PROGRAM RESTRUCTURING IN A MULTILEVEL VIRTUAL MEMORY

by

Edwin J. Lau and Domenico Ferrari

Memorandum No. UCB/ERL M81/26

May 1981

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Program Restructuring in a Multilevel Virtual Memory†

Edwin J. Lau††
Bell Laboratories
Naperville, Illinois 60566

Domenico Ferrari
Computer Science Division
and Electronics Research Laboratory
University of California
Berkeley, California 94720

ABSTRACT

Program restructuring techniques have proven successful in two-level automatically managed memory hierarchies. The possibility of extending them to multilevel environments is investigated. The performance of strategy-oriented restructuring algorithms in a three-level linear hierarchy managed by sampled working set policies or by a combination of sampled working set and local LRU policies is studied both analytically (assuming an independent reference model of program behavior) and by trace-driven simulation. The results of the study show that, for most programs, strategy-oriented restructuring may be as beneficial in a virtual memory with three levels as it is in one with two levels.

Key Words and Phrases: virtual memory, multilevel memory hierarchy, program restructuring, restructuring algorithms, working set policy, LRU policy, page replacement, independent reference model.

1. Introduction

Virtual memory is the computer system feature which makes the storage space at the disposal of a programmer independent of the main memory space assignable to a process. Its implementation requires, besides a hardware mechanism for virtual to real address mapping, at least two levels of memory and a policy for their automatic management. Sometimes, however, two levels are not sufficient, or a multilevel solution is more convenient than a two level one [GECJ74]. The latter would be the case for a system with so large a virtual address space and so large a number of processes that providing for their storage in level II would either be uneconomical or degrade system performance intolerably. This would be the normal case if we considered file space an indistinguishable part of a process' address space instead of treating files as external entities to be explicitly bound to that address space when needed.

A multilevel solution is unavoidable when the system itself has a multilevel control structure. One example is that of systems which support virtual machines, where the global operating system implements virtual memory for the various machines, each of which may offer virtual

† The work reported here was supported in part by the National Science Foundation, Grants DCR74-18375 and MCS78-24618.

†† Work performed when this author was with the Computer Science Division and the Electronics Research Laboratory, University of California, Berkeley, California.

memory facilities to its users [GOLR74]. Another example is found in systems programs like data base management systems, which run with other applications under a general operating system, thereby exploiting its virtual memory feature, and at the same time support the virtual memories of their users [TUEW76]. In these two examples, the level II memory does not have to be (and generally is not) physically distinct from level I or level III. However, from the conceptual viewpoint as well as from a number of practical viewpoints, there is no difference between a three level hierarchy mapped onto a two level physical hierarchy and one implemented by three physical levels. To improve the performance of two level virtual memories, the program restructuring approach has been proposed and proved to be successful (see, for example, [FERD76b]). This paper studies the possibility and the effectiveness of applying the same approach to memory hierarchies with more than two levels. Before describing the study and its results, we shall in Section 2 discuss the principles of restructuring and present in Section 3 an analytic investigation of the properties of two restructuring algorithms which have been quite successful in two level hierarchies.

2. Program Restructuring Principles

Program restructuring consists of relocating various portions of a program (both code and data) with respect to one another to alter the program's locality. Dynamic restructuring refers to those approaches which, in some way, utilize the actual execution time addressing characteristics of the program in order to determine its new layout in the virtual address space. Off-line restructuring procedures are those which are not performed while the program is in execution, but as an entirely separate operation. In the sequel, off-line dynamic restructuring will simply be referred to as program restructuring.

Program restructuring is typically performed in the following five steps:

1. The program, both code and data, is *partitioned* into blocks. Natural blocks occur in programs in the form of procedures, functions, rows of arrays, and so on. Typically, the size of a block should be less than that of a page.
2. A *trace* of the program is then collected. The trace may be a full address trace or only a trace of the references to the blocks described in step 1. The trace represents the dynamic addressing characteristics of the program.
3. Utilizing a *restructuring algorithm* which works on the trace gathered in step 2, a graph containing information about the desirability of placing blocks together is generated. The graph is known as the *restructuring graph*. The nodes of the graph represent the blocks and the edge weights represent the cost of placing two blocks together. The restructuring algorithm determines the edge costs.
4. A mapping of blocks into pages is determined by a *clustering algorithm*. The restructuring graph is clustered with the objective of minimizing the sum of the inter-cluster costs under the constraint that the sum of the block sizes in a cluster not exceed the page size.
5. Finally, the blocks are reloaded according to the layout determined in step 4 and the restructured program is ready for execution.

It has been shown [LAUE79] that, if each page is allowed to contain an arbitrary number of blocks, the clustering problem is NP-complete. Hence, for a large number of blocks, the cost to determine an optimal mapping may be extraordinarily high. However, assuming at most two blocks per page, the problem becomes a weighted matching problem, which may be optimally performed in $O(n^{2.5})$, where n is the number of blocks in the program [EVES75].

Even though it would be advantageous to obtain the optimal ordering, this is not essential. Typically, there is a small number of blocks which, if clustered, will produce a very substantial improvement, and only slight additional improvement usually results from optimal clustering with respect to any suboptimal solution which properly clusters these crucial blocks. Empirical investigations [FERD74,FERD75,FERD76,FERD77] showed that substantial improvement can indeed be obtained using a non-optimal "greedy" clustering algorithm which runs in linear time. This algorithm combines the pair of nodes with the largest edge weight between them into a

single node, whose size is less than or equal to the page size. The weights of the edges connecting other nodes to the combined node are set equal to the sums of the weights the same edges had before the two nodes were combined.

Given a program's address trace, let us assume that the memory policy under which the program will run is known and is such that the blocks which will necessarily be in memory at a given point of the trace can be determined from the knowledge of the trace. In other words, the amount of memory space allotted to a program at each instant of its virtual time is assumed to be unaffected by the behavior of the other programs in the system. Examples of these policies are all the fixed partitioning schemes and the working set and page fault frequency policies. Under this assumption, the effects of restructuring on program behavior and performance can be studied by restricting one's attention to individual programs (for instance, by running uniprogramming experiments). This assumption will be made throughout the rest of this paper.

3. The Critical Working Set and Critical LRU Restructuring Algorithms

Critical set restructuring algorithms [FERD74b] attempt to reduce page faults by clustering blocks which often cause block faults with blocks which are most often in memory when the block fault occurs. This approach is likely to be effective since every page fault corresponds toa block fault and reducing the latter can be expected to reduce the former. Note that the converse is not true, i.e., there are block faults which do not become page faults. Note also that we assume for simplicity that, when a reference to a block is made, the entire block is always brought into memory. In other words, blocks do not cross page boundaries. Two critical set algorithms which are of particular interest in this paper are the critical working set (CWS) algorithm [FERD74a] and critical LRU (CLRU) algorithm [FERD74b].

Let r_i represent the i^{th} reference in the trace, and $B(r_i)$ represent the block referenced at time i. M is the matrix representing the restructuring graph and is called the CWS matrix. $M_{i,j}$ is the entry in row i and column j of M. W(t,T) represents the working set of blocks at time t with window size T. The CWS algorithm constructs the restructuring graph as follows.

1. $k = 0$.
2. $k = k + 1$.
3. If $r_k \epsilon$ W($k-1$,T), then /* r_k is in the block working set */ go to 2.
4. For all $B_l \epsilon$ W($k-1$,T) do /* increment M by 1 */
   $M_{B_l,B(r_k)} = M_{B_l,B(r_k)} + 1$.
5. Go to 2.

Essentially, every time there is a block fault, the algorithm increments the entries of M corresponding to the pairs composed of a block in the working set of blocks and the block causing the fault.

Now let S(t,m) represent the LRU stack of blocks after time t with stack size m. (The stack size equals the number of page frames which will be allocated to the program during its execution after restructuring.) Let s_i(t,m) represent the contents of stack position i in S(t,m). The CLRU algorithm operates as follows.

1. $k = 0$.
2. $k = k + 1$.
3. If $r_k \epsilon$ S($k-1$,m), then /* there is no block fault */ go to 2.
4. For i = 1 to m do /* increment M by 1 */
   $M_{s_i(k-1,m),B(r_k)} = M_{s_i(k-1,m),B(r_k)} + 1$.
5. Go to 2.

In other words, every time there is a block fault, the algorithm increments the entries of the CLRU matrix M connecting the blocks in the block LRU stack with the block causing the fault.

The effectiveness of the CWS and CLRU algorithms has been shown in a number of studies [FERD74a, FERD75, FERD76a]. Their ability to reduce the number of page faults is a result of the reduction in the number of potential page faults. Each block fault or critical block reference is a potential page fault. A potential page fault become real if the block which is referenced is not linked to another block already in memory. The intent of critical set restructuring is to reduce the frequency of these events.

3.1 An IRM Analysis of the CWS Algorithm

The Independent Reference Model (IRM) is a simple probabilistic model of program behavior. It assumes that each block B_i (henceforth to be denoted by i to simplify the notation) is referenced with a probability b_i which is constant and independent of the previously referenced blocks. Obviously, $\sum_{i=1}^{N} b_i = 1$, where N is the number of blocks in the program to be restructured.

Let us first compute the equilibrium page fault rate as a function of the block reference probabilities. We assume that there are exactly two blocks per page. This assumption is made because the restructuring graph only connects pairs of blocks. The contribution of larger clusters to the reduction of the page fault rate would require more information than that contained in the graph. The equilibrium page fault rate under the working set policy has been shown by Denning and Schwartz [DENP72] to be $F(WS) = \sum_{i=1}^{N} p_i * (1-p_i)^T$, where p_i is the probability of referencing page i and N is the number of pages in the program.

Let $b_{(i,1)}$ and $b_{(i,2)}$ denote the block reference probabilities of blocks 1 and 2 of page i. Clearly, the equilibrium page fault rate is given by

$$F(WS) = \sum_{k=1}^{N} [(b_{(k,1)} + b_{(k,2)}) * (1 - b_{(k,1)} - b_{(k,2)})^T]. \tag{1}$$

Under IRM and with two blocks per page, the probability that block i causes a block fault and block j is in the block working set is given by:

$$BF_{ij}(WS) = b_i * ((1 - b_i)^T - (1 - b_i - b_j)^T), \tag{2}$$

since Prob(r_t = i) = b_i and

$$Prob(i \notin WS, j \in WS) = Prob(i \notin WS) - Prob(i \notin WS, j \notin WS)$$
$$= (1 - b_i)^T - (1 - b_i - b_j)^T.$$

BF_{ij}(WS) is proportional to the value of the (i,j) entry in the CWS matrix. The weights of edge (i,j) in the CWS graph will be given by

$$EW_{ij}(WS) = BF_{ij}(WS) + BF_{ji}(WS)$$
$$= b_i * ((1 - b_i)^T - (1 - b_i - b_j)^T) + b_j * ((1 - b_j)^T - (1 - b_j - b_i)^T). \tag{3}$$

Let (x,y) denote the y^{th} block of page x. In the clustering phase of the procedure, we find a mapping of all pairs of blocks onto a set of pages. That mapping, which places blocks i and j into page k so that the two blocks can be denoted by (k,1) and (k,2), is one of those which maximize the following expression:

$$\sum_{k=1}^{N} [(b_{(k,1)} * (1 - b_{(k,1)})^T + b_{(k,2)} * (1 - b_{(k,2)})^T$$
$$- (b_{(k,1)} + b_{(k,2)}) * (1 - b_{(k,1)} - b_{(k,2)})^T]. \tag{4}$$

Note that for a single pair of blocks expression (4) is equal to the right hand side of (3).

Theorem 1: Under IRM, maximizing (4) is equivalent to minimizing the equilibrium page fault

rate F(WS).

Proof: Maximizing expression (4) is equivalent to minimizing the following expression:

$$\sum_{k=1}^{N} [(b_{(k,1)} + b_{(k,2)})^T * (1 - b_{(k,1)} - b_{(k,2)})^T]. \tag{5}$$

since the value of the term

$$\sum_{k=1}^{N} [b_{(k,1)} * (1 - b_{(k,1)})^T + b_{(k,2)} * (1 - b_{(k,2)})^T]$$

is independent of the mapping chosen. Expression (5) coincides with the right-hand side of (1), i.e., with F(WS). Q.E.D.

Thus, if an optimum clustering algorithm were used and pages were not allowed to contain more than two blocks, the CWS restructuring algorithm would be optimum. This conclusion is valid also under a more general Markov model of program behavior [LAUE79] and can probably be extended to any kind of behavior (an informal proof of the latter statement was given in [FERD74a]).

3.2 An IRM Analysis of the CLRU Algorithm

Coffman and Denning [COFE73] and King [KINW71] have presented closed form expressions for the total page fault rate generated by local (fixed partitioning) LRU in statistical equilibrium when the program obeys the IRM. Unfortunately, those expressions do not lend themselves easily to an analysis of the effects of restructuring.

Let us consider the case of two blocks per page. Let π_s denote the equilibrium probability of stack state s, $s = [j_1, j_2, \cdots, j_m]$, where m is the number of elements in the *block stack*. We similarly define $\pi_{s'}$, which however, refers to the *page stack* of size m. Clearly, given a page stack state $\hat{s}'$, a necessary condition for having a fault to a page k is that neither block (k,1) nor (k,2) be in the block stack $\hat{s}$ corresponding to $\hat{s}'$. This, however, is not a sufficient condition since the block stack may contain both blocks belonging to a page and hence we may have pages in the page stack neither of whose blocks appear in the block stack. Reference to these pages at that time would not cause a page fault. Hence, we have

Theorem 2: An upper bound for the LRU equilibrium page fault rate under IRM is given by:

$$AF(LRU) = \sum_{k=1}^{N} [([b_{(k,1)} + b_{(k,2)}] * \sum_{\substack{(k,1) \notin s \\ (k,2) \notin s}} \pi_s)]. \tag{6}$$

Proof: The upper bound on the probability of a fault to page k is given by

$$[b_{(k,1)} + b_{(k,2)}] * \sum_{\substack{(k,1) \notin s \\ (k,2) \notin s}} \pi_s.$$

Summing over all pages in the program, we have (6). Q.E.D.

Assuming two blocks per page and an optimal clustering, let us define the CLRU restructuring graph in terms of IRM probabilities. When there is a reference to a block not in the block stack, the weights of all edges connecting the requested block with those in the block stack are to be incremented by one. Under IRM, this event occurs for each pair of blocks (i,j) with the probability $BF_{ij}(LRU) = b_i * \sum_{\substack{i \notin s \\ j \in s}} \pi_s$. Note that BF_{ij}(LRU) is proportional to the element (i,j) of the restructuring graph. Thus, the weight of the edge (i,j) is proportional to

$$EW_{ij}(LRU) = BF_{ij}(LRU) + BF_{ji}(LRU)$$

$$= b_i * \sum_{\substack{i \notin s \\ j \in s}} \pi_s + b_j * \sum_{\substack{j \notin s \\ i \in s}} \pi_s. \tag{7}$$

We observe that

$$\sum_{i,j \notin s} \pi_s = 1 - \sum_{\substack{i \in s \\ j \notin s}} \pi_s - \sum_{\substack{i \notin s \\ j \in s}} \pi_s - \sum_{i,j \in s} \pi_s. \tag{8}$$

Substituting (8) into (6), we obtain

$$AF(LRU) = \sum_{k=1}^{N} [b_{(k,1)} + b_{(k,2)}] * [1 - \sum_{\substack{(k,1) \in s \\ (k,2) \notin s}} \pi_s - \sum_{\substack{(k,1) \notin s \\ (k,2) \in s}} \pi_s - \sum_{\substack{(k,1) \in s \\ (k,2) \in s}} \pi_s]. \tag{9}$$

Finally, we have the following theorem.

Theorem 3: CLRU minimizes AF(LRU).
Proof: Expanding expression (9), we obtain

$$\sum_{k=1}^{m} (b_{(k,1)} + b_{(k,2)}) * \sum_{\substack{(k,1) \notin s \\ (k,2) \notin s}} \pi_s$$

$$= \sum_{k=1}^{N} [b_{(k,1)} + b_{(k,2)} - b_{(k,1)} * \sum_{\substack{(k,1) \in s \\ (k,2) \notin s}} \pi_s - b_{(k,2)} * \sum_{\substack{(k,1) \in s \\ (k,2) \notin s}} \pi_s -$$

$$b_{(k,1)} * \sum_{\substack{(k,1) \notin s \\ (k,2) \in s}} \pi_s - b_{(k,2)} * \sum_{\substack{(k,1) \notin s \\ (k,2) \in s}} \pi_s - (b_{(k,1)} + b_{(k,2)}) * \sum_{\substack{(k,1) \in s \\ (k,2) \in s}} \pi_s]. \tag{10}$$

The following expressions are constant (i.e., independent of the block layout):

$$\sum_{k=1}^{N} [b_{(k,1)} * (\sum_{\substack{(k,1) \in s \\ (k,2) \notin s}} \pi_s + \sum_{\substack{(k,1) \in s \\ (k,2) \in s}} \pi_s)] = \sum_{k=1}^{N} [b_{(k,1)} * \sum_{(k,1) \in s} \pi_s],$$

$$\sum_{k=1}^{N} [b_{(k,2)} * (\sum_{\substack{(k,1) \notin s \\ (k,2) \in s}} \pi_s + \sum_{\substack{(k,1) \in s \\ (k,2) \in s}} \pi_s)] = \sum_{k=1}^{N} [b_{(k,2)} * \sum_{(k,2) \in s} \pi_s],$$

$$\sum_{k=1}^{N} [b_{(k,1)} * (\sum_{\substack{(k,1) \notin s \\ (k,2) \in s}} \pi_s + \sum_{\substack{(k,1) \notin s \\ (k,2) \notin s}} \pi_s)] = \sum_{k=1}^{N} [b_{(k,1)} * \sum_{(k,1) \notin s} \pi_s],$$

$$\sum_{k=1}^{N} [b_{(k,2)} * (\sum_{\substack{(k,1) \in s \\ (k,2) \notin s}} \pi_s + \sum_{\substack{(k,1) \notin s \\ (k,2) \notin s}} \pi_s)] = \sum_{k=1}^{N} [b_{(k,2)} * \sum_{(k,2) \notin s} \pi_s].$$

By manipulating expression (10) using the constants above, we obtain the following expression:

$$\sum_{k=1}^{m} [b_{(k,1)} * \sum_{\substack{(k,1) \notin s \\ (k,2) \in s}} \pi_s + b_{(k,2)} * \sum_{\substack{(k,1) \in s \\ (k,2) \notin s}} \pi_s] = \sum_{k=1}^{N} [(b_{(k,1)} + b_{(k,2)}) - b_{(k,1)} * \sum_{(k,1) \in s} \pi_s -$$

$$b_{(k,2)} * \sum_{(k,2) \in s} \pi_s - (b_{(k,1)} + b_{(k,2)}) * \sum_{\substack{(k,1) \notin s \\ (k,2) \notin s}} \pi_s]. \tag{11}$$

By (7), finding a block order which maximizes (11) is the goal of the clustering procedure for CLRU. We see that maximizing (11) minimizes expression (9), i.e., AF(LRU). Q.E.D.

4. Extending Strategy Oriented Restructuring

Figure 1 shows a three-level memory hierarchy. The following assumptions are made for this hierarchy.

1. The inclusion property [MATR70] does not necessarily hold: a page which is at level I may not be at level II for certain combinations of memory management policies. However, we will enforce inclusion by causing that page to be loaded into the level II memory

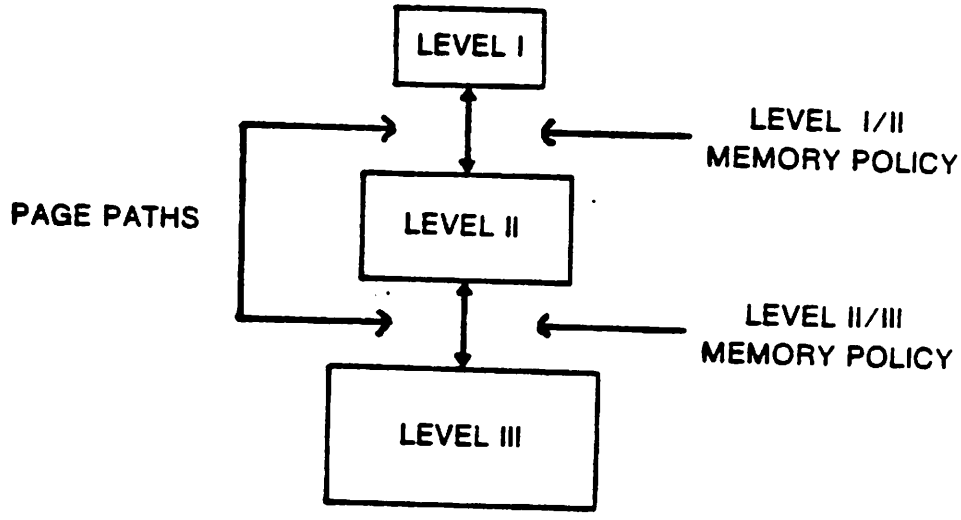


Figure 1. Three Level Memory Hierarchy Model

whenever it is referenced at level I.

2. Paging is performed on demand as stated in Assumption 1: a reference at level I (which did not cause a fault at level I) may cause a fault at level II, if the data item requested is not in level II.

3. The amount of information moved between any two adjacent levels of memory in a transfer is one page.

In any restructuring algorithm, the block-to-page mapping must be the same at all levels. Hence, the restructuring algorithm must create a single restructuring graph to be used at both levels I and II.

One scheme for extending program restructuring to three levels consists of applying the two-level techniques between each of the levels of memory using a common restructuring graph. With this scheme, difficulties may arise from conflicts due to larger contributions to edge costs from one of the levels of memory. If, for example, critical set restructuring algorithms are to be applied between the levels of memory, the number of block faults to level I could be so large that the level-II block faults would not substantially influence the weights in the restructuring graph and the combined scheme would not necessarily reduce the number of level-II faults, whose cost may be much higher than that of level-I faults.

Other performance indices may also be affected. For example, the space-time product (STP) of a program is a function of the fault rate, of the memory occupancy, and of the page transfer time. Certain types of restructuring algorithms have been known to increase the mean memory occupancy. If using a combined scheme leaves the number of expensive faults unchanged and increases the mean memory occupancy, then the most important part of the STP will also be adversely affected.

The rest of this section and the next section will explore the effectiveness of applying some two-level restructuring techniques between each level of a three-level memory hierarchy.

4.1 The CWS-CWS Algorithm

In the three level memory hierarchy illustrated in Figure 1, we assume that each pair of adjacent levels is managed by the working set policy. This means that there are two windows with sizes T' and T'', T' being the window size at level I and T'' the window size at level II. If we assume that $T' < T''$, then the inclusion property holds, i.e., every page in level I is also in level II. When both levels I and II of memory are managed by the working set policy, we may

use the CWS-CWS algorithm to resturcture programs.

If r_i is the i^{th} reference issued by the program to be restructured, $B(r_i)$ the block referenced at time i, M the restructuring matrix, and W(t,T) the working set of blocks at time t with window size T, the CWS-CWS algorithm may be described as follows.

1. $k = 0$.
2. $k = k + 1$.
3. If $r_k \in W(k-1,T')$, then /* r_k is in the level-I block working set*/ go to 2.
4. For all $B_l \in W(k-1,T')$ do /* increment the matrix by 1*/
   $M_{B_l,B(r_k)} = M_{B_l,B(r_k)} + 1$.
5. If $r_k \in W(k-1,T'')$, then /* r_k is in the level-II working set */ go to 2.
6. For all $B_l \in W(k-1,T'')$ do /* increment the matrix by 1 */
   $M_{B_l,B(r_k)} = M_{B_l,B(r_k)} + 1$.
7. Go to 2.

Thus, each time there is a block fault to levels II or III, the algorithm increments the weights of the edges connecting the blocks in the working sets with the block causing the fault. We now assume that there are exactly two blocks per page, and that the reference string of the program obeys the IRM. Then, the equilibrium page fault rate for a three-level memory system with a working set policy between each level is given by:

$$F(WS-WS) = \sum_{k=1}^{N} (p_k * [(1-p_k)^T + (1-p_k)^{T'}]),$$

where p_i represents the probability of referencing page i and N is the number of pages in the program. $F(WS-WS)$ is the sum of the page fault rates to all levels. Note that the definition of $F(WS-WS)$ assumes that all faults have the same weight.

Let $b_{(k,1)}$ and $b_{(k,2)}$ be defined as above. Thus, the equilibrium page fault rate in terms of block reference probabilities is:

$$F(WS-WS) = \sum_{k=1}^{N} ([b_{(k,1)} + b_{(k,2)}] * [(1 - b_{(k,1)} - b_{(k,2)})^T + (1 - b_{(k,1)} - b_{(k,2)})^{T'}]). \quad (12)$$

The optimality result of Theorem 1 can be extended to the CWS-CWS algorithm. Assuming two blocks per page and an optimum clustering, each element BF_{ij} of the CWS-CWS restructuring matrix is proportional to:

$$BF_{ij}(WS-WS) = b_i * [(1-b_i)^T - (1-b_i-b_j)^T + (1-b_i)^{T'} - (1-b_i-b_j)^{T'}], \quad (13)$$

where b_i and b_j are the reference probabilities of blocks i and j.

The non-directed restructuring graph has weights proportional to:

$$EW_{ij}(WS-WS) = BF_{ij}(WS-WS) + BF_{ji}(WS-WS)$$
$$= b_i * [(1-b_i)^T + (1-b_i)^{T'}] + b_j * [(1-b_j)^T + (1-b_j)^{T'}] - (b_i + b_j) * [(1-b_i-b_j)^T + (1-b_i-b_j)^{T'}]. \quad (14)$$

In the clustering phase of the procedure, we find a mapping of all pairs of blocks onto the set of pages which maximizes the following expression:

$$\sum_{k=1}^{N} (b_{(k,1)} * [(1-b_{(k,1)})^T + (1-b_{(k,1)})^{T'}] + b_{(k,2)} * [(1-b_{(k,2)})^T + (1-b_{(k,2)})^{T'}] - b_{(k,1)} * [(1-b_{(k,1)}-b_{(k,2)})^T + (1-b_{(k,1)}-b_{(k,2)})^{T'}] + b_{(k,2)} * [(1-b_{(k,1)}-b_{(k,2)})^T + (1-b_{(k,1)}-b_{(k,2)})^{T'}]). \quad (15)$$

Thus we have

Theorem 4: Under IRM, the CWS-CWS algorithm minimizes the combined equilibrium page fault rate F($WS-WS$).

Proof: Maximizing expression (15) is equivalent to minimizing the following expression

$$\sum_{k=1}^{N} ([b_{(k,1)} + b_{(k,2)}] * [(1 - b_{(k,1)} - b_{(k,2)})^T + (1 - b_{(k,1)} - b_{(k,2)})^{T'}]), \tag{16}$$

since the term

$$\sum_{k=1}^{N} (b_{(k,1)} * [(1 - b_{(k,1)})^T + (1 - b_{(k,1)})^{T'}] + b_{(k,2)} * [(1 - b_{(k,2)})^T + (1 - b_{k,2})^{T'}])$$

is constant, i.e., independent of the mapping. Expression (16) coincides with the right hand side of (12), that is, with F($WS-WS$). Q.E.D.

4.2 The CWS-CLRU Algorithm

A three level memory hierarchy will now be considered in which level I is managed by the working set policy and level II by the LRU policy. The appropriate restructuring algorithm to be used in this case is the CWS-CLRU algorithm.

If r_l, $b(r_l)$, W(t,T), and M represent the same entities as in section 3.1, S(t,M) is the block LRU stack after time t with stack size M and s_l(t,M) represents the contents of position i in stack S(t,M), the CWS-CLRU algorithm constructs the restructuring graph by the following procedure.

1. $k = 0$.
2. $k = k + 1$.
3. If $r_k \in W(k-1,T)$, then /* r_k is in the level-I working set */ go to 5.
4. For all $B_i \in W(k-1,T)$ do /* increment the matrix by 1*/
   $M_{B_i,B(r_k)} = M_{B_i,B(r_k)} + 1$.
5. If $r_k \in$ S(k-1,M), then /* not an LRU block fault */ go to 2.
6. For i = 1 to N do /* increment the matrix by 1 */
   $M_{s_i(k-1,M),B(r_k)} = M_{s_i(k-1,M),B(r_k)} + 1$.
7. Go to 2.

Each time there is a block fault at levels I or II, we increment by 1 the weights of the edges which connect the blocks in the working set or in the LRU stack with the block causing the fault. For this combination of memory management policies, the inclusion property is not guaranteed to hold. In other words, a block which is at level I may not be in level II. Hence, the destination of the conditional branch in step 3 is to step 5 instead of to step 2.

We now assume that the program to be restructured behaves according to the IRM, that there are exactly two blocks per page, and that an optimal clustering can be obtained. The (i,j) element in the CWS-CLRU restructuring matrix is proportional to

$$BF_{ij} = b_i * [\sum_{\substack{i \notin s \\ j \in s}} \pi_s + (1 - b_i)^T - (1 - b_i - b_j)^T], \tag{17}$$

where π_s is the equilibrium probability of block stack s, and T is the working set window size. (The page stack has the same number of positions as the block stack).

Thus, the weight of edge (i,j) in the restructuring graph is proportional to

$$\begin{aligned} EW_{ij}(WS-LRU) &= BF_{ij}(WS-LRU) + BF_{ji}(WS-LRU) \\ &= b_i * \sum_{\substack{i \notin s \\ j \in s}} \pi_s + b_j * \sum_{\substack{i \in s \\ j \notin s}} \pi_s + b_i * (1 - b_i)^T + b_j * (1 - b_j)^T \\ &\quad - (b_i + b_j) * (1 - b_i - b_j)^T. \end{aligned} \tag{18}$$

In the clustering phase of the procedure, we find a mapping of all pairs of blocks onto the set of pages which maximizes

$$\sum_{k=1}^{N} [b_{(k,1)} * \sum_{\substack{(k,1)\notin s \\ (k,2)\in s}} \pi_s + b_{(k,2)} * \sum_{\substack{(k,1)\in s \\ (k,2)\notin s}} \pi_s + b_{(k,1)} * (1 - b_{(k,1)})^T + b_{(k,2)} * (1 - b_{(k,2)})^T - (b_{(k,1)} + b_{(k,2)}) * (1 - b_{(k,1)} - b_{(k,2)})^T]. \quad (19)$$

Expression (19) is a sum of terms each of which is equal to the right-hand side of expression (18), and which are chosen so as to maximize this sum. The effect of maximizing (19) is given in the following theorem.

Theorem 5: Under IRM, the CWS-CLRU restructuring algorithm minimizes an upper bound for the combined equilibrium page fault rate.

Proof: An upper bound for the combined equilibrium page fault rate F(WS-LRU) is given by the following expression:

$$\sum_{k=1}^{N} [(b_{(k,1)} + b_{(k,2)}) * \sum_{\substack{(k,1)\notin s \\ (k,2)\notin s}} \pi_s + (b_{(k,1)} + b_{(k,2)}) * (1 - b_{(k,1)} - b_{(k,2)})^T]. \quad (20)$$

Expression (20) is equal to the sum of expression (5), the working set page fault rate, and expression (9), an upper bound on the LRU equilibrium fault rate in a two-level hierarchy. By Theorems 1 and 3, we have that maximizing expression (19) minimizes expression (20). Q.E.D.

How distant is the upper bound from the actual fault rate? We are confined to computing the upper bound since under LRU there is no direct relationship between the equilibrium block fault rate and the equilibrium page fault rate.

Note that, in expression (8), the probability that blocks i and j are in the block stack s at the same time is given by the term $\sum_{i,j\in s} \pi_s$. If blocks i and j are in the same page, the corresponding page stack, s', will contain pages whose blocks are not in s. (If the number of positions in the block stack were greater than that of the page stack, we could not be sure that all blocks in the block stack would be represented in the page stack).

The size of the difference between the upper bound and the actual fault rate is approximatly proportional for each pair of blocks (k,1) and (k,2) to

$$- \sum_{\substack{(k,1)\in s^* \\ (k,2)\in s^*}} (\pi_{s^*} * (1 - f(\sum_{j=1}^{N} (\sum_{\substack{(j,1)\in s \\ (j,2)\in s \\ s\neq s^* \\ s\neq s^*}} \pi_s) * N, s^*)).$$

where

$$f(x,s) = \begin{cases} \sum_{k=1}^{N} (1 - \sum_{\substack{(k,1)\notin s \\ (k,2)\notin s \\ (k,1)\notin \hat{s} \\ (k,2)\in \hat{s}}} \pi_{\hat{s}} - \sum_{\substack{(k,1)\notin s \\ (k,2)\in s \\ (k,1)\in \hat{s} \\ (k,2)\notin \hat{s}}} \pi_{\hat{s}} - \sum_{\substack{(k,1)\in s \\ (k,2)\notin s \\ (k,1)\notin \hat{s} \\ (k,2)\in \hat{s}}} (\pi_{\hat{s}} * f(\sum_{k=1}^{N} (\sum_{\substack{(k,1)\notin s \\ (k,2)\notin s \\ (k,1)\in \hat{s} \\ (k,2)\in \hat{s}}} \pi_{\hat{s}}) * N, \hat{s})) & \text{if } x \neq 0 \\ 0 & \text{if } x = 0 \end{cases}$$

$\hat{s}$ is obtained from stack s by appending $\lfloor x \rfloor$ stack positions to the end and s^* is a stack of the same size as $\hat{s}$.

The expression representing the difference, though quite complex, is a sum of terms of degree greater than 2. If the term $\sum_{\substack{(k,1)\epsilon s \\ (k,2)\epsilon s}} \pi_s$ is small, then the difference will be small, since function (1-f) has positive values less than 1.

This discussion leads us to the conclusion that, for LRU, in addition to finding an optimal clustering (which is generally not unique), the restructuring procedure should also minimize the value of $\sum_{\substack{(k,1)\epsilon s \\ (k,2)\epsilon s}} \pi_s$.

5. Experimental Results

5.1 Experiments with the CWS-CWS Algorithm

Trace-driven simulation was used for an experimental study of the CWS-CWS algorithm's performance. A simulator was implemented to model a three-level memory hierarchy in which sampled working set was the memory management policy between each level.

The sampled working set policy is defined by two parameters, namely T, the size of the window, and S, the sampling interval. At each time instant which is an integer multiple of S, the execution of the program is interrupted. All pages not referenced within [nS-T+1,nS] are excluded from the working set. The size of the working set can only decrease at those fixed intervals. For the three level case, two sampling intervals and two window sizes for the respective levels are to be specified. The level II window was defined as an integer multiple of the level I window.

Three address traces (APL, FFT, and WATFIV) were used in the experiments. These traces were previously analyzed by Smith in [SMIA76,SMIA77,SMIA78b]. For each trace, a series of 12 simulations was performed. In the experiments, the level I window size T' were set equal to 10 and 20 reference sets (there are 1173 references/reference set) and the level II window size T'' to 40, 60, and 80 reference sets. For each value of the window size, non-restructured and restructured layouts of the various programs in virtual space have been considered. The clustering algorithm placed exactly two blocks in each page.

At each sampling point t, the space-time product STP_i for level i was updated as follows:

$$STP_i = STP_i + MMO_i(t) * (1 + npf_1(t) * TT' + npf_2(t) * TT'')$$

where $MMO_i(t)$ is the level i mean memory occupancy for the interval [t-s+1,t], $npf_i(t)$ is the number of page faults at level i for the interval [t-s+1,t], and TT' and TT'' are the page transfer times for levels I and II respectively. In the simulation runs, TT' was set equal to 10 ms and TT'' to 100 ms.

The results of the simulations are presented in Tables I, II, and III. Figures 2 through 7 show the level I and level II STP's as a function of the level II window size for the three programs.

In all cases, the CWS-CWS algorithm reduced the page fault rate at both levels of the memory hierarchy. The largest percent reduction occurred for the APL trace at level I for $T' = 20$ and $T'' = 80$. A 51 percent reduction in the page fault rate was achieved. The least amount of improvement (only 4 percent) occurred for the level I page fault rate of the FFT trace with $T' = 10$ and $T'' = 80$.

The mean memory occupancy increased in many cases. This phenomenon may be explained by referring to the notion of spatial locality. For programs which exhibit little spatial locality, a critical set algorithm can decrease the mean memory occupancy by reorganizing the layout of the program so that blocks are placed in pages which have a strong likelihood of being referenced following each other. However, if a program is spatially local, a reduction in the page fault rate, which is the objective of critical set restructuring, may be achieved by decreasing spatial locality, thereby increasing mean memory occupancy.

The STP was also decreased in this series of experiments. The largest decrease was 34 percent for the APL trace in level II memory with level I window size equal to 20 and level II window size equal to 40. As shown by Figures 2 through 7, the STP's for the restructured programs are consistently lower than or equal to the STP for the non-restructured programs.

Table III shows that, for the restructured WATFIV program, the page fault rates with $T'' = 60$ were lower than the page fault rates with $T'' = 80$. This is due to the fact that, for that particular program, the page fault rates had almost reached, because of restructuring, their minimal values and that a non-optimal clustering algorithm was used. (Note that the total number of pages in the WATFIV trace was 41 and the number of level II faults for $T'' = 60$ is 44).

5.2 Experiments with the CWS-CLRU Algorithm

A series of experiments, also based on a trace driven simulator and the three address traces APL, FFT, and WATFIV, were performed to test the effectiveness of the CWS-CLRU restructuring algorithm. The page sizes assumed for the three programs were 2048, 1024, and 2048 bytes respectively. The simulator represented a three level memory system in which level I was managed by a sampled working set strategy and level II by LRU. When a fault to the level I memory occurred, a fetch from level II was attempted. If the page resided in the level II memory, the page was brought into the level I working set. If the page did not reside in level II, this caused a level II fault which resulted in a fetch from the level III memory.

To insure that the inclusion property would still hold, in spite of its not being automatically enforced by the memory policy, a level II fault was generated when a page not present in level II memory was referenced even if the page was found at level I. Thus, a page fault at level II was not necessarily a result of a fault at level I. Note, however, that this event may occur only when the level I working set is larger than the level II LRU stack.

For each string, 12 simulations were performed. They exhaustively encompassed the combinations of the values of T (10 and 20 reference sets), of those of the level II stack size (25, 30 and 40 pages), and of the two layouts (non-restructured and restructured) of the three programs. Page transfer times were chosen to be 10 and 100 milliseconds for level I and II respectively.

The main results of the simulation experiments are presented in Tables IV, V, and VI, and some of them are plotted in Figures 8 through 13. In most cases, the effect of restructuring on the page fault rate was to reduce it. However, in some cases, and in particular in the level II faults for the FFT trace, the opposite effect was obtained. This can be attributed to the behavior of the FFT program, which is characterized by cyclic and sequential referencing, and to the fact that the non-restructured FFT program may be operating very close to its optimal performance in the WS-LRU environment. For traces other than FFT, an increase in the fault rate only occurred in level II memory with stack size 40. These small increases could be due to the near minimal number of page faults and to the non-optimality of CLRU and the clustering algorithms. The exceptionally large level II page fault rate of WATFIV with a stack size of 25 pages was due to the fact that the mean memory occupancy for the level I memory was larger than the amount allocated for the level II fixed partition. Hence, many useful pages which were relegated to level III because of insufficient room in level II were subsequently referenced at level I and were then forced back into level II.

As in the CWS-CWS experiments, the level I mean memory occupancy increased in a number of cases. However, a substantial number of restructured layouts resulted in a mean memory occupancy reduction.

For the APL and WATFIV traces the space time product was generally reduced or increased very slightly. The STP of the FFT trace increased in all cases. This was primarily a result of the small reduction in the number of page faults and of the substantial increases in the mean memory occupancy.

The effect of restructuring on the combination of page fault rates and mean memory occupancy was excellent for the APL trace. In most cases, the mean memory occupancy increased very slightly while the number of page faults was dramatically reduced over the non-restructured cases (see Figure 9). As a result, the level I STP was reduced, as shown in Figure 8.

The effectiveness of restructuring was also evident for the WATFIV trace. A significant reduction in page fault rate and a small decrease in the mean memory occupancy was obtained, except when the stack size was 40 pages, in which case the reduction only occurred in the page fault rate. The effect on the STP as reflected in Figure 12 was favorable. The restructured program consistently outperformed the non-restructured one. Figure 13 shows the effectiveness of restructuring at level II. For a given level II memory size, the page fault rate was consistently lower in the restructured case, except at stack size 40. Again, this may be the result of a near-optimum performance in the non-restructured case and of the perturbations introduced for this level II size by the non-optimality of the restructuring and clustering algorithms.

6. Concluding Remarks

Program restructuring has been studied in the context of memory hierarchies with three levels. In particular, the WS-WS case and the WS-LRU case were considered. Three-level extensions of the well known two-level strategy-oriented critical set restructuring algorithms for these two cases were studied theoretically as well as experimentally.

In general, the effectiveness of critical-set program restructuring in reducing the page fault rate has been demonstrated for a three-level hierarchy. This was especially true for the APL and WATFIV traces used in the simulation experiments. For the FFT trace, the results were less satisfactory. since the page fault rate for LRU management increased and for working set management decreased only slightly. This behavior may be attributed to the peculiar (cyclic and sequential) referencing characteristics of the program.

The analytic results of our investigation confirm the effectiveness of the CWS-CWS restructuring algorithm. Aside from the FFT trace, the CWS-CLRU algorithm also performed well in spite of the weaker analytic results. This weakness of the CWS-CLRU algorithm may be an additional factor responsible for the poorer results obtained for the FFT trace in the CWS-CLRU experiments.

As in a two-level hierarchy, mean memory occupancy was generally increased by restructuring. However, the increases in many cases were less than those due to increasing the level I window size from 10 to 20 reference sets.

Finally, the space-time product was generally reduced. In the experiments, the transfer time from level III to level II was assumed to be 10 times that of a transfer between level II and level I. This leads to a very high cost for a level II fault when compared with the cost of memory. Some experiments resulted in a minor decrease in level II page fault rate corresponding to an increase in mean memory occupancy. In most cases, FFT being the exception, the STP was reduced anyway.

The effectiveness of program restructuring has been clearly demonstrated for programs with behaviors similar to the APL and WATFIV traces. For programs which behave like the FFT trace, restructuring cannot be expected to be very effective (even though, for the FFT trace in the working set environment, the page fault rate and the STP were reduced). The improvement of performance in cases of sequential referencing behavior has been explored by a number of investigators [SMIA78a,SMIA78b,TRIK76]. Their approach, based on prefetching policies, seems to be more fruitful than restructuring for FFT-like programs.

References

[COFE73] Coffman, E. G. and P. J. Denning, *Operating Systems Theory*, Prentice Hall, Englewood Cliffs, 1973.

[DENP72] Denning, P. J. and S. C. Schwartz, "Properties of the Working-Set Model," *Comm. ACM* 15, 3 (Mar. 1972) 191-198.

[EVES75] Even, S. and O. Kariv, "An $O(n^{2.5})$ Algorithm for Maximum Matching in General Graphs," *Proc. 16th Symp. on Found. of Comp. Sci.* (1975) 100-112.

[FERD74a] Ferrari, D., "Improving Locality by Critical Working Sets," *Comm. ACM* 17 (Nov. 1974) 614-620.

[FERD74b] Ferrari, D., "Critical-Set Algorithms for Program Locality Improvement," *Proc. 12th Atherton Conf. on Circuit and Systems Theory*, Montecello, ILL (Oct. 1974) 641-648.

[FERD75] Ferrari, D., "Tailoring Programs to Models of Program Behavior," *IBM J. Res. Develop.* 19, 3 (May 1975) 244-251.

[FERD76a] Ferrari, D. and E. Lau, "An Experiment in Program Restructuring for Performance Enhancement," *Proc. 2nd Int. Conf. on Software Engineering*, San Francisco (Oct 1976) 146-150.

[FERD76b] Ferrari, D., "The Improvement of Program Behavior," *Computer* 9, 11 (Nov. 1976) 39-47.

[FERD77] Ferrari, D. and M. Kobayashi, "Program Restructuring Algorithms for Global LRU Environments," *Proc. Int. Computing Symp. 1977*, Liege, Belgium (Apr. 1977) 277-283.

[GECJ74] Gecsei, J., "Determining Hit Ratios for Multilevel Hierarchies," *IBM J. Res Develop.* 18, 4 (July 1974) 316-327.

[GOLR74] Goldberg, R. P. and R. Hassinger, "The Double Paging Anomaly," *AFIPS Conf. Proc.* (NCC 1974) 195-199.

[KINW71] King, W. F., III, "Analysis of Demand Paging Algorithms," *Proc. IFIP Congress 71*, Ljubljana, Yugoslavia (Aug. 1971) TA-3-155 to TA-3-159.

[LAUE79] Lau, E. J., *Performance Improvement of Virtual Memory Systems by Restructuring and Prefetching*. Ph.D Dissertation, U. C. Berkeley, 1979.

[MATR70] Mattson, R. L., J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation Techniques for Storage Hierarchies," *IBM Sys. J.* 9, 2 (1970) 78-117.

[SMIA76] Smith, A. J., "A Modified Working Set Paging Algorithm," *IEEE-TC* C-25, 9 (Sept. 1976) 907-914.

[SMIA77] Smith, A. J., "Two Simple Methods for the Efficient Analysis of Memory Address Trace Data," *IEEE-TSE* SE3, 1 (Jan. 1977) 94-101.

[SMIA78a] Smith, A. J., "Sequentiality and Prefetching in Data Base Systems," *ACM Trans. DBS* 3, 3 (Sept. 1978) 223-247.

[SMIA78b] Smith, A. J., "Sequential Program Prefetching in Memory Hierarchies," *Computer* 11, 12 (Dec. 1978) 7-21.

[TRIK76] Trivedi, K. S., "Prepaging and Applications in Array Algorithms," *IEEE-TC* C25, 9 (Sept 1976) 915-921.

[TUEW76] Tuel, W. G., Jr., "An Analysis of Buffer Paging in Virtual Storage Systems," *IBM J. Res. Develop.* 20, 5 (Sept 1976) 518-520.

	Non-Restructured			Restructured					
	2ND LVL WIN=40	2ND LVL WIN=60	2ND LVL WIN=80	2ND LVL WIN=40		2ND LVL WIN=60		2ND LVL WIN=80	
(1ST LVL WIN 10)									
1ST LVL FAULT RATE	.3057	.3057	.3057	.1727		.1727		.1630	
2ND LVL FAULT RATE	.1214	.0694	.0616	.0813		.0662		.0603	
1ST LVL MMO (Δ%)	22.56	22.56	22.56	25.77	(14)	25.82	(14)	25.57	(13)
2ND LVL MMO (Δ%)	29.39	32.70	34.18	29.14	(-1)	31.21	(-5)	32.20	(-6)
1ST LVL STP X 10^3 (Δ%)	662	393	358	498	(-25)	382	(13)	344	(-4)
2ND LVL STP X 10^3 (Δ%)	1 008	705	697	736	(-27)	660	(-6)	631	(-9)
(1ST LVL WIN 20)									
1ST LVL FAULT RATE	.2119	.2119	.2119	.1119		.1075		.1031	
2ND LVL FAULT RATE	.1214	.0694	.0616	.0696		.0550		.0503	
1ST LVL STP MMO (Δ%)	25.50	25.50	25.50	31.04	(22)	30.83	(21)	30.69	(20)
2ND LVL MMO (Δ%)	29.39	32.70	34.18	32.95	(12)	34.32	(5)	35.07	(3)
1ST LVL STP X 10^3 (Δ%)	776	438	392	531	(-32)	384	(-12)	348	(-11)
2ND LVL STP X 10^3 (Δ%)	952	644	634	631	(-34)	545	(-15)	548	(-14)

TABLE I. APL SIMULATION RESULTS

	Non-Restructured			Restructured					
	2ND LVL WIN=40	2ND LVL WIN=60	2ND LVL WIN=80	2ND LVL WIN=40		2ND LVL WIN=60		2ND LVL WIN=80	
(1ST LVL WIN 10)									
1ST LVL FAULT RATE	.1275	.1275	.1275	.1203		.1191		.1223	
2ND LVL FAULT RATE	.0649	.0516	.0419	.0512		.0374		.0310	
1ST LVL MMO (Δ%)	13.60	13.60	13.60	16.18	(19)	16.48	(21)	16.58	(22)
2ND LVL MMO (Δ%)	16.55	18.00	19.16	18.92	(14)	20.27	(13)	21.29	(11)
1ST LVL STP X 10^3 (Δ%)	208	173	152	203	(-2)	173	(0)	154	(1)
2ND LVL STP X 10^3 (Δ%)	278	258	247	2551	(8)	232	(-10)	227	(-8)
(1ST LVL WIN 20)									
1ST LVL FAULT RATE	.0852	.0852	.0852	.0728		.0736		.0736	
2ND LVL FAULT RATE	.0649	.0516	.0419	.0476		.0358		.0302	
1ST LVL MMO (Δ%)	14.89	14.09	14.89	17.74	(19)	17.77	(19)	17.65	(19)
2ND LVL MMO (Δ%)	16.55	18.00	19.16	19.24	(16)	20.39	(13)	21.03	(10)
1ST LVL STP X 10^3 (Δ%)	231	185	159	209	(-10)	170	(-8)	146	(-8)
2ND LVL STP X 10^3 (Δ%)	259	238	226	223	(-14)	205	(-14)	194	(-11)

TABLE II. FFT SIMULATION RESULTS

	Non-Restructured			Restructured					
	2ND LVL WIN=40	2ND LVL WIN=60	2ND LVL WIN=80	2ND LVL WIN=40		2ND LVL WIN=60		2ND LVL WIN=80	
(1ST LVL WIN 10)									
1ST LVL FAULT RATE	.2404	.2404	.2404	.1551		.1438		.1517	
2ND LVL FAULT RATE	.1000	.0810	.0784	.0625		.0524		.0602	
1ST LV MMO (Δ%)	26.43	26.43	26.43	34.55	(31)	33.84	(28)	34.70	(31)
2ND LVL MMO (Δ%)	31.23	32.07	33.73	37.04	(19)	36.14	(13)	38.09	(13)
1ST LVL STP X 10^3 (Δ)	250	203	206	210	(-16)	167	(-18)	203	(-1)
2ND LVL STP X 10^3 (Δ%)	338	273	295	253	(-25)	198	(-27)	248	(-16)
(1ST LVL WIN 20)									
1ST LVL FAULT RATE	.1614	.1614	.1614	.1045		.1000		.1091	
1ST LVL FAULT RATE	.1000	.0810	.0784	.0568		.0536		.0551	
1ST LVL MMO (Δ%)	28.64	28.64	28.64	40.29	(41)	38.39	(34)	40.20	(40)
2ND LVL MMO (Δ%)	31.23	32.07	33.73	41.77	(34)	39.57	(03)	42.35	(26)
1ST LVL STP X 10^3 (Δ%)	278	215	214	225	(-19)	192	(-11)	213	(0)
2ND LVL STP X 10^3 (Δ%)	316	252	271	226	(-28)	197	(-22)	232	(-14)

TABLE III. WATFIV SIMULATION RESULTS

	Non-Restructured			Restructured					
	2ND LVL STKSIZ=25	2ND LVL STKSIZ=30	2ND LVL STKSIZ=40	2ND LVL STKSIZ=25		2ND LVL STKSIZ=30		2ND LVL STKSIZ=40	
(1ST LVL WIN 10)									
1ST LVL FAULT RATE	.4203	.4203	.4203	.2468		.2227		.2358	
2ND LVL FAULT RATE	.3715	.1897	.0514	.1937		.0927		.0452	
1ST LVL MMO(Δ%)	22.46	22.46	22.46	24.68	(10)	23.50	(15)	24.98	(11)
1ST LVL STP X 10^3 (Δ%)	2178	1228	504	1121	(-49)	663	(-46)	430	(-15)
(1ST LVL WIN 20)									
1ST LVL FAULT RATE	.2600	.2600	.2600	.2086		.1489		.1603	
2ND LVL FAULT RATE	.8715	.1897	.0514	.2033		.0909		.0549	
1ST LVL MMO(Δ%)	25.33	25.33	25.33	22.08	(-13)	24.56	(-3)	28.82	(14)
1ST LVL STP X 10^3 (Δ%)	2326	1298	486	1275	(-45)	682	(-47)	510	(5)

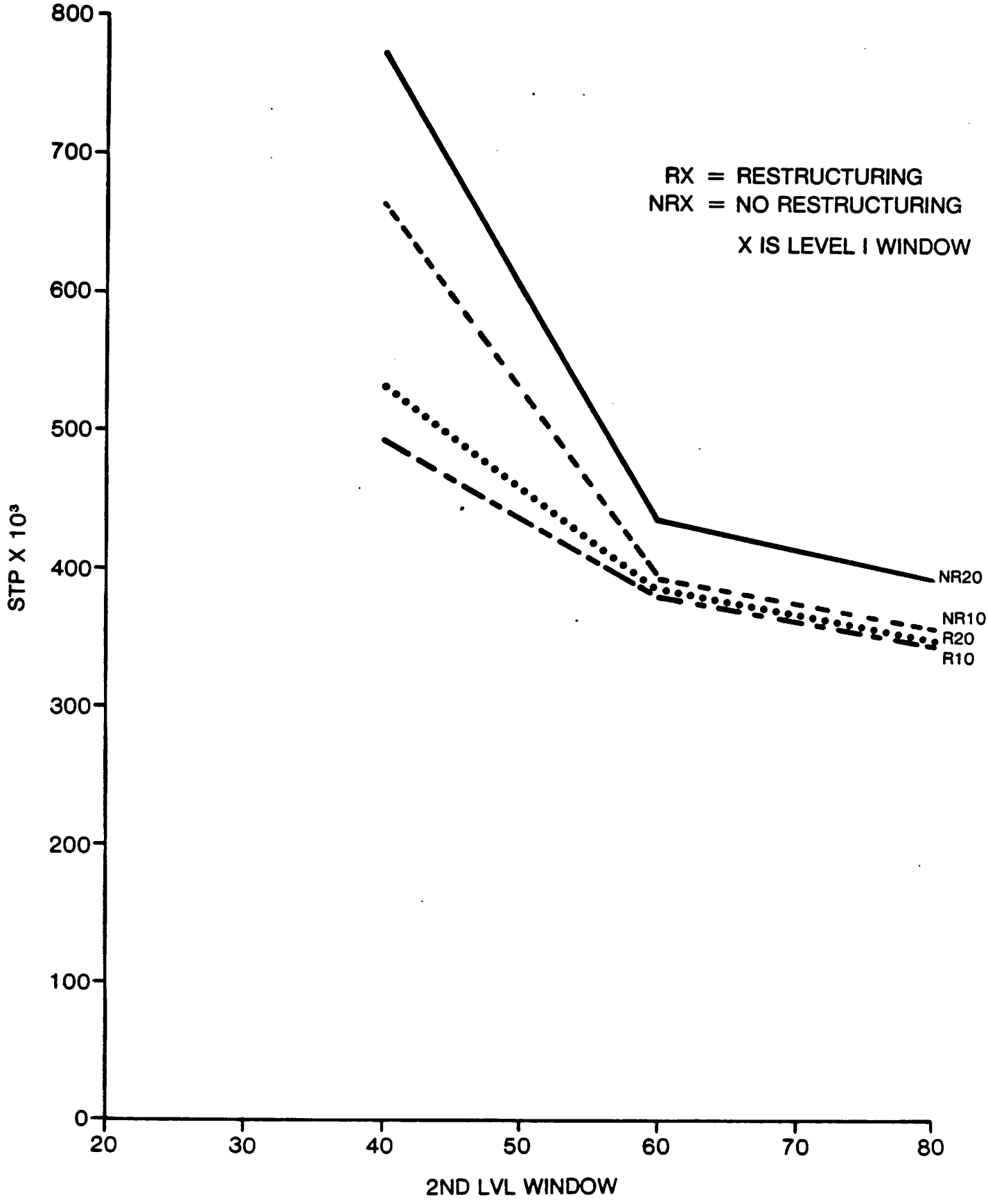
TABLE IV. APL SIMULATION RESULTS

	Non-Restructured			Restructured					
	2ND LVL STKSIZ=25	2ND LVL STKSIZ=30	2ND LVL STKSIZ-40	2ND LVL STKSIZ=25		2ND LVL STKSIZ-30		2ND LVL STKSIZ=40	
(1ST LVL WIN 10)									
1ST LVL FAULT RATE	.1528	.1528	.1528	.1437		.1509		.1437	
2ND LVL FAULT RATE	.0318	.0238	.0163	.0345		.0294		.0163	
1ST LVL MMO (Δ%)	13.61	13.61	13.61	15.89	(17)	16.44	(21)	15.66	(15)
1ST LVL STP X 10^3 (Δ%)	191	164	134	230	(20)	212	(29)	155	(16)
(1ST LVL WIN 20)									
1ST LVL FAULT RATE	.1020	.1020	.1020	.0929		.0945		.0925	
2ND LVL FAULT RATE	.0318	.0238	.0163	.0345		.0302		.0163	
1ST LVL MMO (Δ%)	14.85	14.85	14.85	17.02	(15)	17.53	(18)	17.09	(15)
1ST LVL STP X 10^3 (Δ%)	185	158	126	223	(4)	205	(30)	146	(16)

TABLE V. FFT SIMULATION RESULTS

	Non-Restructured			Restructured					
	2ND LVL STKSIZ=25	2ND LVL STKSIZ=30	2ND LVL STKSIZ-40	2ND LVL STKSIZ=25		2ND LVL STKSIZ-30		2ND LVL STKSIZ=40	
(IST LVL WIN 10)									
1ST LVL FAULT RATE	.3714	.3714	.3714	.2897		.2204		.2293	
2ND LVL FAULT RATE	1.0101	.1499	.0593	.1879		.1264		.0593	
1ST LVL MMO (Δ%)	26.52	26.52	26.52	22.93	(-14)	25.70	(-3)	30.74	(16)
1ST LVL STP X 10^3 (Δ%)	2223	410	209	4351	(-80)	344	(-16)	202	(-3)
(IST LVL WIN 20)									
1ST LVL FAULT RATE	.1957	.1957	.1957	.1902		.1443		.1141	
2ND LVL FAULT RATE	1.0101	.1499	.0593	.1823		.1230		.0604	
1ST LVL MMO (Δ%)	28.74	28.74	28.74	25.11	(-13)	26.98	(-6)	32.66	(14)
1ST LVL STP X 10^3 (Δ%)	2280	404	186	436	(-81)	337	(-17)	188	(1)

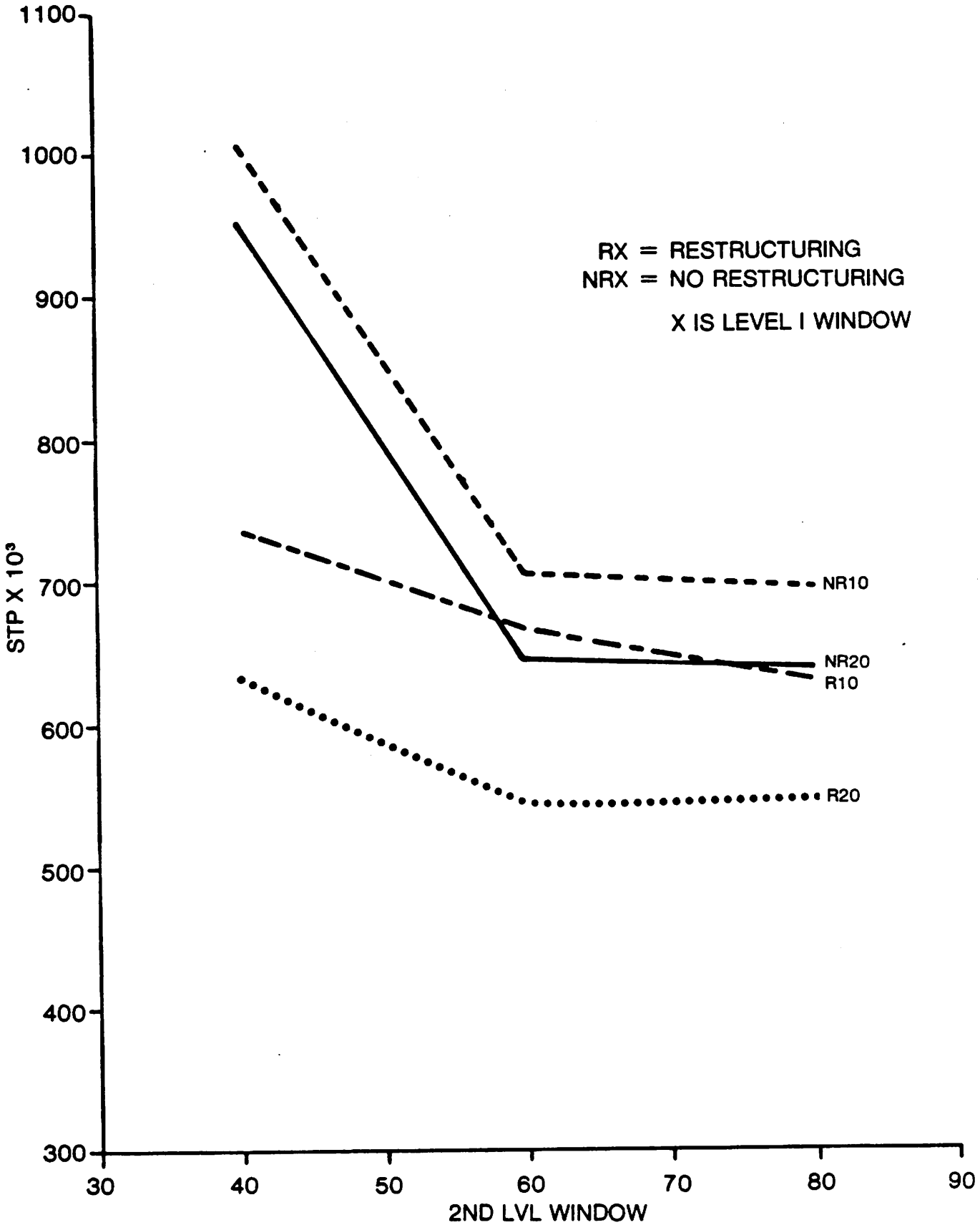
TABLE VI. WATFIV SIMULATION RESULTS



APL First Level Space Time Product vs Second Level Window Size

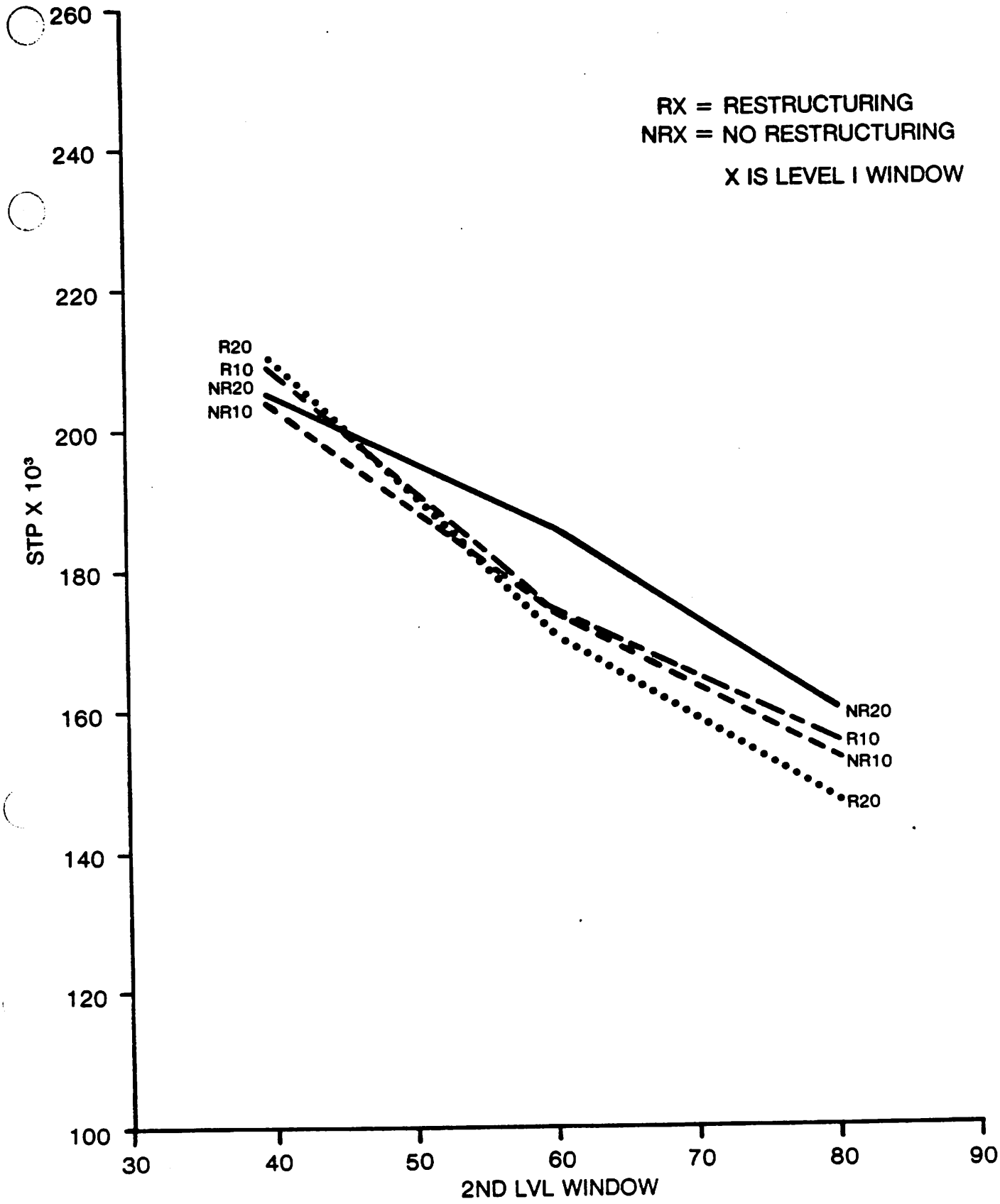
FIGURE 2

M02256013

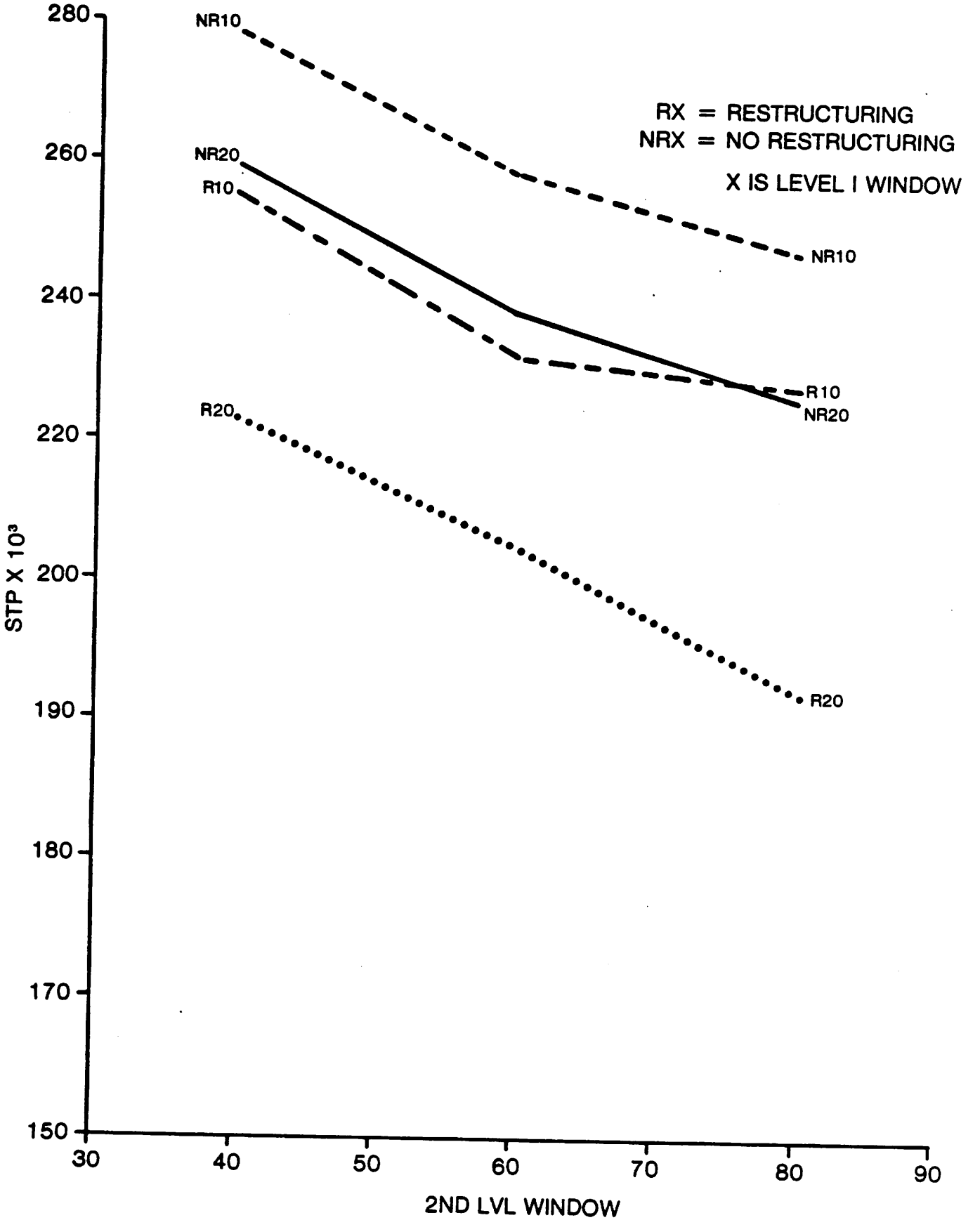


APL Second Level Space Time Product vs Second Level Window Size

FIGURE 3



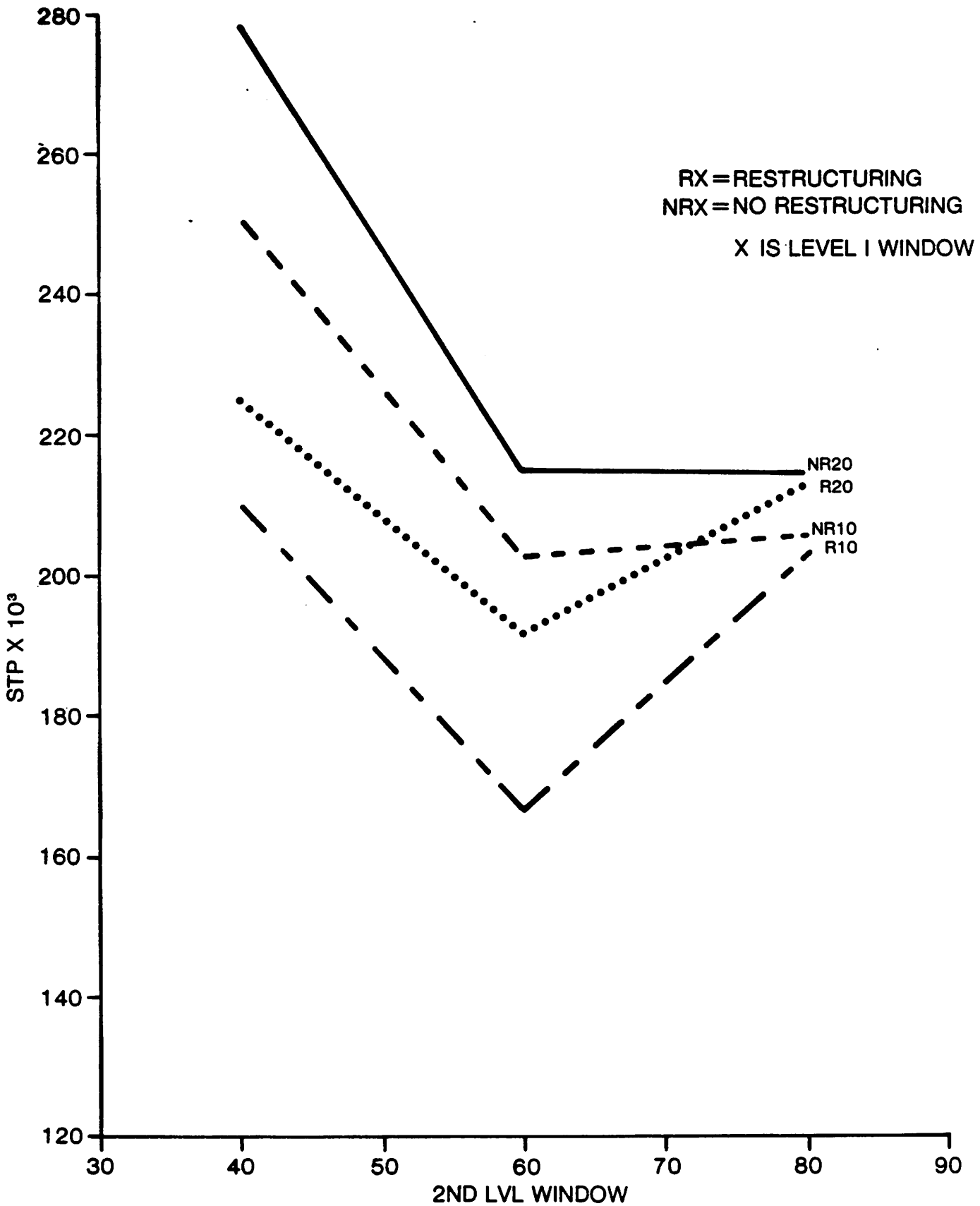
FFT First Level Space Time Product vs Second Level Window



FFT Second Level Space Time Product vs Second Level Window

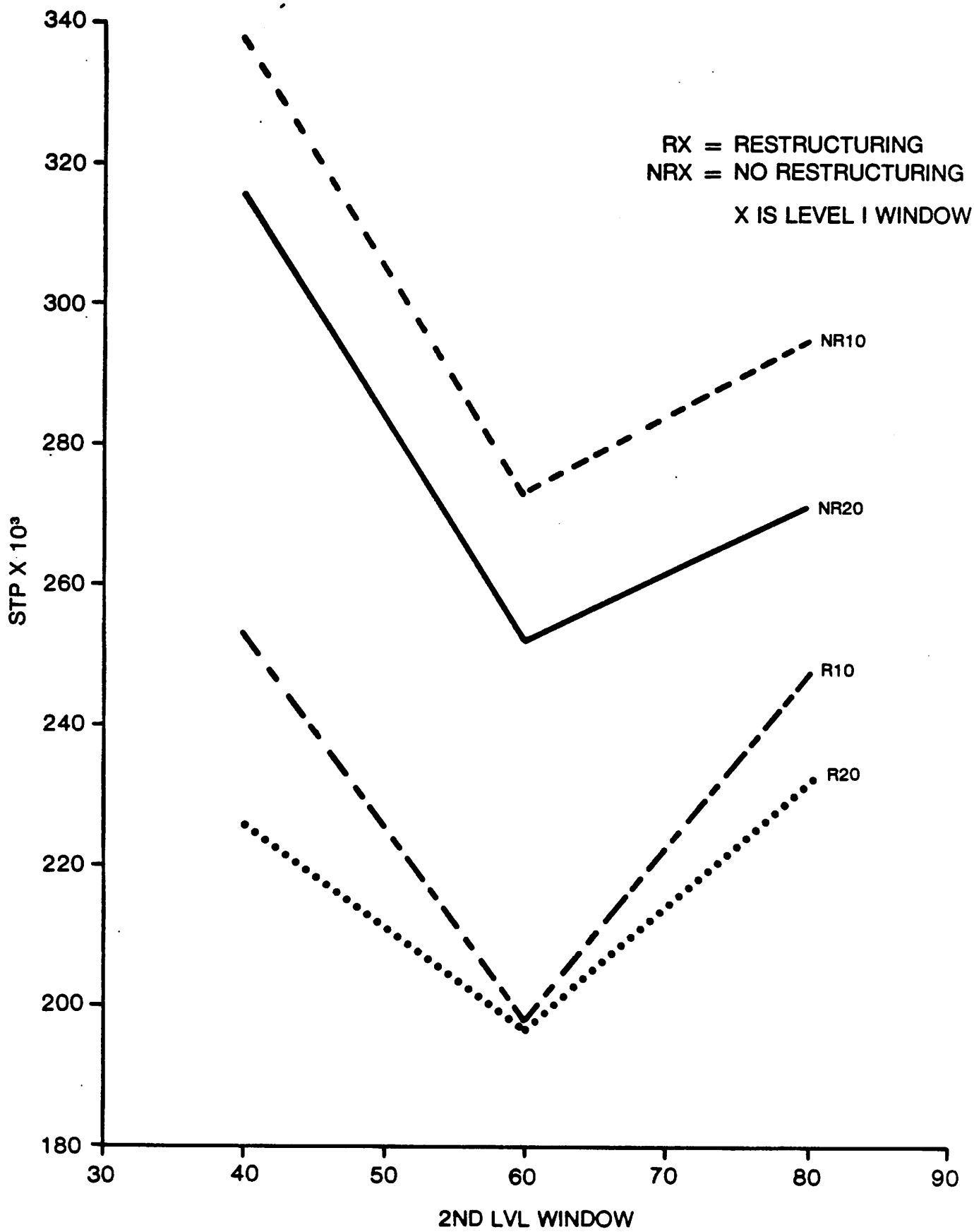
FIGURE 5

M02256010



WATFIV First Level Space Time Product vs Second Level Window

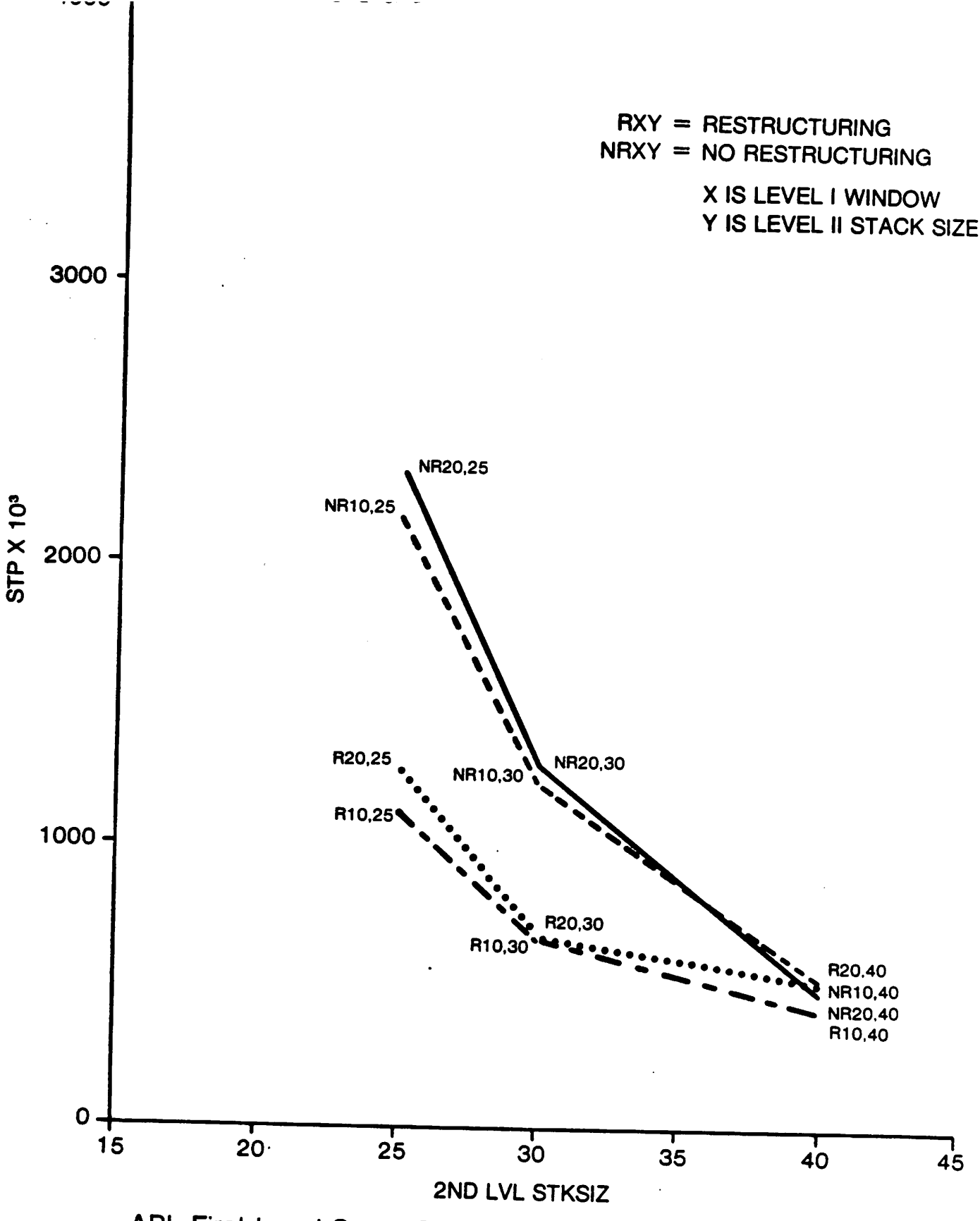
FIGURE 6



WATFIV Second Level Space Time Product vs Second Level Window

FIGURE 7

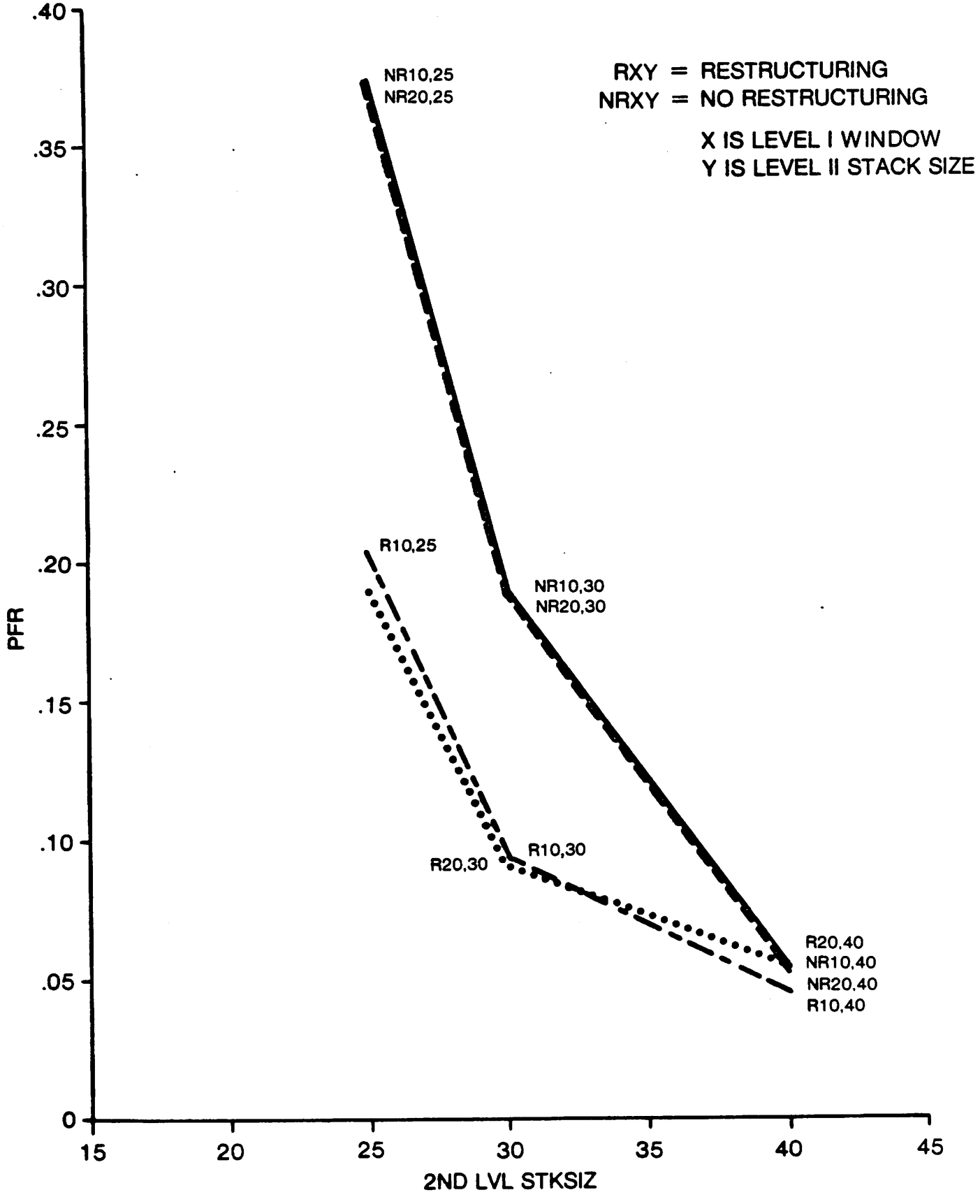
M02256008



APL First Level Space Time Product vs Second Level Stack Size

FIGURE 8

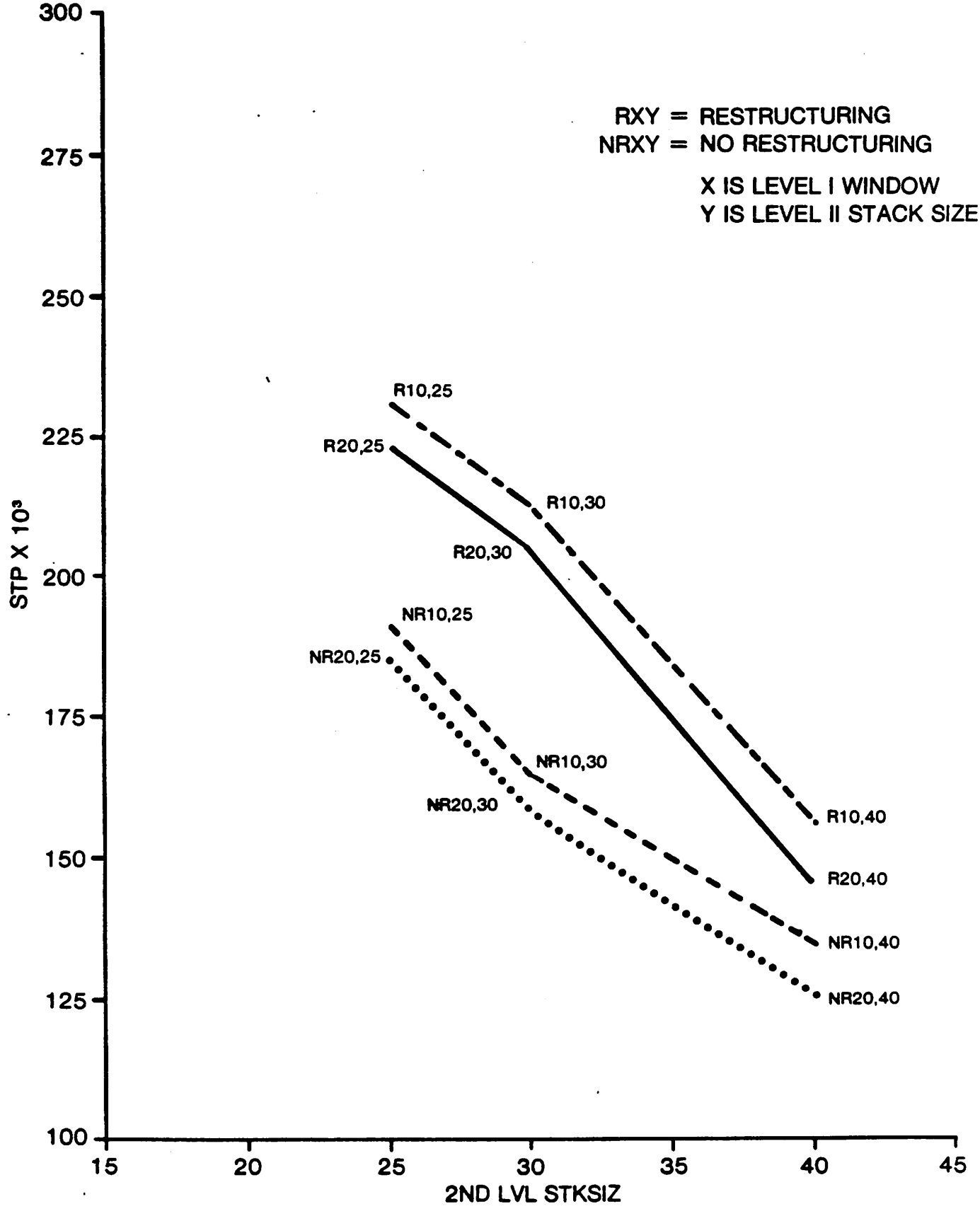
M02256007



APL Second Level Page Fault Rate vs Second Level Stack Size

FIGURE 9

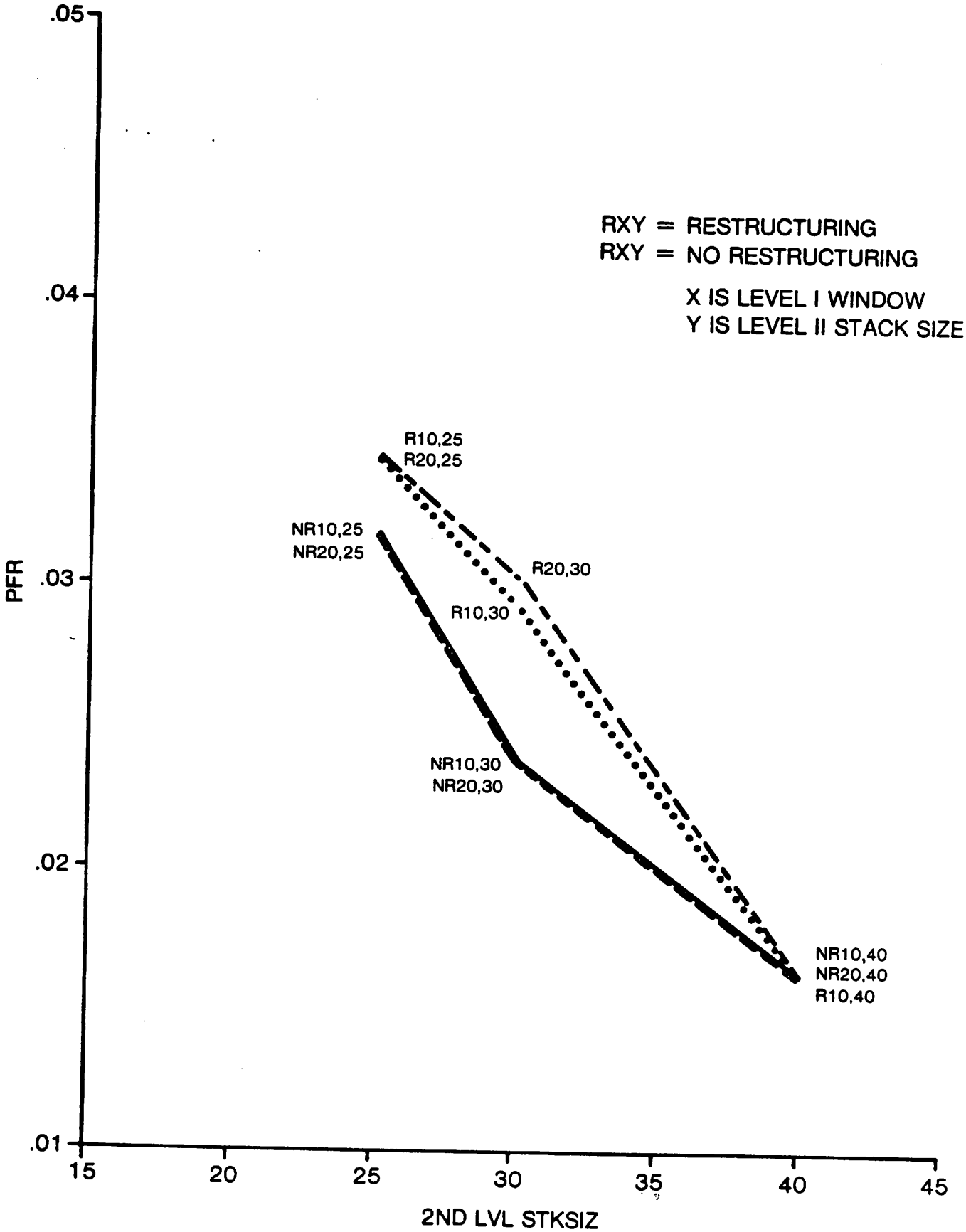
M02256006



FFT First Level Space Time Product vs Second Level Stack Size

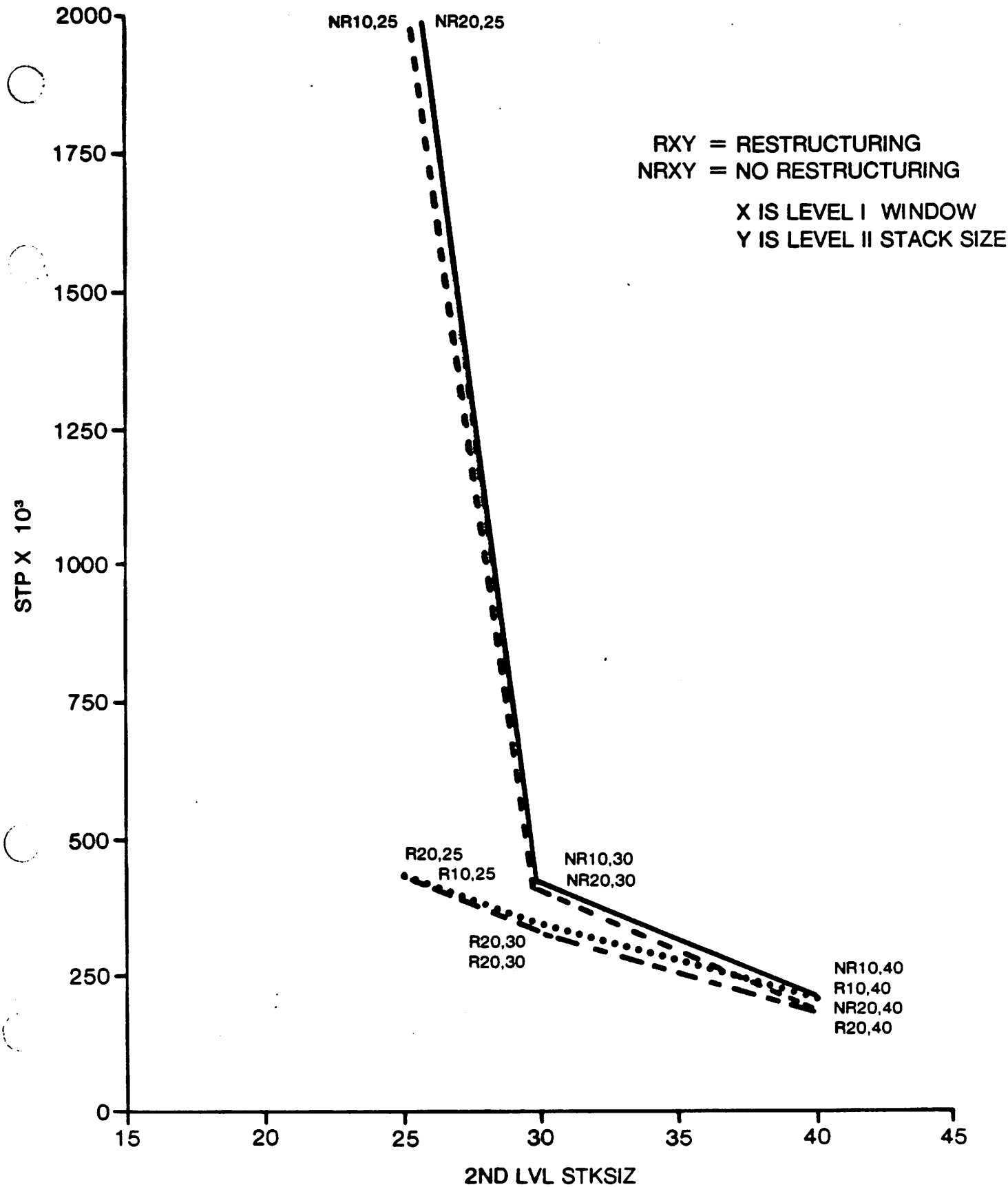
FIGURE 10

M02256005



Second Level Page Fault Rate vs Second Level Stack Size

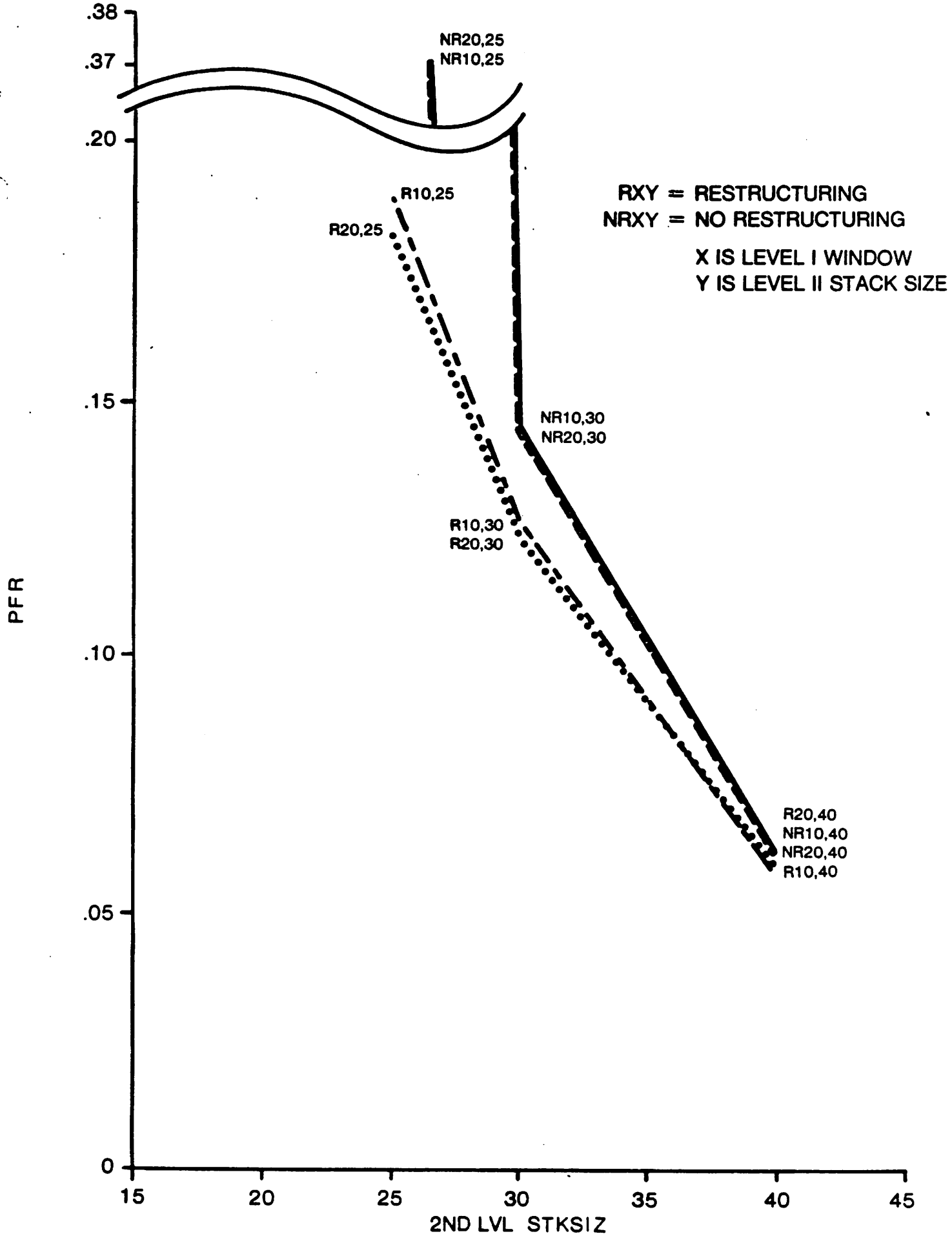
FIGURE 11



WATFIV First Level Space Time Product vs Second Stack Size

FIGURE 12

M02256003



WATFIV Second Level Page Fault Rate vs Second Level Stack Size

FIGURE 13