EXPERIMENTAL RESULTS FOR VECTOR PROCESSING ON THE ALLIANT FX/8*

by

Walid Abu-Sufah and Allen D. Malony

Center for Supercomputer Research and Development
University of Illinois at Urbana-Champaign
104 S. Wright St.
Urbana, Ill. 61801-2987

*This work was supported in part by the **National Science Foundation** under Grant No.'s NSF DCR84-10110, and NSF DCR84-06916, the **US Department of Energy** under Contract No. DOE DE FG02-85ER25001, and the **IBM** Donation.

Abstract

The Alliant FX/8 multiprocessor implements several high-speed computation ideas in software and hardware. Each of the 8 computational elements (CEs) has vector capabilities and multiprocessor support. Generally, the FX/8 delivers its highest processing rates when executing vector loops concurrently [Alli85]. In this paper, we present extensive empirical performance results for vector processing on the FX/8. The vector kernels of the LANL BMK8a1 benchmark are used in the experiments. We execute each kernel on 1 and 8 CEs and show the measured execution rate (in MFLOPS) as a function of vector length. We assess the performance of 1 CE as a vector processor by finding the vector lengths where vector processing exceeds that of scalar processing and calculating Hockney's $n_{1/2}$. For 8 CEs, we give upper/lower bounds on the achieved speedups and on the multiprocessing overhead. We also show the speedup variation as the number of CEs increases from 2 to 8. Our results reveal some interesting phenomena. Vector processing performance in a machine with a multi-level memory hierarchy, such as the FX/8, depends significantly on where the referenced vectors reside. Execution from memory, rather than from cache, degrades performance by a factor up to 3.7. Although speedups around 7 can be achieved for most stride-1 kernels when executed on 8 CEs, the maximum execution rates occur only for a narrow range of vector lengths ($O(1000)$). Performance drops rapidly when the vector lengths deviate slightly from the optimal values. This phenomena is not observed when executing on a single CE; the peak performance is obtained when the vectors are 32 elements long and remains close to the maximum for longer vector lengths ($O(1000)$). The kernels do not gain any appreciable speedup when the number of CEs is increased beyond 4 for short ($O(100)$) or long ($O(10{,}000)$) vectors. Multiprocessing of some indexed vector kernels results in almost no speedup due to the synchronization necessary to enforce output dependencies.

1. Introduction

The Alliant FX/8 is a shared memory multiprocessor system with a maximum advertised performance of 94.4 millions of floating point operations per second (MFLOPS) for single precision computations [Alli85]. Each of its 8 computational elements (CEs) has vector processing capability with a peak advertised execution rate of 11.8 MFLOPS. The FX/8 is one of the several machines which have been announced in the last few years that use different forms of parallelism to exceed the performance attainable from the technology used in the implementation[1]. The FX/8 combines several interesting high-speed computation ideas in both software and hardware [Flyn72], [Kuck78], [Kuck81], [KSCV84], [GGSB85], [HwBr84]. It has an interactive optimizing Fortran compiler which transforms loops in subroutines to execute in vector mode on a single CE, vector-concurrent mode on multiple CEs, or scalar-concurrent mode on multiple CEs [Alli85]. The operating system, Concentrix, is a multiprocessor Unix based on Berkeley 4.2 BSD. Multiprocessing is realized by concurrency control hardware in each CE which is accessed using special concurrency instructions. The 8 CEs of the system are crossbar connected to a shared, direct mapped, cache. The cache is connected to the shared memory via a bus. A more detailed description of the FX/8 is presented in the Section Two.

The performance assessment of a vector multiprocessor machine, like the FX/8, is important because of the great amount of effort that was spent in the last decade to develop

[1] For a rather comprehensive survey of such machines and brief descriptions see [DoDu85], [Dong86].

vector algorithms for different applications and to enhance the capabilities of vectorizing preprocessors to detect vector loops in dusty deck codes [KSCV84]. In addition, there is little empirical data in the literature modeling the behavior of vector multiprocessors [BuSi85], [Flat86]. This paper presents empirical results on the vector performance of the Alliant FX/8 multiprocessor. The thirteen vector kernels of the Los Alamos National Laboratory benchmark **BMK8a1** (for double precision computations) were used in our experiments [GrSi84].

In Section Three we report and discuss the experimental results. We show the delivered performance for each kernel when executed on one CE. A single CE demonstrates the classical performance behavior of a vector processor where the maximum performance is sustained over a wide range of vector lengths. However, as cache misses increase for longer vector lengths, the performance drops sharply to a rate where the cache hit ratio is at a minimum. To characterize the vector performance of 1 CE, we determine the vector length where each kernel starts executing faster in vector mode than in scalar mode and calculate $n_{1/2}$, the vector length at which the CE is supposed to deliver half of its peak performance (r_{∞}), as described by Hockney in [HoJe81]. These results indicate that a single CE processes short vectors efficiently.

For 8 CEs, we present the delivered performance, speedup, and multiprocessing overhead for each kernel. We observe that the execution rate increases as the vector length increases and then drops significantly to a minimum rate. The results show that the maximum performance for each kernel on 8 CEs is sustained over a significantly smaller

range of vector lengths than for 1 CE. This is reflected in the speedup and multiprocessing overhead calculations. Speedup, $S_p(n)$, is defined as t_1/t_p where t_1 and t_p are the execution times of a kernel for vector length n executed on 1 and p CEs, respectively. We define the **machine efficiency**, $E_m(n)$, as the maximum delivered execution rate divided by the peak advertised execution rate of the machine and **multiprocessor efficiency**, $E_p(n)$, as $S_p(n)/p$. Multiprocessing overhead, $OV_p(n)$, is equal to 1-$E_p(n)$. Our results indicate that only modest improvements in speedup are achieved when processing short ($\leq$100) and long ($\geq$10,000) vectors on more than 4 CEs.

To determine the effect that the cache has on performance, we repeat the experiments for each kernel such that cache misses will be encountered whenever possible. Our measurements show that the performance decreases by a factor up to 3.7 when the vectors are referenced from memory instead of from the cache.

The BMK8a1 benchmark contains kernels with subscripted vectors. These vector kernels run slower than the stride-1 kernels. In Section Three, using one of the indexed kernels, we briefly discuss issues which affect the performance of such kernels. In Section Four we make some concluding remarks.

2. The Experimental Environment

We performed our experiments at the Center for Supercomputing Research and Development (CSRD) of the University of Illinois[2]. The configuration of the FX/8 used for these experiments is shown in Figure 1. The computational complex of the FX/8 contains 8 CEs. When executing concurrency instructions, the CEs communicate via a concurrency control bus. Each CE has a computational clock period of 170 nsec with a peak execution rate of 11.8 MFLOPS and 5.9 MFLOPS for single and double precision computation, respectively [Alli85], [DoDu85]. With the 8 CEs working concurrently, the FX/8 advertised peak performance is 47.2 MFLOPS for double precision computations. The CEs are connected by a crossbar switch to a direct-mapped, write-back, shared cache of 16K double precision words[3]. The cache is implemented in 4 quadrants with a peak interleaved bandwidth to the CEs of 47.125 MW/sec. It is connected to a 4 MW shared memory via a bus with a peak bandwidth of 23.5 MW/sec. The system also contains 6 interactive processors (IPs) connected to their own caches as shown in Figure 1. The IPs primarily perform operating system related functions and I/O operations.

A computational element has vector processing capabilities as well as multiprocessing support. It has a rich set of arithmetic, logical and comparison vector instructions plus vector move instructions including scatter, gather and merge. There are 8 32-bit data registers, 8 address registers, 8 double precision floating point registers, and 8 32-

[2] At the time the work reported in this paper was performed, the machine did not run production releases of the OS and the compiler. However, we believe that our conclusions will not change significantly when these releases are available.

[3] A double precision word is 64 bits wide.

element, double precision vector registers in each CE. Operands of vector instructions can come from vector registers, vector and floating point registers, or vector registers and the cache. Chaining is also supported for vector add-multiply and vector multiply-add instructions. Multiprocessing is supported by concurrency instructions which permit iterations of a loop to be executed concurrently across multiple processors in the CE complex.

The Alliant FX/8 Fortran compiler provides automatic detection of vector and/or multiprocessed loops. It optimizes code for scalar, vector and concurrent execution. Based on data dependency analysis, loops are optimized to execute in one of four modes: vector, scalar-concurrent, vector-concurrent, or concurrent-outer/vector-inner [Alli85]. The FX/8 operating system, Concentrix, extends Berkeley Unix 4.2 to provide support for multiple processors and a large virtual space.

The FX/8 system maintains timing information for each program which is accessible through Fortran library routines and can be used for measurement purposes. Our experimentation procedure attempted to remove any inconsistencies that might result in the performance measurements due to the resolution of these timing tools by assuring a long running time relative to the granularity of the timed event. This was achieved by enclosing each kernel in a serial timing loop which repeats the execution of the kernel as many times as needed to obtain reliable timing data. All measurements were performed in stand-alone mode. Each vector kernel was executed five times for vector lengths varying between 1 and 100,000. The repetition of the experiments was necessary due to

significant variations in the execution rates from one run to another for certain regions of vector lengths.

The Los Alamos National Laboratory benchmark BMK8a1 contains thirteen vector kernels designed to reflect the vector statements which are widely encountered in scientific codes [GrSi84]. Each kernel is a different combination of add and multiply operations of vectors and scalars that stores the outcome into a result vector. Some kernels use an additional vector to index an operand or result vector. In our notation used to identify the different kernels, ***v*** is a vector, ***s*** is a scalar, ***p*** denotes addition, ***t*** denotes multiplication, and ***i*** denotes an indexing vector[4]. For instance, ***vts*** is the kernel ***v1 = v2 * s*** and ***vi=vtv*** is the kernel ***v1(i+k) = v2 * v3*** where ***i*** is the indexing vector and ***k*** is a constant. The complete list of vector kernels is:

vps	***vtv***	***vtspvts***	***vips***
vts	***vtvps***	***vtvpv***	***vi=vtv***
vpv	***vpvts***	***vtvpvtv***	***vpvtvi***
			vi=vipvtv

[4] ***vi*** denotes a vector, ***v***, indexed by another vector, ***i***.

3. Experimental Results

The performance of a one CE system is measured in order to calculate speedup and other performance metrics for multiple CEs. Examining the one CE results reveals interesting characteristics of the behavior of a vector processor when accessing data in a multi-level memory. The 8 CEs results show the performance improvement obtainable from vector-concurrent operation. The vector length region where maximum execution rate is achieved using 8 CEs is narrower than for one CE. However, the speedup in this region is around 7 for most stride-1 kernels. Comparing the performance for one and multiple CEs reveals important observations on the number of CEs which can be efficiently employed in a vector multiprocessor system. The performance results for the indexed kernels provides qualitative measure of the difficulties encountered when attempting to improve the performance of some codes using multiple vector processors.

3.1. One CE Performance

Figure 2 shows the maximum measured execution rate as a function of vector length for each kernel running on 1 CE. We observe that the behavior of all the kernels is similar; the computational rate increases as the vector lengths increase and reaches a maximum at vector length n_{peak}. For each kernel, the execution rate stays within a small percentage of the maximum until the vector length reaches a value denoted by n_{drop}. The computational rate then starts to fall until a vector length denoted by n_{min} is reached. For vectors longer than n_{min} the execution rate remains rather constant. These three vector length points are shown for the ***vtspvts*** kernel in Figure 2. We identify four

regions for each performance curve: the **cache rate region** ($1 \leq n \leq n_{drop}$), **maximum rate region** ($n_{peak} \leq n \leq n_{drop}$), the **falloff region** ($n_{drop} < n \leq n_{min}$), and the **minimum rate region** ($n > n_{min}$). In the cache rate region, the size of the data referenced by each kernel is small enough such that the cache hit ratio is maximized. The performance in this region is characteristic of vector processors where the execution rate rapidly increases to a maximum point which is sustained as the vector length increases. The wide range of vector lengths where the execution rate stays within a small percentage of the maximum identifies the maximum rate region. The falloff region begins when the cache hit ratio starts decreasing. As the cache hit ratio continues to decrease for longer vector lengths, the number of the references to the shared memory increases and the performance drops until the cache hit ratio reaches its minimum at n_{min}. The percentage variation between the maximum and minimum rates is the largest in the falloff region due to the non-deterministic behavior of the cache[5]. In the minimum rate region, the size of the referenced data is so large that a cache miss occurs whenever the kernel accesses the first word of a cache block[6].

Several factors affect the delivered performance for a given kernel at a particular vector length. These factors include the number of memory references, the number of floating point operations performed, the types of floating point operations, and the degree of chaining in the kernel. The ***vps*** kernel runs faster than the ***vts*** kernel because of operation type; ***vtvps*** runs faster than ***vtv*** due to the number of floating point operations per-

[5] The percentage variation for all the kernels are shown in Appendix A.

[6] A cache block contains four double precision words.

formed and chaining; ***vtspvts*** runs faster than ***vtvpvtv*** due to the difference in the number of memory references. Table 1 shows the maximum MFLOPS measured for each kernel on 1 CE. The maximum execution rate occurs at vector length 32 for all kernels. This is expected since the vector registers are full at this vector length. Table 2 shows the execution rate for each kernel in the minimum rate region. Performance in the maximum rate region can be two times greater than the performance in the minimum rate region.

In order to determine how efficiently one CE processes short vectors, we found the vector length where execution in vector mode starts to be faster than in scalar mode. More performance can be achieved in vector mode for vector lengths > 2 for the stride-1 kernels and >6 for the indexed kernels. Hockney's $(n_{1/2}, r_{\infty})$ model can also be used to characterize the vector performance for one CE [HoJe81]. Table 3 shows that all kernels have an $n_{1/2} \leq 4$. This indicates that short vectors will be processed efficiently on the FX/8. However, most kernels deliver only around 1/3 of the measured peak execution rate at $n_{1/2}$ when executed on 1 CE instead of half the peak rate as expected by Hockney's model. It can be shown that Hockney's two parameter model $(n_{1/2}, r_{\infty})$ is simplistic when used to model real vector processors [AbuS86], [LeAK84].

3.2. Eight CEs Performance

3.2.1. Delivered Execution Rate

Figure 3 shows the maximum performance results when executing in vector-concurrent mode on 8 CEs. We observe that the execution rate rises more slowly and reaches the maximum rate region for much longer vectors than in the 1 CE case ($O(1000)$ compared to 32). This is partially due to the fact that vector-concurrent execution partitions the vector operation equally among the 8 CEs and longer vectors are required before the vector registers of each CE are maximally utilized[7]. Multiprocessing overhead associated with starting and sustaining vector-concurrent operations accounts for the further increase needed in vector length before the maximum rate is achieved. The performance of a kernel on 8 CEs is affected by the multi-level memory hierarchy for the same reasons as in the 1 CE case. In fact, we observed that the falloff region in the 8 CE performance curves coincides with the falloff region in the 1 CE performance curves for each kernel. However, the percentage drop in MFLOPS in the falloff region is greater with 8 CEs. We also notice that the percentage variation between the maximum and minimum execution rates is largest in the falloff region and is greater than the 1 CE case[8]. Due to the initial slow performance increase to the maximum rate and the fixed falloff region, the maximum rate region spans a much smaller vector length interval than in the 1 CE case. This result has ramifications on how codes should be structured, with respect to vector length, so as to maximize the vector-concurrent performance when running on 8 CEs. In particular, we notice that if vector lengths deviate slightly from the maximum

[7] This analysis is supported by the observation that the execution rate has a local maximum at vector length 256 for almost all of the kernels. At vector length 256, the vector registers for each CE are full.

[8] The percentage variation for 8 CEs is shown in Appendix A.

rate region, performance degrades rapidly.

Table 1 shows the maximum peak MFLOPS measured for each kernel when executing on 8 CEs, the vector length where the peak performance is delivered, and the machine efficiency (E_m) at this vector length[9]. The machine efficiency is less for 8 CEs than for 1 CE due to multiprocessing overhead. All the kernels achieve $< 50\%$ machine efficiency and 10 kernels are $< 30\%$ efficient for 8 CEs. Table 2 is analogous Table 1 except the data for the MFLOPS in the minimum rate region is presented. It can be seen from the MFLOPS and the machine efficiency that a low cache hit ratio significantly reduces the performance. This subject is discussed in more detail in the section 3.3.

3.2.2. Speedup Results

Speedup is defined as $S_p(n) = t_1/t_p$ where t_1 and t_p are the execution times of a kernel for vector length n executed on 1 and p CEs, respectively[10]. Since there were variations in measuring t_1 and t_8 over the five runs, we define the lower bound on the speedup of a kernel with vector lengths n, $L_{S(n)}$, to be the ratio of the smallest of the five t_1's and the largest t_8. The upper bound on the speedup, $U_{S(n)}$ is calculated as the ratio of the largest measured t_1 to the smallest t_8. Figure 4 shows the upper and lower bound speedup curve for the *vtv* kernel[11]. We observe that for all kernels the speedup upper bound is less than 4 for vector lengths smaller than 500 and less than 5 for vector

[9] By definition, the maximum E_m occurs at this vector length.
[10] In the remainder of the paper, $S(n)$ will be used to denoted $S_8(n)$.
[11] The speedup curves for the other kernels are found in Appendix B.

lengths greater than 10,000. The maximum upper bound speedups for all the kernels are shown in Table 4. Eight of the 13 kernels have a maximum upper bound speedup greater than 7. The speedup of 4 of the remaining kernels is between 5 and 7. The indexed kernels have the smallest speedups; ***vi=vtv*** has a maximum upper bound speedup of only 1.32.

By comparing the vector lengths in Tables 1 and 4 we observe that the vector lengths where the peak MFLOPS and maximum speedup occur are not necessarily the same for a given kernel. Table 5 shows the speedup upper bound at the vector lengths where each kernel runs at its peak execution rate. We notice from Figure 4 and Tables 4 and 5 that the lower and upper bounds on speedup can differ significantly (up to 57%). This variation occurs in the falloff region and is primarily a result of the nondeterministic behavior of the cache in this region from one run to another. However, this variation is insignificant for short and long vector lengths ($n<500$ and $n>10{,}000$).

Figure 5 shows the speedup as a function of the number of CEs for stride-1 kernels. The envelopes in the figure enclose the speedup curves for all kernels at vector lengths 100, 1K and 100K. The speedup curve for ***vtvps*** is shown as a dashed line and roughly represents the median speedup within each speedup envelope. We observe that for both short and long vectors the speedup gained by increasing the number of processors beyond 4 is modest for most kernels. As the number of CEs increase from 4 to 8, the speedups approach 4.5 and 4 for vector lengths of 100 and 100K, respectively. For vector lengths of 1K, speedups are close to being linear in the number of CEs. These

speedup results could be very useful to designers of multi-million dollar multiprocessor vector machines (e.g., the new Cray multiprocessors) in light of the mean vector lengths encountered in application codes ($\leq$468 in the Lawrence Livermore National Lab workload [McMa85]).

3.2.3. Multiprocessing Overhead

The percentage multiprocessing overhead of a machine with p processors when executing a kernel with vector length n, $OV_p(n)$, is given by $(1\text{-}S_p(n)/p)*100\%$; $OV(n)$ denotes the overhead when p=8. We let $U_{OV(n)}$ denote the upper bound on the overhead and $L_{OV(n)}$ denote the lower bound. Figure 6 shows the upper and lower bound overhead curve for the ***vpvts*** kernel[12]. For all kernels, $U_{OV(n)}$ is greater than 50% for short ($n \leq 100$) and long ($n \geq 10{,}000$) vectors. For six of the nine stride-1 kernels, $U_{OV(n)}$ is less than 25% for vector lengths in the region $1000 \leq n \leq 2000$. The overheads of the indexed vector kernels are greater than 30% for all vector lengths.

Table 6 identifies the vector lengths where the minimum $U_{OV(n)}$ occurs for all the kernels. We note that for the 9 stride-1 kernels the minimum $U_{OV(n)}$ is between 10-20%. For the indexed kernels, the minimum $U_{OV(n)}$ is close to 30% for 1 kernel, 40% for 2, and 85% for the fourth kernel.

[12] The overhead curves for the other kernels are found in Appendix C.

3.3. Execution from Memory

In order to measure the vector-concurrent execution rate for a kernel with vector length n ($1 \leq n \leq 100{,}000$) such that the the maximum number of cache misses occurs, we reference the vectors as columns of two-dimensional arrays. The kernel is executed for the first column (of length n), then the second column, and so on. Since every column is distinct, new vectors are always being referenced. Also, the two-dimensional array sizes are declared such that when a column is referenced again by the timing loop, none of the data from the previous reference of that column will be present in the cache. We refer to the running of a kernel in this fashion as **execution from memory.** When a kernel is not restricted to execute in this manner and is able to take full advantage of the cache, we say that the kernel is **executing from cache.**

Figure 7 shows the execution rate from memory and from cache for the ***vtspvts*** kernel using 8 CEs[13]. When the kernel executes from memory, the execution rate rises slowly as the vector length increases and reaches a maximum that coincides with the rate obtained in the minimum rate region when the kernel executes from cache. The same behavior is observed for the other kernels [AbMa86]. Table 7 shows the maximum performance degradation factors when kernels execute from memory instead of from the cache. This factor ranges between 1.35 and 2.26 when running on 1 CE. When executing on 8 CEs and using stride-1 kernels, the degradation factor is larger and ranges between 3.03 and 3.67. For indexed kernels, the degradation factor is between 1.49 and 2.08.

[13] The execution rate from memory and from cache for the other kernels are shown in Appendix D.

It is obvious that if the memory speed were increased, the execution performance from memory would increase. It is also expected that by increasing the size of the cache, the maximum rate region of the execution from cache curve would be extended. A current limitation to increasing the performance in the minimum rate region of the two curves is the bandwidth available on the Data Memory Bus [AbMa86]. Improving the data memory bus bandwidth would have the effect of shifting the minimum rate region of the two curves upward.

3.4. The Behavior of Indexed Kernels

When running on a single CE, the execution rate of a kernel will drop if one or more of the referenced vectors are addressed indirectly. This is mainly due to an increase in the number of vector instructions generated by the compiler for an indexed kernel (scatter/gather instructions, etc.). Moreover, the memory access pattern of the indexed vector elements might result in an appreciable decrease in the utilization of the bandwidth of the interleaved memory modules.

The execution rate of a multiprocessed vector kernel could degrade significantly if the result vector is addressed indirectly. This can be seen clearly in Figure 3 when comparing the execution rate curves for kernels ***vtv*** and ***vi=vtv***. Examing the assembly code generated for the two kernels reveals that while the ***vtv*** kernel is executed as a single concurrent vector loop, the concurrent loop of the ***vi=vtv*** kernel encloses two vector loops. Each CE first executes the vector statement ***temp←vtv***, where ***temp*** is a temporary vector. While CE_0 continues by executing a second vector loop to scatter the

contents of ***temp*** to the specified elements of the result vector, each CE_i (i=1,...,7) waits for a synchronization signal from CE_{i-1} indicating that it has finished scattering its results. In this fashion, any output dependence relationships between different instances of the original statement ***vi=vtv*** will be preserved. Figure 8 shows the execution scheme of this kernel using 8 CEs. This explains the lack of any speedup when this kernel is executed concurrently. Synchronization is not required in kernels with no potential output dependencies. Kernels with output dependencies will gain speedup if the time spent evaluating the right hand side expression of the kernel is significantly larger than the time spent in the scattering loop by each CE. This is the case for the kernel ***vi=vipvtv*** where it is possible to attain a maximum speedup of 5.86 compared to 1.32 for the kernel ***vi=vtv***.

ever, interprocessor synchronization to satisfy potential output dependencies is the major reason for performance degradation of indexed kernels.

The successful use of an 8 CEs FX/8 for vector processing depends on observing the principle of locality of reference [Denn70]. This implies that the programmer (or optimizing compiler) should structure the code such that once certain sections of the program's data are resident in the cache, as much computation as possible is performed using this data before processing other sets of data [AbKL79], [AbKL81], [ALMY82], [JaMe86]. Some of the other factors that enhance the delivered performance are reducing the number of memory references in the vector statement, increasing the number of floating point operations performed using the same operands, and taking advantage of the chaining capabilities of the processors.

BIBLIOGRAPHY

[AbKL79] W. Abu-Sufah, D.J. Kuck, and D.H. Lawrie. *Automatic Program Transformations for Virtual Memory Computers.* **Proc. of the 1979 National Computer Conf.**, pp. 969-974, June 1979.

[AbKL81] W. Abu-Sufah, D.J. Kuck, and D.H. Lawrie. *On the Performance Enhancement of Pagin Systems through Program Analysis and Transformations.* **IEEE Trans. on Computers**, Vol. C-30, No. 5, pp. 341-356, May 1981.

[AbuS86] W. Abu-Sufah. *On Modeling the Execution of Vector Loops on SIMD Machines.* In preparation, 1986.

[Alli85] Alliant Computer Systems Corporation. **FX/Series Product Summary.** June 1985

[ALMW82] W. Abu-Sufah, R. Lee, M. Malkawi, and P. Yew. *Experimental Results on the Paging Behavior of Numerical Programs.* **6th Intl. Conf. on Software Eng.**, pp.110-117, Sept. 1982.

[BuSi85] I.Y. Bucher and M.L. Simmons. *Performance Assessment of Supercomputers.* in **Vector and Parallel Processors: Architecture, Applications, and Performance Evaluation**, edited by Myron Ginsberg, North Holland, also Los Alamos National Laboratory Report **LA-UR-85-1505**, 1985.

[Denn70] P.J. Denning. *Virtual Memory.* **Computing Surveys**, Vol. 2, No. 3, pp. 153-180, Sept. 1970.

[DoDu85] J.J. Dongarra and I.S. Duff. *Advanced Architecture Computers.* Mathematics and Computer Science Division, Tech. Memo #57,Argonne National Laboratory, Sept. 1985.

[Dong86] J.J. Dongarra. *The Superminis Today and Trends for the Future.* to appear in **COMPCON '86**, March 1986.

[Flat86] H.P. Flatt. *Some Limitations of Parallel Processing.* invited presentation, Center for Supercomputing Research and Development, Univ. of Illinois, May 1986.

[Flyn72] M. Flynn. *Some Computer Organizations and their Effectiveness.* **IEEE Trans. on Computers**, Vol. C-21, No. 9, pp.948-960, Sept. 1972.

[GGSB85] D. Gajski, D. Gannon, J. Schwartz, and J. Browne. *Classification of Parallel Processor Architectures - Invited Tutorial Session.* **12th Int. Symp. on Computer Arch.**, June 1985.

[GrSi84] J.H. Griffin and M.L. Simmons. *Los Alamos National Laboratory Computer Benchmarking 1983.* Tech. Report LA-10151-MS, Los Alamos National Laboratory, June 1984.

[Hock84] R.W. Hockney. *Performance of Parallel Computers.* in **High-Speed Computation,** edited by J.S. Kowalik, pp. 159-175, Springer-Verlag, Berlin, Germany, 1984.

[HoJe81] R.W. Hockney and C. Jesshope. **Parallel Computers.** Adam Hilger Ltd., Bristol, England, 1981.

[HwBr84] K. Hwang and F. Briggs. **Computer Architecture and Parallel Processing.** McGraw-Hill, New York, New York, 1984.

[JaMe86] W. Jalby and U. Meier. *Optimizing Matrix Operations on a Parallel Multiprocessor with a Memory Hierarchy.* **Inter. Conf. on Parallel Proc.,** 1986.

[KSCV84] D. Kuck, A. Sameh, R. Cytron, A. Veidenbaum, C. Polychronopoulos, G. Lee, T. McDaniel, B. Leasure, M. Wolfe, C. Beckman, J. Davis, and C. Kruskal. *The Effects of Program Restructuring, Algorithm Change, and Architecture Choice on Program Performance.* **1984 Intl. Conf.on Parallel Processing,** pp. 129-138, Aug. 1984.

[Kuck78] D. Kuck. **The Structure of Computers and Computation.** John Wiley and Son, 1978.

[Kuck81] D. Kuck. *Automatic Program Restructuring for High-Speed Computation.* **Proc. of COMPAR81, Conf. on Analysis Problem-Classes and Programming for Parallel Computing,** edited by W. Handler, Nurnberg, Germany, 1981.

[LeAK84] K.Y. Lee, W. Abu-Sufah, and D. Kuck. *On Modeling Performance Degradation Due to Data Movement in Vector Machines.* **Proc. of the 1984 Conf. on Parallel Processing,** pp 269-277, August 21-24, 1984.

[LuMM85] O. Lubeck, J. Moore, and R. Mendez. *A Benchmark Comparison of Three Supercomputers: Fijitsu VP-200, Hitachi S810/20, and Cray X-MP/2.* **IEEE Computer,** pp. 10-24, Dec. 1985.

[McMa85] F.H. McMahon. *L.L.N.L. Fortran Kernels: MFLOPS Program.* Lawrence Livermore National Laboratory, Dec. 1985.

[Same85] A. Sameh. *Numerical Algorithmss on the Cedar System.* **Second SIAM Conference on Parallel Processing,** invited presentation, Norfolk, Virginia, Nov. 1985.

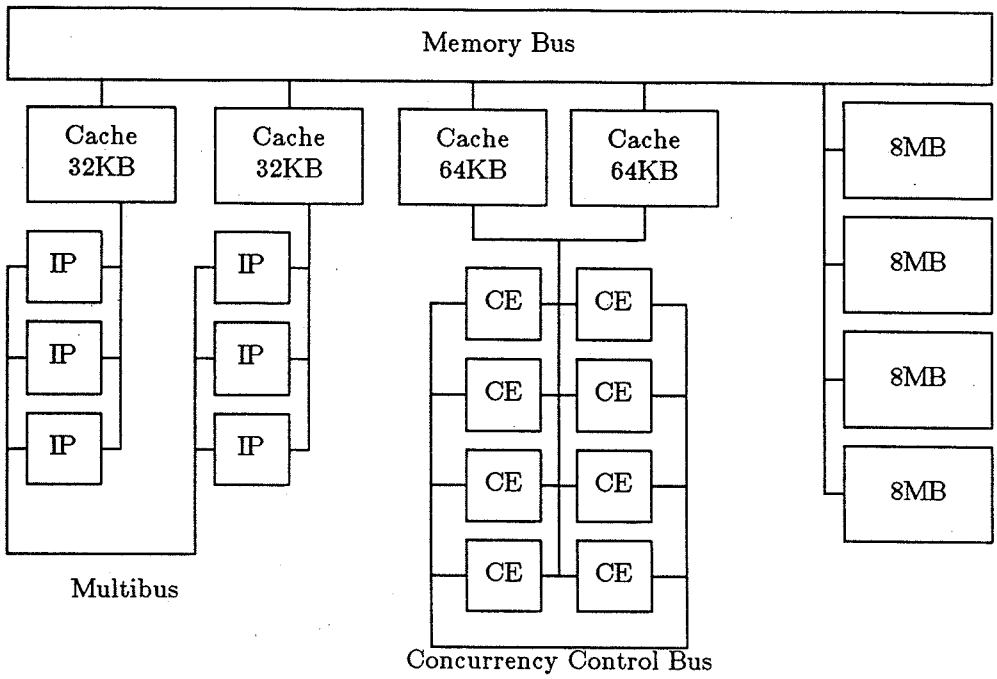


Figure 1.
The Configuration of the Alliant FX/8 used in the Experiments

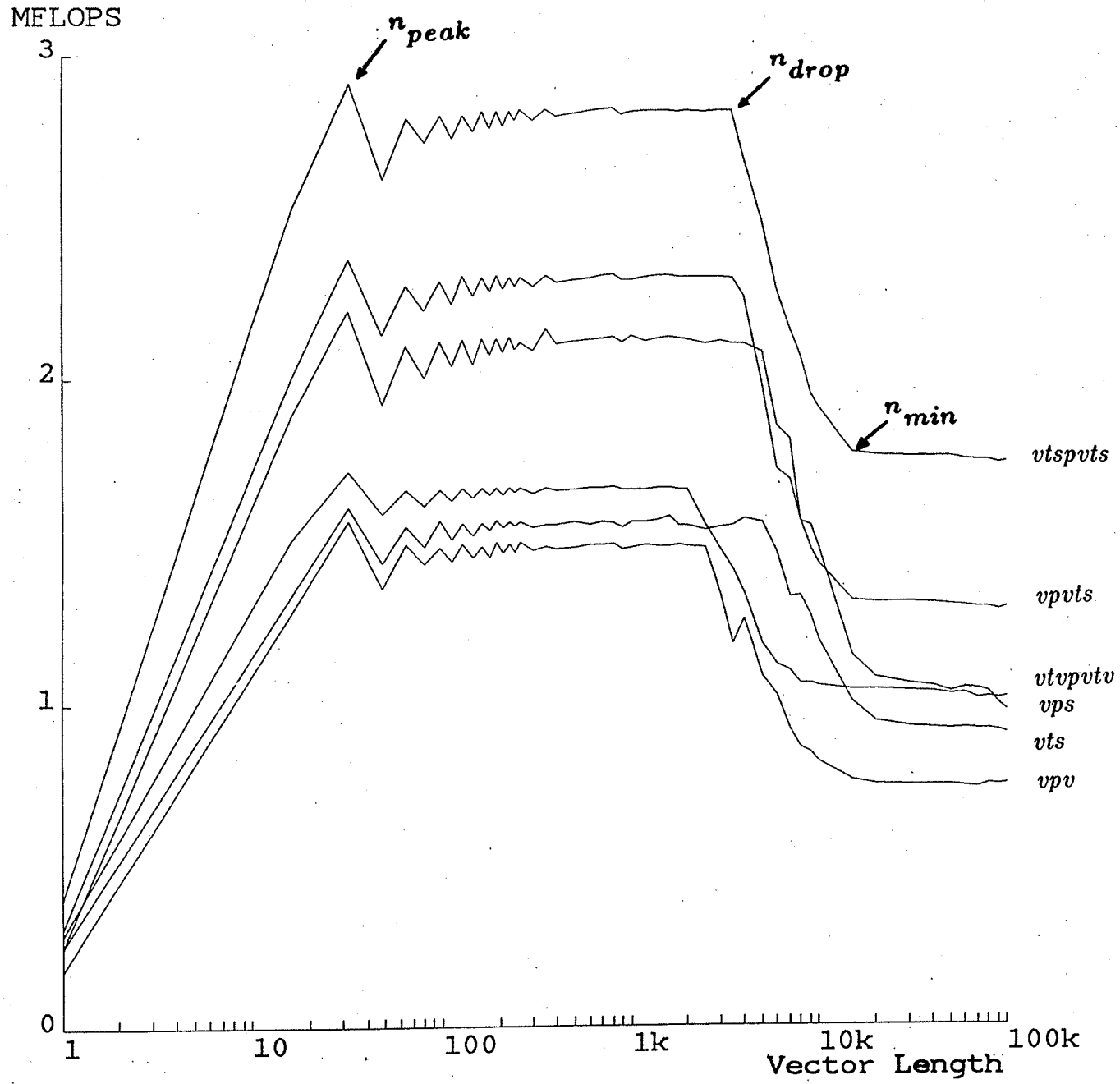


Figure 2-a
The Maximum Delivered Performance for a 1 CE System

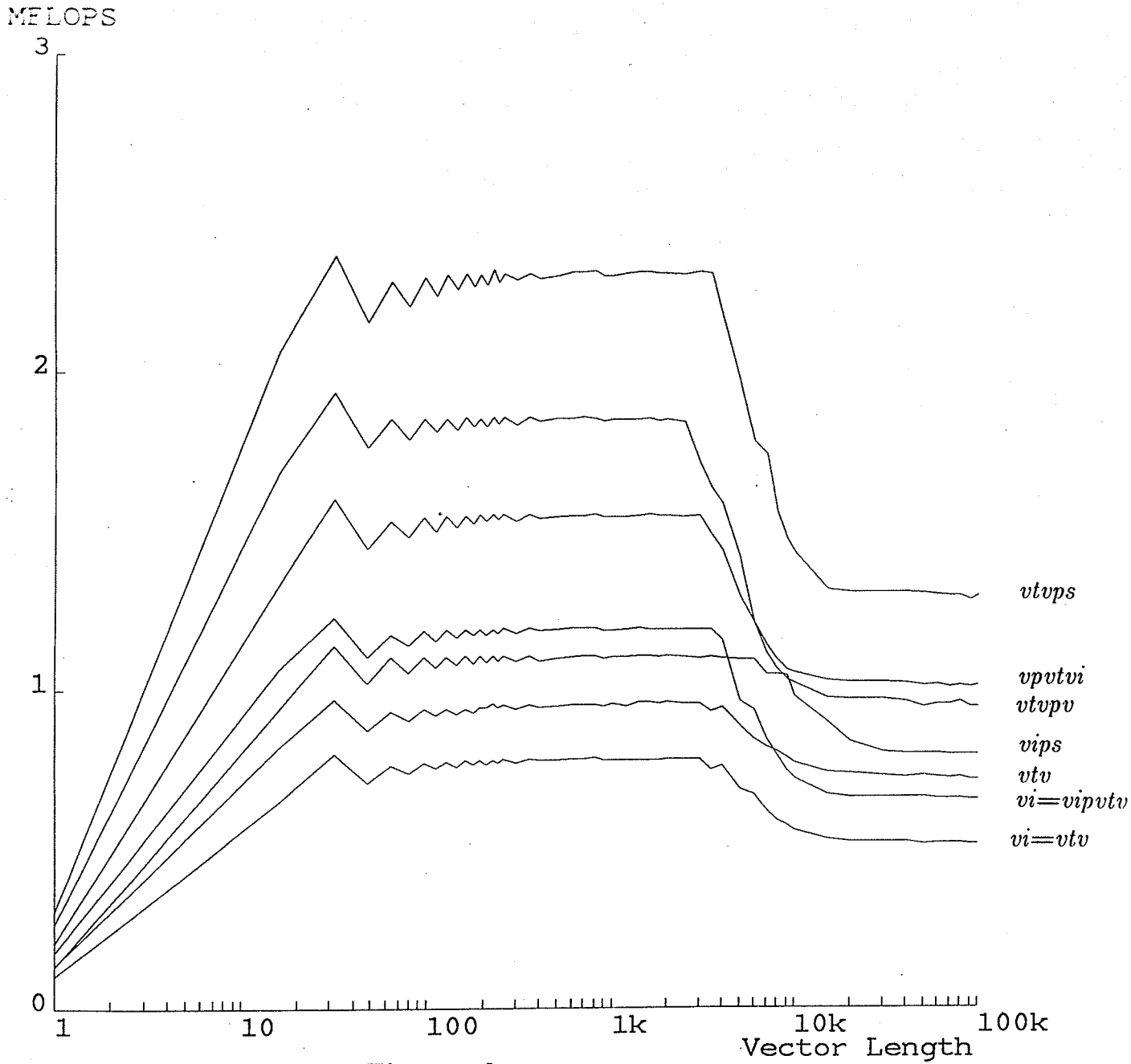


Figure 2-b
The Maximum Delivered Performance
for a 1 CE System (continued)

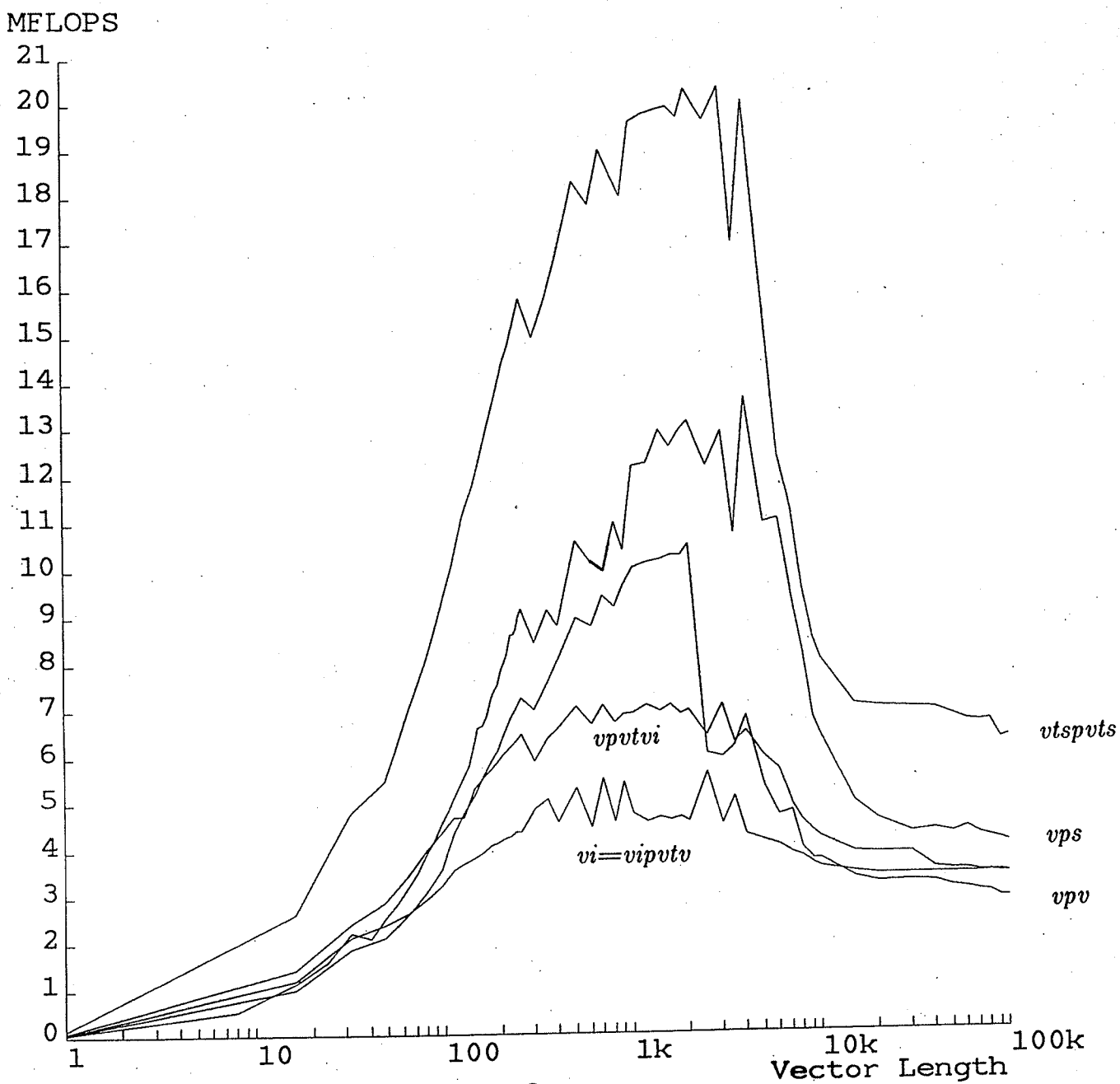


Figure 3-a
The Maximum Delivered Performance
for an 8 CE System

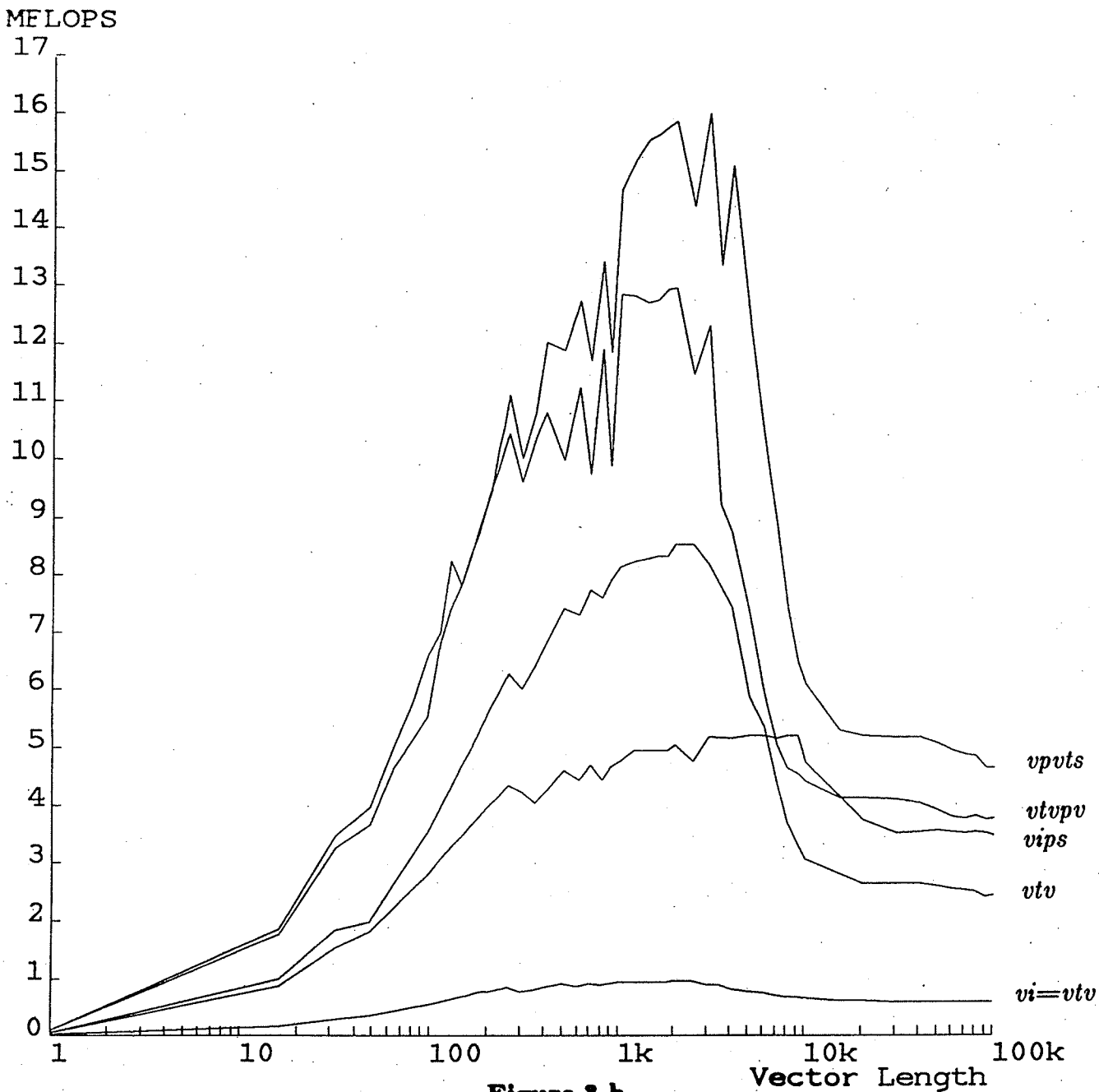


Figure 3-b
The Maximum Delivered Performance
for an 8 CE System (continued)

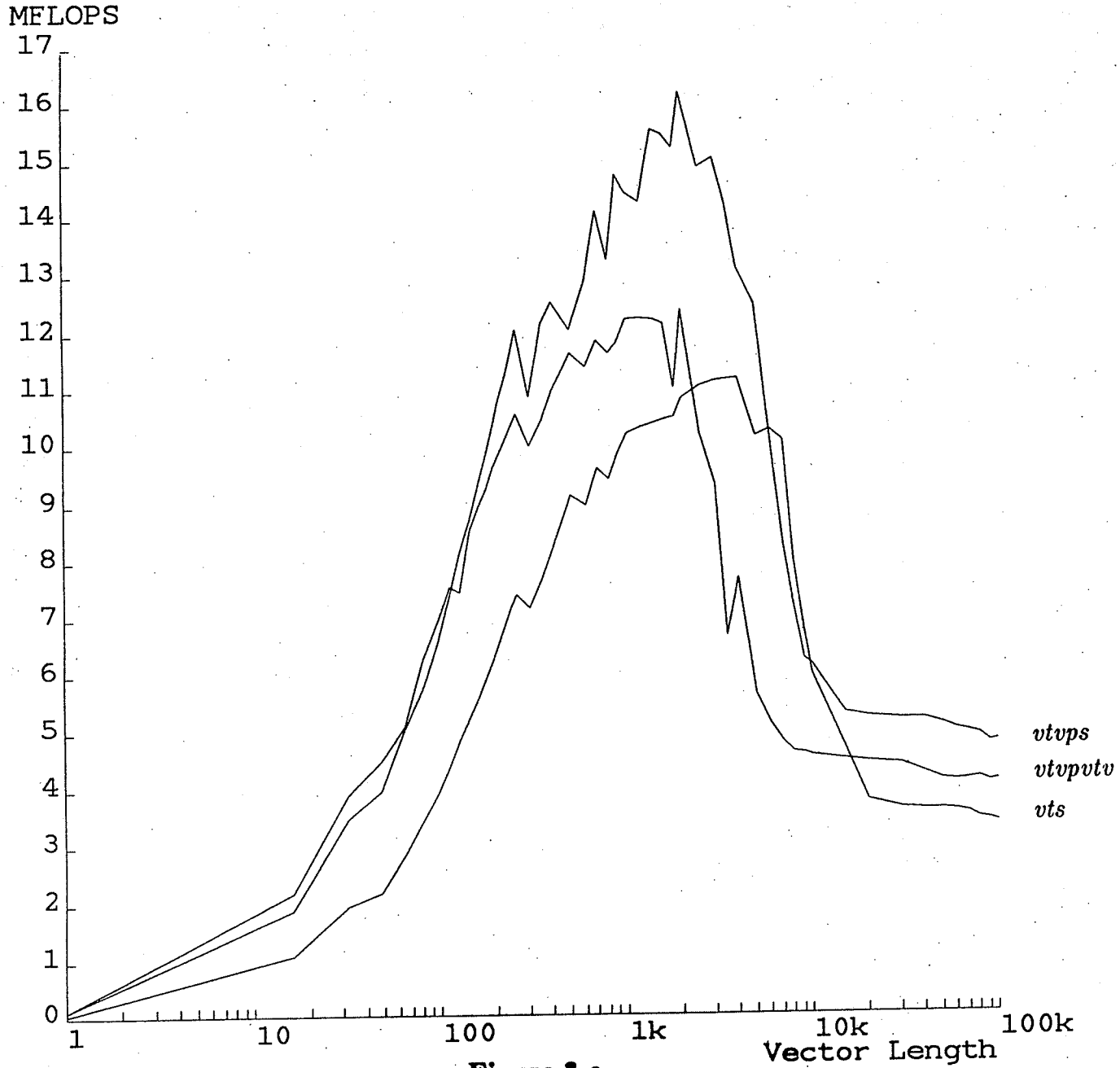


Figure 3-c
The Maximum Delivered Performance
for an 8 CE System (continued)

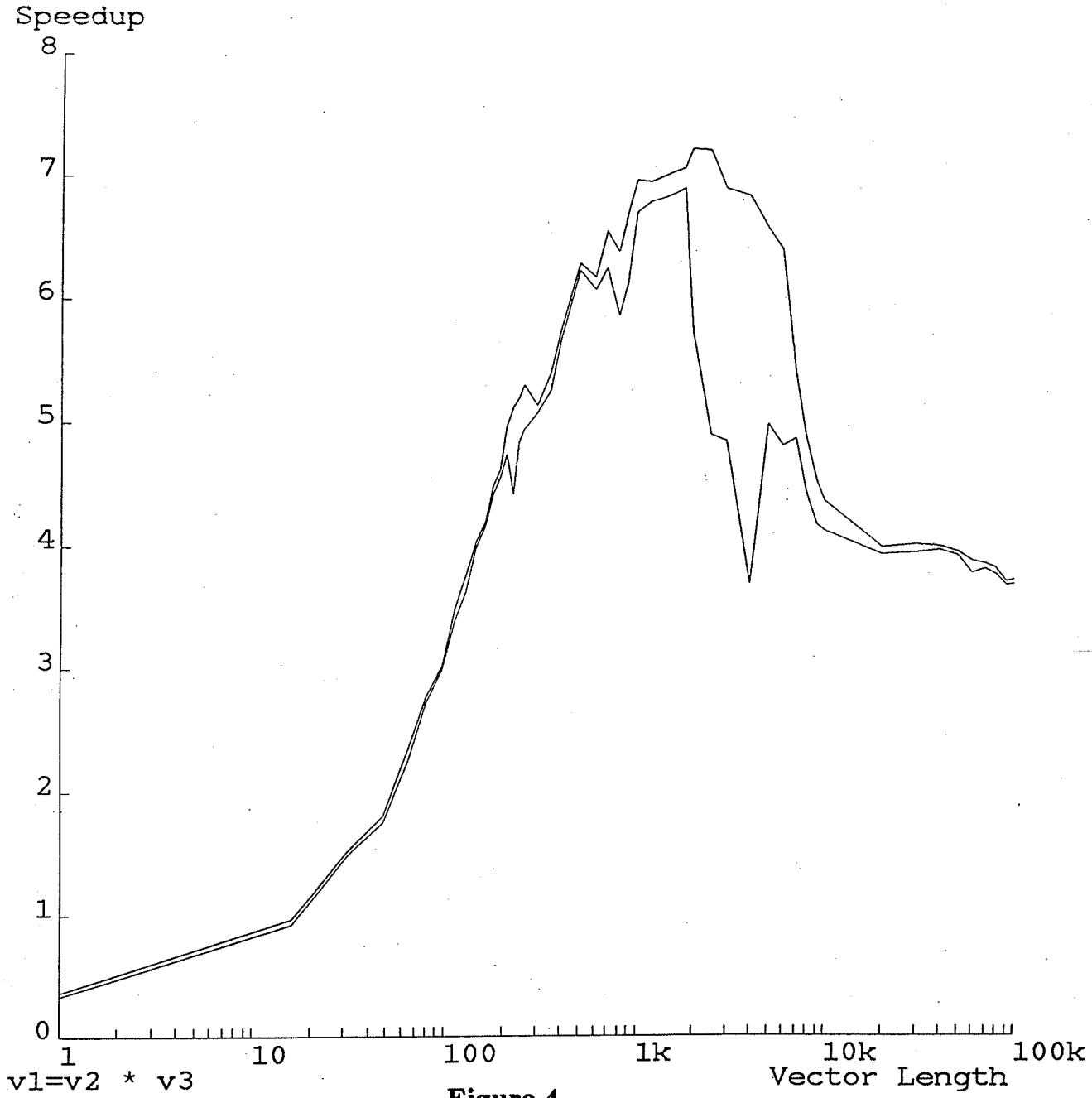


Figure 4
The Speedup Upper/Lower Bounds
for Kernel *vtv*

Figure 5-a.
Speedup vs. #CEs
or Vector Lengths 100 and 1K

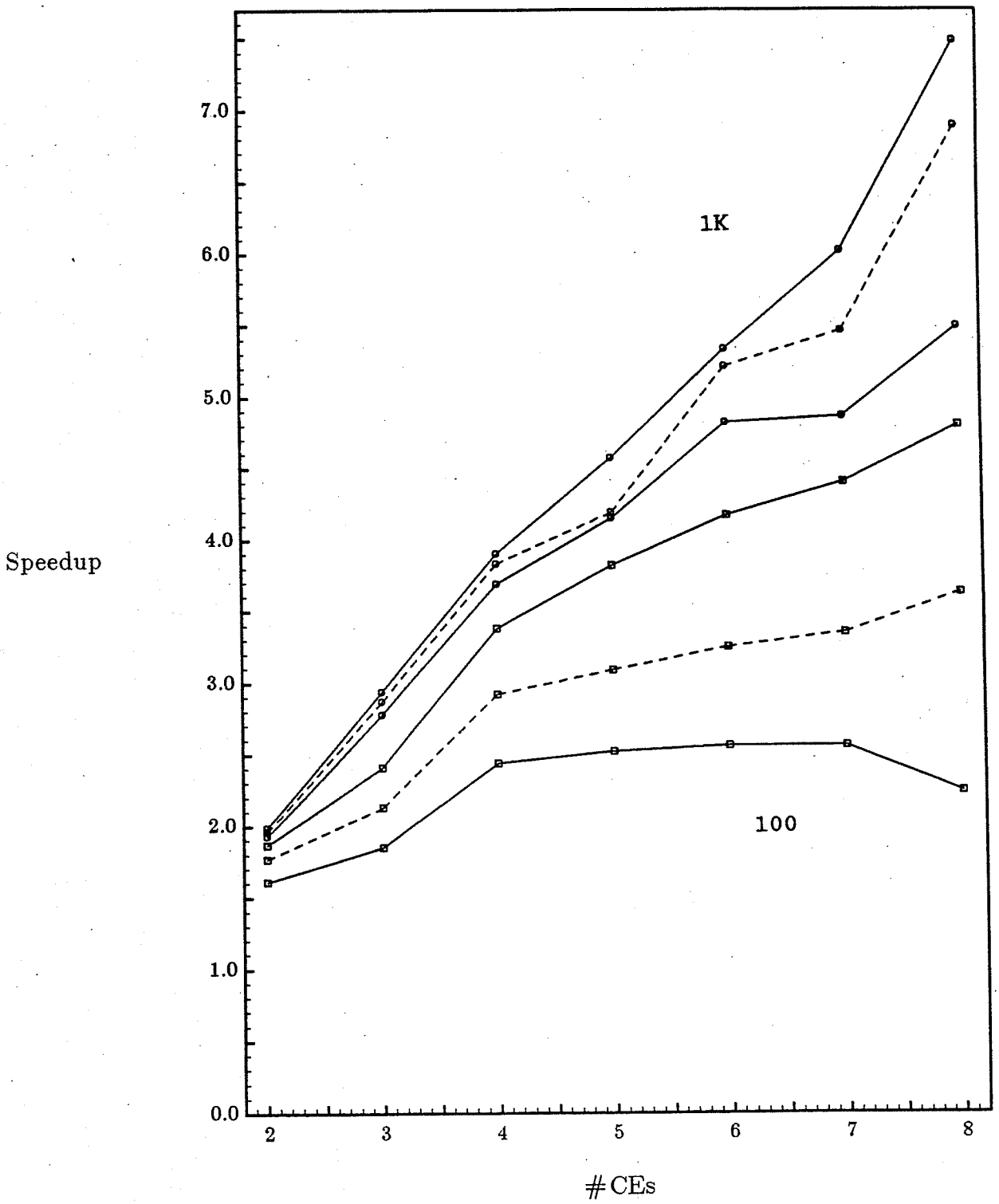
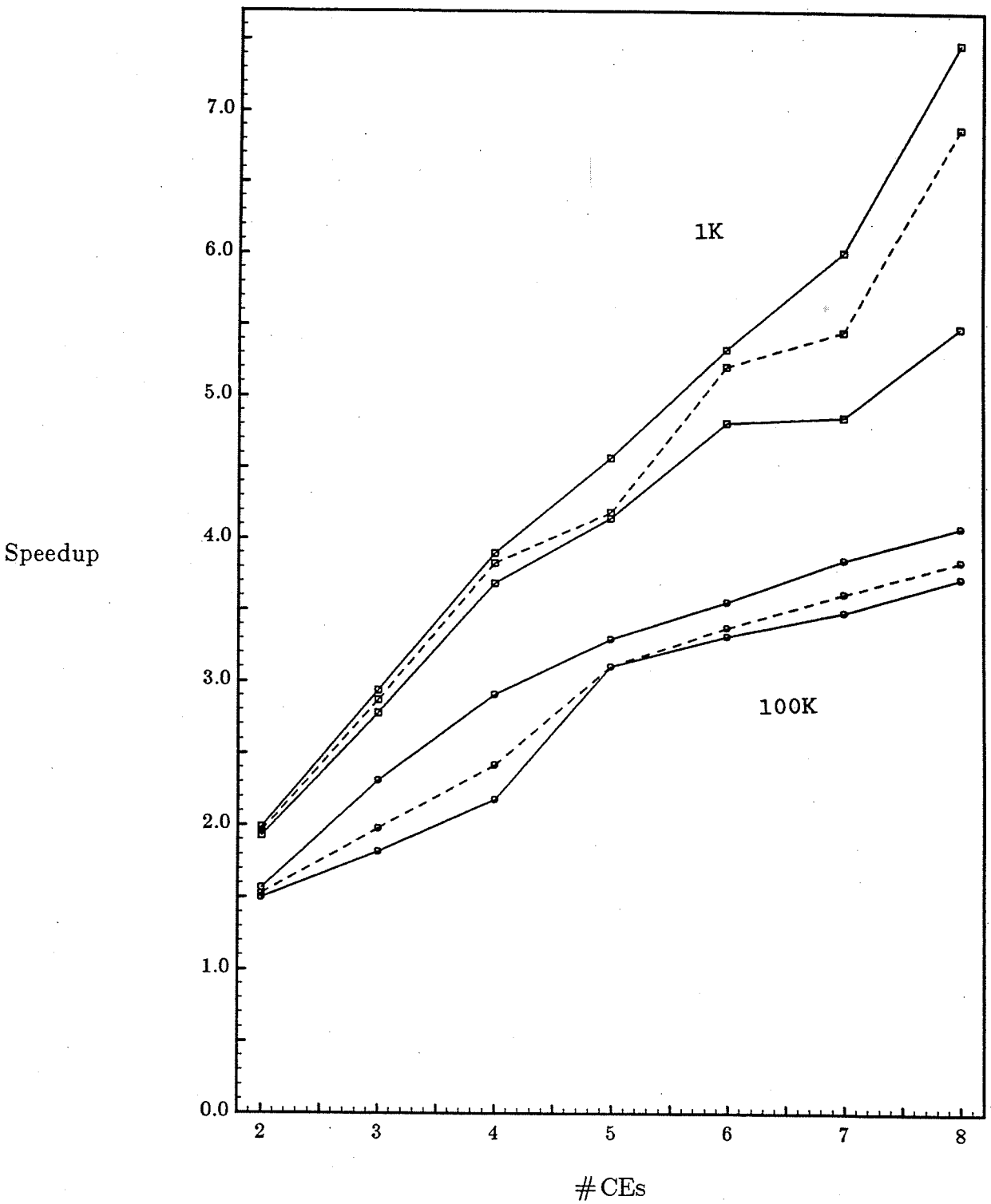


Figure 5-b.
Speedup vs. #CEs
for Vector Lengths 1K and 100K



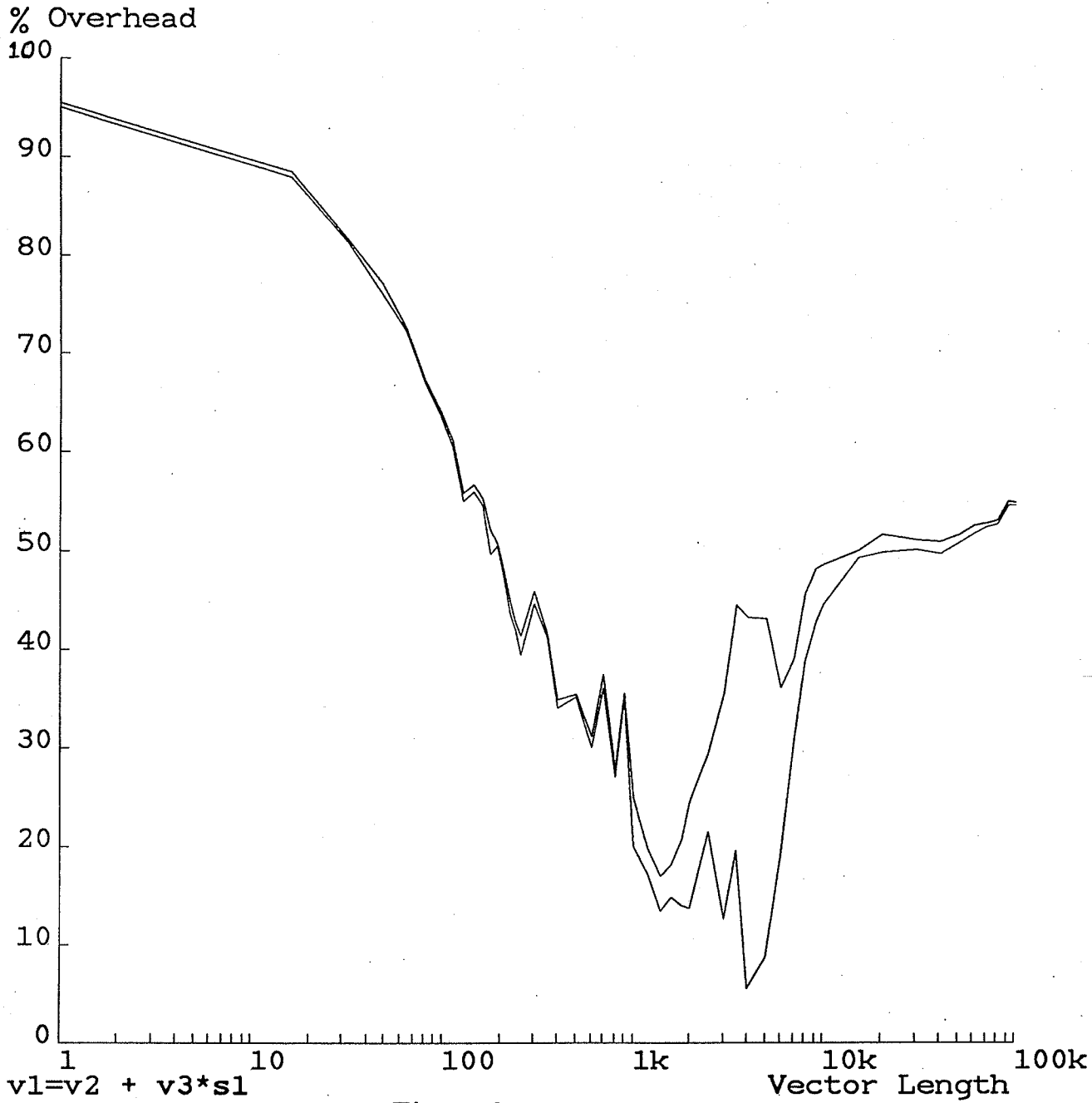


Figure 6
The Percentage Multiprocessing Overhead
Upper/Lower Bounds for Kernel *vpvts*

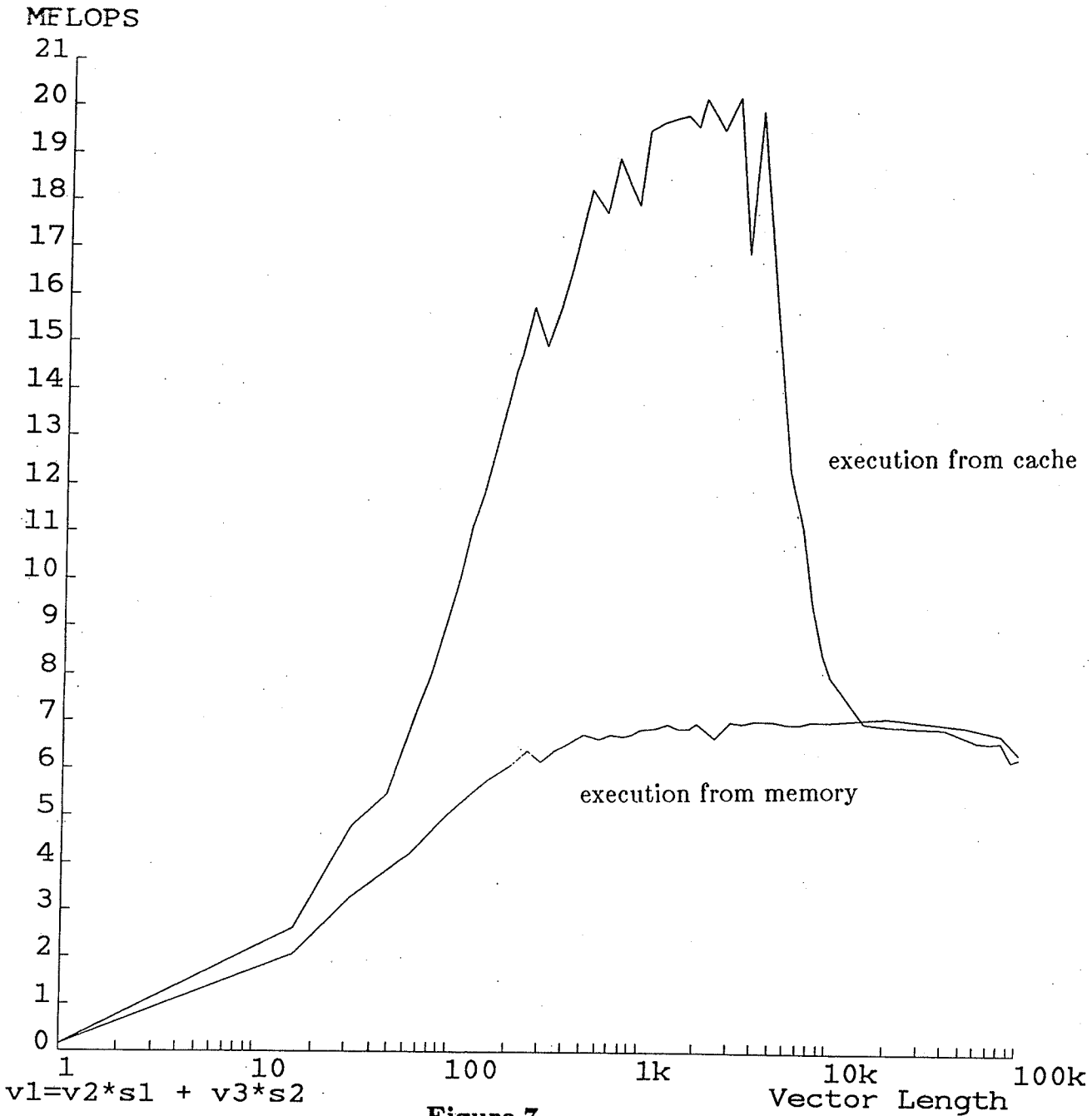


Figure 7
The Execution Rates from Memory and Cache
for the Kernel *vtspvts*

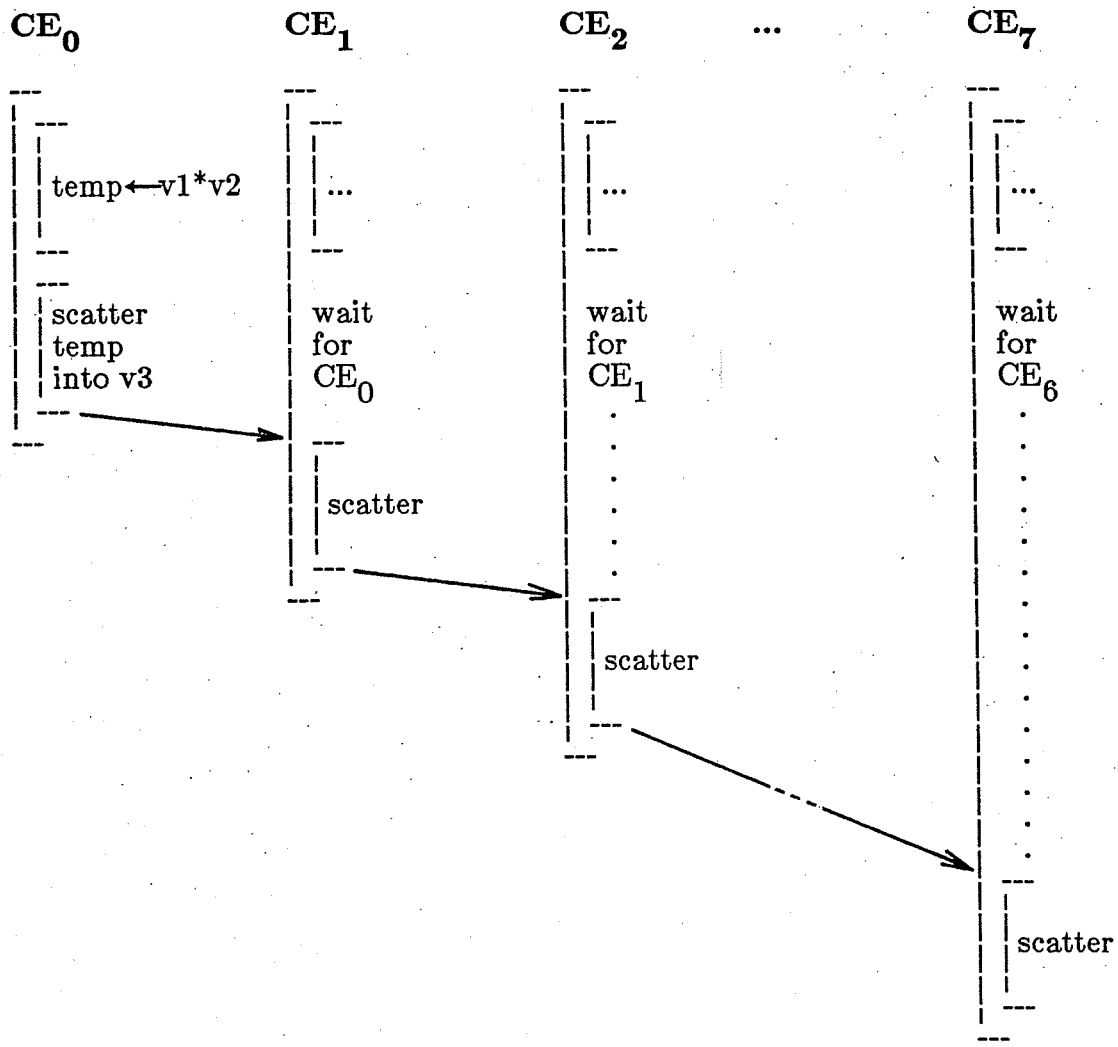


Figure 8.
Vector-Concurrent Execution of the
Kernel *v3(i) = v1 * v2*

Table 1
Peak Execution Rates of Elementary Vector Operations
(Millions of Floating Point Operations per Second, MFLOPS)

Benchmark	1 Computational Processor			8 Computational Processors		
	MFLOPS	Length	E_m (%)	MFLOPS	Length	E_m (%)
vps	2.21	32	37.5	13.65	4000	28.9
vts	1.60	32	27.1	11.21	4000	23.8
vpv	1.56	32	26.4	10.52	2000	22.3
vtv	1.23	32	20.8	8.58	2000	18.2
vtvps	2.37	32	40.2	16.23	2000	34.4
vpvts	2.37	32	40.2	16.08	3000	34.1
vtspvts	2.91	32	49.3	20.33	3000	43.1
vtvpv	1.94	32	32.9	13.03	2000	27.6
vtvpvtv	1.71	32	29.0	12.40	2000	26.3
vips	1.14	32	19.3	5.22	6000	11.1
vi=vtv	.80	32	13.6	.97	2500	2.1
vpvtvi	1.60	32	27.1	7.07	700	15.0
vi=vipvtv	.97	32	16.4	5.61	2500	11.9

Table 2
Execution Rates at Vector Length 100K
of Elementary Vector Operations
(Millions of Floating Point Operations per Second, MFLOPS)

Benchmark	1 Computational Processor		8 Computational Processors	
	MFLOPS	E_m (%)	MFLOPS	E_m (%)
vps	0.98	16.6	4.06	8.6
vts	0.90	15.3	3.36	7.1
vpv	0.74	12.3	2.87	6.1
vtv	0.66	11.2	2.44	5.2
vtvps	1.29	21.9	4.80	10.2
vpvts	1.29	21.9	4.65	9.9
vtspvts	1.74	29.5	6.30	13.3
vtvpv	1.02	17.3	3.79	8.0
vtvpvtv	1.01	17.1	4.09	8.7
vips	0.80	13.5	3.49	7.4
vi=vtv	0.52	8.8	0.60	1.3
vpvtvi	0.95	16.1	3.37	7.1
vi=vipvtv	0.72	12.2	3.38	7.2

Table 3
$n_{1/2}$ and the Percentage of Peak Execution Rate at that Vector Length for 1 Computational Processor

Benchmark	1 Computational Processor	
	$n_{1/2}$*	% of Peak
vps	3	30%
vts	3	35%
vpv	3	33%
vtv	3	36%
vtvps	3	34%
vpvts	3	33%
vtspvts	3	35%
vtvpv	2	25%
vtvpvtv	2	31%
vips	3	31%
vi=vtv	4	36%
vpvtvi	3	33%
vi=vipvtv	4	40%

* $n_{1/2}$ is calculated using a linear least-squares approximation of the vector execution times for vector lengths between 1 and 256.

Table 4
Maximum Upper Bound Speedups
for Elementary Vector Operations

Benchmark	Vector Length	Maximum Upper Bound Speedup	Lower Bound
vps	4000	6.67	5.13
vts	7000	8.15	5.27
vpv	2000	9.68	4.17
vtv	2000	7.22	5.73
vtvps	1400	8.05	6.04
vpvts	4000	7.56	4.54
vtspvts	2000	7.86	7.05
vtvpv	3000	7.84	4.90
vtvpvtv	2000	8.20	5.42
vips	7000	5.45	4.44
vi=vtv	3500	1.32	1.10
vpvtvi	500	6.08	4.53
vi=vipvtv	2500	5.86	5.20

Table 5
Lower/Upper Bound on Speedups of Elementary Vector Operations at Vector Lengths where Peak Execution Rates Occur

Benchmark	Vector Length	Speedup Bounds
vps	4000	5.13 / 6.67
vts	4000	5.88 / 7.36
vpv	2000	4.17 / 9.68
vtv	2000	5.73 / 7.22
vtvps	2000	5.60 / 7.08
vpvts	3000	5.15 / 6.99
vtspvts	3000	6.04 / 7.21
vtvpv	2000	6.14 / 7.07
vtvpvtv	2000	5.42 / 8.20
vips	6000	4.15 / 4.83
vi=vtv	2500	1.08 / 1.24
vpvtvi	700	4.32 / 4.58
vi=vipvtv	2500	5.20 / 5.86

Table 6
Minimum Upper Bound Multiprocessing Overheads for Elementary Vector Operations

Benchmark	Vector Length	Minimum Upper Bound Overhead	Lower Bound
vps	1400	25.0%	22.5%
vts	3000	10.3%	6.0%
vpv	1400	16.0%	13.3%
vtv	1800	13.7%	11.6%
vtvps	1000	21.8%	20.9%
vpvts	1400	16.9%	13.4%
vtspvts	2000	11.9%	1.7%
vtvpv	1000	14.4%	12.6%
vtvpvtv	1200	8.6%	7.2%
vips	3000	41.4%	40.9%
vi=vtv	1200	84.8%	84.5%
vpvtvi	500	43.4%	24.0%
vi=vipvtv	700	29.2%	27.3%

Table 7
Degradation of the Peak Execution Rates of Elementary Vector Operations when Executing from Memory instead of the Cache

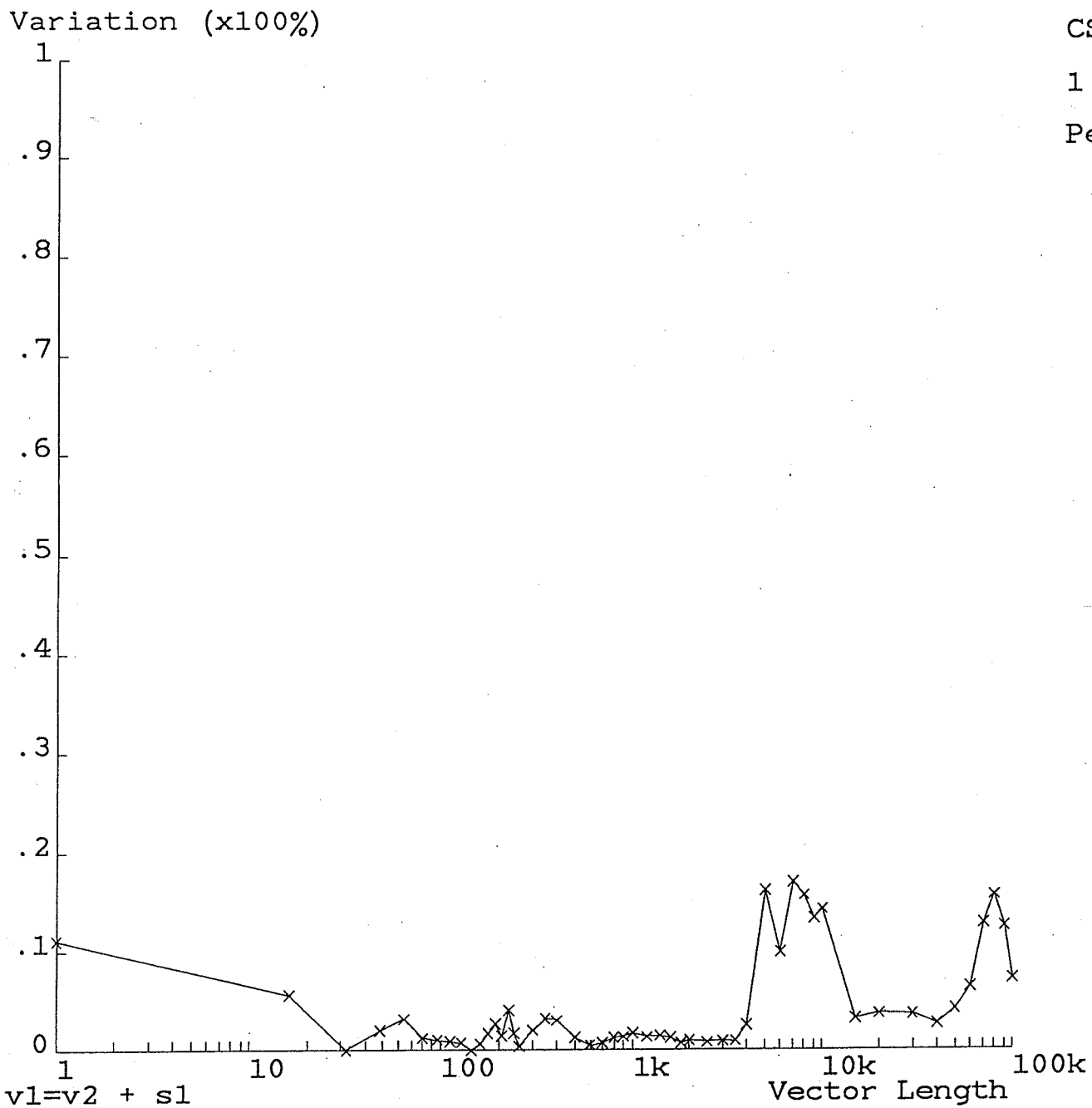
Benchmark	1 Computational Processor	8 Computational Processors
	Degradation Factor *	Degradation Factor *
vps	2.26	3.36
vts	1.78	3.34
vpv	2.13	3.67
vtv	1.86	3.52
vtvps	1.85	3.33
vpvts	1.85	3.45
vtspvts	1.67	3.23
vtvpv	1.89	3.45
vtvpvtv	1.69	3.03
vips	1.43	1.49
vi=vtv	1.54	1.61
vpvtvi	1.69	2.08
vi=vipvtv	1.35	1.67

* degradation factor is calculated as the peak execution rate divided by the execution rate at 100K

Appendix A

The Percentage Variation between the
Maximum and Minimum Rate Runs

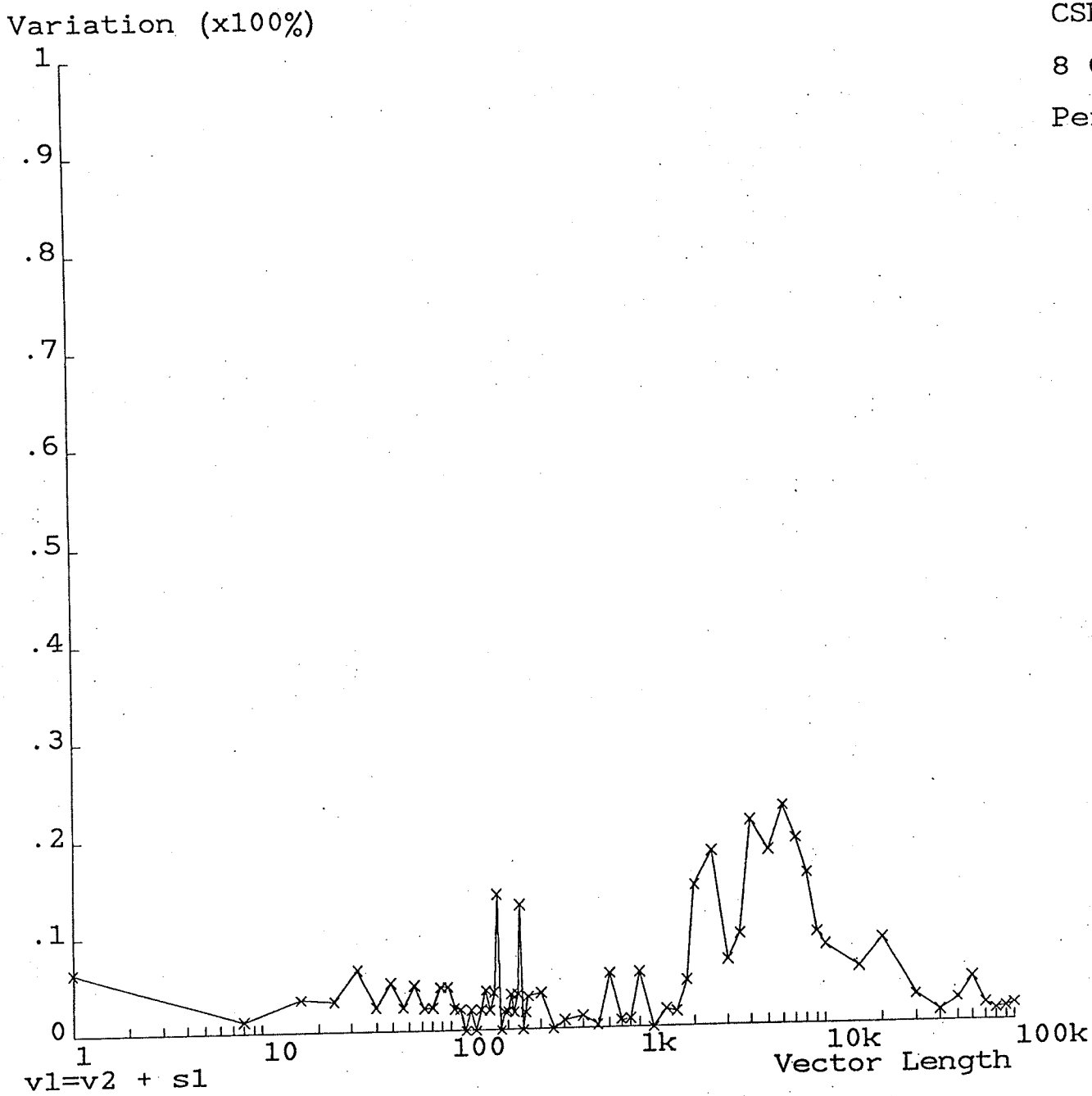
The following plots show the percentage variation for 1 and 8 CEs for the kernels not presented in Figure 3. For a particular kernel, the plot for 1 CE is shown first followed by the 8 CEs plot. The vector lengths range from 1 to 100,000.



CSRD

1 CE

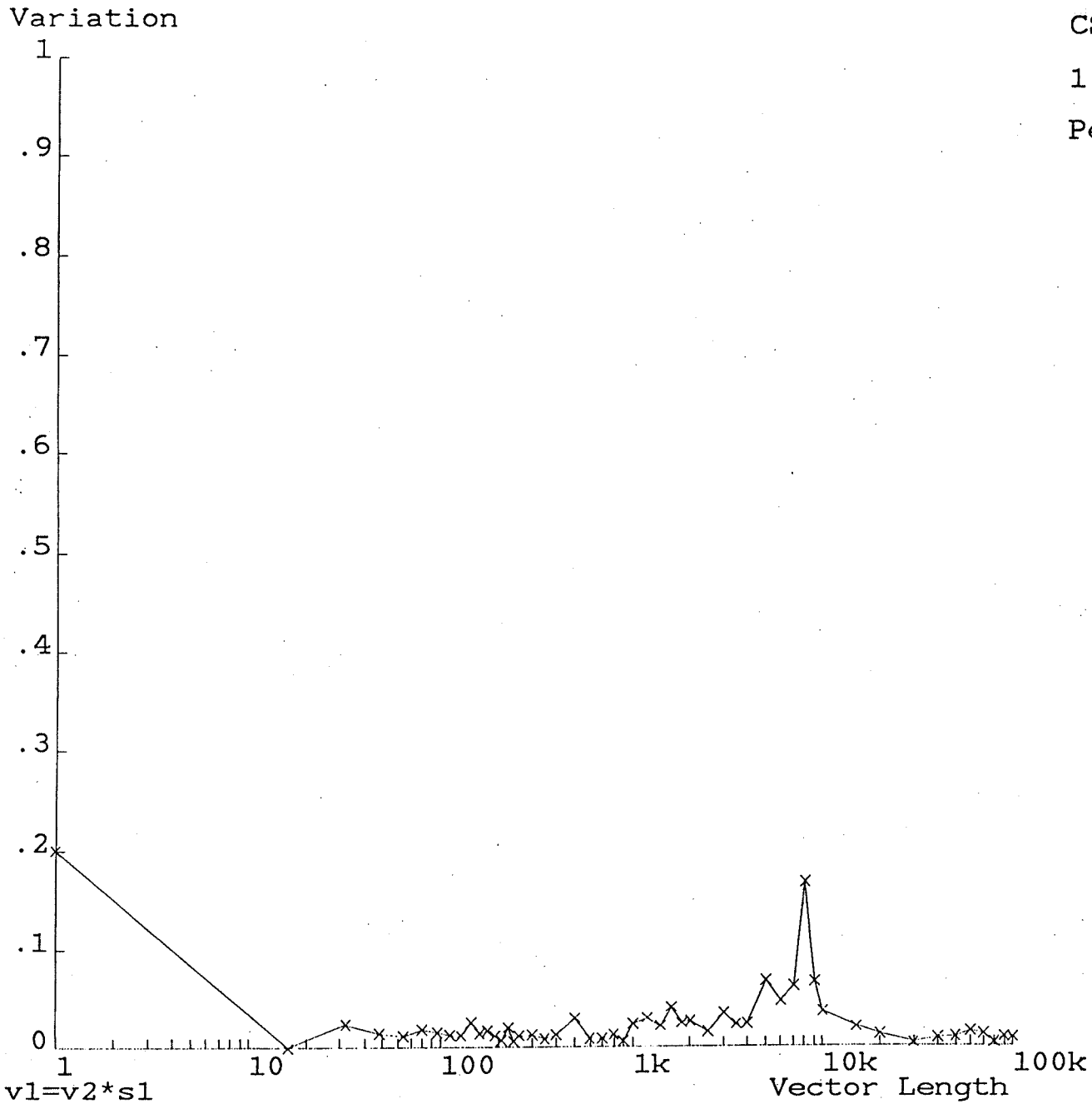
Percent Variation



CSRD

8 CEs

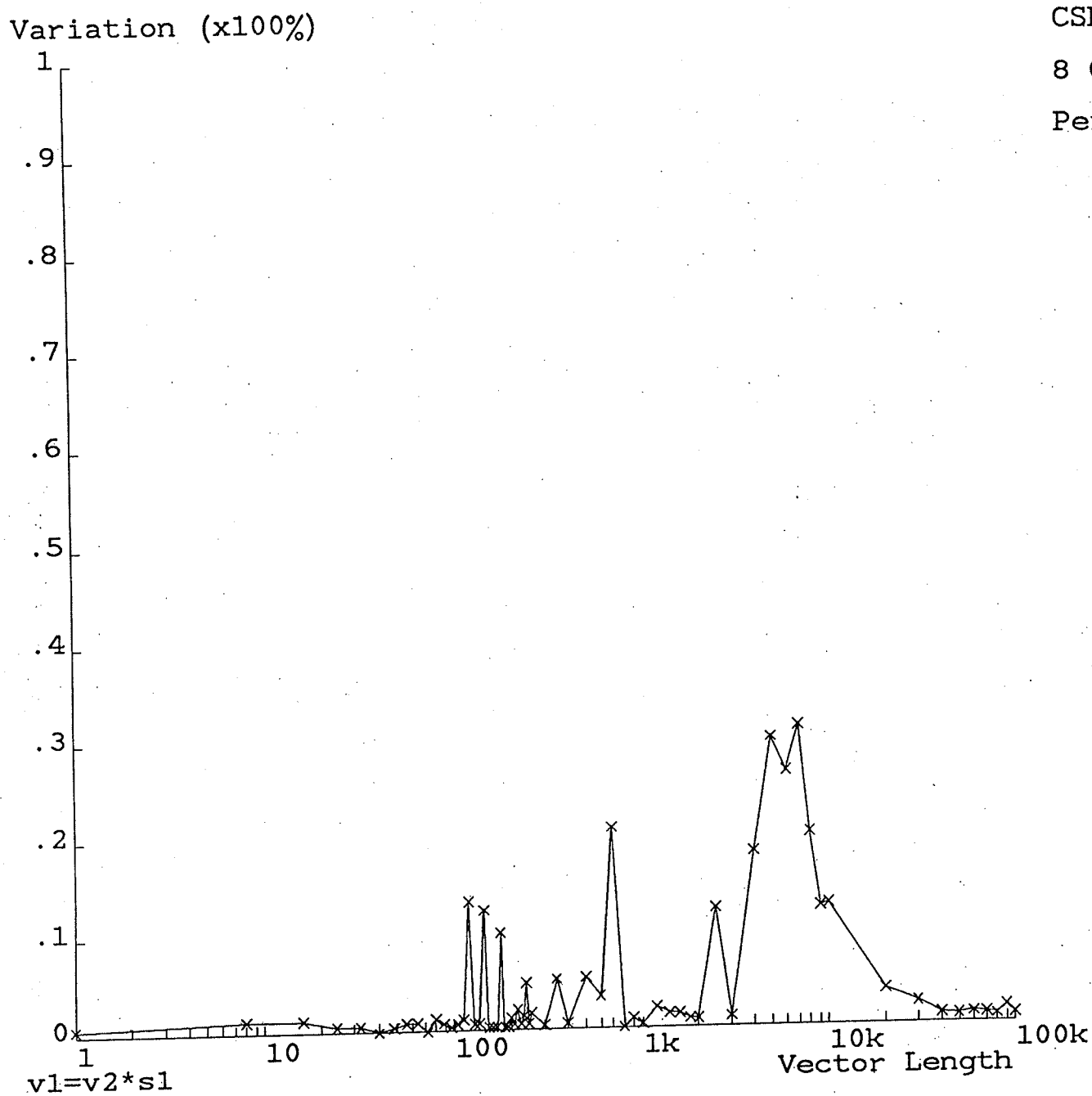
Percent Variation



CSRD

1 CE

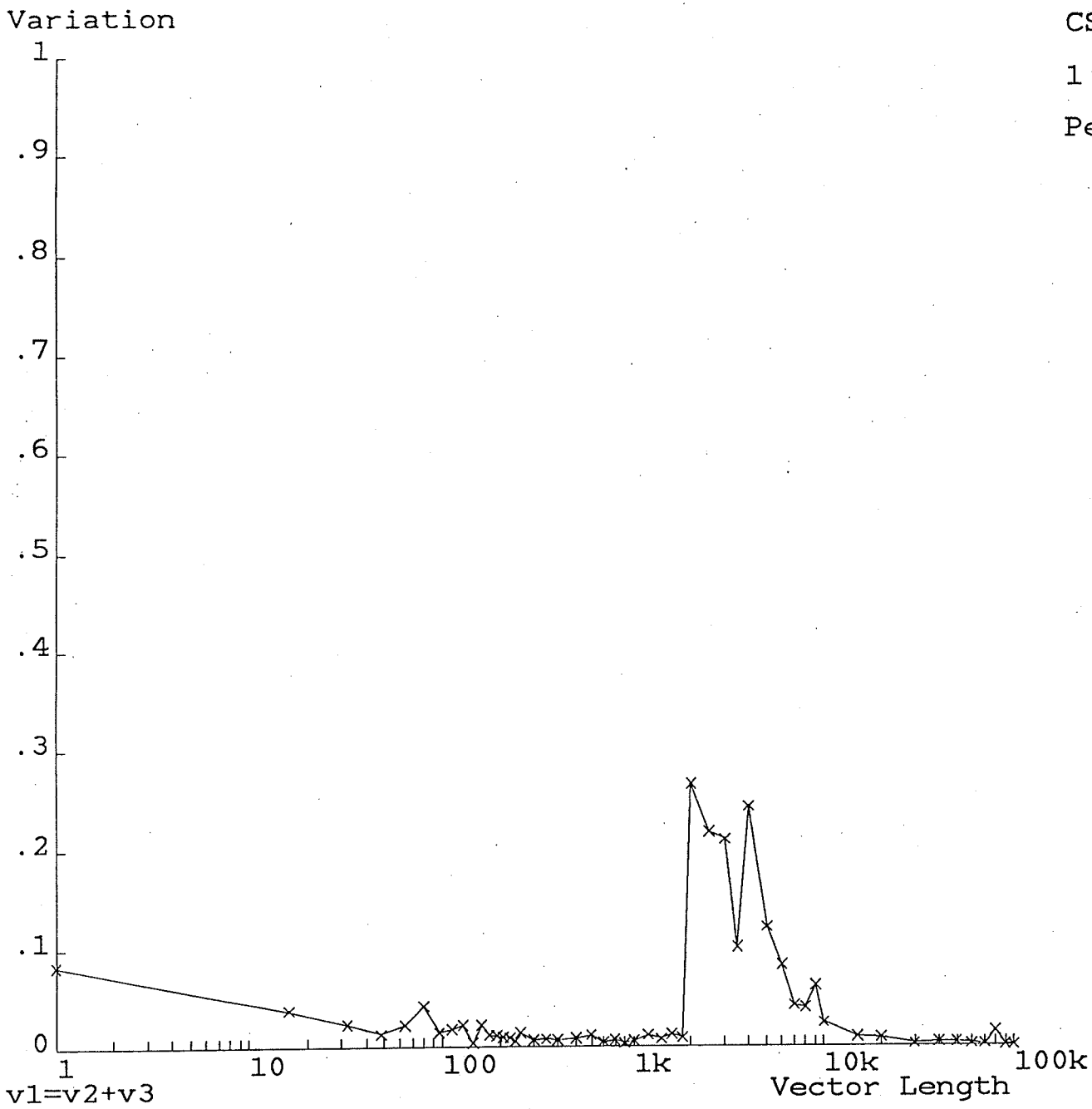
Percent Variation



CSRD

8 CEs

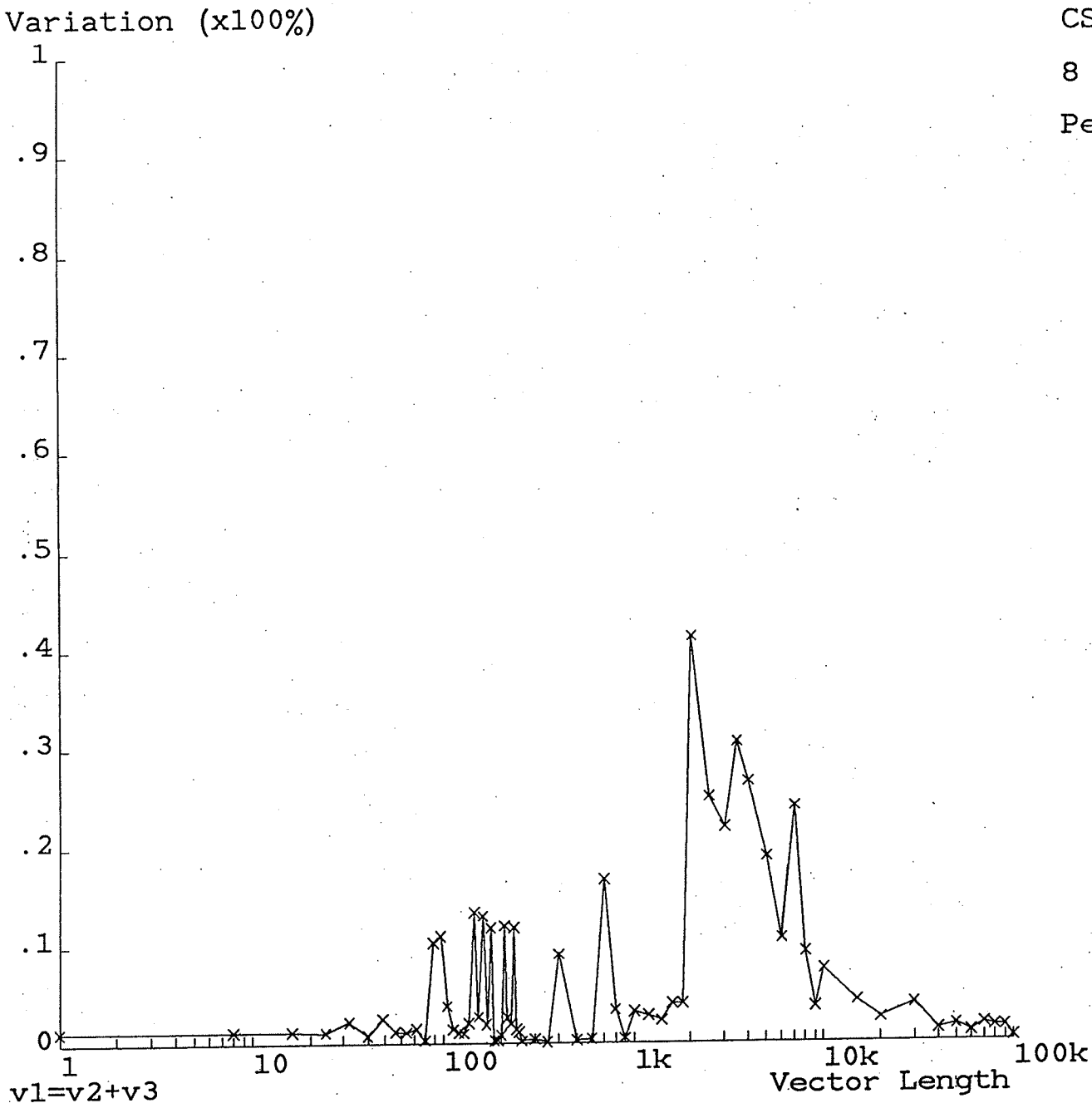
Percent Variation



CSRD

1 CE

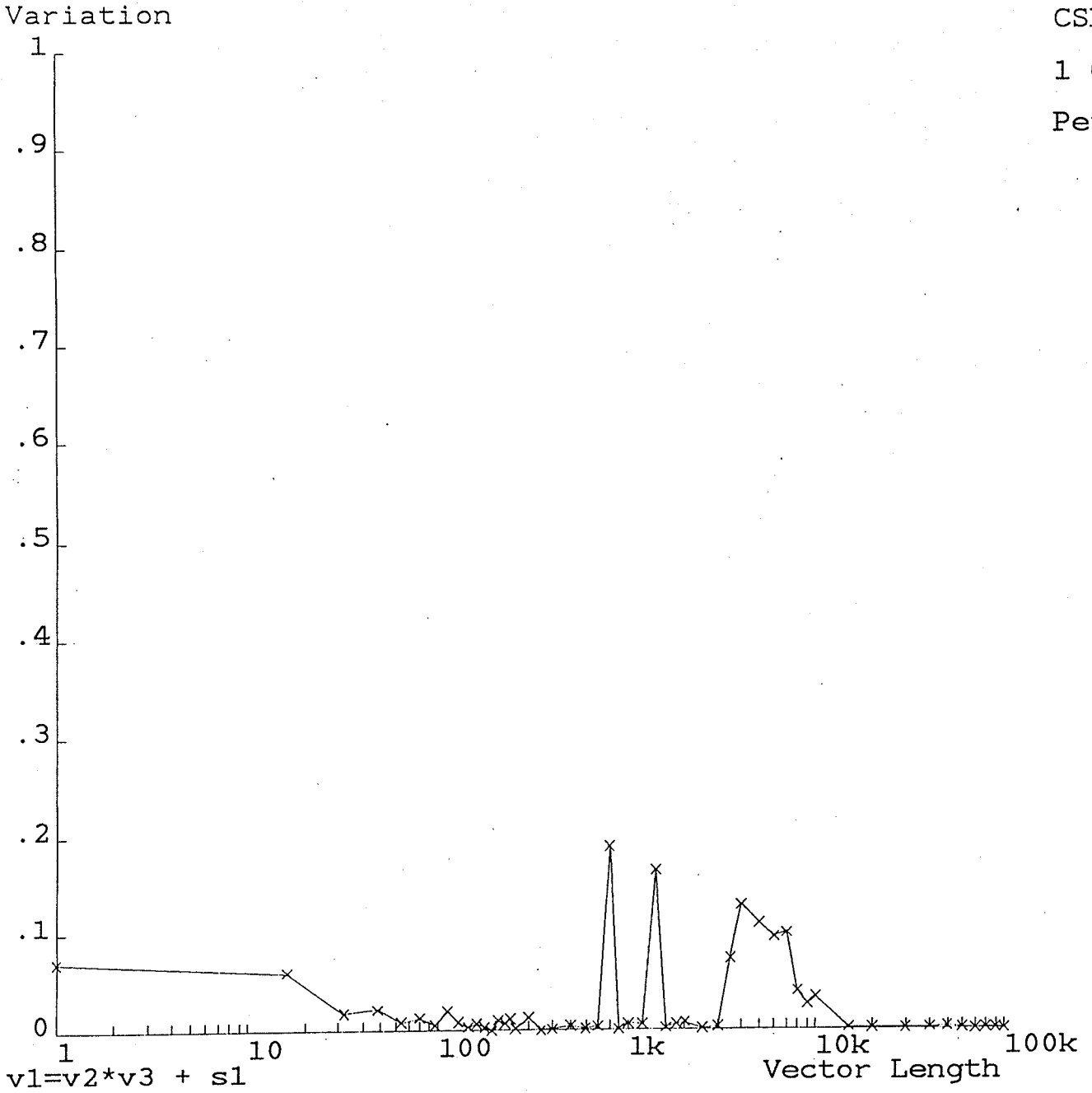
Percent Variation



CSRD

8 CEs

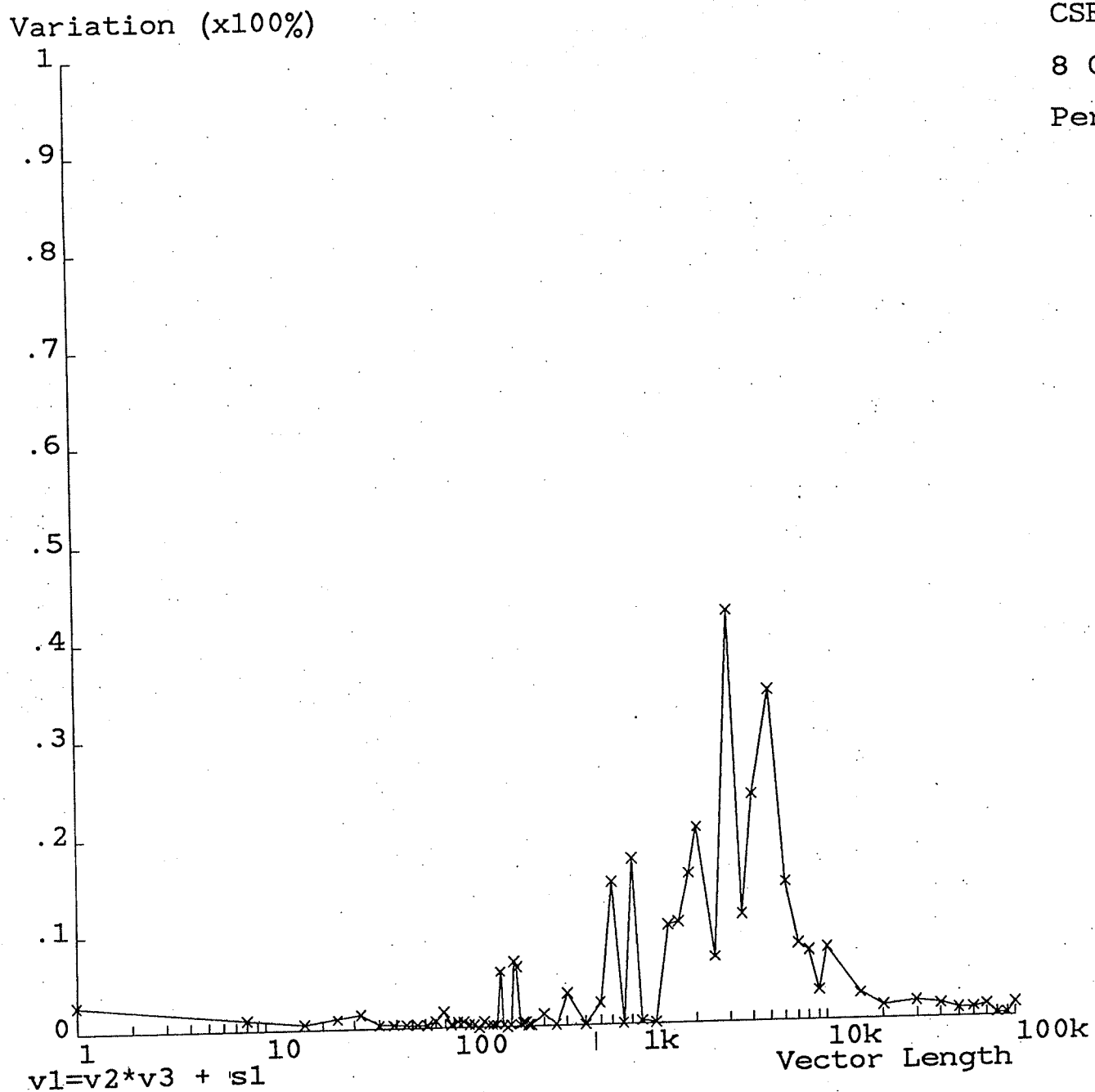
Percent Variation



CSRD

1 CE

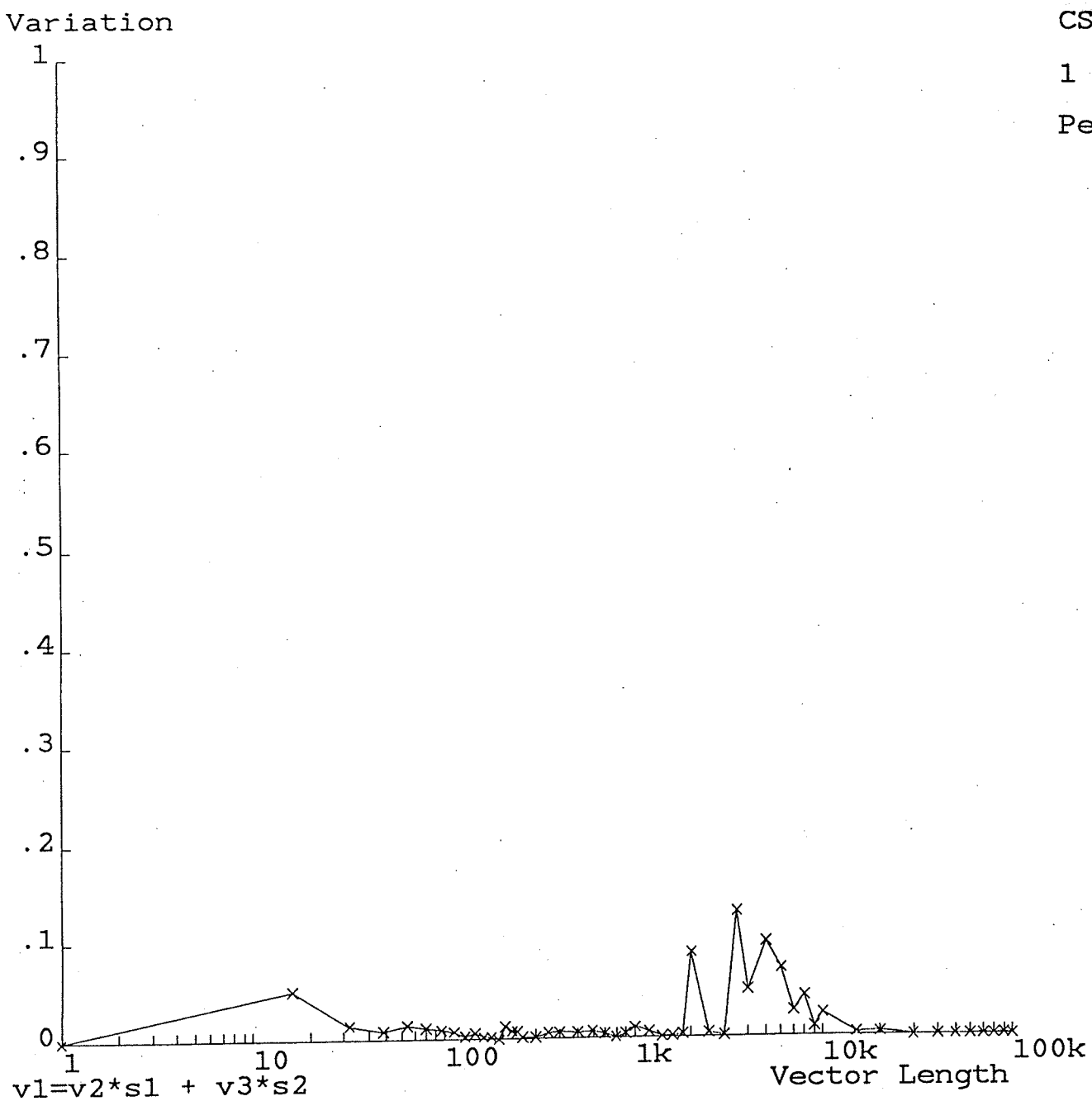
Percent Variation



CSRD

8 CEs

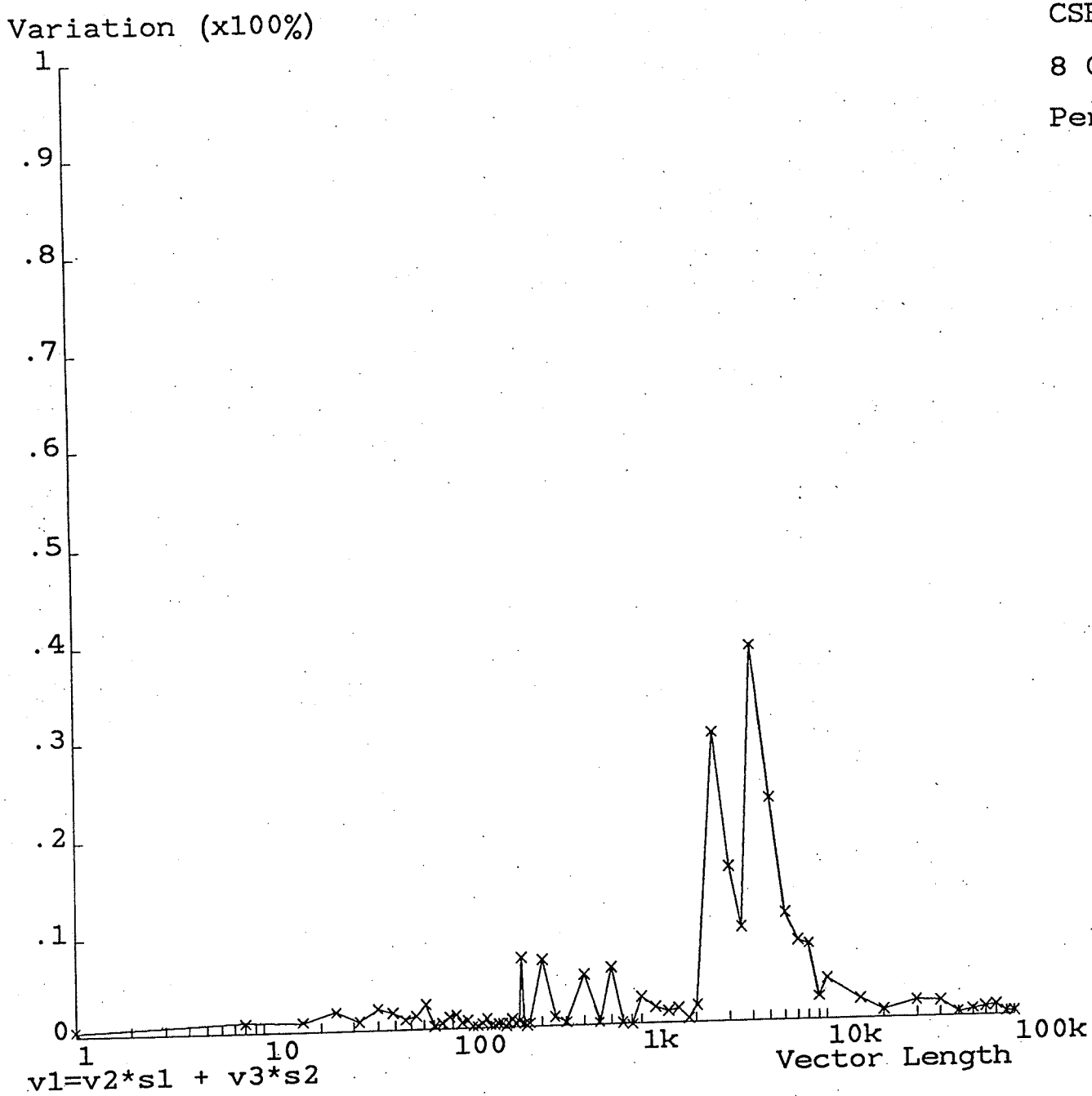
Percent Variation



CSRD

1 CE

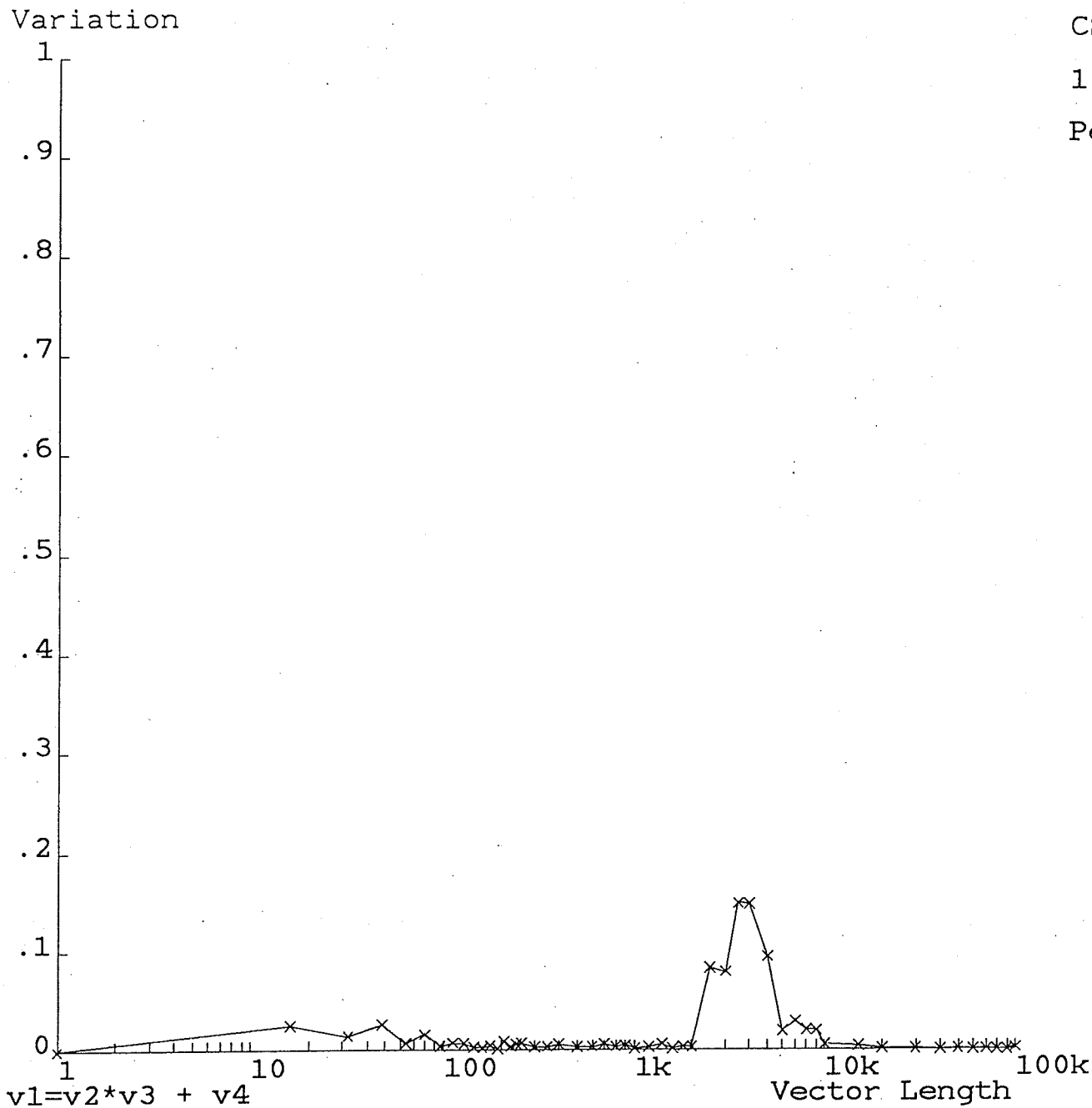
Percent Variation



CSRD

8 CEs

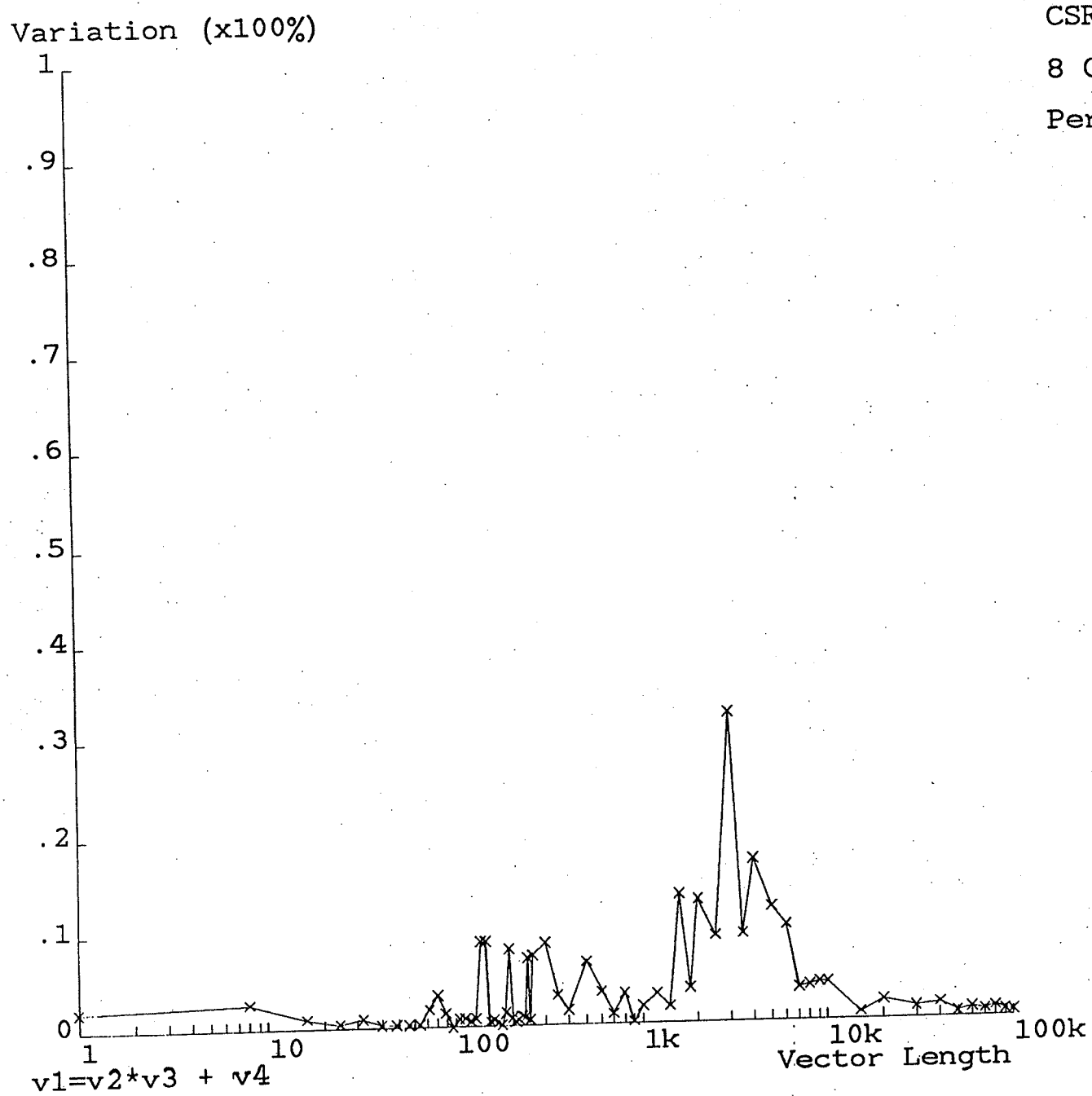
Percent Variation



CSRD

1 CE

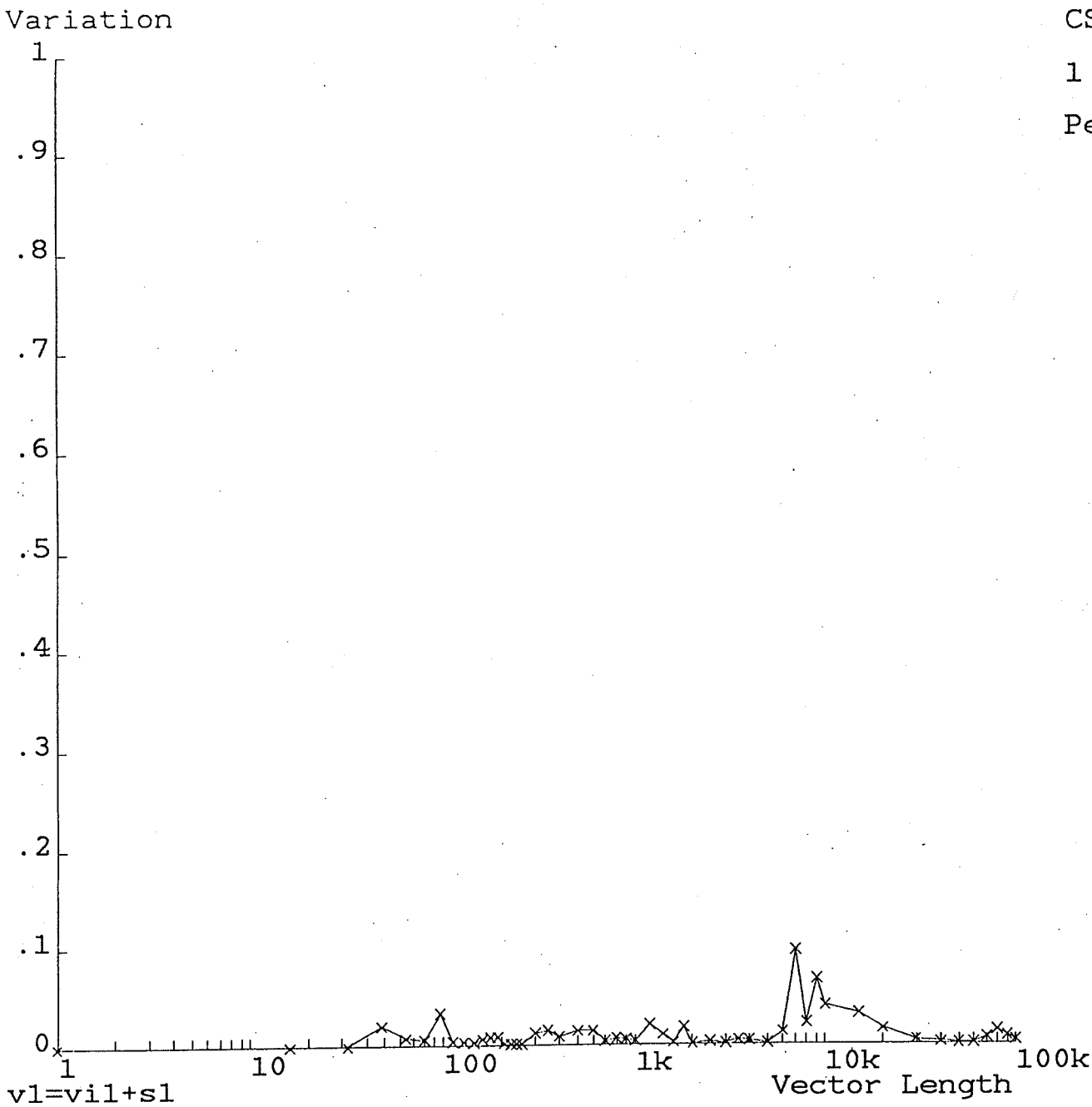
Percent Variation



CSRD

8 CEs

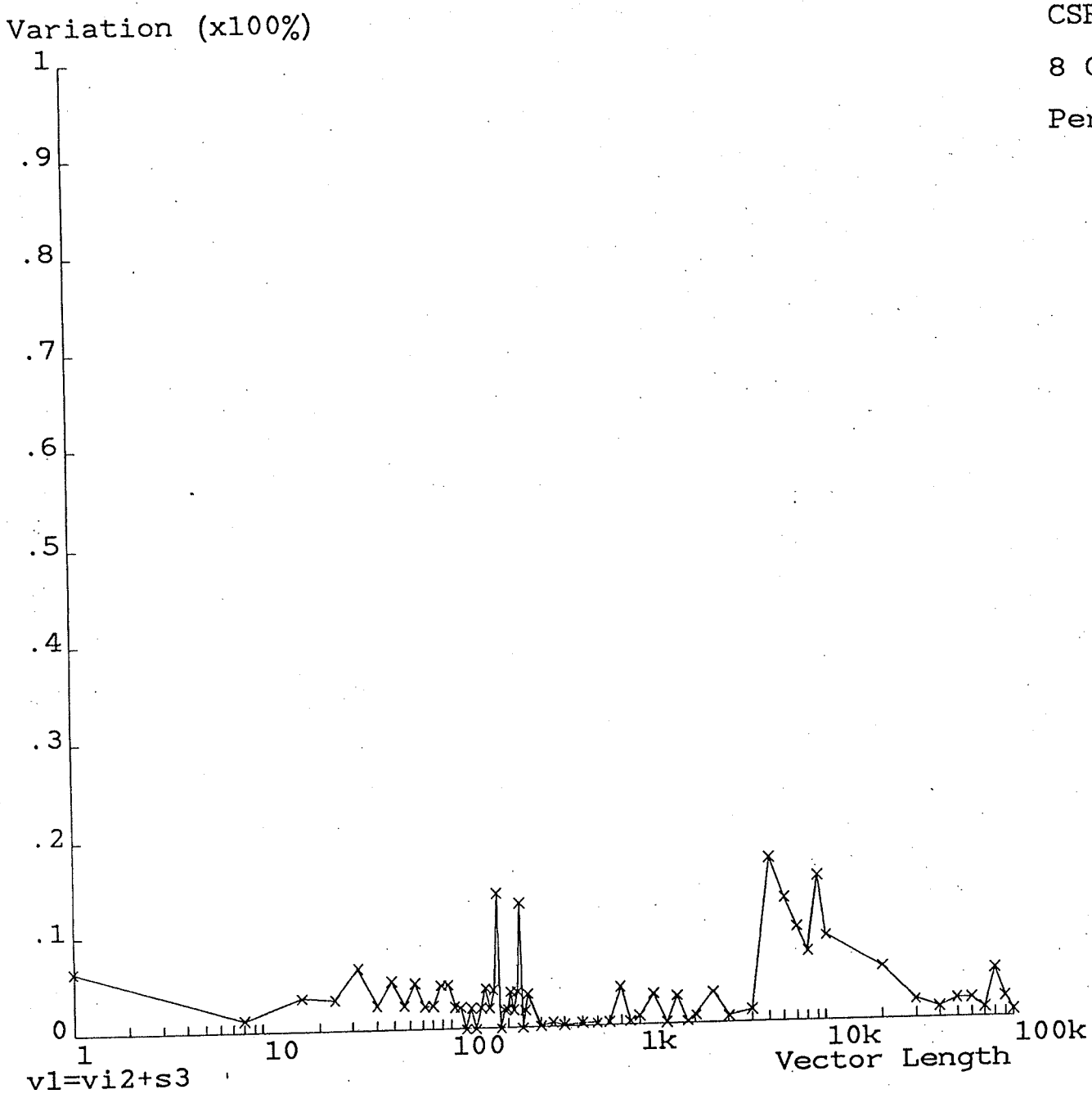
Percent Variation



CSRD

1 CE

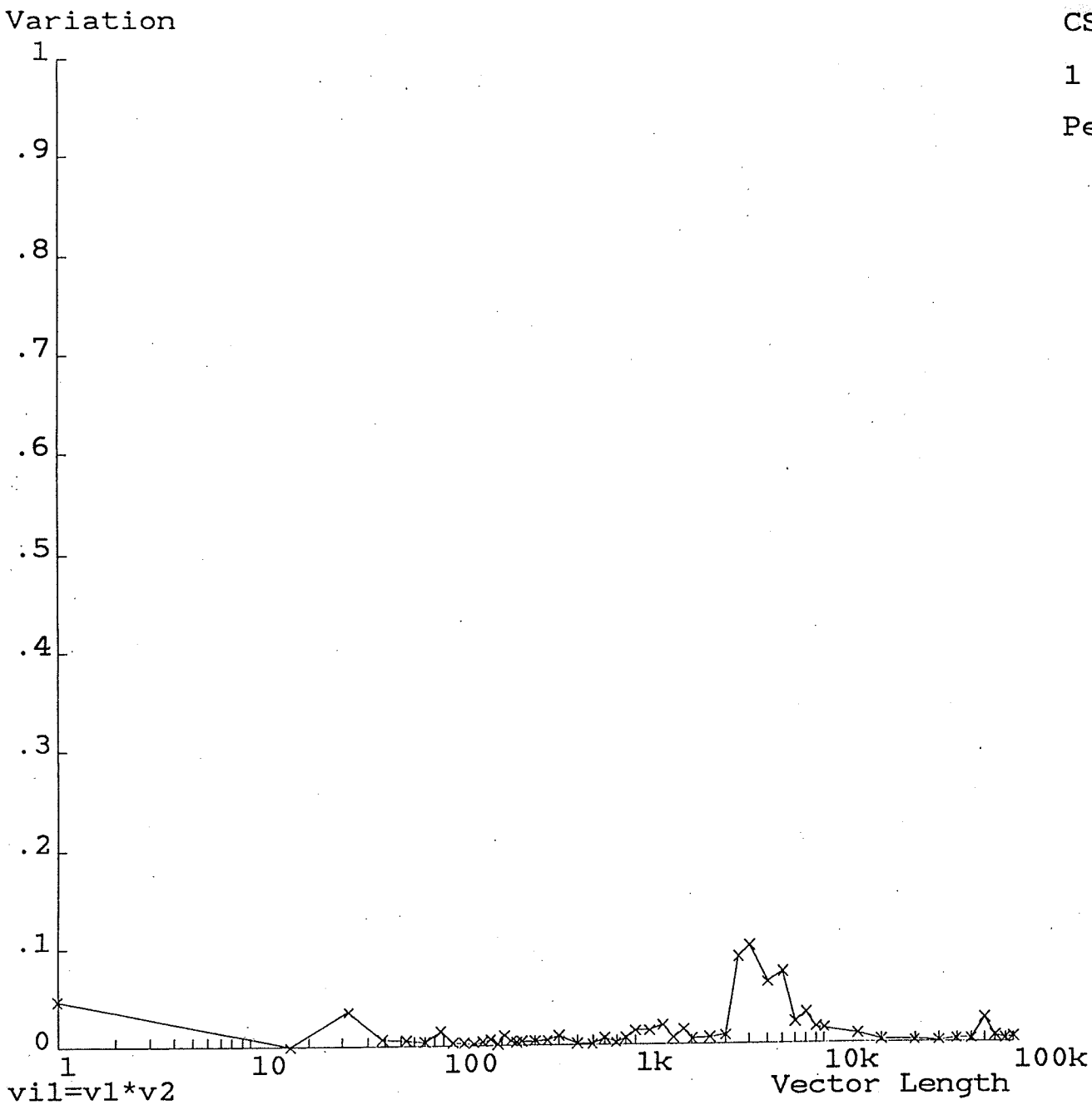
Percent Variation



CSRD

8 CEs

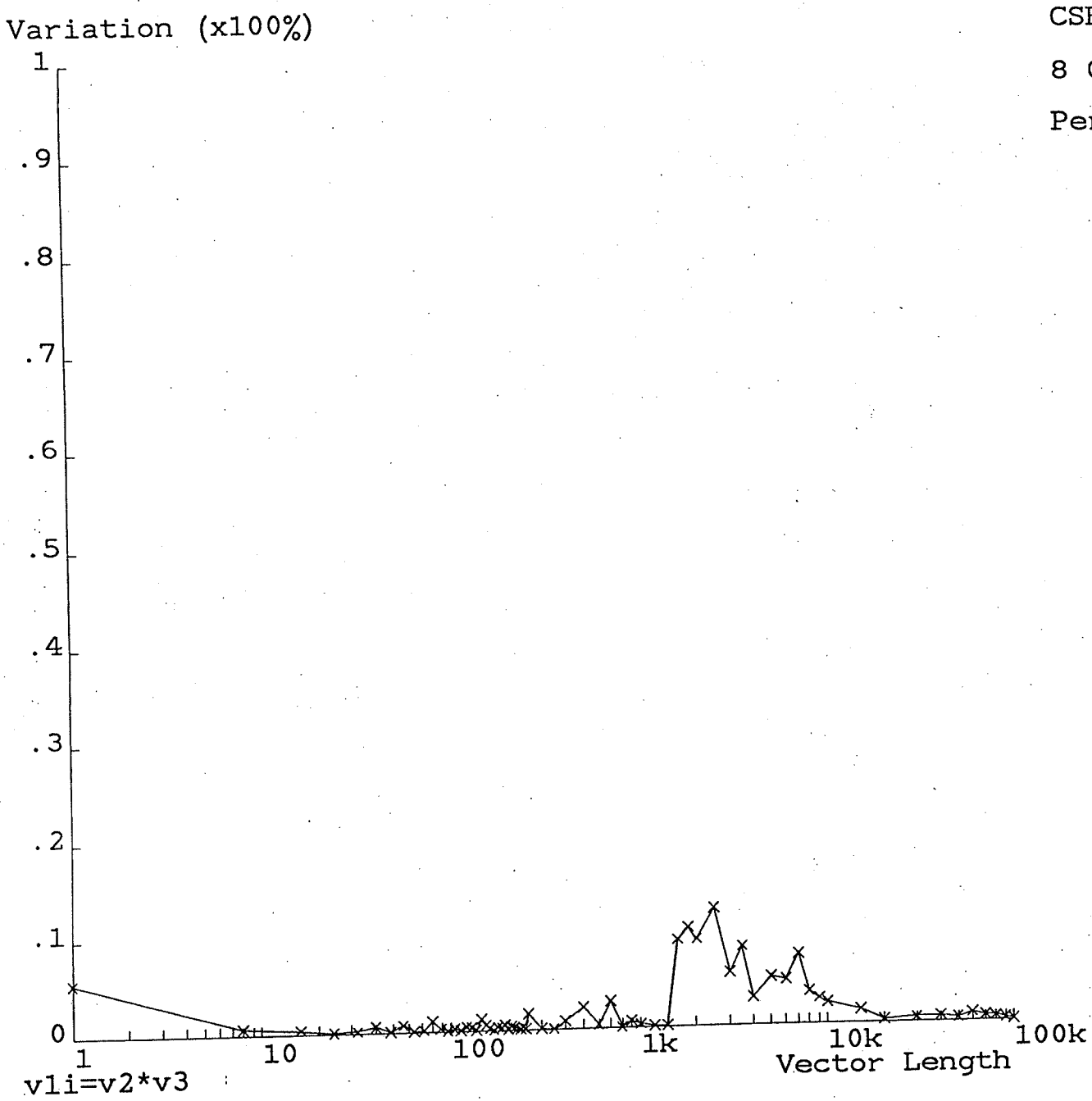
Percent Variation



CSRD

1 CE

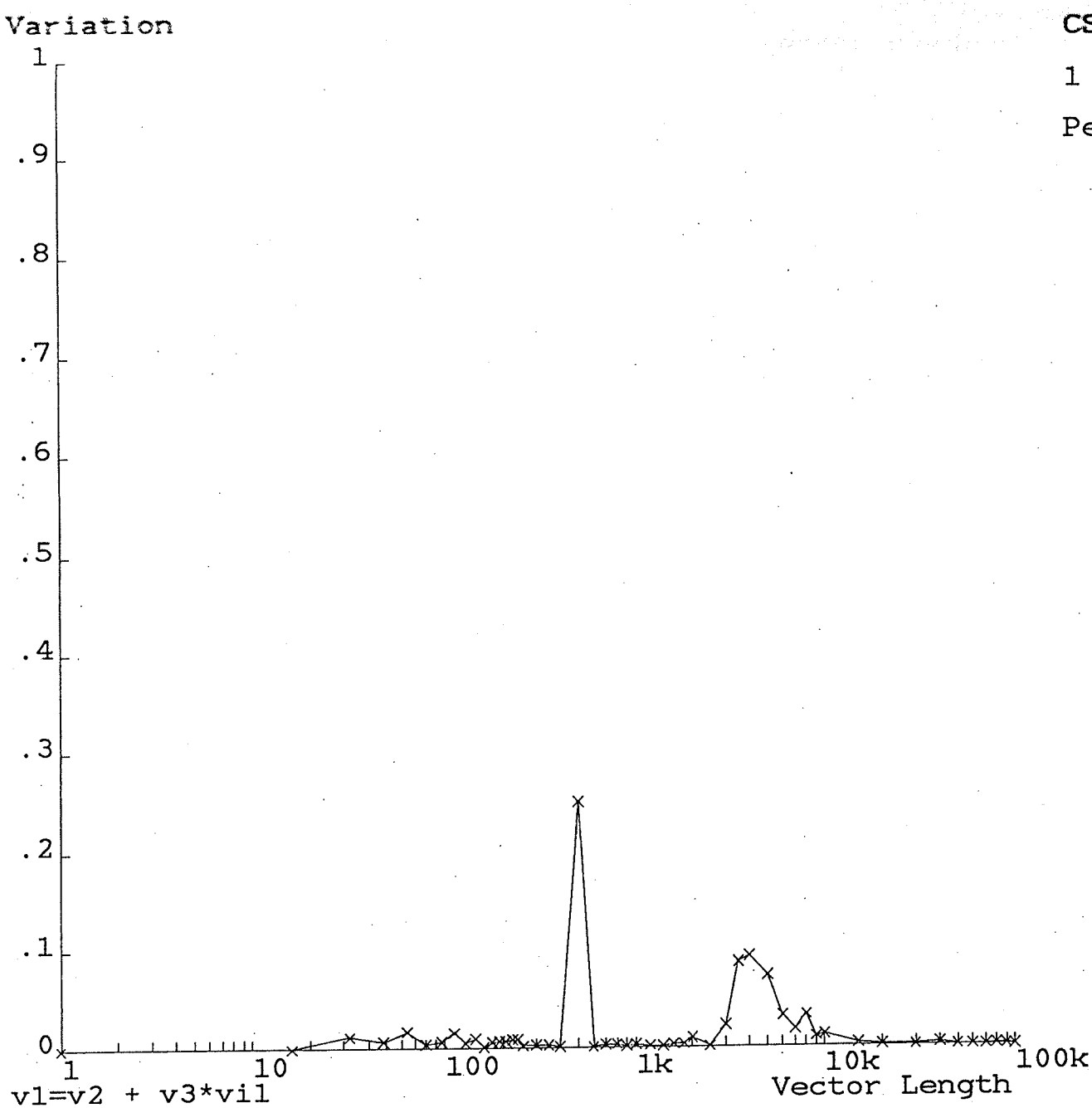
Percent Variation



CSRD

8 CEs

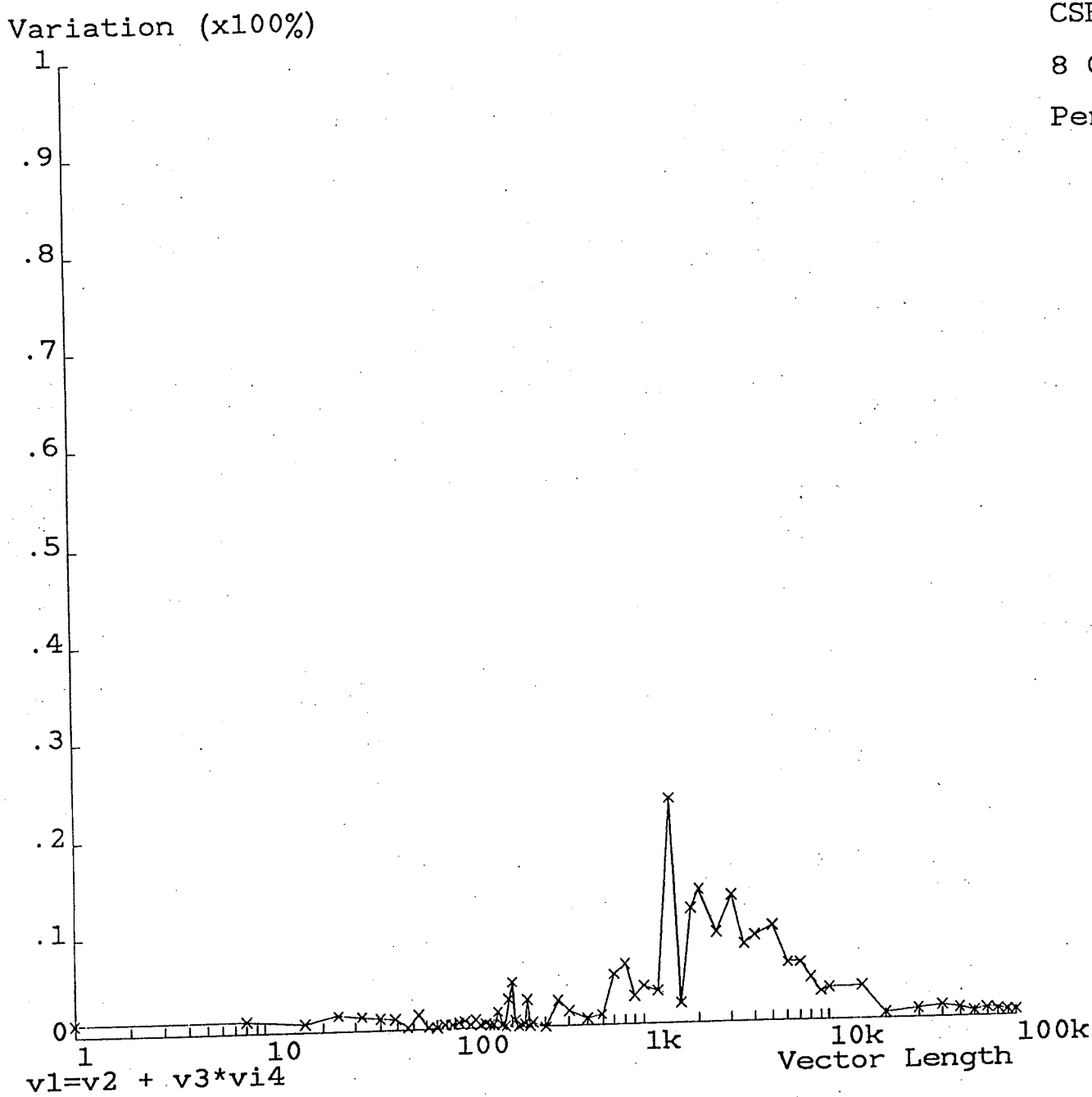
Percent Variation



CSRD

1 CE

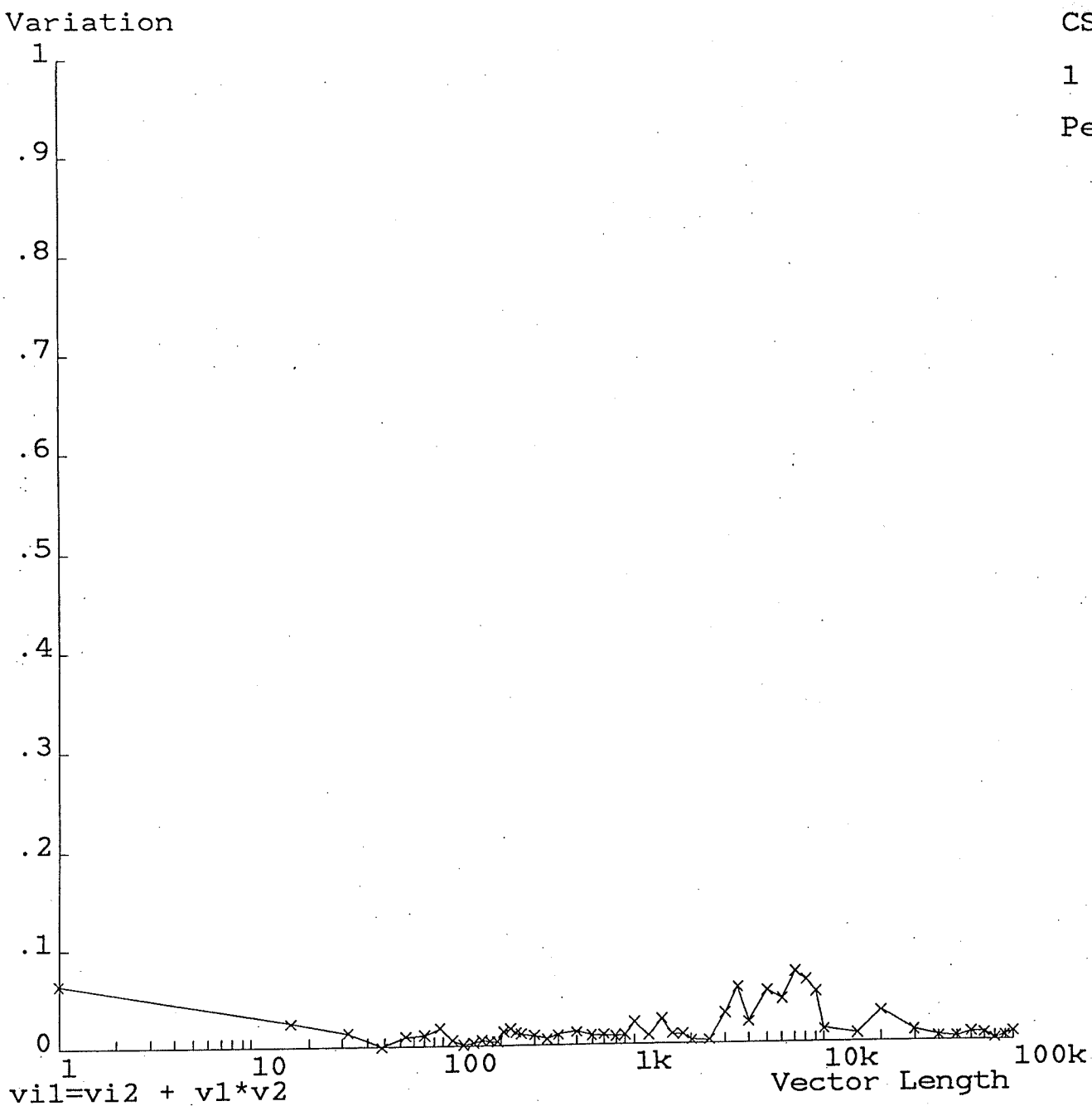
Percent Variation



CSRD

8 CEs

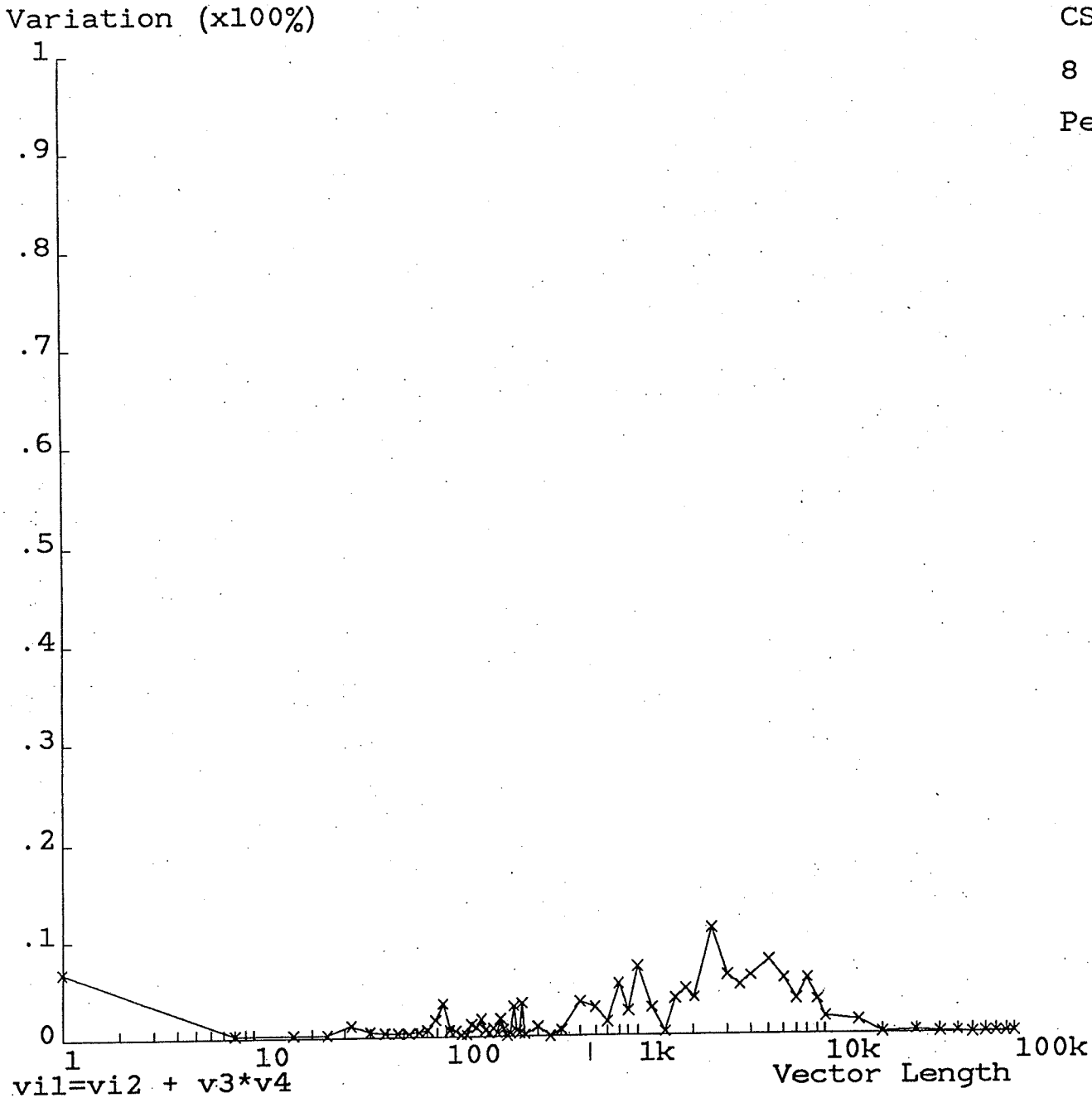
Percent Variation



CSRD

1 CE

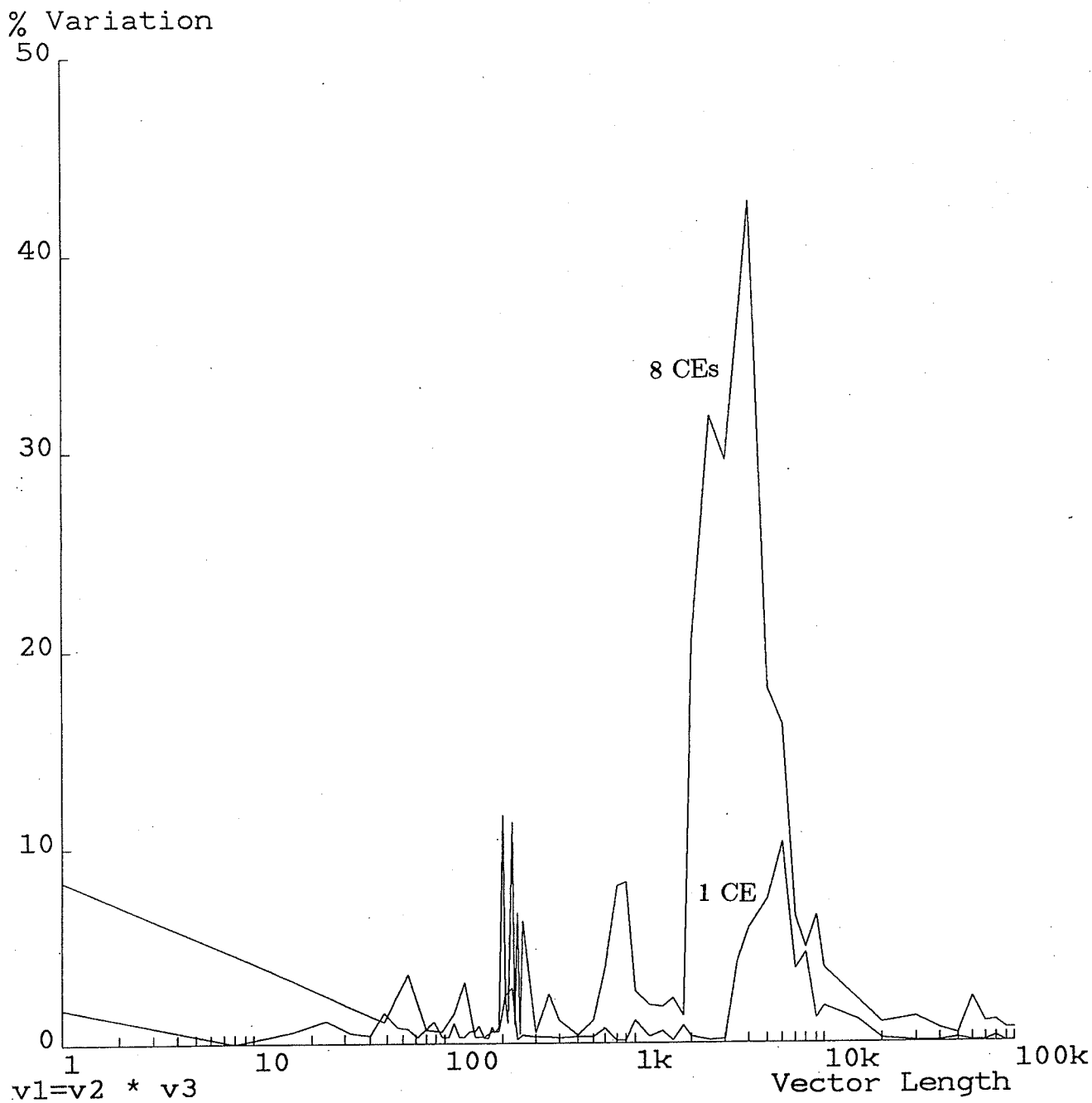
Percent Variation

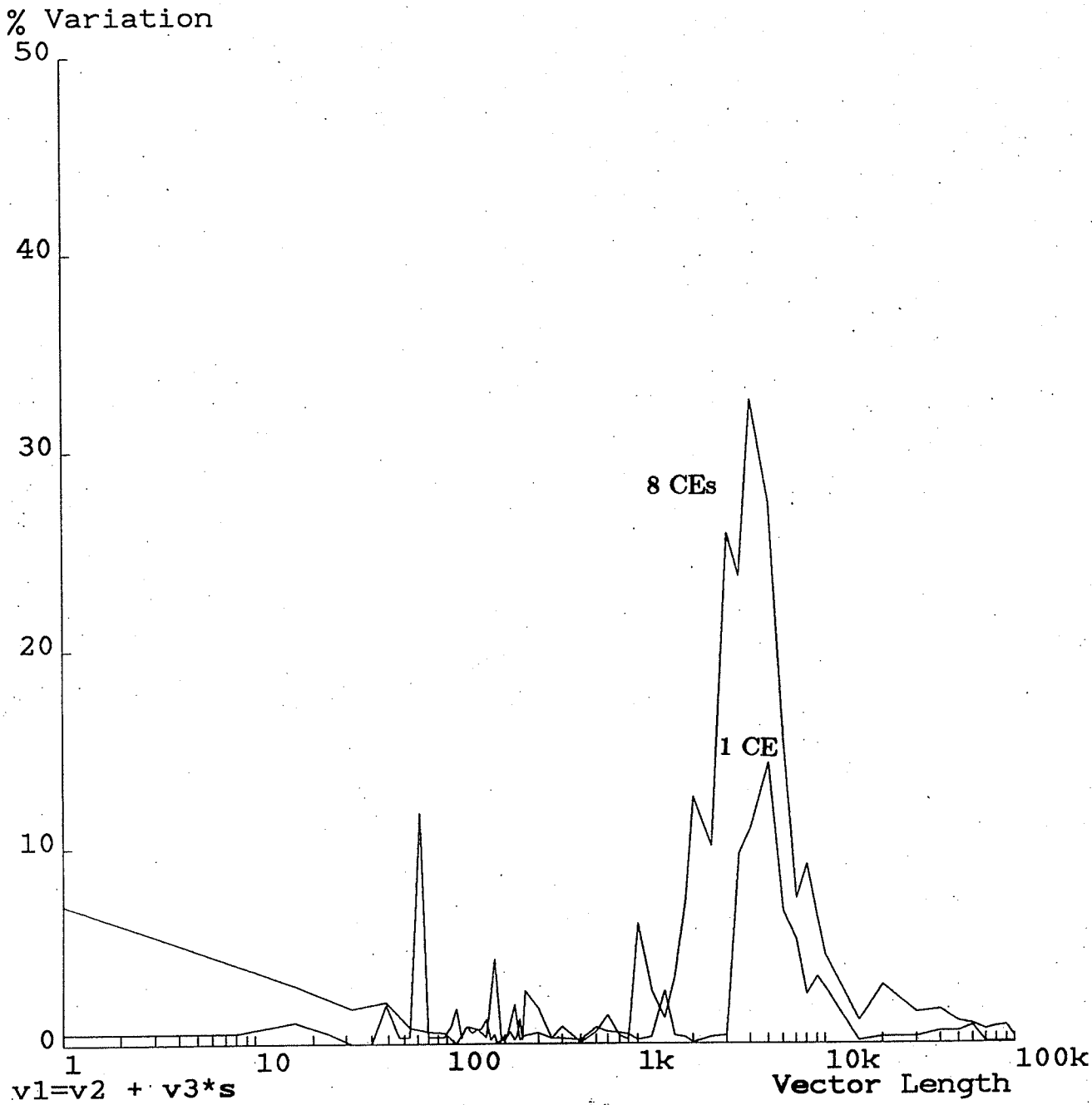


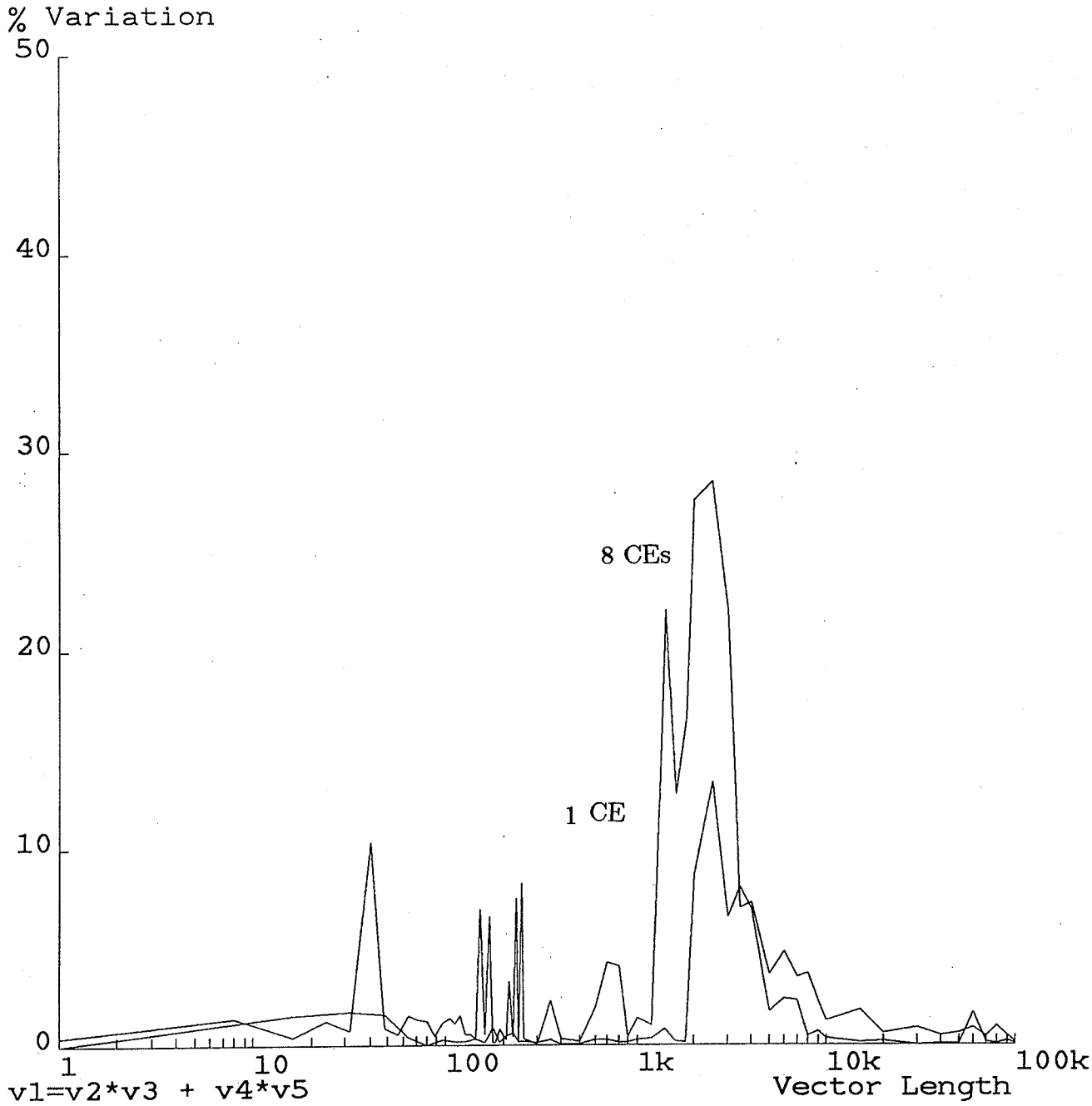
CSRD

8 CEs

Percent Variation


% Variation
50
40
30
20
10
0
8 CEs
1 CE
1
10
100
1k
10k
100k
Vector Length
v1=v2 * v3

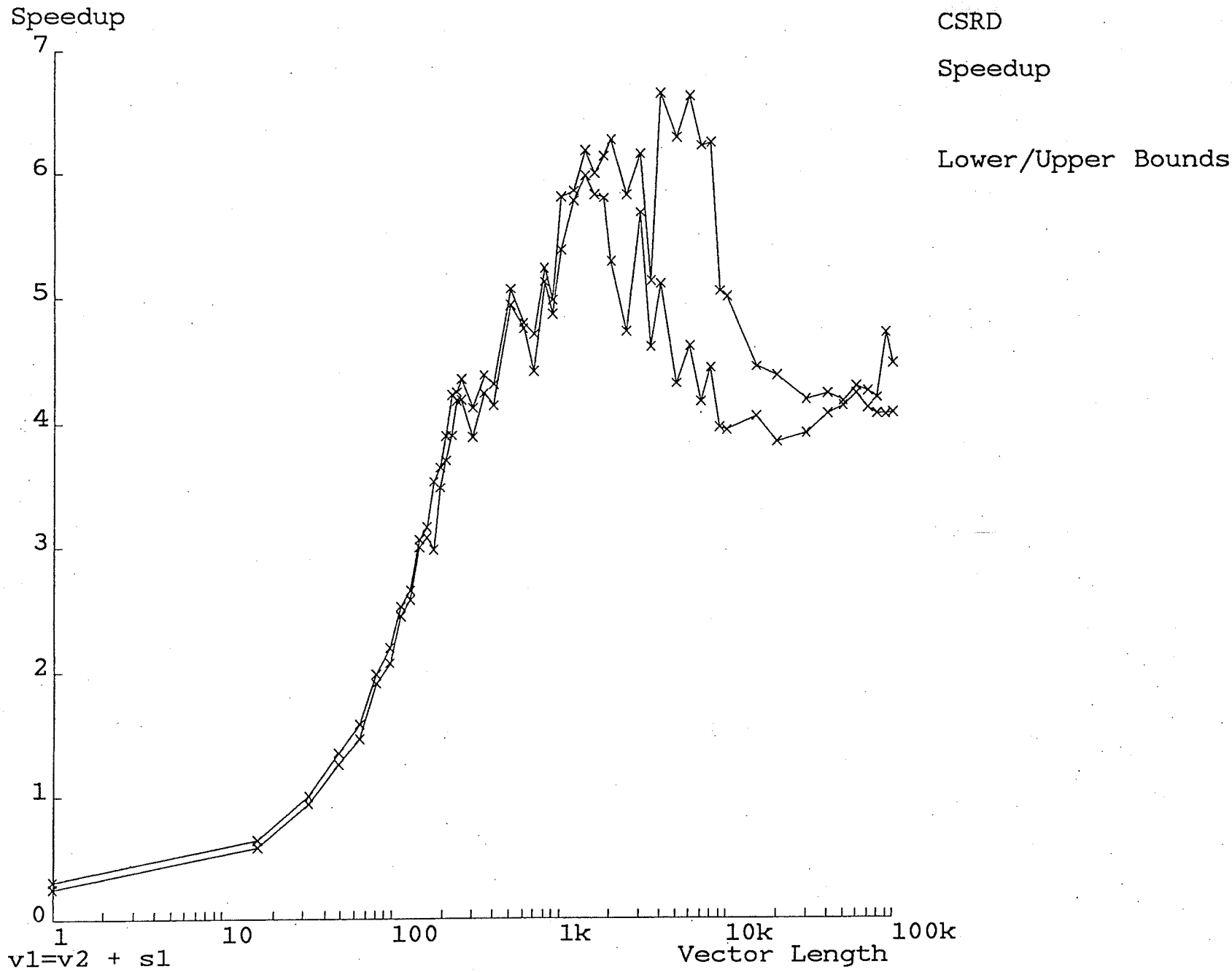

% Variation
50
40
30
20
10
0
8 CEs
1 CE
1
10
100
1k
10k
100k
Vector Length
v1=v2 + v3*s

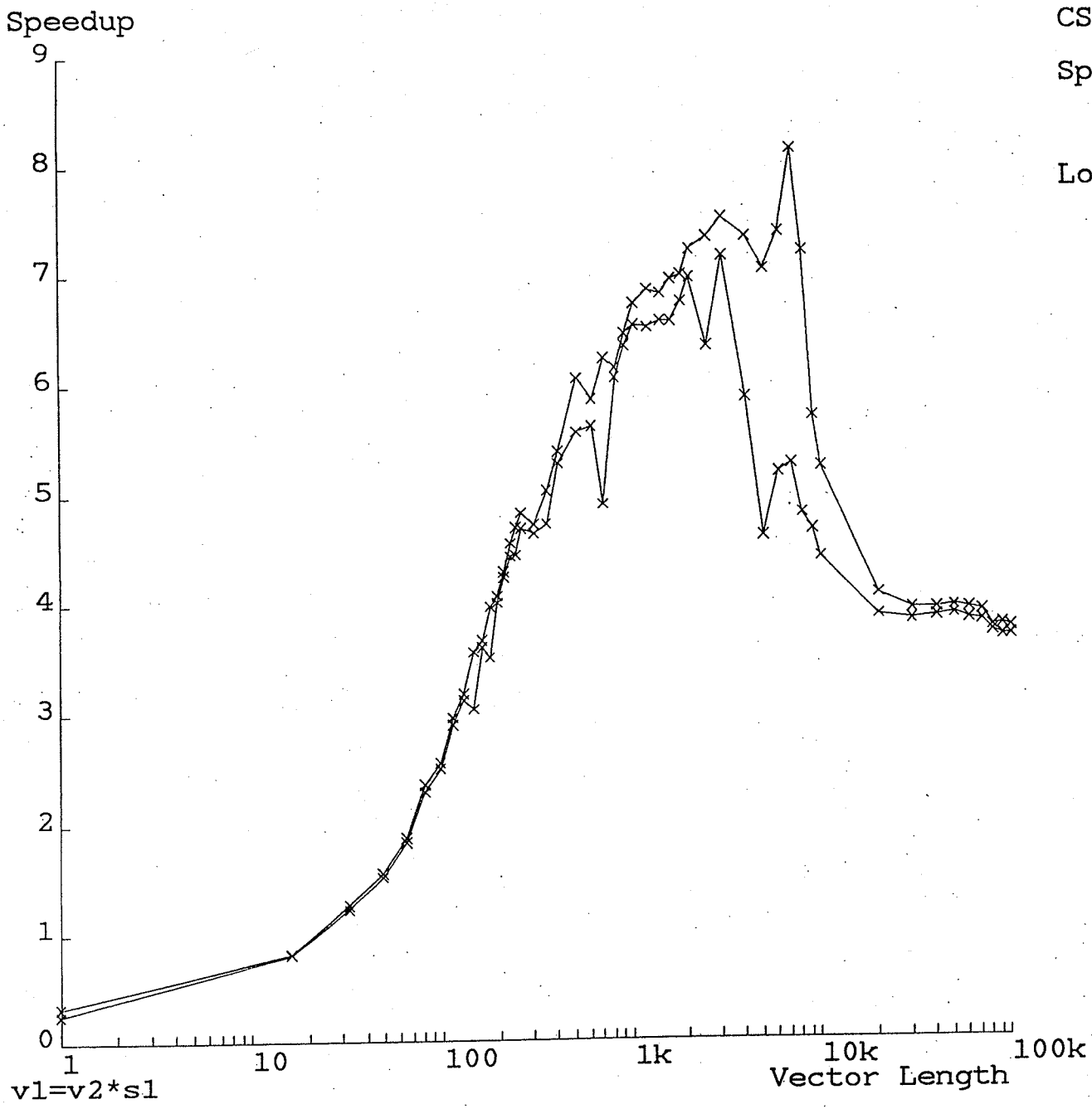

% Variation
50
40
30
20
10
0
8 CEs
1 CE
1
10
100
1k
10k
100k
v1=v2*v3 + v4*v5
Vector Length

Appendix B

The Speedup Upper/Lower Bounds

The following plots show the speedup upper/lower bounds for the kernels not presented in Figure 5. The vector lengths range from 1 to 100,000.

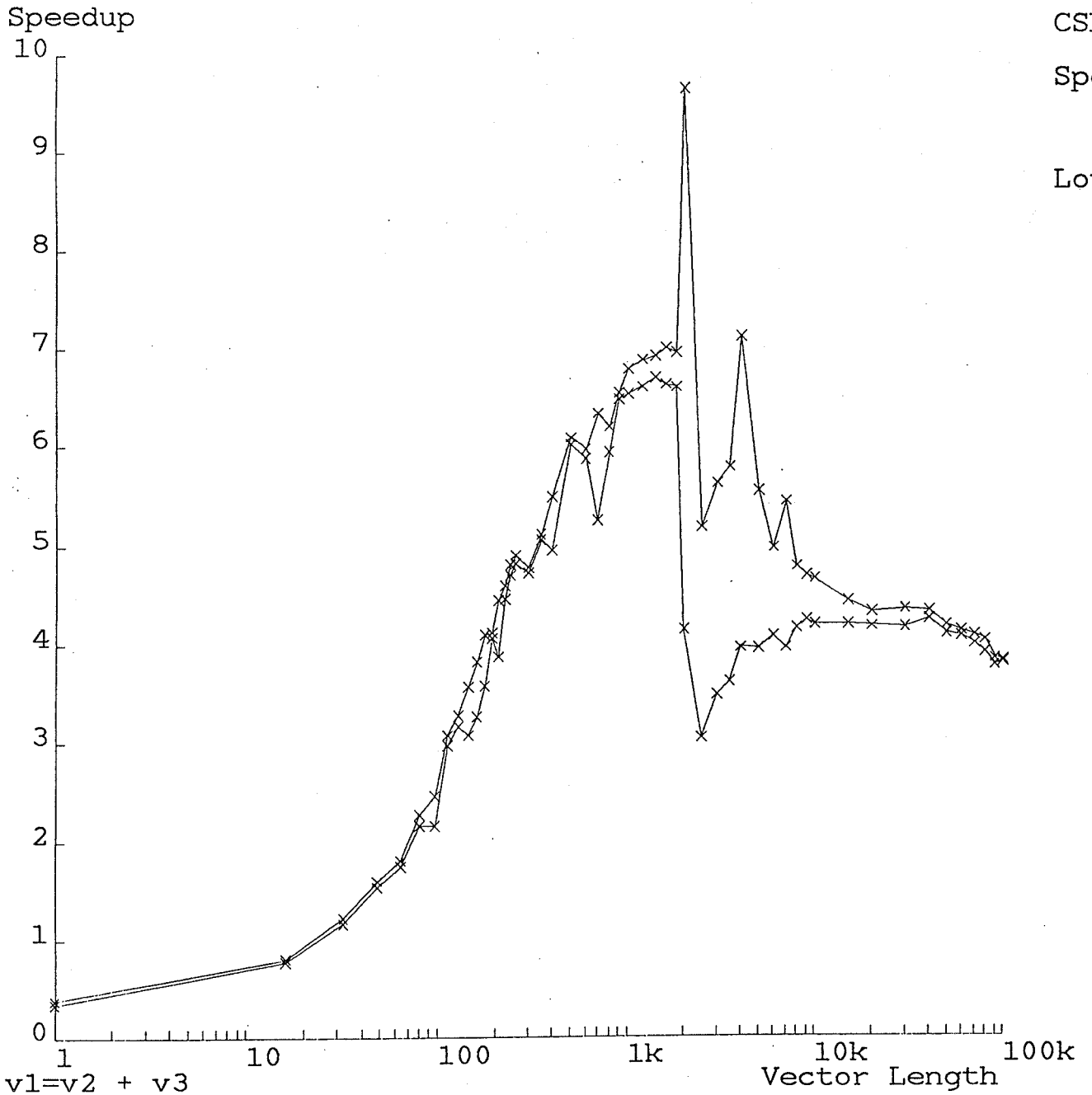

Speedup
7
6
5
4
3
2
1
0
1
10
100
1k
10k
100k
v1=v2 + s1
Vector Length
CSRD
Speedup
Lower/Upper Bounds



CSRD

Speedup

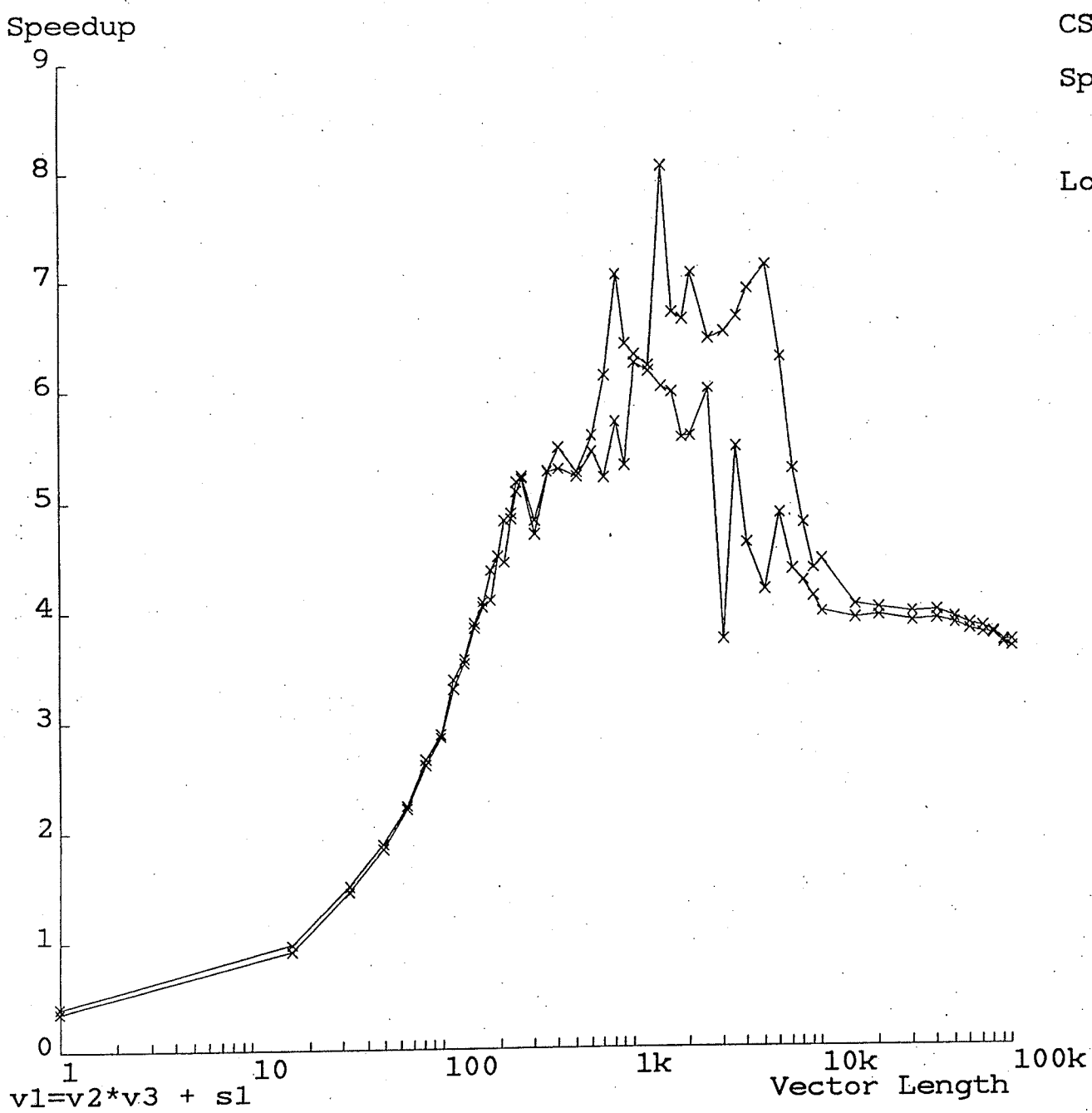
Lower/Upper Bounds



CSRD

Speedup

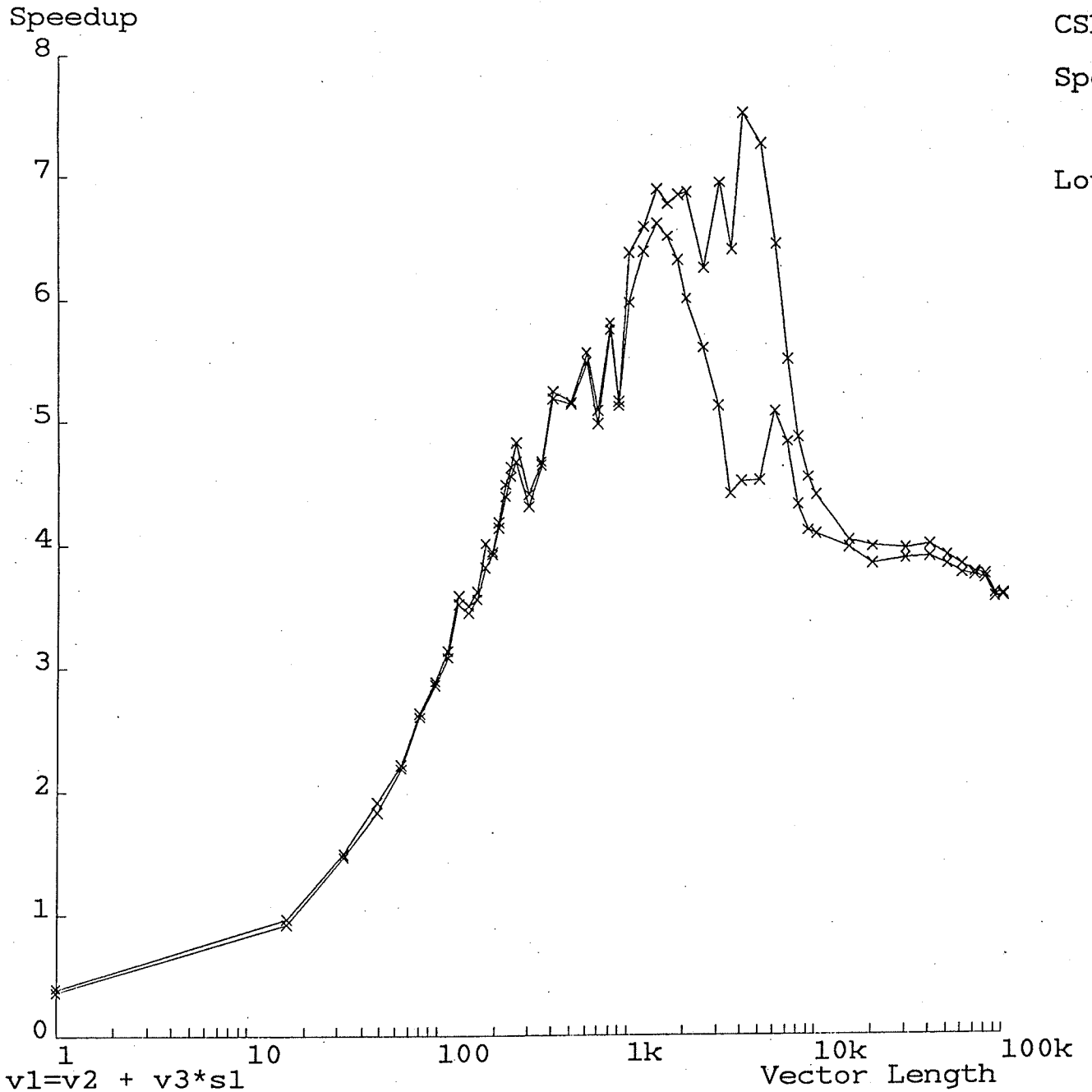
Lower/Upper Bounds



CSRD

Speedup

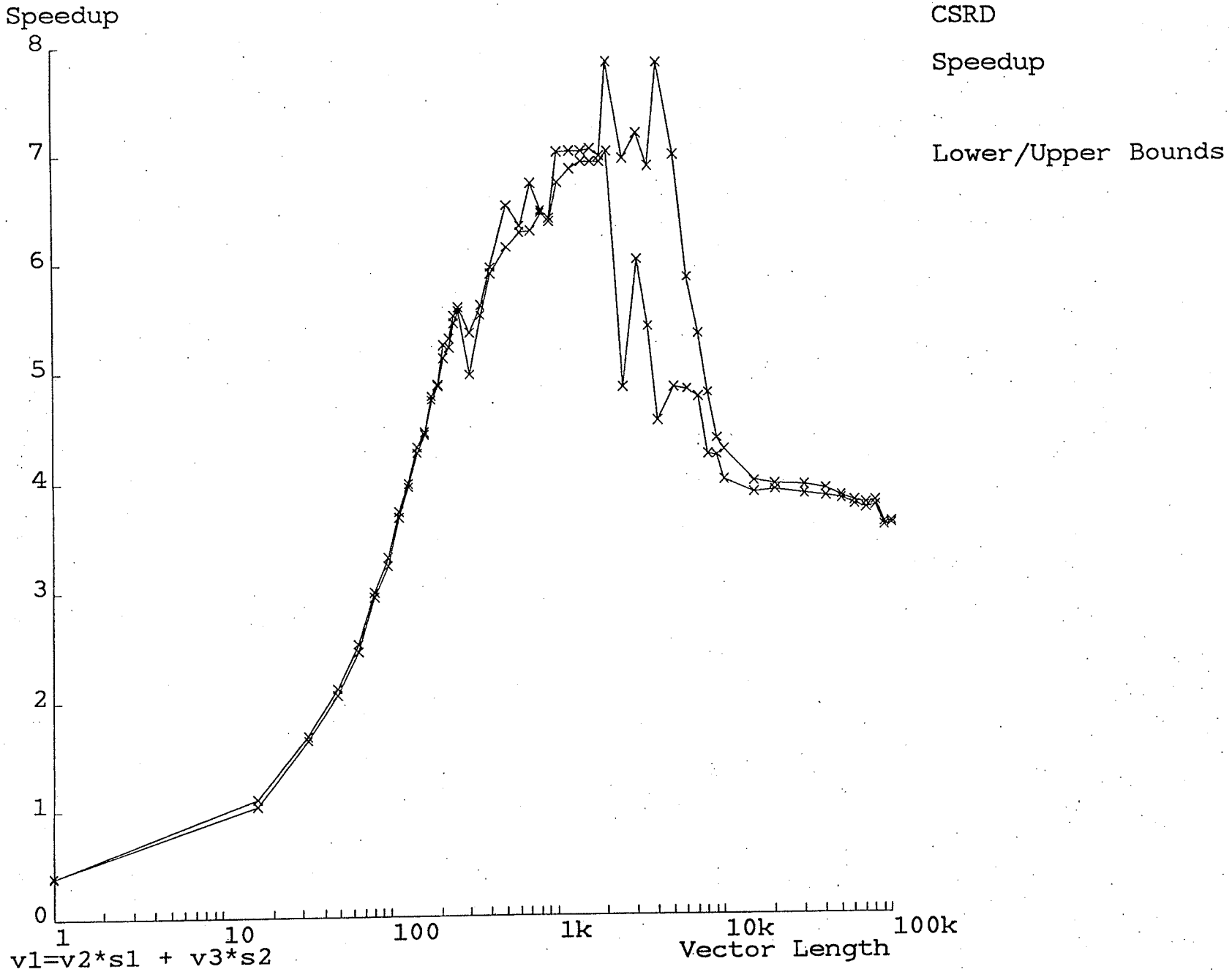
Lower/Upper Bounds

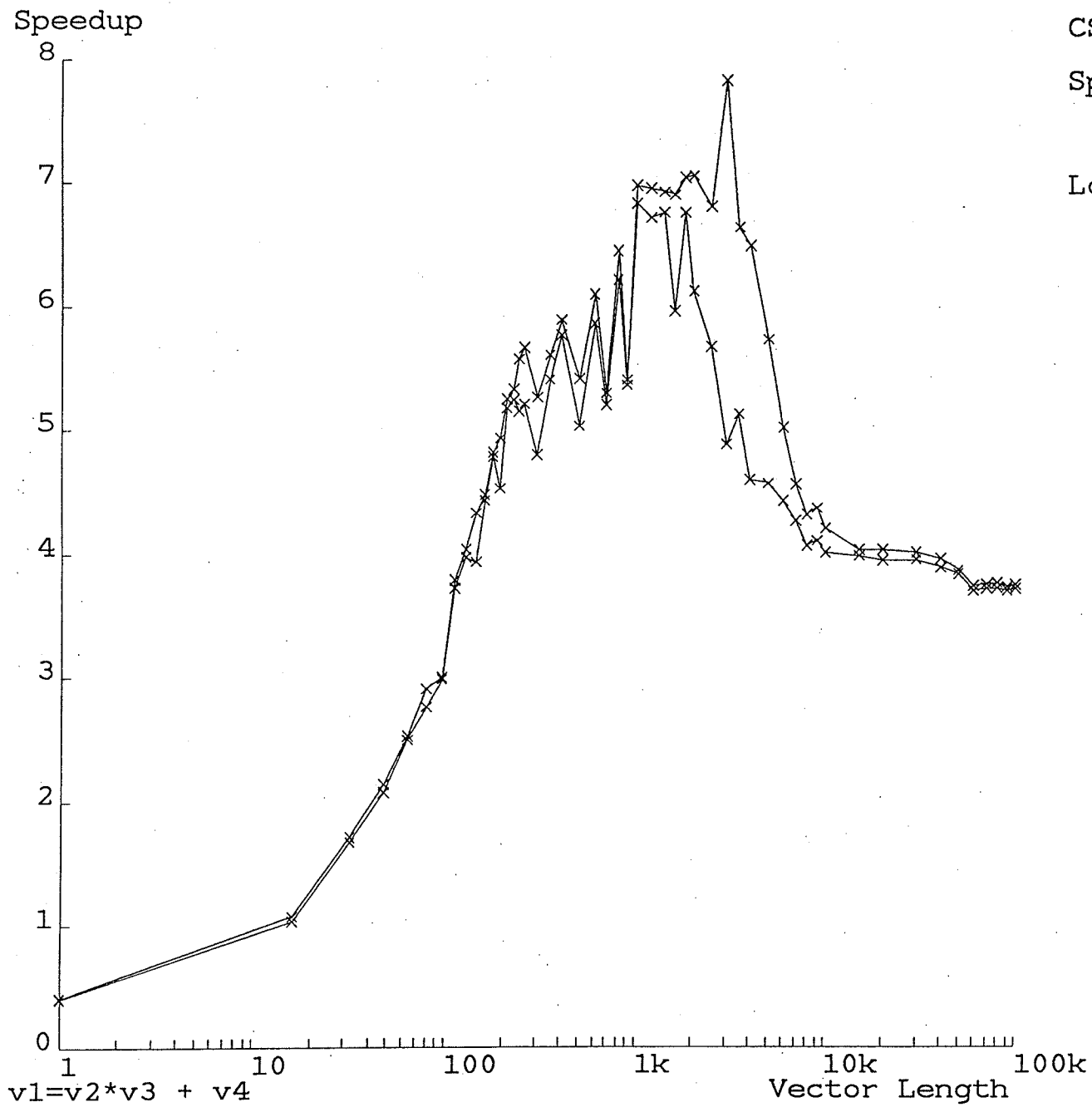


CSRD

Speedup

Lower/Upper Bounds

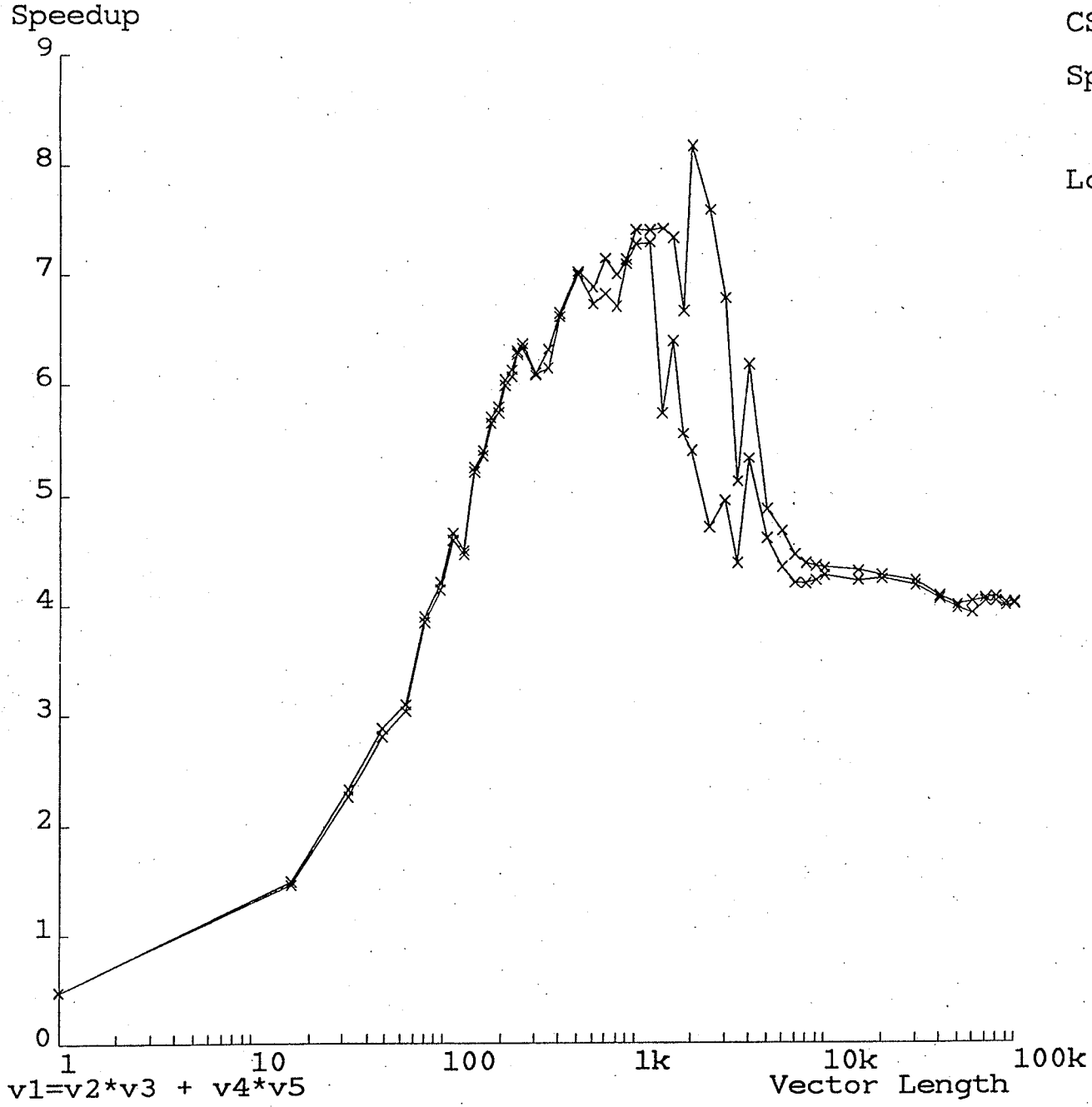

Speedup
8
7
6
5
4
3
2
1
0
1
10
100
1k
10k
100k
Vector Length
v1=v2*s1 + v3*s2
CSRD
Speedup
Lower/Upper Bounds



CSRD

Speedup

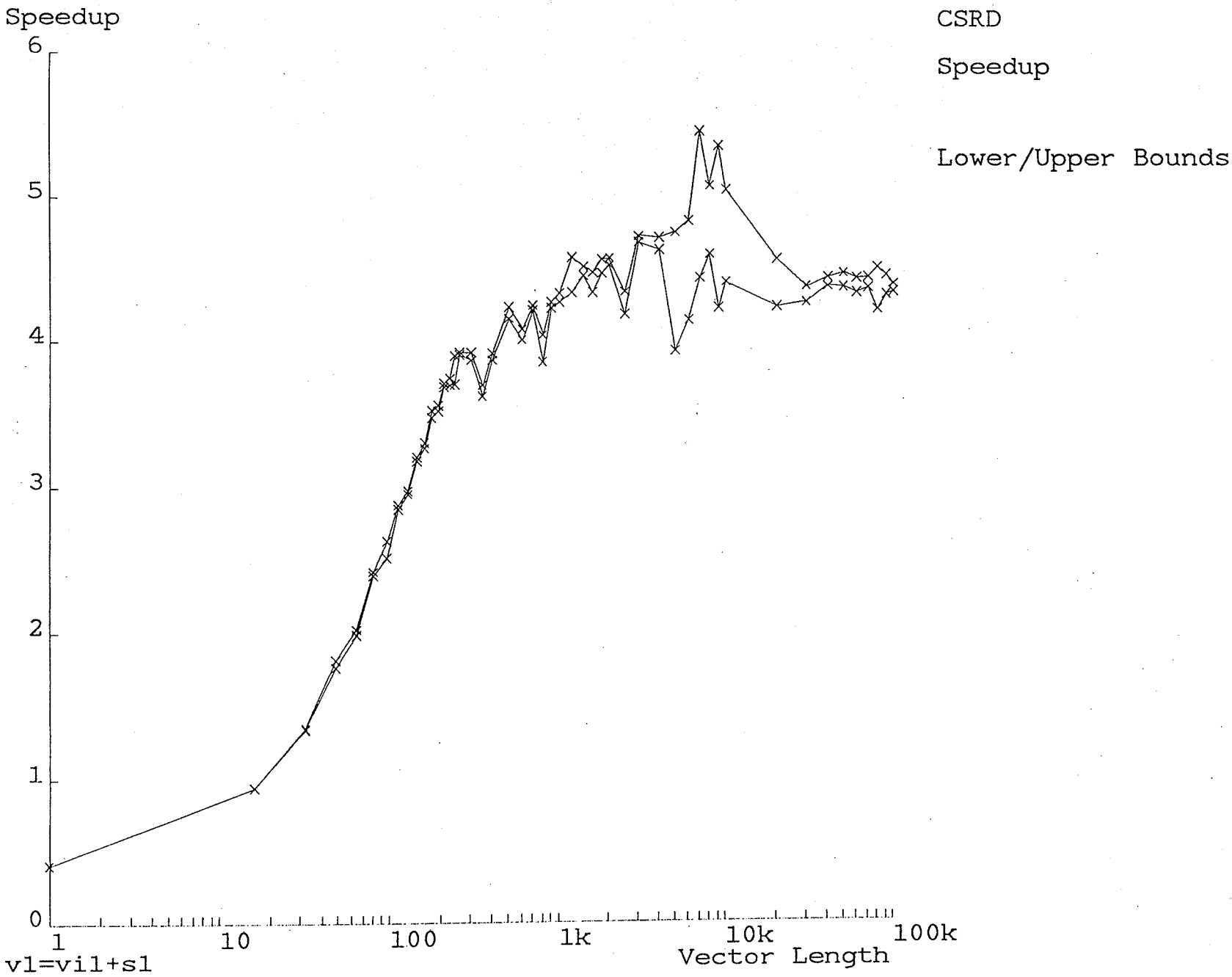
Lower/Upper Bounds

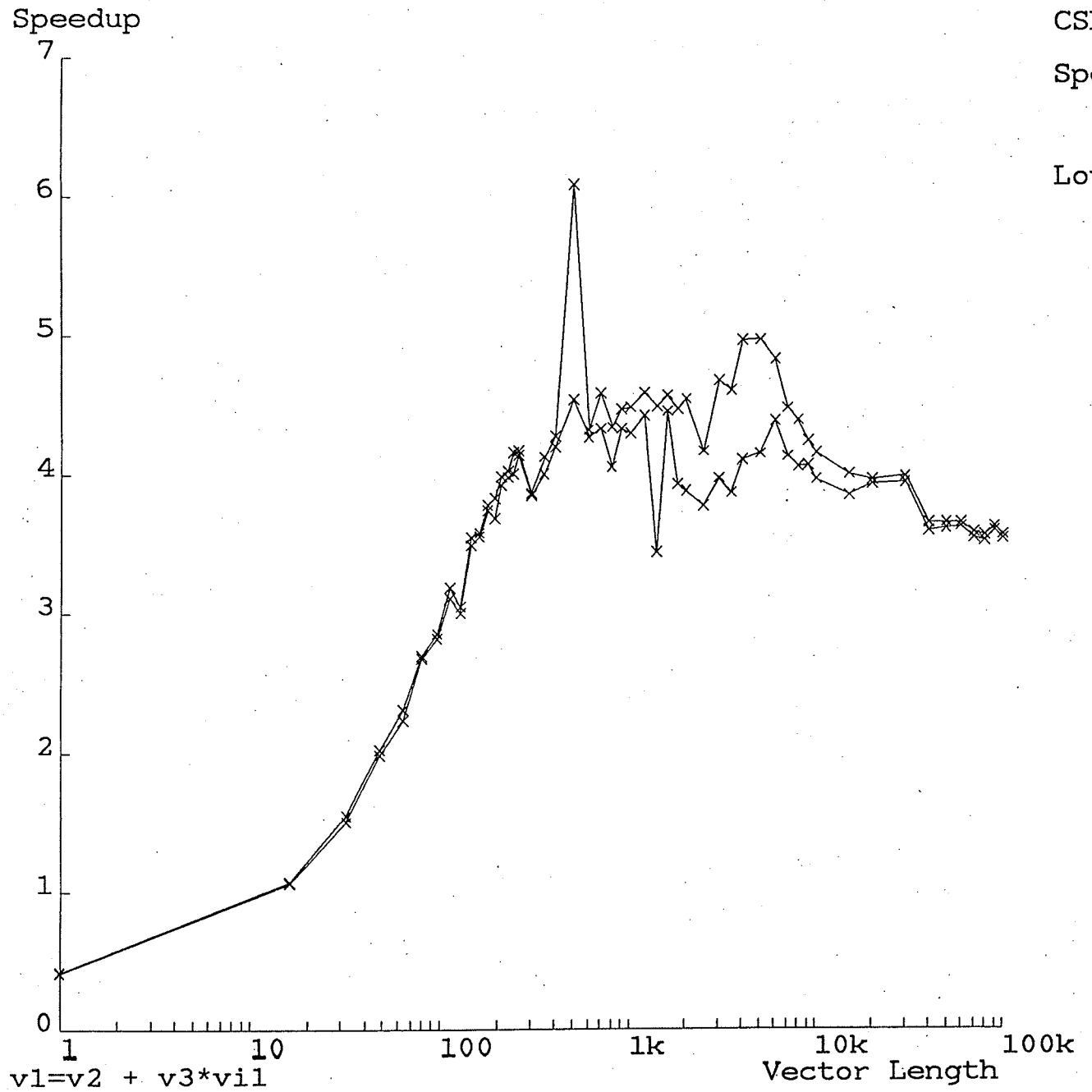


CSRD

Speedup

Lower/Upper Bounds

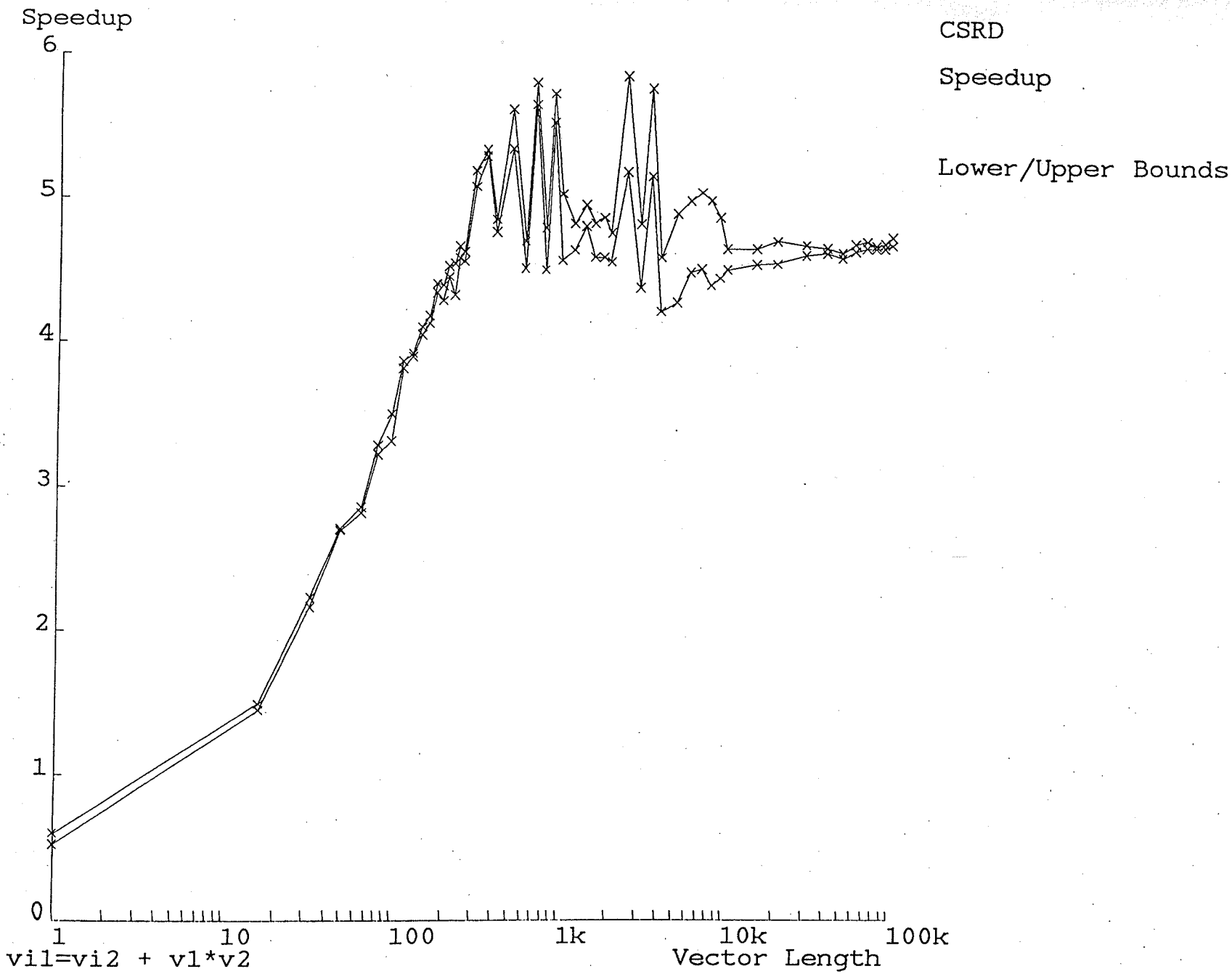

Speedup
6
5
4
3
2
1
0
1
10
100
1k
10k
100k
Vector Length
v1=vi1+s1
CSRD
Speedup
Lower/Upper Bounds

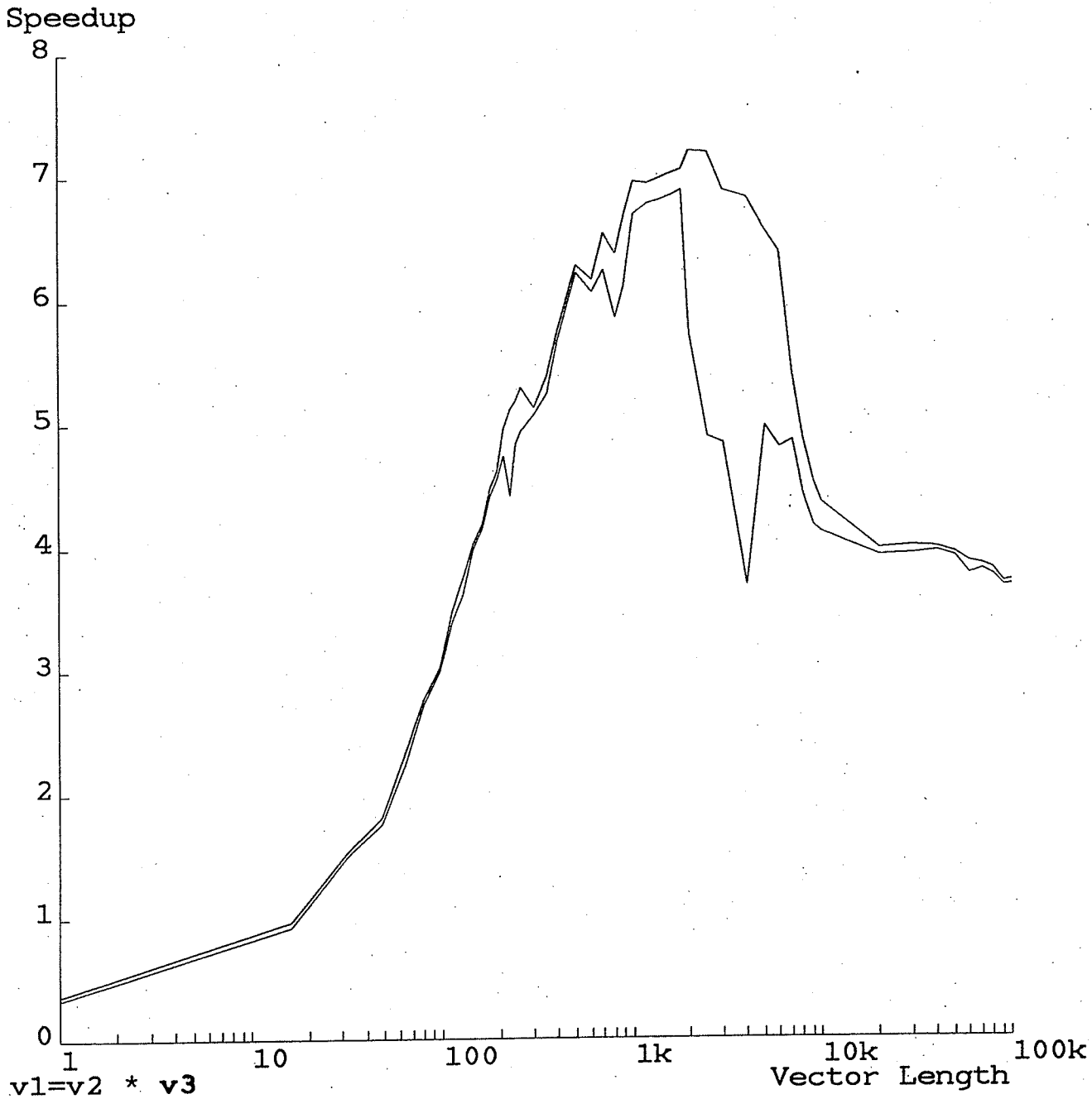


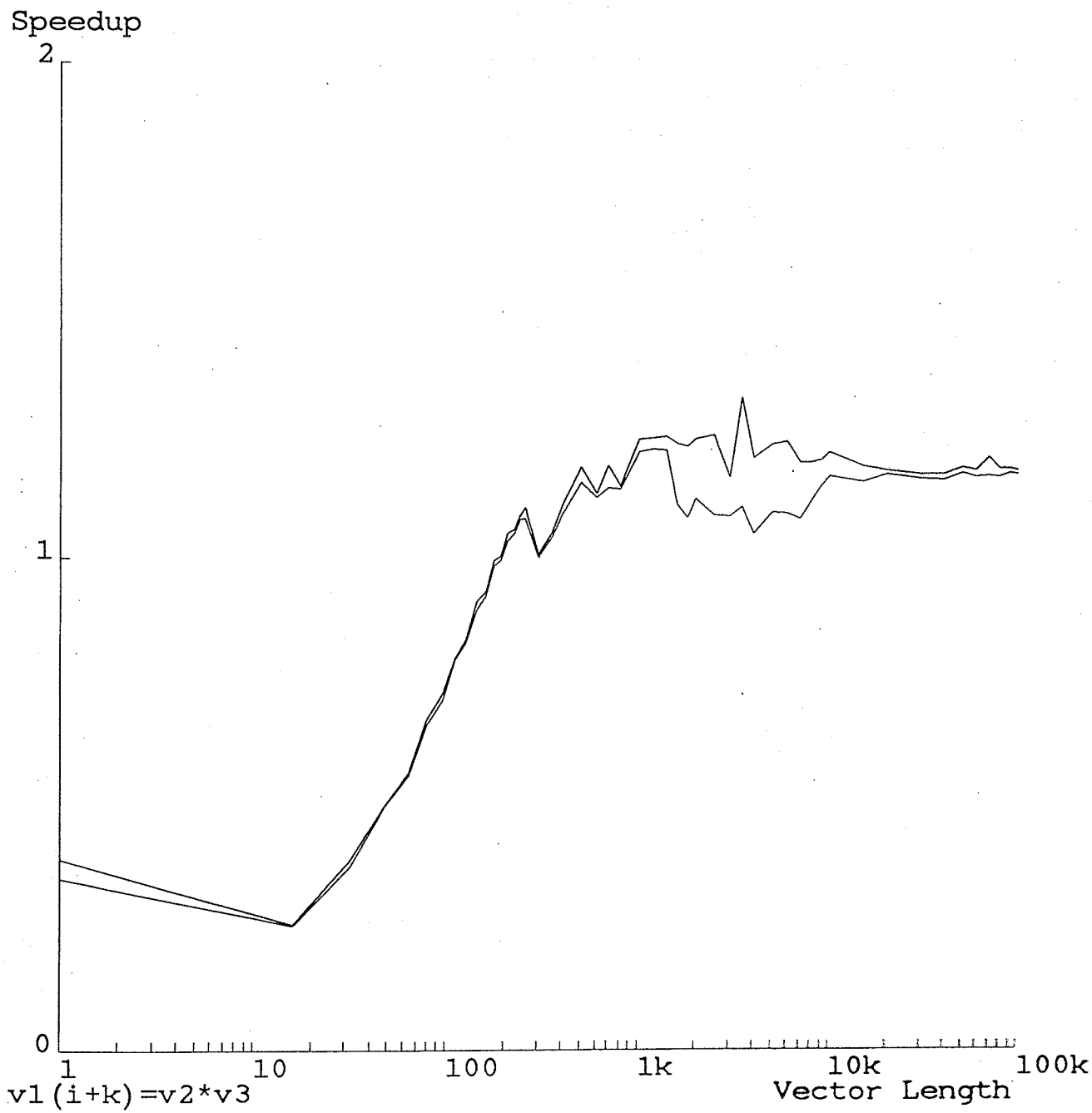
CSRD

Speedup

Lower/Upper Bounds


Speedup
6
5
4
3
2
1
0
1
10
100
1k
10k
100k
vi1=vi2 + v1*v2
Vector Length
CSRD
Speedup
Lower/Upper Bounds

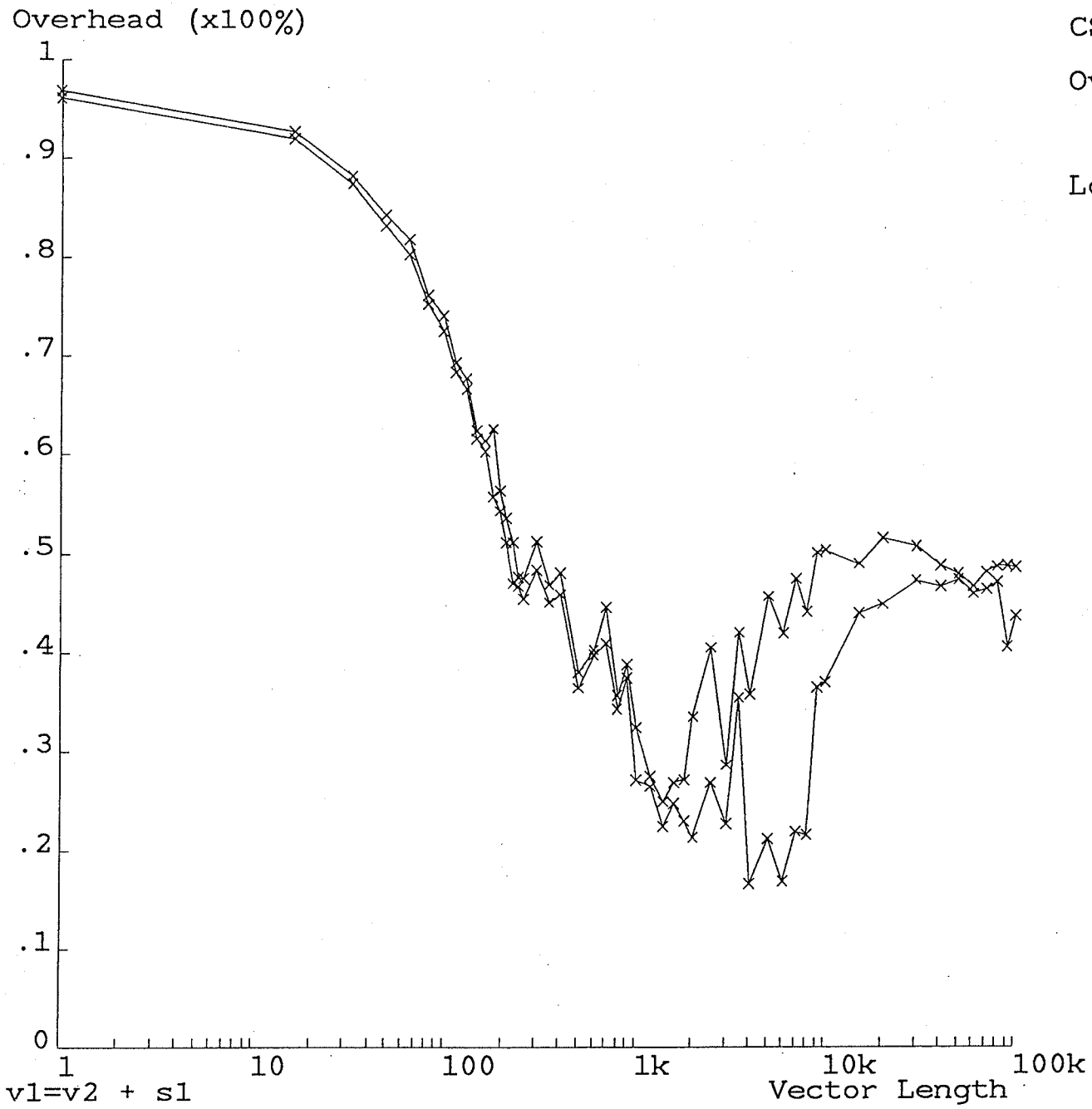

Speedup
8
7
6
5
4
3
2
1
0
1
10
100
1k
10k
100k
Vector Length
v1=v2 * v3


Speedup
2
1
0
1
10
100
1k
10k
100k
v1(i+k)=v2*v3
Vector Length

Appendix C

The Percentage Overhead Upper/Lower Bounds

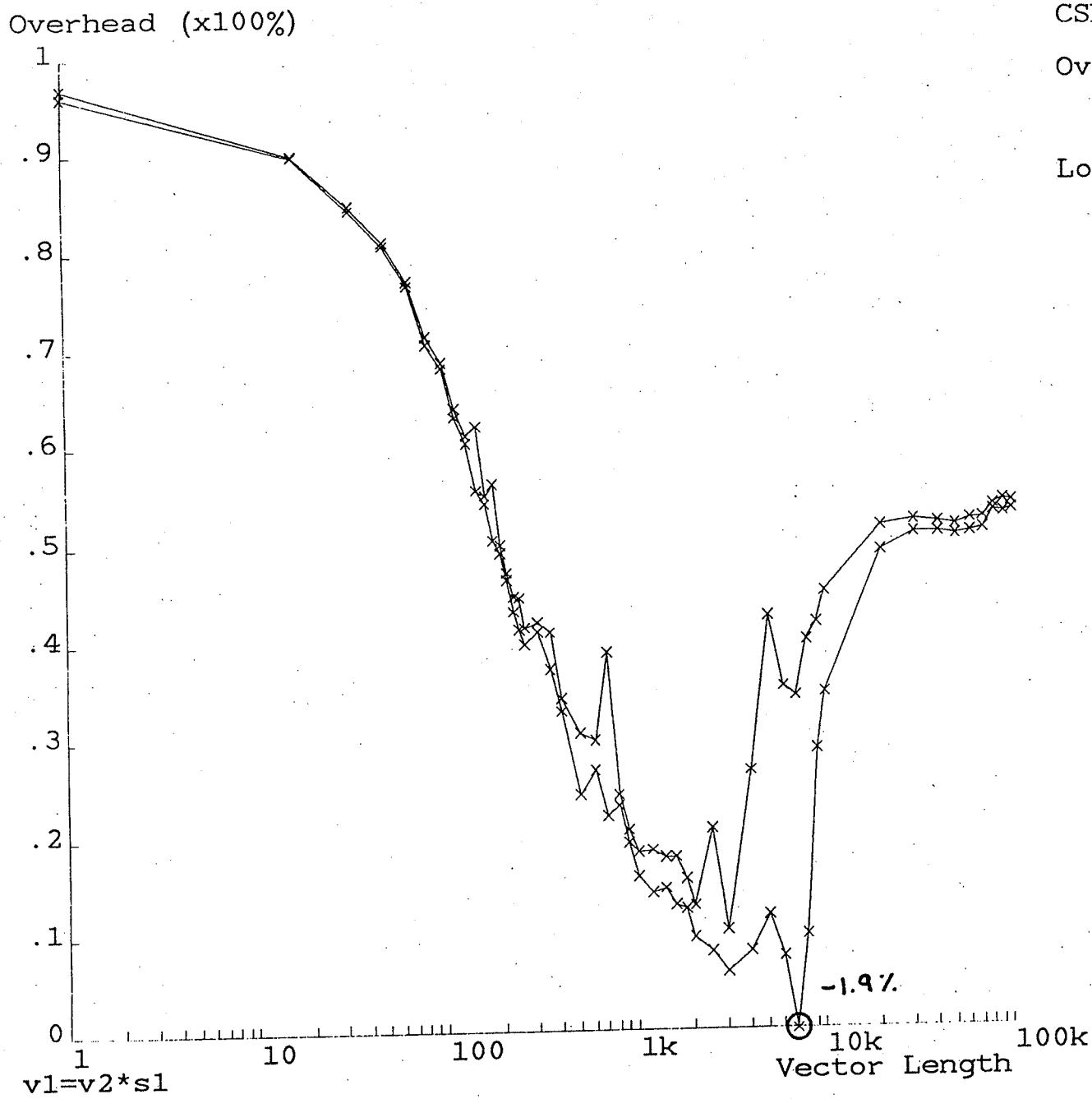
The following plots show the percentage overhead upper/lower bounds for the kernels not presented in Figure 6. The vector lengths range from 1 to 100,000.



CSRD

Overhead Percentage

Lower/Upper Bounds

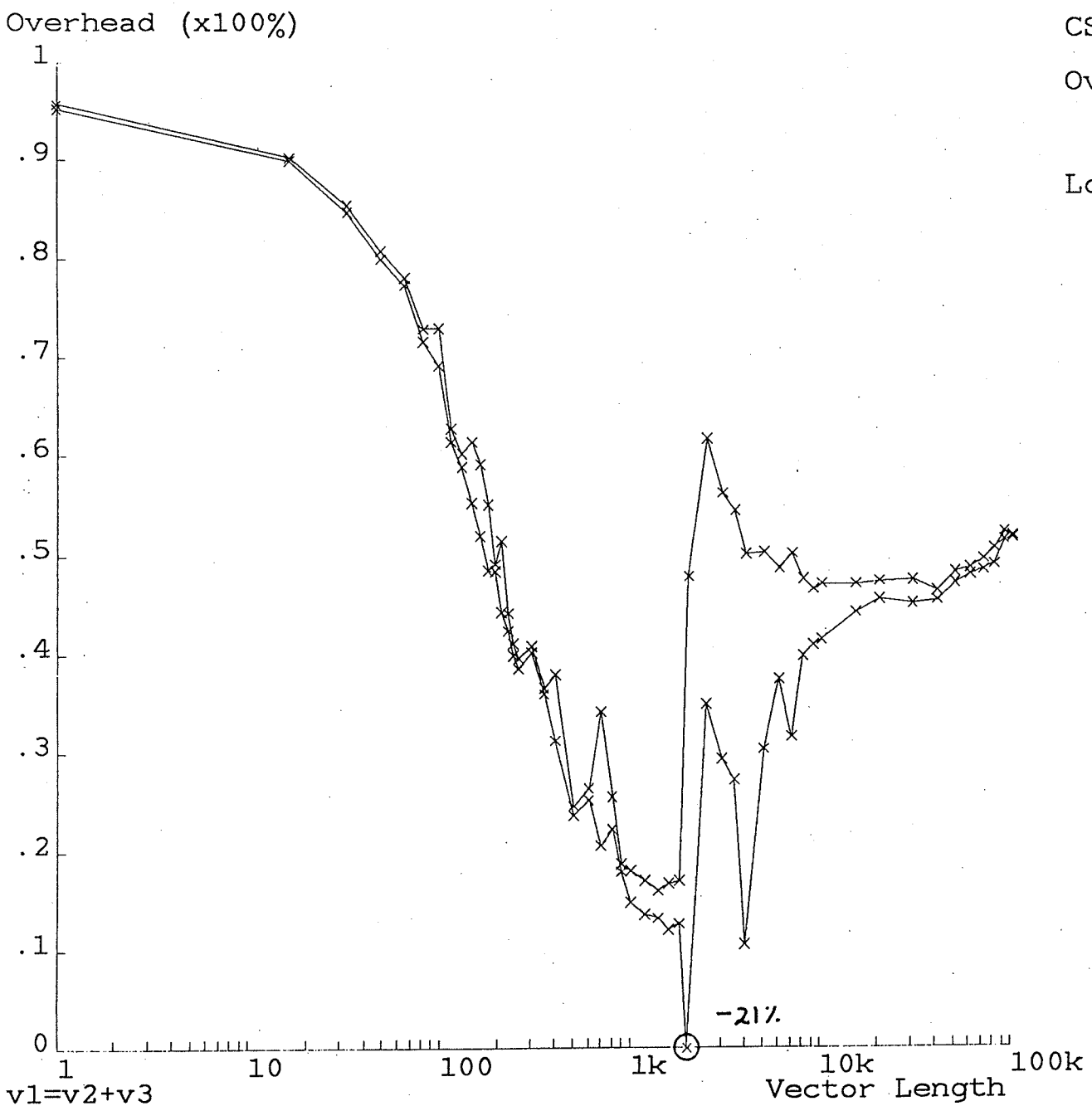


CSRD

Overhead Percentage

Lower/Upper Bounds

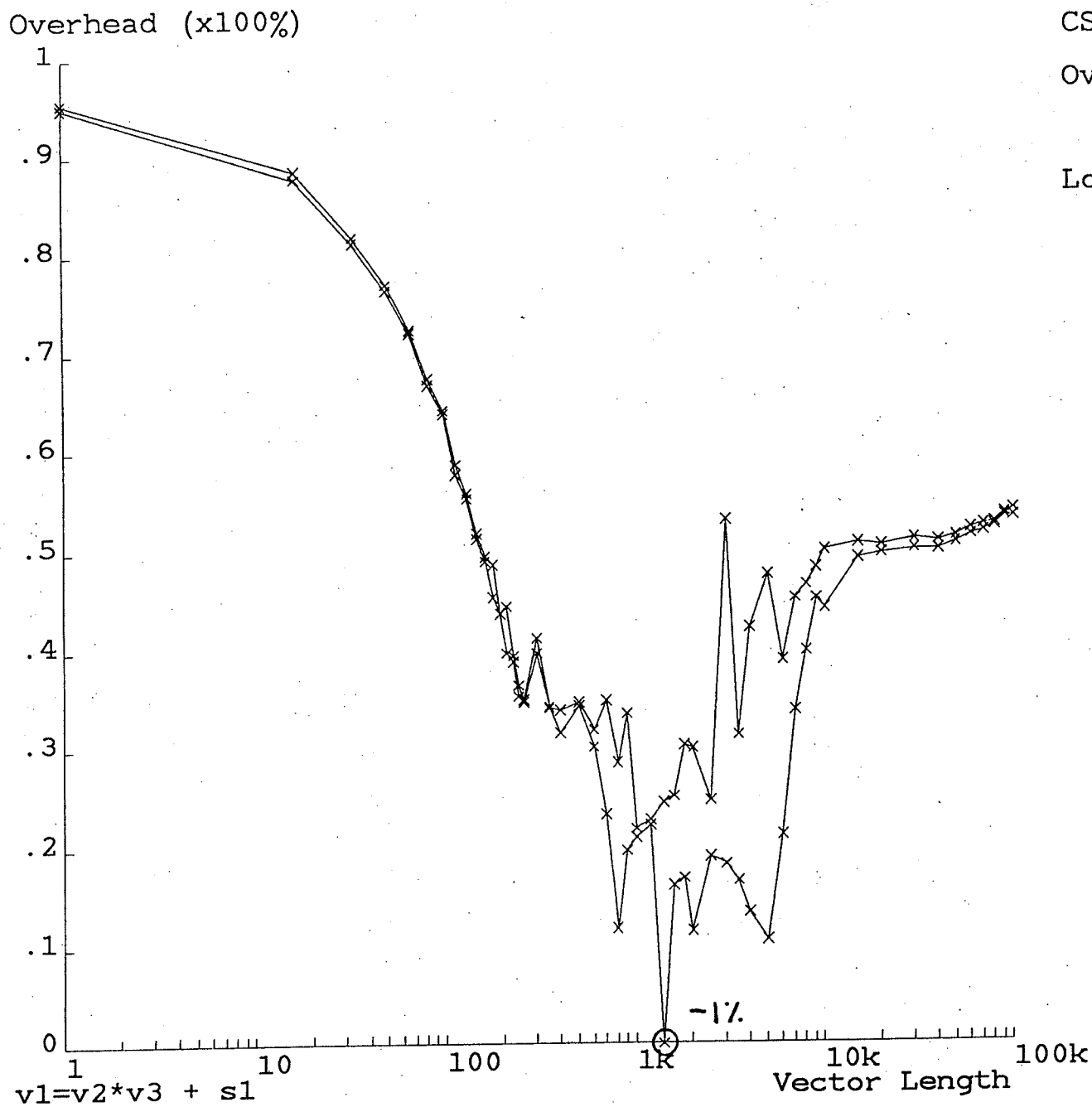
C2



CSRD

Overhead Percentage

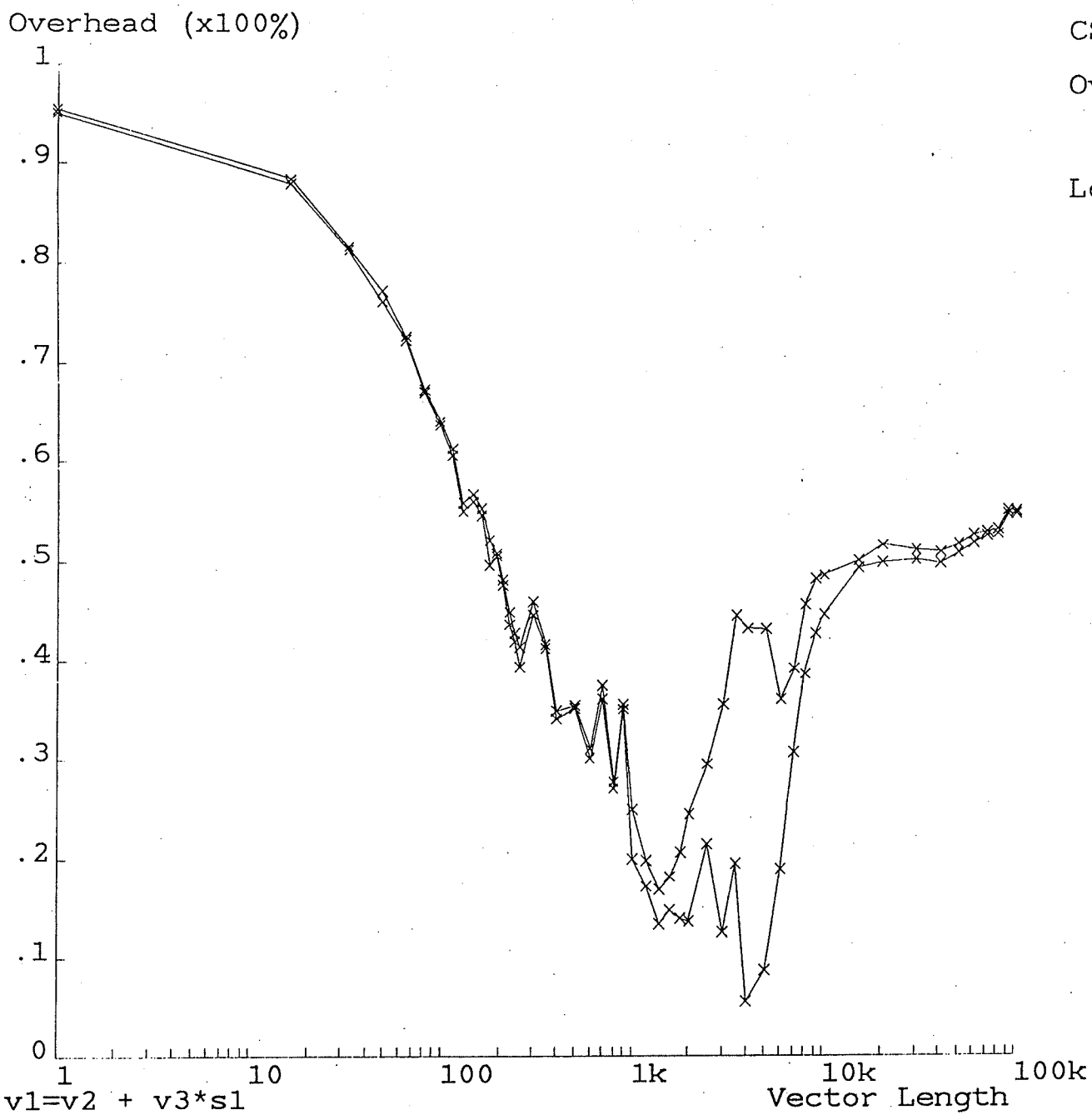
Lower/Upper Bounds



CSRD

Overhead Percentage

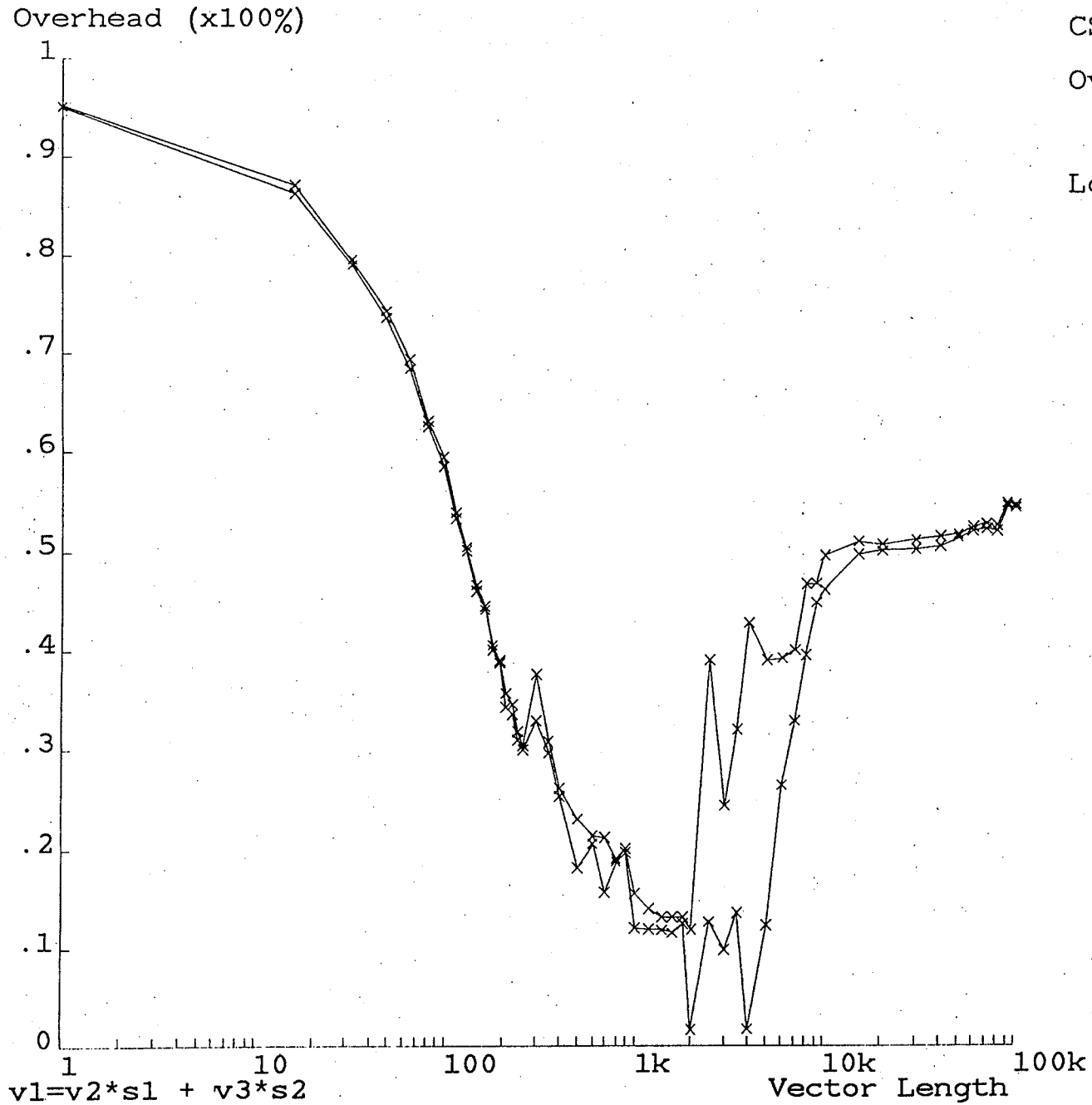
Lower/Upper Bounds



CSRD

Overhead Percentage

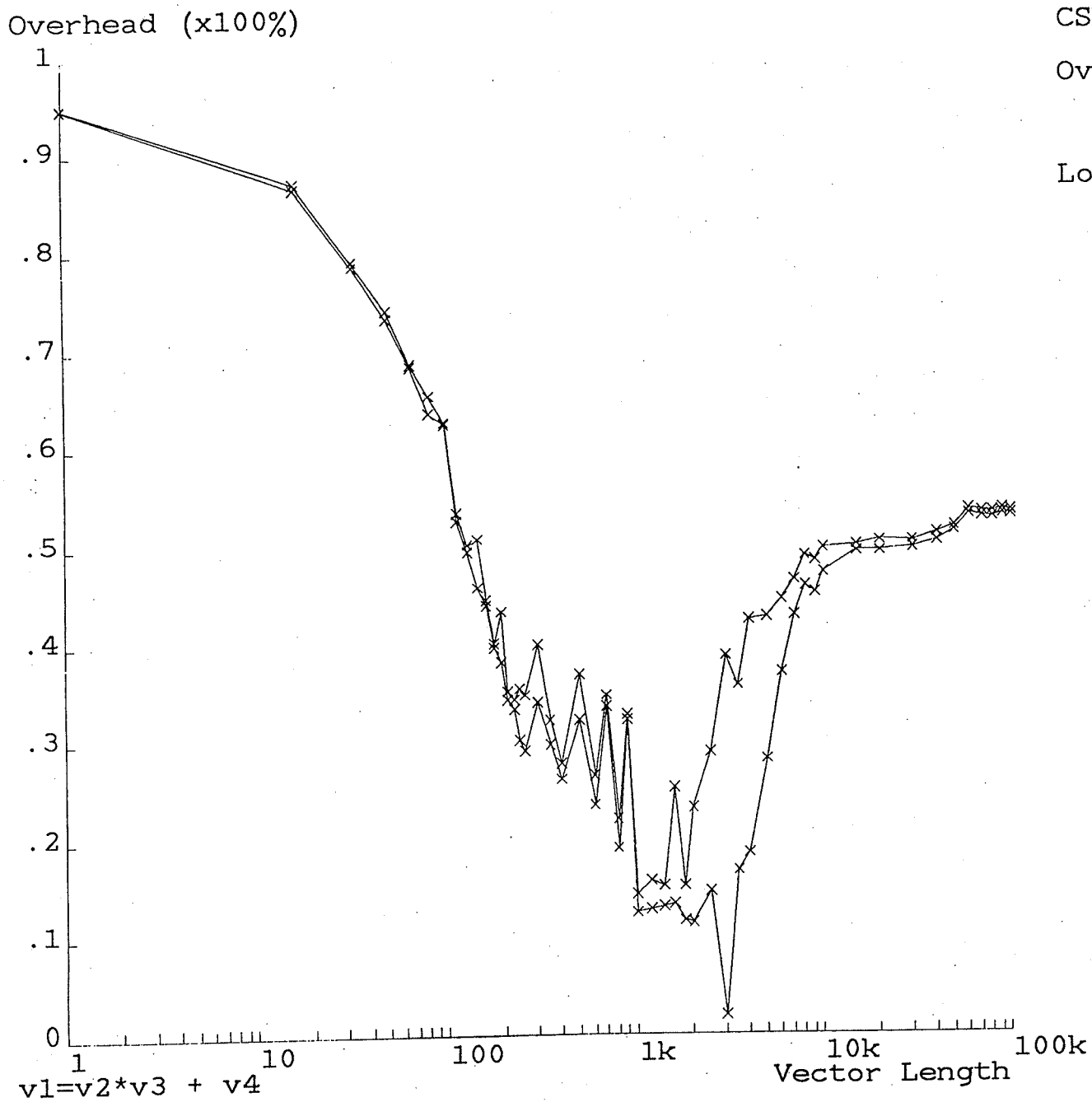
Lower/Upper Bounds



CSRD

Overhead Percentage

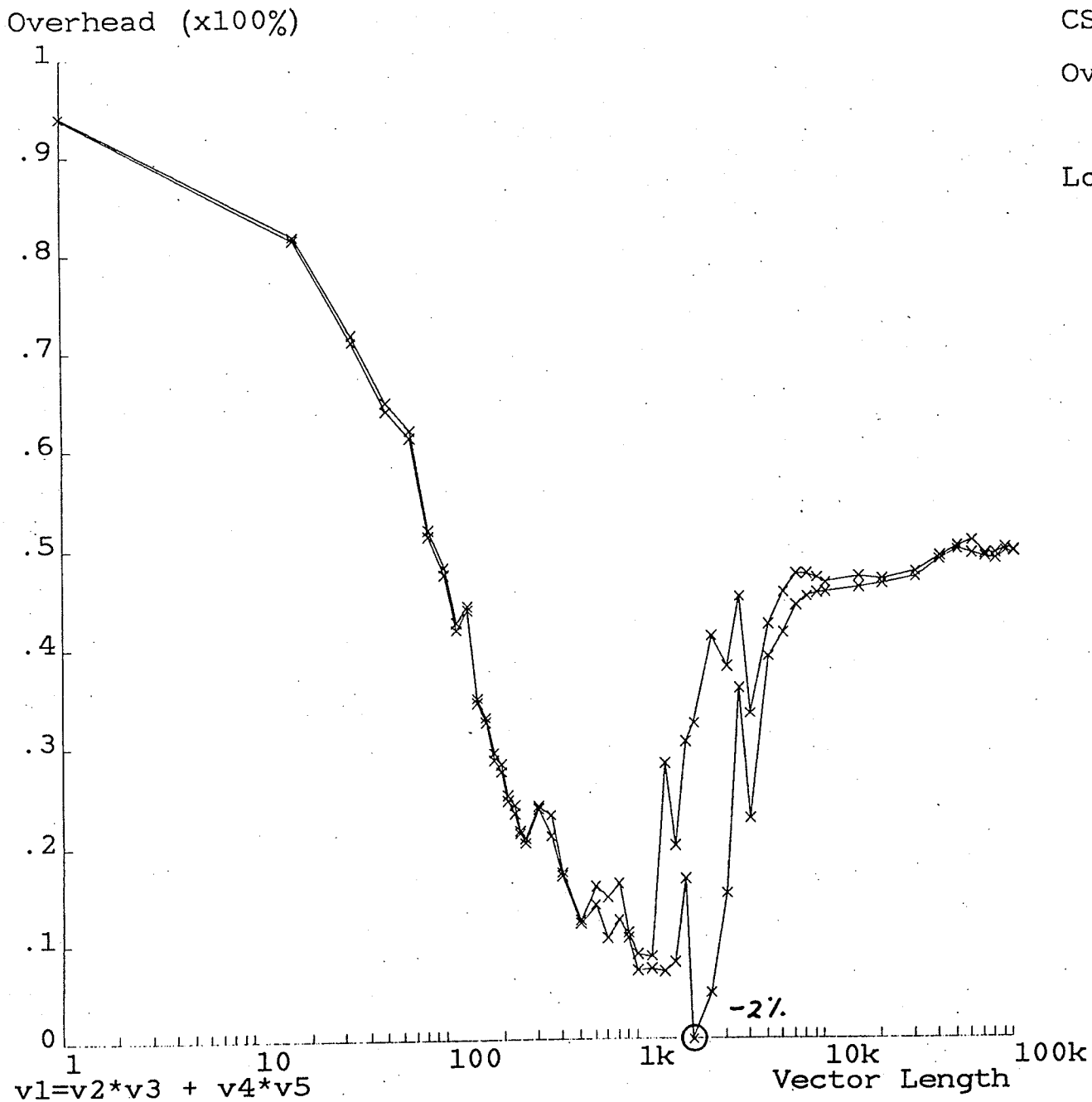
Lower/Upper Bounds



CSRD

Overhead Percentage

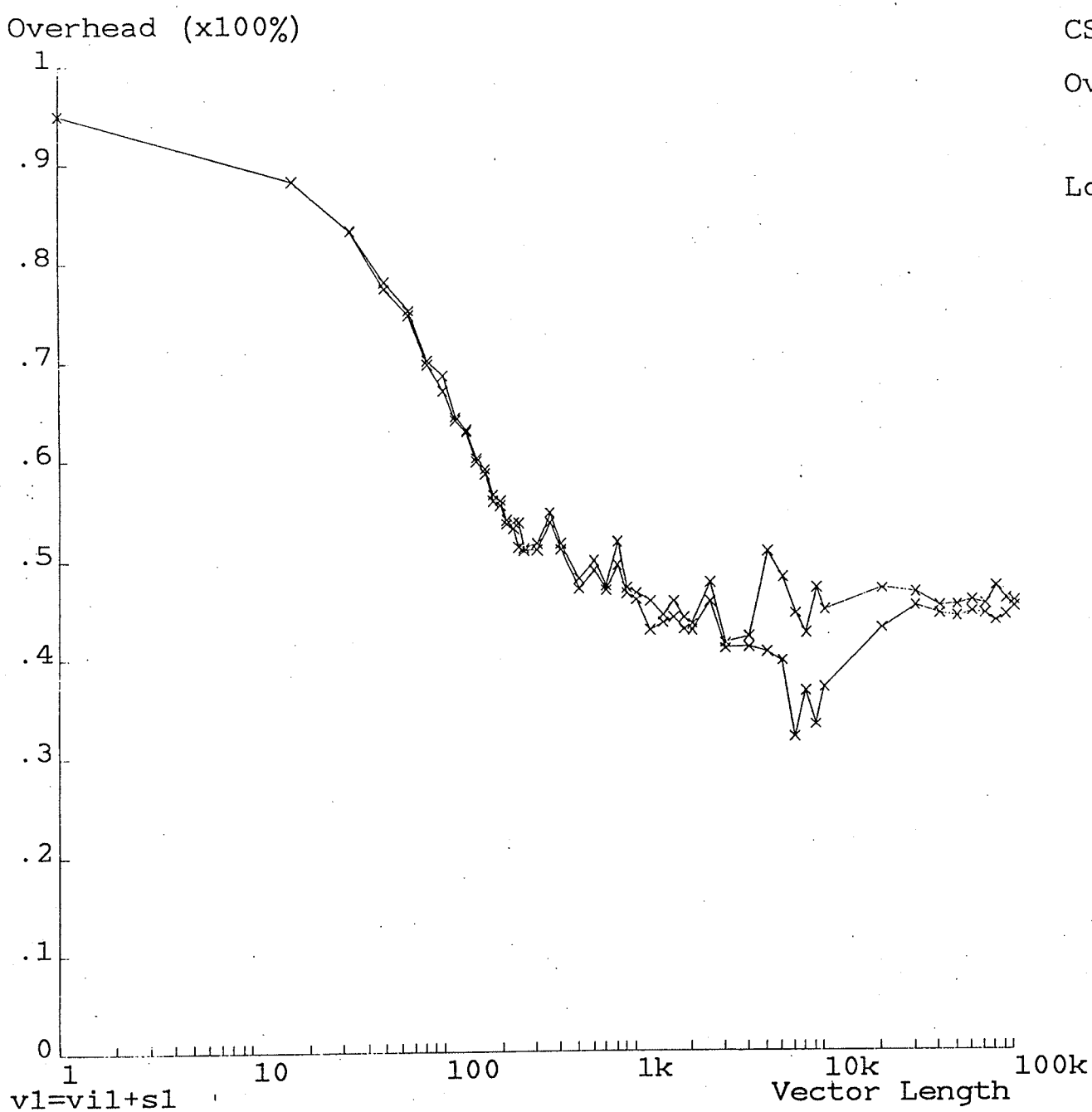
Lower/Upper Bounds



CSRD

Overhead Percentage

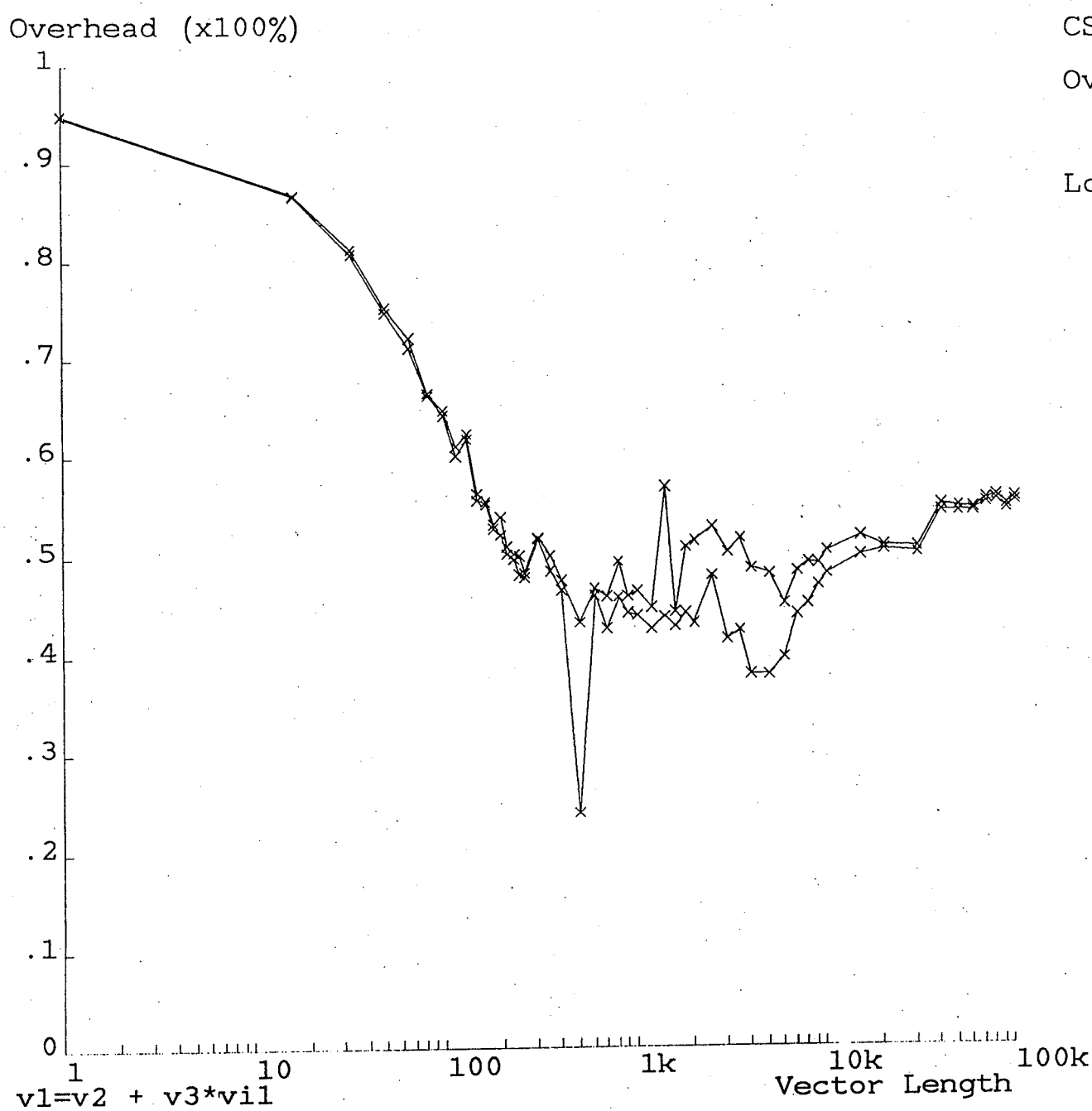
Lower/Upper Bounds



CSRD

Overhead Percentage

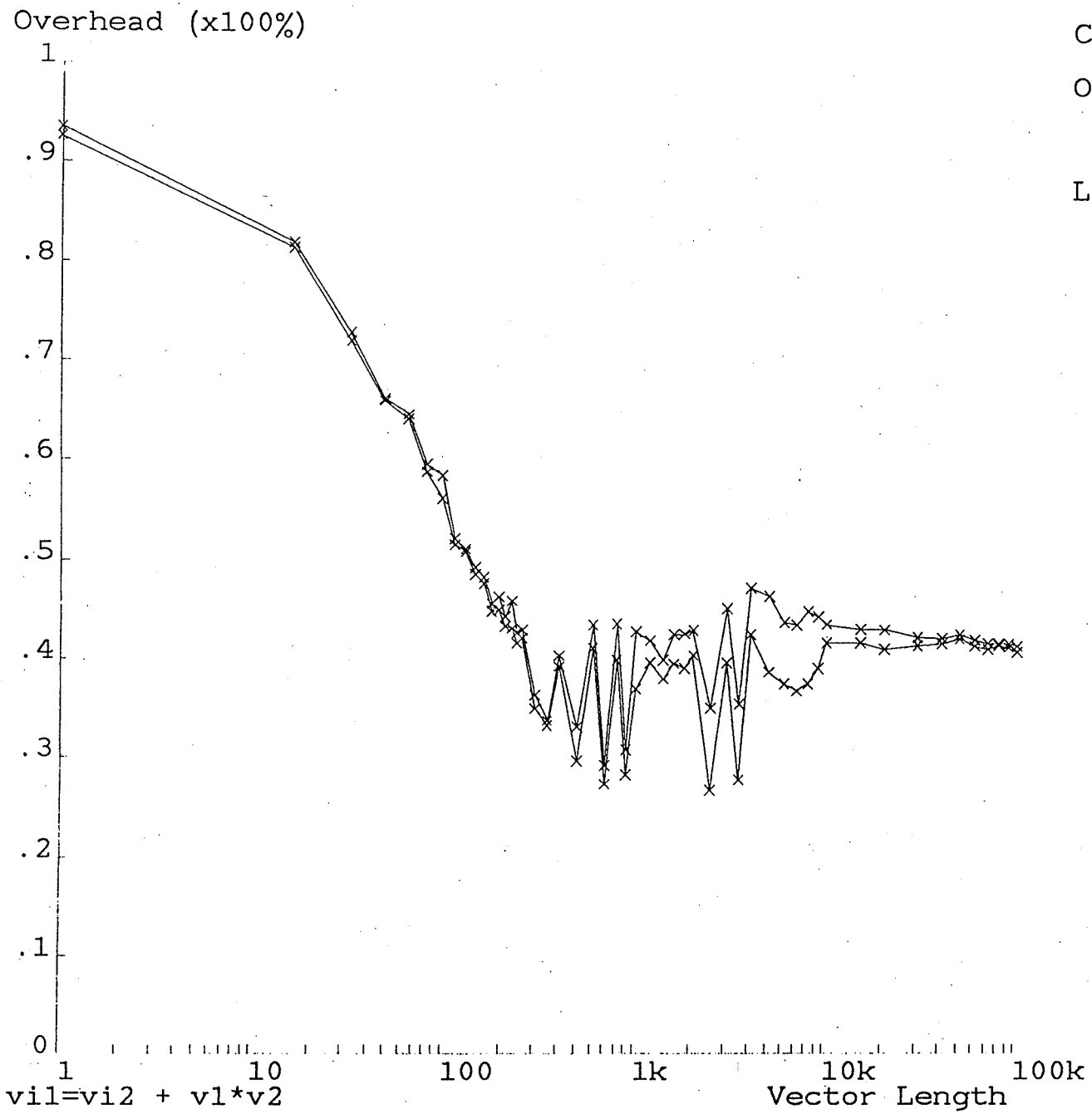
Lower/Upper Bounds



CSRD

Overhead Percentage

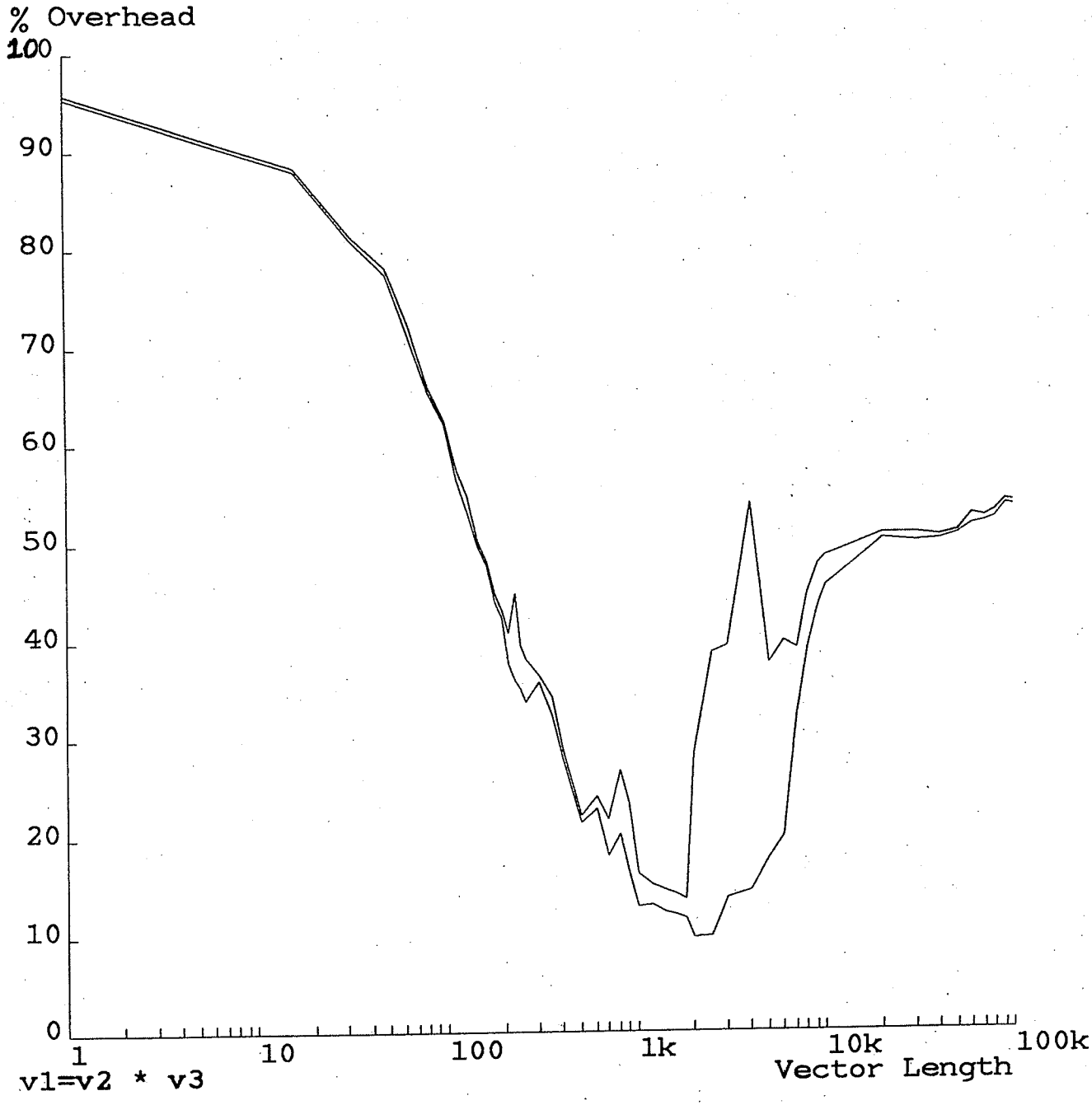
Lower/Upper Bounds

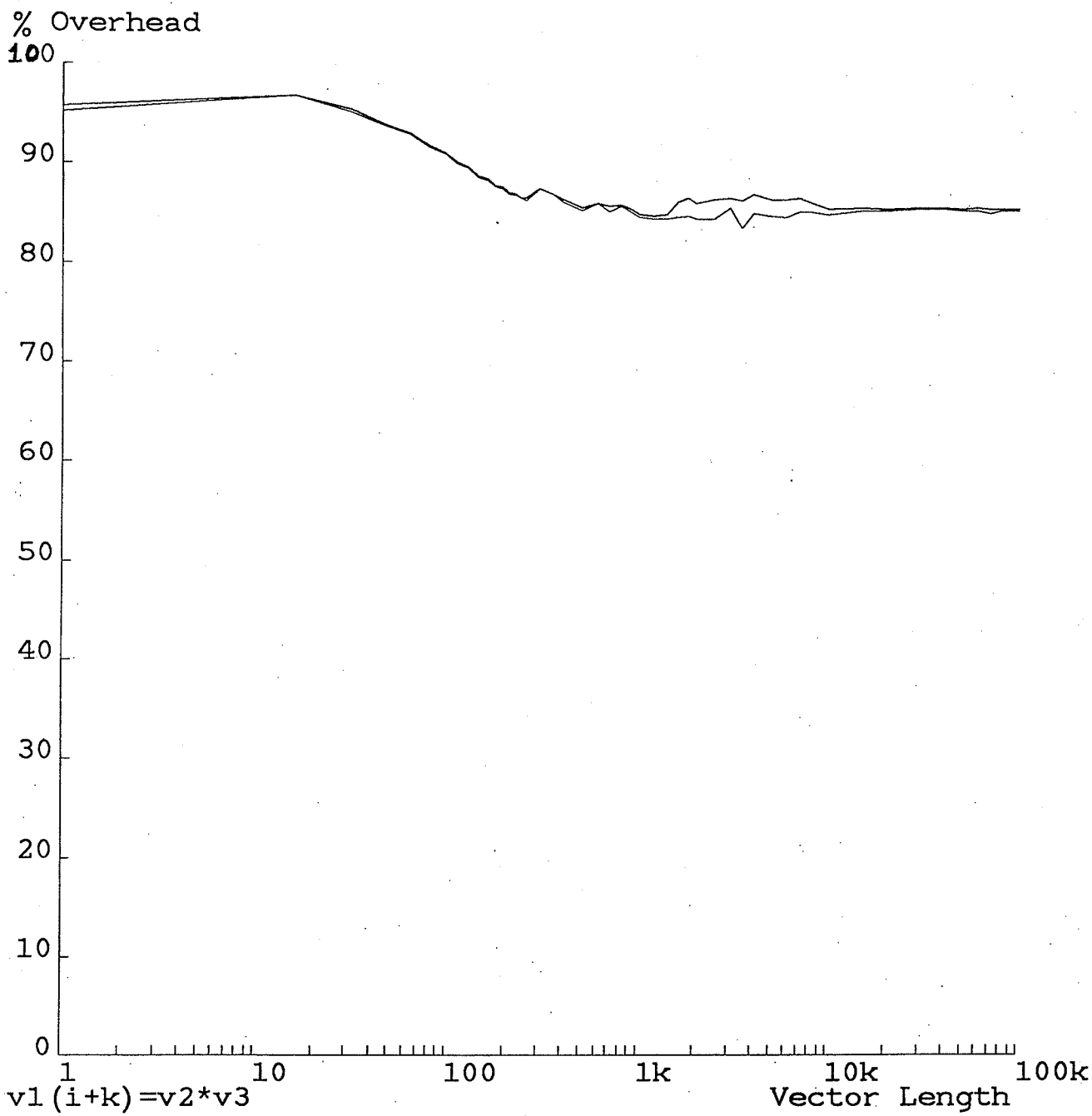


CSRD

Overhead Percentage

Lower/Upper Bounds

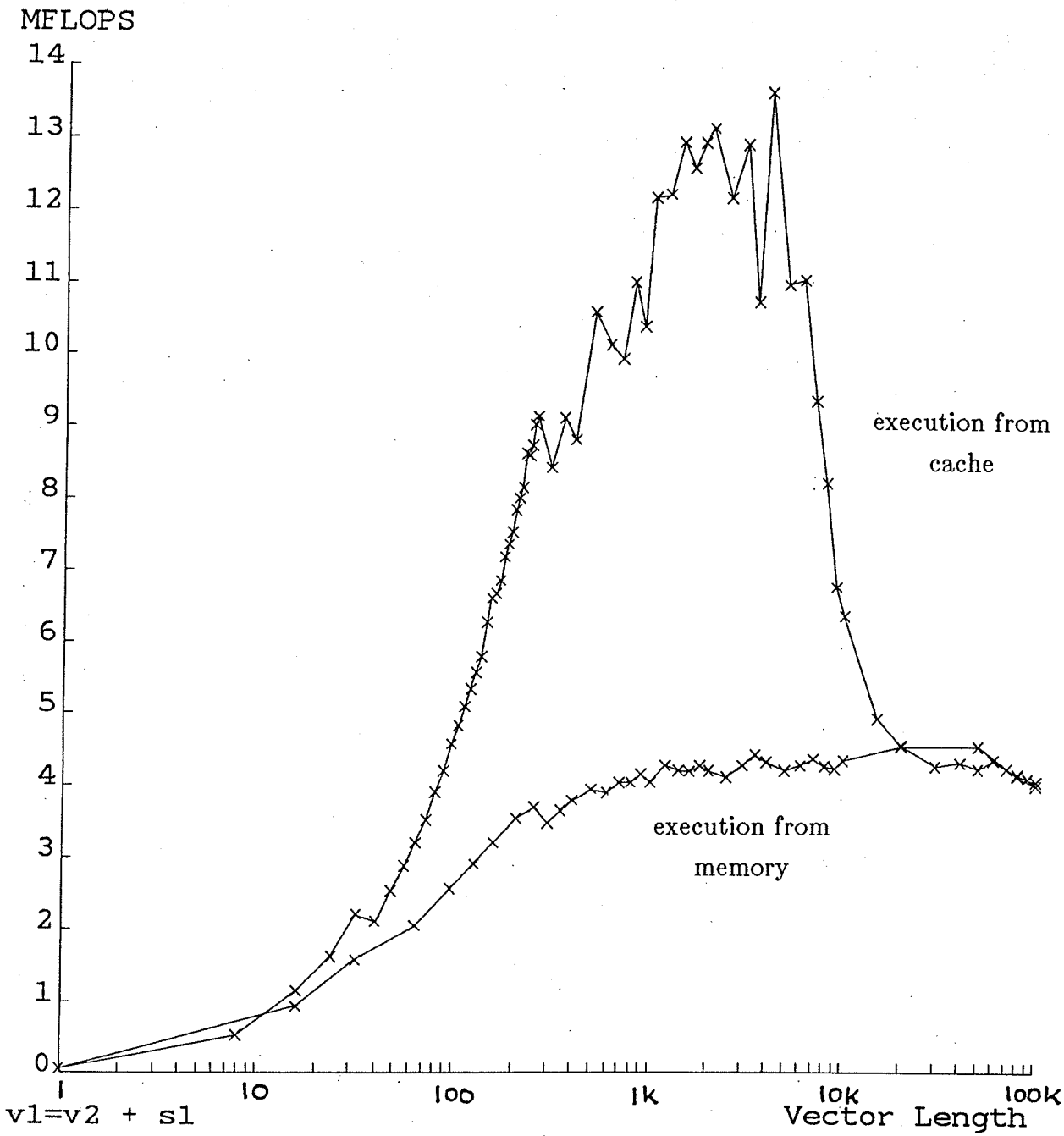

% Overhead
100
90
80
70
60
50
40
30
20
10
0
1
10
100
1k
10k
100k
Vector Length
v1=v2 * v3

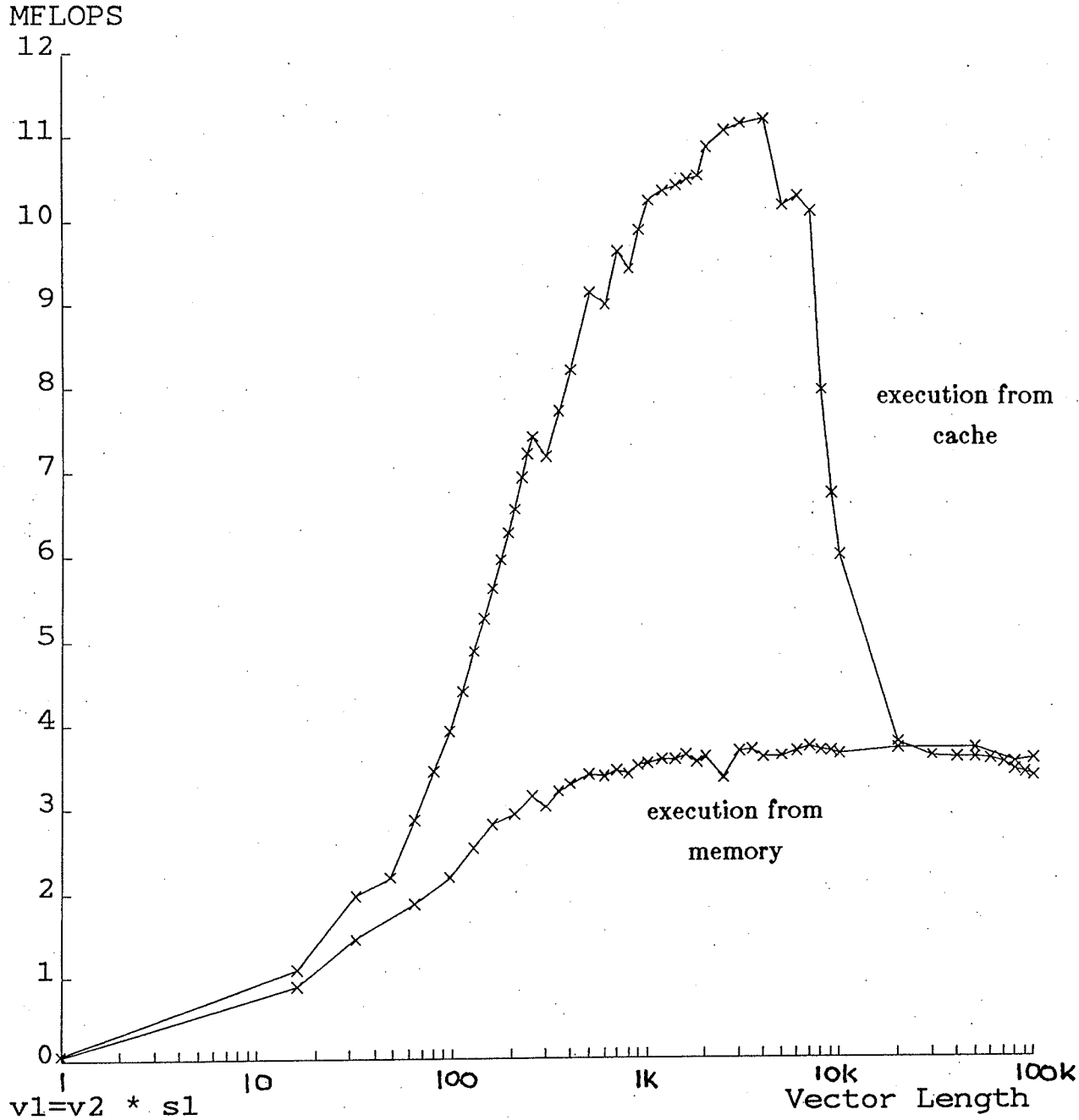

% Overhead
100
90
80
70
60
50
40
30
20
10
0
1
10
100
1k
10k
100k
v1(i+k)=v2*v3
Vector Length

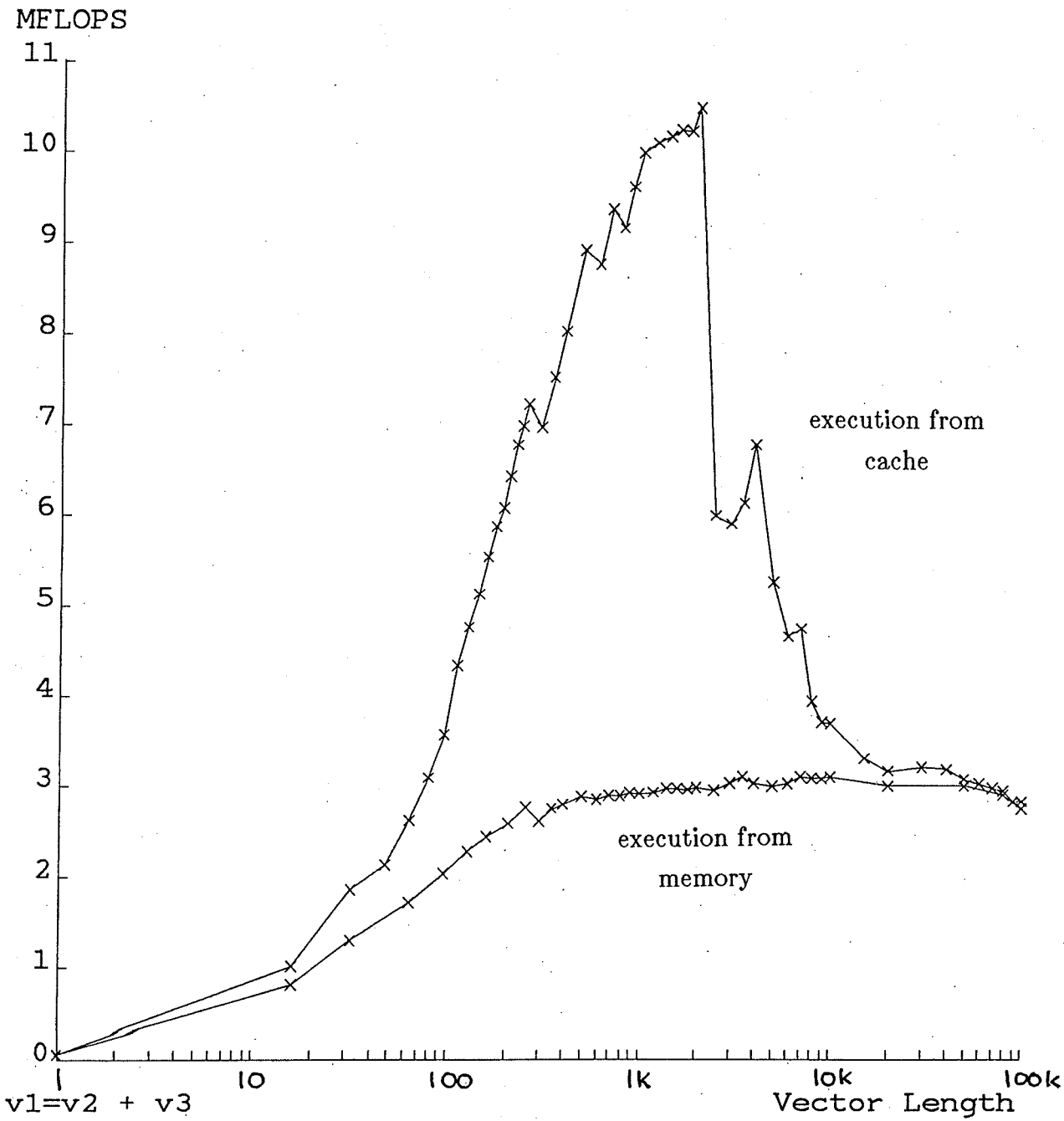
Appendix D

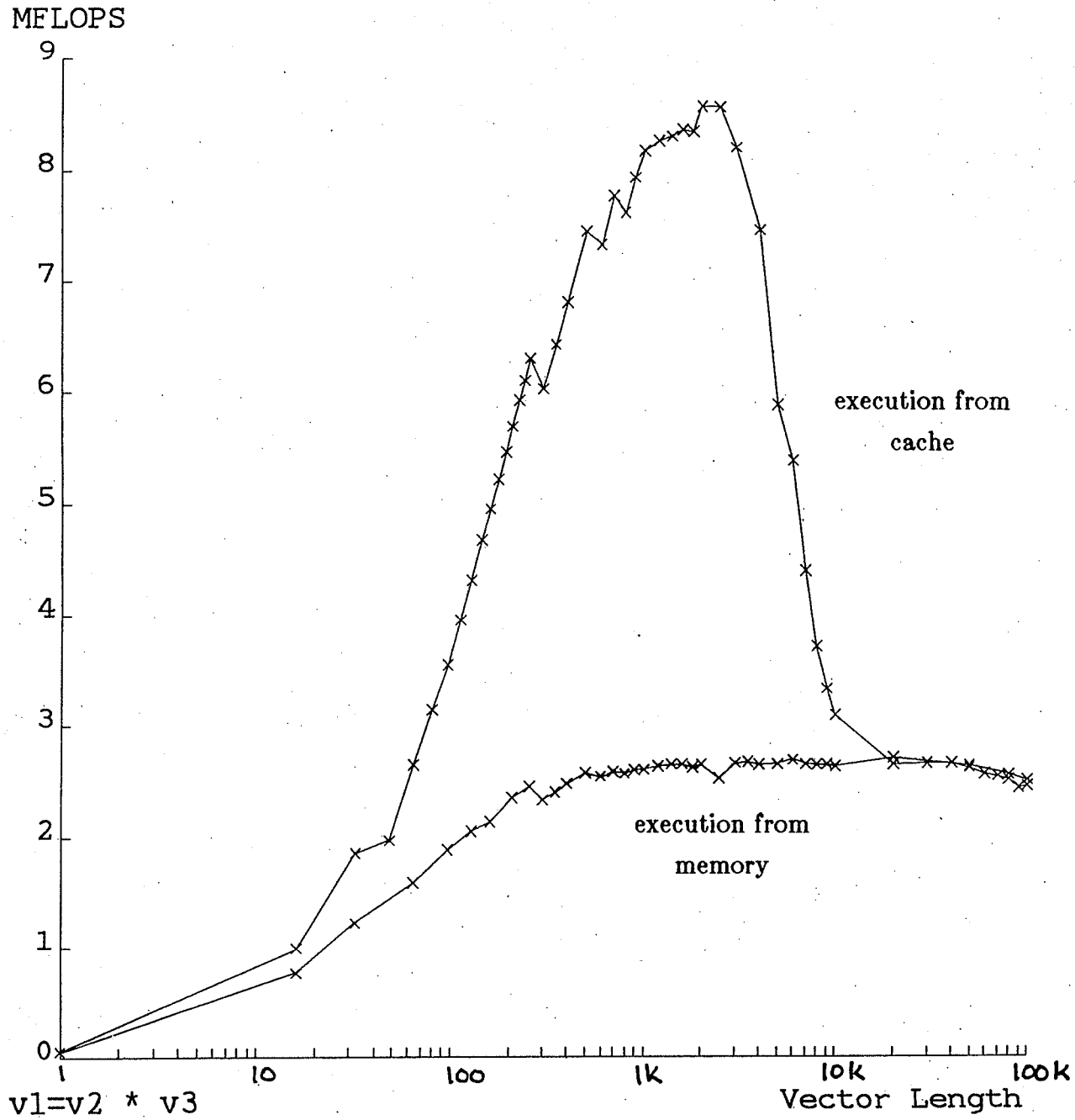
The Execution Rates from Memory and Cache

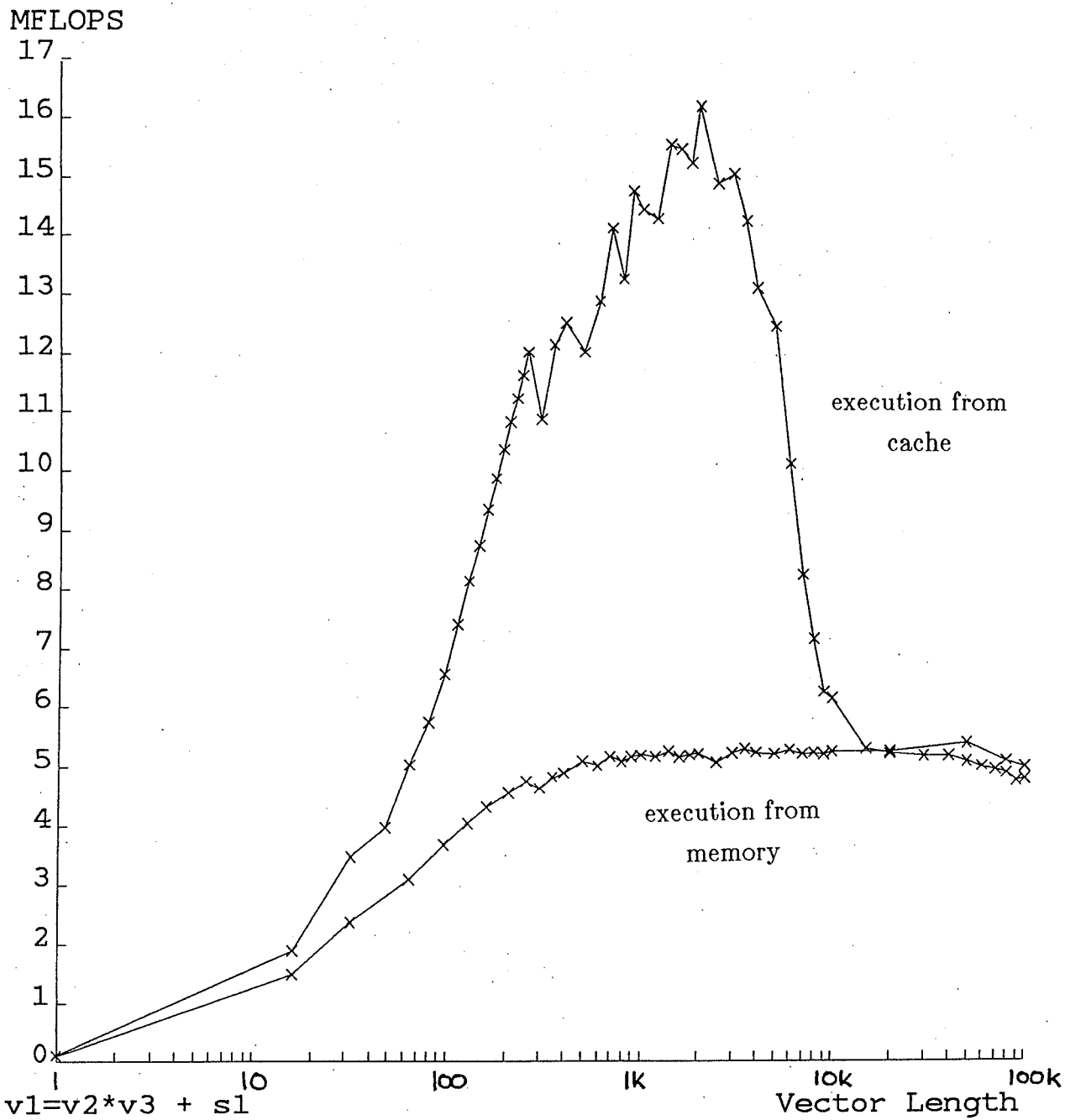
The following plots show the execution rates (in MFLOPS) from the memory and the cache for the kernels not presented in Figure 7 or Figure 8. The vector lengths range from 1 to 100,000.

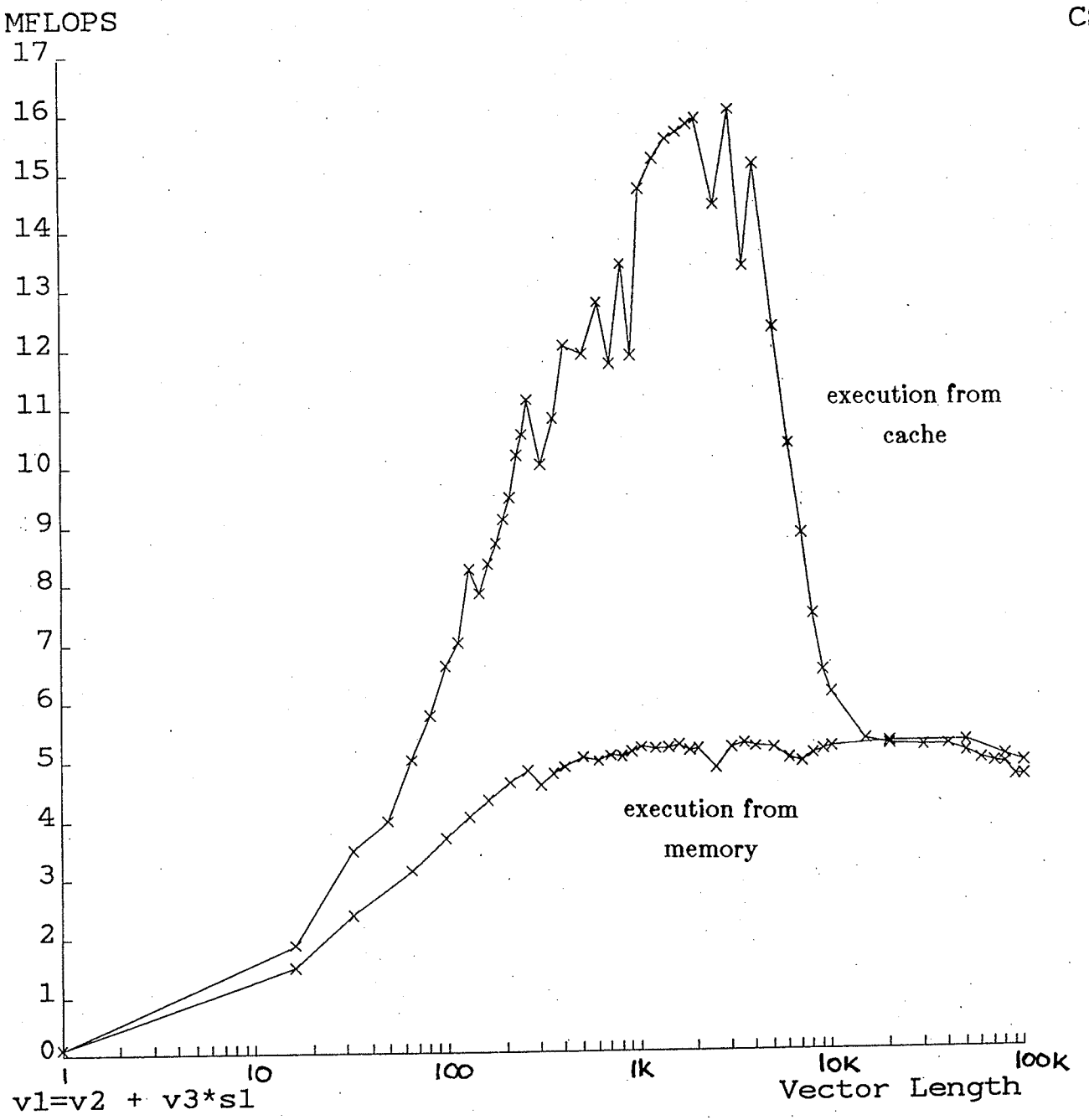

MFLOPS
CSRD
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
execution from
cache
execution from
memory
1
10
100
1k
10k
100k
v1=v2 + s1
Vector Length

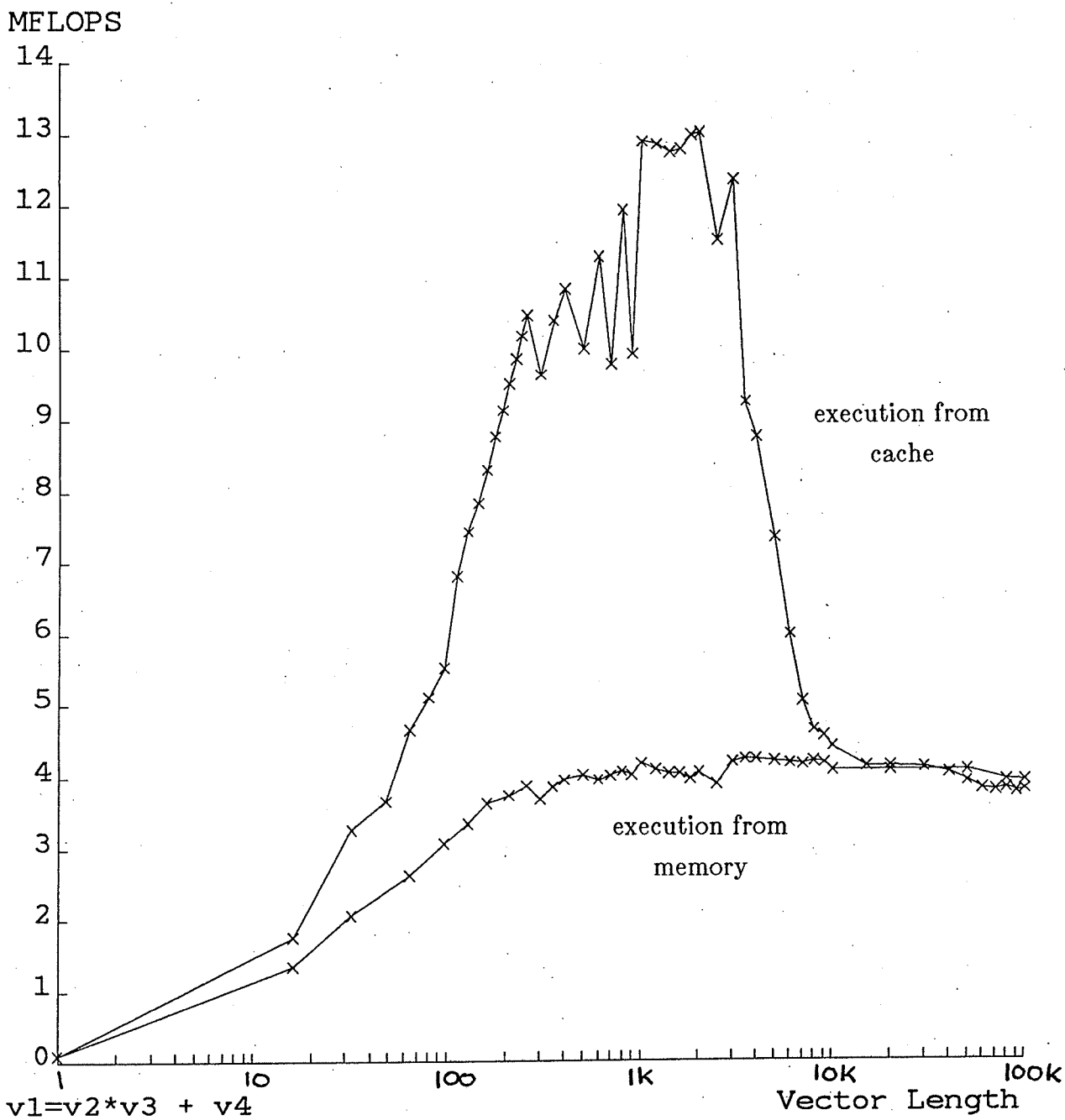

MFLOPS
CSRD
12
11
10
9
8
7
6
5
4
3
2
1
0
execution from
cache
execution from
memory
1
10
100
1k
10k
100k
v1=v2 * s1
Vector Length

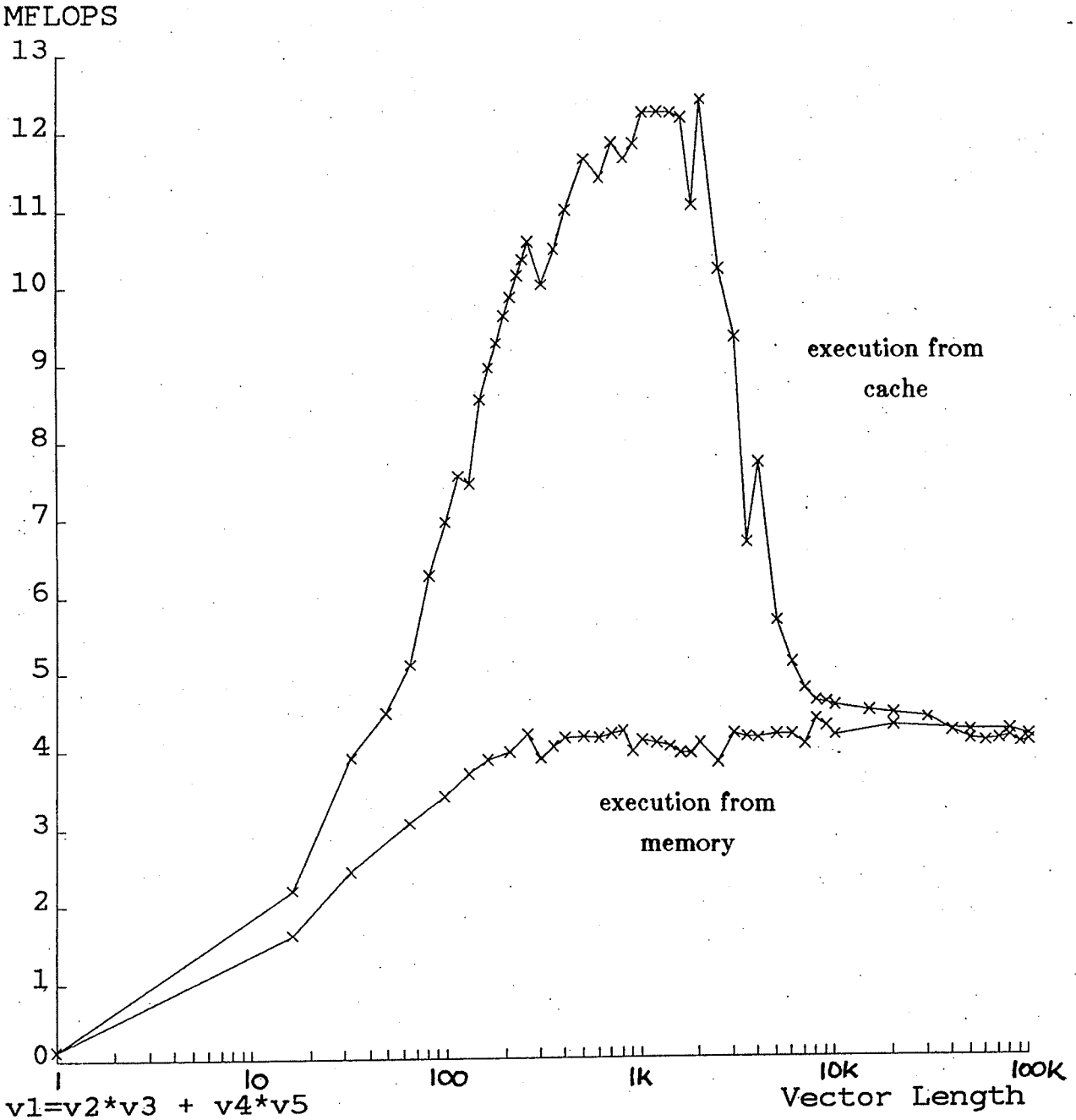

MFLOPS
CSRD
11
10
9
8
7
6
5
4
3
2
1
0
execution from
cache
execution from
memory
1
10
100
1k
10k
100k
v1=v2 + v3
Vector Length

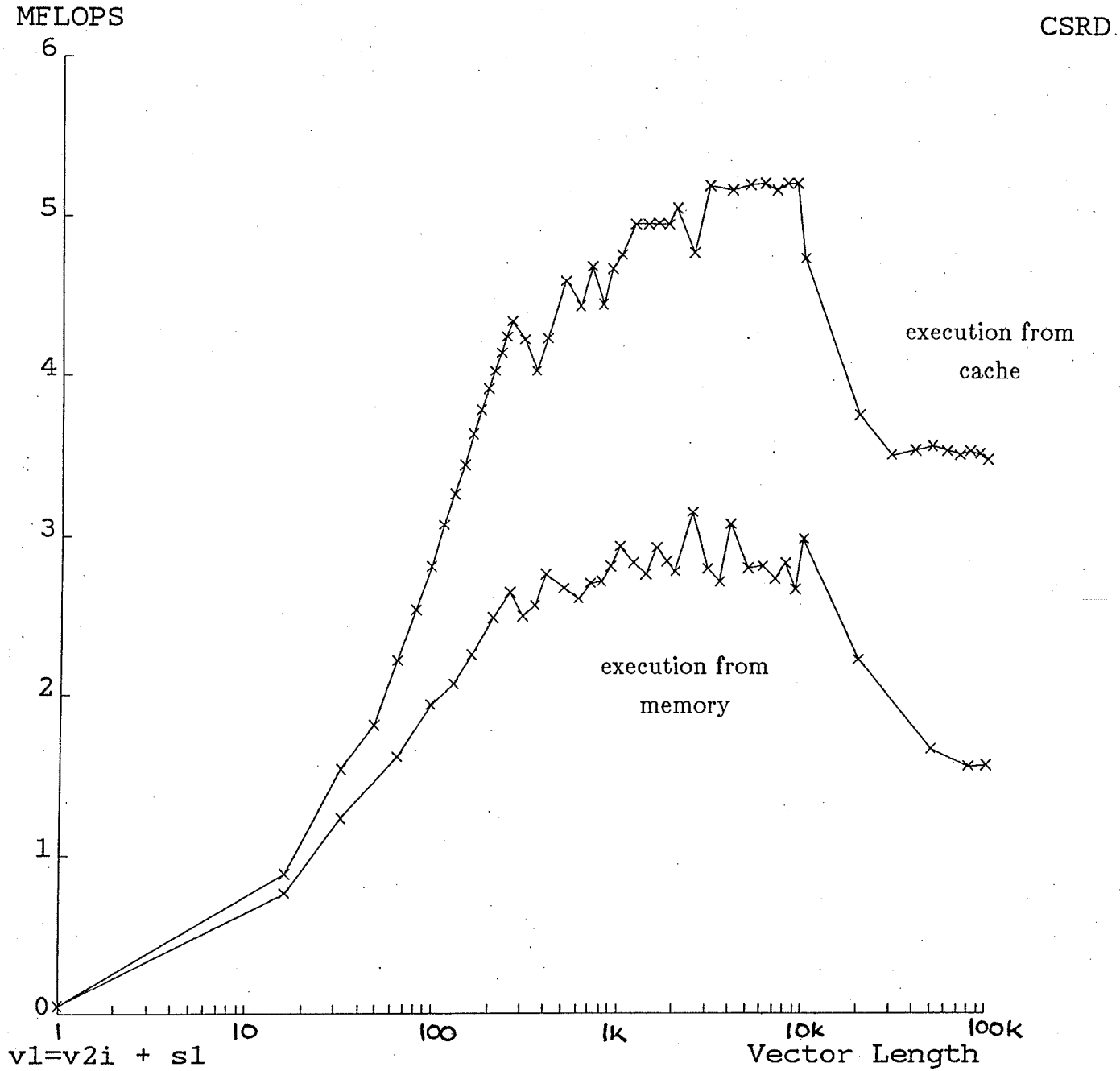

MFLOPS
CSRD
9
8
7
6
5
4
3
2
1
0
execution from
cache
execution from
memory
1
10
100
1k
10k
100k
v1=v2 * v3
Vector Length

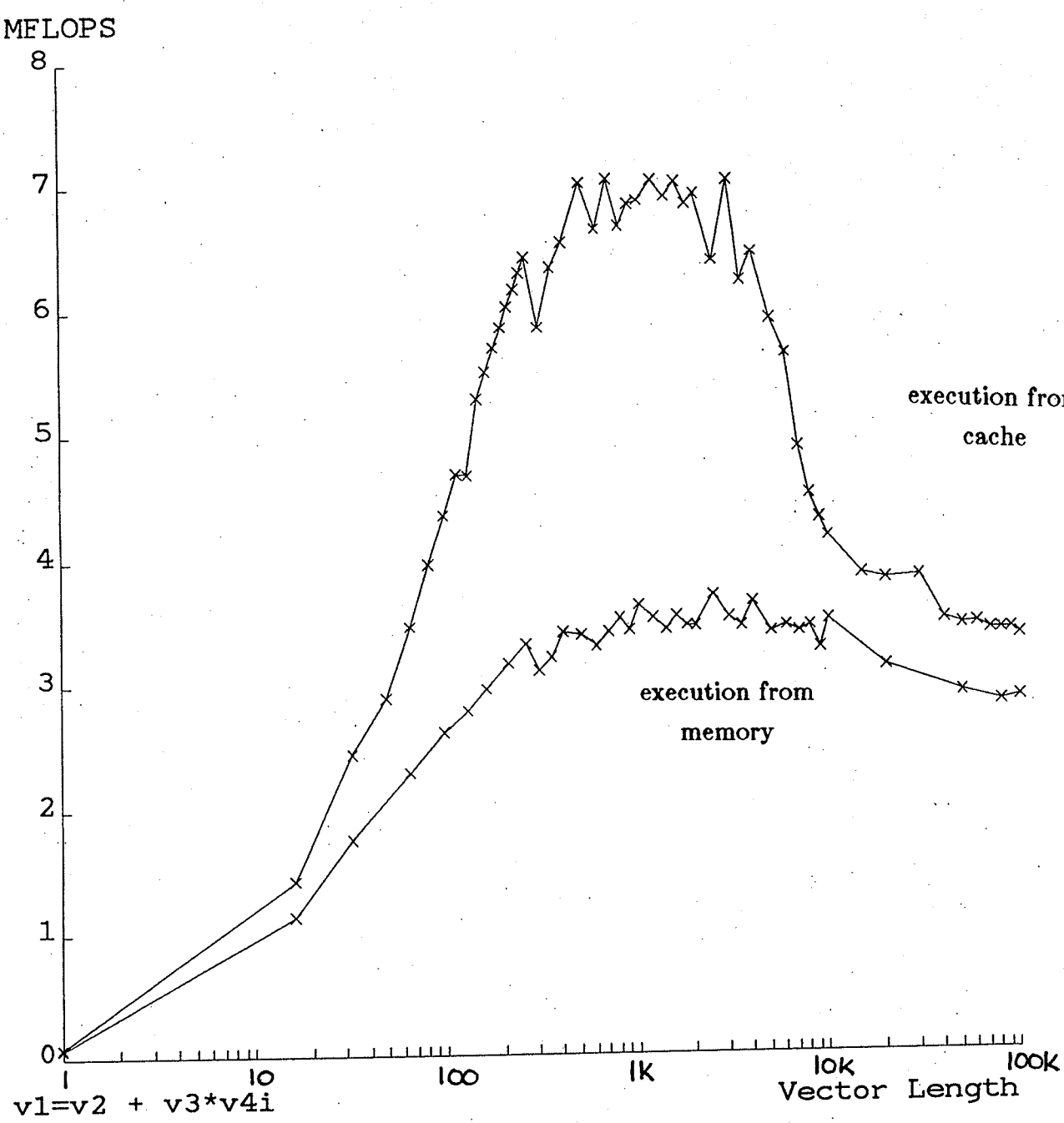

MFLOPS
CSRD
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
execution from
cache
execution from
memory
1
10
100
1k
10k
100k
v1=v2*v3 + s1
Vector Length

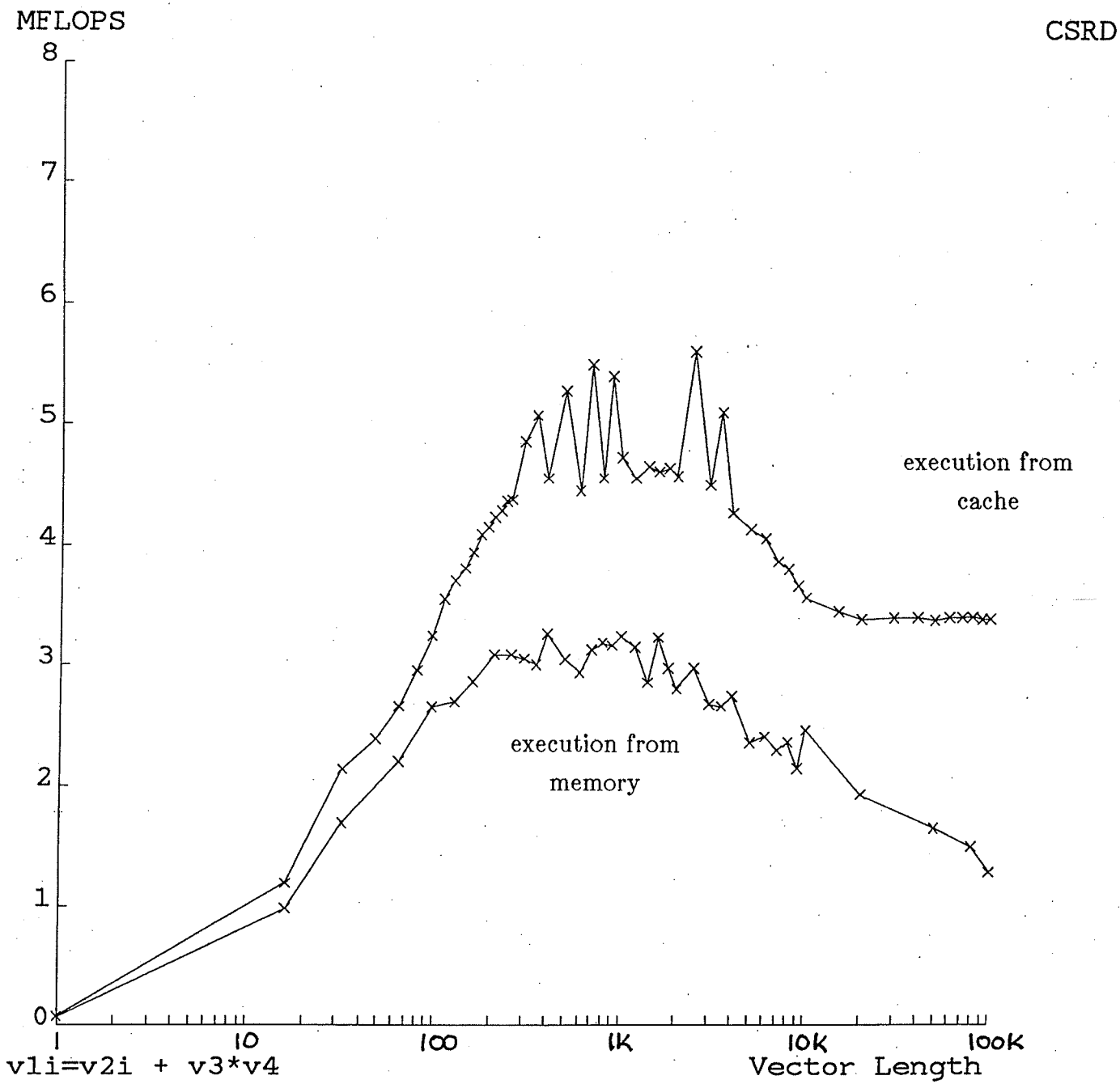

MFLOPS
CSRD
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
execution from
cache
execution from
memory
1
10
100
1k
10k
100k
Vector Length
v1=v2 + v3*s1

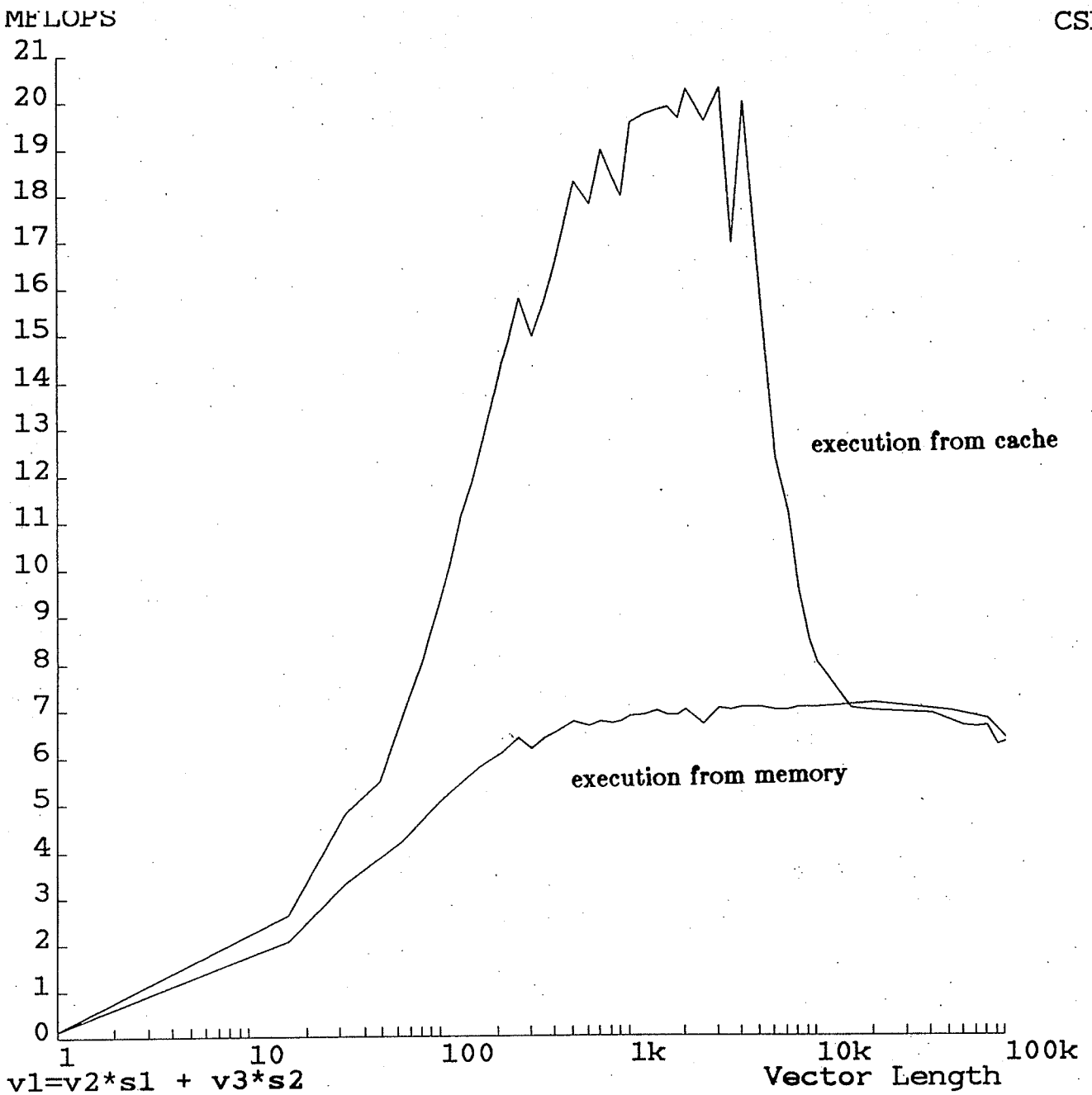

MFLOPS
CSRD
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
execution from
cache
execution from
memory
1
10
100
1k
10k
100k
v1=v2*v3 + v4
Vector Length

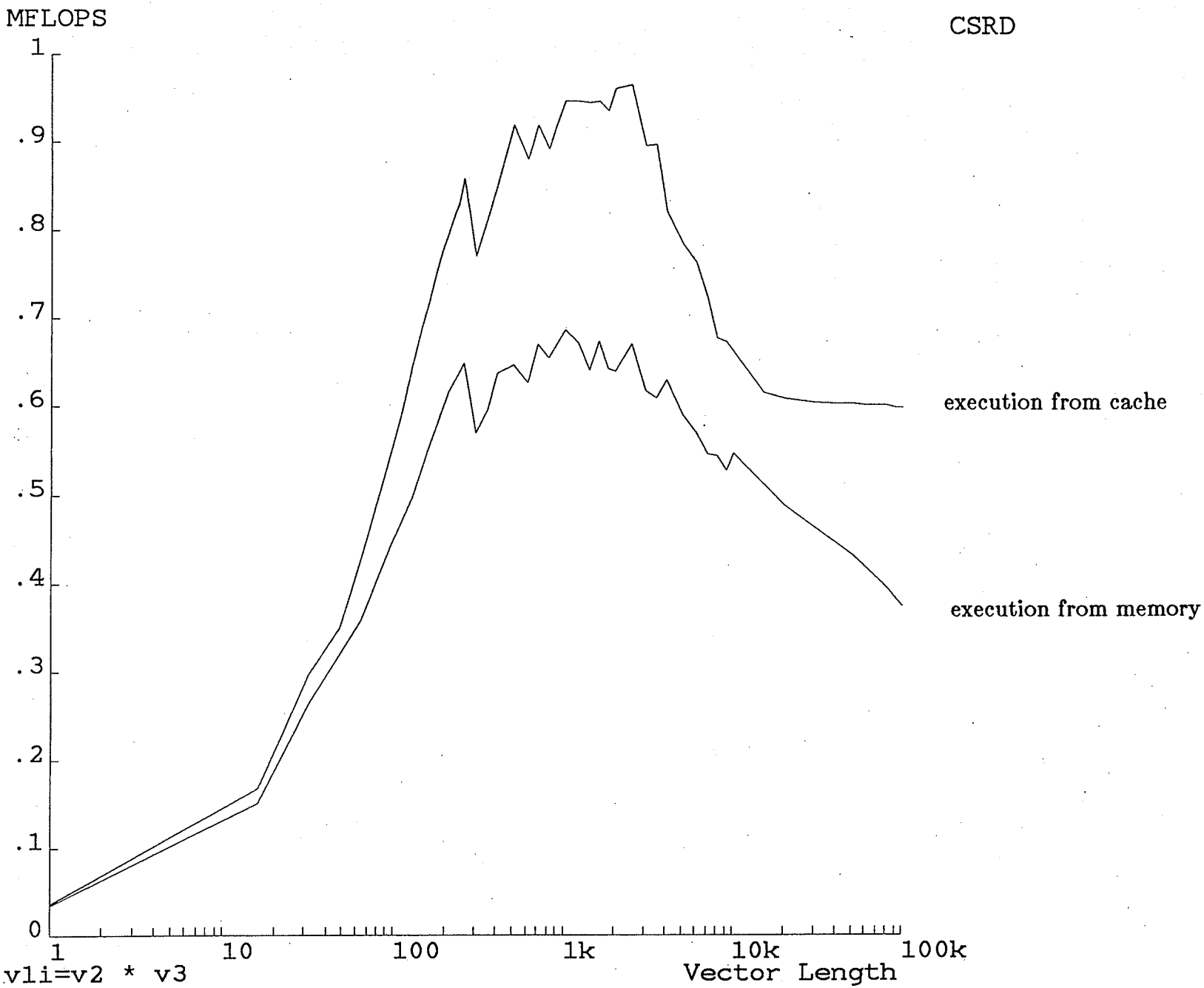

MFLOPS
CSRD
13
12
11
10
9
8
7
6
5
4
3
2
1
0
execution from
cache
execution from
memory
1
10
100
1k
10k
100k
Vector Length
v1=v2*v3 + v4*v5


MFLOPS
CSRD
6
5
4
3
2
1
0
execution from
cache
execution from
memory
1
10
100
1k
10k
100k
v1=v2i + s1
Vector Length


MFLOPS
CSRD
8
7
6
5
4
3
2
1
0
execution from
cache
execution from
memory
1
10
100
1k
10k
100k
Vector Length
v1=v2 + v3*v4i


MFLOPS
CSRD
8
7
6
5
4
3
2
1
0
execution from
cache
execution from
memory
1
10
100
1K
10K
100K
v1i=v2i + v3*v4
Vector Length

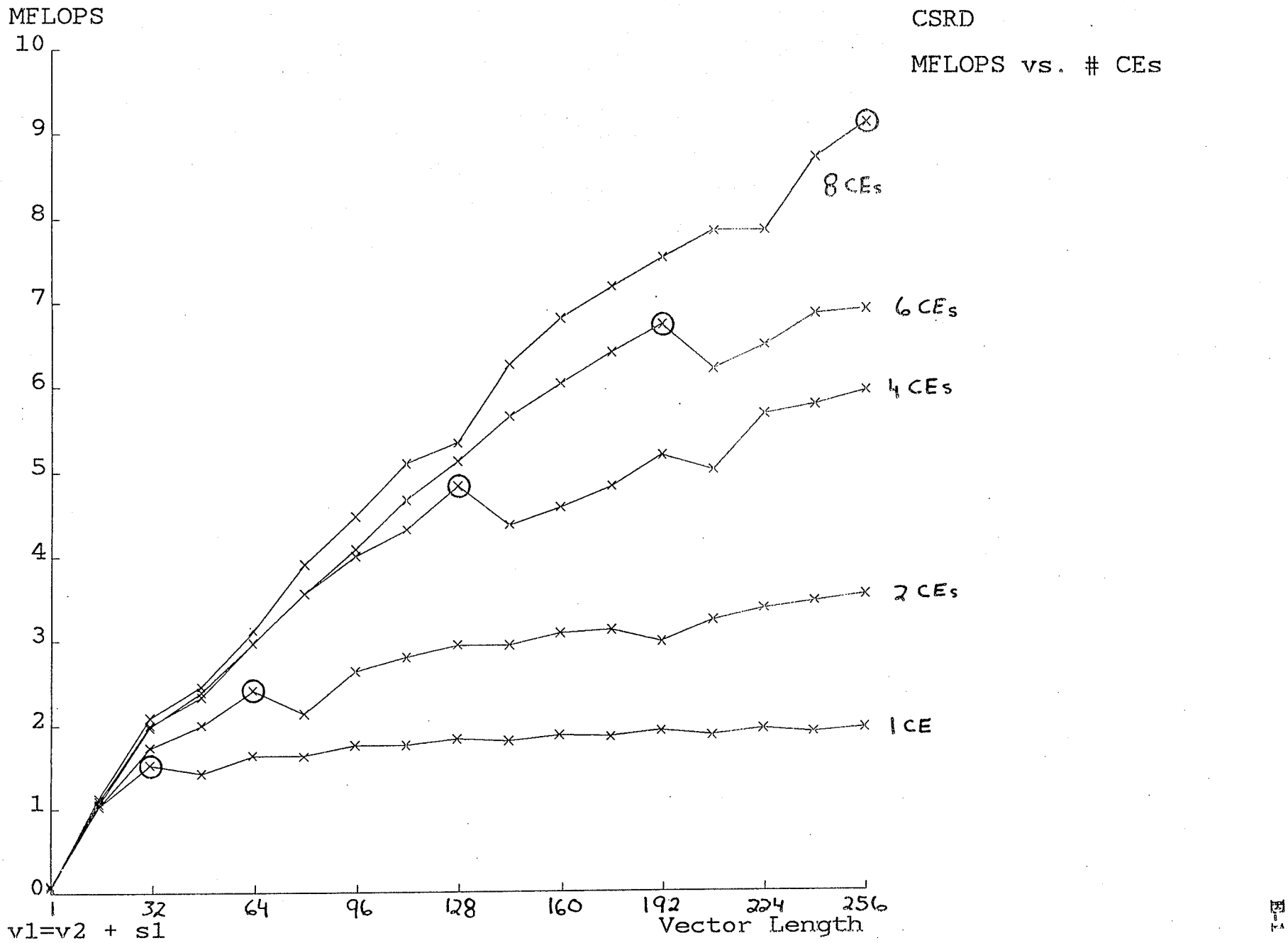

MFLOPS
CSRD
21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
execution from cache
execution from memory
1
10
100
1k
10k
100k
v1=v2*s1 + v3*s2
Vector Length

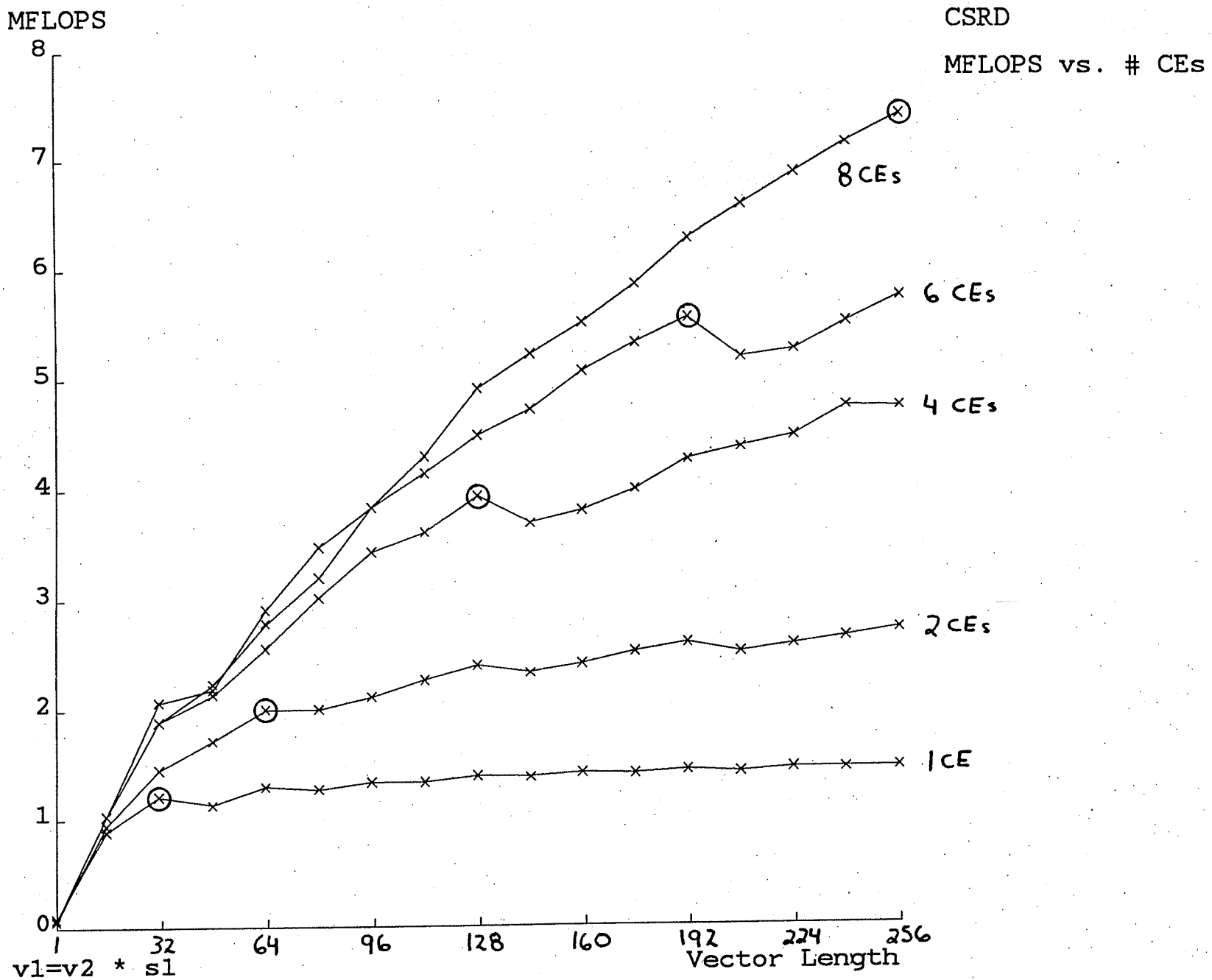

MFLOPS
CSRD
1
.9
.8
.7
.6
.5
.4
.3
.2
.1
0
1
10
100
1k
10k
100k
v1i=v2 * v3
Vector Length
execution from cache
execution from memory

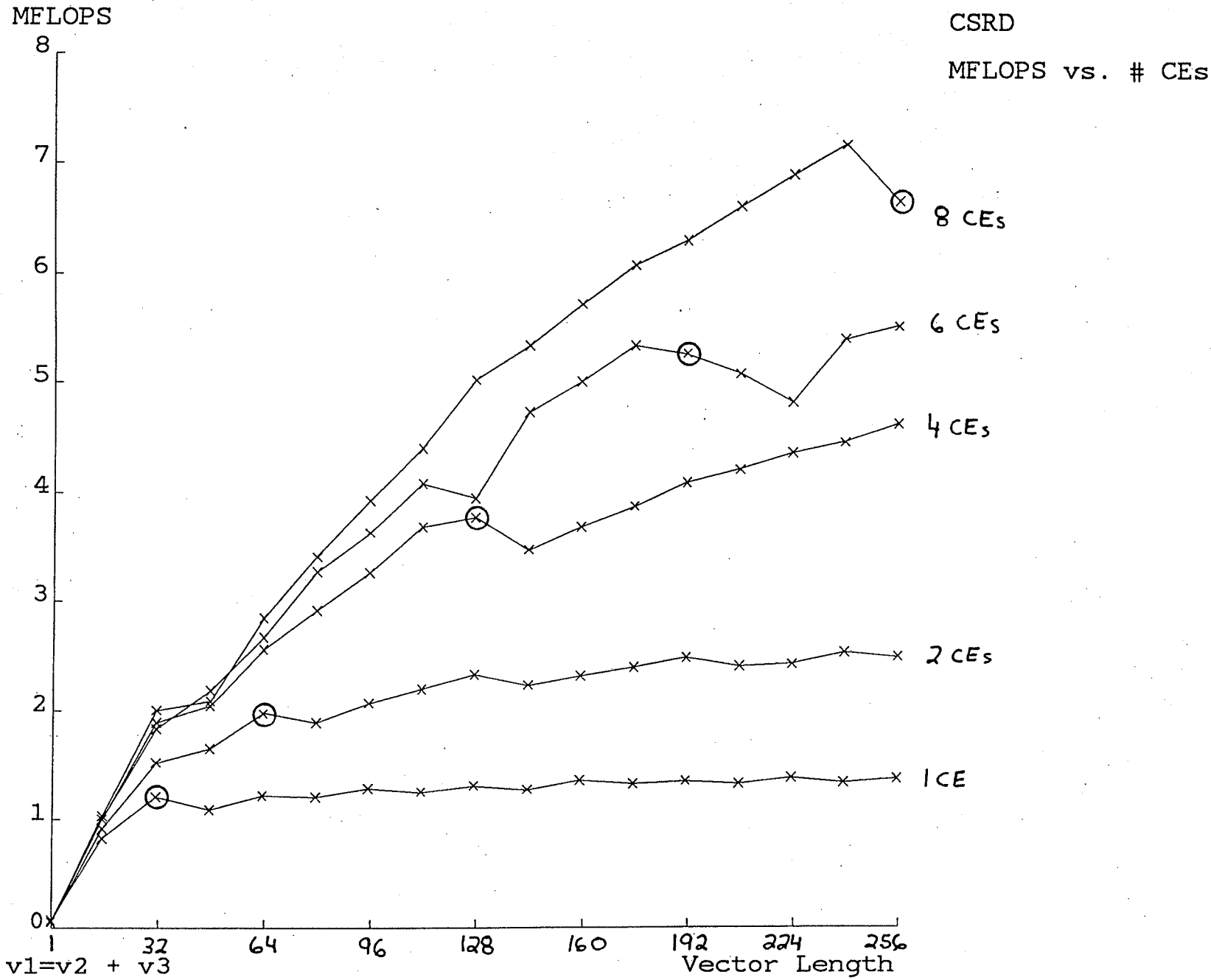
Appendix E

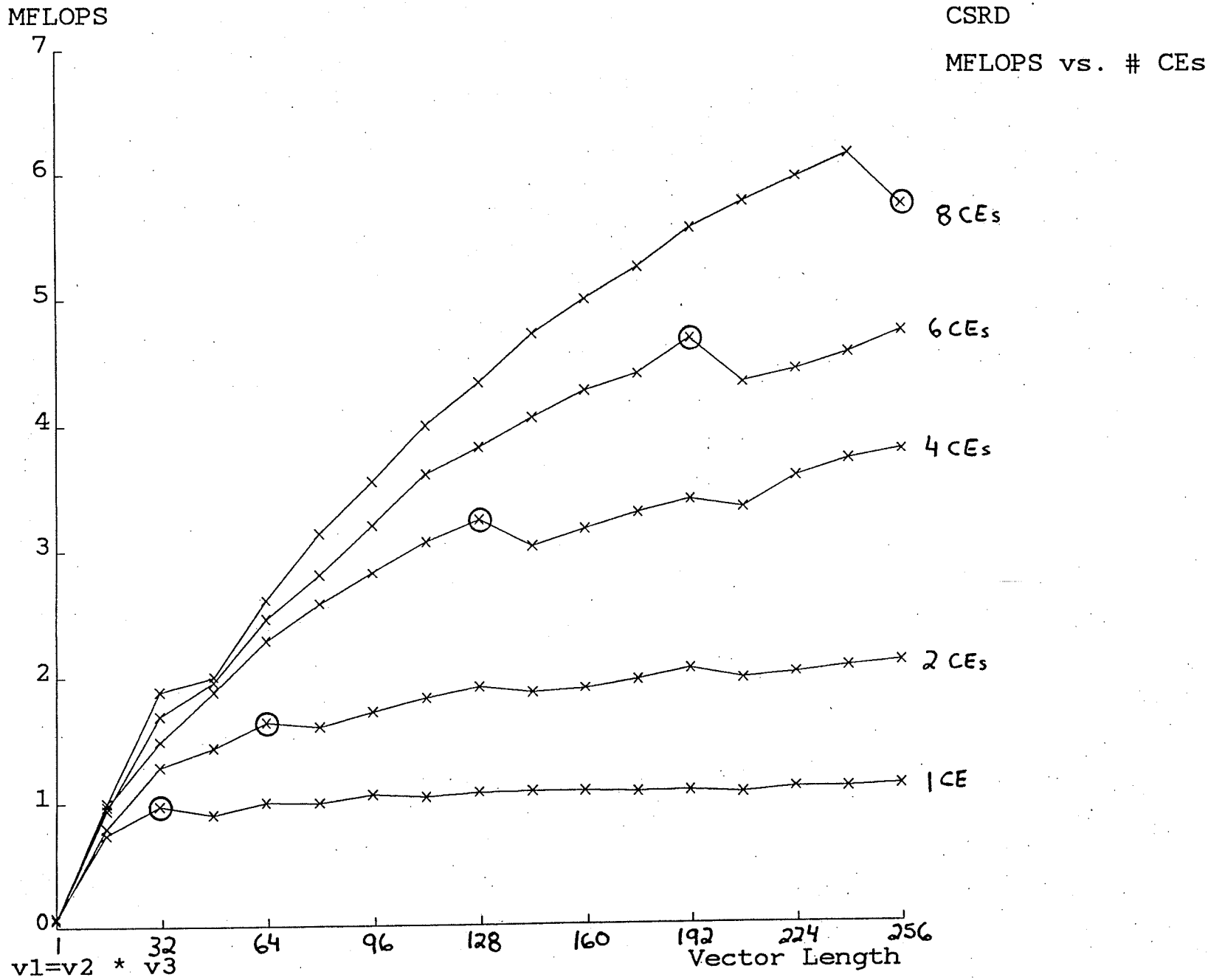
The Execution Rate for 1, 2, 4, 6, and 8 Computational Elements

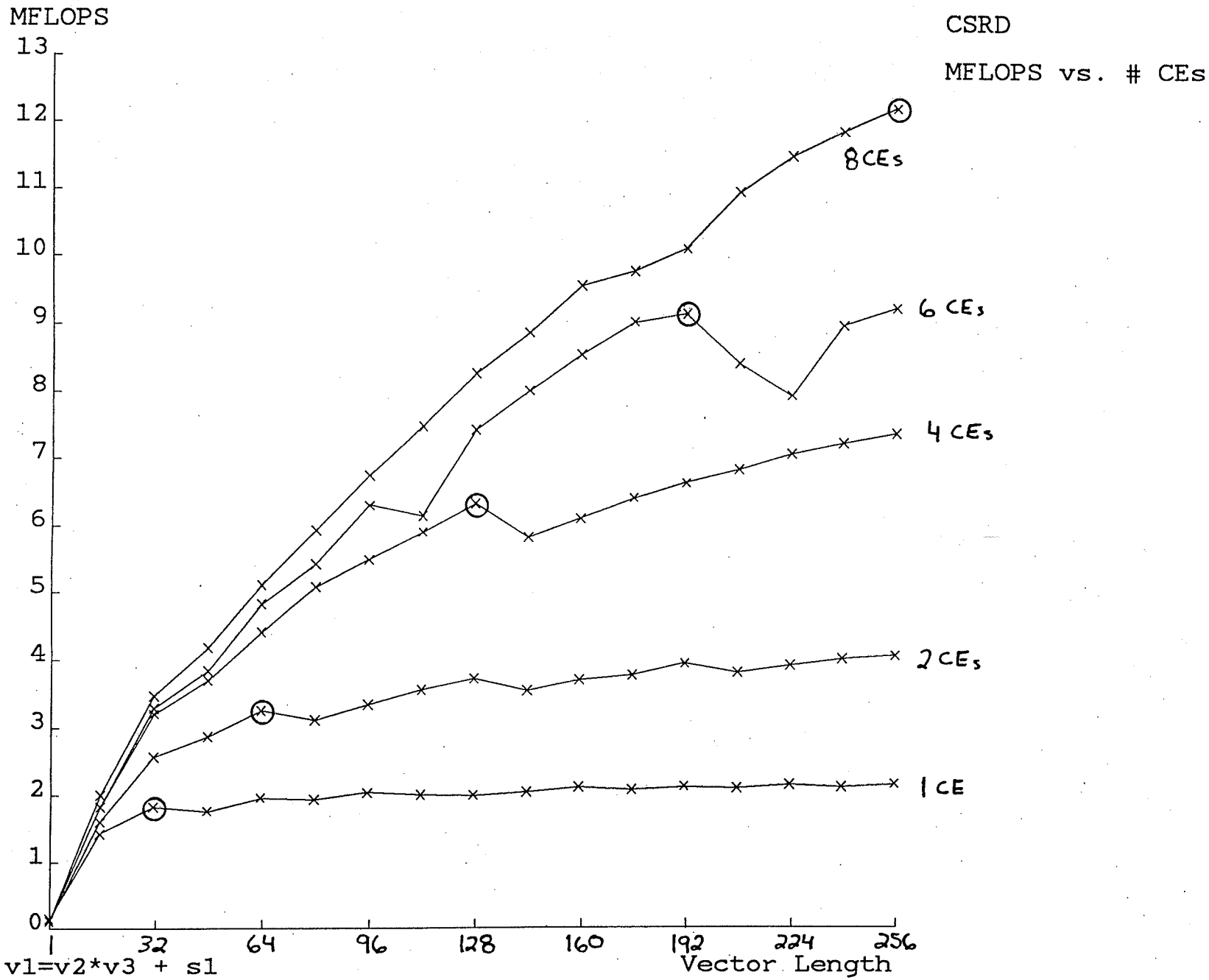
The following plots show the execution rate (in MFLOPS) for 1, 2, 4, 6, and 8 CEs for vector lengths 1 to 256. The circled point on each curve (corresponding to running on 1, 2, 4, 6, or 8 CEs) shows the execution rate at the vector length needed to make use of all the vector register elements in each CE. For most kernels, these points form a straight line indicating linear performance improvement as the number of CEs increases.

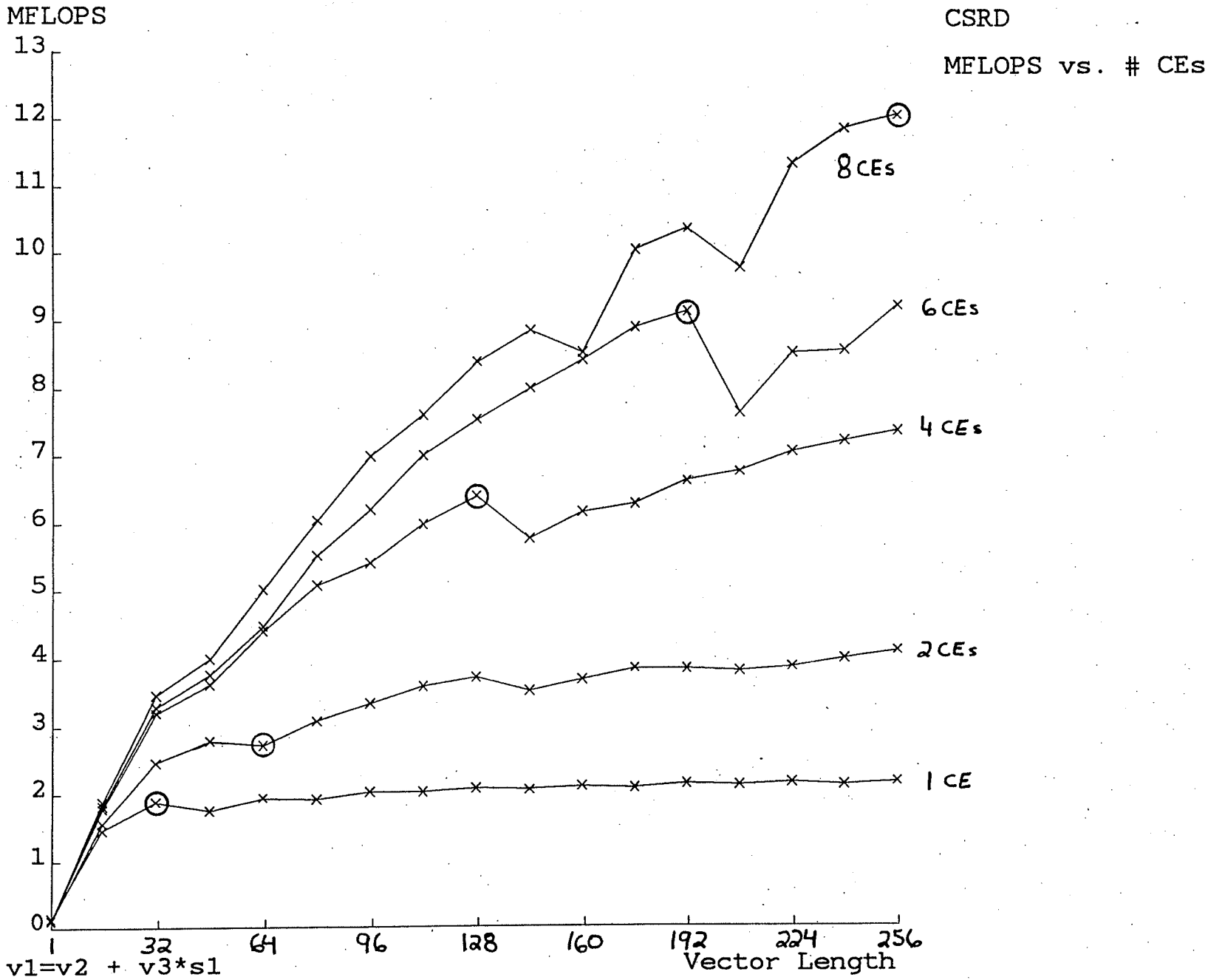

MFLOPS
10
9
8
7
6
5
4
3
2
1
0
1
32
64
96
128
160
192
224
256
Vector Length
v1=v2 + s1
CSRD
MFLOPS vs. # CEs
8 CEs
6 CEs
4 CEs
2 CEs
1 CE

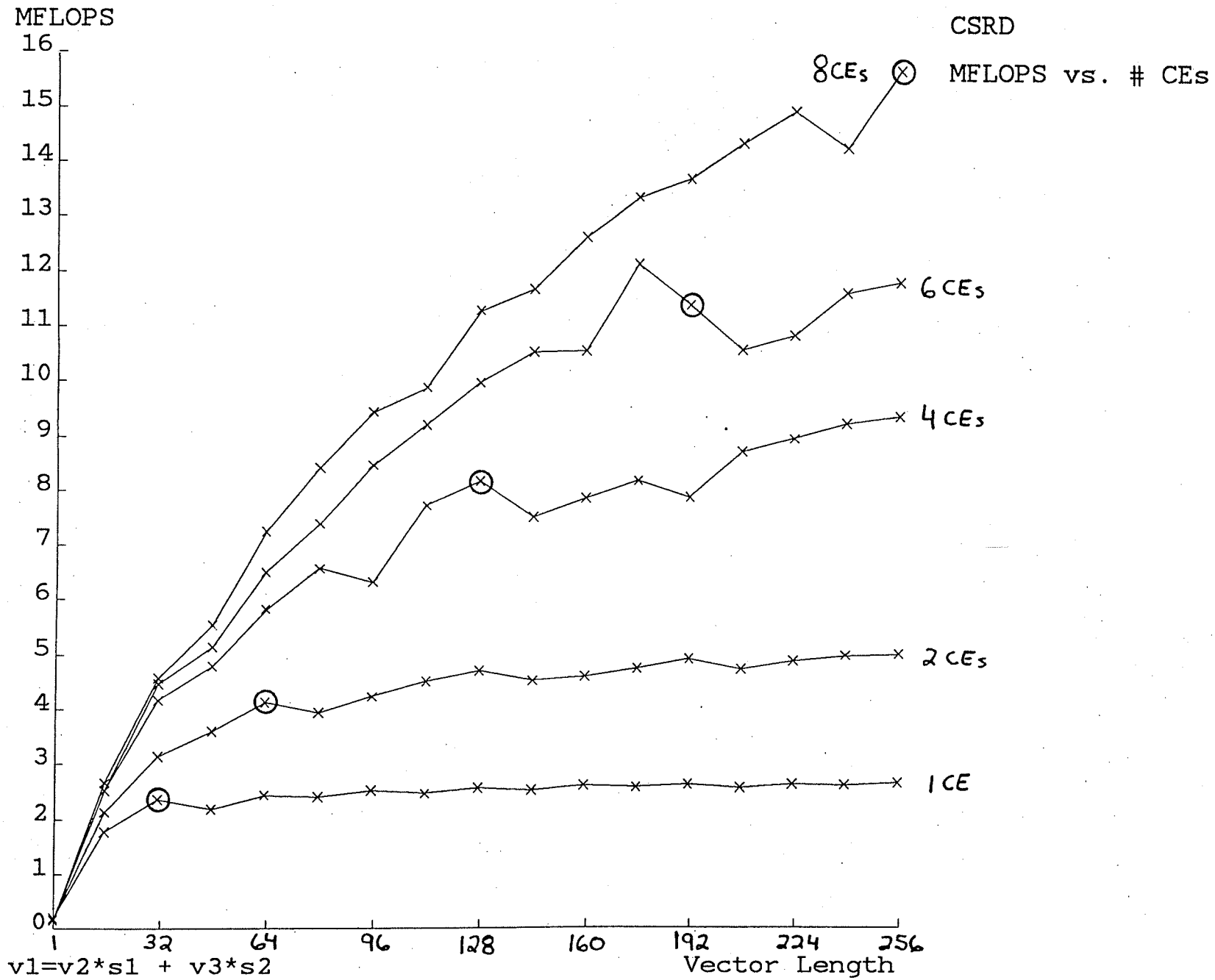

MFLOPS
CSRD
MFLOPS vs. # CEs
8
7
6
5
4
3
2
1
0
8 CEs
6 CEs
4 CEs
2 CEs
1 CE
1
32
64
96
128
160
192
224
256
v1=v2 * s1
Vector Length

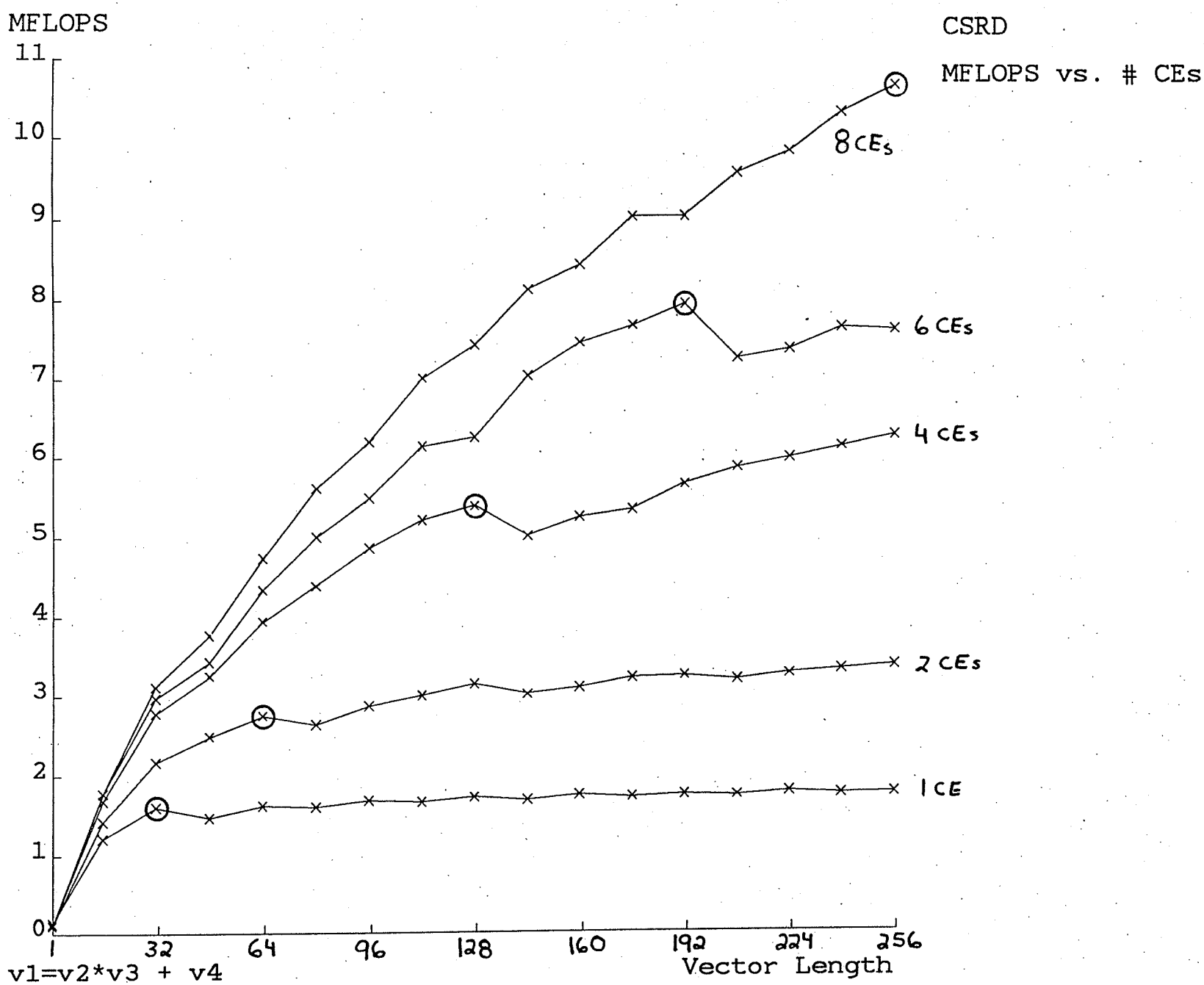

MFLOPS
CSRD
MFLOPS vs. # CEs
8
7
6
5
4
3
2
1
0
8 CEs
6 CEs
4 CEs
2 CEs
1 CE
1
32
64
96
128
160
192
224
256
v1=v2 + v3
Vector Length

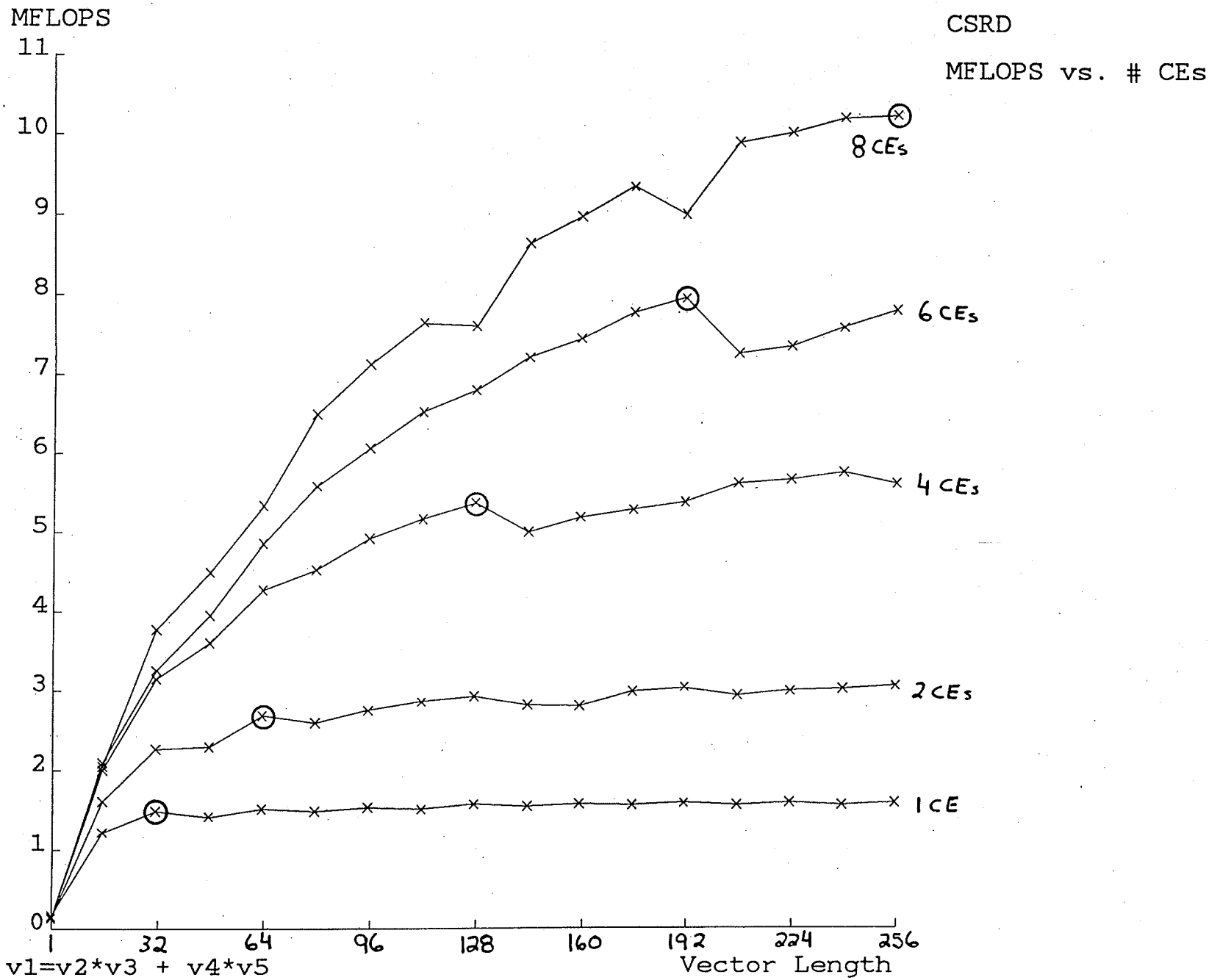

MFLOPS
CSRD
MFLOPS vs. # CEs
7
6
5
4
3
2
1
0
8 CEs
6 CEs
4 CEs
2 CEs
1 CE
1
32
64
96
128
160
192
224
256
v1=v2 * v3
Vector Length

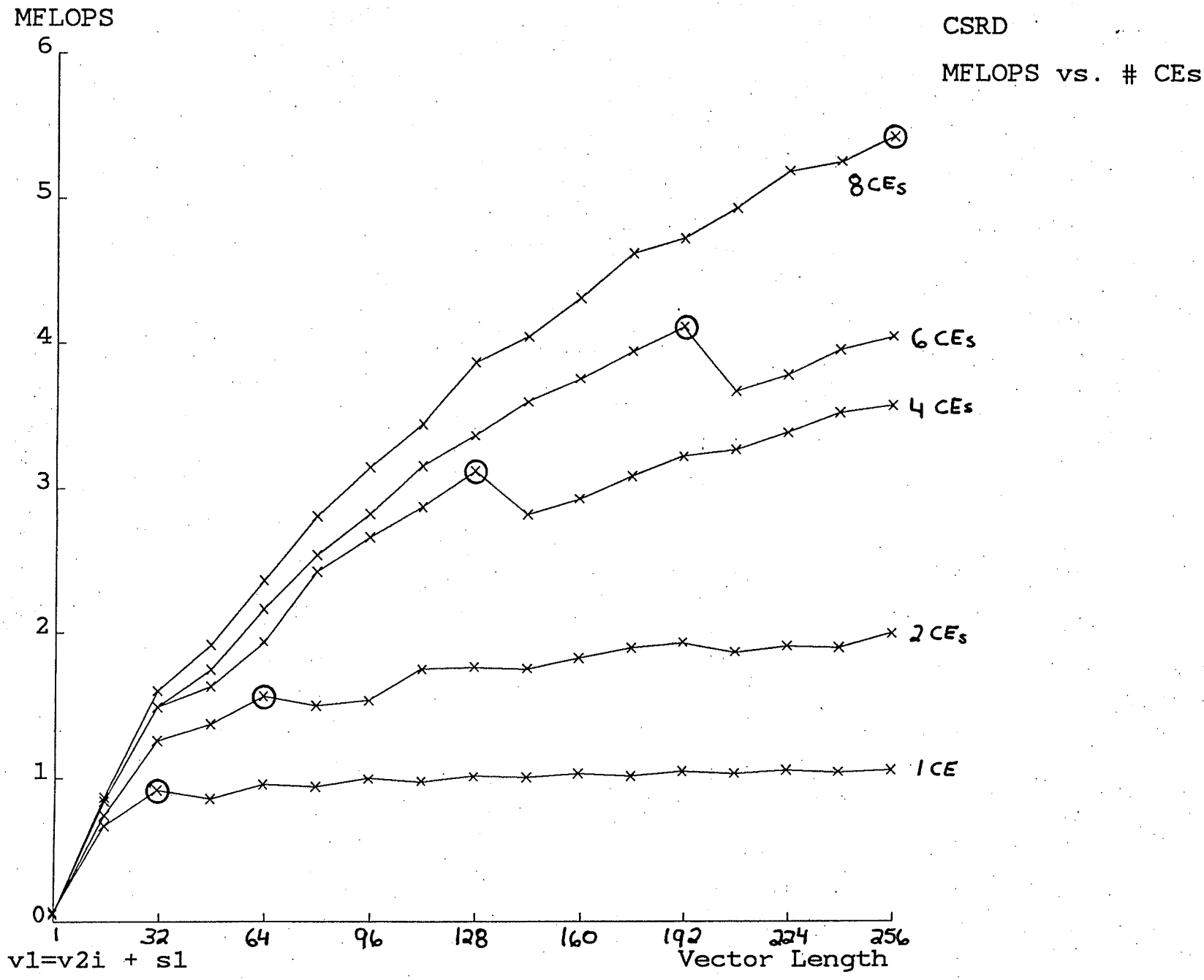

MFLOPS
13
12
11
10
9
8
7
6
5
4
3
2
1
0
1
32
64
96
128
160
192
224
256
v1=v2*v3 + s1
Vector Length
CSRD
MFLOPS vs. # CEs
8 CEs
6 CEs
4 CEs
2 CEs
1 CE

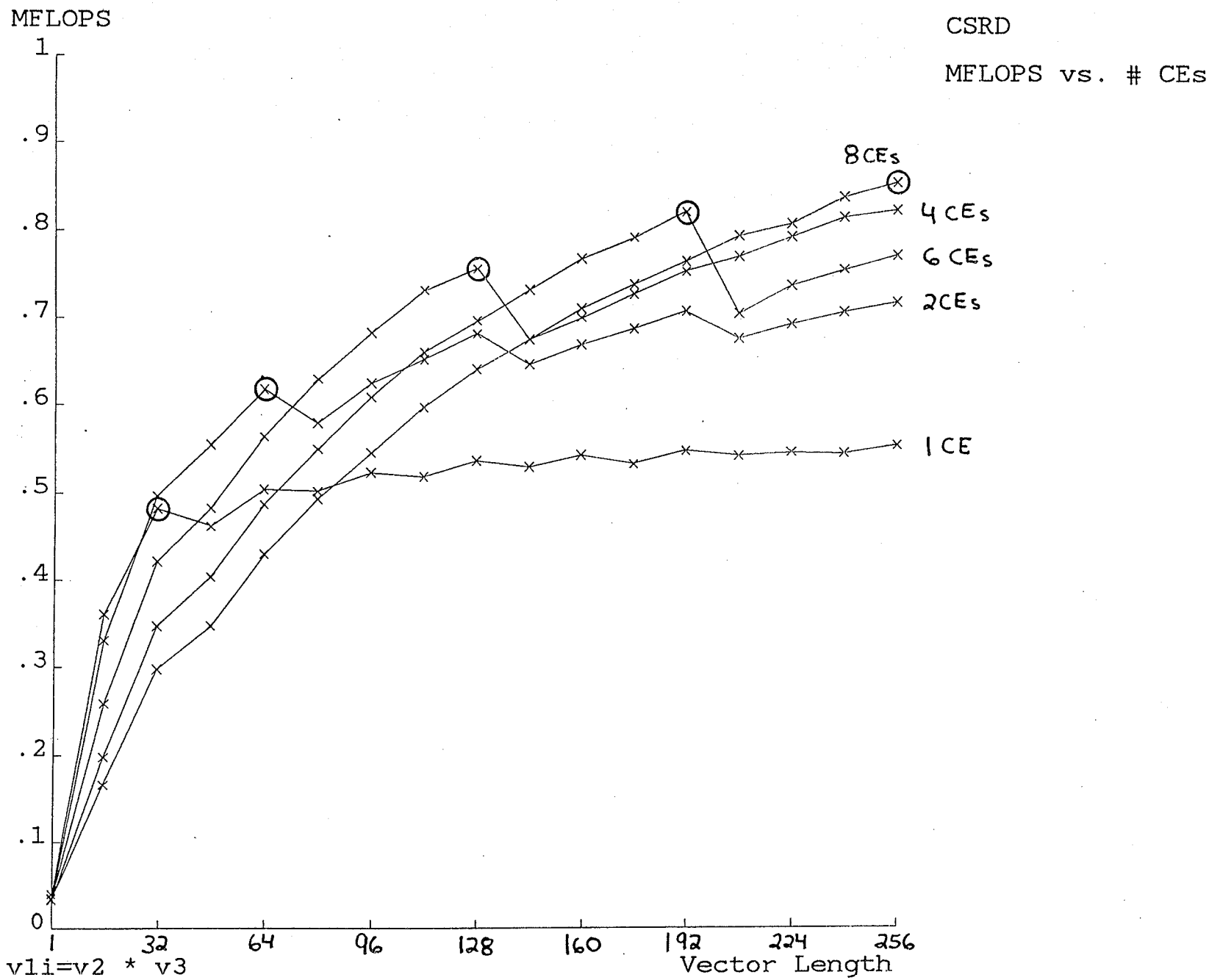

MFLOPS
CSRD
MFLOPS vs. # CEs
13
12
11
10
9
8
7
6
5
4
3
2
1
0
8 CEs
6 CEs
4 CEs
2 CEs
1 CE
1
32
64
96
128
160
192
224
256
v1=v2 + v3*s1
Vector Length


MFLOPS
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
1
32
64
96
128
160
192
224
256
v1=v2*s1 + v3*s2
Vector Length
CSRD
MFLOPS vs. # CEs
8 CEs
6 CEs
4 CEs
2 CEs
1 CE


MFLOPS
CSRD
MFLOPS vs. # CEs
11
10
9
8
7
6
5
4
3
2
1
0
8 CEs
6 CEs
4 CEs
2 CEs
1 CE
1
32
64
96
128
160
192
224
256
Vector Length
v1=v2*v3 + v4

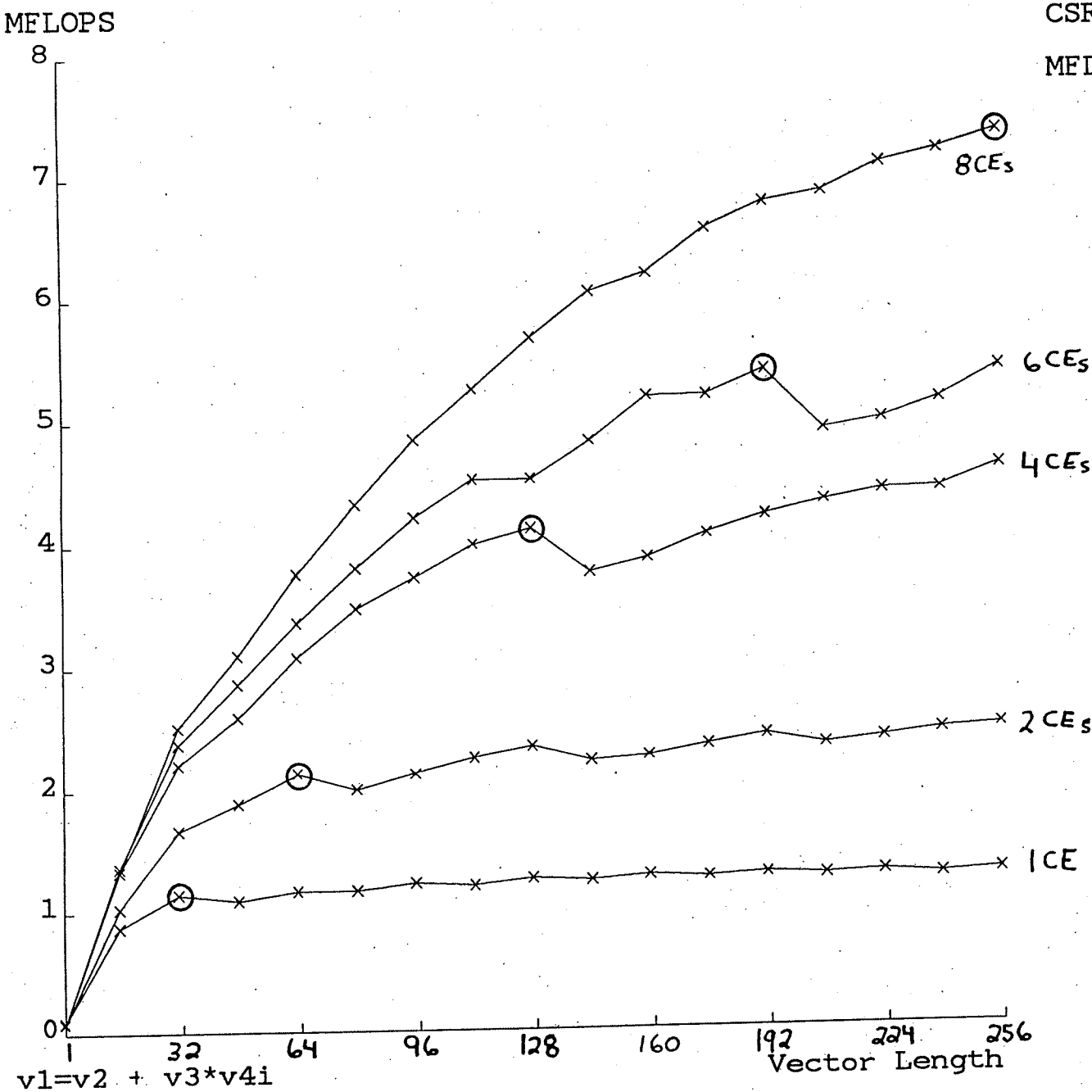

MFLOPS
CSRD
MFLOPS vs. # CEs
11
10
9
8
7
6
5
4
3
2
1
0
8 CEs
6 CEs
4 CEs
2 CEs
1 CE
1
32
64
96
128
160
192
224
256
v1=v2*v3 + v4*v5
Vector Length

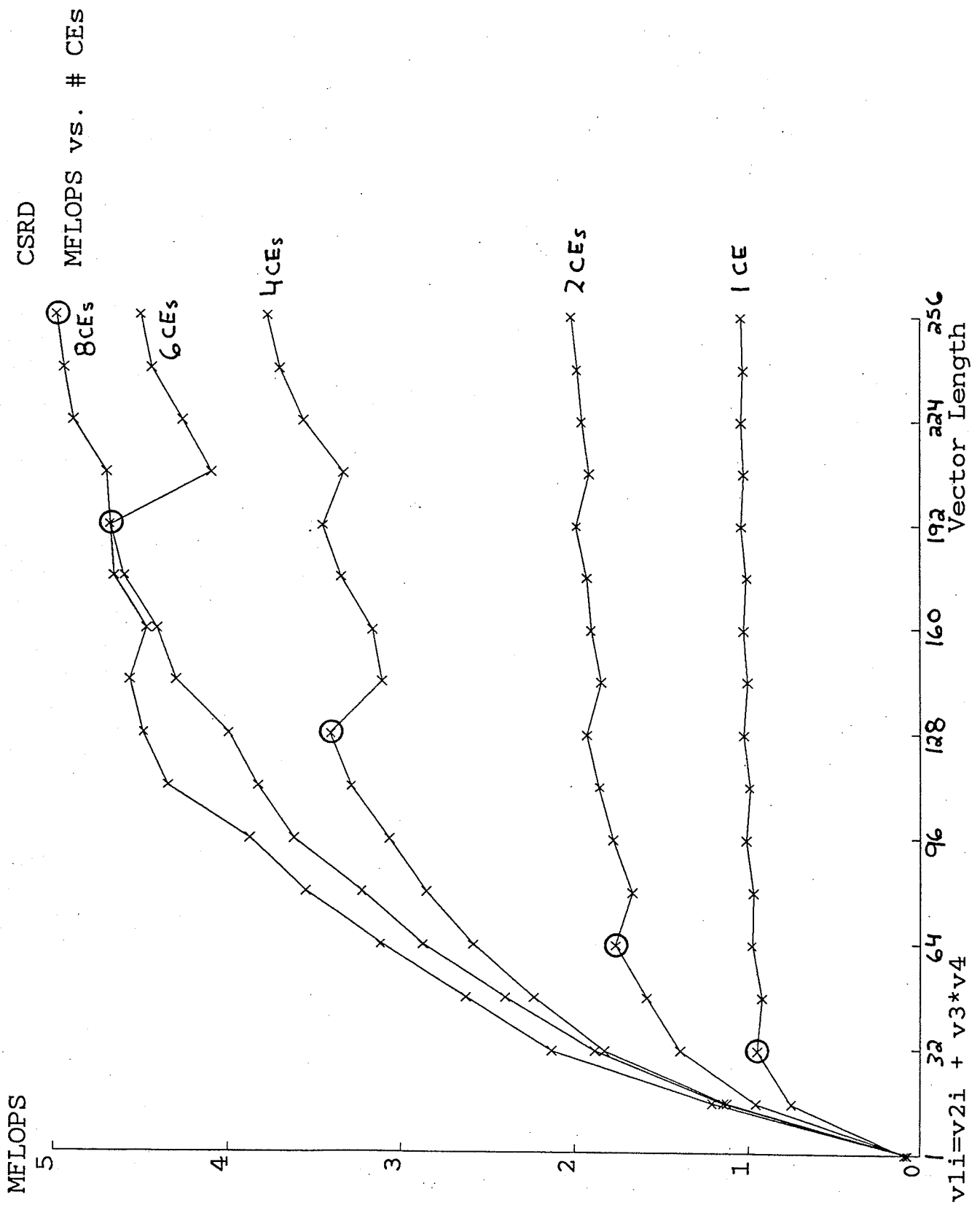

MFLOPS
CSRD
MFLOPS vs. # CEs
6
5
4
3
2
1
0
8 CEs
6 CEs
4 CEs
2 CEs
1 CE
1
32
64
96
128
160
192
224
256
v1=v2i + s1
Vector Length


MFLOPS
CSRD
MFLOPS vs. # CEs
1
.9
.8
.7
.6
.5
.4
.3
.2
.1
0
8 CEs
4 CEs
6 CEs
2 CEs
1 CE
1
32
64
96
128
160
192
224
256
v1i=v2 * v3
Vector Length

CSRD

MFLOPS vs. # CEs




CSRD
MFLOPS vs. # CEs
MFLOPS
8 CEs
6 CEs
4 CEs
2 CEs
1 CE
0
1
2
3
4
5
32
64
96
128
160
192
224
256
Vector Length
v1i=v2i + v3*v4

BIBLIOGRAPHIC DATA SHEET	1. Report No. CSRD-539	2.	3. Recipient's Accession No.

4. Title and Subtitle

EXPERIMENTAL RESULTS FOR VECTOR PROCESSING ON THE ALLIANT FX/8

5. Report Date: February 11, 1986

6.

7. Author(s)

Walid Abu-Sufah and Allen D. Malony

8. Performing Organization Rept. No. CSRD-539

9. Performing Organization Name and Address

University of Illinois at Urbana-Champaign
Center for Supercomputing Research and Development
Urbana, IL 61801-2932

10. Project/Task/Work Unit No.

11. Contract/Grant No.
US NSF DCR84-10110, US NSF DCR84-06916, US DOE DE-FG02-85ER25001

12. Sponsoring Organization Name and Address

National Science Foundation, Washington, D. C.; and
U. S. Department of Energy, Washington, D. C.

13. Type of Report & Period Covered
Technical Report

14.

15. Supplementary Notes

16. Abstracts

The Alliant FX/8 multiprocessor implements several high-speed computation ideas in software and hardware. Each of the 8 computational elements (CEs) has vector capabilities and multiprocessor support. The FX/8 delivers its highest processing rates when executing vector loops concurrently. In this paper, we present extensive performance results for vector processing on the FX/8. The vector kernels of the LANL BMK8a1 benchmark are used in the experiments. Each kernel is executed in stand-alone mode for vector lengths varying between 1 and 100,000. For each kernel running on 1 and 8 CEs, we show the measured execution rate (in MFLOPS) as a function of vector length. For 1 CE, we show the vector lengths for which the vector processing rate exceeds that of scalar processing and calculate Hockney's $n_{\frac{1}{2}}$. For 1 and 8 CEs, we locate the points where each kernel achieves its maximum computational rate. We also present the degradation factor when the referenced / vectors do not fit in the cache. We give upper/lower bounds on the achieved speedups and on the multiprocessing overhead. We show the speedup variation as the number of CEs is increased from 2 to 8. Our results reveal some interesting phenomena. The shared cache, the bandwidths of the computational processors and the shared memory, and the handling of kernels with indirect addressing are major factors affecting the vector processing performance of the FX/8.

17. Key Words and Document Analysis. 17a. Descriptors

Performance evaluation
Vector processors
Multiprocessing
The Alliant FX/8
Shared caches

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
Release Unlimited	UNCLASSIFIED	127
	20. Security Class (This Page) UNCLASSIFIED	22. Price