1998 International Computer Symposium
Workshop on Computer Architecture
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

A New Scheme to Reducing Data Stall with Data Prefetching Table and History Table

Lung-Hsiung Wang, Yen-Hsin Wang, and Jih-Fu Tu

Department of Computer Science and Engineering,
Tatung Institute of Technology, Taipei, Taiwan, R.O.C.
Email: {lhwang, yhwang,d8506002}@cseserv.cse.ttit.edu.tw

ABSTRACT

Large scale programs have been developed that has more frequent data access from memory, thus incurring data hazard and data access latency. This often degrades CPU performance, and more seriously, the system may stall. This paper proposes a new scheme to solve the data dependence and to avoid the data hazard. Two tables are added into the DLX pipeline architecture: one is Data Prefetching Table (DPT), and the other is History Table (HT). The HT is used to process the instruction recognition for data dependency detection. If data dependency occurs, the system will immediately send a "reusing" signal to DPT, noticing DPT to deliver the deeded data to ALU. This eliminates the data access latency and expedites CPU. Based on this, the Data Prefetching Processor (DPP), i.e., the pipeline CPU with DPT and HT, is modeled and simulated using the SES/*workbench* object-oriented graphical modeling and simulation software. Performance comparison between the enhanced structure and traditional pipeline architecture is done to verify the suitability of our proposed scheme.

Keywords: data prefetching, Data Prefetching Processor (DPP), pipeline architecture, SES/*workbench*, and data stall.

1. INTRODUCTION

Upon to date, CPU is still the most important element of all kind of computers. Whereas, how to speed up a CPU and improve the cost/performance rate is still the major concern of computer designers and all users. Basically a powerful CPU requires higher hit ratio, lower miss penalty, lower data access stall, higher speed, lower cost, etc.

Penalty or stall in pipeline architecture is the result of non-optimal hardware/software designs of architecture. Recently, large-scale programs have been developed that need larger memory space and have more frequent operation access from memory. These programs are highly data dependent that could cause data hazard and access latency, or degrade CPU's performance, or even worse make system stall.

The following techniques may be used to improve CPU's speed and performance: (1) improving CPU's internal structure, (2) expanding memory bandwidth and memory size, (3) increasing cache size. Many researches have discussed these in great detail. CPU structure improvement had been discussed in [1], memory issues had been talked in [2,3], and cache performance improvement had been discussed in [4,5]. A great majority of articles only dealt with improvement of some features of the processor and aimed at the instruction prefetch. Although data dependence had been discussed in [6,7,8], it didn't go very far in discussing data prefetch. Moreover, only a few of them discussed reasons of the data penalty. Thus, there still has space for improvement in processor design.

A number of techniques exist to improve CPU's performances that are doing cache prefetching. The idea of prefetching is to predict data access needs in advance so that a specific piece of data is loaded from the main memory before it is actually needed by the application. Many papers have been issued for data prefetching study, such as [9] focuses on instruction prefetching to reduce the memory access penalty during instruction phase. [10] Uses the victim cache to reduce the data stalls. Some hardware prefetching works use one-block-lookahead(OBL) scheme for prefetching cache lines, stream buffers, stream cache, or stride predication table (SPT). A number of techniques also exist that did software. A technique for prefetching certain types of array data was proposed by [11]. The most practical software prefetch scheme is available in [12].

Many early researches have devoted to raising CPU's performance using techniques such as scoreboard, VLIW or reservation table. Those schemes cannot effectively solve the access penalty, and are often complex and costly. In reality, two consecutive instructions may have data dependency and the CPI accrue.

Our purpose in this study is to provide a new scheme to solve the data dependence and to reduce the access latency. We will add two tables into the DLX pipeline architecture:

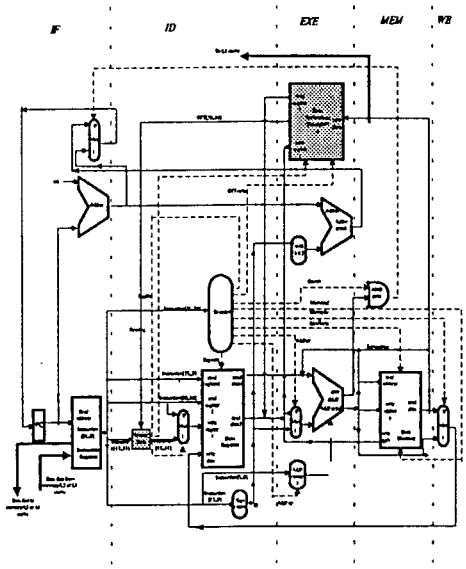

Fig. 1. The DPP architecture.

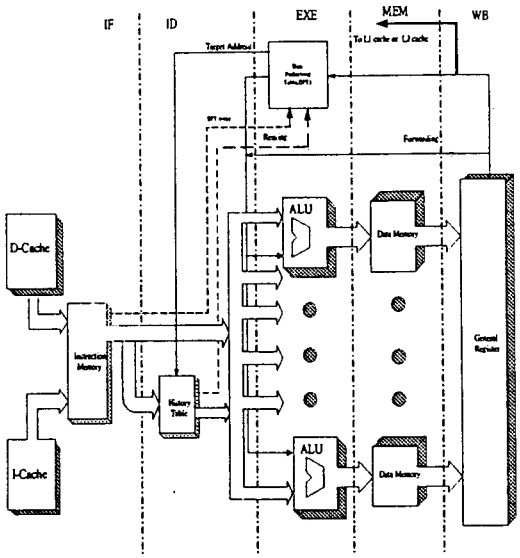

Fig. 2. The relative of HT and DPT.

One is Data Prefetching Table it has a 64-entry register used to store the previous execution result, the other is History Table, a 16-entry FIFO buffer used to store the target address from DPT. When two instructions have data dependency, i.e., the successive instruction and one of the previous 15 instructions use the same data, the system uses HT to verify and sends a "reusing" signal to DPT. The successive instruction needed data has been stored in DPT, and to be delivered from DPT to ALU's.

By adding few hardware into the DLX pipeline CPU, we attempt using HT and DPT to reduce the latency and speed up CPU, and to improve the system performance. We assemble the DLX pipeline processor, DPT and HT to form a Data Prefetching Processor (DPP). The DPP's has five pipeline stages: Instruction Fetch (IF), Instruction Decode (ID), Execute (EXE), Memory (MEM), and Write Back (WB). Note that the original DLX pipeline and our DPP have the same data and instruction flow path. In this paper, it is our intention to use the data prefetch scheme to reduce data latency. In this paper we uses the R-type instruction to verify our new scheme. Simulation of DPP is done in SES/*workbench* [13], an object-orient graphical simulation environment.

The rest of this paper is organized as follows. Section II presents the organization of DPP and dedicates an illustration of how data prefetching process on a pipeline system can be achieved. Section III further describes the HT and DPT. Section IV defines our simulation model and discusses the simulation result. Finally, the conclusion is made in Section V.

2. DATA PREFETCHING PROGRAM AND DATA PREFETCHING SCHEME

Fig. 1 illustrates the architecture of Data Prefetching Processor (DPP) evolved from DLX pipeline. As can be seen, we use hardware technique to achieve the data prefetcher. Fig. 2 illustrates the HT and DPT is placed in the ID and EXE stage respectively. We now proceed to describe the data prefetching control scheme, the details of HT and DPT structure leaf to the next section.

From Fig. 1, an instruction feed to IR then divides into two ports: the Opcode (the highest 6 bits) and operand stream (the other 26 bits). The Opcode stream flows to the decoder which can be decoded to several control signals to control ALU_write, ALU_read, DPT_write, Register_Write and Register_Read, etc. The source operand (10 bits) of the operand stream goes to HT. We use HT to recognize and compare the source address of new instruction and the target address of previously instruction. If they match, a "reusing" signal will be sent from HT to DPT. Upon receiving the "reusing" signal, DPT immediately delivers previous store data to ALU.

For an I-type instruction, or called reference memory instruction such as LOAD/STORE instruction, if it is decoded, the decoder will send a "DPT_write" signal to DPT and store the immediate value into DPT's target value field. If one of the two source operands is equal to DPT's target address field data. Then, a "reusing" signal will be sent to DPT from HT to complete the data prefetching process. The DPT will send the assigned entry data to data register of ALU in the ID stage .The reusing process in sum has two steps:

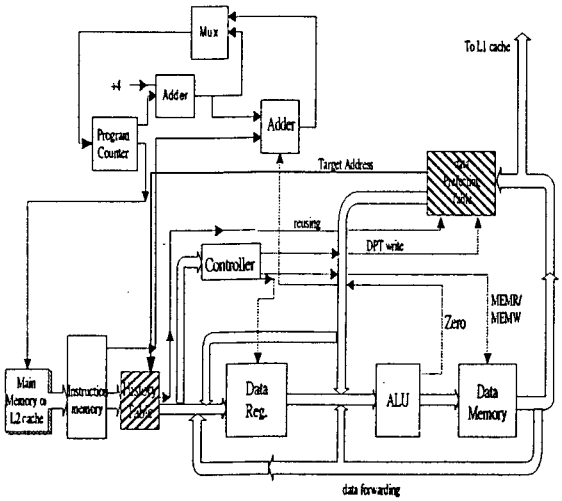

Fig. 3. Data and Control Flow Between HT and DPT.

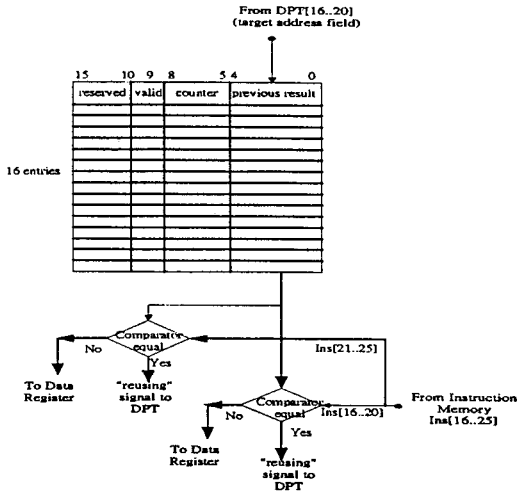

Fig. 4. The structure of HT.

.1) The control signal path (involving reusing signal):

DPT_{busy} ← History Table, i.e., a one bit reusing signal is transferred from HT to DPT busy field

2) The prefetch data path:

ALU_{input} ← $DPT_{target\ value}$, i.e., DPT's target value is transferred to the ALU's input port

3. THE HT AND DPT STRUCTION

Fig. 3 shows the relationship between HT and DPT. The comparison unit (HT) can detect and predict whether the successor instruction reuses the same data of predecessor instruction. If this is true, it will send a "reusing" signal to DPT. This procedure not only achieves data prefetching but also delivers correct data to ALU.

3.1 The History Table (HT)

The HT acts as a recognizing and comparison unit, illustrating in. Fig. 4. It is a 16-entry buffer that can record 16 instructions to recognize the target address of previous instruction and the source address of the new instruction data address. Here, we separate the instruction into two main parts, and define them as follows:

1) Each Opcode uses 6 bits, which is located at the significant positions (bit 26~ bit 31).

2) The other 26 bits (bit 0 ~ bit 25) are used for operands; they are first source, second source, target address, shift and ALUop.

Thus, we can use HT to separate the Opcode and operand of each instruction. The Opcode of instruction streaming from HT delivers to the controller unit. When the instruction streams to HT, the data stream may send to Data Register concurrently, the Opcode and operand can concurrently deliver in the same phase of CPU. Before delivered to ALU, the source operand of new instruction is compare with the previous target operand reserved in HT. If they match a "reusing" signal will be sent to DPT, and the previously executed result from DPT transport to ALU for reuse; otherwise, the new source data is directly delivered to data register.

For an R-type instruction the source 1 field is located in *Ins*[25..21], the source 2 field is located in *Ins*[20..16], the target field is located in *Ins*[11..15]. When a new R-type instruction is accessed, the HT will compare the target field of previous instruction with the two source fields of new instruction. The successive instruction and the previous one, both use the same operand that results in the data dependency.

3.2 Data Prefetching Table(DPT)

Fig. 5 illustrate the DPT, It similar a data cache, in that data may come from data register if it is a new load data or, come from data memory if it updates or modifies a executable operand. If the new instruction and prior instruction both use the same address, the data is obtained from DPT in the EXE to complete the data prefetching.

TABLE I
DPT ENTRY BIT DEFINITION.

Field name	Bits number	Description
Reserved	3 (29-31)	No using
Busy	1 (28)	busy = "1" means the data has been reused
Valid	1 (27)	valid="1" means counter value is less than 64
Counter	6 (21-26)	Each instruction can stay in DPT 64 clocks long
Target address	5 (16-20)	To storing the address, parameter, or register name
Target value	16 (0-15)	The latest results reused by a new instruction

If an operand is reused more than two times, the reusing signal has to flash again. The counter may be reset to zero and the busy bit will remain "1". If an operand stays in DPT over 64 clocks and no reusing occurs, it means the data are not valid. Then it's occupied entry may be replaced by over-write method for a latter instruction's result. The result of any instruction after executed by ALU would be store in L1 cache and also DPT concurrently. If the count value of DPT exceed 64, the result data will be overwritten by a successive result data. The DPT is similar to a cache storage, which has 64-entry. Each entry has 32 bits to record the operation result and status of an instruction. Table I show each entry's bit definition of the DPT.

4. SIMULATION MODEL AND RESULT ANALYSIS

For a quantitative analysis, we use an object-orient graphical simulations software SES/*workbench*, for model construction and simulation. Figure 6 shows the model of the data prefeching system in SES/*workbench* modeling. It is a hierarchical directed graphical model with eight sub-models they are reference models to PC ALU, HT, DPT, ...,and Register. Each reference model, i.e., sub-model, has similar from of such hierarchical directed graph and can be travailed down once it is called (i.e. reference) and returned when process is done. Here, each sub-model is carefully. We are simulations in SES/workbench. The simulation period is 10 (from 0 to 10), number of events since last reset is 112. The metric has: With/without DPP, category performance comparison for six parameters (MEAN, minimum, maximum, variance, stdev, ending value),

Fig. 7 shows the variance value of each category under different SD and MD, the x-axis indicate the execution time by exponent numerical, the y-axis indicate category variance.

Fig. 7(a) shows the reusing and *Ins_0_25* category has the highly Maximum value, it means reusing and

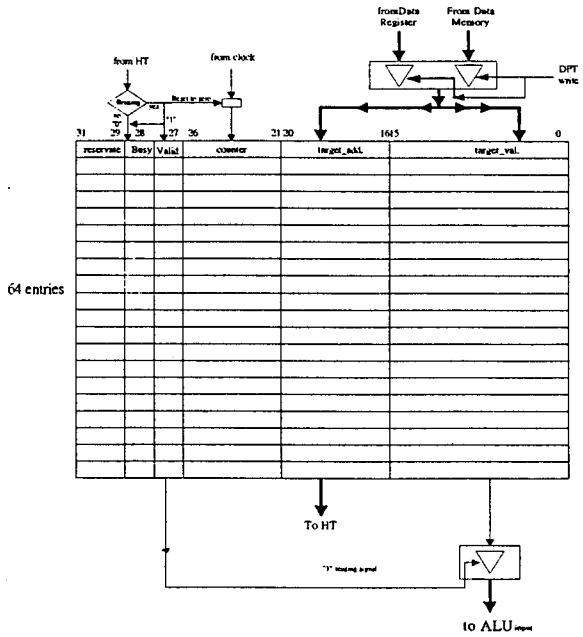

Fig. 5. The Structure of DPT.

Ins_0_25 category with high utilization, because HT is a instruction recognize unit. Fig. 7(b) shows the prefeching category has highly execution time DPT sub-model, it means a data prefetch operation in DPT sub-model frequency, and the prefetching function is available. Fig. 7 (c) shows the ALU sub-model execution results for different replication, when a category related to HT or DPT, example *Ins_0_25*, prefetching, reusing, it will has highly execution time.

We also compare the throughput of DPP and tradition pipeline architecture. Both of they are running in the same condition, example, 10 replications 1 batch, and processor execution time of each sub-model. Fig. 8(a) shows the DPP architecture has higher total through about 90% utilization of HT and ALU, either the DPT has 50%. We compare to traditional pipeline architecture, to see Figure 8(b). We compare Fig. 8(a) and Fig. 8(b), the enhanced architecture has high throughput to traditional pipeline, for HT is 20%, and for ALU about 90%.

From Fig. 8, we understand the throughput in lower replication the tradition pipe architecture may prior to the DPP architecture, this reason due to no more reuse require, and in more times replicate cycle, the system may occur highly reuse require.

5. CONCLUSION

How to design a powerful CPU architecture is decided to your initialization idea. Whether the system utilizes or not, to progress simulation before you want implement is very import.

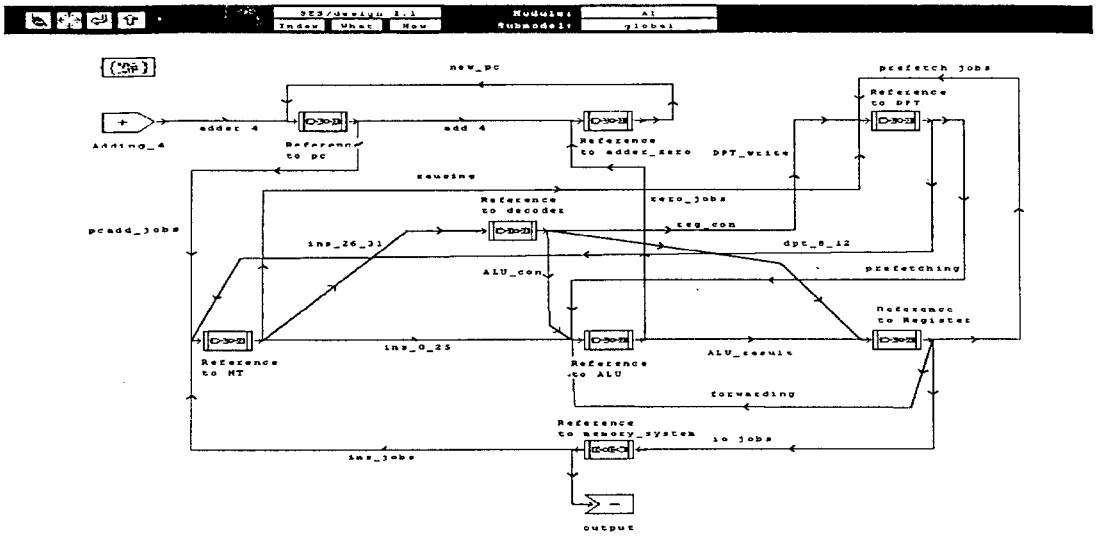

Fig. 6. The Simulation model of SES/*workbench*.

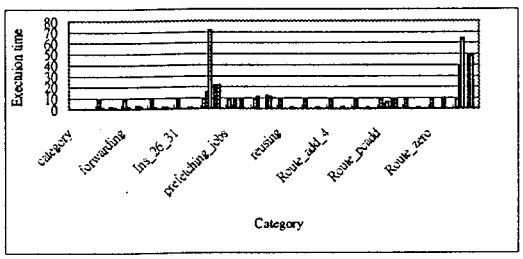

Fig. 7(a). In SD=HT utilization.

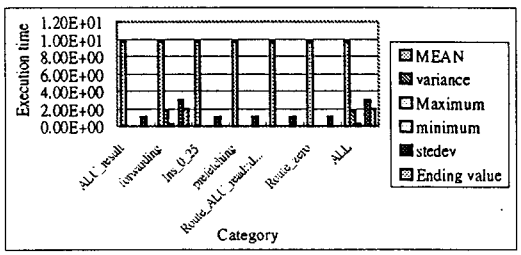

Fig. 7 (b). In SD = DPT utilization.

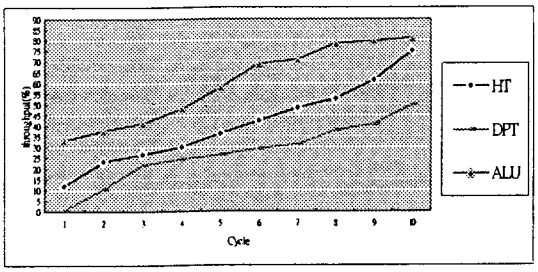

Fig. 8(a). DPP organize throughput.

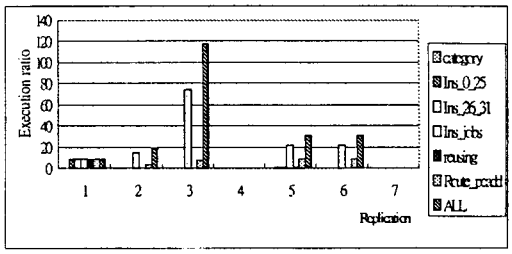

Fig. 7(c). In SD=ALU utilization.

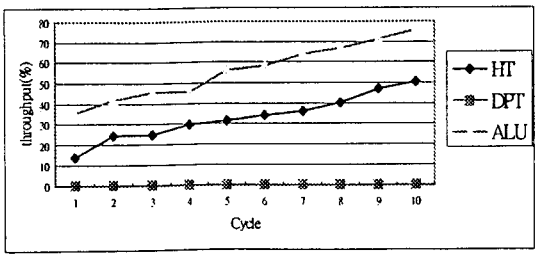

Fig. 8(b). Traditional pipeline throughput.

TABLE II

THE SUB-MODEL OF SIMULATION.

Name	Describe
PC	Program Computer
Adder-zero	New PC calculate
DPT	Date Prefetching Table
HT	History Table
Decoder	Operand decode unit
ALU	ALU unit of CPU
Register	Data register
Memory-system	L1 data cache has 512 k

We only add little hardware to the tradition of pipeline architecture to complete our assumption. Modeling a system to architects evaluate its performance for a set of design assumptions when chosen workload, the system bottlenecks are easy resolute.

We apply this DPP scheme to reduce data access stall, and through modeling simulation method created in SES/*workbench* to prove our assumption. We compare the new scheme processor and tradition one via the simulation result, and provide our assumption is executive. The enhanced architecture has speed up the ALU unit of CPU compare to tradition pipeline architecture.

REFERENCES

[1] J. L. Hennessy and D. A. Paterson, *Computer Architecture - A Quantitative Approach,* Morgan Kanfman, Ver.2, 1995.

[2] Stephan Olariu, and Albert Y. Zomaya, " A Time- and Cost- Optimal Algorithm for Interlocking Sets-With Applications," *IEEE Transactions on Parallel and Distribute System*, Vol. 7, No. 10, Oct. 1996, pp.1009-1025.

[3] Subbarao Palacharla and R.E.Kessler, "Evaluating stream buffers as a secondary cache replacement," *In proc. of the 21st annual International Symposium on Computer Architecture*, April 1994, pp. 24-33.

[4] Pei.Cao and Edward W. Feleten Kai Li, "Implementation and Performance of Integrated Application-Controlled File Caching, Prefetching, and Disk Scheduling," *IEEE Micro*, Vol. 14, No. 4, Nov. 1996, pp.124-135.

[5] James E. Bennett, Michael J. Flynn, "Reducing Cache Miss Rates Using Prediction Caches, " Technical Report No. CSL-TR-96-707, pp.1-18.

[6] John W.C. Fu and Janak H. Patel, "Data prefetching in Multiprocessor vector cache memories," *In Proc. of the 18th Annual International Symposium on Computer Architecture*, May 1991, pp. 54-63.

[7] T. Mowry, M.Lam, and A. Gupta, "Design and Evaluation of a Computer Algorithm for prefetching," In SIGPLAN Notices, September 1992, pp. 62-73,

[8] Shlomit S. Pinter and Adi Yoaz, "Tango: A Hardware-based Data Prefetching Technique For Superscalar Processors," *In Proc. of the 25th Annual International Symposium on Computer Architecture*, May 1997, p.p.214-225.

[9] A.K. Porterfield, "Software Methods for Improvement of Cache Performance on Supersomputer Applications," *Technical Report COMP TR 89-93, Rice University*, May 1989.

[10] Dimtrios Stiliadis and Anujan Varma, "Selective Victim Caching: A Method to Improve the Performance of Direct-Mapped Caches," *IEEE Transactions on Computers*, Vol. 46, No. 5, May 1997, pp. 603 – 609.

[11] Tse-Yu and Yale N. Patt, "Alternative Implementations of Two level Adaptive Branch Predictions," *In proceedings of the 19th Annual International Symposium of Computer Architecture*, Gold Coast, Australia, May 1992, pp. 124-134.

[12] Tse-Yu and Yale N. Patt, "A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History," *In proceedings of the 20th Annual International Symposium of Computer Architecture,* San Diego, CA, May 1993, pp. 257-266.

[13] Scientific and Engineering Software, inc., *SES/workbench User's Manual*, Release 2.1 Scientific and Engineering Software Austin, TX, USA, February 1992.