NASA-CR-172,205

NASA Contractor Report 172205

A Performance Analysis of the PASLIB Version 2.1X SEND and RECV Routines on the Finite Element Machine

J. D. Knott

Kentron International, Inc.
Kentron Technical Center
Hampton, Virginia 23666

Contract NAS1-16000
August 1983

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

1. SUMMARY

The Finite Element Machine is an experimental array processor designed to support research in parallel algorithms and architectures. This report presents the results of a case study of communications using the SEND and RECV system software routines on the Finite Element Machine, followed by a discussion of the impact of communications overhead on the efficiency of parallel algorithms.

2. INTRODUCTION

Computer performance has traditionally been determined by the architecture of the computer and the level of technology used to implement that architecture. Computer performance can be enhanced by using a faster technology for a given architecture, or through the use of innovative architectures which allow the various components of computation to proceed in parallel. The technological approach has brought about significant increases in computational power, but technology is thought by many to be approaching its physical limits. In contrast, the exploration of parallel architectures is in its infancy. The use of highly parallel architectures to provide supercomputer performance has become a major area of interest in computer science research. An example of such research is the Finite Element Machine (FEM) currently under construction at the NASA Langley Research Center.

The Finite Element Machine is an array of 36 asynchronous microcomputers (the Array), each capable of functioning as a stand alone computer, coupled to a minicomputer front end (the Controller). Current topics of interest in the FEM project include computer architecture, data management, Array control, and parallel algorithms.

A critical factor affecting algorithm structure and efficiency on array computers is the interconnection of processors and the efficiency of inter-processor I/O. This paper presents an overview of the architecture of the Finite Element Machine and the systems level software implemented to support applications, and investigates the performance of I/O on the Array using the SEND and RECV system library routines. Several examples are then given to illustrate the impact of I/O performance on the parallel decomposition of algorithms.

3. THE FINITE ELEMENT MACHINE ARCHITECTURE

The Finite Element Machine is an experimental parallel array processor designed to support research in architectures and algorithms for parallel computation [1]. FEM consists of a minicomputer front-end and an array of 36 asynchronous microcomputers. A block diagram of the FEM architecture is shown in figure 1. An overview of the FEM architecture is given below. For a detailed explanation of the architecture of the Array, see [2].

3.1 The Controller

The Controller is a conventional sequential minicomputer which is used to provide program development tools and mass storage for the Array. Programs for the Finite Element Machine are compiled and link edited on the Controller and then downloaded to the Array. The

N83-34597#

Controller also hosts the user interface to the Array which supports problem definition, task initiation and monitoring, interactive debugging of user tasks on the Array, and the uploading and analysis of results.

3.2 The Array

Each microcomputer in the Array contains 3 circuit boards known as the CPU, I/O-1, and I/O-2 boards.

The CPU board contains a 16 bit microprocessor, 16 Kbytes of Eraseable Programmable Read Only Memory (EPROM), 32 Kbytes of dynamic Random Access Memory (RAM), a floating point arithmetic unit, two timers, and serial and parallel I/O interfaces. There is no shared memory in the Finite Element Machine. However, processors can share information over any of the four communication paths provided by the architecture. These communication paths consist of a network of nearest neighbor connections (local links), a time multiplexed global bus, a cooperative binary flag network, and a cooperative sum/maximum computation network. The SEND and RECV routines only utilize the local links and the global bus.

The I/O-1 board contains circuitry for twelve reconfigurable serial communication links and the cooperative sum/maximum network. Each of the local communication links is a 1.5 Mhz bit serial interface with an associated hardware FIFO buffer capable of storing up to 16 words (16 bits per word) of input data. The local links are normally connected in a toroidal eight nearest neighbor scheme (see figure 2.). The link configuration is determined by the physical connections made on the front edge of the I/O-1 board and is in no way constrained by software. This allows the connectivity of processors in the Array to be modified (prior to execution) to support a wide range of interconnection schemes. The perfect shuffle, perfect shuffle nearest neighbor, and the cube connected cycle are but a few examples of the interconnection topologies possible on FEM [3,4].

The I/O-2 board contains the circuitry for the global bus and the cooperative flag network. The global bus is a 1.25 Mhz time multiplexed 16-bit parallel bus which can transmit to individual processors in the Array, or to all cooperating processors in the Array. All processors on the global bus have equal priority and the bus awards priority on first come, first served basis [5]. The global bus has hardware FIFO buffers on the input and output lines, each capable of storing up to 64 words of data.

4. SYSTEM SOFTWARE

The system software packages support applications on FEM. FEM Array Control Software (FACS) provides the user interface to the Array, the Nodal Executive (Nodal Exec) is the microcomputer operating system, and the PASCAL Library (PASLIB) supports PASCAL access to the unique architectural features of the Finite Element Machine.

4.1 FEM Array Control Software (FACS)

FACS is a collection of menu driven, user friendly commands used to control the Finite Element Machine. FACS resides on the Controller and communicates with the Array via the global bus. The Controller appears to the Array as just another processor on the global bus, and interfaces directly with the Nodal Executive operating system on a request/acknowledge basis.

4.2 Nodal Executive

Nodal Exec is the microcomputer operating system embedded in EPROM on each processor in the Array. Nodal Exec contains a kernel and a set of command routines. The kernel provides standard operating system functions such as monitoring, I/O primitives, interrupt handling, and memory management. The command routines support the Controller/Array interface and are accessed via the operating system monitor. Command routine services are requested by the Controller for such functions as downloading object code, starting and stopping tasks, and uploading results from the Array.

4.3 PASCAL Library (PASLIB)

PASLIB is a PASCAL callable subroutine library which allows programmers to access the nonstandard architectural features of FEM. Application programs for the Finite Element Machine are written in PASCAL with PASLIB procedures linked in as external procedures. PASLIB routines are analagous to the supervisor call on conventional systems in that they provide an interface to the Nodal Exec operating system on the microcomputer, but they additionally provide run-time support for mathematical functions utilizing the floating point unit, and for the special I/O capabilities of the Array.

5. I/O PERFORMANCE OF SEND AND RECV

SEND and RECV are PASLIB routines used to transmit and receive data items over the global bus or local communication links. Data items range in size from 1 to 255 sixteen bit words. This section describes the operation of the SEND and RECV PASLIB routines and their associated interrupt handlers.

5.1 SEND

The SEND routine is used to transfer data to a neighboring processor over the global bus or a local link. Data can be transferred in a synchronous or asynchronous I/O mode although the mode is transparent to the SEND call itself.

When the SEND call is executed the processor branches to the SEND subroutine and copies the data to be transferred into an output buffer. SEND then calculates and appends a checksum, places the buffer on the appropriate output queue (either local or global), enables send interrupts, and returns to the calling program.

When the send interrupt occurs, control is transferred to the interrupt handling routine and data is transferred from the output queue to the receiving processor. If the receiver can handle the incoming data as fast as or faster than the transmitting processor, one interrupt entry is all that is required. However, if the receiver removes the incoming data at a slower rate, the hardware FIFO buffer fills and the transmitting processor will return control to the interrupted process until the FIFO's can again accept data.

5.2 RECV

The RECV routine is used to receive data transmitted over the global bus or a local link. The operation of the RECV call is determined by the I/O mode, either synchronous or asynchronous.

When data is detected on either the global bus or local links, a receive interrupt is generated and the receiving processor branches to the receive interrupt handling routine.

In the synchronous I/O mode, the interrupt routine buffers incoming data in a first-in first-out software queue and a call to the RECV routine will return the first buffer on the queue. If the queue is empty, a RECV call in synchronous mode will wait for data to arrive.

In contrast, the asynchronous mode keeps a copy of the last complete data buffer received and uses a temporary buffer to assemble incoming data. Whenever the buffer being assembled is complete, the temporary buffer is written over the permanent copy. The RECV routine will not wait for data to be received in the asynchronous mode and will return only the most recently received buffer; any given buffer may be read more than once, or may be overwritten by a more recent buffer before it can be read.

5.3 METHODOLOGY

The performance of I/O on the Array is determined by the overhead of the initial PASLIB routine call and the subsequent interrupt processing activity. The time for the initial PASLIB calls for the SEND and RECV routines is governed by two factors, a fixed overhead for context switching, buffer allocation, and table lookups, and an incremental cost for data manipulation based on the buffer size. Similarly, the interrupt processing routines associated with the SEND and RECV calls consist of a fixed and incremental cost.

Although it is possible to determine the asymptotic rates of performance for the communications hardware and the instruction execution times for the associated PASLIB and interrupt handling routines, these figures do not accurately predict the performance of I/O under actual operating conditions. fixed and incremental cost. The interrupt processing routines may be entered one or more times for any given SEND call and it is precisely this factor which makes it difficult to predict the performance of I/O on the Array without measuring it in operation.

In order to measure the performance of the SEND and RECV PASLIB routines in actual use, a series of timing studies were run utilizing both synchronous and asynchronous communication modes on the local links and global bus. Since the SEND and RECV routines are entered only once per call, these times were measured first and then used as a basis for determining the interrupt times. Interrupt times were then determined by measuring the time for a series of I/O transmissions and subtracting the known cost of program control statements and the PASLIB routine calls. The results of the timing runs for the SEND and RECV routines are given below.

5.4 SEND PERFORMANCE

The SEND call itself consists of a fixed amount of code for context swapping and mapping, and a variable amount of code which is dependent on the size of the buffer being transmitted. This time will vary for different buffer sizes but will remain constant for each call with the same buffer size. Likewise, the send interrupt routine consists of a fixed overhead for entry and exit and a fixed time for each data item transferred, but the number of entries into the interrupt handler depends on the availability of the transmitting medium and can vary from call to call. The first task to accomplish in measuring the SEND time was to determine the fixed time for the PASLIB routine call.

The fixed overhead and cost per word for the SEND procedure call was measured and found to be the same for both communications paths in either I/O mode. This was expected since the I/O mode is transparent to the SEND routine. Table 1 gives the fixed overhead and the cost per word for all calls to the SEND routine.

	Fixed Overhead	Cost Per Word
All SEND Calls	0.7123	0.0227

Table 1. SEND Procedure Call - Time in Milliseconds

Once the cost of the SEND call was known, the time spent in processing send interrupts was determined. The results for both I/O modes are given in figures 3.a through 3.d. Here it can be seen that although the SEND call rate is constant, the interrupt rate changes at a fixed point for both the local links and the global bus. This is directly attributable to the receive interrupt processing rate and the depth of the hardware FIFOs. The receive interrupt rate, as we shall see in the next section, is substantially slower than the send interrupt rate. Therefore, the send interrupt is loading the hardware FIFOs faster than the receive interrupt can read them, and as soon as the FIFOs fill up the send interrupt routine exits and waits for room. With the additional overhead for repeated send interrupt calls, the send interrupt cost increases dramatically. This occurs on about the 23rd transmission on the local links (16 word FIFOs) and after approximately 180 transmissions on the global bus (combined FIFO depth of 128 words). If we graph this data as time per word (figures 4.a. through 4.d.), it is obvious that this transition point yields the lowest cost per word when sending data.

5.5 RECV PERFORMANCE

The RECV PASLIB procedure and the receive interrupt processing routine both contain a fixed overhead and a cost per word. The total time for a RECV call is constant for any given buffer size because there is exactly one procedure call per buffer transfer. The receive interrupt time can vary depending on the availability of data and the number of entries into the interrupt processing routine, although it was discovered that the receive interrupt routine will be entered no more than once per SEND in the present version of the Nodal Exec operating system.

The fixed overhead and cost per word for the RECV procedure call was measured in the synchronous and asynchronous I/O modes on both the local links and global bus. The fixed overhead and a cost per word for each of the RECV I/O modes is given in Table 2.

	Fixed Overhead	Cost Per Word
Local Sync	0.3492	0.0548
Global Sync	0.3492	0.0548
Local Async	0.3817	0.0163
Global Async	0.3817	0.0163

Table 2. RECV Procedure Call - Time in Milliseconds

In contrast to the SEND routine, the fixed overhead and cost per word in the RECV is different for synchronous and asynchronous transmissions. This was expected since the I/O mode is embedded in the RECV and receive interrupt routines. The asynchronous receive must lock out receive interrupts while reading the last complete copy of data to prevent the interrupt routine from overwriting the data being read. As a result, the fixed overhead for the asynchronous mode is greater than the fixed overhead for the synchronous mode. However, the synchronous cost per word is much greater than the asynchronous because the synchronous mode must handle queue pointers in a circular buffer whereas the asynchronous simply reads from a fixed buffer.

Having found the cost of the RECV call, a series of timing tests was run to determine the interrupt processing time. The results are given in figures 5.a through 5.d. Here it can be seen that both the RECV call rate and the interrupt processing rate are linear. This is expected since the receive interrupt processing rate is slower than the send interrupt rate, and indicates that all data is received in one interrupt. The representation of the interrupt time on a per word basis is asymptotic (figures 6.a through 6.d) and therefore the larger the buffer the less receiving a buffer costs.

6. IMPACT ON ALGORITHMS

The cost of I/O on parallel processing arrays is a fundamental factor in the performance of most algorithms. In general, efficient decomposition of algorithms into parallel components requires that the time saved for arithmetic operations exceed the time for the I/O required to support the distributed computations. This can be demonstrated by doing an analysis of a simple summation of 32 single precision values. Given that a single precision add takes 475 us, and using the total I/O time of 3400 us for two word (single precision) transfers, let us look at several approaches to the partial summation problem as outlined by Hockney and Jesshope [6]. The first approach is to simply compute the sum sequentially on one processor. This requires n-1 additions and no transfers. Given n = 32, the sequential approach takes (n-1) * 475 us, or a total of 14,725 us. A second approach is to distribute pairs of values to sixteen processors, sum the pairs, pair results, sum the pairs, pair results, etc. This method requires $\log_2 n$ parallel additions and $(\log_2 n)-1$ parallel transfers and would seem to promise an improvement over the sequential method. However, the total time for this method is (5 * 475) + (4 * 3400) = 15,975 which is more than the sequential time. Here we have eliminated 26 adds at a savings of 12,350 us, but we have introduced an additional 13,600 us for transferring data. A third method for summation might be to divide the work between only two processors, requiring (n/2) additions and only 1 transfer. The total time for this third approach is only 11,000 us, a 25% reduction of the sequential time. This savings results from eliminating 15 adds while introducing only one transfer. Clearly, given the ratio of I/O to arithmetic cost the solution time is more dependent on the I/O introduced by the transmission of data than it is on the degree of parallelism achieved in the algorithm. This might require an approach other than the original FEM concept [7] of assigning one finite element node to each processor. While assigning one finite element node per processor minimizes the time for floating point arithmetic by distributing the work across the Array, it can increase the total solution time by introducing highly inefficient I/O. The Multi-Color SOR algorithm [8] demonstrates the effectiveness of balancing the I/O and arithmetic on the Array by assigning multiple nodes to a single processor.

Since the I/O SEND and RECV routines contain a fixed overhead, one way to decrease the overall cost of I/O is to transmit blocks of data. By transmitting data in blocks, the overall cost per word is decreased because the fixed overhead for I/O is distributed over the entire block of data. figures 7.a, 7.b, and 7.c. provide a summary of the performance of the SEND and RECV routines in version 2.1X of PASLIB. This total I/O time is given on a cost per word basis in figures 8.a, and 8.b. These graphs provide a basis for the optimization of algorithms by allowing FEM applications programmers to choose the optimal I/O mode and medium for any given buffer size. For example, the cost per word of transmitting 20-word buffers (10 single precision real numbers) on the global bus in synchronous mode is only a third of the cost per word for 2-word buffers. By selecting the appropriate communication link and mode, and by taking advantage of blocking, the cost of I/O can be substantially reduced.

Another factor directly impacting the cost of I/O is the connectivity of the Array. The ability to transmit data directly to distant processors significantly increases I/O performance on algorithms requiring such transfers. Let us consider the above summation problem on two possible switching networks. Assuming that data is initially distributed across thirty-two processors (one value per processor), the basic algorithm is to pair values and sum the pairs. This is repeated until one processor contains the sum of all values. Using a nearest neighbor (strictly left-right) connectivity the summation would require n-1 parallel shifts (transfers) and $\log_2 n$ parallel additions at a total cost of 107,775 us. This same algorithm executed on the Array using a nearest neighbor perfect shuffle connectivity would still require $\log_2 n$ additions, but the transfers are reduced to $\log_2 n$ for a total cost of only 19,375 us. Obviously, the ability to transfer to any processor in the Array offers an order of magnitude decrease in execution time for algorithms requiring long distance communication.

Many algorithms requiring long distance communication can be implemented by properly configuring the local links to provide the necessary communications paths. In cases where this is not possible, the global bus can provide the point to point communications required across the Array. The only restriction on using the global bus in this capacity is the potential for bus contention when many processors are attempting to transfer at once. The timing results of this study indicate that bus contention will never be an issue on the current implementation of the 36 processor FEM. In fact, the global bus bandwidth is sufficient to support all 36 processors simultaneously transmitting at their maximum rate on the global bus. The rate at which processors can place data on the global bus is governed by the send interrupt routine. The fastest interrupt time per word, 108.5 us, was for asynchronous global bus transmissions of 255 words. However, this time includes the overhead for repeated calls to the send interrupt routine. To determine the maximum rate at which any processor could load data onto the global bus, it was necessary to examine the code contained in the inner loop of the send interrupt routine. This code was found to take approximately 55 us per word which for 36 processors gives an maximum rate of one word every 1.53 us. Since the global bus hardware is capable of transferring one 16 bit data word and its associated identifiers every 800 ns, there is no possibility of a bus conflict in the current version of Nodal Exec.

7. CONCLUDING REMARKS

The Finite Element Machine is an excellent testbed for the exploration of parallel architectures and algorithms. The fact that point-to-point communications are available for all processors in the Array with no potential for hardware contention on the global bus opens the door to a broad range of research algorithms and readily facilitates architecture simulation and modeling.

The I/O performance data provided by this study identifies the sections of code in the Nodal Exec operating system and PASLIB which will benifit most from optimization in subsequent versions of system software. Any optimization of these routines in future revisions of the system software will invalidate these timings. However, several concepts highlighted by this study will remain valid regardless of future I/O performance on FEM, and also apply to MIMD architectures in general.

First, it is clear that whatever the ratio of I/O time to arithmetic on the Array, this ratio is critical to the efficiency of the algorithm. Decomposition of algorithms to maximize the parallelism in the problem is not a guarantee of efficiency and can actually cause a loss in performance over a sequential implementation of the same algorithm. Second, the I/O performance on the Array can be vastly improved by properly blocking the data to exploit the buffering provided by the hardware FIFOs and to distribute the fixed overhead over a number of words. In general, reducing the cost per word of I/O allows for greater distribution of arithmetic across the Array, which allows efficient exploitation of an algorithm's parallelism.

REFERENCES

1. Storaasli, O.O.; Peebles, S.W.; Crockett, T.W.; Knott, J.D.; and Adams, L.: The Finite Element Machine: An Experiment in Parallel Processing. NASA TM #84514, July 1982.

2. Jordan, Harry F., Ed.: The Finite Element Machine Programmer's Reference Manual. Computer Systems Design Group, University of Colorado, Boulder, 1979.

3. Stone, H. S.: Parallel Processing with the Perfect Shuffle. IEEE Transactions on Computers, Vol. C-20, No. 2, Feb. 1971, pp. 153-161.

4. Preparata, F. P.; and Vuillemin, J.: The Cube-Connected Cycles: A Versatile Network for Parallel Computation. Communications of the ACM, Vol. 4, No. 5, May 1981, pp. 300-309.

5. Knott, J.D.; and Crockett, T.W.: Fair Dynamic Arbitration for a Multiprocessor Communications Bus. Computer Architecture News, Vol. 10, No. 5, September 1982, pp. 4-9.

6. Hockney, R.W.; and Jesshope, C. R. Parallel Computers. Adam Hilger Ltd., Bristol, Great Britain, 1981.

7. Jordan, Harry F.; and Sawyer, Patricia L.: A Multi-Microprocessor System for Finite Element Structural Analysis. Trends in Computerized Structural Analysis and Synthesis. A. K. Noor and H. G. McComb, Jr., Editors, Pergamon Press, Oxford, 1978, pp. 21-29.

8. Adams, L.; and Ortega, J.: A Multi-Color SOR Method for Parallel Computation. Proceedings of the 1982 International Conference on Parallel Processing, August 1982, pp.53-57.

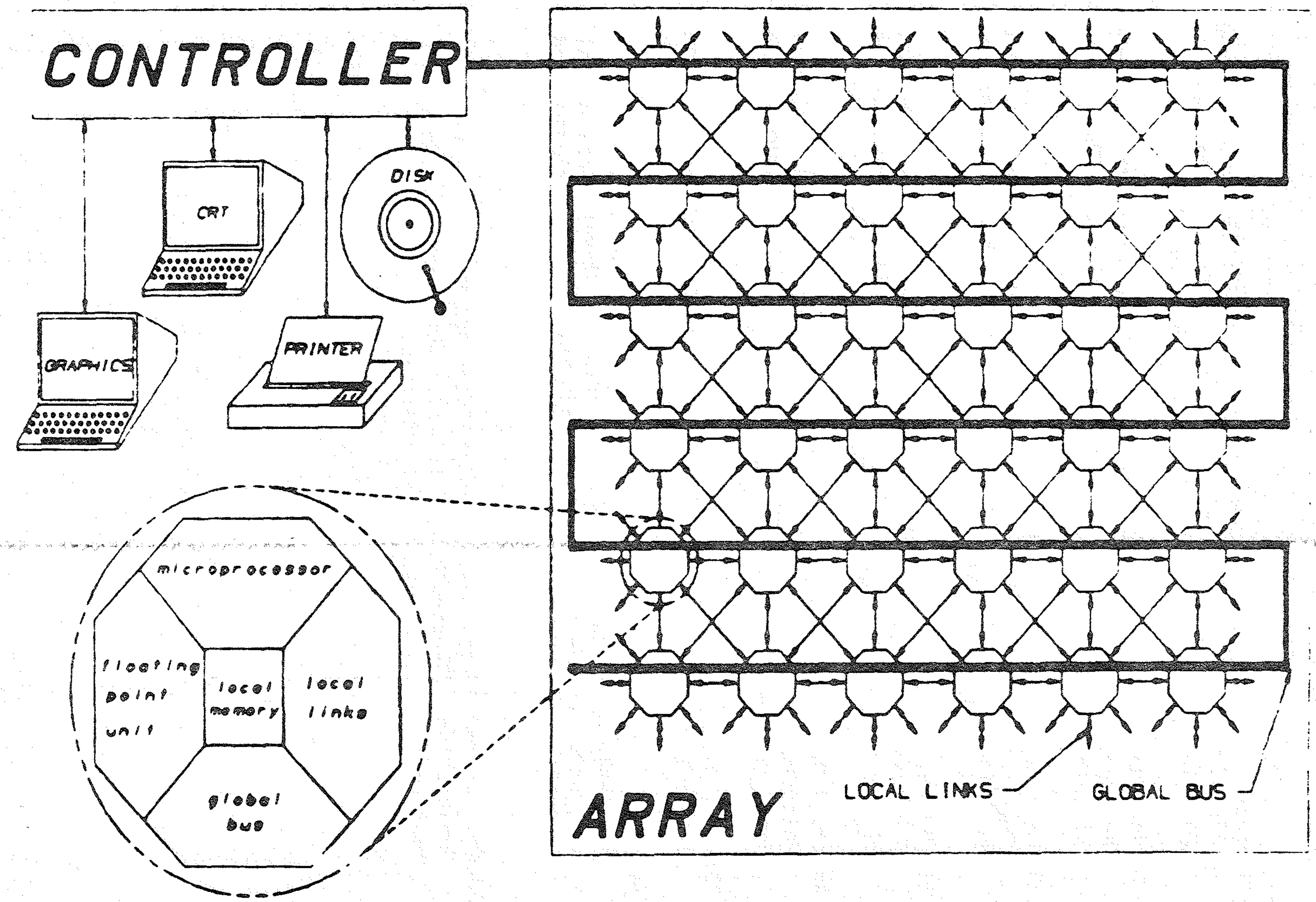

Figure 1. FINITE ELEMENT MACHINE BLOCK DIAGRAM

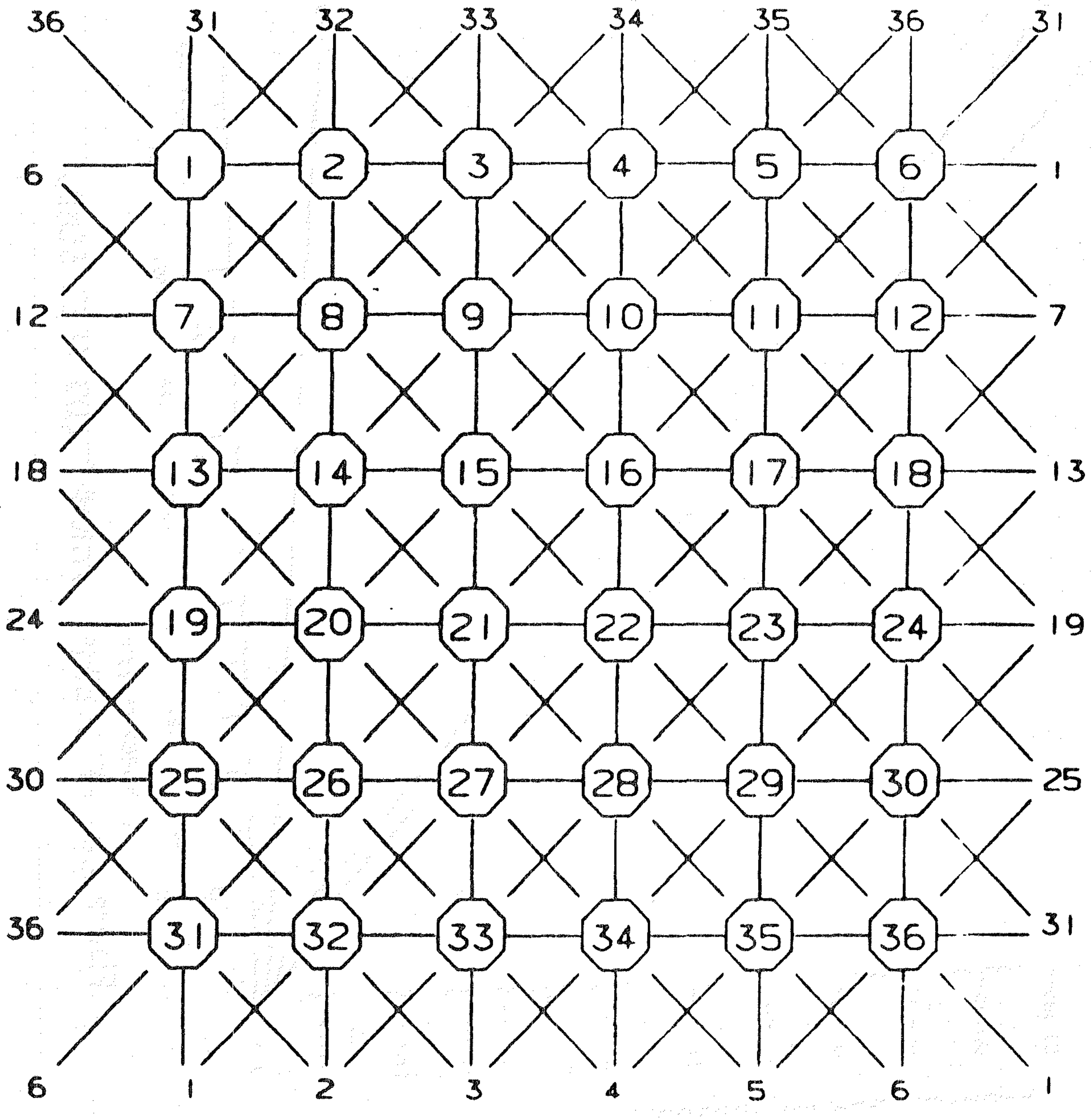

Figure 2. EIGHT NEAREST NEIGHBOR TOPOLOGY.

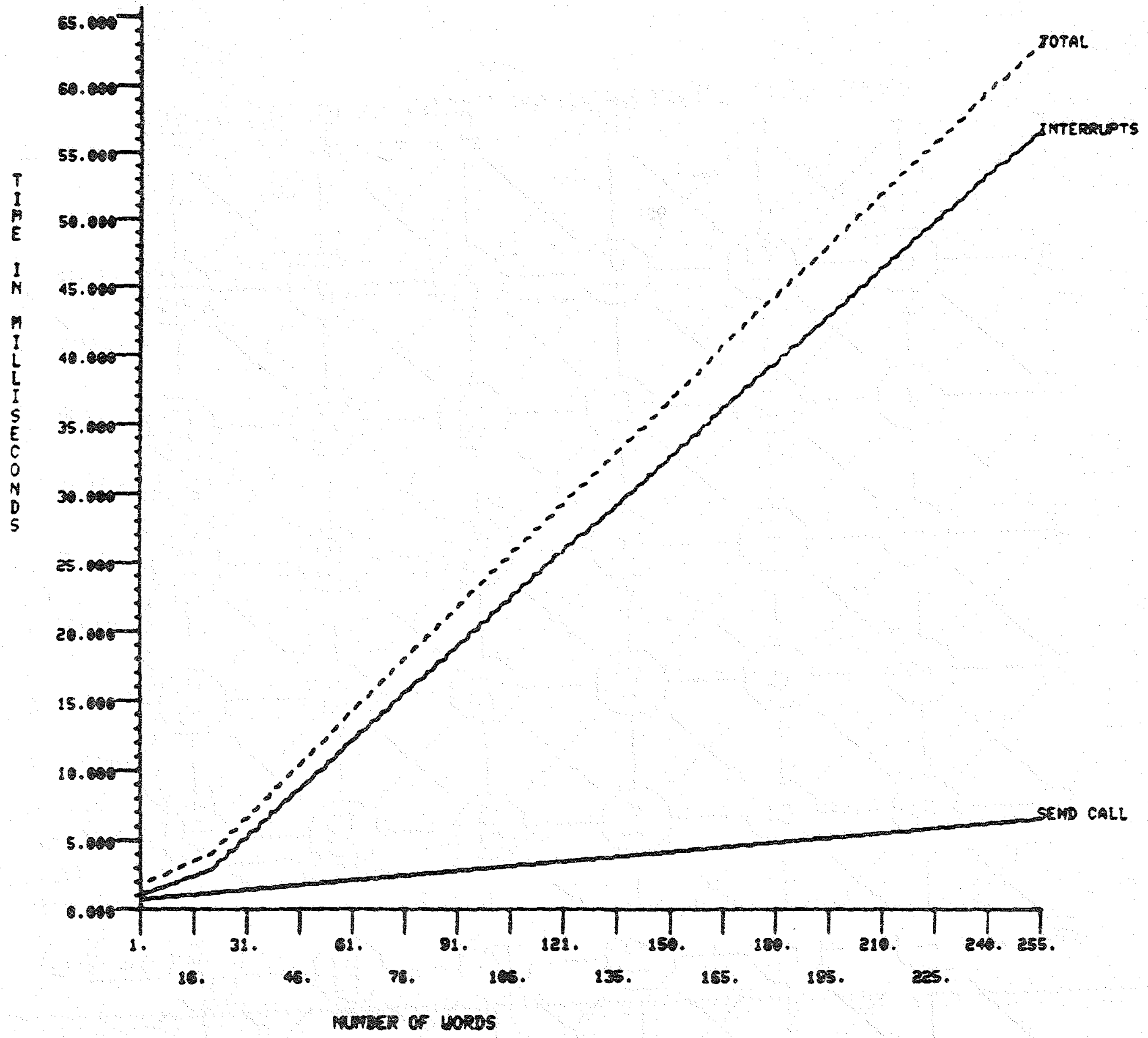

Figure 3.a. SEND LOCAL SYNCHRONOUS (PER SEND)

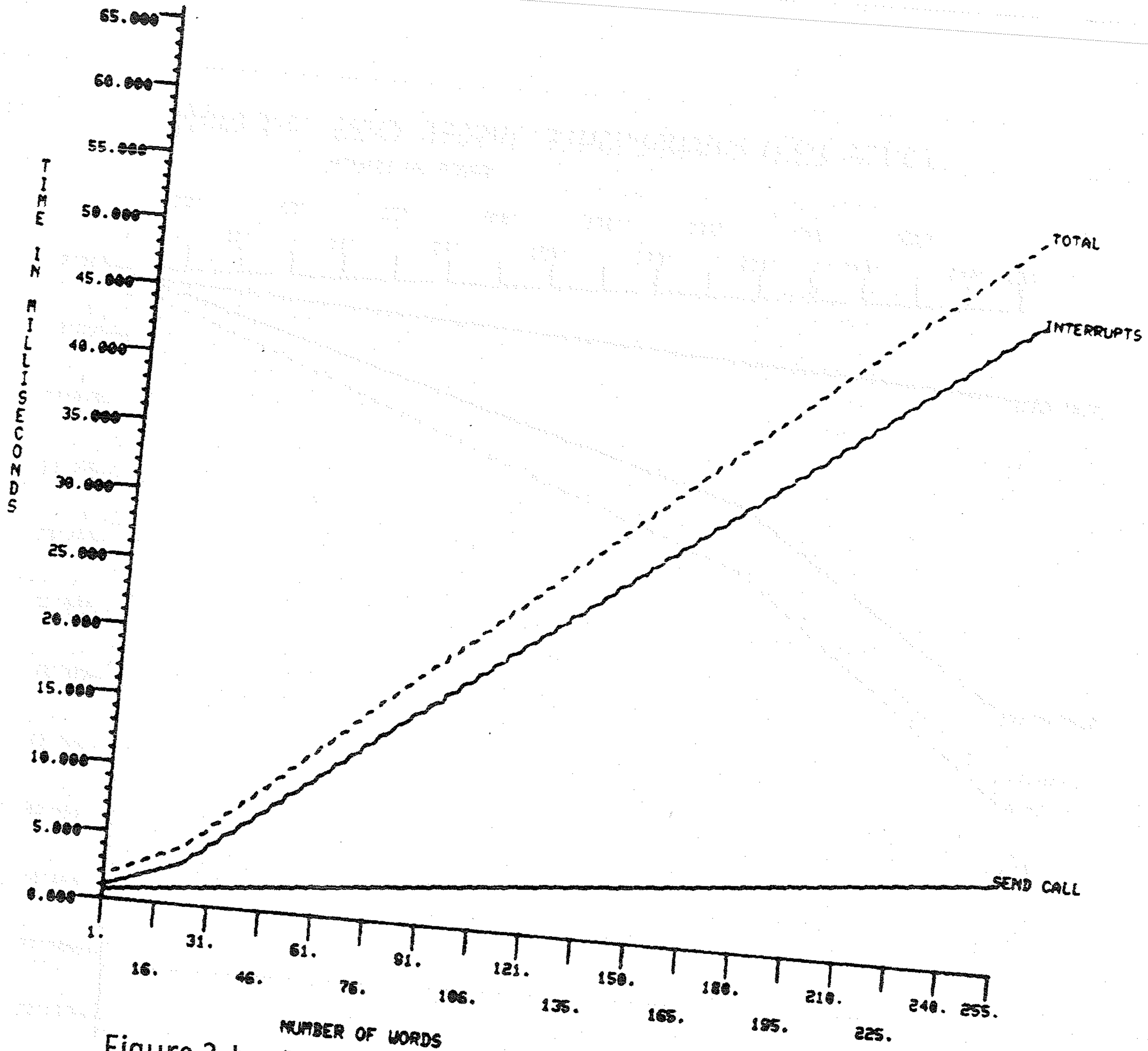

Figure 3.b. SEND LOCAL ASYNCHRONOUS (PER SEND)

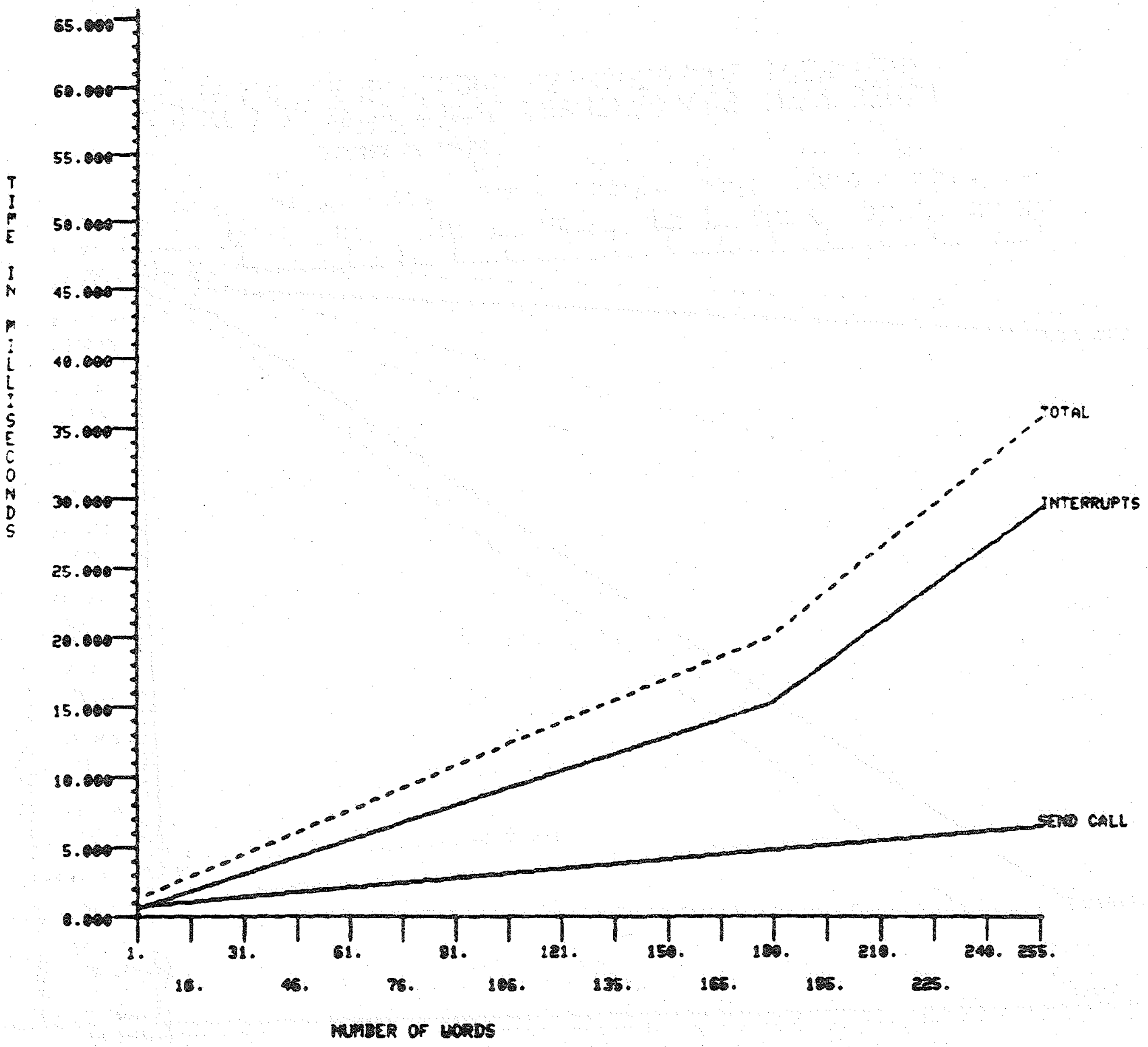

Figure 3.c. SEND GLOBAL SYNCHRONOUS (PER SEND)

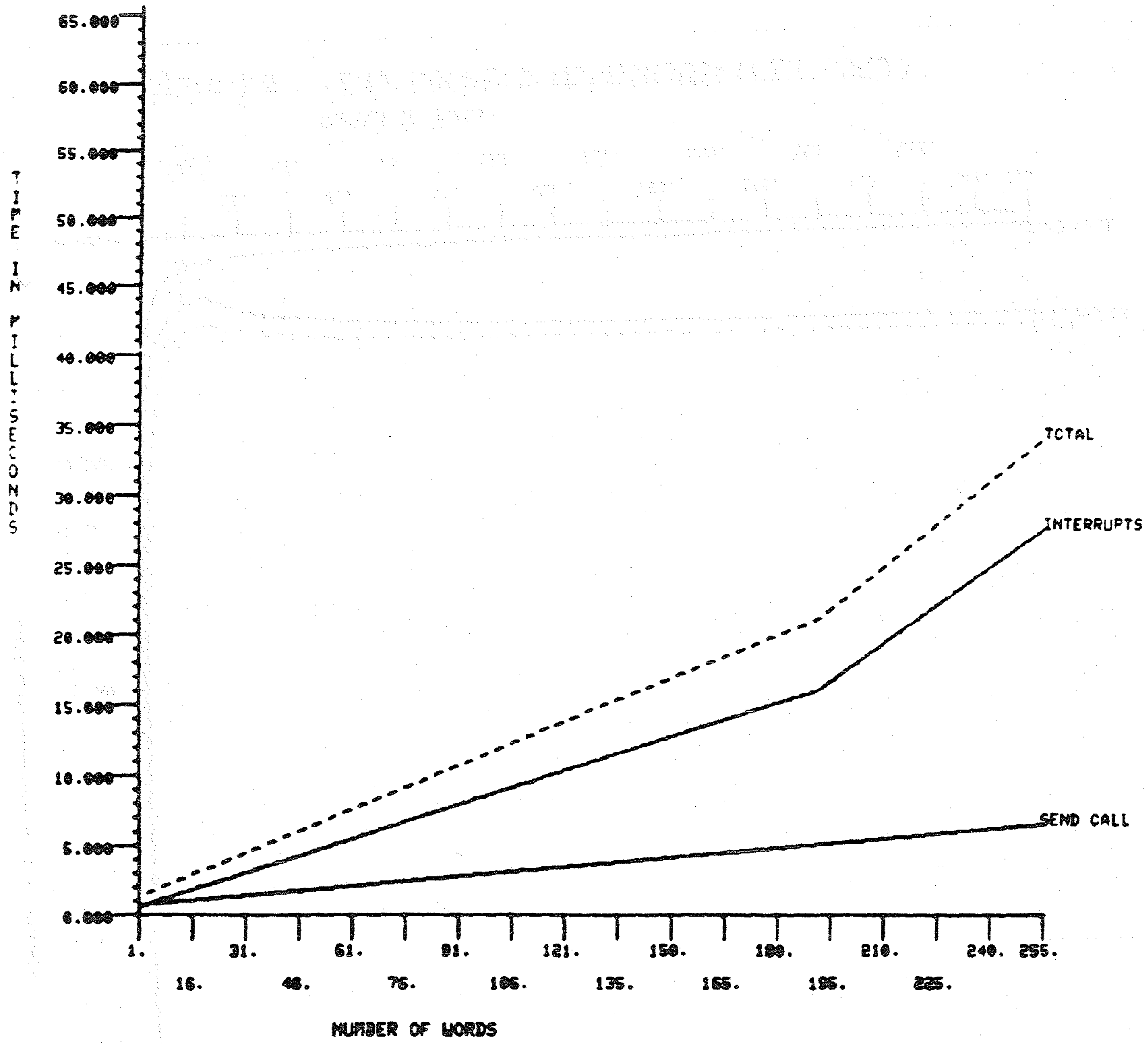

Figure 3.d. SEND GLOBAL ASYNCHRONOUS (PER SEND)

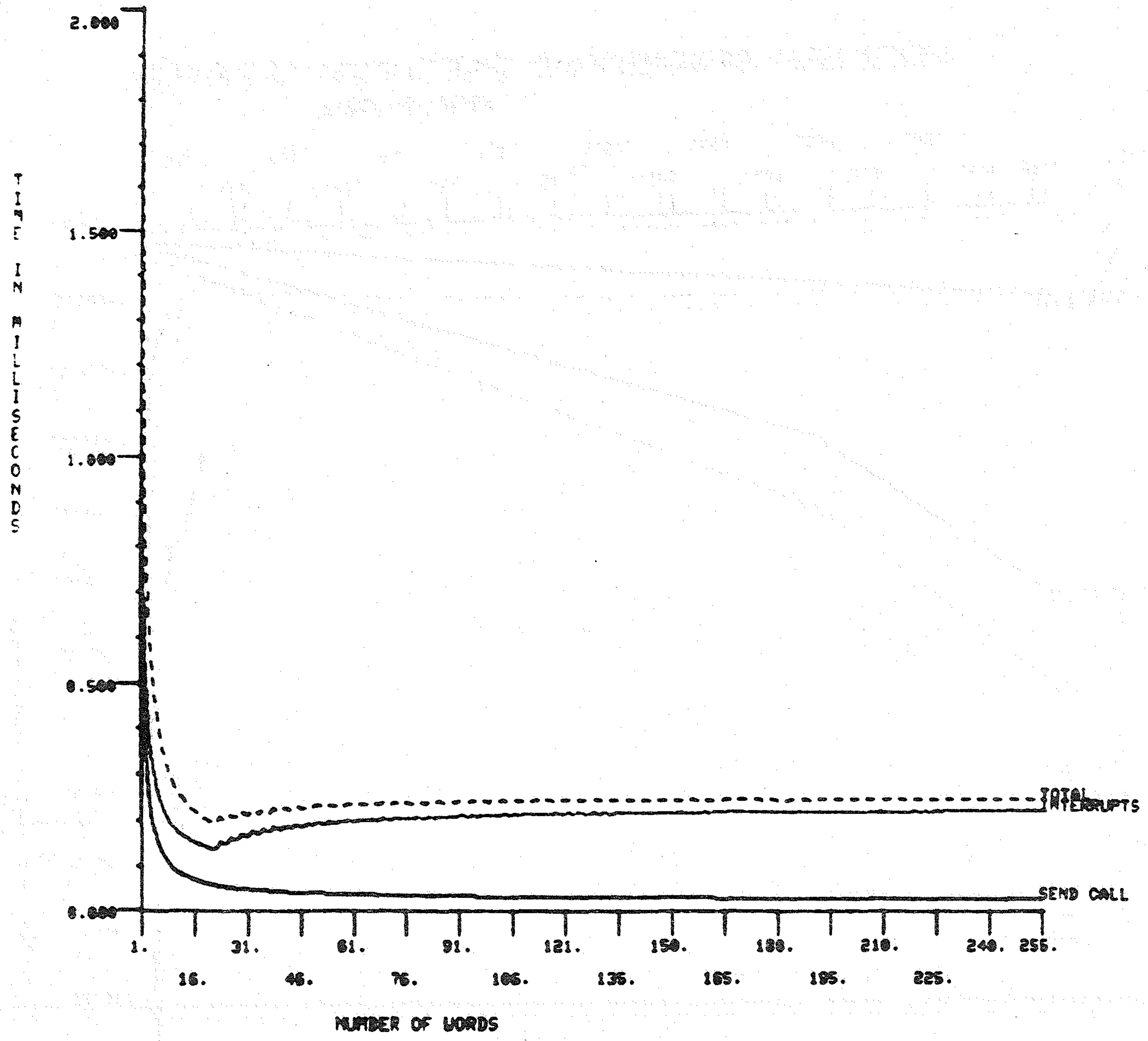

Figure 4.a SEND LOCAL SYNCHRONOUS (PER WORD)

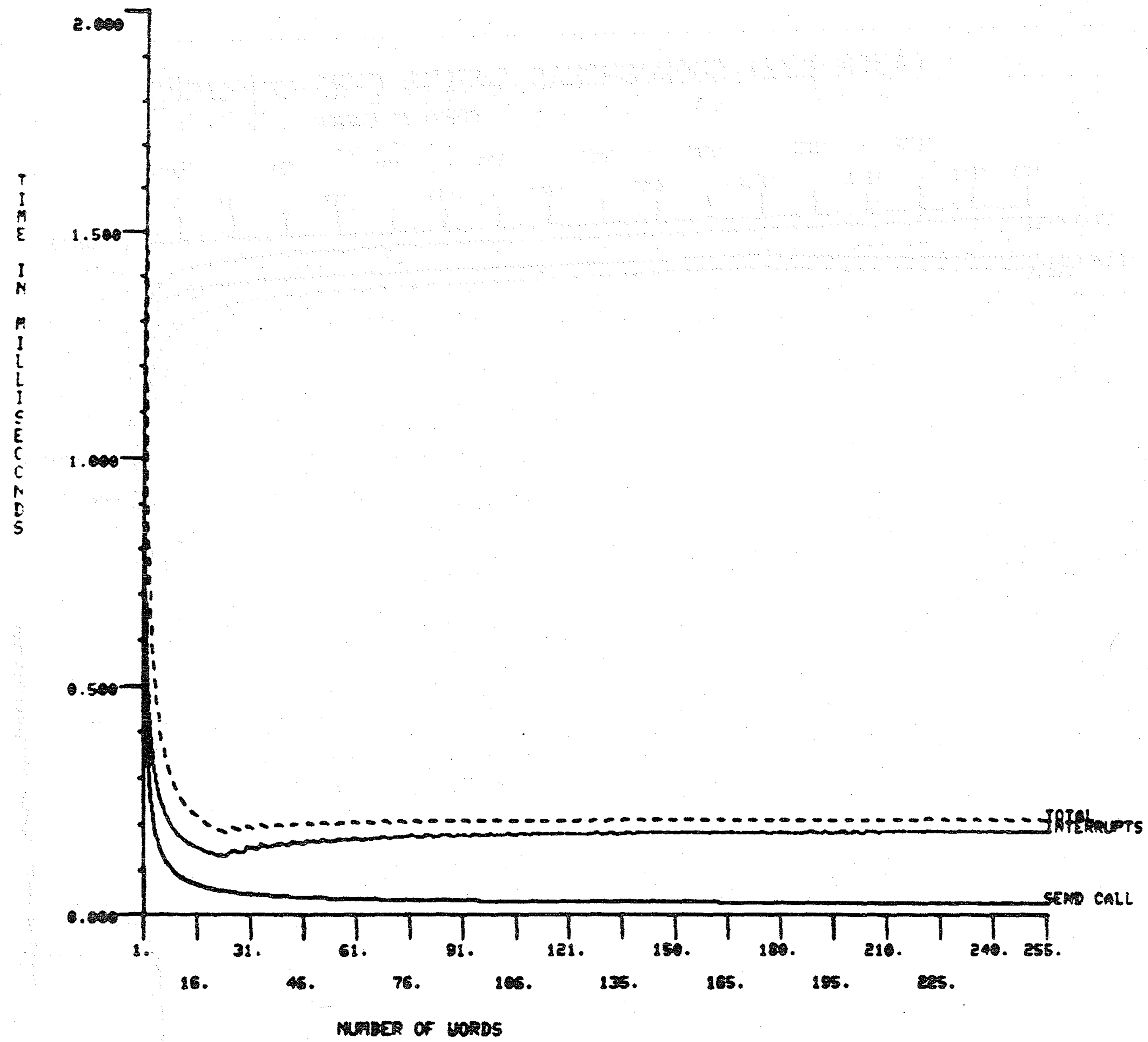

Figure 4.b. SEND LOCAL ASYNCHRONOUS (PER WORD)

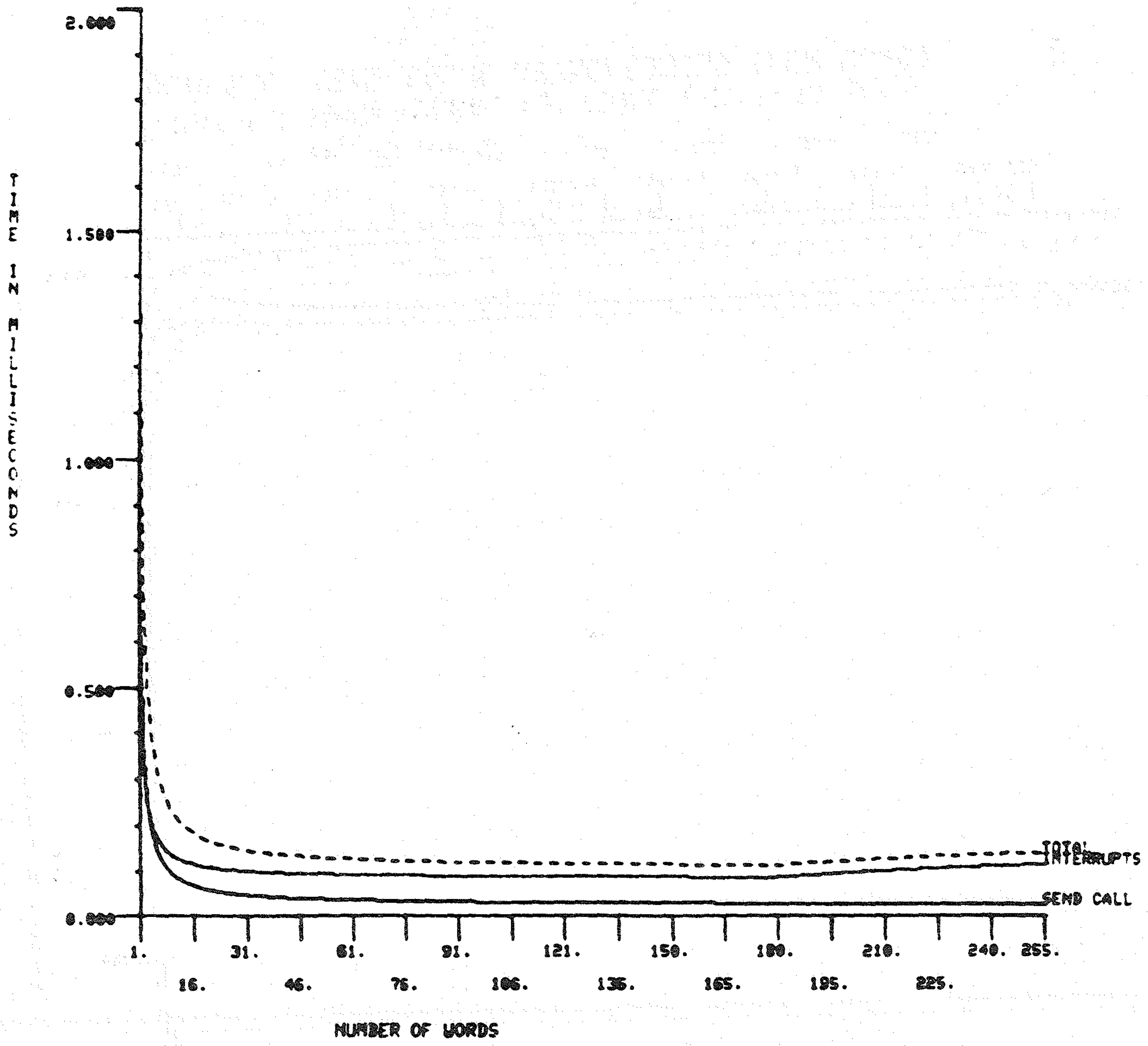

Figure 4.c. SEND GLOBAL SYNCHRONOUS (PER WORD)

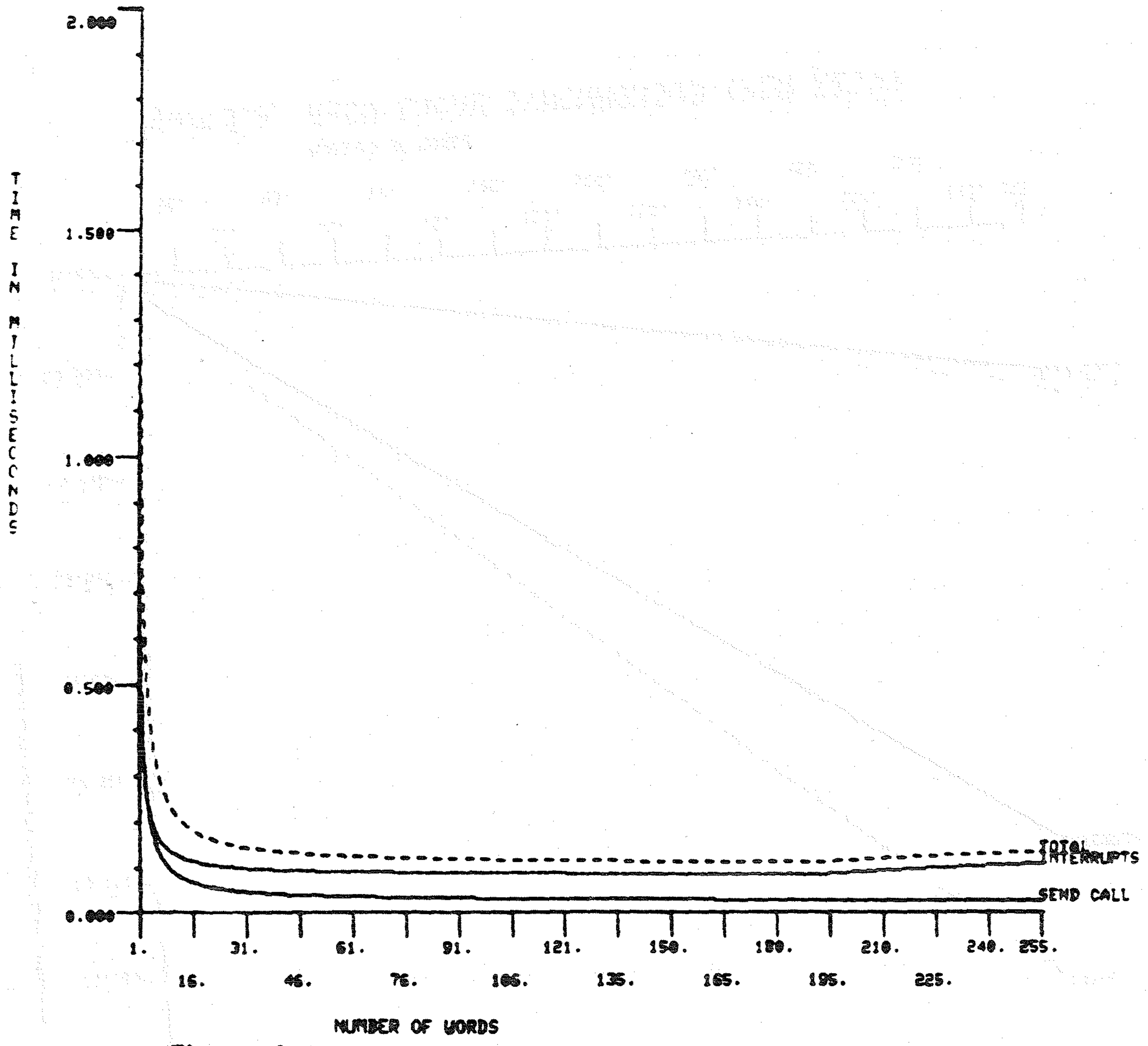

Figure 4.d. SEND GLOBAL ASYNCHRONOUS (PER WORD)

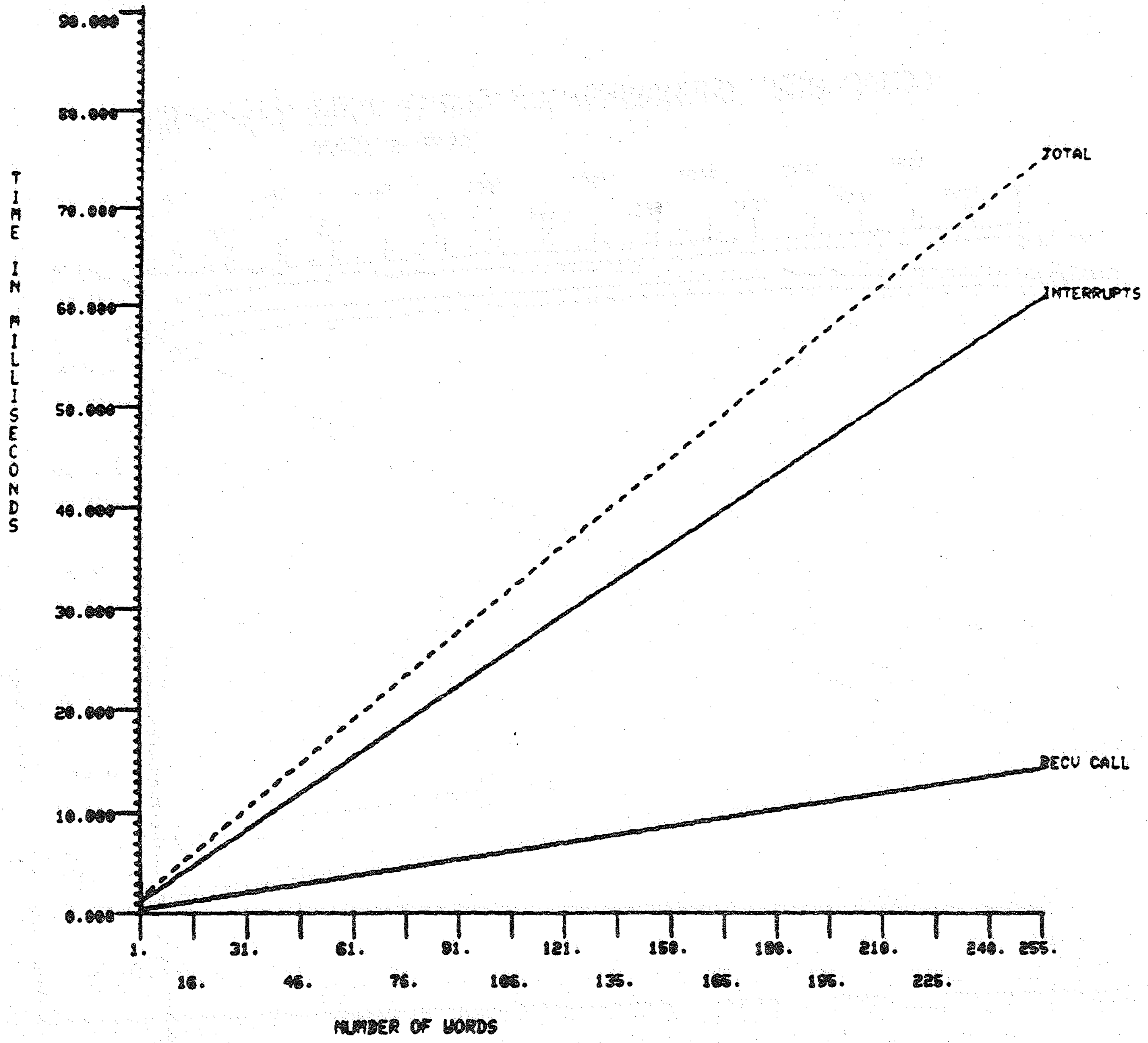

Figure 5.a. RECV LOCAL SYNCHRONOUS (PER RECV)

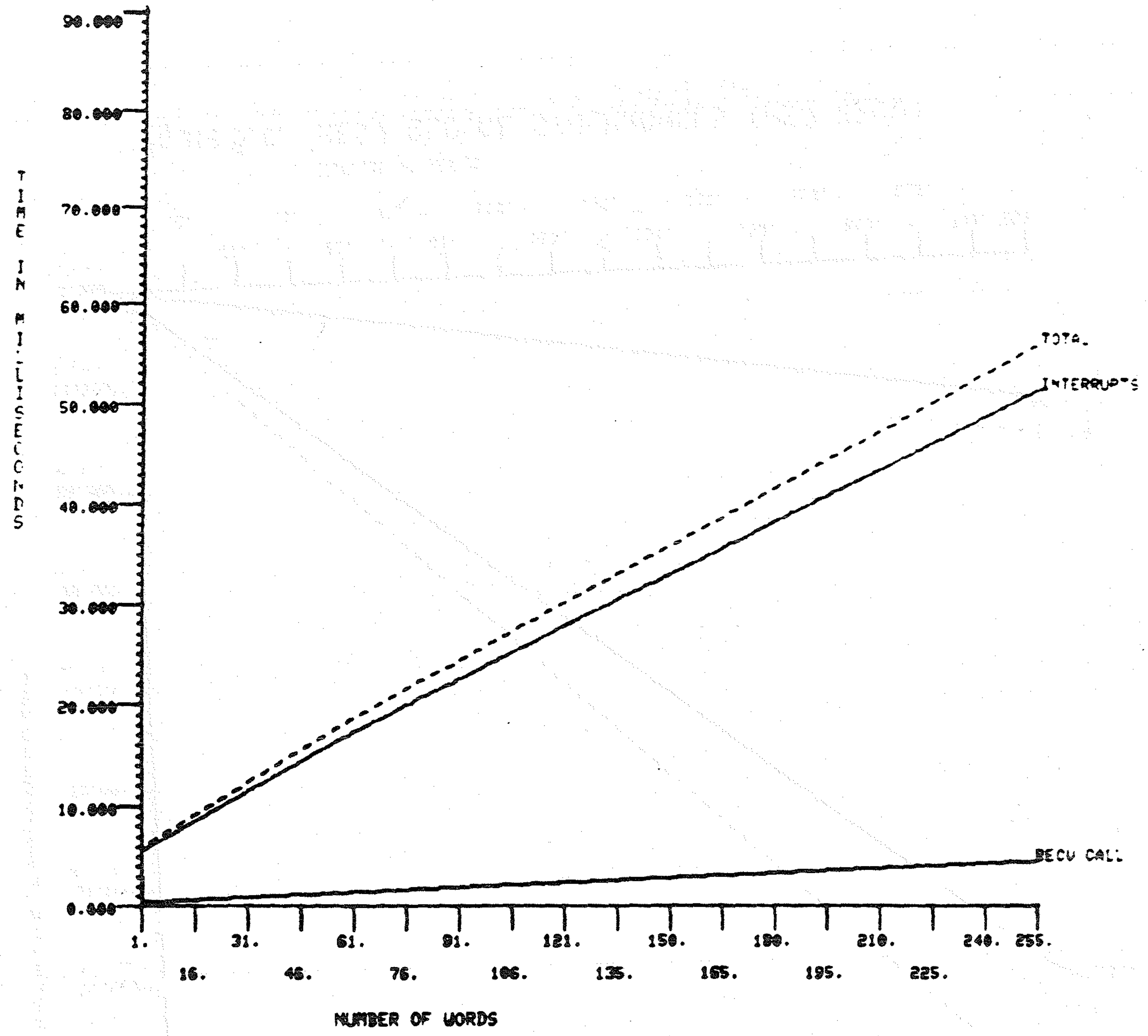

Figure 5.b. RECV LOCAL ASYNCHRONOUS (PER RECV)

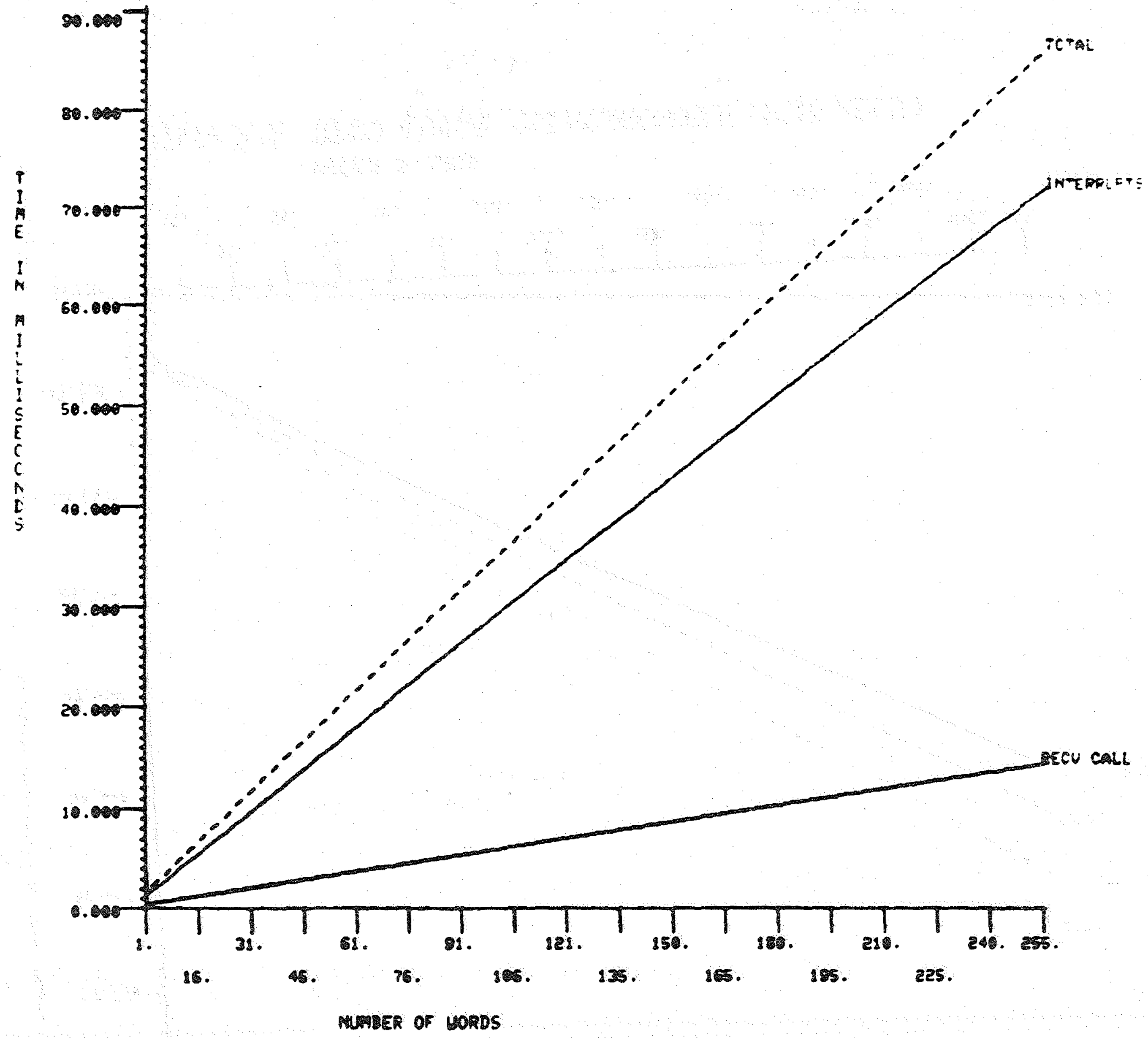

Figure 5.c. RECV GLOBAL SYNCHRONOUS (PER RECV)

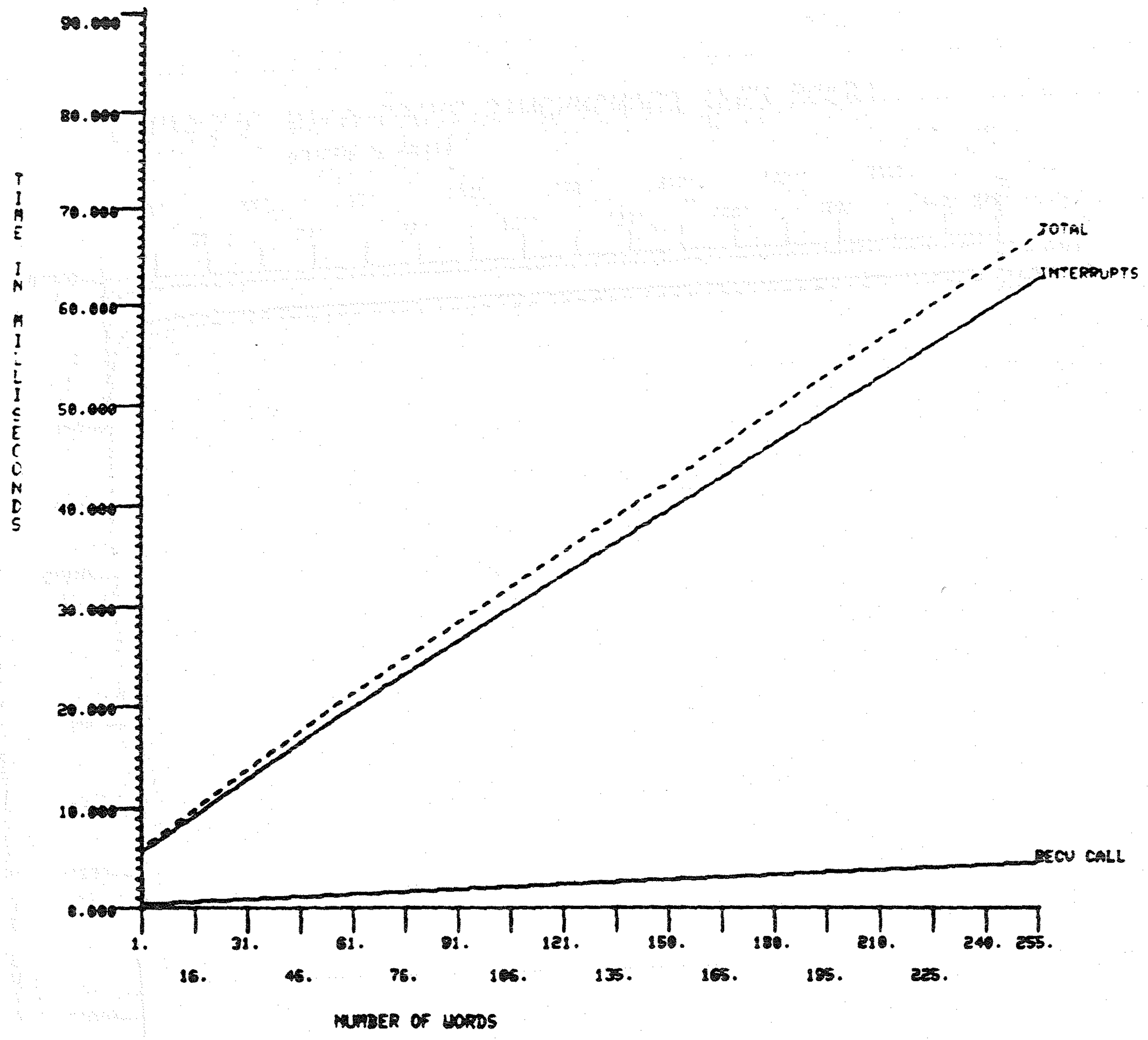

Figure 5.d. RECV GLOBAL ASYNCHRONOUS (PER RECV)

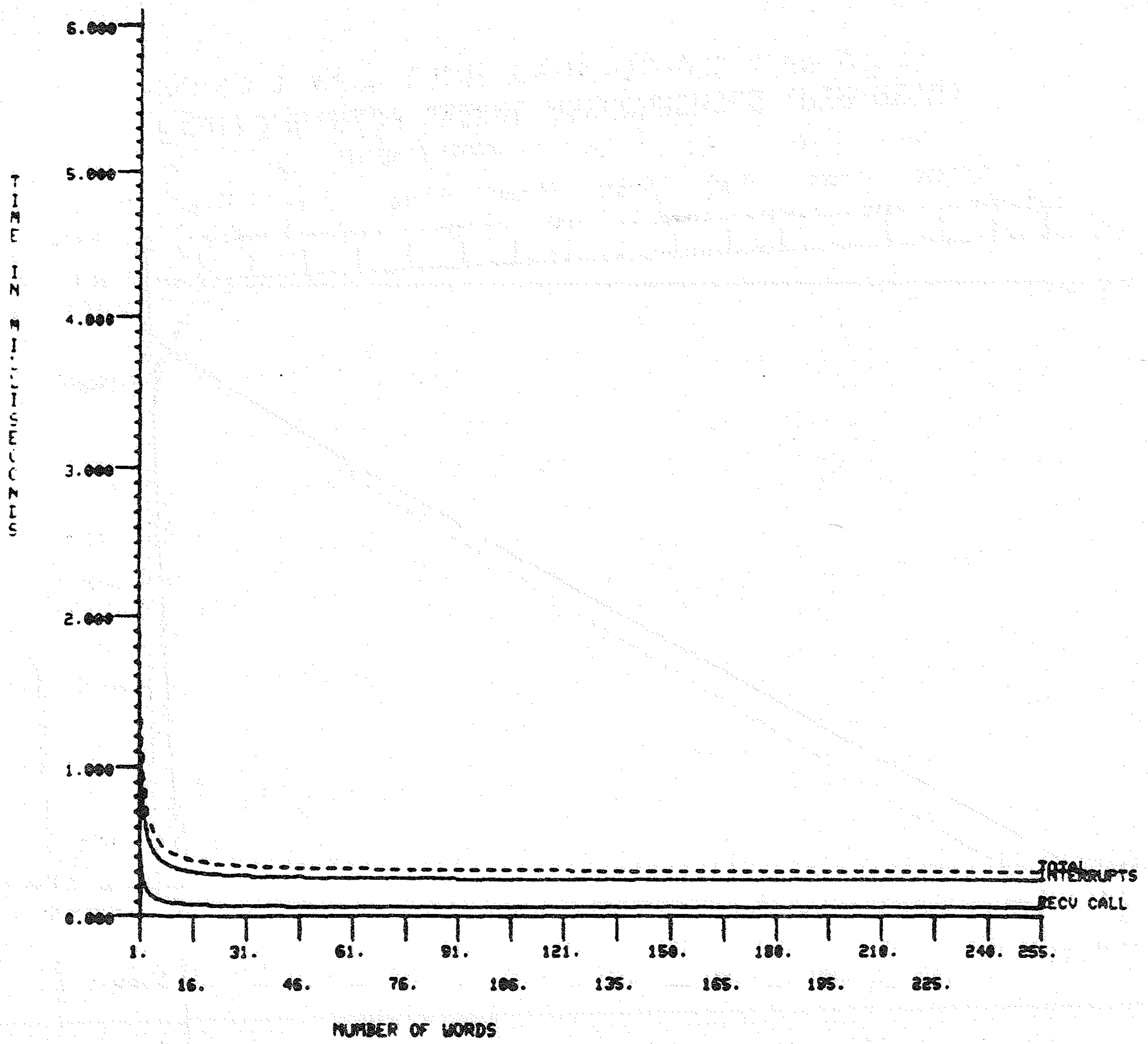

Figure 6.a. RECV LOCAL SYNCHRONOUS (PER WORD)

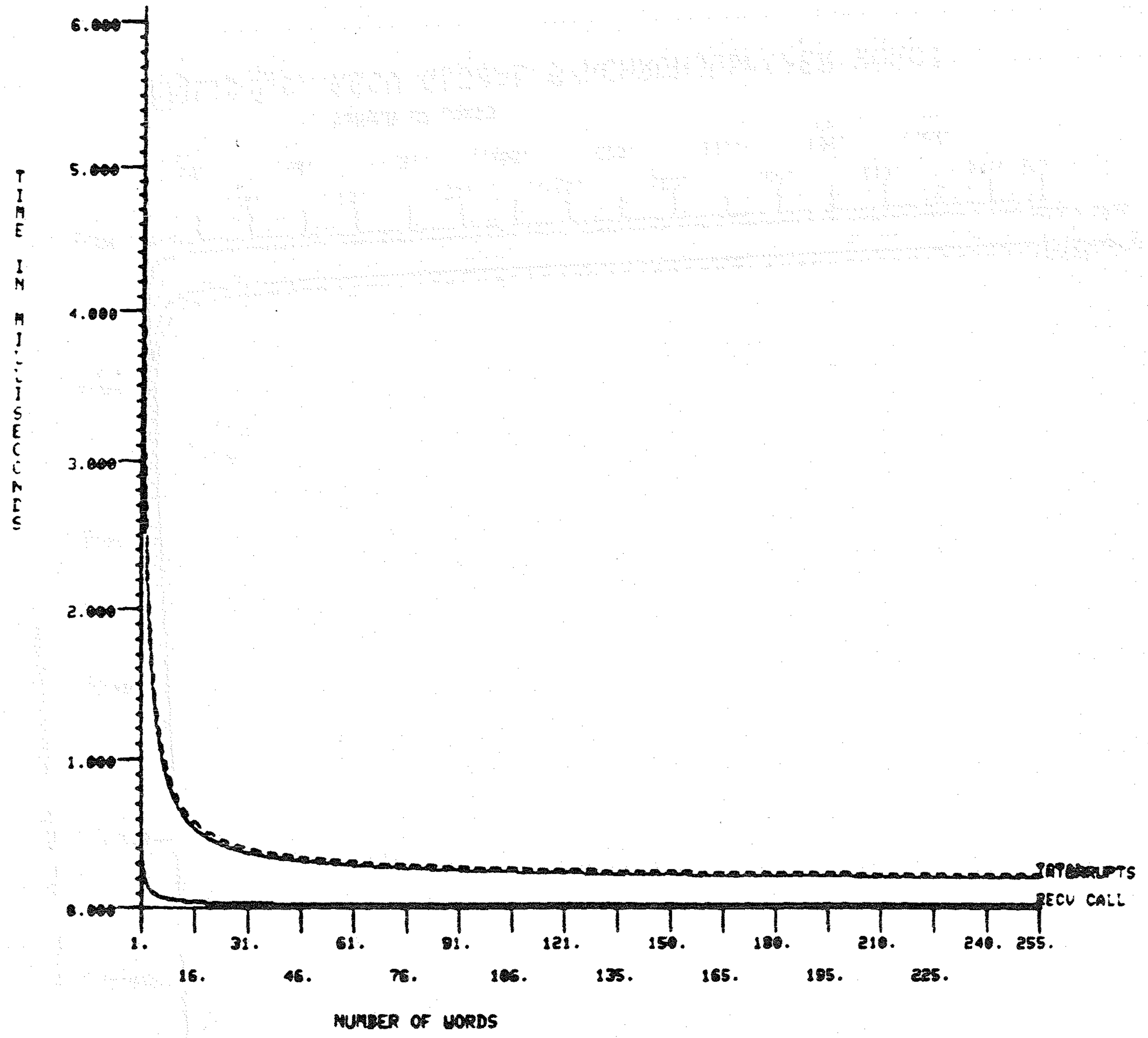

Figure 6.b. RECV LOCAL ASYNCHRONOUS (PER WORD)

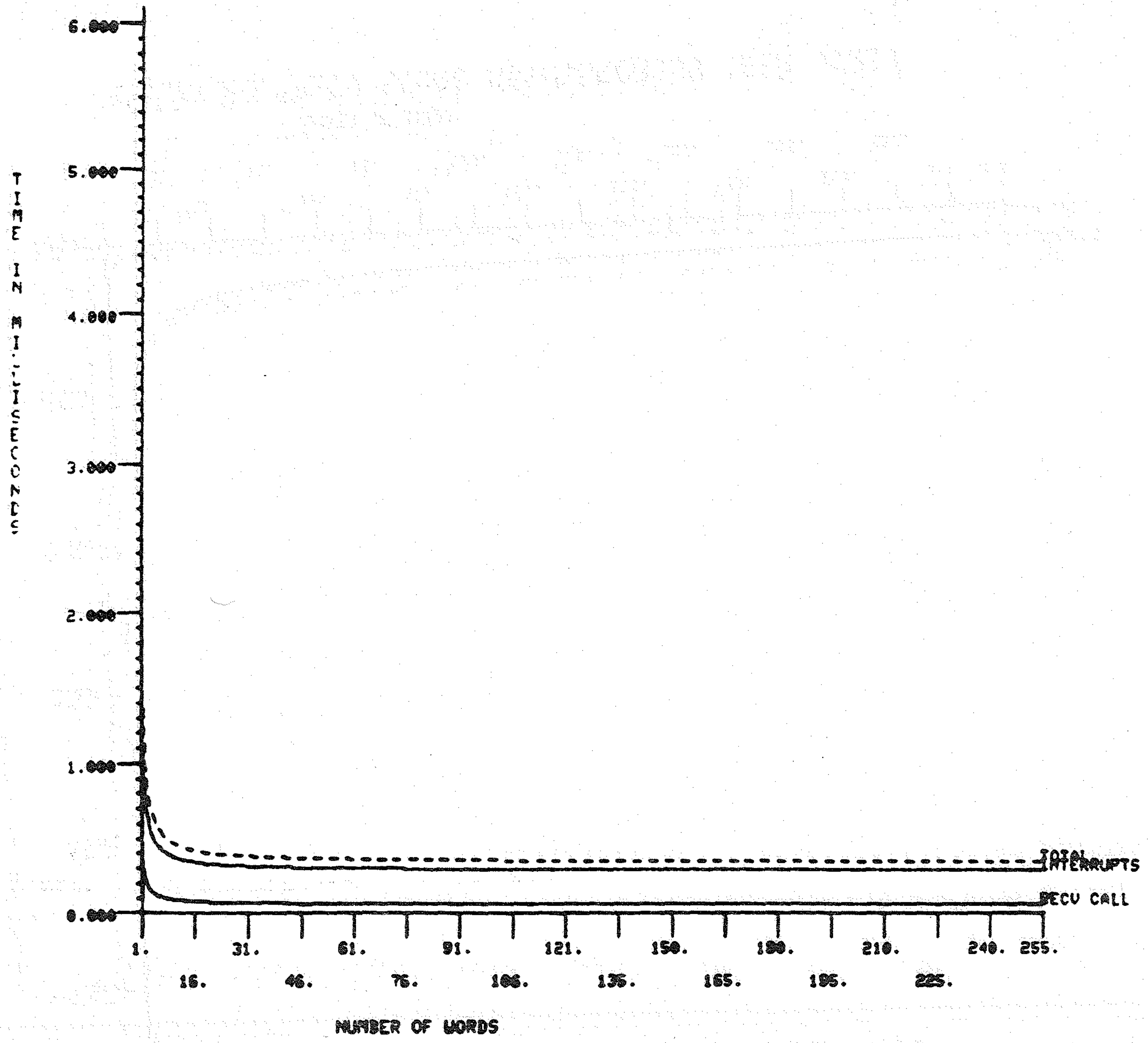

Figure 6.c. RECV GLOBAL SYNCHRONOUS (PER WORD)

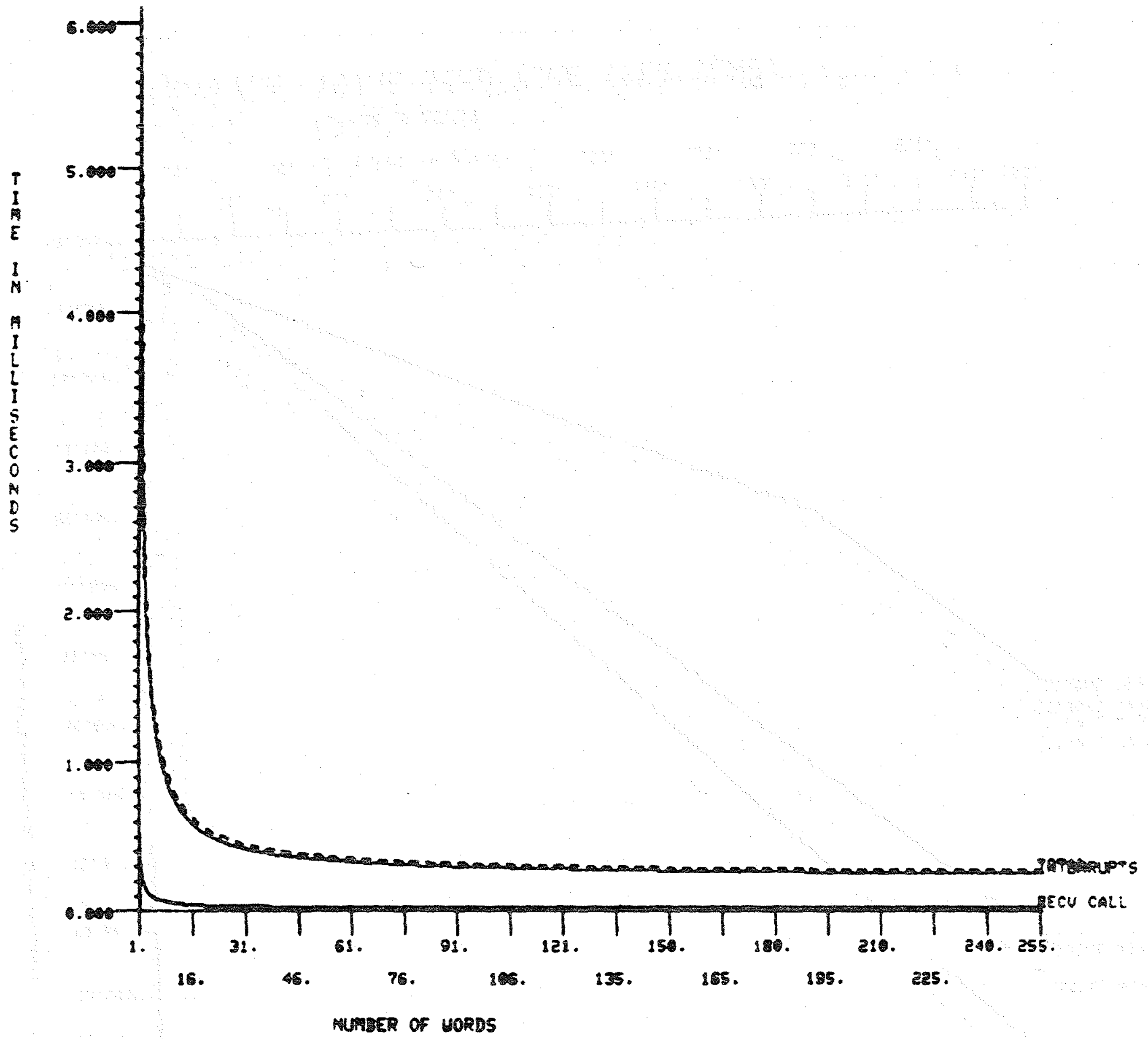

Figure 6.d. RECV GLOBAL ASYNCHRONOUS (PER WORD)

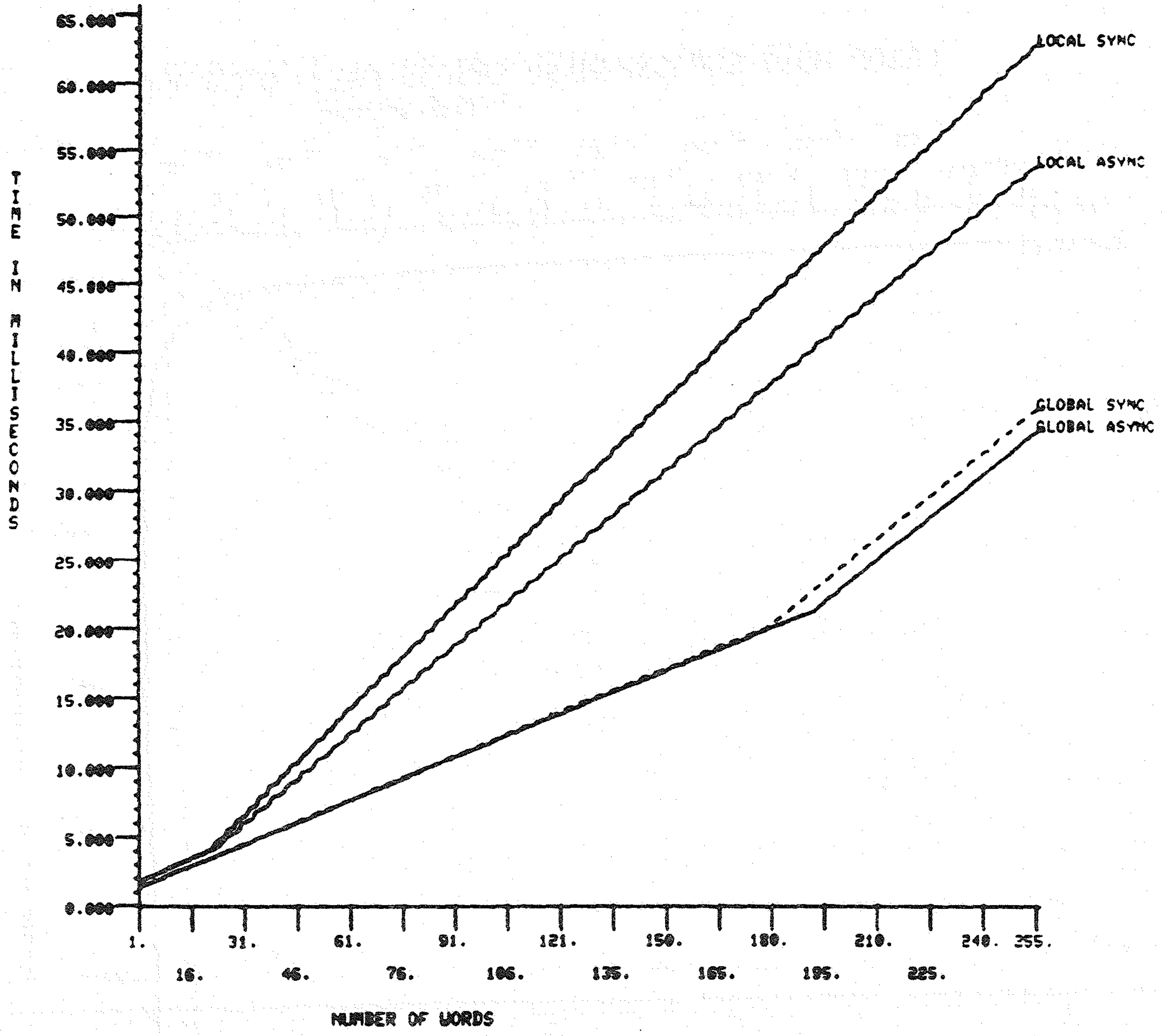

Figure 7.a. TOTAL SEND TIME (PER SEND)

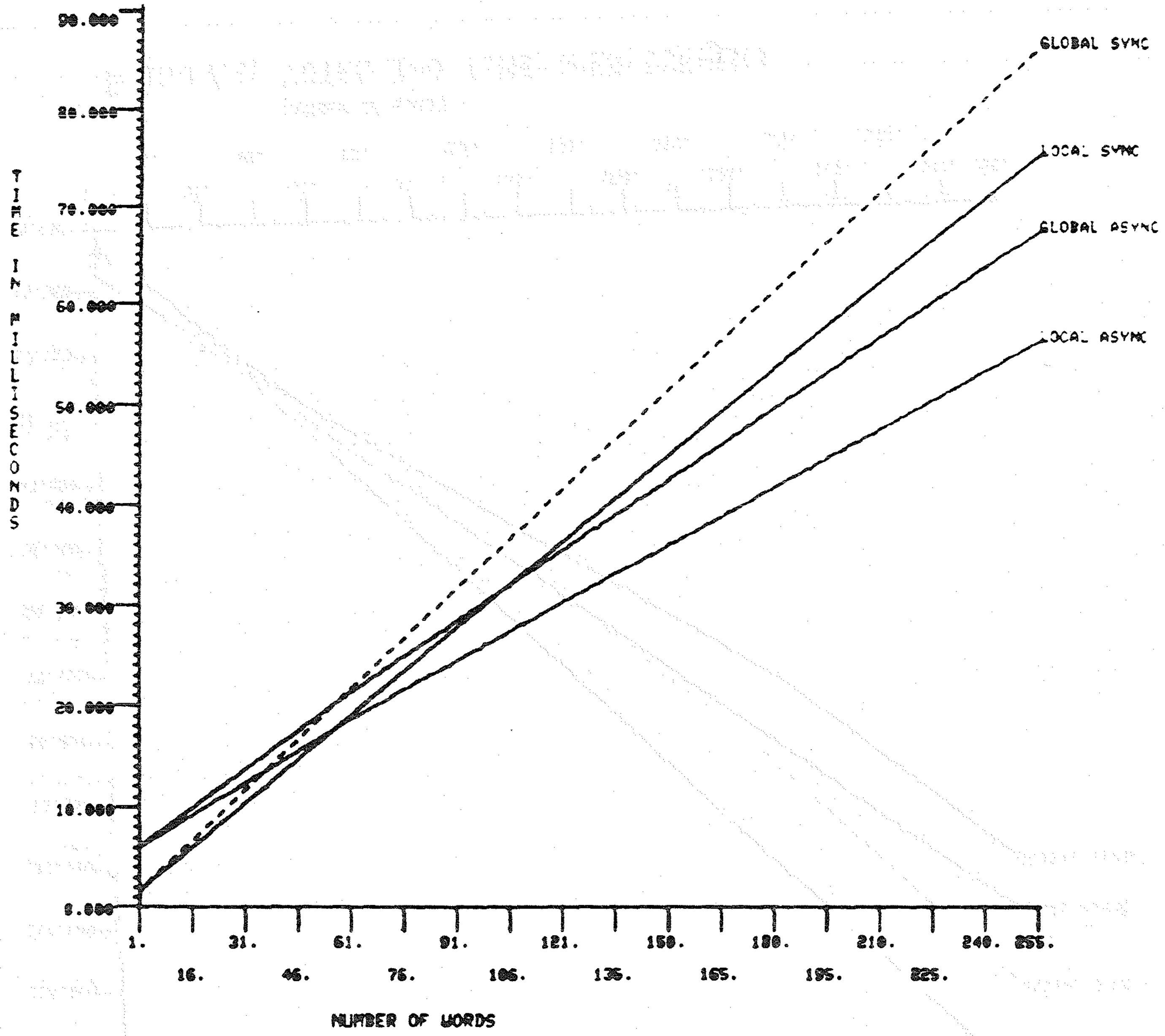

Figure 7.b. TOTAL RECV TIME (PER RECV)

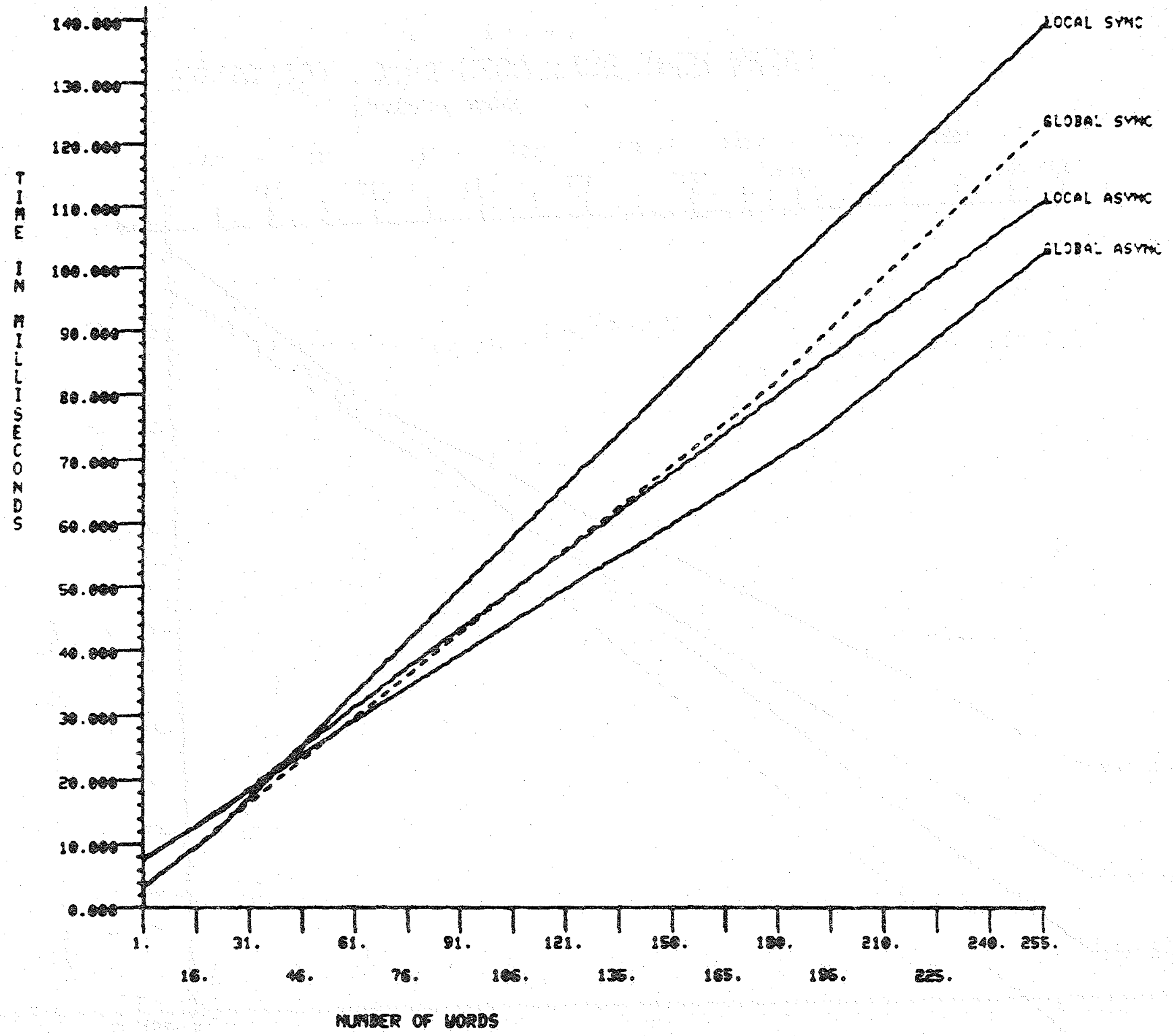

Figure 7.c. TOTAL I/O TIME (PER BUFFER)

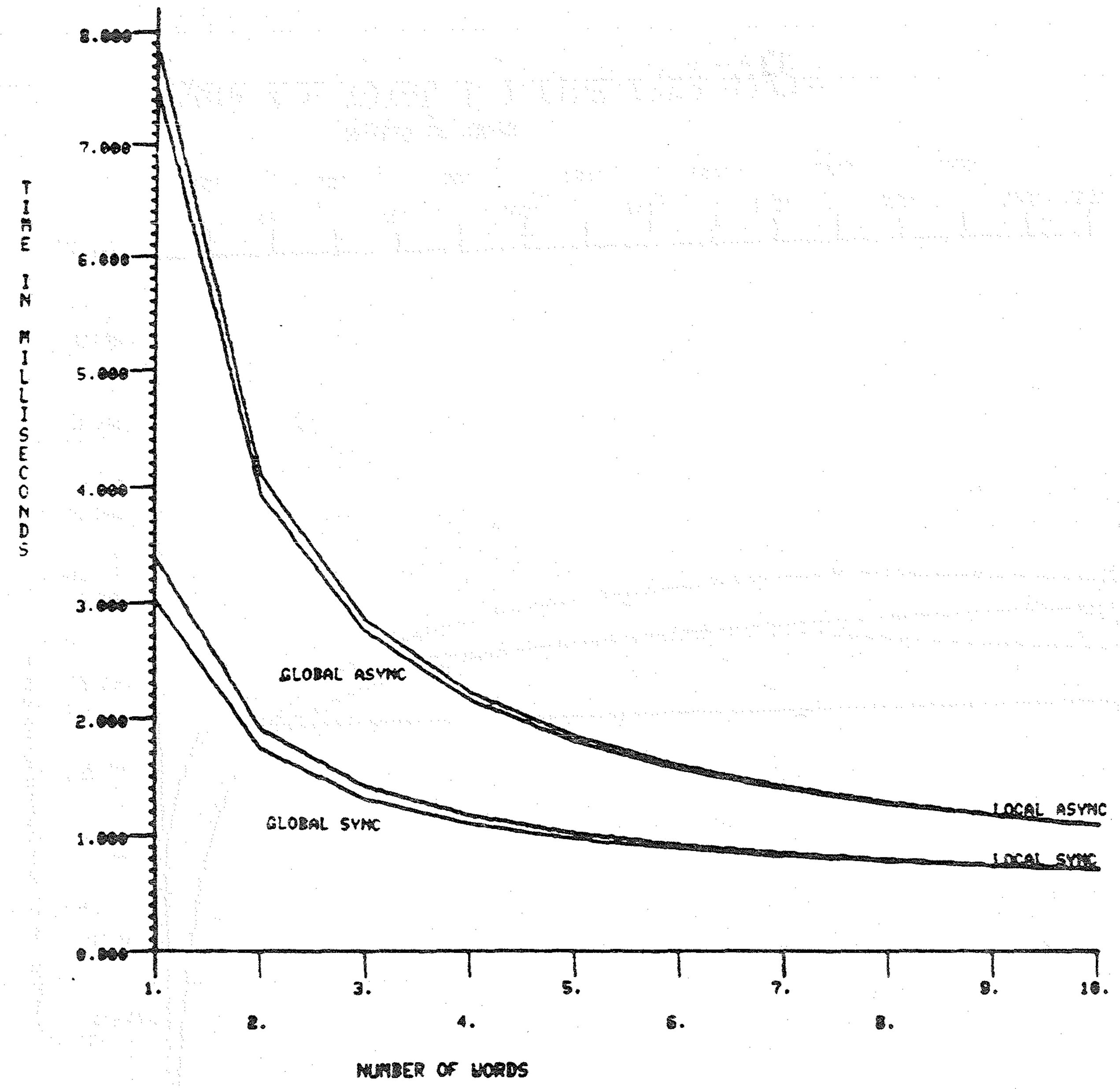

Figure 8.a. TOTAL I/O TIME (PER WORD)

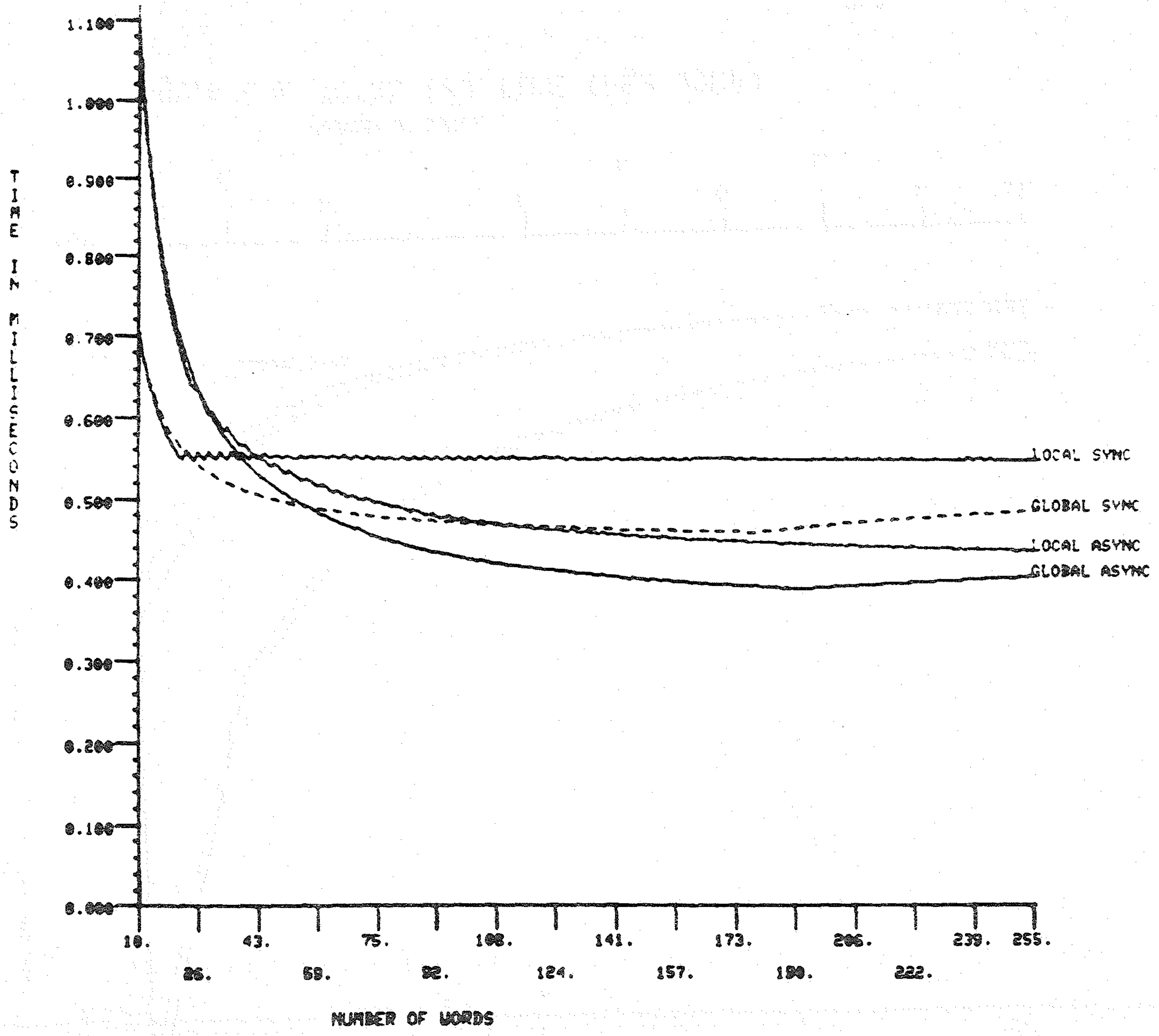

Figure 8.b. TOTAL I/O TIME (PER WORD)

1 Report No NASA CR-172205	2 Government Accession No	3 Recipient's Catalog No
4 Title and Subtitle A Performance Analysis of the PASLIB Version 2.1X SEND and RECV Routines on the Finite Element Machine		5 Report Date August 1983
		6 Performing Organization Code
7 Author(s) J. D. Knott		8 Performing Organization Report No
9 Performing Organization Name and Address Kentron International, Inc. Kentron Technical Center 3221 N. Armistead Ave. Hampton, VA 23666		10 Work Unit No
		11 Contract or Grant No NAS1-16000
12 Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546		13 Type of Report and Period Covered Contractor Report
		14 Sponsoring Agency Code 505-37-13-01

15 Supplementary Notes

Langley Technical Monitor: Dr. Olaf O. Storaasli

16 Abstract

The Finite Element Machine is an experimental array processor designed to support research in parallel algorithms and architectures. This report presents a case study of communications using the SEND and RECV system software routines on the Finite Element Machine, followed by a discussion of the effect of I/O performance on the efficiency of parallel algorithms.

17 Key Words (Suggested by Author(s)) parallel array processor, parallel architecture, Finite Element Machine		18 Distribution Statement Unclassified - Unlimited Subject Category - 60	
19 Security Classif (of this report) Unclassified	20 Security Classif (of this page) Unclassified	21 No of Pages 33	22 Price A03

End of Document