 419 – 424.

[7] R. A. Fatoohi and C. E. Grosch, *Implementation of an ADI Method on Parallel Computers*, Journal of Scientific Computing, Vol. 2, No. 2, 1987, pp. 175 – 193.

[8] R. A. Fatoohi and C. E. Grosch, *Implementation of a Four Color Cell Relaxation Scheme on the MPP, Flex/32, and Cray/2*, in Proceedings of the 1987 International Conference on Parallel Processing, (St. Charles, August 1987), Penn State Press, pp. 424 – 426.

[9] C. Levit, *Grid Communication on the Connection Machine: Analysis, Performance, and Improvements*, Int. J. High Speed Computing, Vol. 1, No. 2, 1989, pp. 367 – 381.

[10] MasPar Computer Co., *MP-1 Family Data-Parallel Computers*, PL006.0190, 1990.

[11] R. Pozo, *Performance Modeling of Parallel Architectures for Scientific Computing*, Ph.D. thesis, Dept. of Computer Science, Univ. of Colorado at Boulder, 1991.

[12] H. D. Simon, (ed.), *Scientific Applications of the Connection Machine*, World Scientific, Singapore, 1989.

[13] T. Weinberg, L. Tennies, A. Vasilevsky, and Y. Tosa, *PEAC Language*, Thinking Machines Corp., Internal document, 1991.

Figure 1. Shift cost on a 8k CM2 (M:Measured,E:Estimated)

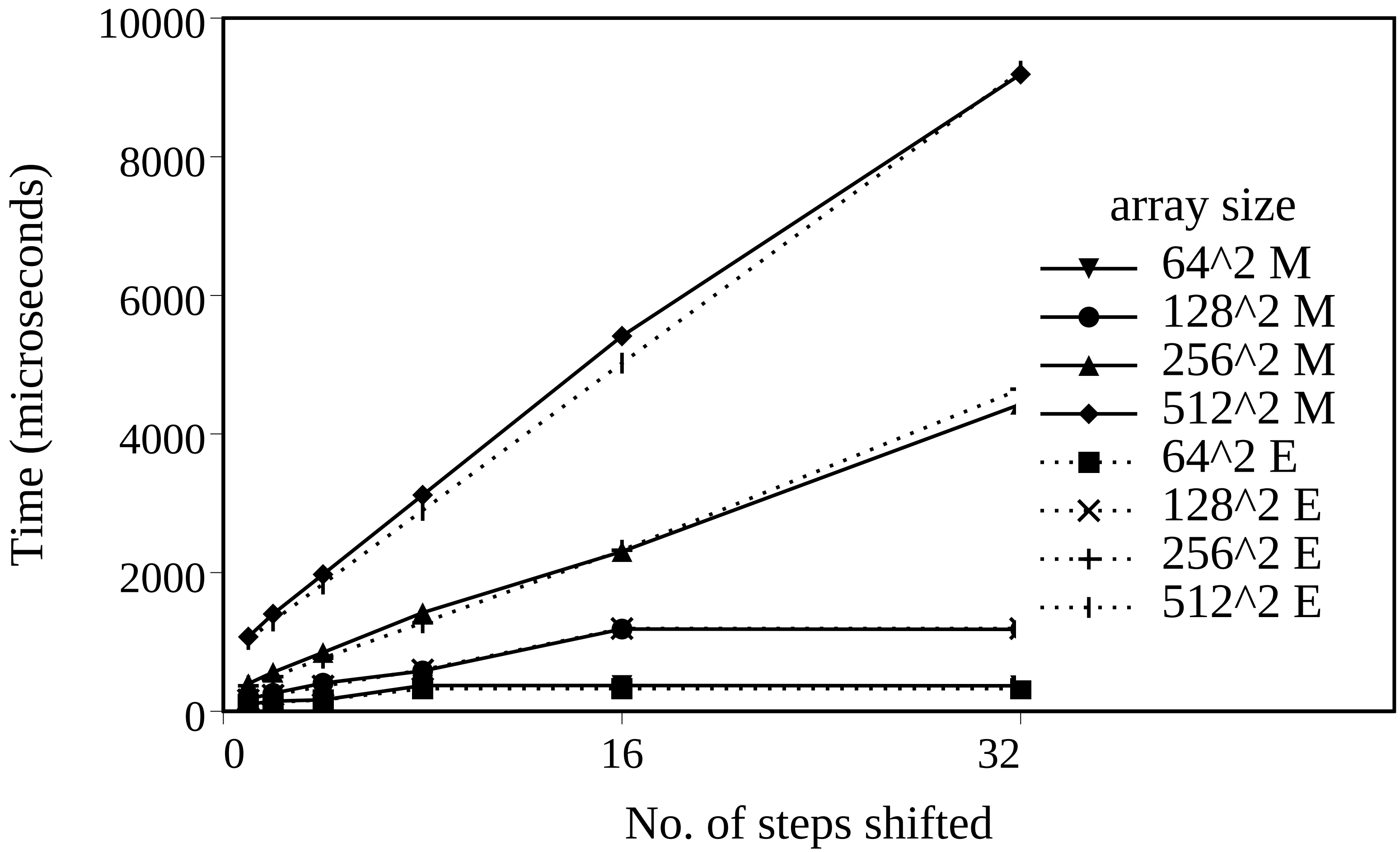


Figure 2. Shift cost on a 32k CM2 (M:Measured,E:Estimated)

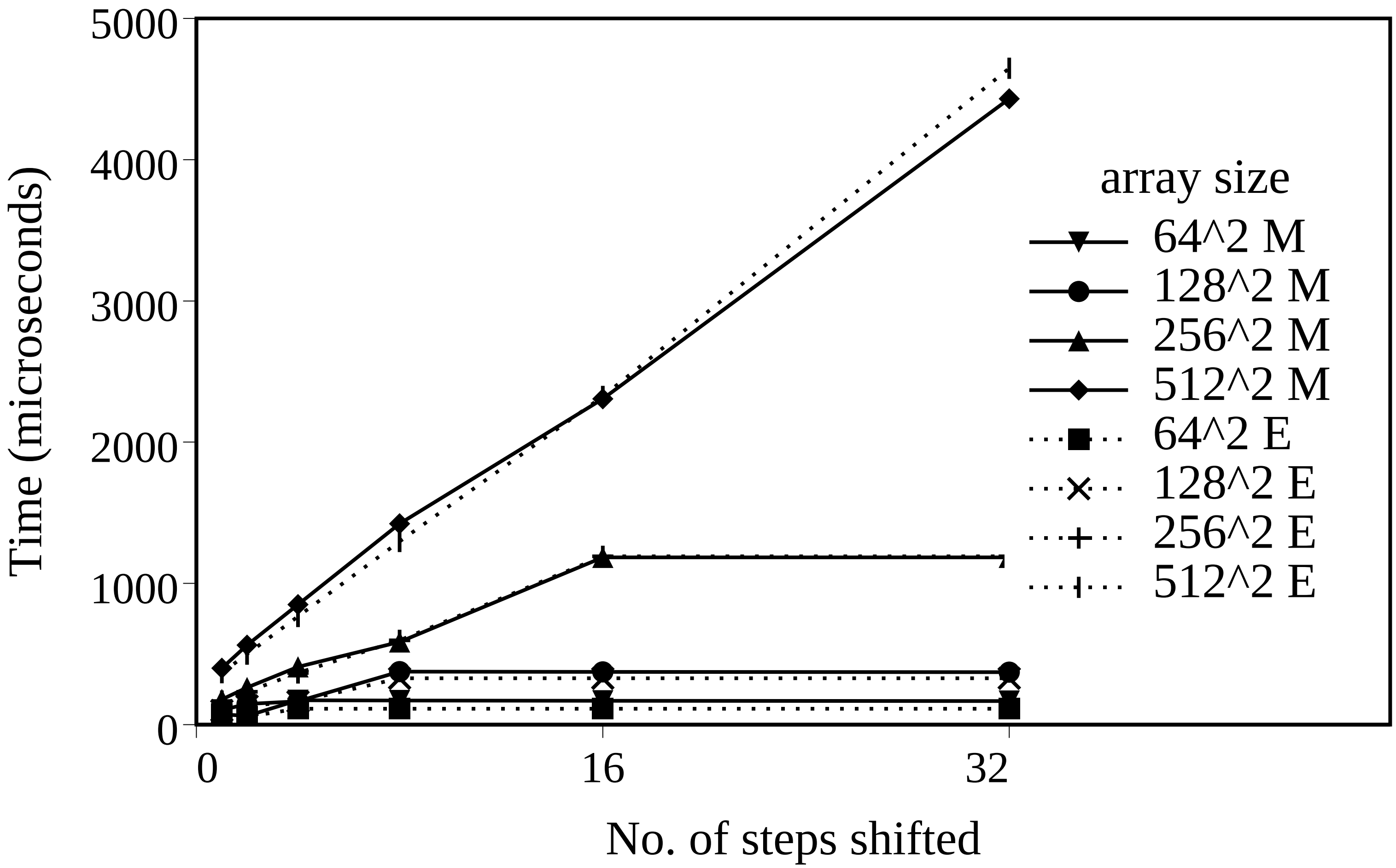


Figure 3. Shift cost on the MPP (16k).

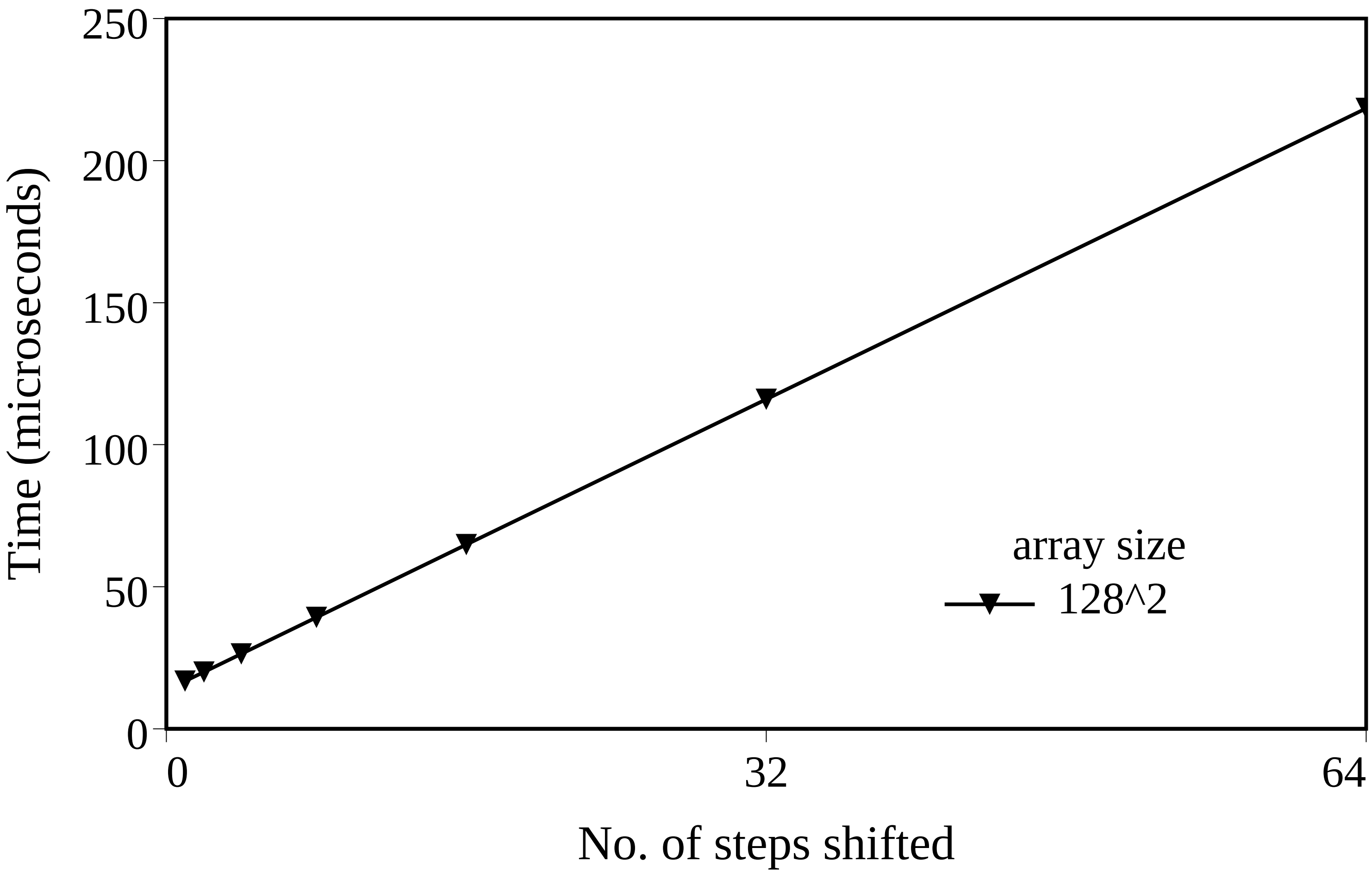


Figure 4. Shift cost on a 4k MP-1 (M:Measured,E:Estimated)

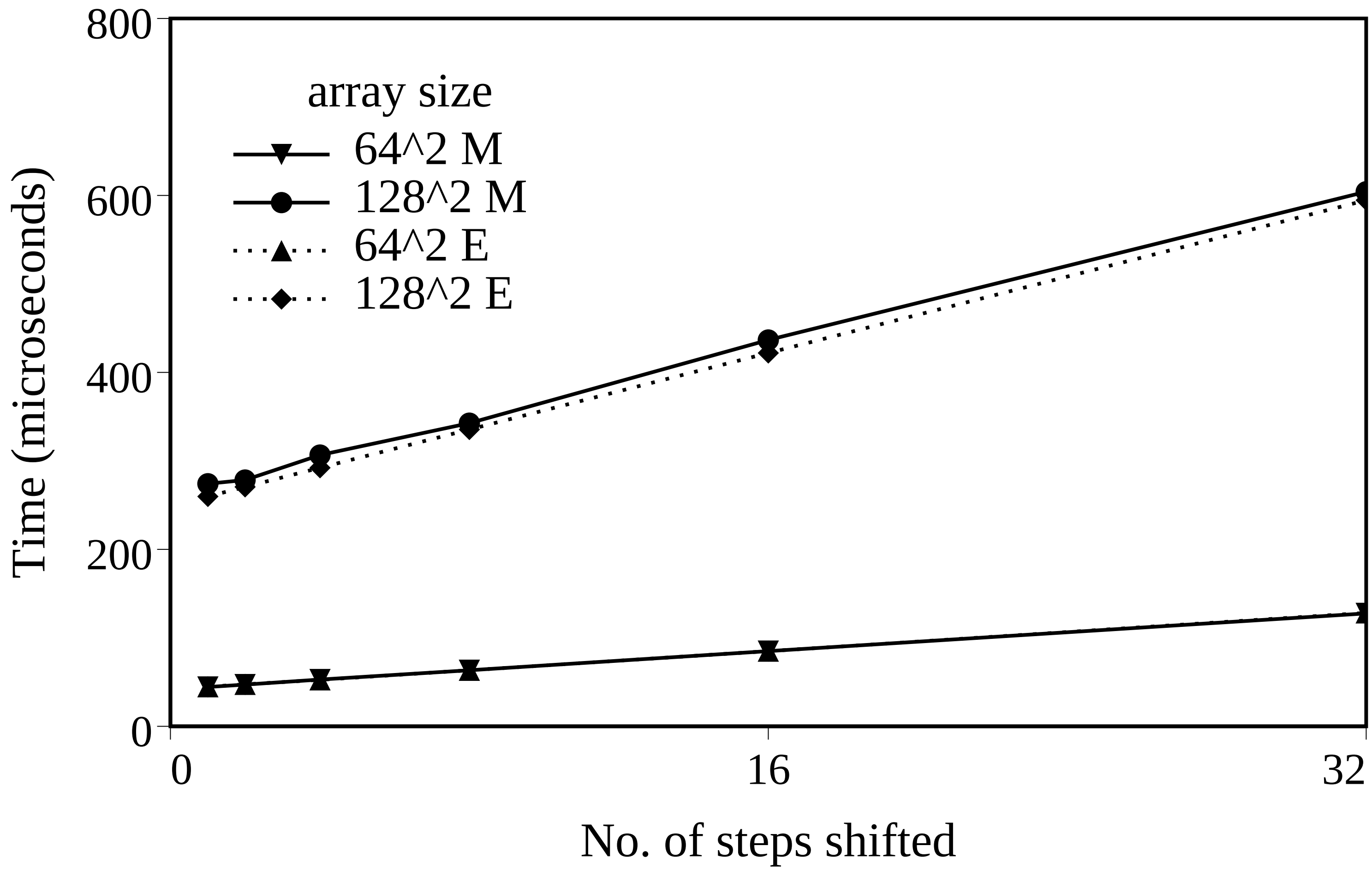
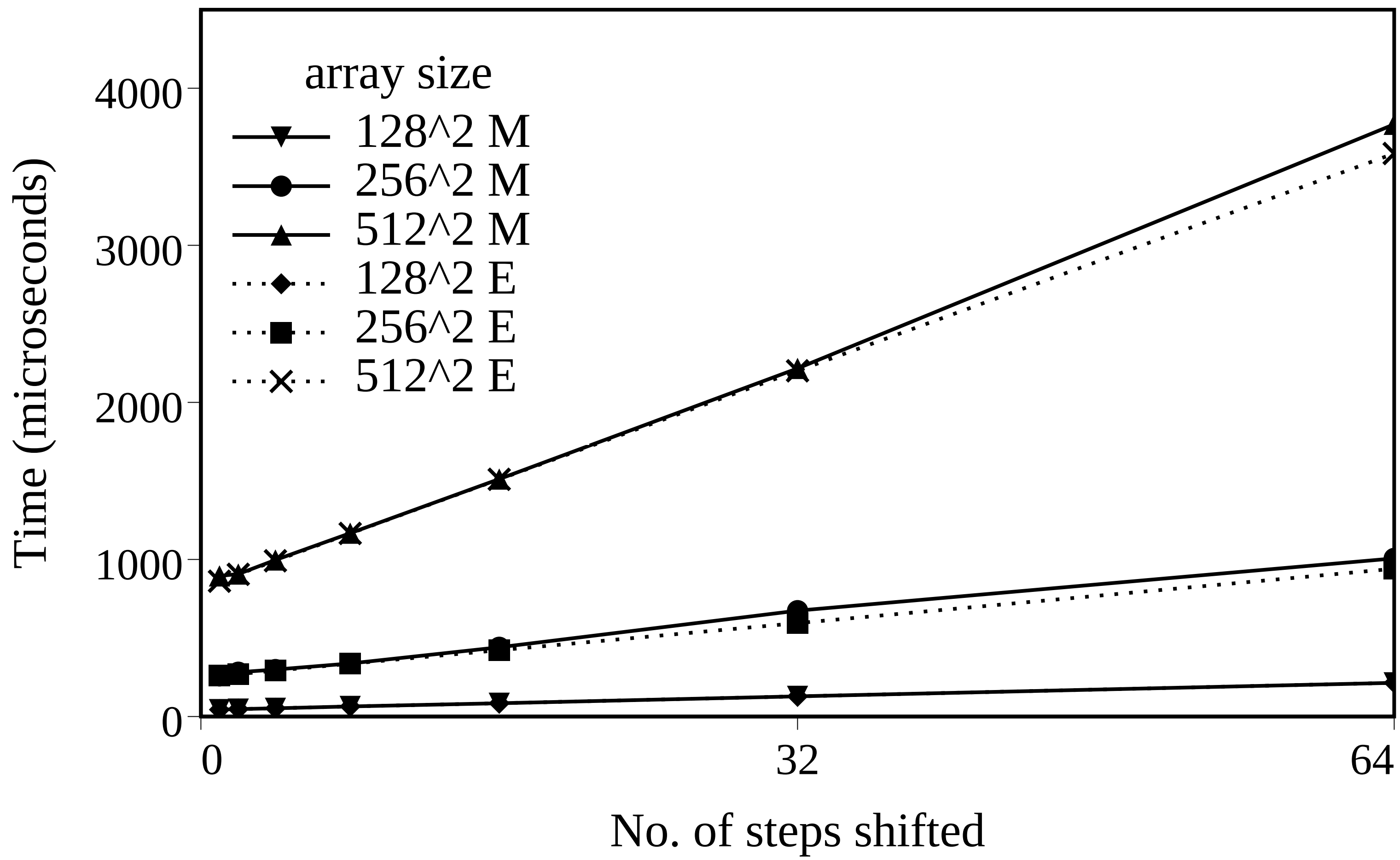


Figure 5. Shift cost a 16k MP-1 (M:Measured,E:Estimated).



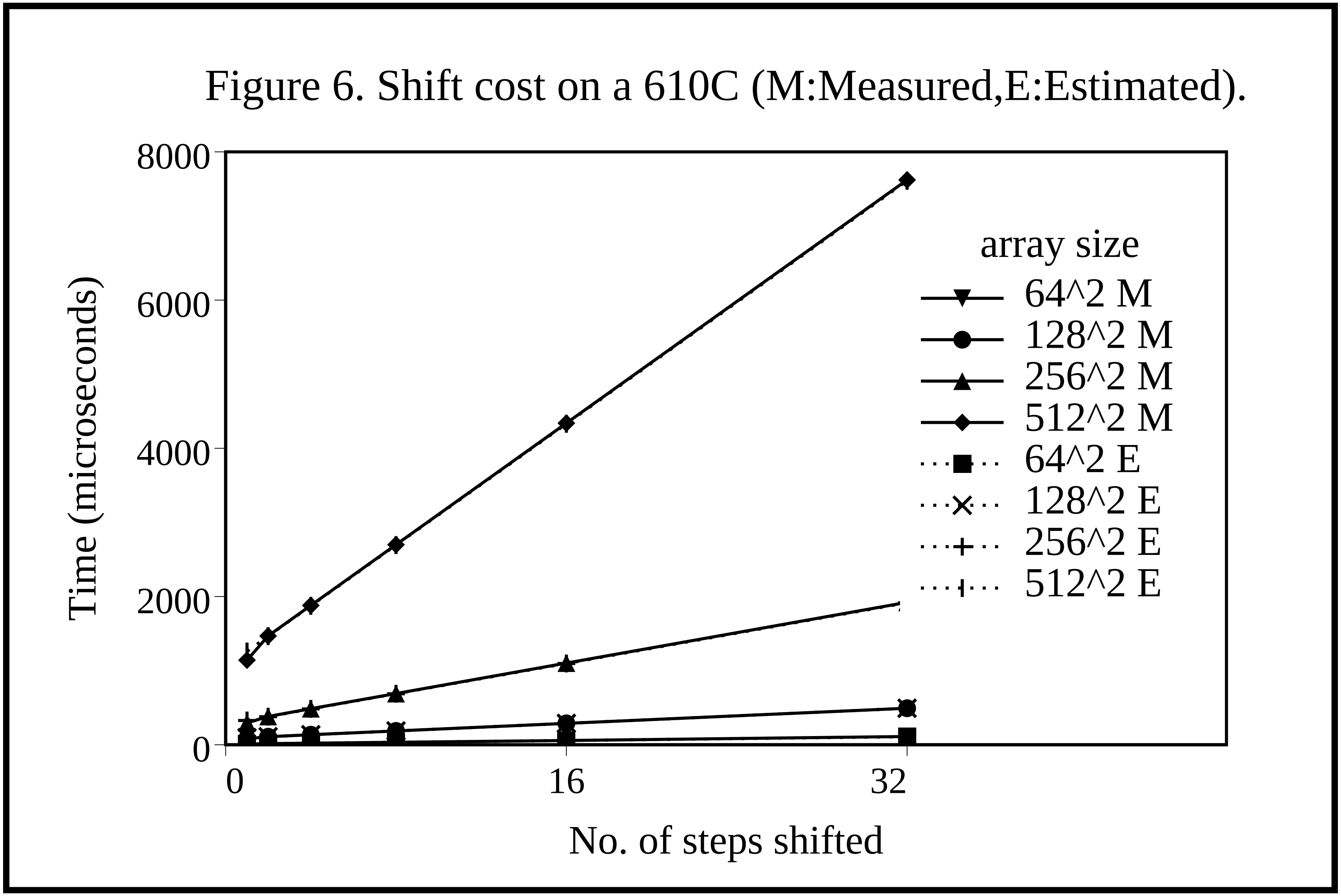


Figure 6. Shift cost on a 610C (M:Measured,E:Estimated).
Time (microseconds)
8000
6000
4000
2000
0
0
16
32
No. of steps shifted
array size
64^2 M
128^2 M
256^2 M
512^2 M
64^2 E
128^2 E
256^2 E
512^2 E

Figure 7. Measured shift cost for the 64^2 array

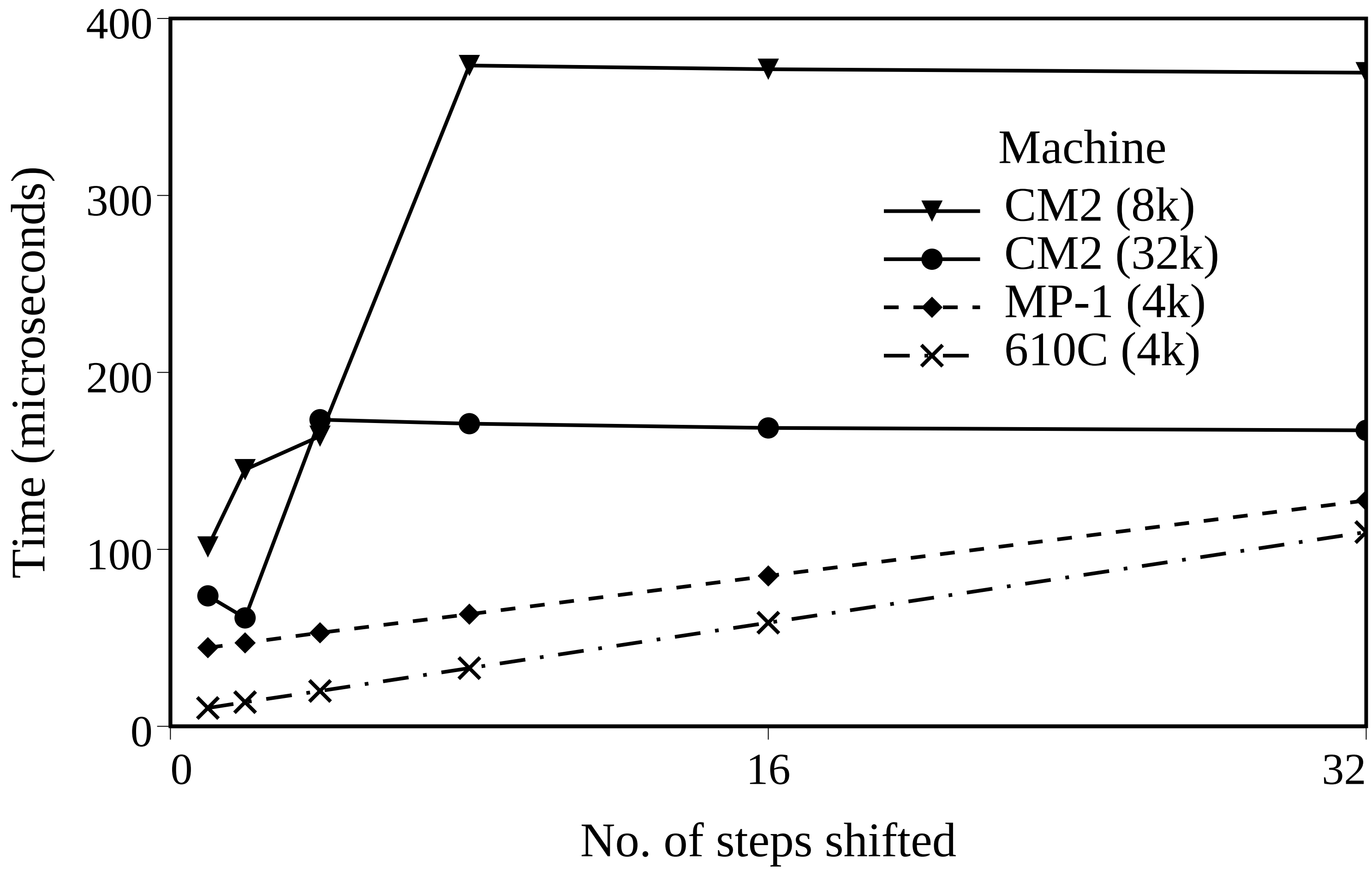


Figure 8. Measured shift cost for the 128^2 array

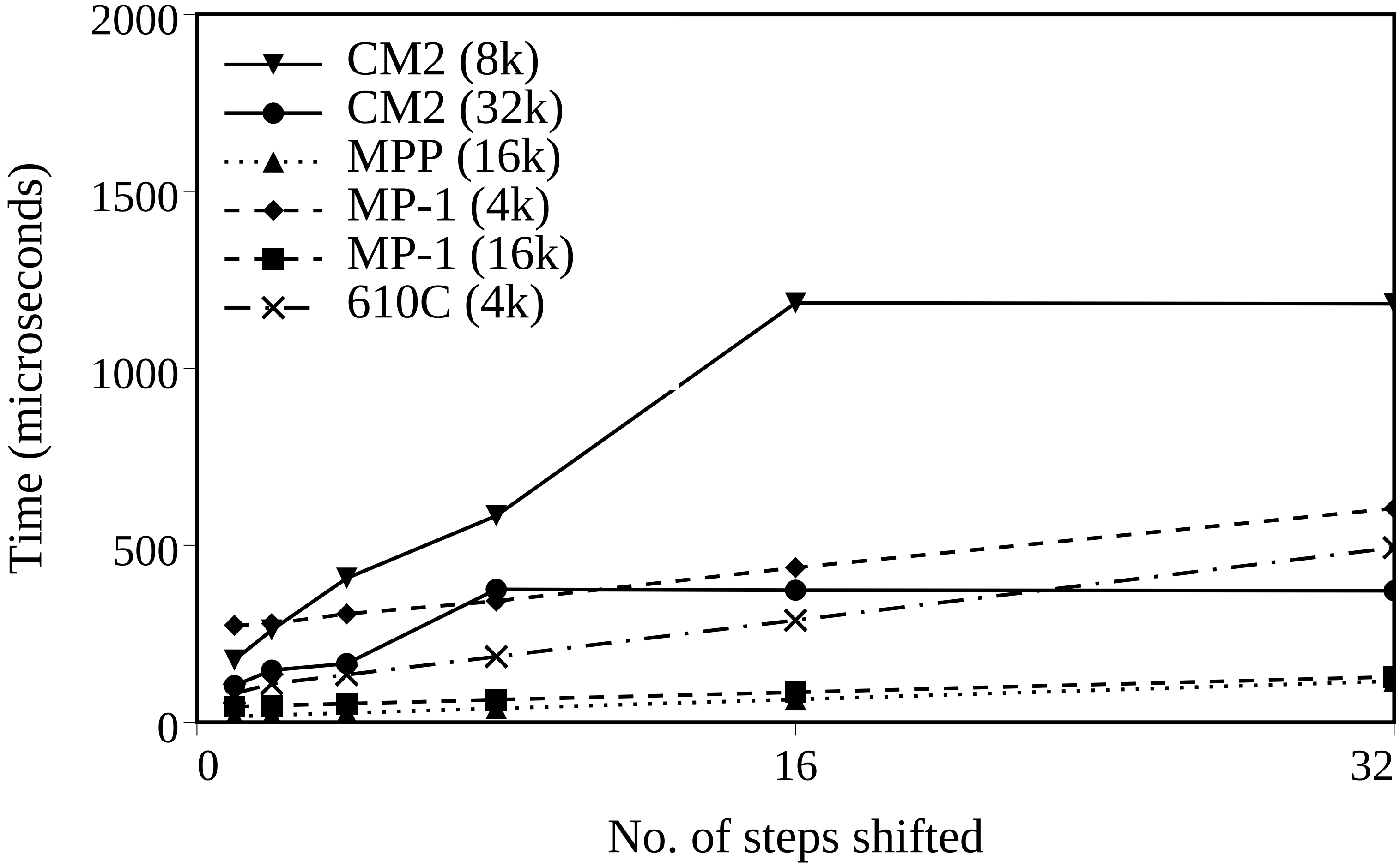


Figure 9. Measured shift cost for the 256^2 array

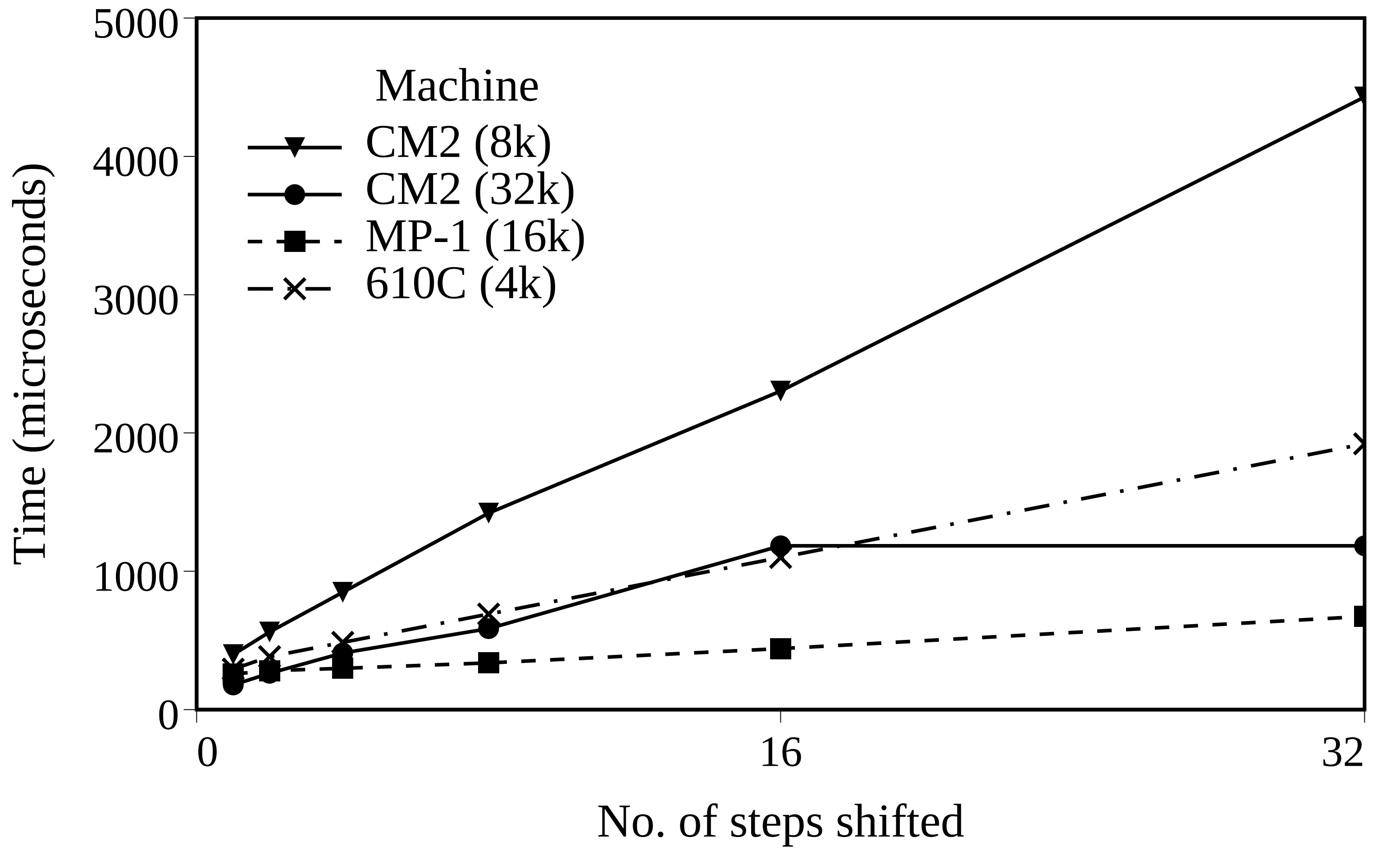


Figure 10. Measured shift cost for the 512^2 array.

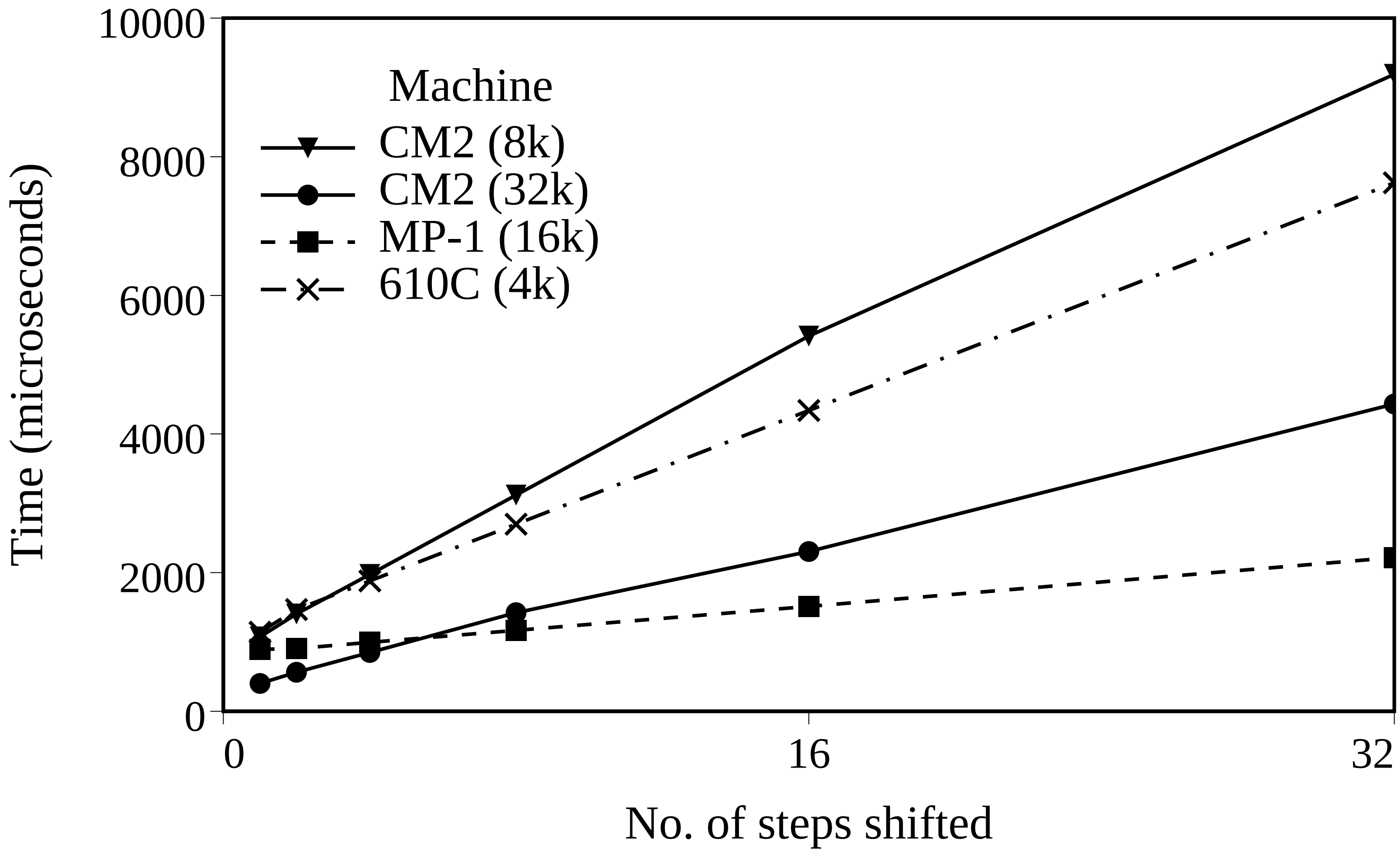


Figure 11. Alpha (Compute/shift speed) on a 8k CM2.

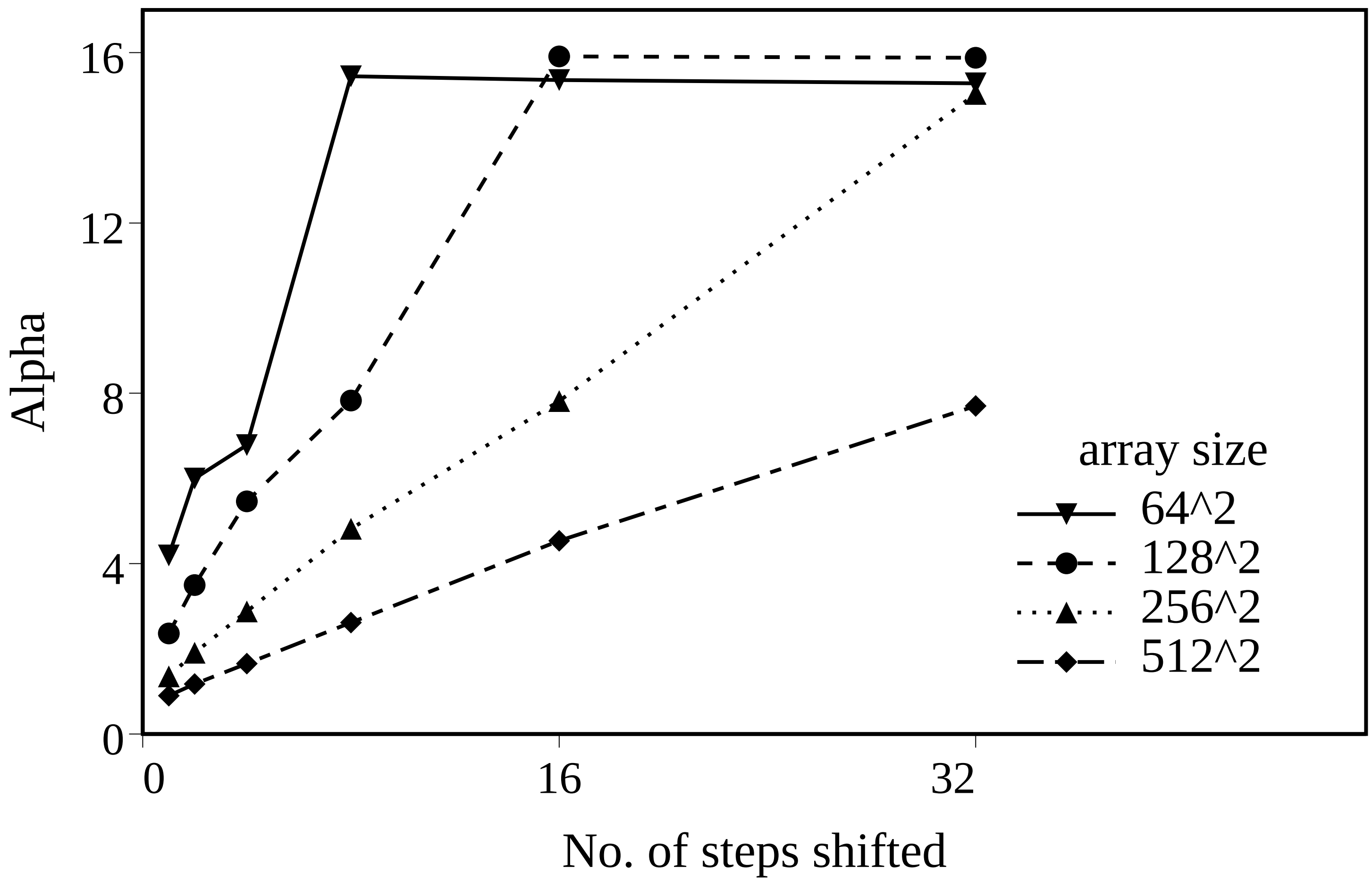


Figure 12. Alpha (Compute/shift speed) on a 32k CM2.

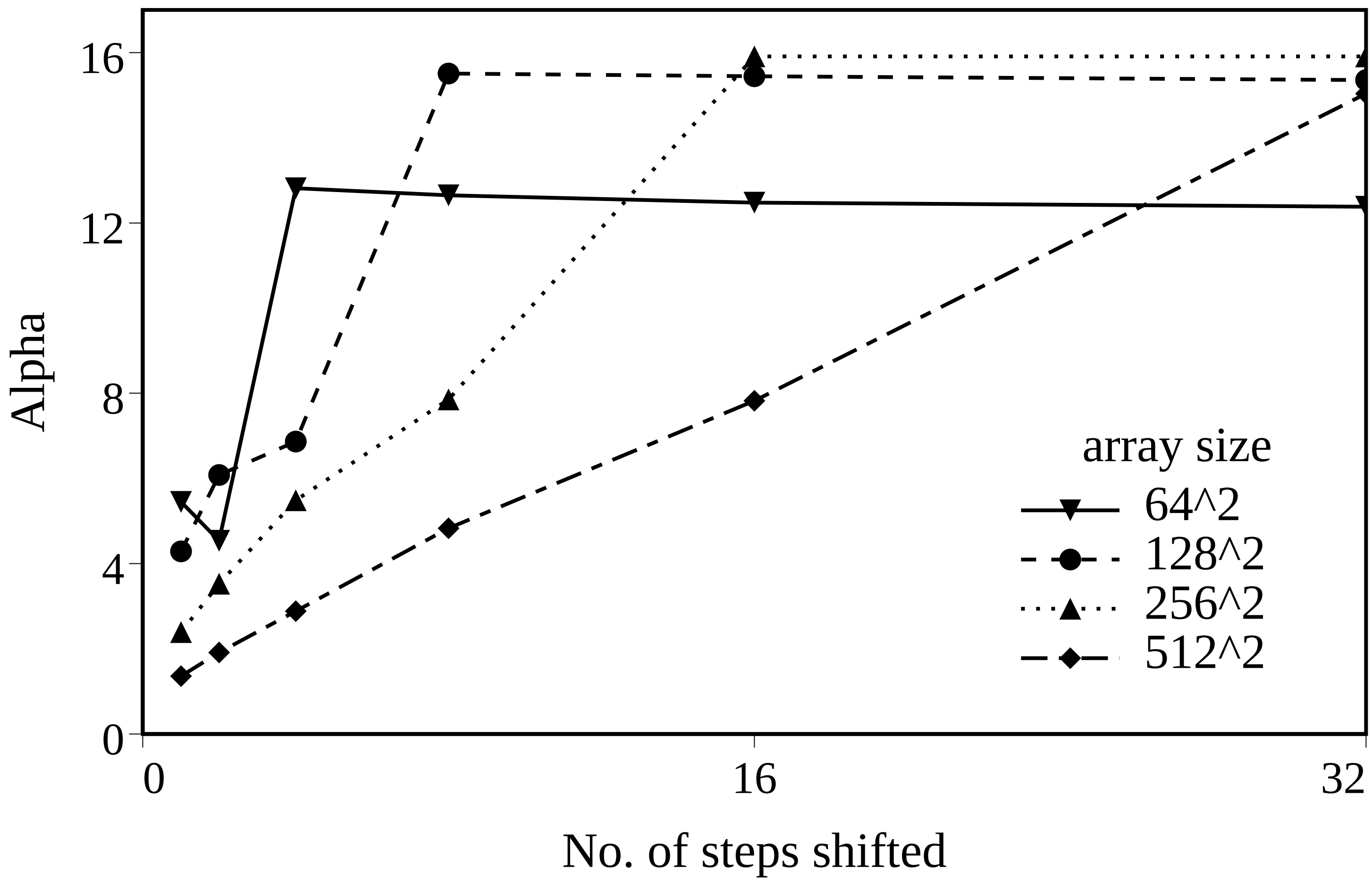


Figure 13. Alpha (Compute/shift speed) on the MPP (16k).

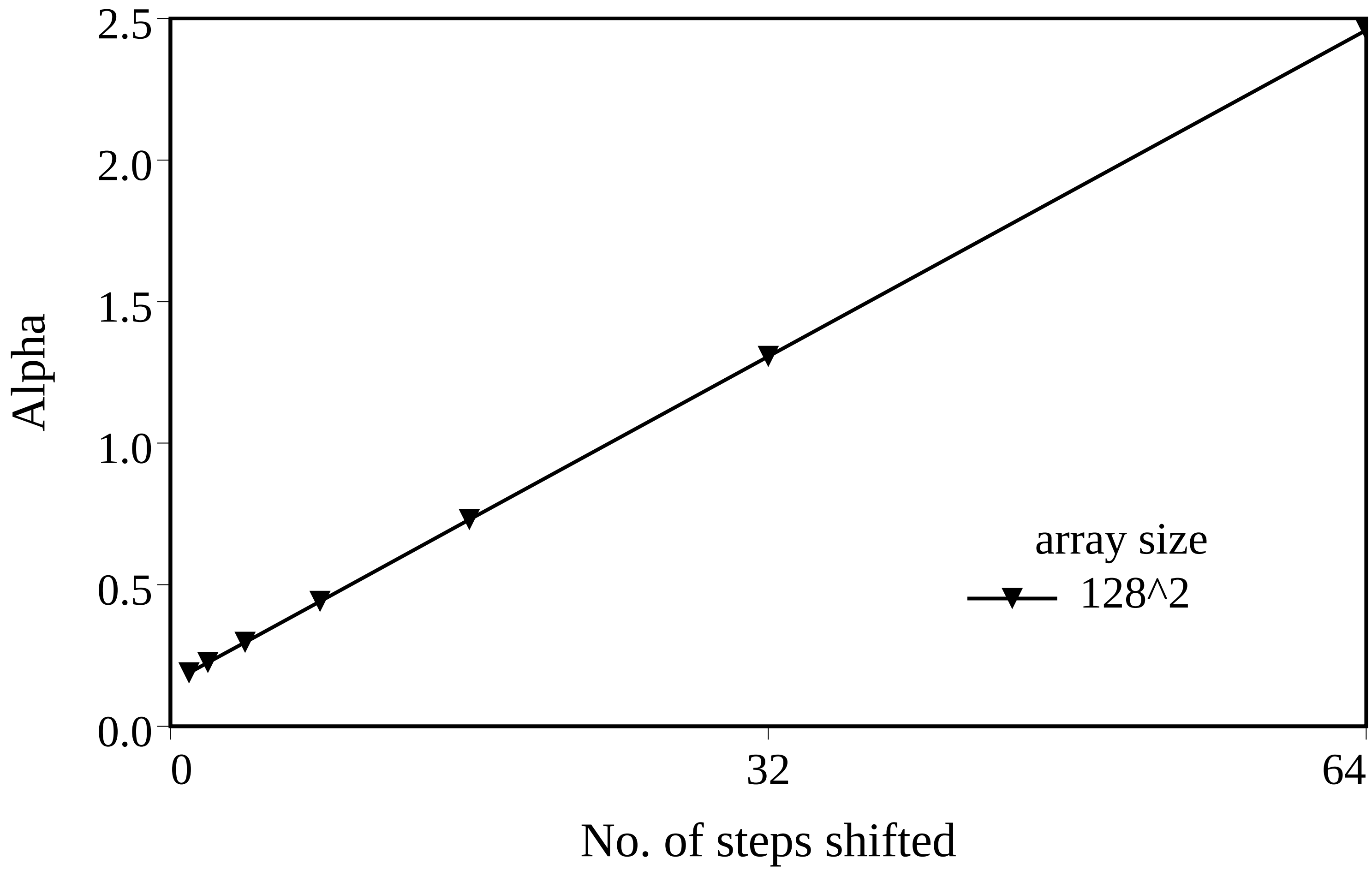


Figure 14. Alpha (Compute/shift speed) on a 4k MP-1.

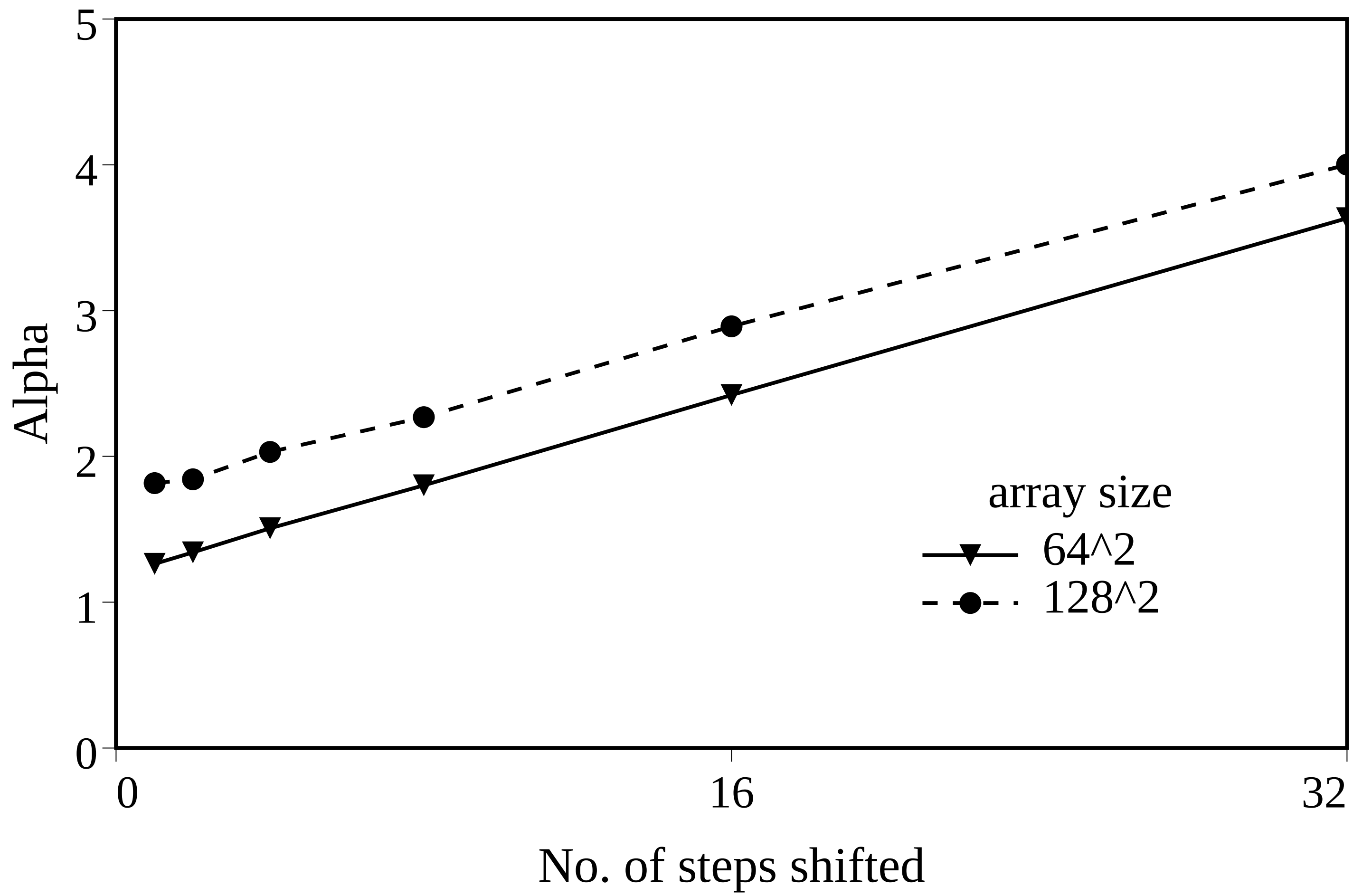


Figure 15. Alpha (Compute/shift speed) on a 16k MP-1.

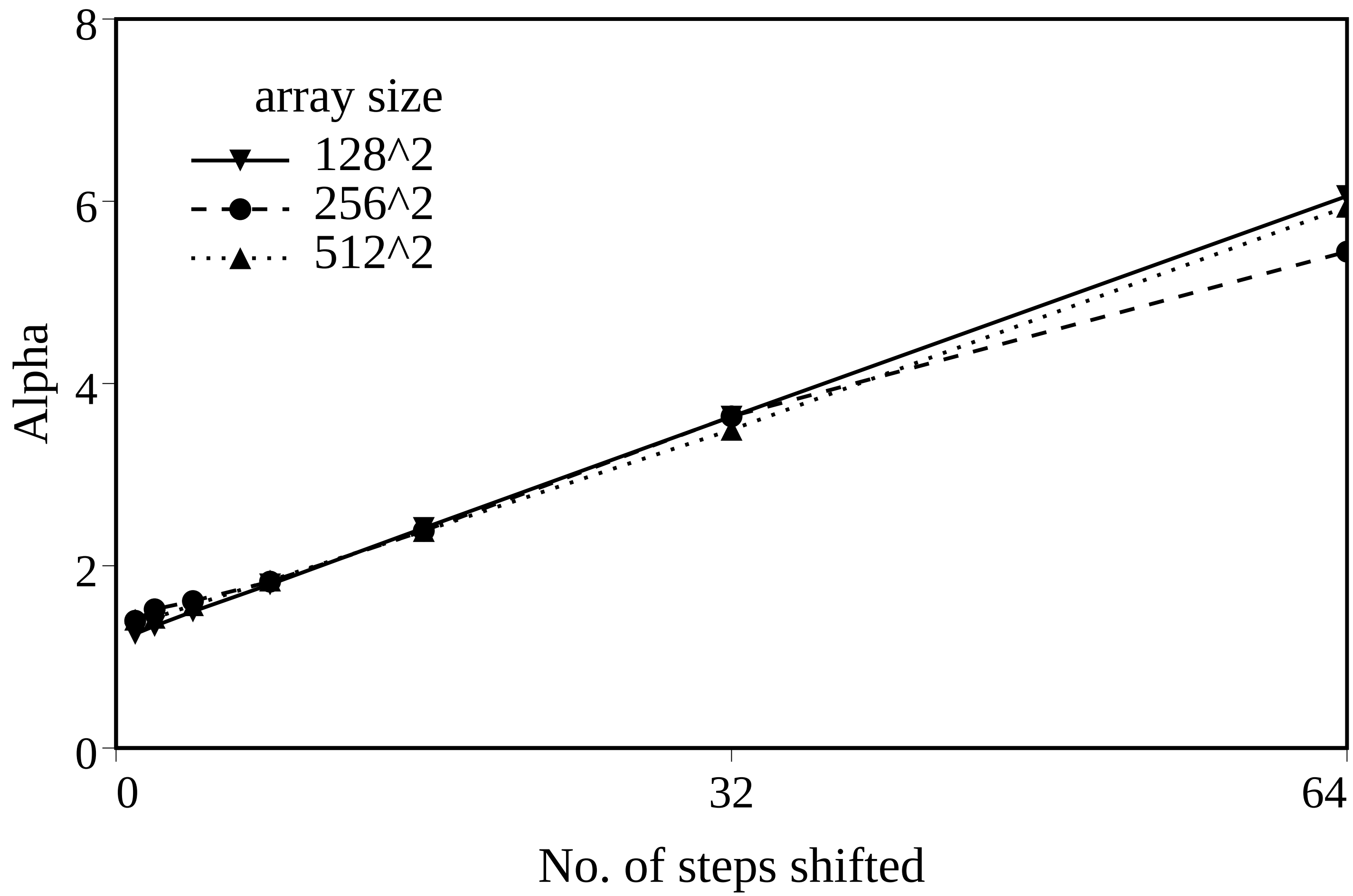


Figure 16. Alpha (Compute/shift speed) on a 610C (4k).