IMPACT OF PREDICTION ACCURACY ON THE PERFORMANCE OF A PIPELINE COMPUTER *

Anirban Basu **
Distributed Computing Group

SCS-TR-111
March 1987

School of Computer Science,
Carleton University,
Ottawa K1S 5B6
Canada

* Included in the Proceedings of the International Computer Symposium,Taiwan, December 1986

** On leave from the Computer Science Unit, Indian Statistical Institute, Calcutta 700035, India under the Commonwealth Fellowship Plan.

ABSTRACT

A general weakness of Pipeline Computers is 'bubbles' in the pipeline due to conditional branching. Most of the solutions to the branch problem attempt to predict whether or not a branch will be taken. If the prediction is correct, then initiation of the correct sequence of instructions can continue without delay. In this report, the performance of a Pipeline Processor is analysed to give a measure of the efficiency or utilisation of pipeline segments in terms of the probability of correct branch prediction (referred to as the prediction accuracy).

The analysis is based on the space-time relationship. The performance of a Pipeline Processor with unequal segment times is analysed noting that the two types of conditional branch instructions in the instruction sets of computers have different effects on the performance. The result of the analysis enable one to study the sensitivity of the performance of a Pipeline Processor to the number of conditional branch instructions and the probability of correct branch prediction as well as to estimate the utility of the different branch prediction strategies that have been proposed recently [2]. Although it is extremely difficult to obtain the value of 1 for prediction accuracy, this study reveals that a value of 0.8 can give reasonably good values of utilisation.

1. Introduction

A common architecture of most of today's supercomputing machines revolves around pipeline processing. However, a general weakness of Pipeline Processors is 'bubbles' in the pipeline due to the presence of conditional branch instructions in an instruction stream. Although a great deal of effort has been invested in overcoming the performance degradation due to the presence of conditional branches, its adverse effect cannot be overcome completely. Performance degradation from branches in the instruction stream can be reduced in a number of ways namely loop buffers, multiple instruction streams, prefetch branch target, delayed branch, taken/not taken switch and the branch target buffer. Most of the recent approaches to the branch problem attempt to predict whether or not a branch will be taken. If the prediction is correct, then execution of the correct sequence of instructions continues without delay. But if the prediction is wrong, the initiated instructions have to be 'squashed'. The performance of a Pipeline Processor therefore depends on the *prediction accuracy* i.e., on the probability of correct branch prediction. In this report, the performance of a Pipeline Processor is analysed to give a quantitative measure of the performance, relating it to the probability of correct branch prediction.

Performance of Pipeline Processors has been studied by a number of researchers. Chen [4] was the first to suggest the derivation of an expression for Utilisation or Efficiency of a Pipeline Processor from the space-time relationship. The processing of an instruction stream requires the occupation of equipment space (i.e., pipeline segments) over certain lengths of time. This is represented by the enclosed area of space-time diagram as in Figure 1(a), which shows the actual segment usage involved in the processing of L instructions of a job in a four segment pipeline. In [4], the Efficiency or Utilisation of a Pipeline Processor in executing a job has been given by the ratio of space-time of job to the space-time area swept by Pipeline Processor, when all the pipeline segments take the same amount of time for execution, successive instructions are independent and there is a steady flow of instructions through the pipeline. Ramamoorthy and Li [5] have given an expression for Efficiency when segment times are unequal and successive instructions are independent. Both Chen [4] and Ramamoorthy and Li [5] have given expressions for Efficiency on the assumption of a steady flow of instructions through the pipeline, which unfortunately does not hold in practice due to precedence constraints. Baskett and Keller [6] have reported the results of the studies conducted for evaluation of the performance of CRAY-1 computer while Holgate et. al. [7] have given the results of measurements made by hardware monitoring on MU5 system. Ramamoorthy and Wah [8] have studied the degradation in memory utilisation in a pipelined processor due to dependencies in the instruction stream.

In a Pipeline Processor, the different segments usually take different amounts of time for performing the assigned suboperation and the throughput is governed by the speed of the slowest segment referred to as the 'bottleneck'. Here the expression for Efficiency or Utilisation of pipeline segments is deduced from the space-time or the geometric model for a pipeline with unequal segment times incorporating the effect of conditional branches and the probability of correct branch prediction. Study of the instruction set of different computers reveals that most of them have two types of conditional branch instructions: those which branch to a target address depending on the testing of the condition code set by some previous instruction and those which transfer control to some address depending on the outcome of certain computation which it performs. They have different effects on the performance and the analysis takes this into account. To the author's knowledge this is the first time that such an analysis is done. The results given here enable one to study the sensitivity of the performance of a pipeline processor to various factors including prediction accuracy. The simplicity of the analysis followed for obtaining the values of the different parameters makes the methodology attractive.

For the purpose of analysis, it is assumed that the Pipeline Processor is provided with a fully asynchronous control structure i.e., it allows independent instructions after the conflicting one to continue. This overcomes the problem created by data dependencies among the instructions. Further, the pipeline has multiple arithmetic logic units to avoid the effect of operational dependency. Therefore the performance of the pipeline is degraded only when a branch instruction is encountered. The expression for Efficiency is deduced in terms of the number of conditional branch instructions and the total number of instructions executed. These can be estimated from the run time characteristics of a program to determine its suitability for processing on a Pipeline Processor.

The methodology followed here is illustrated in section 2 by taking the case when successive instructions are independent. In section 3, this technique is applied to obtain all the necessary expressions in the more general case. The effect of prediction accuracy is taken into account in the expressions deduced in section 4.

2. Efficiency with no Precedence Constraint

In this section, an expression for Efficiency is deduced for a pipeline with s segments, when instructions are independent and segment times are unequal.

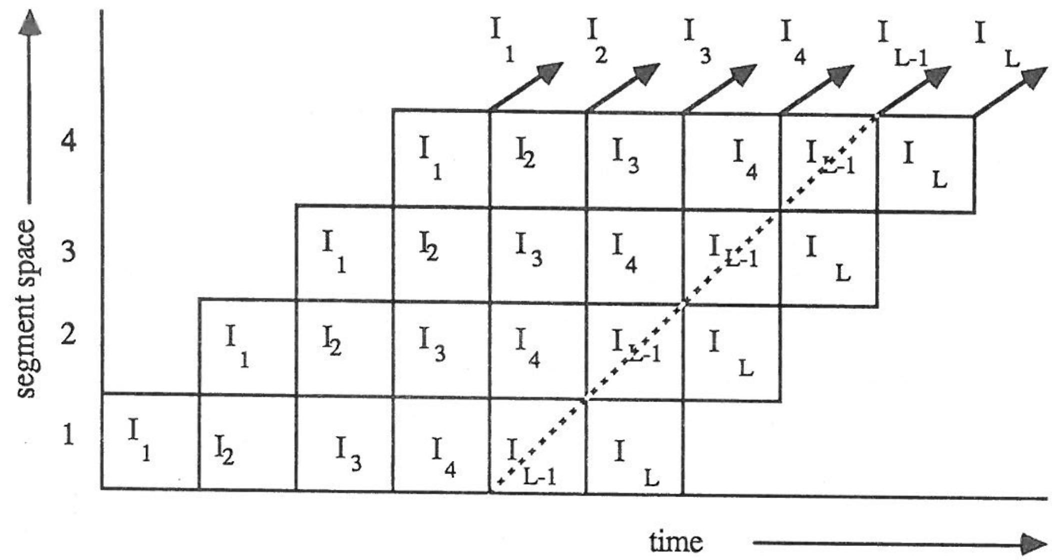

Figure 1(a) Space-time diagram when processing L instructions ($I_1, I_2, .., I_L$) in a four segment pipeline

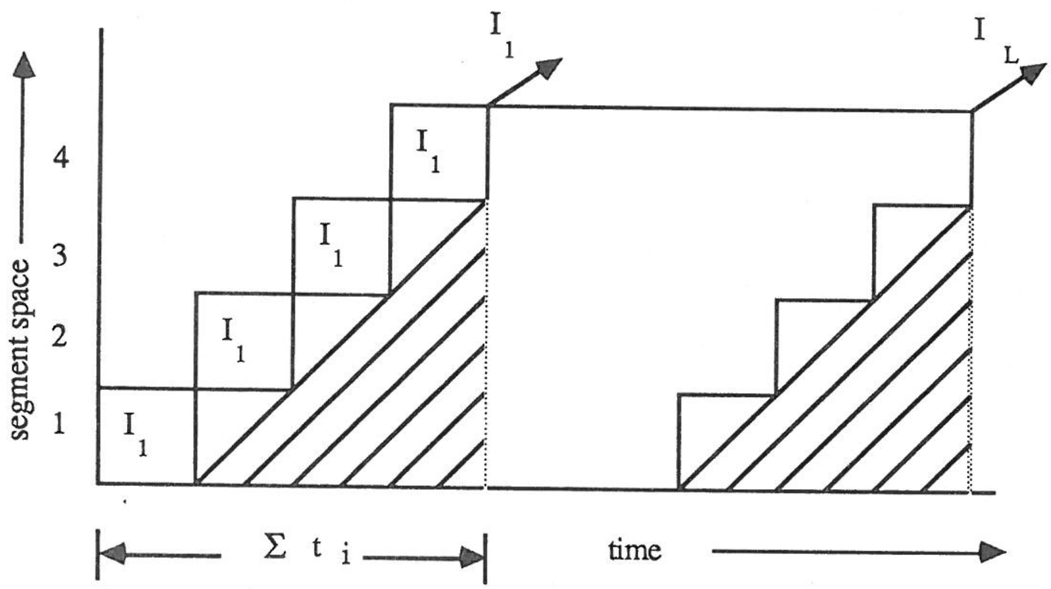

Figure 1(b) Space-time diagram of Figure 1(a) recast.
The shaded regions are equal in area

Let the segments of the pipeline be $s_1, s_2, s_3, \ldots\ldots, s_s$.

Time required by segments $s_1, s_2, \ldots\ldots, s_s$ are $t_1, t_2, t_3, \ldots, t_s$ respectively.

The processing of a sequence of L instructions is analysed.

Each instruction occupies a space time $\Sigma^{s}_{i=1} t_i$ assuming segment space to be unity.

Space-time area of job consisting of L instructions $= L\, \Sigma^{s}_{i=1} t_i$

Space-time area swept by pipeline processor will include the time lost before the first result is outputted i.e., during initialisation and also during termination of last instruction.

If segment times are equal to τ, then

Space-time lost due to initialisation $= \Sigma^{s-1}_{j=1} (s - j)\, \tau$

where $t_1 = t_2 = t_3 = \cdots = t_s = \tau$

Space-time lost during termination $= \Sigma^{s-1}_{j=1}\, j\, \tau$

Total space-time lost due to initialisation and termination is by the addition of the above two expressions $= s(s - 1)\, \tau$

The space-time diagram may be recast for $L > s$, as shown in Figure 1(b)

Time required for outputting the first instruction $= \Sigma^{s}_{i=1} t_i$

For L-1 instructions, the space-time area swept by Pipeline Processor

$= s\,(L - 1) * \max\,(t_1, t_2, \ldots\ldots, t_s)$

$= s\,(L - 1)\, t_b$

where t_b corresponds to the time of the slowest segment of pipeline called the 'bottleneck' $=$ $\max\,(t_1, t_2, \ldots\ldots t_s)$

$$\text{Efficiency } \eta = \frac{\text{space-time of job}}{\text{space-time swept by a pipeline processor}}$$

$$= \frac{L\, \Sigma^{s}_{i=1} t_i}{s\, \Sigma^{s}_{i=1} t_i + s(L - 1)\, t_b}$$

If segment times are equal then $t_1 = t_2 = t_3 = \ldots = t_s = \tau$

then Efficiency $\eta = \frac{L}{s + L - 1}$ as shown by Chen [4]

3. Efficiency with Procedural Dependency

The above analysis holds for a continuous excitation of the pipeline assuming no time is lost for satisfying dependencies among instructions, which delays the execution time of the program and results in lower utilisation of the pipeline segments and loss of Efficiency. Maximum penalty is paid when the result of instruction i is used by instruction $i+1$. When segment times are assumed equal to τ, a maximum delay of $(s - 1)\,\tau$ may be introduced into the stream for satisfying a dependency, if conflict is detected in the first stage of the pipeline. In a Pipeline Processor instructions are normally executed in sequence till a branch instruction causes a disruption of the sequence. If the instruction is an unconditional branch, then the dependency can be resolved without much delay and removed by fetching the correct sequence of instructions. But, if it is a conditional branch instruction, data test has to be made for the specified condition. Only then can the correct sequence of instructions be fetched and execution of the instructions is held up till the condition is tested and branch path determined. Here the degradation is serious and unavoidable.

As in section 2, for simplifying the analysis it is assumed that the Pipeline Processor is equipped with a fully asynchronous control structure which allows independent instructions after the conflicting one to continue and the pipeline has a number of arithmetic logic units so that there is no delay due to contention for the execution unit. It is assumed that no computation is required for branch address calculation such as addition of displacement to a base or index register. The instruction fetch stage may have a buffer for storing instructions in order to reduce main memory access and has logic for prefetching the branch target. That is, when a branch is recognised, a special mechanism calculates and prefetches the target of the branch. Thus, if the branch is found to be taken, the target instruction is loaded immediately into the instruction decode stage of the pipeline, with no additional delay for fetching the instruction.

Conditional branch instructions in an instruction stream are of two types: those which branch to a target address depending on the testing of the condition code set by some previous instruction and those which transfer control to some address depending on the outcome of certain computation which it performs. For example in IBM 360/370 system

[10], branch instructions with pnemonics BC and BCR fall into the first category and BXLE, BXH, BCT, BCTR are of the second type. Let the number of instructions in these two categories be d_1 and d_2 respectively. Let d_3 be the number of unconditional branch instructions in an instruction stream among the total number of instructions L.

τ is the segment time.

If N be the number of instructions in between the instruction setting a condition code and the instruction which tests this result then for $N \geq s - 1$, no delay is encountered.

For $N < s - 1$, the dependency causes a delay of $(s - 1 - N)\ \tau$ to the execution time of the program.

Conditional branch instructions of the first type (i.e., d_1) and Unconditional branch instructions (i.e., d_3) are assumed to occupy no segment space.

Then, space-time of job = $(L - d_1 - d_3)\ s\ \tau$

Space-time lost waiting for condition code generation for one conditional branch instruction =

$(s - 1 - N)\ s\ \tau$

Total space-time lost waiting for condition code generation due to d_1 branch instructions would be:

$d_1(s - 1)\ s\ \tau - (\Sigma^{d_1}{}_{i=1} N_i)\ s\ \tau = d_1\ [s - 1 - N_{av}]\ s\ \tau$, where N_i is the number of instructions in between the instruction setting a condition code and the branch instruction testing it for the i th branch instruction and N_{av} = Average value of $N_i = \Sigma^{d_1}{}_1\ N_i / d_1$

Conditional branch instructions of the second type cause serious degradation in performance as successive instructions cannot be initiated till these branch instructions finish the computation and determine the target address. If the target instruction is present in the instruction buffer in the Instruction Fetch stage, then memory access time may be neglected.

For each of this type of instruction, the execution of the instruction stream is delayed by $(s - 1)$ τ time units.

Hence space-time lost due to d_2 conditional branch instructions is $d_2\ (s - 1)\ s\ \tau$.

It has already been shown in section 2 that the space-time lost due to initialisation and termination operation = $s\ (s - 1)\ \tau$

Total space-time area swept by Pipeline Processor =

$$s\,(s-1)\,\tau + d_1(s-1-N_{av})\,s\,\tau + d_2\,(s-1)\,s\,\tau + (L-d_1-d_3)\,s\,\tau$$

Therefore, Efficiency $\eta = \dfrac{(L-d_1-d_3)\,s\,\tau}{(L-d_1-d_3)\,s\,\tau + s\,(s-1)\,\tau + d_1(s-1-N_{av})\,s\,\tau + d_2(s-1)\,s\,\tau}$

$$= \frac{(L-d_1-d_3)}{(L-d_1-d_3)+(s-1)+d_1\,(s-1-N_{av})+d_2\,(s-1)} \qquad (1)$$

When segment times are unequal the following holds:

Space-time of job = $(L-d_1-d_3)\ \Sigma^{s}_{i=1}\ t_i$

Space-time area swept by Pipeline Processor is obtained from the recast space-time diagram in the same way as discussed in section 2 and is given by

$$s[\,\Sigma^{s}_{i=1}\ t_i + \{\,d_1(s-1-N_{av}) + (L-d_1-d_3-1) + d_2(s-1)\}\ t_b]$$

Therefore, $\eta = \dfrac{(L-d_1-d_3)\ \Sigma^{s}_{i=1}\ t_i}{s\,\Sigma^{s}_{i=1}\ t_i + s\,\{d_1\,(s-1-N_{av}) + d_2(s-1) + (L-d_1-d_3-1)\}\ t_b}$ (2)

4. Effect of Prediction Accuracy

Review of the mechanisms that have been proposed for reducing the delay due to presence of conditional branch instructions in an instruction stream reveals that an acceptable solution has to be based on predicting whether or not a branch will be taken and conditionally forwarding the next instruction to the pipeline either from the target address or from the next sequential address depending upon the guess. If the prediction is correct, then execution continues instantaneously without time delay. To take the effect of prediction into account the expression for Efficiency deduced above is modified, as correct prediction will reduce the waiting time. This is done as follows.

If p is the probability that the branch prediction is correct (which is referred to as the prediction accuracy), then the value of the Efficiency can be obtained by substituting $d_1*(1 - p)$ and $d_2*(1 - p)$ for d_1 and d_2 in the expressions (1) and (2) deduced above.

In this case, the expression for η is as follows:

$$\eta = \frac{\{L - d_1(1-p) - d_3\} \Sigma^s_{i=1} t_i}{s\, \Sigma^s_{i=1} t_i + s\{ d_1(1-p)*(s-1-N_{av}) + d_2(1-p)*(s-1) + L - d_1(1-p) - d_3 - 1\} t_b} \qquad (3)$$

When segment times are equal, the value of η becomes,

$$\eta = \frac{\{L - d_1(1-p) - d_3\}}{s + d_1(1-p)*(s-1-N_{av}) + d_2(1-p)*(s-1) + L - d_1(1-p) - d_3 - 1} \qquad (4)$$

The expressions (3) and (4) deduced above enable one to study the sensitivity of the performance of a Pipeline Processor to the prediction accuracy. The variation of η with p may be obtained from this expression for different values of d_1, d_2, d_3, L and N_{av} for a pipeline with s number of segments. Values of d_1, d_2, d_3, L and N_{av} can be obtained by dynamic measurement on application programs. In Figure 2, η is plotted against p for some typical values of d_1, d_2, d_3, L and N_{av} to show the variation of Efficiency with the prediction accuracy for a five segment pipeline with equal segment times. The results plotted in the graph of Figure 2 indicate that a prediction accuracy of 0.8 can give reasonably good values of utilisation of pipeline segments.

4. Conclusion

Here an expression for efficiency of a Pipeline Processor has been deduced in terms of L, d_1,d_2,d_3, N_{av} and p. Efficiency or Utilisation of the segments of a Pipeline Processor varies with the number of different types of branch instructions in an instruction stream, which in turn depends on the application program. While the degradation in the performance of a Pipeline Processor due to the presence of conditional branch instructions can not be nullified with all the methods that have been tried, research on Pipeline Processors is presently directed

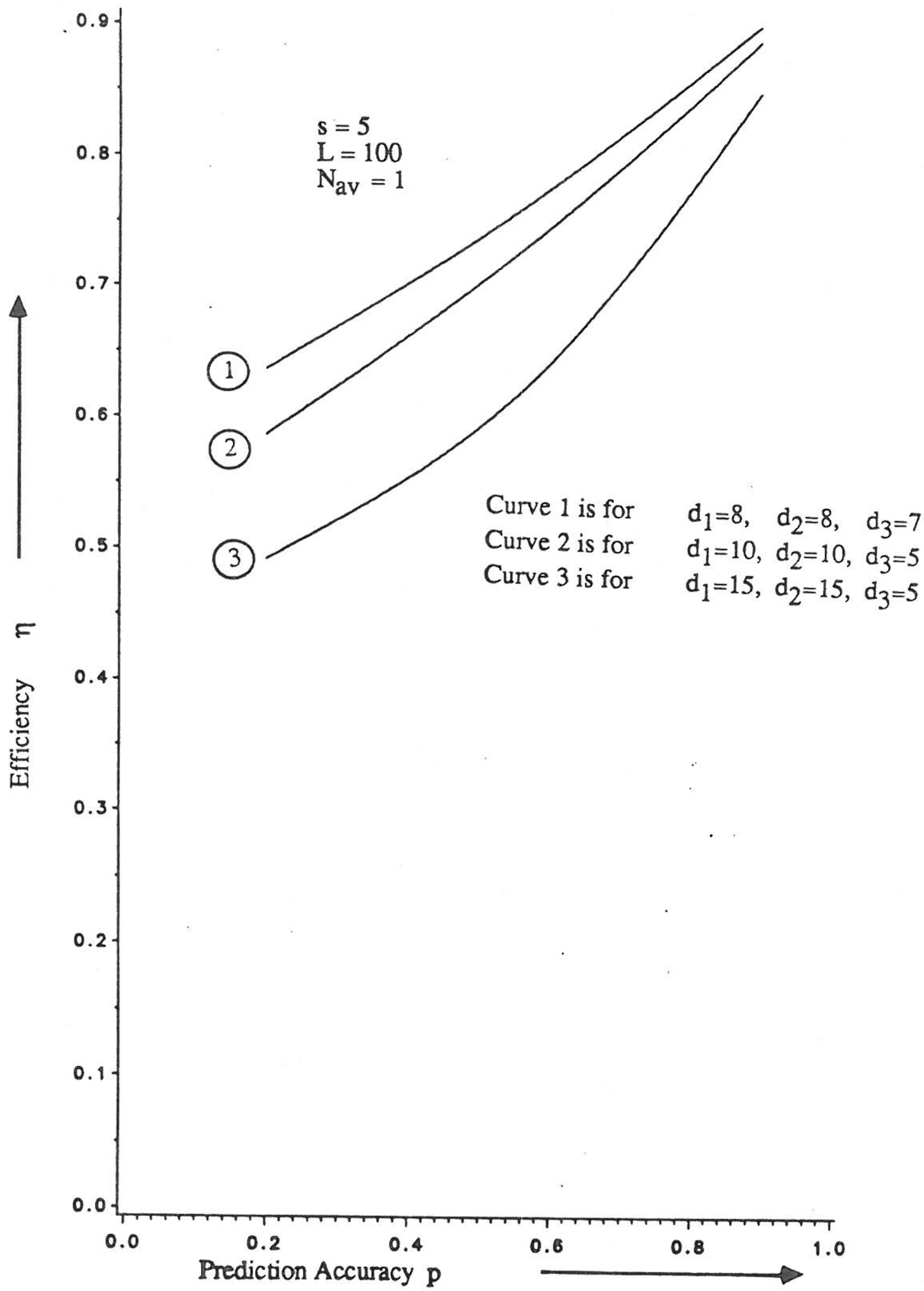

Figure 2 Variation of *Efficiency* with *Prediction Accuracy* for a Pipeline with equal segment times.

at reducing its adverse effect. The design philosophy for that purpose has centered around predicting the branch direction whenever a conditional branch instruction is encountered. Correct prediction enables initiation of instructions without any delay. Many strategies have been studied [2][3] for the improvement of the prediction accuracy and the results indicate that it is extremely difficult to obtain the value of 1 for p. However, this study reveals that a prediction accuracy of 0.8 gives reasonably good values of utilisation. It should be mentioned here, that this value of prediction accuracy can be obtained with most of the branch prediction strategies that have been proposed.

Although the performance of Pipeline Processors has been studied in the past, this the first time that an expression has been deduced from the geometric model which enables one to study the sensitivity of the performance of a Pipeline Processor to the value of prediction accuracy. Here the main aim has been to develop a methodology which can give an expression for efficiency by simple analysis. Besides studying the effect of prediction accuracy on the performance of a Pipeline Processor, the expressions deduced in this report can also be used to determine the suitability of a program for processing on a given Pipeline Processor. For this, the values of d_1, d_2, d_3, L and N_{av} have to be measured dynamically i.e., during execution of the program.

Acknowledgement

The author wishes to thank Prof. Nicola Santoro of the Distributed Computing Group, School of Computer Science for his valuable suggestions.

References

[1] C V Ramamoorthy and H F Li , 'Pipeline Architecture', ACM Computing Surveys, Vol 9, No.1, March 1977, pp. 61-102

[2] J K F Lee and A J Smith , 'Branch Prediction Strategies and Branch Target Buffer Design ', IEEE Computer, Vol. 17, No.1, January 1984

[3] J E Smith, 'A Study of Branch Prediction Strategies', Proc. Eighth Symposium on Computer Architecture, 1981, pp. 135-148

[4] T C Chen , 'Parallelism, Pipelining and Computer Efficiency', Computer Design, Vol. 10, No.1, January 1971, pp. 69-74

[5] C V Ramamoorthy and H F Li, 'Efficiency in Generalised Pipeline Networks', Proc. AFIPS Nat. Computer Conference, 1974, pp. 625- 635

[6] F Baskett and T W Keller, 'An Evaluation of the CRAY -1 Computer', in Kuck et.al. (Eds.), High Speed Computer and Algorithm Organisation, New York: Academic Press, 1977, pp.71-84

[7] R W Holgate and R N Ibbett , 'An Analysis of Instruction Fetching Strategies in Pipelined Computers', IEEE Trans. on Computers, Vol. C-29, No.4, April 1980, pp. 325-329

[8] C V Ramamoorthy and B W Wah, 'An Optimal Algorithm for Scheduling Requests on Interleaved Memories for a Pipelined Processor', IEEE Trans. on Computers, Vol. C-30, No. 10, October, 1981, pp. 787-799

[9] D W Anderson, F J Sparacio and R M Tomasulo, 'The IBM System /360 Model 91: Machine Philosophy and Instruction Handling', IBM Journal of Research and Development, Vol.11, No. 1, January 1967, pp. 8-24

[10] IBM Corporation, IBM System /360 Principles of Operation, New York, 1970

Carleton University, School of Computer Science
Bibliography of Technical Reports
Publications List (1985 -->)

School of Computer Science
Carleton University
Ottawa, Ontario, Canada
KIS 5B6

SCS-TR-66 — **On the Futility of Arbitrarily Increasing Memory Capabilities of Stochastic Learning Automata**
B.J. Oommen, October l984. Revised May l985.

SCS-TR-67 — **Heaps in Heaps**
T. Strothotte, J.-R. Sack, November l984. Revised April l985.

SCS-TR-68 out-of-print — **Partial Orders and Comparison Problems**
M.D. Atkinson, November l984. See Congressus Numerantium 47 ('86), 77-88

SCS-TR-69 — **On the Expected Communication Complexity of Distributed Selection**
N. Santoro, J.B. Sidney, S.J. Sidney, February l985.

SCS-TR-70 — **Features of Fifth Generation Languages: A Panoramic View**
Wilf R. LaLonde, John R. Pugh, March l985.

SCS-TR-7l — **Actra: The Design of an Industrial Fifth Generation Smalltalk System**
David A. Thomas, Wilf R. LaLonde, April l985.

SCS-TR-72 — **Minmaxheaps, Orderstatisticstrees and their Application to the Coursemarks Problem**
M.D. Atkinson, J.-R. Sack, N. Santoro, T. Strothotte, March l985.

SCS-TR-73 — **Designing Communities of Data Types**
Wilf R. LaLonde, May l985.
Replaced by SCS-TR-108

SCS-TR-74 out-of-print — **Absorbing and Ergodic Discretized Two Action Learning Automata**
B. John Oommen, May l985. See IEEE Trans. on Systems, Man and Cybernetics, March/April 1986, pp. 282-293.

SCS-TR-75 — **Optimal Parallel Merging Without Memory Conflicts**
Selim Akl and Nicola Santoro, May l985

SCS-TR-76 — **List Organizing Strategies Using Stochastic Move-to-Front and Stochastic Move-to-Rear Operations**
B. John Oommen, May 1985.

SCS-TR-77 — **Linearizing the Directory Growth in Order Preserving Extendible Hashing**
E.J. Otoo, July 1985.

SCS-TR-78 — **Improving Semijoin Evaluation in Distributed Query Processing**
E.J. Otoo, N. Santoro, D. Rotem, July 1985.

SCS-TR-79 **On the Problem of Translating an Elliptic Object Through a Workspace of Elliptic Obstacles**
B.J. Oommen, I. Reichstein, July 1985.

SCS-TR-80 **Smalltalk - Discovering the System**
W. LaLonde, J. Pugh, D. Thomas, October l985.

SCS-TR-81 **A Learning Automation Solution to the Stochastic Minimum Spanning Circle Problem**
B.J. Oommen, October 1985.

SCS-TR-82 **Separability of Sets of Polygons**
Frank Dehne, Jörg-R. Sack, October 1985.

SCS-TR-83 out-of-print **Extensions of Partial Orders of Bounded Width**
M.D. Atkinson and H.W. Chang, November 1985. See Congressus Numerantium, Vol. 52 (May 1986), pp. 21-35.

SCS-TR-84 **Deterministic Learning Automata Solutions to the Object Partitioning Problem**
B. John Oommen, D.C.Y. Ma, November 1985

SCS-TR-85 out-of-print **Selecting Subsets of the Correct Density**
M.D. Atkinson, December 1985. To appear in Congressus Numerantium, Proceedings of the 1986 South-Eastern conference on Graph theory, combinatorics and Computing.

SCS-TR-86 **Robot Navigation in Unknown Terrains Using Learned Visiblity Graphs. Part I: The Disjoint Convex Obstacles Case**
B. J. Oommen, S.S. Iyengar, S.V.N. Rao, R.L. Kashyap, February 1986

SCS-TR-87 **Breaking Symmetry in Synchronous Networks**
Greg N. Frederickson, Nicola Santoro, April 1986

SCS-TR-88 **Data Structures and Data Types: An Object-Oriented Approach**
John R. Pugh, Wilf R. LaLonde and David A. Thomas, April 1986

SCS-TR-89 **Ergodic Learning Automata Capable of Incorporating Apriori Information**
B. J. Oommen, May 1986

SCS-TR-90 **Iterative Decomposition of Digital Systems and Its Applications**
Vaclav Dvorak, May 1986.

SCS-TR-91 **Actors in a Smalltalk Multiprocessor: A Case for Limited Parallelism**
Wilf R. LaLonde, Dave A. Thomas and John R. Pugh, May 1986

SCS-TR-92 **ACTRA - A Multitasking/Multiprocessing Smalltalk**
David A. Thomas, Wilf R. LaLonde, and John R. Pugh, May 1986

SCS-TR-93 **Why Exemplars are Better Than Classes**
Wilf R. LaLonde, May 1986

SCS-TR-94 **An Exemplar Based Smalltalk**
Wilf R. LaLonde, Dave A. Thomas and John R. Pugh, May 1986

SCS-TR-95 **Recognition of Noisy Subsequences Using Constrained Edit Distances**
B. John Oommen, June 1986

SCS-TR-96 **Guessing Games and Distributed Computations in Synchronous Networks**

J. van Leeuwen, N. Santoro, J. Urrutia and S. Zaks, June 1986.

SCS-TR-97 **Bit vs. Time Tradeoffs for Distributed Elections in Synchronous Rings**
M. Overmars and N. Santoro, June 1986.

SCS-TR-98 **Reduction Techniques for Distributed Selection**
N. Santoro and E. Suen, June 1986.

SCS-TR-99 **A Note on Lower Bounds for Min-Max Heaps**
A. Hasham and J.-R. Sack, June 1986.

SCS-TR-100 **Sums of Lexicographically Ordered Sets**
M.D. Atkinson, A. Negro, and N. Santoro, May 1987.

SCS-TR-102 **Computing on a Systolic Screen: Hulls, Contours, and Applications**
F. Dehne, J.-R. Sack and N. Santoro, October 1986.

SCS-TR-103 **Stochastic Automata Solutions to the Object Partitioning Problem**
B.J. Oommen and D.C.Y. Ma, November 1986.

SCS-TR-104 **Parallel Computational Geometry and Clustering Methods**
F. Dehne, December 1986.

SCS-TR-105 **On Adding *Constraint Accumulation* to Prolog**
Wilf R. LaLonde, January 1987.

SCS-TR-107 **On the Problem of Multiple Mobile Robots Cluttering a Workspace**
B. J. Oommen and I. Reichstein, January 1987.

SCS-TR-108 **Designing Families of Data Types Using Exemplars**
Wilf R. LaLonde, February 1987.

SCS-TR-109 **From Rings to Complete Graphs - $\Theta(n \log n)$ to $\Theta(n)$ Distributed Leader Election**
Hagit Attiya, Nicola Santoro and Shmuel Zaks, March 1987.

SCS-TR-110 **A Transputer Based Adaptable Pipeline**
Anirban Basu, March 1987.

SCS-TR-111 **Impact of Prediction Accuracy on the Performance of a Pipeline Computer**
Anirban Basu, March 1987.

SCS-TR-112 **ε-Optimal Discretized Linear Reward-Penalty Learning Automata**
B.J. Oommen and J.P.R. Christensen, May 1987.

SCS-TR-113 **Angle Orders, Regular n-gon Orders and the Crossing Number of a Partial Order**
N. Santoro and J. Urrutia, June 1987.

SCS-TR-115 **Time Is Not a Healer: Impossibility of Distributed Agreement in Synchronous Systems with Random Omissions**
N. Santoro, June 1987.

SCS-TR-116 **A Practical Algorithm for Boolean Matrix Multiplication**
M.D. Atkinson and N. Santoro, June 1987.

SCS-TR-117 **Recognizing Polygons, or How to Spy**
James A. Dean, Andrzej Lingas and Jörg-R. Sack, August 1987.

SCS-TR-118 **Stochastic Rendezvous Network Performance - Fast, First-Order Approximations**
J.E. Neilson, C.M. Woodside, J.W. Miernik, D.C. Petriu, August 1987.

SCS-TR-120 **Searching on Alphanumeric Keys Using Local Balanced Trie Hashing**
E.J. Otoo, August 1987.