DYNAMIC FUNCTION EXCHANGING MECHANISM IN POLY-PROCESSOR SYSTEM

M. Sato, S. Nishikawa, K. Murakami and S. Takahira

Musashino Electrical Communication Laboratory, NTT
Musashino-shi, Tokyo, 180
Japan

Abstract

Microprogramming is a basic technique that realizes functionally specialized processors in a functionally distributed multiprocessor system which consists of many small processors. In such a multiprocessor system, it is necessary to change processor functions dynamically in order to back up a heavily loaded processor or a failed processor by a lightly loaded processor. This paper proposes a mechanism for exchanging processor functions in firmware level and shows a hardware organization which realizes the mechanism in an experimental system. In order to clarify the design conditions of an operating system and microprograms, the performance of the mechanism is analyzed from the load balancing point of view. Moreover, the conditions which the mechanism must satisfy are derived.

1. Introduction

In this paper, the Poly-Processer System (PPS) is defined as a computer complex consisting of many small, tightly coupled, and functionally dedicated processors. The objective of PPS is to realize a computer system which has easier expandability, higher reliability, and better cost to performance ratio than existing systems [1,2,3].

In PPS, microprogramming is one of the basic techniques, since it is useful for realizing functionally specialized processors without changing processor hardware and is thought to attain an economical advantage by using a large amount of LSIs of the same kind. The results of PPS software simulation show that the load for each processor varies according to the type of user program involved. Therefore, several kinds of load balancing mechanism are necessary to avoid overload for each processor [1]. Since there is a possibility that some processor failures could completely halt system operations, some system reconfiguration mechanisms are necessary [3]. In PPS, in order to realize the above mechanisms, a lightly loaded processor is used to back up a heavily loaded processor or a failed processor. Hence, it is necessary that microprograms can be dynamically changed in order to change processor functions dynamically.

Many schemes, which alter microprograms dynamically, have been proposed. These schemes have the following merits and demerits.

(1) The scheme, in which control storage is implemented as being in the same address space as the main memory, provides the capability of writing or reading microprograms into the writable control storage (WCS) like a main memory [4]. However, it is difficult to prevent some failures in one space from spreading to another space effectively.

(2) The other scheme, in which control storage is separated from main memory in terms of physical construction and addressing logic, provides the capability of preventing software failures from spreading into microprograms. In this scheme, micrprograms can be loaded into WCS by calling utility programs that treat WCS as an I/O device [5]. However, it is inefficient to load a variable small number of microinstructions into WCS.

This paper proposes a mechanism for exchanging processor functions in firmware level and shows the hardware construction realizing the mechanism in a PPS pilot model system. Characteristics of the mechanism were analyzed from the viewpoint of effects on load balancing among processors. Moreover, the conditions which the mechanism must satisfy are shown.

2. Address Mapping Mechanism

In PPS pilot model, an address mapping mechanism is provided for each processor to enable dynamic function exchanging.

2.1 Addressing Mechanism

The address mapping mechanism allows use of both control storage and main memory interchangeably and keeps control storage space separated from main memory space.

As illustrated in Fig. 1, the mechanism defines bus windows. Window (Wm) maps main memory space (a-b) to control storage space (c-d). Window (Wc) maps control storage space (e-f) to main memory space (g-h). Word size of the control storage is n times as large as that of the main memory. A control storage word is divided into n blocks of equal size. Windows are opened under the control of an operating system or microprograms, only when necessary to operate the mechanism.

CH1394-6/79/0000-0130 $00.75©1979 IEEE

The mechanism operates as follows:

(1) If operand address X in the main memory space is a X b, data is fetched from the (X-a)(mod. n)-th block of c+[(X-a)/n] in the control storage space.

(2) If the next micronstruction address Y in the control storage space is e Y f, the five words beginning from g+n(Y-e) in the main memory space are fetched and executed.

Dynamic microprogramming, which uses the address mapping mechanism, has the following advantages.

(A1) Since control storage space and main memory space are separated from each other, the mutual interference between microprograms and programs can be decreased.

(A2) Since control storage is accessible, as if it were a main memory, it is easy to load microprograms dynamically under program control.

(A3) Since microprograms in main memory can be directly executed, the microprogram execution area is enlarged.

(A4) The mechanism provides operating system software with two means of exchanging processor functions. One is overlaying microprograms and the other is direct execution of microprograms in main memory.

2.2 Hardware Organization

The processor organization of PPS pilot model is shown in Fig. 2. The processor is a conventional microprogrammed minicomputer, to which the address mapping mechanism is added. The address mapping hardware consists of an extended control storage unit (ECS) and an address space control unit (ASC). ASC is connected to the main memory by the memory bus (M-bus). ECS is connected to ASC by the bus transmitting microprogram data (MP-bus). Hardware characteristics are described in Table 1.

Extended Control Storage Unit. As shown in Fig. 2, ECS consists of writable control storage and a controller (ECSC). ECSC activates writable control storage read/write operations or microinstruction execution, as ordered by ASC. If the operation is finished, ECSC reports machine states or a next microinstruction address to ASC.

Address Space Control Unit. As shown in Fig. 2, ASC consists of an address mapping module, an M-bus control module, a data transmission bus and a controller (ASCC).

(S1) The address mapping module performs two kinds of mapping. One is mapping from main memory space to control memory space. An operand address on M-bus is examined to determine whether it is in window (Wm) or not. If the address is in the window and the window is open, the signal for enabling mapping, or Gmc is activated, is sent to the M-bus control module. At the same time, the operand address is transformed to a control storage address and block number pair. The other is mapping from control memory space to main memory space. The next microinstruction address on MP-bus is examined to determine whether it is in window (Wc) or not. If the address is in the window and the window is open, the signal for enabling mapping, or Gcm is activated, is sent to the ASCC. At the same time, the microinstruction address is tranformed to a main memory address.

(S2) The M-bus control module either controls accesses issued from the ASC to the main memory or makes a response to accesses issued from the CPU to the control storage. In case of acces control, if Gcm is activated, the M-bus control module begins main memory access sequences under the control of the ASCC. In case of response operation, if Gcm is activated, the M-bus control module sends an acknowledge signal to the CPU instead of to the main memory and also sends a signal to the ASCC to begin a control storage access operation.

(S3) The ASCC operates as follows: When the ASCC receives a termination signal for a microinstruction, it examines Gcm. If Gcm is not activated, it sends an initiation signal for the next microinstruction to the ECS. If Gcm is activated, it fetches a next microinstruction from the main memory through the M-bus control module and sends it to the ECS. Then, it sends an initiation signal for the next microinstruction to ECS. When the ASCC recognizes that Gmc is activated, it sends a signal to the ECS to initiate a control memory access operation.

2.3 Firmware Organization

The emulator program structure is shown in Fig. 3 [6]. In PPS, semantic routines are divided into two classes. One class is a set of semantic routines, Basic Semantic Routines, which realize the basic functions common to all processors. The other class is a set of semantic routines, Optional Semantic Routines, which realize the functions local to each processor, or microprogrammed operating system functions. Copies of the optional semantic routines are stored in the common memory. The functions of each processor are dynamically changed by exchanging optional semantic routines.

As described before, in PPS pilot model, there are two methods of dynamically exchanging processor functions; (1) mapping control storage references into main memory, thus switching which microprogram is being executed, and (2) mapping main memory into control storage, so that microprograms can be over-written.

3. Effects on Load Balancing

The address mapping mechanism may affect system characteristics, such as allowed performance and load imbalance. Therefore, its effects must be analyzed and requirements for a load balancing scheme must be investigated for designing an operating system and firmware.

3.1 Analysis Model

From the load balancing point of view, the system is defined to be in a balanced state when an average response time of each processor is equal to that of all the others and less than the fixed value, or permitted limit for response time: Rt. As shown in Fig. 4, a load balancing process is modeled that a request to a heavily loaded processor, CPU-b, is processed at random by a lightly loaded processor, CPU-a, with probability X.

Followings are assumed in this model:

(P1) Arrivals of type A requests follow a Poisson process with rate λa, and the service time is exponentially distributed with average service time $1/\mu a$. Arrivals of type B requests follow a Poisson process with rate λb, and the service time is exponentially distributed with average service time $1/\mu b$. Expected response time for each processor is equal to Rt.

(P2) The values of λa, λb, μa and μb are changed without delay.

(P3) The overhead time for overlaying is in proportion to the size of overlayed microprograms. The size of microprograms to be overlayed is in proportion to the number of its running steps. The ratio of the overhead time to the original processing time is called "overlaying overhead", V.

(P4) The processing time required for a processor to execute other processor functions needs more processing time than the original processing time. The ratio of the incremental time to the original time is called "substituting overhead", U.

(P5) A processor must not exchange its functions while it is processing a request.

(P6) When processor functions are exchanged, a system is assumed to be in statistical equilibrium without delay.

From the above assumptions, the average CPU-a service time varies as follows;

$$1/\mu a'(A\to A)=(1-\Delta\mu a)/\mu a \tag{1}$$

$$1/\mu a'(A\to B)=(1+\Delta\mu b)(1+V+U)/\mu b \tag{2}$$

$$1/\mu a'(B\to B)=(1+\Delta\mu b)(1+U)/\mu b \tag{3}$$

$$1/\mu a'(B\to A)=(1-\Delta\mu a)(1+V)/\mu a \tag{4}$$

where $\mu a'(A\to B)$, for example, represents the average service rate, when a B-type request is processed after an A-type request.

3.2 Expected CPU-a Response Time

Two types of requests arrive at CPU-a at random and independently. Queue discipline is first-come, first-served. The changeover times are involved in changes from type-A to type-B, and from type-B to type-A. This kind of queueing model with service orientation times was analyzed by d. p. Gaver [7].

In the following, it is assumed that overlaying of microprograms corresponds to the case where U=0; and direct execution of microprograms in main memory corresponds to the case where V=0. For simplicity, it is assumed that μa equals μb and that both $\Delta\mu a$ and $\Delta\mu b$ equal 0.

Microprogram Overlaying. The expected CPU-a response time, Rt'(A), in case of microprogram overlaying (U=0) is computed as

$$Rt'(A)=\frac{1}{\lambda a'(1-\rho a')}\Big[\rho a'+\frac{\lambda a'^2}{\mu a^2}-\rho a'^2 + 3\frac{\lambda a'(A)\lambda a'(B)}{\mu a^2}(V^2+2V)-2V\lambda a'(A)\lambda a'(B) + \frac{V}{\mu a}\{Fa(0)\lambda a'(B)+Fb(0)\lambda a'(A)\}\Big], \tag{5}$$

where

$$\rho a'=\frac{\lambda a'}{\mu a}+\frac{2V\lambda a'(A)\lambda a'(B)}{\lambda a'\mu a},$$

$$\lambda a'(A)=\lambda a-\Delta\lambda a,$$

$$\lambda a'(B)=\chi(\lambda b+\Delta\lambda b),$$

$$\lambda a'=\lambda a'(A)+\lambda b'(B),$$

and Fa(0) and Fb(0) represent boundary conditions for the steady state equations.

If the total arrival rate, $\lambda a'$, is constant, Rt'(A) takes a maximum value, where $\lambda a'(A)$ is equal to $\lambda a'(B)$.

Direct Execution. The expected CPU-a response time for direct execution of microprograms in main memory (V=0) is computed as

$$Rt'(A)=\frac{1}{\lambda a'(1-\rho a')}\Big\{\rho a'+\frac{\lambda a'^2}{\mu a^2}-\rho a'^2 + \lambda a'(A)\lambda a'(B)(U^2+2U)\Big\}, \tag{6}$$

where

$$\rho a'=\frac{\lambda a'}{\mu a}\,\frac{U\lambda a'(B)}{\mu a}$$

3.3 Expected CPU-b Response Time

Since the CPU-b service is modeled as M/M/1, the exepected response time, Rt'(B), is computed as

$$Rt'(B)=\Big\{\frac{\mu b}{1+\Delta\mu b}-(1-\chi)(\lambda b+\Delta\lambda b)\Big\}^{-1}. \tag{7}$$

3.4 Total System Request

Let $\lambda t'$ represent the maximum value of the total arrival rate the system can service under the limited response time constraint, Rt. Then, $\lambda t'$ is

$$\lambda t' = (\lambda a - \Delta\lambda a) + (\lambda b + \Delta\lambda b\text{-max}) \tag{8}$$

where $\Delta\lambda b$-max is the maximum increment of λb. To that extent, CPU-a can process B-type requests.

Microprogram Overlaying. The results of paragraph 3.2 show that, if the total load for CPU-a is constant, the expected CPU-a response time takes a maximum value when B-type load is equal to A-type load. Hence, $\lambda t'$ takes the minimum value:

$$\lambda t' = \frac{1-(\mu a-\lambda a)(1+\frac{V}{2})\mu a^{-1}}{\frac{1+\frac{V}{2}}{\mu a}+\frac{\mu a-\lambda a}{\mu a^2}\frac{V^2}{4}} . \tag{9}$$

Direct Execution. From Eq. (6), it is shown that $\lambda t'$ decreases as $\Delta\lambda a$ increases. Therefore, when CPU-a processes only B-type requests, $\lambda t'$ takes a minmum value:

$$\lambda t' = \frac{1-(\mu a-\lambda a)(1+2U)\mu a^{-1}}{\frac{1+U}{\mu a}-\frac{\mu a-\lambda a}{\mu a^2}(2U+U^2)} . \tag{10}$$

Numerical results of $\lambda t'$ are shown in Fig. 5, where

$$\lambda t = \lambda a + \lambda b .$$

In case of no dynamic function exchanging, $\lambda t'$ is

$$\lambda t' = \lambda a - \Delta\lambda b + \lambda b . \tag{11}$$

3.5 Applying Conditions

In the following, conditions are derived for U and V to satisfy when processor functions are exchanged dynamically in order to balance the load.

It is assumed that the system operates under the following conditions:

(C1) In any imbalanced state, it is assured that a ratio of a total system load to that of the balanced state is greater than a constant value, L.

(C2) The utilization factor, for each processor, is equal to ρ in a balanced state.

(C3) For parameters shown in Fig. 4, it is assumed that λa is equal to λb and that μa is equal to μb.

To satisfy the condition (C1) by means of the overlay, from Eq. (9), V is required to satisfy

$$0 \le V \le \frac{\rho - 1 + (1-\rho)^2 + 8(1-L)(2L-1)\rho(1-\rho)}{(2L-1)\rho(1-\rho)} \left(\equiv Vu\right) . \tag{12}$$

To satisfy the condition by means of the direct execution of microprogram in main memory, from Eq. (10), U is required to satisfy

$$0 \le U \le \frac{1}{2(2L-1)\rho(1-\rho)}\left\{2(2L-1)\rho^2-(2L+1)\rho+2 - \sqrt{\{2(2L-1)\rho^2-(2L+1)\rho+2\}^2-8(2L-1)(1-L)\rho^2(1-\rho)}\right\} \left(\equiv Uu\right) . \tag{13}$$

Next, the implications of conditions for U and V are examined. Parameters, V and U, are approximated by

$$V = \frac{So \cdot W}{n(Ds+Dc)E} + \frac{C \cdot E}{n(Ds+Dc)E} , \tag{14}$$

$$U = \frac{So}{So+Sb} \cdot \frac{W}{E} \cdot \frac{Ds}{Ds+Dc} . \tag{15}$$

Where, the means of variables are follows:

- Sb: Average size of total semantic routines common to all processors.
- So: Average size of total semantic routines which characterize functions local to a processor.
- Dc: Sum of the average number of running steps for the control process and that for the decoding process.
- Ds: Average number of running steps for semantic routines.
- N: Average number of software running steps per request.
- W: Time required to load a microinstruction into writable control storage: this is nearly equal to the time required to fetch a microinstruction from the main memory.
- C: Average number of running steps required to set up microprogram overlaying.
- E: Average execution time of a microinstruction which resides in writable control storage.

If the system has been constructed, parameters have fixed values, except for So. Therefore, Eq. (12) gives

$$So \le Vu \frac{E}{W}\{N(Ds+Dc)-C\} , \tag{16}$$

and Eq. (13) gives

$$So \le Sb\left(\frac{1}{Vu}\frac{W}{E}\frac{Ds}{Ds+Dc}-1\right)^{-1} . \tag{17}$$

Table 2 shows numerical examples of these parameters. Values for W and E are determined from Table 1, respectively. Experiments with PPS pilot model give the values of Sb, Dc and Ds. The results of the software simulation give the value of N.

Above results show that the direct execution of microprograms in the main memory can have a load balancing effect, only when the size of microprograms located in the main memory is less than 13.5% of that of basic semantic routines. On the other hand, the overlaying of microprograms can have an effect when the size of microprograms to be loaded is less than 7% of the total microprogram running steps per request. Therefore, the overlaying is effective in general cases, while the direct execution of microprograms in the main memory is effective only in a case where there are few functional differences between processors.

The results of paragraph 3.4 show that, as more B-type requests are processed by CPU-a, the total system load takes a minimum, in case of overlaying, and decreases monotonically, in case of direct execution. Therefore, more load balancing effects can be gained, if both schemes are used interchangeably. The former is used when B-type requests are less than A-type requests and the latter is used when B-type requests are more than A-type requests.

4. Conclusion

In PPS, microprogramming technique plays a major role in realizing functionally specialized processors. Moreover, it is necessary to change microprograms dynamically in order to back up a failed or heavily loaded processor by a lightly loaded processor. This paper proposes an address mapping mechamism as a dynamic function exchanging mechanism in the PPS pilot model system, and the implementation of the mechanism is described. In addition, effects of the mechanism on load balancing are analyzed, and conditions which the mechanism must satisfy are obtained.

The conditions of applying the mechanism are shown as follows.

		Load Imbalance Degree	
		Small	Large
Functional Difference Between Processors	Small	o	*
	Large	*	*

o: A suitable area for direct execution of microprogram in the main memory.
*: A suitable area for overlaying.

Load balancing programs are now under development in the operating system of PPS pilot model. The important problems to be solved are the detection method of load imbalance and the scheduling policy. In this paper, it is assumed that a request to a heavily loaded processor is selected at random and serviced on a first-come, first-served discipline by a lightly loaded processor. Since changeover times are incurred every time the service changes between two types of requests, this scheduling policy may be inefficient. Hence, it seems better to adopt another policy in which one type of request is processed successively for some time duration once a change occurred.

Acknowledgement

The authors wish to thank the members of the First Research Section for the construction of PPS pilot model hardware.

References

[1] K.Murakami, S.Nishikawa and M.Sato: Poly-Processor System Analysis and Design, Proc. of 4th Annual Symposium on Computer Architecture, PP. 49-56, 1977.

[2] S.Nishikawa, M.Sato and K.Murakami: Interconnection Unit for Poly-Processor System: Analysis and Design, Proc. of 5th Annual Symposium on Computer Architecture, PP. 216-222, 1978.

[3] S.Takahira, K.Murakami, S.Nishikawa and M.Sato: A Reliability Aspect of Function Distribution System: PPS, Proc. of 3rd UJCC, 1978.

[4] A.B.Tucker and M.J.Flynn: Dynamic Microprogramming: Processor Organization and Programming, CACM, 14, 4, PP. 240-250, 1971.

[5] A.K.Agrawala and T.G.Raucher: Foundations of Microprogramming, Academic Press, 1976.

[6] V.R.Lessor: An Introduction to the Direct Emulation of Control Structures by a Parallel Microcomputer, IEEE Tans. on Computers, C-20, 7, PP. 751-764, 1971.

[7] D.P.Gaver, Jr.: A Comparison of Queue Disciplines When Service Orientation Times Occur, Naval Research Logistics Quartery, 10, PP. 219-235, 1963.

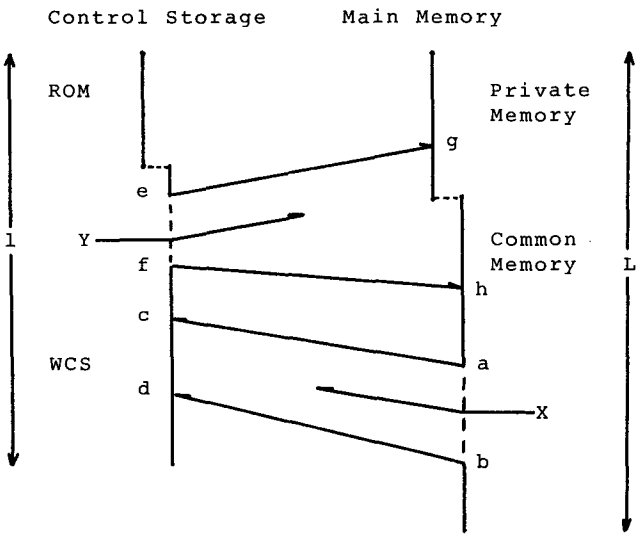


Fig. 1 Address Space Organization

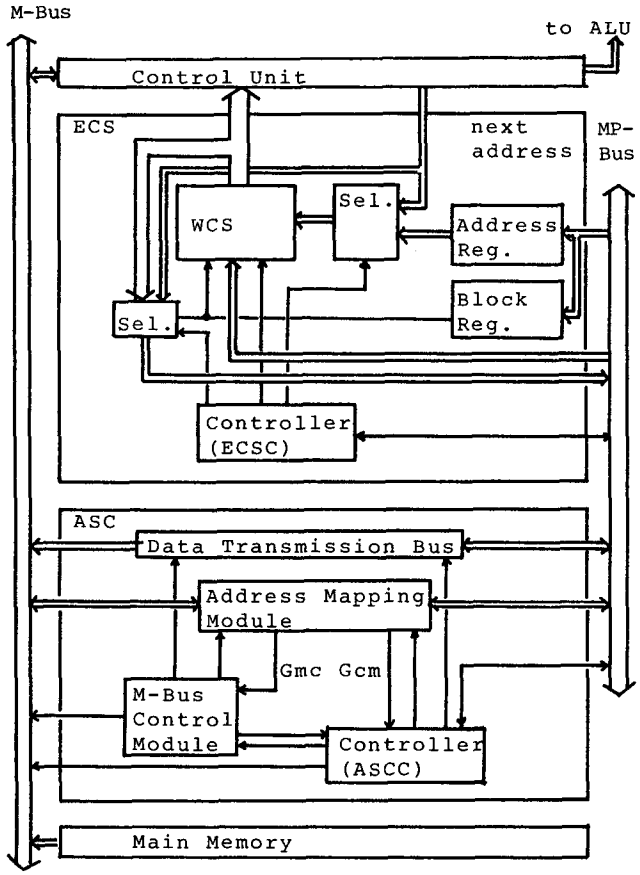
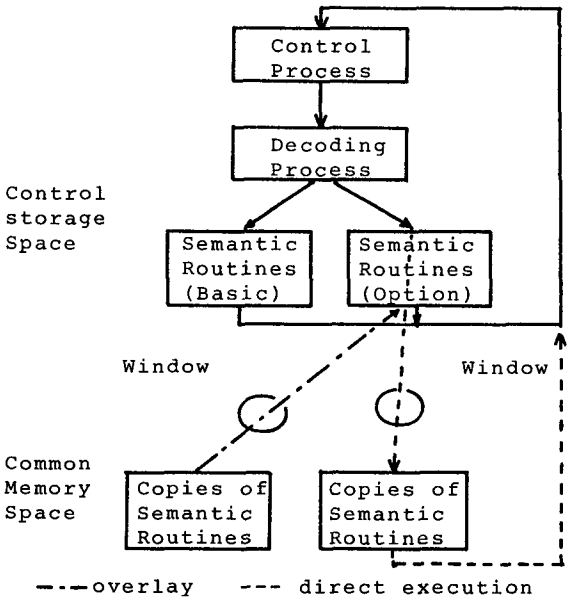


Fig. 2 Processor Hardware Oarganization



. Fig. 3 Firmware Organization

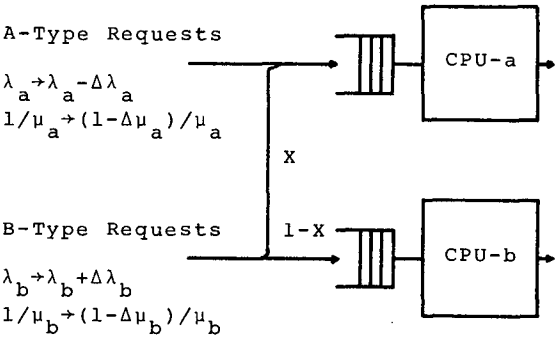


Fig. 4 Analysis Model

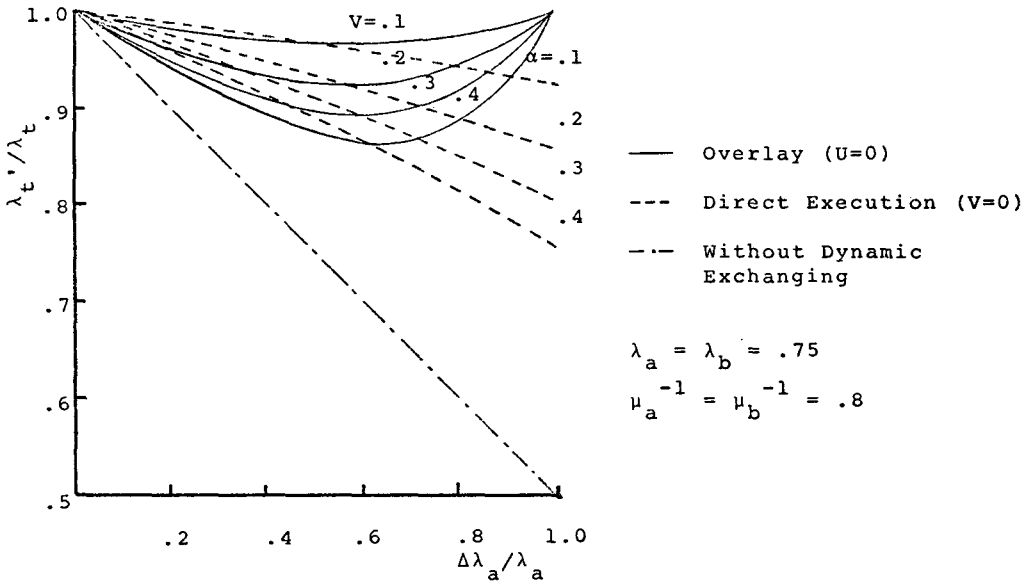


Fig. 5 Total Request Rate

Table 1 Hardware Characteristics

		Cycle Time	Word Size	Capacity	Comments
Control Storage	ROM	240ns	80bits	256w/PU	1=2kw
	WCS	480ns	80bits	1kw/PU	n=5
Main Memory	Private	1.2μs	16bits	8kw/PU	L=64kw
	Common	1.2μs	16bits	7x8kw	

Table 2 Numerical Examples

Parameter	L	ρ	Sb (w)	Dc (step)	Ds (step)	N (step)	W (μs)	C (step)	E (μs)
Value	0.8	0.8	388	17.89	19.29	211.8	6	67	.48
Overlay	Vu = .8858, So < 553.3								
Direct Execution	Uu = .7734, So < 52.5								