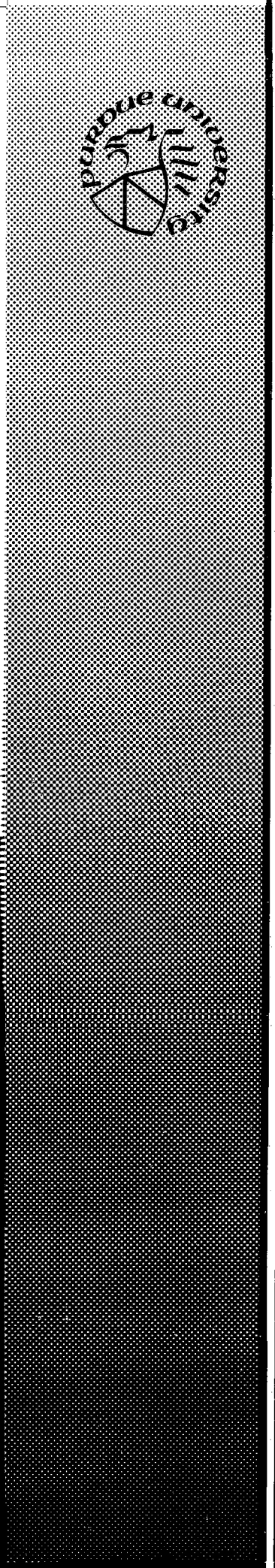
12-1-1985

A Comparison of E/D-MESFET Gallium Arsenide and CMOS Silicon for VLSI Processor Design

Mark K. Bettinger
Purdue University



A Comparison of E/D-MESFET Gallium Arsenide and CMOS Silicon for VLSI Processor Design

Mark K. Bettinger

TR-EE 85-18
December 1985

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

A COMPARISON OF E/D-MESFET GALLIUM ARSENIDE AND CMOS SILICON FOR VLSI PROCESSOR DESIGN

Mark K. Bettinger

TR-EE 85-18

December 1985

ACKNOWLEDGMENTS

I acknowledge the guidance and assistance of my major professor, Veljko Milutinović. He has provided opportunities to learn that I appreciate. I am also indebted to RCA-ATL for their support and guidance as well as their funding. I would also like to thank those at RCA-ATL who have provided assistance: Tom Geigel, Bill Heagerty, Walt Helbig, Wayne Moyers, Jeff Pridmore, and Rich Zeigert.

Little of this work would have been completed without the support of my officemates. I also thank my good friend Lee Bissonette for all of her cheerful assistance. Finally, I am grateful to my family and especially my wife, Tammy, for their encouragement and support throughout all the last minute late nights. Without Tammy's help and encouragement this work would not have been possible.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

ABSTRACT

Bettinger, Mark K. MSEE, Purdue University. December 1985. A Comparison of E/D-MESFET Gallium Arsenide and CMOS Silicon for VLSI Processor Design Major Professor: Veljko Milutinovic.

Gallium Arsenide (GaAs) circuits have long been known for their speed. They are now being considered for single chip processors since GaAs chips are reaching VLSI complexities. Design constraints that affect both system and processor design accompany the new technology. The goal of this work is to compare and contrast designs in GaAs-E/D MESFET and Si-CMOS technologies as they apply to ALU design. These differences are emphasized by examining the design of several structures in GaAs for implementation of Stanford University's MIPS processor in GaAs. The three topics discussed are adder design, multiplier placement and design, and cache effects on multiplier design. The comparisons were made to help optimize the design of 32-bit GaAs microprocessor for RCA. The results show that the high speed of GaAs devices allows serial rather than parallel implementation of structures in GaAs; these serial structures use less area than their parallel counterparts without any degradation of performance. The total reduction in area is necessary to compensate for the area used by large fanin and fanout structures. In addition, any solutions proposed for each structure must also take into account the long off-chip delays.

CHAPTER 1
INTRODUCTION

Device physicists and circuit designers have long been interested in gallium arsenide (GaAs) technology. As GaAs technology matured, it caught the attention of digital circuit designers. Recently, the speed of GaAs devices has captured the serious interest of system designers; they see the potential for order of magnitude increases in computer system performance over silicon systems becoming a reality.

Although speed is a major advantage, it is not the sole reason that computer architects are taking a closer look at designing systems in GaAs. GaAs fabrication techniques have made significant advances in recent years and have allowed GaAs chips to reach VLSI levels of complexity. As designers become more aware of the advantages and disadvantages of the new technology, they are realizing that their old design methodologies are based on the properties of silicon. Therefore, they are searching for new design methodologies that take advantage of the different design parameters of GaAs. Lack of design experience and unfamiliarity with the advantages and disadvantages of GaAs technology and how they relate to silicon technology are obstacles preventing full exploitation of the characteristics of GaAs.

To help overcome these obstacles, knowledge of the advantages and disadvantages of GaAs and how they relate to the design methodology must be increased. To accomplish this goal, work is being conducted to study issues relevant to the design of a 32-bit GaAs processor. As an early participant in the design of GaAs processors [HeScZ85], RCA corporation selected Purdue University as a partner to study several of these issues. The goal of the study is to experimentally determine the effect of the silicon/GaAs differences on possible designs. The results can then be used as guidelines for a set of design solutions which use the advantages of GaAs while minimizing the disadvantages of GaAs. These results were also used by RCA to help optimize the design of a 32-bit GaAs microprocessor.

This portion of the study presents three sets of experiments covering three different aspects of the design issues as they relate to ALU design in a wide

sense. Arithmetic and logic unit design, and multiplier design and related issues are the issues discussed in this thesis. Other processor issues such as pipelining, instruction format, and register file design are covered in another thesis [Fura85].

The remainder of this introductory chapter will provide a brief perspective on the differences between GaAs and silicon, the logic families used in GaAs and their characteristics, and a comparison between the GaAs and silicon design parameters.

1.1. GaAs Advantages and Disadvantages

The advantages and disadvantages of using GaAs technology in computer systems are the result of inherent differences between current silicon and GaAs technologies. The differences are attributed to two major advantages and disadvantages of GaAs compared with silicon. The advantages of GaAs technology are a higher resistance to adverse environmental conditions and faster switching speeds than silicon. The disadvantages of GaAs technology are higher cost and lower transistor count per chip than silicon.

The advantages are the result of the physics of the materials. Because of the short minority carrier lifetime, GaAs devices have a high resistance to radiation and can withstand dosages of 10-100 million RADS compared to 5-6 thousand RADS for silicon devices [Heage85]. The large bandgap allows a range of operating temperatures from -200 and +200 degrees centigrade [EdLiW83] as opposed to silicon CMOS operating temperatures of -55 to +125 degrees centigrade. These two facts alone make GaAs very promising for both military and aerospace applications where extreme operating conditions are the rule rather than the exception.

GaAs also performs better than silicon devices when comparing switching speeds [EdLiW83]. The larger mobility and peak velocity of carriers allows GaAs gates to switch faster than silicon Transistor-Transistor Logic (TTL) gates by up to an order of magnitude. In addition to the switching speed advantage, GaAs gates also consume less power than the fastest silicon gates. The power consumed is an order of magnitude lower than that of the fastest silicon logic, Emitter-Coupled Logic (ECL), yet the gates remain faster than silicon ECL gates. This power speed product makes GaAs even more attractive for high performance computers.

GaAs technologies unfortunately have tremendous manufacturing costs: two orders of magnitude more expensive than their silicon counterparts. There are several reasons for the high cost and only five of these will be mentioned

here. One of the basic reasons is the scarcity of gallium. Silicon makes up a large percentage of the earth's crust while gallium is a rare metal. Second, since GaAs is a composite material, the additional processing needed to create the compound and verify its composition [Namor84] increases the cost. Third, because of its structure, GaAs is also characterized by a higher dislocation density than silicon [Walle84]. This results in a poorer yield for GaAs fabrication processes. The fourth factor that raises the cost of GaAs is the sophisticated processing techniques required to produce working GaAs transistors. One example is the uniformity of threshold voltages that must be maintained. Voltage swings for some GaAs devices are as low as 0.5 volts and require that threshold voltages be uniform to within a very narrow range. This narrow range also adds to the fifth factor: design costs. Digital designers are discouraged from using NAND logic because they cannot use multiple input ($>$ 2) NAND gates because each additional input requires a significant amount of additional voltage swing and the total voltage required quickly passes the 0.5 volt limit. Designers are also limited by the low fanin and fanout maximums of the GaAs E/D-MESFET technology. Although these problems exist, the solutions are currently being pursued. By the end of the decade, solutions will be found which should reduce the cost difference to only one order of magnitude [Namor84].

The second disadvantage of GaAs processes, their low transistor count, is limited by two factors. Initially, the high defect density required that chips be manufactured with a small area and corresponding low transistor count to achieve adequate yields. As the quality of GaAs materials and processing improved, larger and more complex chips became possible. An increase in power that strained the limits of available heat dissipation techniques followed the increase in complexity. Although new techniques were developed to improve the heat dissipation, the added power increased the number of potential reliability problems. Currently, the restrictions on transistor count are power dissipation and defect density.

Although the problems present in the GaAs environment may be resolved, it is believed that none of these four GaAs-Si advantages and disadvantages are temporary in nature; they result from inherent differences between GaAs and silicon materials [Coope84b]. Therefore, conclusions that are based on these four fundamental characteristics will remain valid as GaAs technology matures.

1.2. Design Methodology Changes

Because the differences between the GaAs and the silicon environment are not trivial in nature and affect some of the fundamental design decisions, merely copying existing silicon designs into the GaAs environment will not obtain the most robust GaAs performance. The new environment sets up a new set of rules and challenges. Although the hurdles that must be leaped to meet these challenges are higher, the rewards for successfully exploiting this technology are also greater. With the proven high speeds of GaAs circuits and the VLSI integration levels that are now appearing, we are on the verge of achieving single chip processors capable of speeds for scalar operations approaching that of present-day supercomputers [Hwang84].

To achieve the performance expected from GaAs, the design must be considered from more than one viewpoint. The differences can be looked at from a processor standpoint. How the GaAs technology affects the internal processor logic becomes important. The transistor count limits how many and what functions can be placed on the chip. The device limitations determine how structures such as the ALU can be designed and which structures are possible in the GaAs environment.

The design can also be considered from a system standpoint. How device delays affect offchip communication delays becomes important. The device delays determine what structures can be placed offchip without adversely degrading system performance and the communication of these structures with the system. The transistor count limits the complexity of the support chips and the number of support functions that can be efficiently used by the system.

1.3. Overview of Thesis

The purpose of this thesis is to relate these new design parameters to specific applications. The parameters will be examined to see how they affect design both from the system standpoint and from the processor standpoint in relation to CMOS and GaAs designs. Tradeoffs which must be made when going from CMOS designs to GaAs designs, and the impact of technology on the designs will be examined. There have already been papers on the general problems of GaAs from both standpoints [MiSiF86][MiFuH86]. This thesis will cover details of the differences as they apply to specific problems. The effect of the technology on the design of the processor's adder will be covered to acquaint the reader with processor design problems. ALU design is covered in a broad sense by examining possible multiplier choices and tradeoffs between multipliers and barrel shifters. These choices cover placement as well as

complexity and structure. Both adder and multiplier performance can be affected by system issues such as cache design. Therefore the impacts of the technology change on cache design and how this affects adder and multiplier design will also be examined. GaAs designs in each of these three areas will be contrasted with similar CMOS designs to provide the basis for comparison of the technology generated differences.

Chapter two begins the comparison by examining the differences between GaAs and silicon relevant to these structures (adders, multipliers, cache) regardless of application. Chapter three describes the tools used for the evaluation and the evaluation methodology. Chapters four through six describe in detail the experiments done to evaluate the GaAs/silicon differences in designs. Chapter seven summarizes the results of the experiments in the previous three chapters.

CHAPTER 2
TECHNOLOGY AND IC DESIGN

As Si technology matured, it went through major changes. Similarly, GaAs technology is still developing and maturing. This growth is accompanied by the development of new logic families such as High Electron Mobility Transistors (HEMT), Hetero-Junction Bipolar Transistors (HJBT), and Enhancement mode MEtal Semiconductor FETs (E-MESFET). These families are an addition to existing families such as IGFETs and JFETs, and the earliest family, Depletion mode MESFETs.

As GaAs IC development matured, the level of integration of each family also increased. The family chosen for implementation of single chip processors and their support chips such as those discussed here must be able to support VLSI levels of integration. The MESFET families have reached a level of integration higher than any other logic family. Other families have achieved better performance than MESFETs but no others have achieved VLSI levels of integration. Therefore, we have confined our work to the GaAs MESFET family.

Some of this information is also presented in [MiFuH86]. This chapter contains an update of this information and additional material dealing with E/D-MESFET technology is presented here. The material is presented here as an aid to the reader.

2.1. GaAs MESFET Logic Families

Both depletion-mode MESFETs (D-MESFETs) and enhancement mode MESFETs (E-MESFETs) have been used to build MESFET logic circuits in GaAs. The depletion mode devices are generally considered better than the enhancement mode devices because of several important differences. D-MESFETs have better noise immunity, have fewer fabrication problems, are less sensitive to increases in fanin and fanout, and are generally faster than E-MESFETs. Their disadvantages are that D-MESFETs require a second power supply and extra logic to provide level shifting. E-MESFET designs with their low area requirements are often used because they require less power and less

complex circuit design than D-MESFETs [EdWeZ79]. In addition, E-MESFET designs are not saddled with the disadvantages of D-MESFET designs.

Three principal GaAs MESFET logic families are in use today. They are Buffered FET Logic (BFL) and Schottky Diode FET Logic (SDFL) of the D-MESFET family, and Direct Coupled FET Logic (DCFL) of the E-MESFET family [NuPeB82].

The earliest work in GaAs digital circuits was done with BFL D-MESFETs [VanLi74]. The disadvantages of these gates are their requirement for much power and area, and their need for two power supplies and voltage shifting logic. These disadvantages are compensated for by the large fanout capabilities and large noise margin of BFL gates. In an effort to reduce power requirements, low-power BFL (LPBFL) circuits were introduced.

Early BFL gates were characterized by propagation delays of 34 ps with power dissipation of 41.0 mw/gate [NuPe82]. The LPBFL gates reduced power dissipation to 6.0 mw/gate with delays of 250 ps as part of a 40 gate 4-bit ripple carry adder [PeDaN83]. A more recent LPBFL design used 420 gates to implement a 32-bit adder with gate delays of 230 ps dissipating only 2.8 mw/gate [YaHiA83]. The most advanced BFL design is a 12x12 bit multiplier implemented with only 1083 gates [FuTaI84]. The gate delays were only 170 ps and each gate dissipated 1.7mw.

SDFL D-MESFET gates are a low power alternative to BFL logic gates since SDFL gates generally require less power and area than BFL gates. Because of the reduction in area, this family has received considerable interest for LSI circuit applications [EdWeZ79][NuPeB82] and was the first to reach LSI levels of integration. Despite the improvements, SDFL gates still need two power supplies and level shifting logic.

One of the first LSI GaAs applications was an 8x8 bit multiplier containing 1008 gates using SDFL logic [LeKaW82]. Gate delays were 150 ps and power dissipation was only 1.5 mw/gate. More recently, an SDFL RAM/gate array chip with 8000 devices has been built [VuRoN84]. Approximately 3000 FETs and 5000 diodes were used to create a chip with 432 programmable SDFL cells, 64 bits of RAM, and 32 interface cells. Although the complexity had increased over previous designs, the speed and power remained the same. Low-power predictions for the same design suggest that gate delays of 300 ps and power levels of 0.2 mw/gate are possible.

Although E-MESFETs lack some of the advantages of D-MESFETs, they are considered suitable for VLSI implementations because of their low power requirements and simpler circuit designs. They have long been considered

suitable for VLSI implementation [NuPeB82]. Their suitability is enhanced because they require no logic level shifting logic and only a single power supply. The most common E-MESFET logic gate is the Enhancement mode driver/ Depletion mode load MESFET (E/D-MESFET). Since E/D-MESFETs are more difficult to fabricate than the previous two logic families, it was not until recent advances in GaAs fabrication technology were made that this family was able to reach VLSI levels of integration and began to dominate GaAs digital designs.

A number of designs have been introduced using DCFL E/D MESFET logic gates. A 1000 gate gate array has been designed with loadless gate delays of 100 ps and power dissipation of 0.2 mw/gate [IkToM84]. This was proof that DCFL circuits could switch faster while using less power than SDFL circuits. A slightly larger 16x16 bit parallel multiplier containing 3168 gates with gate delays of 150 ps at 0.3 mw/gate has also been reported [NaSuS83]. The highest level of integration reported so far for any GaAs process is a 16kx1 SRAM containing 102,300 devices with an access time of 4.1 ns and power dissipation of 2.5 w [IsInI84].

2.2. Characteristics of GaAs MESFET Designs

Table 1. is presented here to compare the performance characteristics of some GaAs MESFET designs. Based on the power and complexity of the chips presented in the table, GaAs technology is clearly becoming a suitable vehicle for microprocessor implementations. Based on its low power requirements, the most promising MESFET solution is provided by the DCFL E/D-MESFET approach. The merits of DCFL are demonstrated by the SRAM which provides a high level of integration, 102,300 FETs, with low power, 2.5 watts [IsInI84]. The current major drawback to this approach is its fabrication complexity and resulting low yield and high cost. Based on the present rate of fabrication technology improvement, however, the introduction of GaAs microprocessors should occur within the next two or three years. We must, therefore, begin to understand the GaAs environment now and determine how the characteristics of GaAs will influence processor and system architecture design in the new environment. We will then be ready as it steps to the forefront of high-speed, environmentally adverse applications.

Table 1. Performance Characteristics of GaAs Designs [Fura85].

Unit	Speed (ns)	Power (W)	Device Count	Reference
ARITHMETIC				
32-bit adder(BFL)	2.9 total	1.2W	2.5K	[YaHiA83]
8x8 multiplier (SDFL)	5.2 total	2.2W	6.0K	[LeKaW82]
16x16 multiplier (DCFL)	10.5 total	1.0W	10.0K	[NaSuS83]
CONTROL				
gate array/SRAM (SDFL)	0.15/gate	3.0W	8.0K	[VuRoN84]
1000-gate gate (DCFL)	0.10/gate	0.4W	3.0K	[IkToM84]
MEMORY				
1K bit SRAM (DCFL)	2.0 total	0.5W	7.1K	[AsKuH83]
4K bit SRAM (DCFL)	2.8 total	1.2W	26.6K	[HiInM84]
16K bit SRAM (DCFL)	4.1 total	2.5W	102.3K	[IsInI84]

2.3. GaAs/Silicon Comparison

Since DCFL has been shown to have a sufficient level of integration with acceptable speed and power, all discussion of GaAs circuits will be based on the DCFL E/D MESFET logic family. All of our Si discussion will be based on CMOS/SOS (silicon on sapphire).

Table 2 [BasNe84] compares several characteristics of GaAs and Si technologies that are important for processor and system design and optimization. Significant GaAs/Si differences can be observed in four areas: (1) transistor count per chip, (2) on-chip gate delay, (3) the ratio of off-chip to on-chip memory access time, and (4) gate fanin and fanout. The rest of this section will concentrate on the GaAs/Si differences from Table 2. The implications of these differences on processor and system design will be presented in the following sections.

The differences between GaAs and Si are substantial and will affect processor and system designs. The next section will discuss the implications of GaAs characteristics on aspects of specific design issues.

Table 2. Performance Comparison of E/D-MESFET GaAs, CMOS/SOS, and Bulk Silicon.

	GaAs	CMOS/SOS	CMOS/BULK
COMPLEXITY			
Transistor Count/Chip	20-30K	150 K plus	150 K plus
Chip Area	yield & power dependent	yield & power dependent	yield & power dependent
SPEED			
Gate Delay	50-150 ps	.8-1.5 ns	1-3 ns
On-chip Memory Access	0.5-2.0 ns	10-20 ns	20-40 ns
Off-chip/On-package Memory Access	4-8 ns	30-40 ns	40-80 ns
Off-chip/Off-package Memory Access	10-60 ns	60-100 ns	100-200 ns
IC DESIGN			
Transistors/Gate	1 + fanin	2*fanin	2*fanin
Transistors/Memory Cell			
Static	6	5-6	5-6
Dynamic	1	n/a	n/a
Fanin (typical transistor size)	2-3	5	5
Fanout (typical transistor size)	3-5	5	5
Gate Delay Increase for each Additional Fanout (relative to gate delay with fanout=0)	25-45%	25-40%	20-30%

CHAPTER 3
EVALUATION TOOLS AND METHODOLOGY

For each of the three areas of interest (ALU, multiplier, and cache effects on multiplier design). Experiments were run to determine optimal implementations. The circuit level and instruction level simulators used required two sets of support tools for the evaluation. The circuit level simulator was used for the adder experiments and the instruction level simulator was used for the multiplier and cache experiments.

3.1. Circuit Level Simulation

The goal of the circuit level simulation was to obtain a realistic measure of circuit complexity and delay and to determine if the resulting VLSI adder implementation was fast enough and small enough to meet the limits imposed by RCA. Delay estimates for the circuits were included. The relevant aspects of this methodology, the delay estimation and the area estimation, are now discussed.

3.1.1. Area Estimation

The circuit level simulation is based on the Hardware Description Language (HDL) [TI84] design tool made available to us by RCA. A digital design could be entered into HDL using the description language. Once the circuit was properly described, usually on a gate by gate basis, the input file was compiled into a database and a simulation could be run on the compiled database. This database was also used as an input to RCA's Multi-Port 2-Dimensional (MP2D) placement and routing program that could provide layouts of the circuits described. The goal of this was to obtain a realistic measure of complexity and to determine if the area of the VLSI implementation was small enough. Rather than estimating gate counts and wiring space for an adder, these programs allowed us to obtain layout statistics for structures without having to go through fabrication.

Each design was entered into the database using HDL. Once the circuit was described on a gate by gate basis, the input file was compiled into the

database. A list of devices and connections was then extracted from the compiled data base for the layout program, MP2D. MP2D then took the lists and created a layout that could be used for fabrication of the devices. Five different word lengths were used for the layouts. The areas for the layouts were used to derive the equations used to approximate the area occupied by each adder. The equations were then entered into a C-language program which was used to approximate the area for individual adders. This program was used to calculate the area required for the 35 adders that were not described on a gate by gate basis using HDL.

3.1.2. Delay Estimation

The design delays were implemented in C-language because access to the circuit simulation software was not granted. Optimization was done in a manner analogous to hand optimization in all but the most complex design choices. Many of the choices were also fixed before optimization began. The use of the C-language programs allowed the flexibility to change any of the delay parameters at any time and see the results in the period of a few minutes rather than a few days. Accuracy compared to HDL was not lost from lack of wiring information since the circuit simulation program did not include wire delays in any delay calculations. The section on adder design shows that this was not a severe problem.

3.2. Instruction Level Simulation

The instruction level simulator was used to obtain program execution time to determine the performance of the architectures. The relevant aspects of this methodology are the workload model, the baseline architecture, and the method of translation between them.

3.2.1. Workload model

The workload model was a set of ten small benchmark programs obtained from Stanford University written in the high level language PASCAL. The following list gives their names and their functions:

(1) ack — a highly recursive program to compute Ackermann's function,
(2) bubble — a program to perform a bubble sort of 500 integers,
(3) fib — a highly recursive program to compute a Fibonacci number,

(4)	intmm	a computation heavy program to multiply two 40x40 element integer matrices,
(5)	perm	a highly recursive program to calculate all permutations of the numbers one through seven,
(6)	puzzle	an iteration heavy, computation heavy program to solve a three dimensional cube packing problem,
(7)	queen	a program to solve the eight queens problem,
(8)	quick	a program to perform a quick sort of 5000 integers,
(9)	sieve	a program which implements Erathosthenes sieve to compute the number of primes between 0 and 8190,
(10)	towers	a highly recursive program to solve the towers of hanoi problem with 18 discs.

This set of benchmarks represents a broad range of program classification, from highly recursive programs such as *towers* and *ack* to computationally intensive programs such as *puzzle* and *intmm*.

3.2.2. Architecture Simulator and Analysis

The architecture simulation tools are based on an instruction level simulator [Gross84] written by Stanford University for the SU-MIPS processor. SU-MIPS is an example of a Reduced Instruction Set Computer (RISC) architecture, and was one of the first "RISCs", preceded only by the IBM 801 [Radin83] and Berkeley RISC [Katev83]. The SU-MIPS architecture was selected by Darpa [Barne85] because its low transistor count is compatible with GaAs E/D MESFET capabilities of the near future.

To better understand the experiments, several SU-MIPS features must be explained. First, SU-MIPS uses a "delayed branching" scheme with a branch delay of one. This means that the first instruction after every branch operation is always executed. This places a burden on the compiler to find a useful instruction for the fillin slot or to insert a NOOP into the slot if no instruction can be found. Second, the data from a data load operation is not valid until after the instruction following the data load instruction. The compiler must therefore find useful instructions for the fillin slots after load instructions. Third, "instruction packing" is employed by the SU-MIPS processor. Certain SU-MIPS instructions contain two operations, one of which is always an ALU operation. The operations are executed sequentially in the time necessary for a single instruction fetch. Since not all instruction combinations may be packed, instructions may contain either one or two operations. In addition, the system

clock runs at twice the frequency of instruction fetches to subdivide each instruction cycle.

These features of SU-MIPS result from implementing the pipeline interlock mechanism in software rather than hardware. This eliminates hardware that suspends the pipeline waiting for data loads and branches. In addition to a reduced transistor count, performance increases since some instructions following a branch or load will be executed. In a conventional architecture, the processor is suspended during that time and does nothing.

I designed and helped implement the cache simulator that was used as part of the SU-MIPS simulator. The cache simulator receives both an address and data and returns the number of instruction cycles required for the access while updating the data and tag information. The cache simulator was designed to allow run-time modifications to the cache size, block size, prefetch strategy, cache miss delay, and memory access delay.

The different architectural features were studied by making appropriate modifications to the SU-MIPS simulator and cache simulation programs and then recording the benchmark execution time. These changes were necessary due to the differences between the SU-MIPS architecture and the architecture being studied. The changes were also generated by the GaAs processor implementation.

3.2.3. SU-MIPS Compiler and Translation Software

This study used a compiler, reorganizer, optimizer, and linker/loader written by Stanford. This translation software was not modified by us, and hence, was limited to translation of PASCAL source code into the instruction set of the Si based SU-MIPS instruction set. Modifications were not done because they involved understanding and rewriting the topic of a PhD thesis [Gross83]. Much of the difficulty in performing the experiments resulted from our inability to generate optimized code targeted to each of the candidate architectures. The implications of this are described in the following sections.

CHAPTER 4
CHOICE OF ADDER DESIGNS

4.1. Introduction

As previously mentioned, the differences between Si and GaAs technologies require different solutions to the same problem. Silicon technologies typically require high-speed adders to achieve high performance. This is because of the relatively small ratio of the memory access time to the data-path times. For example, the NMOS-silicon HP-FOCUS utilized a full carry look-ahead adder to satisfy its 55ns cycle time [BeDoF81].

Several adder designs for GaAs technologies are available ranging from the high-speed, large area full-carry-look-ahead adder to the low-speed, small area ripple-carry adder. Others having speeds and resource requirements between these two extremes include conditional-sum and carry-select adders [Hwang79].

When the switch from Si to GaAs technologies is made, the design changes are the direct result of three of the major Si/GaAs differences and the indirect result of the fourth. The differences as they apply to adder design are examined because the adder is an integral part of the cpu and directly affects the data-path time and performance.

The transistor count limits the complexity of any adder that is implemented. If the chip is limited to 30K transistors, then any adder that requires 10K transistors is unacceptable. Even lower limits may be established if some chip area is reserved for a large register file. The low on-chip gate delays can enhance performance by simply replacing their Si counterparts and reducing total delay. Often, implementation of high-speed adders requires high gate count designs.

The limited gate fanin and fanout affect both the area needed and the delay incurred. Single gates with high fanin and fanout must be implemented as a series of gates with low fanin and fanout which increases transistor count. Because delay is highly dependent on load capacitance, high fanout devices have a relatively large delay. If a tree is built in a random fashion, this dependence may cause the delay through N levels of high fanout gates to

exceed the delay through N+1 levels of low fanout gates.

The ratio of off-chip memory access time to on-chip memory access time indirectly creates additional GaAs/Si differences. When this ratio is large, a memory pipeline is often used and is several levels deep. When this depth exceeds the average distance between branch instructions or the branch delay, NOOPs must be found to fill the pipeline until execution of the branch instruction is completed.

4.1.1. Pipeline Depth

The memory pipeline depth is determined by ratio of the memory fetch time to the data path time. If the data path time is lengthened by allowing the adder to take longer, then the pipeline depth decreases and fewer NOOPs must be inserted into the instruction stream. This decreases both idle time of each processor stage and memory usage. The lengthening of the adder time can be done by replacing a full carry look-ahead adder with a ripple carry adder. This will probably occur only in a GaAs environment since most Si systems do not have deep memory pipelines. Without a deep pipeline, increasing the data path time would increase execution time. In addition, the delay of the ripple carry adder is of the same order of magnitude as the delay of the full carry look-ahead adder with GaAs technology. The narrow separation of the delays is different from Si technology where the delays are vastly different.

4.1.2. Adder Type

As mentioned, this set of changes (transistor count per chip, on-chip gate delay, the ratio of off-chip to on-chip memory access time, and gate fanin and fanout) affects the choice of adders in many ways. Traditionally, the full carry look-ahead has been preferred for high speed applications and ripple carry adders have been discounted for all but the shortest wordlength. Reevaluation of the adders may show that the fastest adder in Si is not the best or fastest in GaAs technology. The new technology will also improve the performance of the slower adders. Therefore, this section is devoted to the change in adder performance from the Si environment to the GaAs environment and how it affects the choice of adder designs.

4.2. Evaluation Methodology

The choice of adder structures was influenced by previous research [FurMi85], [Sherb84]. To cover the range of possibilities for our analysis, an adder with a low number of stages but with high fanin and fanout requirements and another adder with low fanin and fanout requirements were chosen: the full carry look-ahead adder and the ripple carry adder. To make the list complete, an adder that was a compromise between the two extremes was chosen. This choice was the carry select adder presented in [Katev83] which ran stages in parallel rather than serially while still keeping the fanin and fanout requirements low.

To complete the list of adder parameters for various applications, each adder type was examined for bit lengths from four to thirty-two bits. To determine the effect of technology changes, designs with the following fanin and fanout limitations were examined: fanin = two and fanout = two, fanin = two and fanout = five, and fanin = five and fanout = five. To show the effects of different technologies when determining gate delays, the delay per fanin and fanout and the base delay for each gate was varied. These parameters were then used to generate delay information for each adder for the first test. Adder delays give no information about implementation difficulties. Therefore, for the second test, we generated a hardware description for a subset of adder lengths for each adder type. The descriptions were input to the automatic layout tool, MP2D, which then calculated areas for each adder type.

4.3. Experiment Procedure

The first of the two tests was to determine the delay for each adder type by deriving the delay formulas for speed comparisons based on fanin, fanout and bit length. In addition, the delay per additional fanin and fanout and the base delay for gates could be controlled. These formulas could then be used to compare the adders. When the formulas were developed the circuit simulation software was not available. Therefore, the formulas were implemented in C language for each of the three adder types.

4.3.1. Adder Delays

The first step was to derive equations for a simple NOR gate which include the delays for the different inputs. Each NOR gate has five parameters:

Fast	the rise and fall delay of the fast input,
Slow	the rise and fall delay of the slow input,
C_{fast}	the fast input load capacitance,
C_{out}	the fanout capacitance,
Fast*C_{out}	the delay per additional fanout,
maxfanin	the maximum fanin,
maxfanout	and the maximum fanout.

The delay formula is

$$\text{delay} = (\lceil \log_{\text{maxfanin}}(\text{Fanin}) \rceil^{\dagger} - 1)*(\text{base}_{\text{slow}}(\text{maxfanin}) + \text{Slow} * C_{\text{fast}}$$

$$+ (2 * \lfloor (\lceil \log_{\text{maxfanout}}(\text{Fanout}) \rceil + 1)/2 \rfloor^{\ddagger} - 1)$$

$$* (\text{base}_{\text{fast}}(1) + \text{Fast}*C_{\text{fast}}) + \text{base}_{\text{fast}}(1) + \text{Fast}*C_{\text{out}}.$$

where

$$\text{base}_{\text{slow}}(\text{fanin}) = \text{delay}_{\text{base}} + \text{fanin}*\text{delay per fanin}$$

and

$$\text{base}_{\text{fast}}(\text{fanin}) = \text{delay}_{\text{base}} + \text{fanin}*\text{delay per fanin}$$

Another question is whether the delay caused by the conductors is significant compared to the delay of each gate. Given information on the Tektronix E/D MESFET process and information from their GaAs standard cell library, I was able to determine that the capacitance of the longest expected conductor was less than 1/4 of the lowest input capacitance of any device in the Tektronix GaAs standard cell library *.

The delay for each adder depends on several parameters:

D - the delay of the critical path,
N - the number of bits in the adder,
B - the value of the base time,
Cfi - the capacitance per additional fanin,
Cfo - the capacitance per additional fanout,
Nfi - the number of fanins,

---

† $\lceil X \rceil$, the ceiling function, picks the smallest integer greater than X.
‡ $\lfloor X \rfloor$, the floor function, picks the largest integer smaller than X.
* This information is not specified because it is Tektronix Confidential.

Nfo - the number of fanouts,
T - adder type.

The overall formulas for delay generated always consider the varying delays between inputs. The formulas also take into account varying fanin and fanout limitations. The general formula varies as D(N) = f(N, B, Cfi, Cfo, T, Nfi, Nfo).

For D(N) given T=ripple carry as shown in figure 4.1,

$$D(N) = xor(Cfi_{fast}+Cfi_{slow}) + carry(2*Cfi_{fast})*N + xor_{fast}(Cfi_{fast})$$

where

$$xor(C_{out}) = max(\ base_{fast}(2)+Fast*Cfi_{fast}*2 +$$
$$base_{fast}(2) +Fast*Cfi_{slow} + base_{slow}(2) +Slow*C_{out}\ ,$$
$$base_{slow}(2)+Slow*Cfi_{fast}*2 +$$
$$base_{fast}(2) +Fast*Cfi_{fast} + base_{fast}(2) +Fast*C_{out}\ ),$$

$$xor_{fast}(C_{out}) = min(\ base_{fast}(2)+Fast*Cfi_{fast}*2 +$$
$$base_{fast}(2) +Fast*Cfi_{slow} + base_{slow}(2) +Slow*C_{out}\ ,$$
$$base_{slow}(2)+Slow*Cfi_{fast}*2 +$$
$$base_{fast}(2) +Fast*Cfi_{fast} + base_{fast}(2) +Fast*C_{out}\ ),$$

$$carry(C_{out}) = base_{fast}(2)+Fast*Cfi_{fast} + base_{fast}(2)+Fast*C_{out}\ ,$$

and once again

$$base_{slow}(fanin) = delay_{base}+fanin*delay\ per\ fanin$$

and

$$base_{fast}(fanin) = delay_{base}+fanin*delay\ per\ fanin.$$

For D(N) given T=carry select as shown in figure 4.2, when $m = \lceil \sqrt{N} \rceil$

$$D(N) = xor(Cfi_{fast}+Cfi_{slow}) + carry(2*Cfi_{fast})*m + xor_{fast}(Cfi_{fast}) +$$
$$mux(m,\ 2*Cfi_{fast})*\lfloor N/m \rfloor + mux(N\%m,2*Cfi_{fast})$$

where

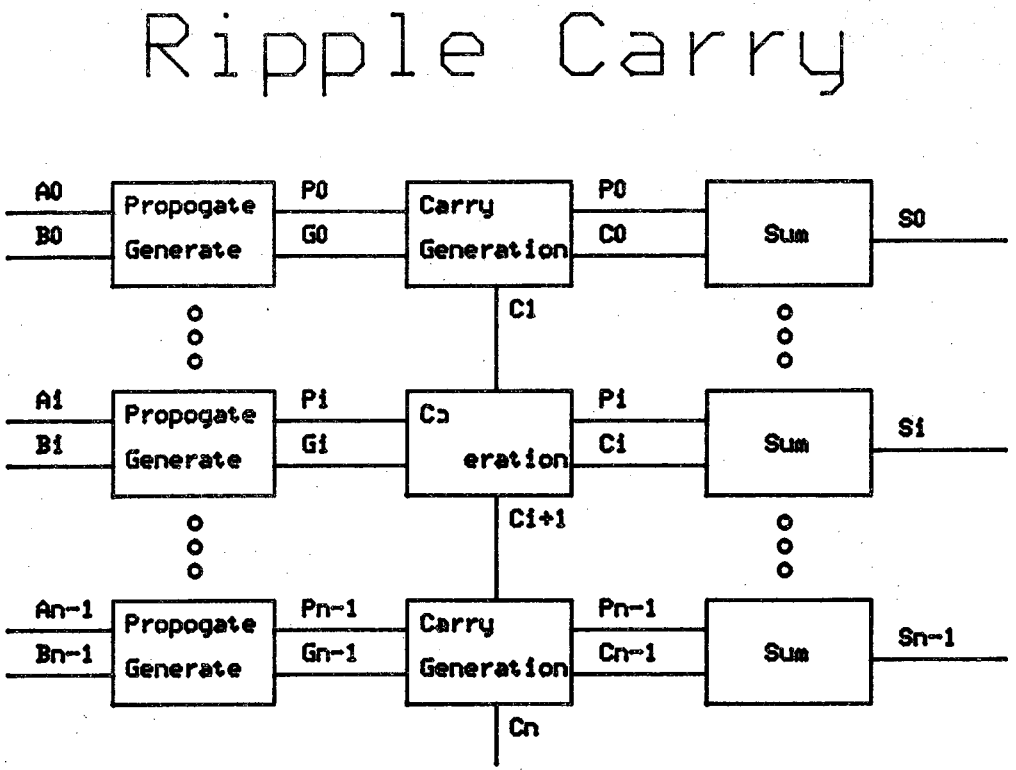


Figure 4.1 Block Diagram of Ripple Carry Adder

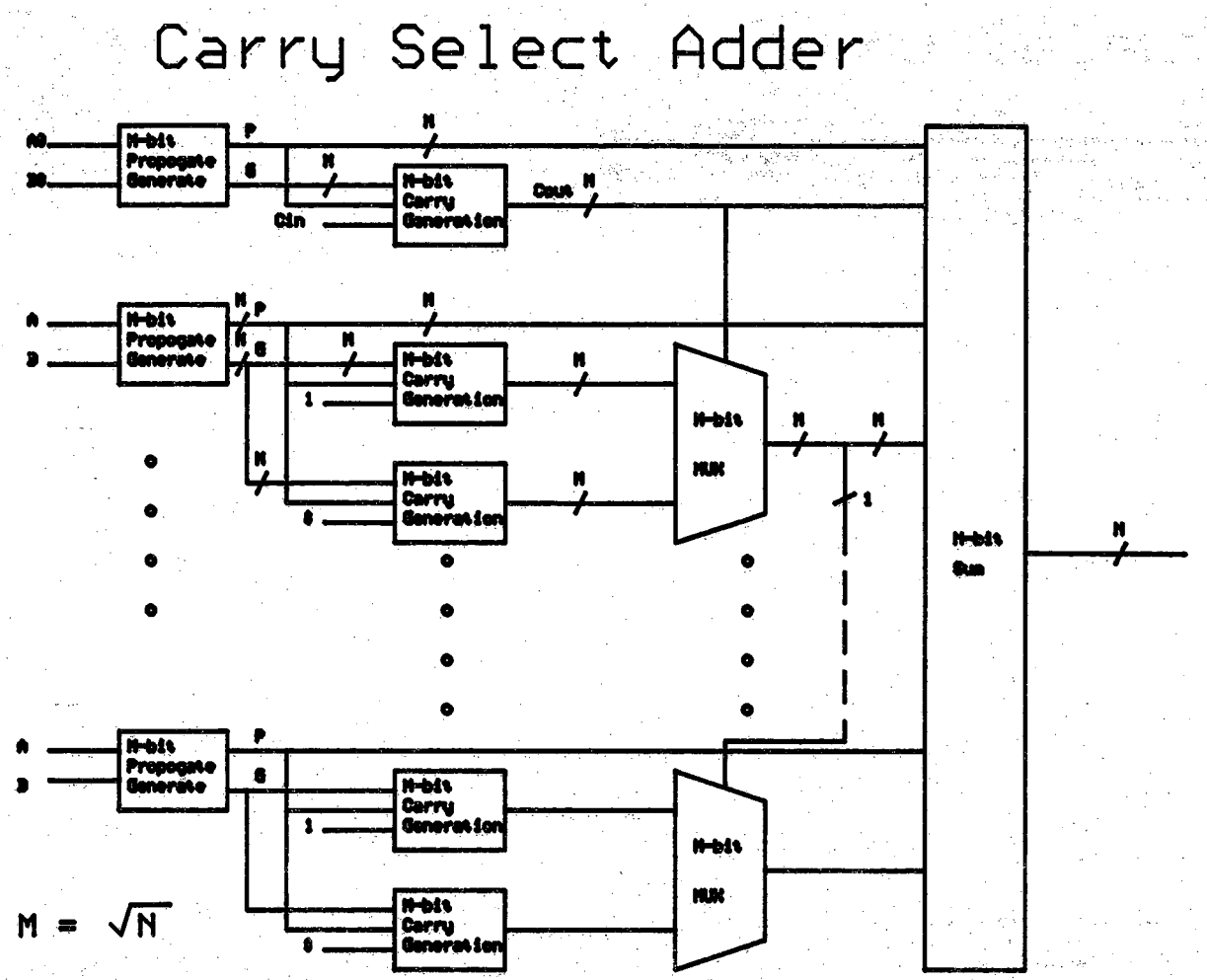


Figure 4.2 Block Diagram of Carry Select Adder

$$mux(m, C_{out}) = outtree(m, Cfi_{fast}) + base_{fast}(1)+Fast*Cfi_{fast} + base_{fast}(2)+Fast*Cfi_{fast} + base_{fast}(2)+Fast*c_{out},$$

and

$$outtree(m, C_{out}) = (\lceil \log_{maxfanout}(m) \rceil -1) * (base_{fast}(1)+Fast*Cfi_{fast}*maxfanout) + base_{fast}+Fast*C_{out}*maxfanout,$$

but if $\lceil \log_{maxfanout}(m) \rceil$ is odd then
$delay = delay+base_{fast}(1)+Fast*Cfi_{fast}$.

For D(N) given T=full carry look ahead as shown in figure 4.3,
$D(N) = xor(Cfi_{fast}+Cfi_{slow}) + carrytree(N, 2*Cfi_{fast}) + xor_{fast}(Cfi_{fast})$.
where

$$carrytree(m,C_{out}) = outtree(\frac{(m^2+m)}{2}+1,Cfi_{fast}) + intree(m,Cfi_{fast}) + intree(m,C_{out}),$$

and

$$intree(m, C_{out}) = (\lceil \log_{maxfanin}(m) \rceil -1) * ((base_{slow}(maxfanin)+Slow*Cfi_{fast}) + base_{fast}(1)+Fast+Cfi_{slow}) + base_{slow}(maxfanin)+Slow*C_{out}.$$

These equations were then implemented in C for each adder. The base gate delays and delays per additional fanin and fanout were implemented as variables set at compile time. This information was then plotted to show the difference between the adders as the technology parameters were varied.

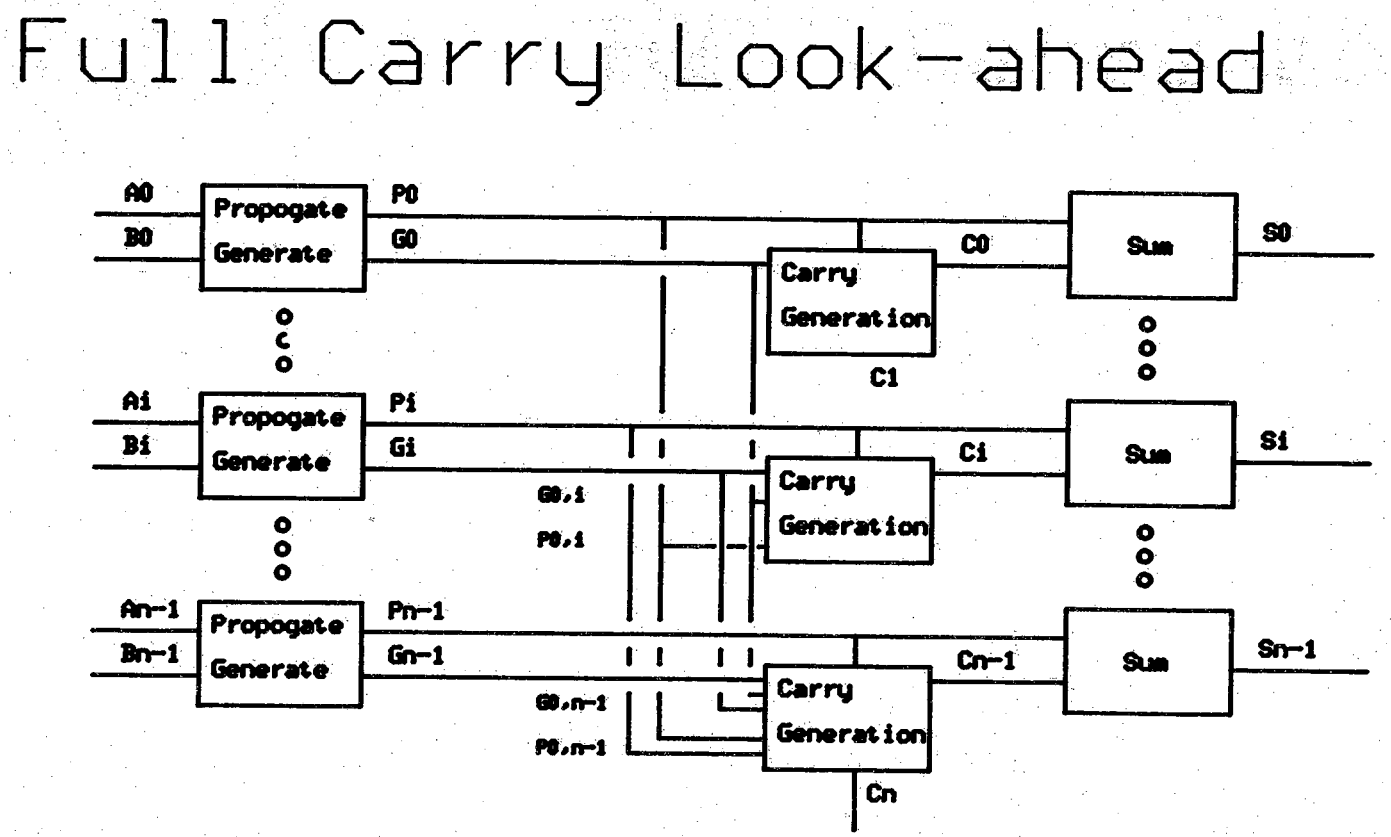


Figure 4.3 Block Diagram of Full Carry Look-Ahead Adder

4.3.2. Adder Area

So far only the delay of the individual adders has been discussed. The delays can then be used in conjunction with the area consumed to evaluate each adder. Determining the area consumed was done as part of the second test where each adder was created with HDL. Gate level descriptions of each adder complete with timing information could be entered using the description language. The resulting description was run through a translation program that extracted and formatted the data for MP2D, the automatic layout program. MP2D, multi-port 2-dimensional placement and routing program, took the translated HDL information and generated layouts for portions of the adders and for complete adders. Included with this information was the actual layout and area that each design required, as well as the approximate area per transistor including wiring areas. The data was then used to build general curves and equations to approximate any of our adders. These curves allowed us to compare the adders on this second criterion to better estimate their viability in a GaAs environment.

The area of ripple carry adders was calculated by interpolation along a straight line between known points. This was relatively accurate since the data were very linear. The area for a carry select adder of bit length N, when $m=\lceil\sqrt{N}\rceil$, is approximated by the following formulas.

$$A(N)=area_{pg(m)}+area_{pg(N\%m)}$$
$$+(m-1)*2*(area_{cg(m)}+area_{cg(N\%m)})+area_{cg(m)}$$
$$+(m-1)*(area_{mux(m)}+area_{mux(N\%m)})$$
$$+m*(area_{sum(m)}+area_{sum(N\%m)})$$

where

pg(m) = area of an m-bit propagate generate box,
cg(m) = area of an m-bit carry generation box,
mux(m) = area of an m-bit mux, and
sum(m) = area of an m-bit summation box.

The area for a full carry look-ahead adder of bit length N is approximated by the following formulas.

$$A(N)=area_{pg(N)}+area_{sum(N)}+fanout_{carry-in}(N)$$

$$+\sum_{i=0}^{i=n-1}(\text{markfanout}_{\text{generate}}(N-i)$$

$$+\text{fanout}_{\text{propagate}}((i+1)*(n-i)+1)+\text{area}_{\text{carry}}(i+1))$$

where

fanout(m)= area of an m-bit fanout tree and
carry(m)= area of the m_{th} carry bit.

4.4. Presentation of Results

Using all the delays and the areas for each adder type, the difference in adder design for Si and GaAs technologies can be described.

4.4.1. Adder Delays

The delay programs used the given formulas to generate the delay data presented in figures 4.4 through 4.9. The first set of three figures displays the delay of each adder in a GaAs technology for the three fanin and fanout configurations. The second set of three figures displays the delay of the adders in a Si technology for the fanin and fanout configurations. Each figure shows delay in picoseconds versus word length in bits. Simplifying the adder equations shows that the delay of the ripple carry adder increases by O(N) (Order N), the carry select adder increases by $O(\log(N)N^{1/2})$, and the full carry look-ahead adder increases by O(log(N)), where N is the bit length of each adder. For large N, the comparison shows that the full carry look-ahead adder will always be faster than the carry select adder, and both will be faster than the ripple carry adder. The technology determines the key parameters that determine the magnitude of the difference in delay time and the fastest adder for short bit length adders.

Although the ripple carry adder has delays increasing linearly to values greater than either of the other two adders, the delays of the carry select adder and the full carry look-ahead adder are increasing at almost the same rate. For the bit length of interest (32 bits), the propagation delay through the full carry look-ahead adder is two-thirds that of the carry select adder. Figure 4.6, where the maximum fanin and fanout are five, shows this much more clearly than figure 4.4 where the maximum fanin and fanout are only two. The differences in the delays are due to the slow rise time of the E/D-MESFET gates and the long base delays of the CMOS/SOS gates.

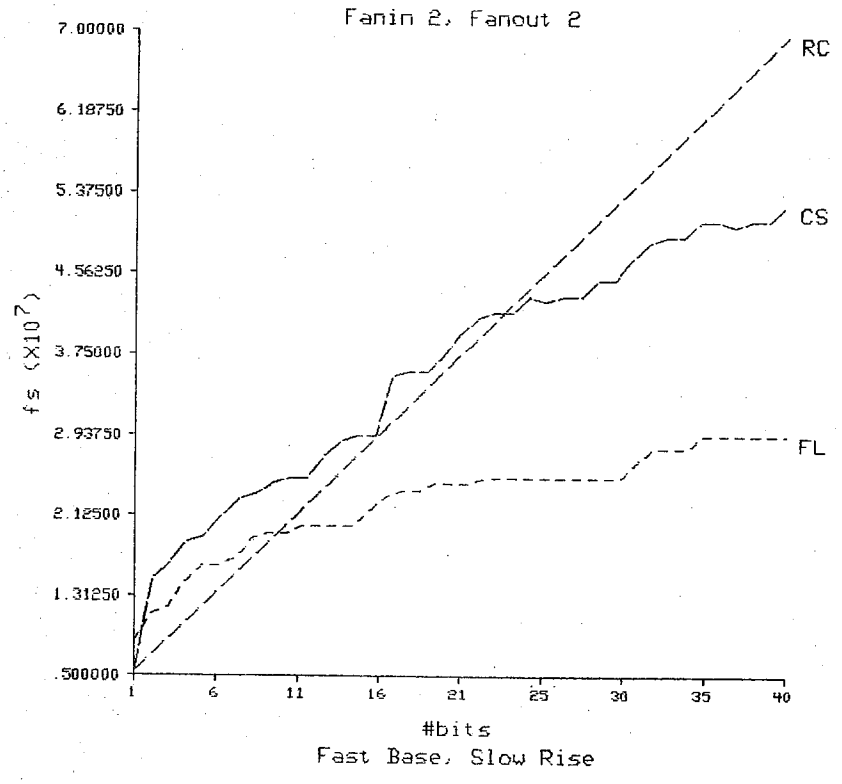


Figure 4.4 Adder Delays with GaAs E/D-MESFET Parameters with maximum fanin=2, maximum fanout=2

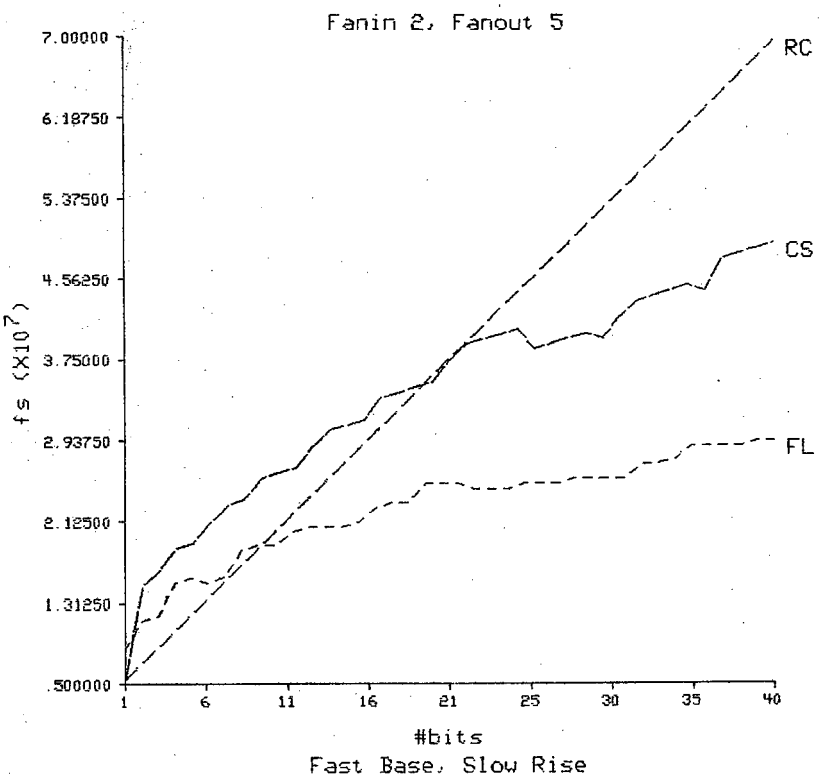


Figure 4.5 Adder Delays with GaAs E/D-MESFET Parameters with maximum fanin=2, maximum fanout=5

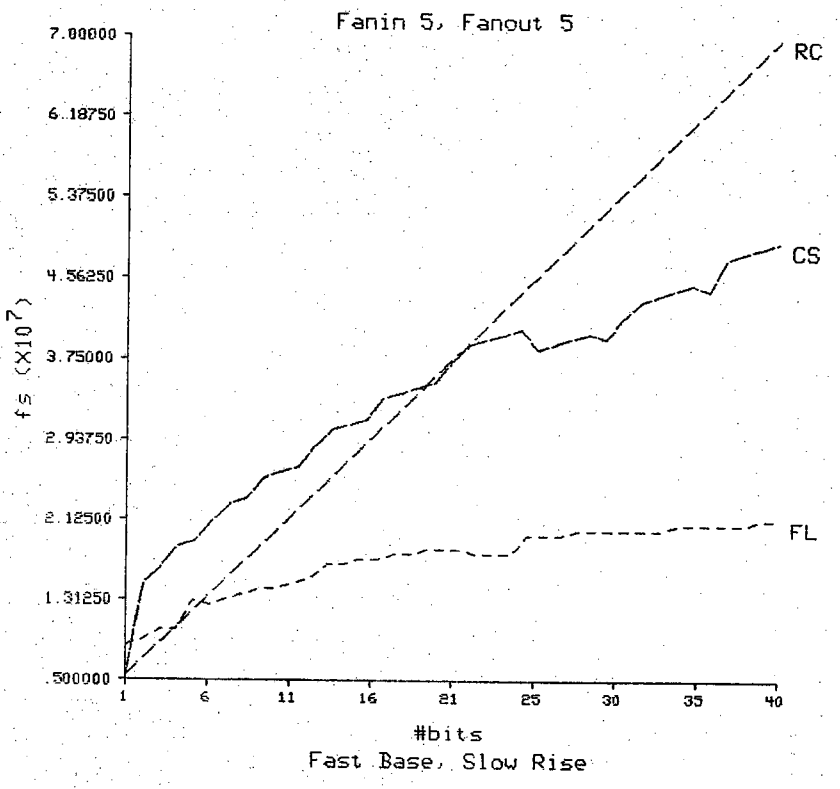


Figure 4.6 Adder Delays with GaAs E/D-MESFET Parameters with maximum fanin=5, maximum fanout=5

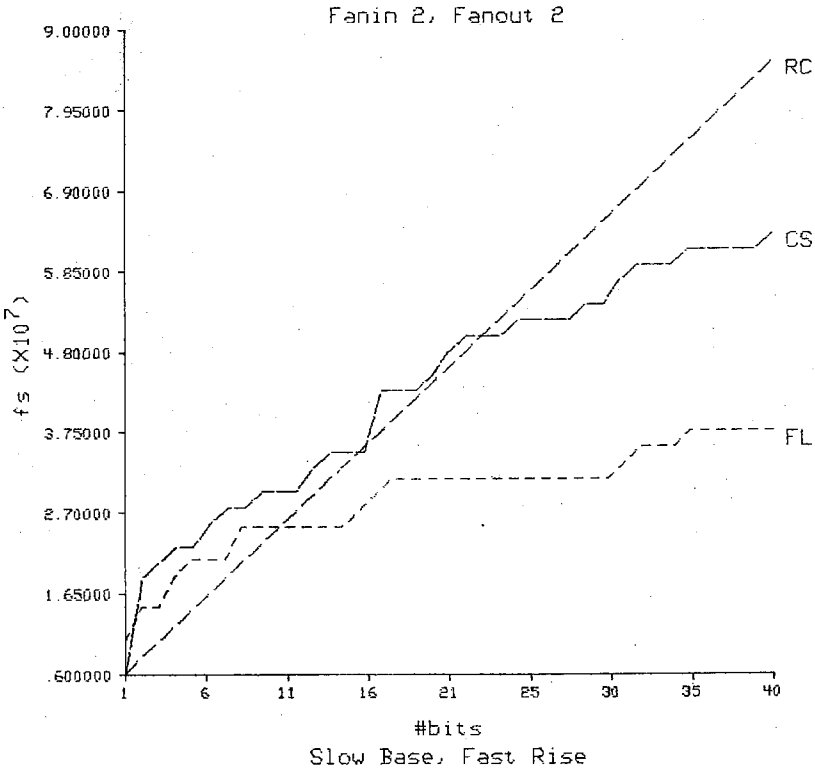


Figure 4.7 Adder Delays with Si CMOS/SOS Parameters with maximum fanin=2, maximum fanout=2

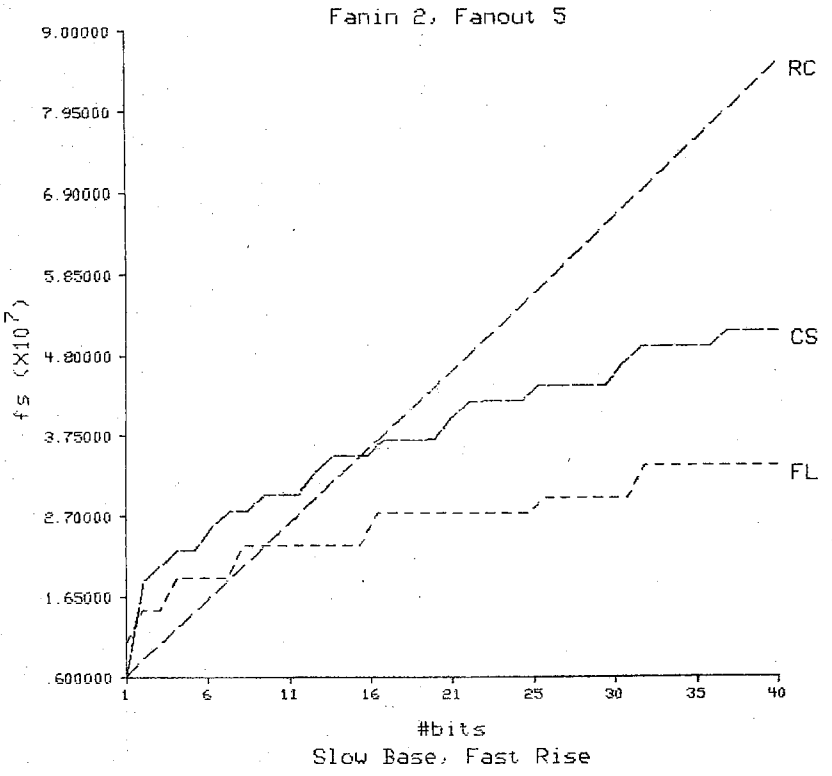


Figure 4.8 Adder Delays with Si CMOS/SOS Parameters with maximum fanin=2, maximum fanout=5

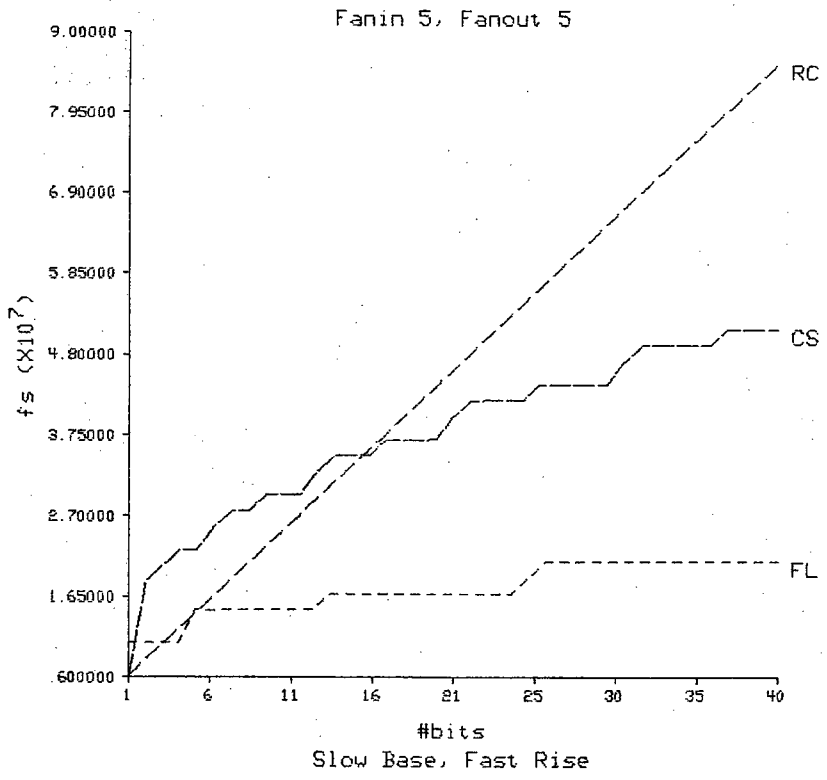


Figure 4.9 Adder Delays with Si CMOS/SOS Parameters with maximum fanin=5, maximum fanout=5

The slow rise time severely degrades performance as fanin and fanout increases as shown by the large difference in execution times for figures 4.4 and 4.6, and 4.7 and 4.9. When only the rise time is changed, the speed of the individual adders increases dramatically as shown in figures 4.10 through 4.12. Unfortunately, the GaAs E/D-MESFET process is not characterized by such fast rise times and fast total delays.

If the designer wants to avoid an excessively deep memory pipeline, then he should also consider the ripple carry adder. Since the design is characterized by low fanout, the change in maximum fanout from Si has almost no effect. In addition, the delay is only twice that of the full carry look-ahead adder and could increase the total data path time of a processor such as SU-MIPS [Gross82] by 30 percent. This would allow the designer to reduce the memory pipeline depth by the same amount.

4.4.2. Adder Area

As mentioned before, comparisons solely on the basis of adder delays are deficient. Therefore, the area consumed by each of the adders was plotted and is presented in figures 4.13 through 4.15. The area of each adder is shown in terms of mils squared and is plotted against the bit length of each adder. The area required by the ripple carry adder increases linearly with bit length and rises at the slowest rate of the three adders. The area of the carry select adder is growing a little faster: O(N) plus an $N^{1/2}$ component. The fastest adder, the full carry look-ahead adder, consumes area at a much higher rate, $O(N^3)$. Although this is not surprising, the cubic growth rate quickly uses up the available area on a chip. This may not be obvious where fanout limitations are large and the fanout trees will not occupy a large area. A quick glance at the graphs quickly shows that even for large maximum fanin and fanout, the area consumed by the full carry look-ahead adder is unreasonable for large bit length adders. Only the ripple carry adder and the carry select adder conserve enough area to allow other structures to be placed onchip.

In summary, the GaAs environment quickly challenges long-standing conventions. Ripple carry adders are capable of performing the job without severe degradation of performance and can help reduce memory pipeline depths and improve performance. This supports using serial operations in GaAs, particularly when such operations have low fanin and fanout requirements. For adders such as full carry look-ahead adders, the parallel nature uses area in such large quantities that they are not useful in a GaAs environment. Those adders that serialize operations with high fanin and fanout requirements while

parallelizing operations with low fanin and fanout requirements make a good compromise. Carry select adders, which follow this principle, are shown as one of the fastest adders available as well as one of the more area efficient structures.

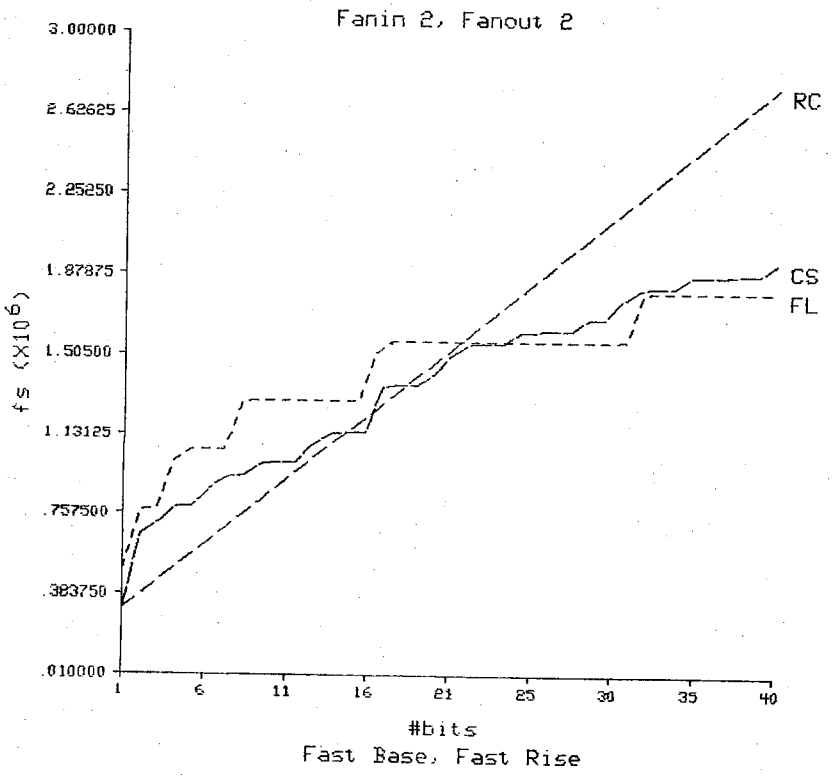


Figure 4.10 GaAs Adder Delays with Decreased Rise Time with maximum fanin=2, maximum fanout=2

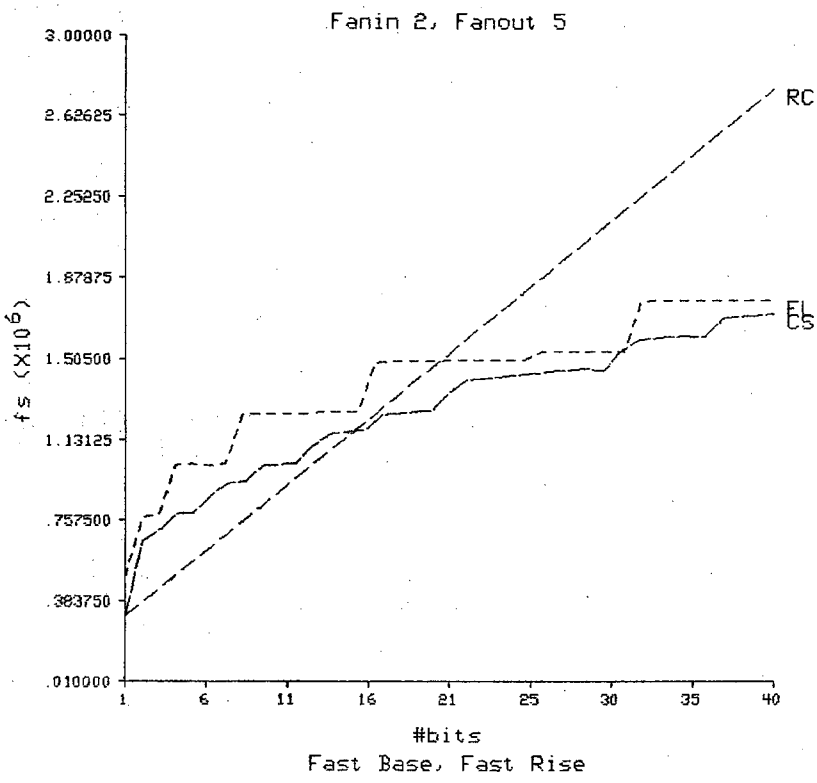


Figure 4.11 GaAs Adder Delays with Decreased Rise Time with maximum fanin=2, maximum fanout=5

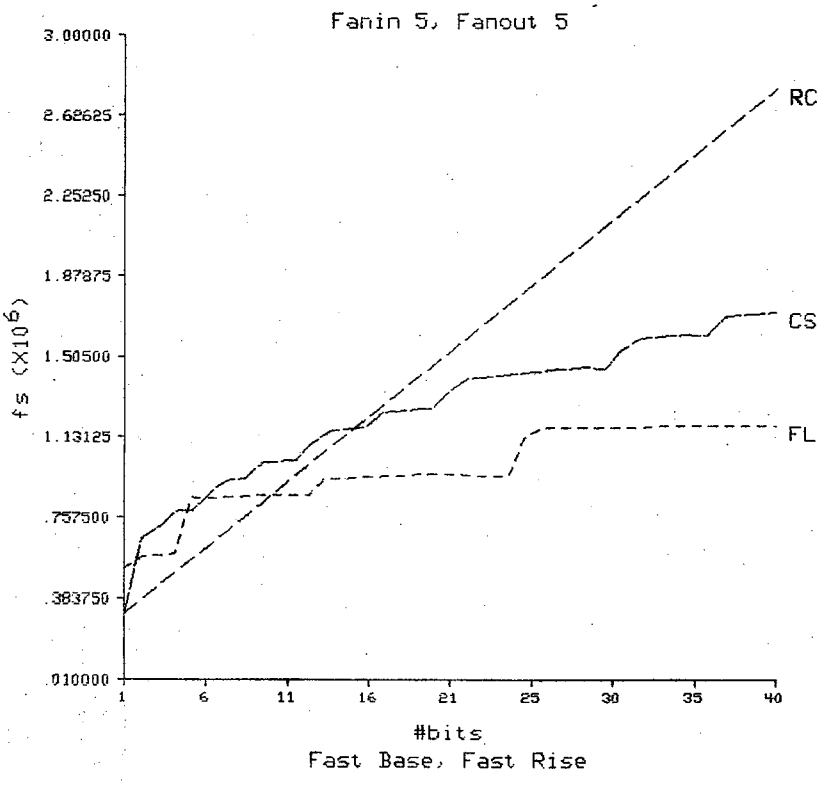


Figure 4.12 GaAs Adder Delays with Decreased Rise Time with maximum fanin=5, maximum fanout=5

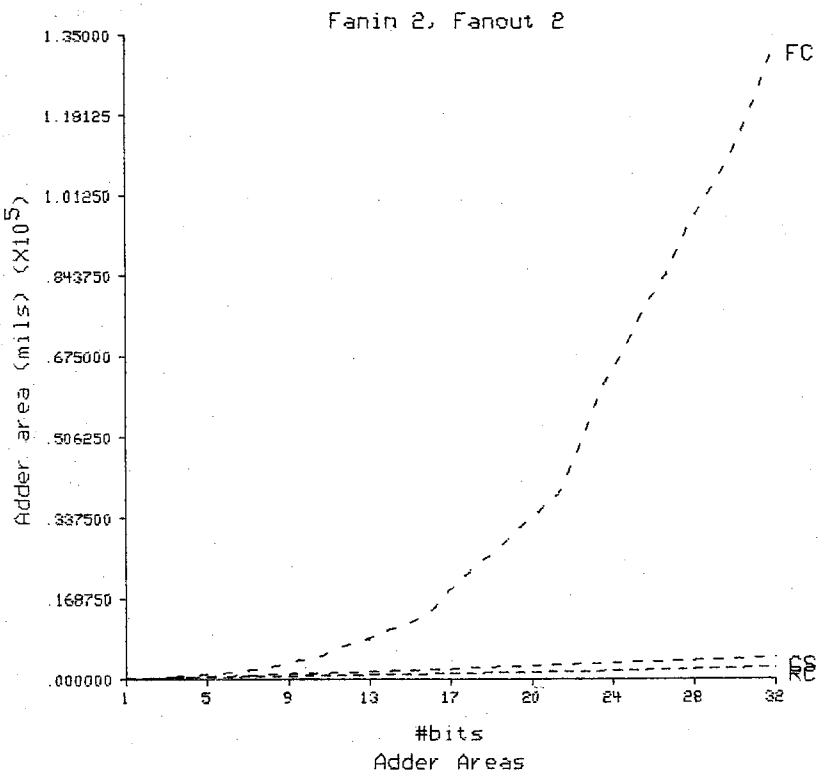


Figure 4.13 GaAs Adder Area with maximum fanin=2, maximum fanout=2

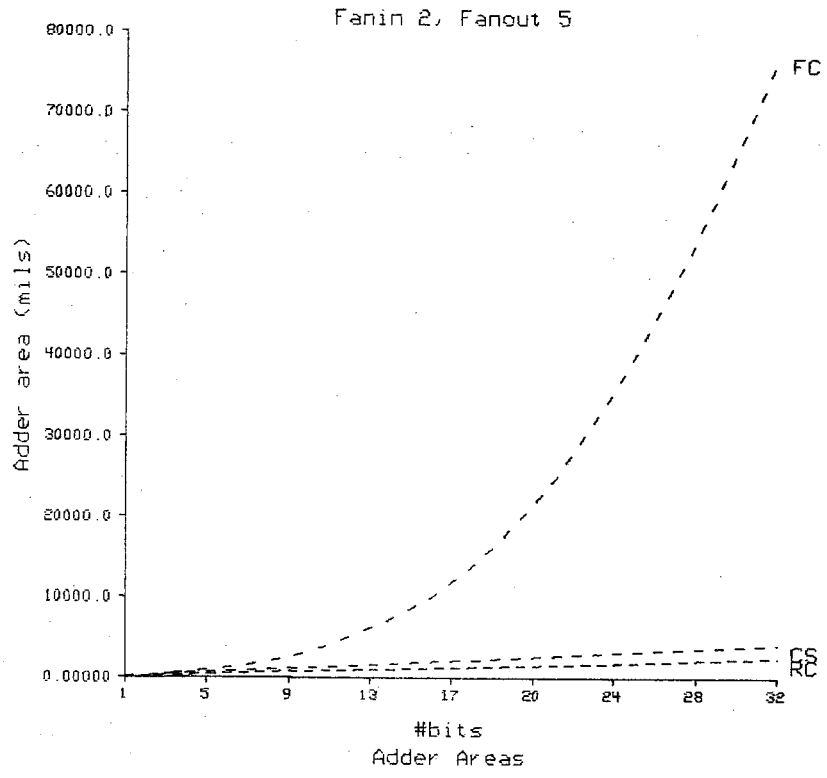


Figure 4.14 GaAs Adder Area with maximum fanin=2, maximum fanout=5

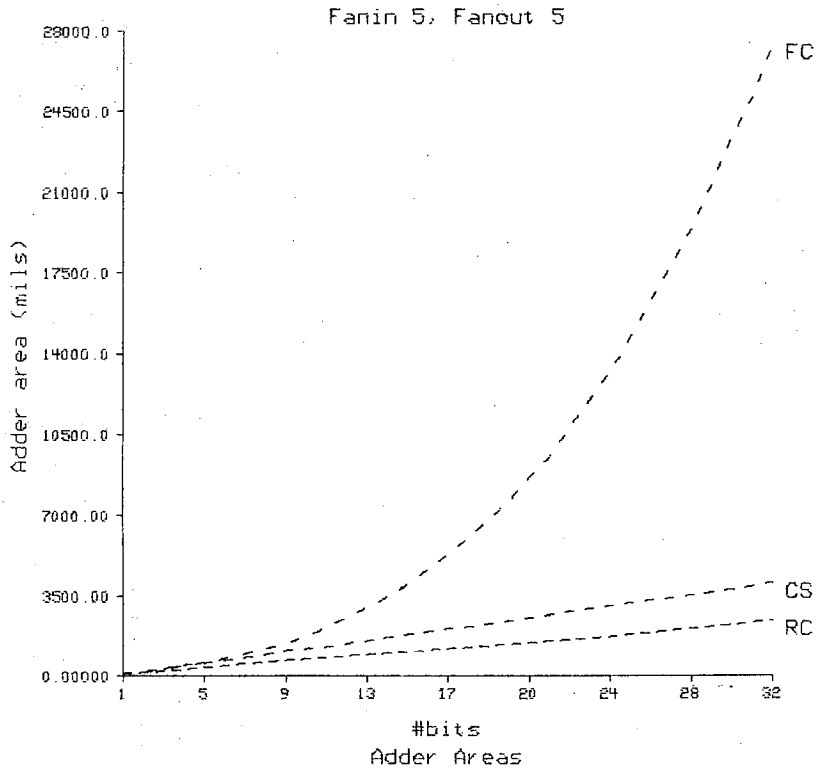


Figure 4.15 GaAs Adder Area with maximum fanin=5, maximum fanout=5

CHAPTER 5
MULTIPLIER PLACEMENT

5.1. Introduction

Current RISC processors such as UCB-RISC [Patte85] have shown the usefulness of large register files and of windowed register sets. The area limitations of the GaAs environment do not allow such large structures. Possible GaAs architectures include as many as 32 registers, but this may not be the best use of space. The proposals discussed here use the area of 16 of those 32 registers for other structures such as barrel shifters and multipliers.

The choice of the proposed multiplier and barrel shifter was heavily influenced by previous work in this area. One influence was from the instruction mixes presented in [Knuth71]. Although these mixes are predominantly simple arithmetic and branching operations, multiplication was still significantly more frequent than operations such as division and shifting. By themselves, these mixes do not show the most important operations to investigate. However, the instruction mix of the benchmarks presented in figure 5.1 is used to describe the possible applications of the processor and help highlight promising operations to investigate. The instruction mix distribution shows that after the common operations such as loads, branches and simple arithmetic operations, multiplication was the most frequent operation and shift or rotate instructions occurred relatively infrequently. Since area is often sacrificed for performance and vice-versa, shifting speed can be reduced and area can be freed while multiplication speed can be increased by using the freed area. Since multiplication occurs so frequently and shifting occurs so infrequently, performance should improve when these changes are made.

5.1.1. Shifter Choice

The type of multiplier and barrel shifter still must be decided since there is a wide variety in their speed and complexity. The choice of shifters is relatively simple: use a full barrel shifter or a barrel shifter with a small shift capability. The implementation of any shifter for SU-MIPS allows a maximum shift of 2^n-1. Since the shifter was chosen to minimize area and still be able to

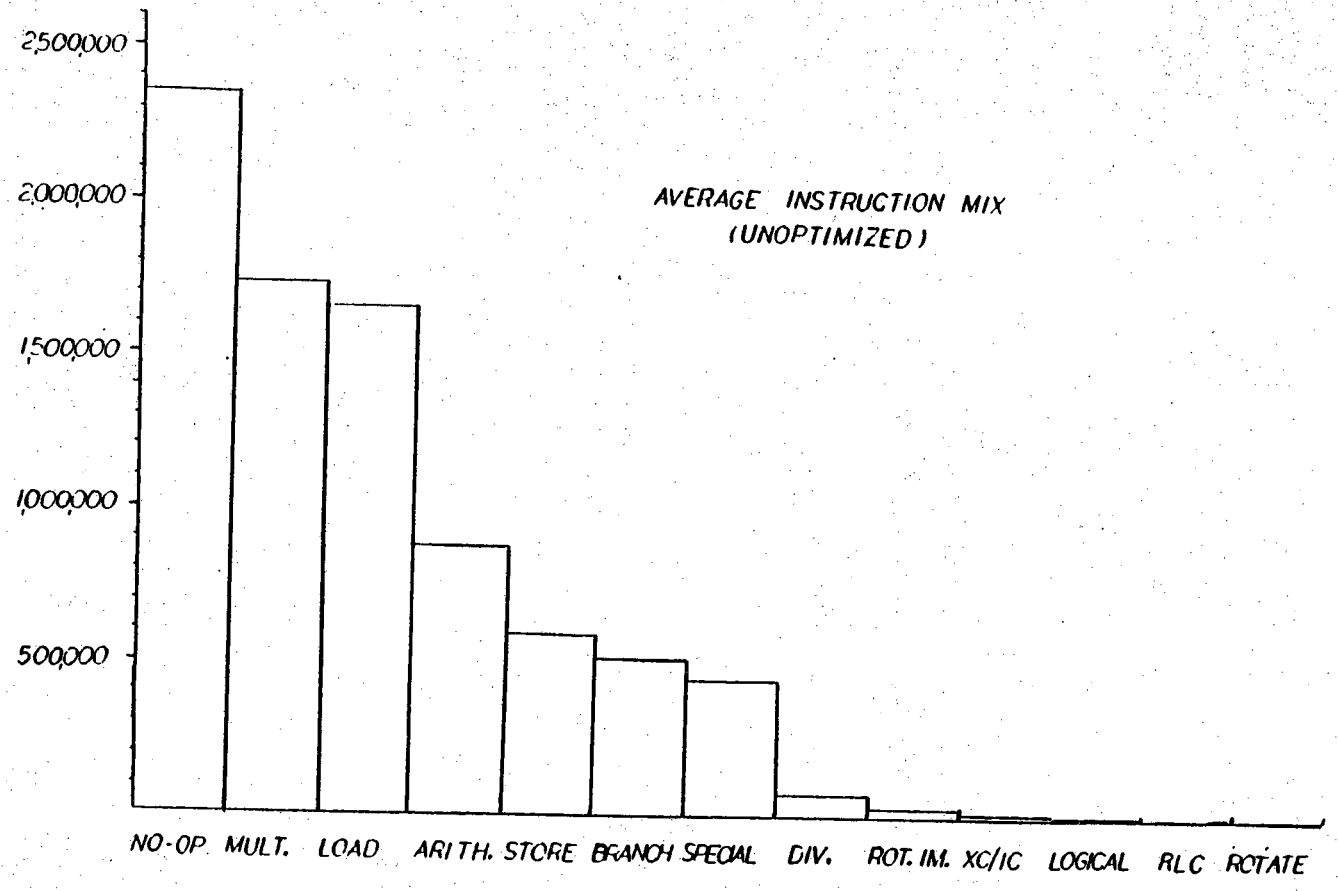


Figure 5.1 Instruction Mix of Application Benchmarks

shift one and two bit positions, the maximum shift was chosen to be three. Choosing a shifter which shifts by one, two, or three bit positions requires that all large shift counts be synthesized with a series of smaller shifts. This does not incur a large penalty when there are very few shift instructions or most shift instructions have a small displacement.

5.1.2. Multiplier Choice

The choice of multipliers is a little more complex, ranging from parallel multipliers to synthesis of multiplication from shift and add instructions. After the initial evaluation, parallel multipliers were discarded because their complexity violates the transistor count limitations of the chip. Synthesis is discarded because of the excessive time necessary for a multiplication. Two of the remaining options are booth-step algorithms built into the hardware or bit-serial multipliers.

The booth algorithm can be built to multiply two bits of the multiplicand during each instruction cycle so that a 32x32 multiply takes 16 instructions plus the overhead instructions (loading and storing the operands and results). Although the booth-step algorithm can be built into the ALU and will be part of the data path, the bit-serial multiplier must be off the data path because it takes many system clock cycles to complete the multiplication. The bit-serial multiplier works serially and can take only one bit at a time which results in 32 clock cycles to do a multiplication. Only 16 instruction cycles are needed to complete a multiply since the system clock runs at twice the instructions fetch rate. The performance can be improved by increasing the frequency of the bit-serial multiplier clock. The range of clock frequencies examined and the execution time are discussed later in this chapter. Another possibility investigated is to put the bit-serial multiplier off-chip and access it as a coprocessor.

Either of the bit-serial multiplier solutions create a delay fillin problem. Since the multiplier takes n cycles to complete, n-1 slots must be filled with other instructions waiting for the product to become valid. This is aggravated in the case of the off-chip multiplier since the delay to send operands off-chip and receive the results increases the delay fillin time. If the frequency of the bit-serial multiplier clock were increased, multiplication would take less time. This would require less delay fillin which consequently conserves memory.

Temporal conflicts between multiplication operations in multiplication intensive applications are also reduced. In multiplication intensive applications, the number of instructions (distance) between multiplication

instructions is small. If the delay fillin is always smaller than this distance, then the reorganizer may be able to fillin with useful instructions rather than NOOPs. Large delay fillins, however, may exceed the distance between multiplication instructions and require the reorganizer to insert NOOPs into the code. These unproductive instructions then reduce performance and increase memory usage.

The addition of instructions also affects how the processor performs with cache. If the instructions are part of a loop, an instruction or data cache may improve performance.

5.2. Multiplier Placement Evaluation Methodology

With the above facts in mind, three implementations were examined. The first implementation is the original SU-MIPS [HeJoP83]. This implementation supports the booth-step multiplication algorithm which is built into the ALU and includes a full 32-bit barrel shifter. The SU-MIPS also has 16 32-bit general purpose registers. The second implementation, RCA-MIPS#M1, retains the 16 registers but the barrel shifter is replaced by a simple shifter capable of shifting a 32-bit word one, two, or three bit positions. The space freed by the smaller shifter is used by a 32x32 bit-serial multiplier running on its own clock and faster than the system clock. The third implementation, RCA-MIPS#M2, uses the same one-two-three shifter as RCA-MIPS#M1, but the bit-serial multiplier is moved to an off-chip on-package location. The space no longer used by the multiplier and shifter is used to increase the register file size from 16 to 32 registers.

5.3. Multiplier Experimental Procedure

The analysis of the various implementations was done in four parts. First, the SU-MIPS simulator [Gross83] was run to determine the performance of the baseline architecture for each benchmark. Second, the RCA-MIPS#M1 version of the simulator was run with the bit-serial multiplier clock rate set to two, four, six, eight, and ten times the system clock in the GaAs tests and one, two, four, six, and eight times the system clock in the Si tests. The difference between GaAs and Si parameters was due to the smaller ratio of instruction cycle time to device delay time in the Si technology compared to the GaAs technology. The number of NOOPs replaced by useful instructions during the multiply delay was set to zero, one, or two.

Third, the RCA-MIPS#M2 version of the simulator was run with the same fillin and system clock parameters. In addition, the time to get

information from the cpu to the off-chip on-package multiplier was set equal to two and four instruction cycles in the GaAs tests and one and two in the Si tests. The difference in parameters was due to the larger ratio of off-chip memory access time to on-chip memory access time in GaAs compared to Si. Each set of tests was run for the case of an optimized program and an unoptimized program. This optimization was an assembly level optimizer and is described in [GroHe82].

Before conducting each test, the changes in executable code for each of the simulator versions was determined. The simulator was then modified to interpret the execution of the instructions being examined. Each shift by a constant was transformed to a series of smaller shifts demonstrated by sll #11, R_{src} which is transformed into

```
sll #3, Rsrc
sll #3, Rsrc
sll #3, Rsrc
sll #2, Rsrc.
```

Shifts using the Lo register for a variable count shifts use a jump table as shown below.

```
        srl, #2, Rcnt,Rtmp
        subr #8, Rtmp
        bra #1, L1 [Rtmp]
        nop
L1:
        sll #2, Rsrc        sll #2, Rsrc
        sll #2, Rsrc        sll #2, Rsrc
        sll #2, Rsrc        sll #2, Rsrc
        sll #2, Rsrc        sll #2, Rsrc
        sll #2, Rsrc        sll #2, Rsrc
        sll #2, Rsrc        sll #2, Rsrc
        sll #2, Rsrc        sll #2, Rsrc
        sll #2, Rsrc        sll #2, Rsrc
        and #3, Rcnt,Rtmp
        beq #0, Rtmp, LEND
        nop
        beq #1, Rtmp, LEND
        sll #1, Rsrc
        beq #2, Rtmp, LEND
```

```
        sll #1, R_src
        sll #1, R_src
LEND:
```

Shifting by a count was not done with a loop because the SU-MIPS instructions allowed a shorter assembly code implementation for a jump table. When a multiply or shift instruction was interpreted, the original instruction was executed. The benchmark statistics were then modified to reflect the execution of the sequence of instructions for the modified architecture.

Since the original code was always executed, the statistics were modified by the simulator to reflect the instruction mix and execution time of the modified code. This meant that the reorganizer could not do compile time fillin after multiplication instructions and forced us to assume the compiler would do delay fillin with a certain efficiency. We assume that the compiler will only fillin zero, one, or two instructions since the SU-MIPS reorganization algorithm was efficient for a fillin of one, was marginally effective for a fillin of two, and produced no noticeable improvement for fillin greater than two.

In addition, we tried to reduce the workload by limiting the performance analysis to the five benchmarks which have multiplication in them: *bubble*, *intmm*, *puzzle*, *quick*, and *towers*. The information gathered for the multiplication tests is shown in tables 3 through 8 for GaAs data and tables 9 through 14 for Si data.

5.4. Presentation of Results

All the figures presented show the results with unoptimized code. The figures with optimized code are similar. However, the curves are all shifted down and the curves in each family have less separation between them. Figures 5.2 through 5.5 display the execution time of a benchmark against the ratio of the multiplier clock to the system clock for the on-chip bit-serial multiplier of RCA-MIPS#M1 for the GaAs parameters. The execution time for the Si parameters is shown in figures 5.6 through 5.9. Each curve represents a different delay fillin constant: zero, one, or two. Although the curves show the execution time decreasing as the multiplier clock ratio increases, the execution time is never better than in the case of the baseline architecture. This is true for any values of the delay fillin parameter used and for both GaAs and Si technologies. This could be due to many shift instructions with a large shift count. An example is the sra #31,R_{src} instruction which is frequently used for sign extension.

The benchmarks available to run on the simulator never fully utilize the 16 registers available on SU-MIPS. Therefore, they could not possibly use all 32 registers on RCA-MIPS#M2. For this reason, changing the simulator to use 32 registers would not change the results and, hence, was not done. As an alternative, a certain percentage of all loads and stores are assumed to occur because the register file has overflowed and memory had to be used for information storage. The load and store operations are tallied during execution. A percentage of the load and store operations corresponding to the amount of register file overflow are then subtracted from the total execution time. This percentage could then be varied for each run of the simulator. The effect of this is indicated in figures 5.10 through 5.13 for GaAs parameters and figures 5.14 through 5.17 for Si parameters. When the offchip delay parameters are increased, the plots shift up to show the increased execution time. Since the addition is linear, the shape of the graphs is unchanged as the delay is increased.

The next sets of curve families include register utilization percentages. Figures 5.18 through 5.21 again map the execution time of a typical benchmark against the clock ratio for GaAs parameters while figures 5.22 through 5.25 map the execution time for Si parameters. Notice that as the percentage of load and store operations for register overflow increases from 30 to 90 percent, the execution time decreases. This would be an ideal situation, especially considering that the execution time of the baseline architecture was surpassed at around 20 percent. Even during parameter passing, the biggest user of the register file, none of our benchmarks came even close to complete utilization of 16 registers. This low utilization gives a percentage of loads and stores used for overflow close to zero. An architecture with a larger register file would perform better with a compiler that utilizes registers more fully. Such compilers would have to do more register lifetime maximization, do more parameter passing in registers, save more data or pointers in registers, and otherwise utilize the register file more fully. Until such software is written, and benchmarks which will make use of such software are available, the point on the curve remains in question. The software was not modified because of time limitations and the large amount of software which would have to be modified. The details of this software are part a PhD dissertation [Gross84].

The cache experiments could not be done by putting a cache simulator into the MIPS simulator because of our method of expanding and interpreting shift and multiply instructions at runtime. Inserting the expanded instructions into memory at runtime would change the model of memory during execution and would make the cache model inaccurate. In addition, the complexity of

modifying branch addresses on the fly is a larger task than could be accomplished in the available time. Alternatively, the compiler could be rewritten to do multiply fillin, but once again, the amount of work required is massive. Cache and multiplier simulation in the SU-MIPS simulator is of questionable value because of the implementation of the multiplier simulator. Therefore, the following chapter presents an experiment to determine the effect of cache issues on multiplier design.

Table 3
Execution Time for GaAs Onchip Multiplier with Optimized Code (in terms of instruction fetches)

bench	ackp	bubblep	fibp	intmmp	perm	puzzlep	queen	quick	sievep	towersp
baseline	3186804	1196574	584330	1761026	337582	5040988	6529	1132966	166649	3585189
i= 2,s=0	3186845	1240597	584330	2605849	337605	4985102	7290	1587726	166690	4501212
i= 2,s=1	3186845	1240097	584330	2532249	337605	4985102	7290	1582726	166690	4386712
i= 2,s=2	3186845	1239597	584330	2458649	337605	4985102	7290	1577726	166690	4272212
i= 4,s=0	3186845	1238597	584330	2311449	337605	4985102	7290	1567726	166690	4043212
i= 4,s=1	3186845	1238097	584330	2237849	337605	4985102	7290	1562726	166690	3928712
i= 4,s=2	3186845	1237597	584330	2164249	337605	4985102	7290	1557726	166690	3814212
i= 6,s=0	3186845	1238097	584330	2237849	337605	4985102	7290	1562726	166690	3928712
i= 6,s=1	3186845	1237597	584330	2164249	337605	4985102	7290	1557726	166690	3814212
i= 6,s=2	3186845	1237097	584330	2090649	337605	4985102	7290	1552726	166690	3699712
i= 8,s=0	3186845	1237597	584330	2164249	337605	4985102	7290	1557726	166690	3928712
i= 8,s=1	3186845	1237097	584330	2090649	337605	4985102	7290	1552726	166690	3699712
i= 8,s=2	3186845	1236597	584330	2017049	337605	4985102	7290	1547726	166690	3585212
i=10,s=0	3186845	1237097	584330	2090649	337605	4985102	7290	1552726	166690	3585212
i=10,s=1	3186845	1236597	584330	2017049	337605	4985102	7290	1547726	166690	3585212
i=10,s=2	3186845	1236597	584330	2017049	337605	4985102	7290	1547726	166690	3585212

Table 4
Execution Time for GaAs Onchip Multiplier with Unoptimized Code (in terms of instruction fetches)

bench	ackp	bubblep	fibp	intmmp	perm	puzzlep	queen	quick	sievep	towersp
baseline	3101439	3171079	602121	3852427	431167	19313270	39286	1984792	397121	3585189
i= 2,s=0	3101480	3215102	602121	5708450	431190	25456897	40047	2559365	397162	4501212
i= 2,s=1	3101480	3214602	602121	5508450	431190	24689809	40047	2554365	397162	4386712
i= 2,s=2	3101480	3214102	602121	5308450	431190	23922721	40047	2549365	397162	4272212
i= 4,s=0	3101480	3213102	602121	4908450	431190	22388545	40047	2539365	397162	4043212
i= 4,s=1	3101480	3212602	602121	4708450	431190	21621457	40047	2534365	397162	3928712
i= 4,s=2	3101480	3212102	602121	4508450	431190	20854369	40047	2529365	397162	3814212
i= 6,s=0	3101480	3212602	602121	4708450	431190	21621457	40047	2534365	397162	3928712
i= 6,s=1	3101480	3212102	602121	4508450	431190	20854369	40047	2529365	397162	3814212
i= 6,s=2	3101480	3211602	602121	4308450	431190	20087281	40047	2524365	397162	3699712
i= 8,s=0	3101480	3212102	602121	4508450	431190	20854369	40047	2529365	397162	3814212
i= 8,s=1	3101480	3211602	602121	4308450	431190	20087281	40047	2524365	397162	3699712
i= 8,s=2	3101480	3211102	602121	4108450	431190	19320193	40047	2519365	397162	3585212
i=10,s=0	3101480	3211602	602121	4308450	431190	20087281	40047	2524365	397162	3699712
i=10,s=1	3101480	3211102	602121	4108450	431190	19320193	40047	2519365	397162	3585212
i=10,s=2	3101480	3211102	602121	4108450	431190	19320193	40047	2519365	397162	3585212

Table 5
Execution Time for GaAs Offchip Multiplier with Optimized Code
(in terms of instruction fetches)
Offchip Delay of 2

bench	ackp	bubblep	fibp	intmmp	permp	puzzlep	queenp	quick	sievep	towersp
baseline	3186804	1196574	584330	1761026	337582	5040988	6529	1132966	166649	13265771
l= 2,s=0	3186845	1246597	584330	3489049	337605	4985102	7290	1647726	166690	14175970
l= 2,s=1	3186845	1245597	584330	3341849	337605	4985102	7290	1637726	166690	14175970
l= 2,s=2	3186845	1244597	584330	3194649	337605	4985102	7290	1627726	166690	14175970
l= 4,s=0	3186845	1242597	584330	2900249	337605	4985102	7290	1607726	166690	14175970
l= 4,s=1	3186845	1241597	584330	2753049	337605	4985102	7290	1597726	166690	14175970
l= 4,s=2	3186845	1240597	584330	2605849	337605	4985102	7290	1587726	166690	14175970
l= 6,s=0	3186845	1241597	584330	2753049	337605	4985102	7290	1597726	166690	14175970
l= 6,s=1	3186845	1240597	584330	2605849	337605	4985102	7290	1587726	166690	14175970
l= 6,s=2	3186845	1239597	584330	2458649	337605	4985102	7290	1577726	166690	14175970
l= 8,s=0	3186845	1240597	584330	2605849	337605	4985102	7290	1587726	166690	14175970
l= 8,s=1	3186845	1239597	584330	2458649	337605	4985102	7290	1577726	166690	14175970
l= 8,s=2	3186845	1238597	584330	2311449	337605	4985102	7290	1567726	166690	14175970
l=10,s=0	3186845	1239597	584330	2458649	337605	4985102	7290	1577726	166690	14175970
l=10,s=1	3186845	1238597	584330	2311449	337605	4985102	7290	1567726	166690	14175970
l=10,s=2	3186845	1238597	584330	2311449	337605	4985102	7290	1567726	166690	14175970
l= 2,s=0,R=0.3	2618438	1129175	472780	3369988	271642	4398145	6462	1555527	142754	11705637
l= 2,s=1,R=0.3	2618438	1128175	472780	3222788	271642	4398145	6462	1545527	142754	11705637
l= 2,s=2,R=0.3	2618438	1127175	472780	3075588	271642	4398145	6462	1535527	142754	11705637
l= 4,s=0,R=0.3	2618438	1125175	472780	2781188	271642	4398145	6462	1515527	142754	11705637
l= 4,s=1,R=0.3	2618438	1124175	472780	2633988	271642	4398145	6462	1505527	142754	11705637
l= 4,s=2,R=0.3	2618438	1123175	472780	2486788	271642	4398145	6462	1495527	142754	11705637
l= 6,s=0,R=0.3	2618438	1124175	472780	2633988	271642	4398145	6462	1505527	142754	11705637
l= 6,s=1,R=0.3	2618438	1123175	472780	2486788	271642	4398145	6462	1495527	142754	11705637
l= 6,s=2,R=0.3	2618438	1122175	472780	2339588	271642	4398145	6462	1485527	142754	11705637
l= 8,s=0,R=0.3	2618438	1123175	472780	2486788	271642	4398145	6462	1495527	142754	11705637
l= 8,s=1,R=0.3	2618438	1122175	472780	2339588	271642	4398145	6462	1485527	142754	11705637
l= 8,s=2,R=0.3	2618438	1121175	472780	2192388	271642	4398145	6462	1475527	142754	11705637
l=10,s=0,R=0.3	2618438	1122175	472780	2339588	271642	4398145	6462	1485527	142754	11705637
l=10,s=1,R=0.3	2618438	1121175	472780	2192388	271642	4398145	6462	1475527	142754	11705637
l=10,s=2,R=0.3	2618438	1121175	472780	2192388	271642	4398145	6462	1475527	142754	11705637
l= 2,s=0,R=0.9	1481625	894331	249681	3131866	139716	3224231	4807	1371129	94883	6764972
l= 2,s=1,R=0.9	1481625	893331	249681	2984666	139716	3224231	4807	1361129	94883	6764972
l= 2,s=2,R=0.9	1481625	892331	249681	2837466	139716	3224231	4807	1351129	94883	6764972
l= 4,s=0,R=0.9	1481625	890331	249681	2543066	139716	3224231	4807	1331129	94883	6764972
l= 4,s=1,R=0.9	1481625	889331	249681	2395866	139716	3224231	4807	1321129	94883	6764972
l= 4,s=2,R=0.9	1481625	888331	249681	2248666	139716	3224231	4807	1311129	94883	6764972
l= 6,s=0,R=0.9	1481625	889331	249681	2395866	139716	3224231	4807	1321129	94883	6764972
l= 6,s=1,R=0.9	1481625	888331	249681	2248666	139716	3224231	4807	1311129	94883	6764972
l= 6,s=2,R=0.9	1481625	887331	249681	2101466	139716	3224231	4807	1301129	94883	6764972
l= 8,s=0,R=0.9	1481625	888331	249681	2248666	139716	3224231	4807	1311129	94883	6764972
l= 8,s=1,R=0.9	1481625	887331	249681	2101466	139716	3224231	4807	1301129	94883	6764972
l= 8,s=2,R=0.9	1481625	886331	249681	1954266	139716	3224231	4807	1291129	94883	6764972
l=10,s=0,R=0.9	1481625	887331	249681	2101466	139716	3224231	4807	1301129	94883	6764972
l=10,s=1,R=0.9	1481625	886331	249681	1954266	139716	3224231	4807	1291129	94883	6764972
l=10,s=2,R=0.9	1481625	886331	249681	1954266	139716	3224231	4807	1291129	94883	6764972

Table 6
Execution Time for GaAs Offchip Multiplier with Unoptimized Code
(in terms of instruction fetches)
Offchip Delay of 2

bench	ackp	bubblep	fibp	intmmp	perm	puzzlep	queen	quick	sievep	towersp
baseline	3101439	3171079	602121	1761026	431167	19313270	39286	1984792	397121	3585189
l=2,s=0	3101480	3221102	602121	3108450	431190	34661953	40047	2619365	397162	5875212
l=2,s=1	3101480	3220102	602121	3489049	431190	33127777	40047	2609365	397162	5646212
l=2,s=2	3101480	3219102	602121	3341849	431190	31593601	40047	2599365	397162	5417212
l=4,s=0	3101480	3217102	602121	6508450	431190	28525249	40047	2579365	397162	4959212
l=4,s=1	3101480	3216102	602121	2900249	431190	26991073	40047	2569365	397162	4730212
l=4,s=2	3101480	3215102	602121	2753049	431190	25456897	40047	2559365	397162	4501212
l=6,s=0	3101480	3216102	602121	6108450	431190	26991073	40047	2569365	397162	4730212
l=6,s=1	3101480	3215102	602121	2753049	431190	25456897	40047	2559365	397162	4501212
l=6,s=2	3101480	3214102	602121	2605849	431190	23922721	40047	2549365	397162	4272212
l=8,s=0	3101480	3215102	602121	5708450	431190	25456897	40047	2559365	397162	4501212
l=8,s=1	3101480	3214102	602121	2605849	431190	23922721	40047	2549365	397162	4272212
l=8,s=2	3101480	3213102	602121	2458649	431190	22388545	40047	2539365	397162	4043212
l=10,s=0	3101480	3214102	602121	5308450	431190	23922721	40047	2549365	397162	4272212
l=10,s=1	3101480	3213102	602121	2458649	431190	22388545	40047	2539365	397162	4043212
l=10,s=2	3101480	3213102	602121	2458649	431190	22388545	40047	2539365	397162	4043212
l=2,s=0,R=0.3	2429734	2910617	479951	7741229	348160	31839963	32516	2293219	368312	5369185
l=2,s=1,R=0.3	2429734	2909617	479951	3369988	348160	30305787	32516	2283219	368312	5140185
l=2,s=2,R=0.3	2429734	2908617	479951	3222788	348160	28771611	32516	2273219	368312	4911185
l=4,s=0,R=0.3	2429734	2906617	479951	6141229	348160	25703259	32516	2253219	368312	4453185
l=4,s=1,R=0.3	2429734	2905617	479951	2781188	348160	24169083	32516	2243219	368312	4224185
l=4,s=2,R=0.3	2429734	2904617	479951	2633988	348160	22634907	32516	2233219	368312	3995185
l=6,s=0,R=0.3	2429734	2905617	479951	5741229	348160	24169083	32516	2243219	368312	4224185
l=6,s=1,R=0.3	2429734	2904617	479951	2633988	348160	22634907	32516	2233219	368312	3995185
l=6,s=2,R=0.3	2429734	2903617	479951	2486788	348160	21100731	32516	2223219	368312	3766185
l=8,s=0,R=0.3	2429734	2904617	479951	5341229	348160	22634907	32516	2233219	368312	3995185
l=8,s=1,R=0.3	2429734	2903617	479951	2486788	348160	21100731	32516	2223219	368312	3766185
l=8,s=2,R=0.3	2429734	2902617	479951	2339588	348160	19566555	32516	2213219	368312	3537185
l=10,s=0,R=0.3	2429734	2903617	479951	4941229	348160	21100731	32516	2223219	368312	3766185
l=10,s=1,R=0.3	2429734	2902617	479951	2339588	348160	19566555	32516	2213219	368312	3537185
l=10,s=2,R=0.3	2429734	2902617	479951	2339588	348160	19566555	32516	2213219	368312	3537185
l=2,s=0,R=0.9	1086242	2289646	235612	7006787	182099	26195984	17454	1640927	310614	4357132
l=2,s=1,R=0.9	1086242	2288646	235612	3131866	182099	24661808	17454	1630927	310614	4128132
l=2,s=2,R=0.9	1086242	2287646	235612	2984666	182099	23127632	17454	1620927	310614	3899132
l=4,s=0,R=0.9	1086242	2285646	235612	5406787	182099	20059280	17454	1600927	310614	3441132
l=4,s=1,R=0.9	1086242	2284646	235612	2543066	182099	18525104	17454	1590927	310614	3212132
l=4,s=2,R=0.9	1086242	2283646	235612	2395866	182099	16990928	17454	1580927	310614	2983132
l=6,s=0,R=0.9	1086242	2284646	235612	5006787	182099	18525104	17454	1590927	310614	3212132
l=6,s=1,R=0.9	1086242	2283646	235612	2395866	182099	16990928	17454	1580927	310614	2983132
l=6,s=2,R=0.9	1086242	2282646	235612	2248666	182099	15456752	17454	1570927	310614	2754132
l=8,s=0,R=0.9	1086242	2283646	235612	4606787	182099	16990928	17454	1580927	310614	2983132
l=8,s=1,R=0.9	1086242	2282646	235612	2248666	182099	15456752	17454	1570927	310614	2754132
l=8,s=2,R=0.9	1086242	2281646	235612	2101466	182099	13922576	17454	1560927	310614	2525132
l=10,s=0,R=0.9	1086242	2282646	235612	4206787	182099	15456752	17454	1570927	310614	2754132
l=10,s=1,R=0.9	1086242	2281646	235612	2101466	182099	13922576	17454	1560927	310614	2525132
l=10,s=2,R=0.9	1086242	2281646	235612	2101466	182099	13922576	17454	1560927	310614	2525132

Table 7
Execution Time for GaAs Offchip Multiplier with Optimized Code
(in terms of instruction fetches)
Offchip Delay of 4

bench	ackp	bubblep	fibp	intmmp	perm	puzzlep	queen	quick	sievep	towersp
baseline	0	1196574	584330	1761026	337582	5040988	6529	1132966	166649	13265771
i= 2,s=0	0	1247597	584330	3636249	337605	4985102	7290	1657726	166690	14175970
i= 2,s=1	0	1246597	584330	3489049	337605	4985102	7290	1647726	166690	14175970
i= 2,s=2	0	1245597	584330	3341849	337605	4985102	7290	1637726	166690	14175970
i= 4,s=0	0	1243597	584330	3047449	337605	4985102	7290	1617726	166690	14175970
i= 4,s=1	0	1242597	584330	2900249	337605	4985102	7290	1607726	166690	14175970
i= 4,s=2	0	1241597	584330	2753049	337605	4985102	7290	1597726	166690	14175970
i= 6,s=0	0	1242597	584330	2900249	337605	4985102	7290	1607726	166690	14175970
i= 6,s=1	0	1241597	584330	2753049	337605	4985102	7290	1597726	166690	14175970
i= 6,s=2	0	1240597	584330	2605849	337605	4985102	7290	1587726	166690	14175970
i= 8,s=0	0	1241597	584330	2753049	337605	4985102	7290	1597726	166690	14175970
i= 8,s=1	0	1240597	584330	2605849	337605	4985102	7290	1587726	166690	14175970
i= 8,s=2	0	1239597	584330	2458649	337605	4985102	7290	1577726	166690	14175970
i=10,s=0	0	1240597	584330	2605849	337605	4985102	7290	1587726	166690	14175970
i=10,s=1	0	1239597	584330	2458649	337605	4985102	7290	1577726	166690	14175970
i=10,s=2	0	1239597	584330	2458649	337605	4985102	7290	1577726	166690	14175970
i= 2,s=0,R=0.3	0	1130175	472780	3517188	271642	4398145	6462	1555527	142754	11705637
i= 2,s=1,R=0.3	0	1129175	472780	3369988	271642	4398145	6462	1555527	142754	11705637
i= 2,s=2,R=0.3	0	1128175	472780	3222788	271642	4398145	6462	1545527	142754	11705637
i= 4,s=0,R=0.3	0	1126175	472780	2928388	271642	4398145	6462	1525527	142754	11705637
i= 4,s=1,R=0.3	0	1125175	472780	2781188	271642	4398145	6462	1515527	142754	11705637
i= 4,s=2,R=0.3	0	1124175	472780	2633988	271642	4398145	6462	1505527	142754	11705637
i= 6,s=0,R=0.3	0	1125175	472780	2781188	271642	4398145	6462	1515527	142754	11705637
i= 6,s=1,R=0.3	0	1124175	472780	2633988	271642	4398145	6462	1505527	142754	11705637
i= 6,s=2,R=0.3	0	1123175	472780	2486788	271642	4398145	6462	1495527	142754	11705637
i= 8,s=0,R=0.3	0	1124175	472780	2633988	271642	4398145	6462	1505527	142754	11705637
i= 8,s=1,R=0.3	0	1123175	472780	2486788	271642	4398145	6462	1495527	142754	11705637
i= 8,s=2,R=0.3	0	1122175	472780	2339588	271642	4398145	6462	1485527	142754	11705637
i=10,s=0,R=0.3	0	1123175	472780	2486788	271642	4398145	6462	1495527	142754	11705637
i=10,s=1,R=0.3	0	1122175	472780	2339588	271642	4398145	6462	1485527	142754	11705637
i=10,s=2,R=0.3	0	1122175	472780	2339588	271642	4398145	6462	1485527	142754	11705637
i= 2,s=0,R=0.9	0	895331	249681	3279066	139716	3224231	4807	1381129	94883	6764972
i= 2,s=1,R=0.9	0	894331	249681	3131866	139716	3224231	4807	1371129	94883	6764972
i= 2,s=2,R=0.9	0	893331	249681	2984666	139716	3224231	4807	1361129	94883	6764972
i= 4,s=0,R=0.9	0	891331	249681	2690266	139716	3224231	4807	1341129	94883	6764972
i= 4,s=1,R=0.9	0	890331	249681	2543066	139716	3224231	4807	1331129	94883	6764972
i= 4,s=2,R=0.9	0	889331	249681	2395866	139716	3224231	4807	1321129	94883	6764972
i= 6,s=0,R=0.9	0	890331	249681	2543066	139716	3224231	4807	1331129	94883	6764972
i= 6,s=1,R=0.9	0	889331	249681	2395866	139716	3224231	4807	1321129	94883	6764972
i= 6,s=2,R=0.9	0	888331	249681	2248666	139716	3224231	4807	1311129	94883	6764972
i= 8,s=0,R=0.9	0	889331	249681	2395866	139716	3224231	4807	1321129	94883	6764972
i= 8,s=1,R=0.9	0	888331	249681	2248666	139716	3224231	4807	1311129	94883	6764972
i= 8,s=2,R=0.9	0	887331	249681	2101466	139716	3224231	4807	1301129	94883	6764972
i=10,s=0,R=0.9	0	888331	249681	2248666	139716	3224231	4807	1311129	94883	6764972
i=10,s=1,R=0.9	0	887331	249681	2101466	139716	3224231	4807	1301129	94883	6764972
i=10,s=2,R=0.9	0	887331	249681	2101466	139716	3224231	4807	1301129	94883	6764972

Table 8
Execution Time for GaAs Offchip Multiplier with Unoptimized Code
(in terms of instruction fetches)
Offchip Delay of 4

bench	ackp	bubblep	ffbp	intmmp	perm	puzzlep	queen	quick	sievep	towersp
baseline	3101439	3171079	602121	3852427	431167	19318270	39286	1984792	397121	3585189
I= 2,s=0	3101480	3222102	602121	8508450	431190	36196129	40047	2629365	397162	6104212
I= 2,s=1	3101480	3221102	602121	8108450	431190	34661953	40047	2619365	397162	5875212
I= 2,s=2	3101480	3220102	602121	7708450	431190	33127777	40047	2609365	397162	5646212
I= 4,s=0	3101480	3218102	602121	6908450	431190	30059425	40047	2589365	397162	5188212
I= 4,s=1	3101480	3217102	602121	6508450	431190	28525249	40047	2579365	397162	4959212
I= 4,s=2	3101480	3216102	602121	6108450	431190	26991073	40047	2569365	397162	4730212
I= 6,s=0	3101480	3217102	602121	6508450	431190	28525249	40047	2579365	397162	4959212
I= 6,s=1	3101480	3216102	602121	6108450	431190	26991073	40047	2569365	397162	4730212
I= 6,s=2	3101480	3215102	602121	5708450	431190	25456897	40047	2559365	397162	4501212
I= 8,s=0	3101480	3216102	602121	6108450	431190	26991073	40047	2569365	397162	4730212
I= 8,s=1	3101480	3215102	602121	5708450	431190	25456897	40047	2559365	397162	4501212
I= 8,s=2	3101480	3214102	602121	5308450	431190	23922721	40047	2549365	397162	4272212
I=10,s=0	3101480	3215102	602121	5708450	431190	25456897	40047	2559365	397162	4501212
I=10,s=1	3101480	3214102	602121	5308450	431190	23922721	40047	2549365	397162	4272212
I=10,s=2	3101480	3214102	602121	5308450	431190	23922721	40047	2549365	397162	4272212
I= 2,s=0,R=0.3	2429734	2911617	479951	8141229	348160	35374139	32516	2303219	368312	5598185
I= 2,s=1,R=0.3	2429734	2910617	479951	7741229	348160	31839963	32516	2293219	368312	5369185
I= 2,s=2,R=0.3	2429734	2909617	479951	7341229	348160	30305787	32516	2283219	368312	5140185
I= 4,s=0,R=0.3	2429734	2907617	479951	6541229	348160	27237435	32516	2263219	368312	4682185
I= 4,s=1,R=0.3	2429734	2906617	479951	6141229	348160	25703259	32516	2253219	368312	4453185
I= 4,s=2,R=0.3	2429734	2905617	479951	5741229	348160	24169083	32516	2243219	368312	4224185
I= 6,s=0,R=0.3	2429734	2906617	479951	6141229	348160	25703259	32516	2253219	368312	4453185
I= 6,s=1,R=0.3	2429734	2905617	479951	5741229	348160	24169083	32516	2243219	368312	4224185
I= 6,s=2,R=0.3	2429734	2904617	479951	5341229	348160	22634907	32516	2233219	368312	3995185
I= 8,s=0,R=0.3	2429734	2905617	479951	5741229	348160	24169083	32516	2243219	368312	4224185
I= 8,s=1,R=0.3	2429734	2904617	479951	5341229	348160	22634907	32516	2233219	368312	3995185
I= 8,s=2,R=0.3	2429734	2903617	479951	4941229	348160	21100731	32516	2223219	368312	3766185
I=10,s=0,R=0.3	2429734	2904617	479951	5341229	348160	22634907	32516	2233219	368312	3995185
I=10,s=1,R=0.3	2429734	2903617	479951	4941229	348160	21100731	32516	2223219	368312	3766185
I=10,s=2,R=0.3	2429734	2903617	479951	4941229	348160	21100731	32516	2223219	368312	3766185
I= 2,s=0,R=0.9	1086242	2290646	235612	7406787	182099	27730160	17454	1650927	310614	4586132
I= 2,s=1,R=0.9	1086242	2289646	235612	7006787	182099	26195984	17454	1640927	310614	4357132
I= 2,s=2,R=0.9	1086242	2288646	235612	6606787	182099	24661808	17454	1630927	310614	4128132
I= 4,s=0,R=0.9	1086242	2286646	235612	5806787	182099	21593456	17454	1610927	310614	3670132
I= 4,s=1,R=0.9	1086242	2285646	235612	5406787	182099	20059280	17454	1600927	310614	3441132
I= 4,s=2,R=0.9	1086242	2284646	235612	5006787	182099	18525104	17454	1590927	310614	3212132
I= 6,s=0,R=0.9	1086242	2285646	235612	5406787	182099	20059280	17454	1600927	310614	3441132
I= 6,s=1,R=0.9	1086242	2284646	235612	5006787	182099	18525104	17454	1590927	310614	3212132
I= 6,s=2,R=0.9	1086242	2283646	235612	4606787	182099	16990928	17454	1580927	310614	2983132
I= 8,s=0,R=0.9	1086242	2284646	235612	5006787	182099	18525104	17454	1590927	310614	3212132
I= 8,s=1,R=0.9	1086242	2283646	235612	4606787	182099	16990928	17454	1580927	310614	2983132
I= 8,s=2,R=0.9	1086242	2282646	235612	4206787	182099	15456752	17454	1570927	310614	2754132
I=10,s=0,R=0.9	1086242	2283646	235612	4606787	182099	16990928	17454	1580927	310614	2983132
I=10,s=1,R=0.9	1086242	2282646	235612	4206787	182099	15456752	17454	1570927	310614	2754132
I=10,s=2,R=0.9	1086242	2282646	235612	4206787	182099	15456752	17454	1570927	310614	2754132

Table 9
Execution Time for Silicon Onchip Multiplier with Optimized Code (in terms of instruction fetches)

bench	ackp	bubblep	fibp	intmmp	permp	puzzlep	queenp	quickp	sievep	towersp
baseline	3186804	1196574	584330	1761026	337582	5040988	6529	1132966	166649	13265771
l= 1,s=0	3186845	1244597	584330	3194649	337605	4985102	7290	1627726	166690	14175970
l= 1,s=1	3186845	1244097	584330	3121049	337605	4985102	7290	1622726	166690	14175970
l= 1,s=2	3186845	1243597	584330	3047449	337605	4985102	7290	1617726	166690	14175970
l= 2,s=0	3186845	1240597	584330	2605849	337605	4985102	7290	1587726	166690	14175970
l= 2,s=1	3186845	1240097	584330	2532249	337605	4985102	7290	1582726	166690	14175970
l= 2,s=2	3186845	1239597	584330	2458649	337605	4985102	7290	1577726	166690	14175970
l= 4,s=0	3186845	1238597	584330	2311449	337605	4985102	7290	1567726	166690	14175970
l= 4,s=1	3186845	1238097	584330	2237849	337605	4985102	7290	1562726	166690	14175970
l= 4,s=2	3186845	1237597	584330	2164249	337605	4985102	7290	1557726	166690	14175970
l= 6,s=0	3186845	1238097	584330	2237849	337605	4985102	7290	1562726	166690	14175970
l= 6,s=1	3186845	1237597	584330	2164249	337605	4985102	7290	1557726	166690	14175970
l= 6,s=2	3186845	1237097	584330	2090649	337605	4985102	7290	1552726	166690	14175970
l= 8,s=0	3186845	1237597	584330	2164249	337605	4985102	7290	1557726	166690	14175970
l= 8,s=1	3186845	1237097	584330	2090649	337605	4985102	7290	1552726	166690	14175970
l= 8,s=2	3186845	1236597	584330	2017049	337605	4985102	7290	1547726	166690	14175970

Table 10
Execution Time for Silicon Onchip Multiplier with Unoptimized Code (in terms of instruction fetches)

bench	ackp	bubblep	fibp	intmmp	permp	puzzlep	queenp	quickp	sievep	towersp
baseline	3101439	3171079	602121	3852427	431167	19313270	39286	1984792	397121	3585189
l= 1,s=0	3101480	3219102	602121	7308450	431190	31593601	40047	2599365	397162	5417212
l= 1,s=1	3101480	3218602	602121	7108450	431190	30826513	40047	2594365	397162	5302712
l= 1,s=2	3101480	3218102	602121	6908450	431190	30059425	40047	2589365	397162	5188212
l= 2,s=0	3101480	3215102	602121	5708450	431190	25456897	40047	2559365	397162	4501212
l= 2,s=1	3101480	3214602	602121	5508450	431190	24689809	40047	2554365	397162	4386712
l= 2,s=2	3101480	3214102	602121	5308450	431190	23922721	40047	2549365	397162	4272212
l= 4,s=0	3101480	3213102	602121	4908450	431190	22388545	40047	2539365	397162	4043212
l= 4,s=1	3101480	3212602	602121	4708450	431190	21621457	40047	2534365	397162	3928712
l= 4,s=2	3101480	3212102	602121	4508450	431190	20854369	40047	2529365	397162	3814212
l= 6,s=0	3101480	3212602	602121	4708450	431190	21621457	40047	2534365	397162	3928712
l= 6,s=1	3101480	3212102	602121	4508450	431190	20854369	40047	2529365	397162	3814212
l= 6,s=2	3101480	3211602	602121	4308450	431190	20087281	40047	2524365	397162	3699712
l= 8,s=0	3101480	3212102	602121	4508450	431190	20854369	40047	2529365	397162	3814212
l= 8,s=1	3101480	3211602	602121	4308450	431190	20087281	40047	2524365	397162	3699712
l= 8,s=2	3101480	3211102	602121	4108450	431190	19320193	40047	2519365	397162	3585212

Table 11
Execution Time for Silicon Offchip Multiplier with Optimized Code
(in terms of instruction fetches)
Offchip Delay of 1

bench	ackp	bubblep	fibp	intmmp	permp	puzzlep	queenp	quickp	sievep	towersp
baseline	3186804	1196574	584330	1761026	337582	5040988	6529	1132966	166649	13285771
I= 1,s=0	3186845	1254097	584330	4593049	337605	4985102	7290	1722726	166690	14175970
I= 1,s=1	3186845	1253097	584330	4445849	337605	4985102	7290	1712726	166690	14175970
I= 1,s=2	3186845	1252097	584330	4298649	337605	4985102	7290	1702726	166690	14175970
I= 2,s=0	3186845	1246097	584330	3415449	337605	4985102	7290	1642726	166690	14175970
I= 2,s=1	3186845	1245097	584330	3268249	337605	4985102	7290	1632726	166690	14175970
I= 2,s=2	3186845	1244097	584330	3121049	337605	4985102	7290	1622726	166690	14175970
I= 4,s=0	3186845	1242097	584330	2826649	337605	4985102	7290	1602726	166690	14175970
I= 4,s=1	3186845	1241097	584330	2679449	337605	4985102	7290	1592726	166690	14175970
I= 4,s=2	3186845	1240097	584330	2532249	337605	4985102	7290	1582726	166690	14175970
I= 6,s=0	3186845	1241097	584330	2679449	337605	4985102	7290	1592726	166690	14175970
I= 6,s=1	3186845	1240097	584330	2532249	337605	4985102	7290	1582726	166690	14175970
I= 6,s=2	3186845	1239097	584330	2385049	337605	4985102	7290	1572726	166690	14175970
I= 8,s=0	3186845	1240097	584330	2532249	337605	4985102	7290	1582726	166690	14175970
I= 8,s=1	3186845	1239097	584330	2385049	337605	4985102	7290	1572726	166690	14175970
I= 8,s=2	3186845	1238097	584330	2237849	337605	4985102	7290	1562726	166690	14175970
I= 1,s=0,R=0.3	2618438	1136675	472780	4473988	271642	4398145	6462	1630527	142754	11705637
I= 1,s=1,R=0.3	2618438	1135675	472780	4326788	271642	4398145	6462	1620527	142754	11705637
I= 1,s=2,R=0.3	2618438	1134675	472780	4179588	271642	4398145	6462	1610527	142754	11705637
I= 2,s=0,R=0.3	2618438	1128675	472780	3296388	271642	4398145	6462	1550527	142754	11705637
I= 2,s=1,R=0.3	2618438	1127675	472780	3149188	271642	4398145	6462	1540527	142754	11705637
I= 2,s=2,R=0.3	2618438	1126675	472780	3001988	271642	4398145	6462	1530527	142754	11705637
I= 4,s=0,R=0.3	2618438	1124675	472780	2707588	271642	4398145	6462	1510527	142754	11705637
I= 4,s=1,R=0.3	2618438	1123675	472780	2560388	271642	4398145	6462	1500527	142754	11705637
I= 4,s=2,R=0.3	2618438	1122675	472780	2413188	271642	4398145	6462	1490527	142754	11705637
I= 6,s=0,R=0.3	2618438	1123675	472780	2560388	271642	4398145	6462	1500527	142754	11705637
I= 6,s=1,R=0.3	2618438	1122675	472780	2413188	271642	4398145	6462	1490527	142754	11705637
I= 6,s=2,R=0.3	2618438	1121675	472780	2265988	271642	4398145	6462	1480527	142754	11705637
I= 8,s=0,R=0.3	2618438	1122675	472780	2413188	271642	4398145	6462	1490527	142754	11705637
I= 8,s=1,R=0.3	2618438	1121675	472780	2265988	271642	4398145	6462	1480527	142754	11705637
I= 8,s=2,R=0.3	2618438	1120675	472780	2118788	271642	4398145	6462	1470527	142754	11705637
I= 1,s=0,R=0.9	1481625	901831	249681	4235866	139716	3224231	4807	1446129	94883	6764972
I= 1,s=1,R=0.9	1481625	900831	249681	4088666	139716	3224231	4807	1436129	94883	6764972
I= 1,s=2,R=0.9	1481625	899831	249681	3941466	139716	3224231	4807	1426129	94883	6764972
I= 2,s=0,R=0.9	1481625	893831	249681	3058266	139716	3224231	4807	1366129	94883	6764972
I= 2,s=1,R=0.9	1481625	892831	249681	2911066	139716	3224231	4807	1356129	94883	6764972
I= 2,s=2,R=0.9	1481625	891831	249681	2763866	139716	3224231	4807	1346129	94883	6764972
I= 4,s=0,R=0.9	1481625	889831	249681	2469466	139716	3224231	4807	1326129	94883	6764972
I= 4,s=1,R=0.9	1481625	888831	249681	2322266	139716	3224231	4807	1316129	94883	6764972
I= 4,s=2,R=0.9	1481625	887831	249681	2175066	139716	3224231	4807	1306129	94883	6764972
I= 6,s=0,R=0.9	1481625	888831	249681	2322266	139716	3224231	4807	1316129	94883	6764972
I= 6,s=1,R=0.9	1481625	887831	249681	2175066	139716	3224231	4807	1306129	94883	6764972
I= 6,s=2,R=0.9	1481625	886831	249681	2027866	139716	3224231	4807	1296129	94883	6764972
I= 8,s=0,R=0.9	1481625	887831	249681	2175066	139716	3224231	4807	1306129	94883	6764972
I= 8,s=1,R=0.9	1481625	886831	249681	2027866	139716	3224231	4807	1296129	94883	6764972
I= 8,s=2,R=0.9	1481625	885831	249681	1880666	139716	3224231	4807	1286129	94883	6764972

Table 12
Execution Time for Silicon Offchip Multiplier with Unoptimized Code
(in terms of instruction fetches)
Offchip Delay of 1

bench	ackp	bubblep	fibp	intmmp	permp	puzzlep	queenp	quickp	sievep	towersp
baseline	3101439	3171079	602121	3852427	431167	19313270	39286	1984792	397121	3585189
I= 1,s=0	3101480	3228602	602121	11108450	431190	46168273	40047	2694365	397162	7592712
I= 1,s=1	3101480	3227602	602121	10708450	431190	44634097	40047	2684365	397162	7363712
I= 1,s=2	3101480	3226602	602121	10308450	431190	43099921	40047	2674365	397162	7134712
I= 2,s=0	3101480	3220602	602121	7908450	431190	33894865	40047	2614365	397162	5760712
I= 2,s=1	3101480	3219602	602121	7508450	431190	32360689	40047	2604365	397162	5531712
I= 2,s=2	3101480	3218602	602121	7108450	431190	30826513	40047	2594365	397162	5302712
I= 4,s=0	3101480	3216602	602121	6308450	431190	27758161	40047	2574365	397162	4844712
I= 4,s=1	3101480	3215602	602121	5908450	431190	26223985	40047	2564365	397162	4615712
I= 4,s=2	3101480	3214602	602121	5508450	431190	24689809	40047	2554365	397162	4386712
I= 6,s=0	3101480	3215602	602121	5908450	431190	26223985	40047	2564365	397162	4615712
I= 6,s=1	3101480	3214602	602121	5508450	431190	24689809	40047	2554365	397162	4386712
I= 6,s=2	3101480	3213602	602121	5108450	431190	23155633	40047	2544365	397162	4157712
I= 8,s=0	3101480	3214602	602121	5508450	431190	24689809	40047	2554365	397162	4386712
I= 8,s=1	3101480	3213602	602121	5108450	431190	23155633	40047	2544365	397162	4157712
I= 8,s=2	3101480	3212602	602121	4708450	431190	21621457	40047	2534365	397162	3928712
I= 1,s=0,R =0.3	2429734	2918117	479951	10741229	348160	43346283	32516	2368219	368312	7086685
I= 1,s=1,R =0.3	2429734	2917117	479951	10341229	348160	41812107	32516	2358219	368312	6857685
I= 1,s=2,R =0.3	2429734	2916117	479951	9941229	348160	40277931	32516	2348219	368312	6628685
I= 2,s=0,R =0.3	2429734	2910117	479951	7541229	348160	31072875	32516	2288219	368312	5254685
I= 2,s=1,R =0.3	2429734	2909117	479951	7141229	348160	29538699	32516	2278219	368312	5025685
I= 2,s=2,R =0.3	2429734	2908117	479951	6741229	348160	28004523	32516	2268219	368312	4796685
I= 4,s=0,R =0.3	2429734	2906117	479951	5941229	348160	24936171	32516	2248219	368312	4338685
I= 4,s=1,R =0.3	2429734	2905117	479951	5541229	348160	23401995	32516	2238219	368312	4109685
I= 4,s=2,R =0.3	2429734	2904117	479951	5141229	348160	21867819	32516	2228219	368312	3880685
I= 6,s=0,R =0.3	2429734	2905117	479951	5541229	348160	23401995	32516	2238219	368312	4109685
I= 6,s=1,R =0.3	2429734	2904117	479951	5141229	348160	21867819	32516	2228219	368312	3880685
I= 6,s=2,R =0.3	2429734	2903117	479951	4741229	348160	20333643	32516	2218219	368312	3651685
I= 8,s=0,R =0.3	2429734	2904117	479951	5141229	348160	21867819	32516	2228219	368312	3880685
I= 8,s=1,R =0.3	2429734	2903117	479951	4741229	348160	20333643	32516	2218219	368312	3651685
I= 8,s=2,R =0.3	2429734	2902117	479951	4341229	348160	18799467	32516	2208219	368312	3422685
I= 1,s=0,R =0.9	1086242	2297146	235612	10006787	182099	37702304	17454	1715927	310614	6074632
I= 1,s=1,R =0.9	1086242	2296146	235612	9606787	182099	36168128	17454	1705927	310614	5845632
I= 1,s=2,R =0.9	1086242	2295146	235612	9206787	182099	34633952	17454	1695927	310614	5616632
I= 2,s=0,R =0.9	1086242	2289146	235612	6806787	182099	25428896	17454	1635927	310614	4242632
I= 2,s=1,R =0.9	1086242	2288146	235612	6406787	182099	23894720	17454	1625927	310614	4013632
I= 2,s=2,R =0.9	1086242	2287146	235612	6006787	182099	22360544	17454	1615927	310614	3784632
I= 4,s=0,R =0.9	1086242	2285146	235612	5206787	182099	19292192	17454	1595927	310614	3326632
I= 4,s=1,R =0.9	1086242	2284146	235612	4806787	182099	17758016	17454	1585927	310614	3097632
I= 4,s=2,R =0.9	1086242	2283146	235612	4406787	182099	16223840	17454	1575927	310614	2868632
I= 6,s=0,R =0.9	1086242	2284146	235612	4806787	182099	17758016	17454	1585927	310614	3097632
I= 6,s=1,R =0.9	1086242	2283146	235612	4406787	182099	16223840	17454	1575927	310614	2868632
I= 6,s=2,R =0.9	1086242	2282146	235612	4006787	182099	14689664	17454	1565927	310614	2639632
I= 8,s=0,R =0.9	1086242	2283146	235612	4406787	182099	16223840	17454	1575927	310614	2868632
I= 8,s=1,R =0.9	1086242	2282146	235612	4006787	182099	14689664	17454	1565927	310614	2639632
I= 8,s=2,R =0.9	1086242	2281146	235612	3606787	182099	13155488	17454	1555927	310614	2410632

Table 13
Execution Time for Silicon Offchip Multiplier with Optimized Code
(in terms of instruction fetches)
Offchip Delay of 2

bench	ackp	bubblep	fibp	intmmp	permp	puzzlep	queenp	quickp	sievep	towersp
baseline	3186804	1196574	584330	1761026	337582	5040988	6529	1132966	166649	13265771
I= 1,s=0	3186845	1254597	584330	4666649	337605	4985102	7290	1727726	166690	14175970
I= 1,s=1	3186845	1253597	584330	4519449	337605	4985102	7290	1717726	166690	14175970
I= 1,s=2	3186845	1252597	584330	4372249	337605	4985102	7290	1707726	166690	14175970
I= 2,s=0	3186845	1246597	584330	3489049	337605	4985102	7290	1647726	166690	14175970
I= 2,s=1	3186845	1245597	584330	3341849	337605	4985102	7290	1637726	166690	14175970
I= 2,s=2	3186845	1244597	584330	3194649	337605	4985102	7290	1627726	166690	14175970
I= 4,s=0	3186845	1242597	584330	2900249	337605	4985102	7290	1607726	166690	14175970
I= 4,s=1	3186845	1241597	584330	2753049	337605	4985102	7290	1597726	166690	14175970
I= 4,s=2	3186845	1240597	584330	2605849	337605	4985102	7290	1587726	166690	14175970
I= 6,s=0	3186845	1241597	584330	2753049	337605	4985102	7290	1597726	166690	14175970
I= 6,s=1	3186845	1240597	584330	2605849	337605	4985102	7290	1587726	166690	14175970
I= 6,s=2	3186845	1239597	584330	2458649	337605	4985102	7290	1577726	166690	14175970
I= 8,s=0	3186845	1240597	584330	2605849	337605	4985102	7290	1587726	166690	14175970
I= 8,s=1	3186845	1239597	584330	2458649	337605	4985102	7290	1577726	166690	14175970
I= 8,s=2	3186845	1238597	584330	2311449	337605	4985102	7290	1567726	166690	14175970
I= 1,s=0,R=0.3	2618438	1137175	472780	4547588	271642	4398145	6462	1635527	142754	11705637
I= 1,s=1,R=0.3	2618438	1136175	472780	4400388	271642	4398145	6462	1625527	142754	11705637
I= 1,s=2,R=0.3	2618438	1135175	472780	4253188	271642	4398145	6462	1615527	142754	11705637
I= 2,s=0,R=0.3	2618438	1129175	472780	3369988	271642	4398145	6462	1555527	142754	11705637
I= 2,s=1,R=0.3	2618438	1128175	472780	3222788	271642	4398145	6462	1545527	142754	11705637
I= 2,s=2,R=0.3	2618438	1127175	472780	3075588	271642	4398145	6462	1535527	142754	11705637
I= 4,s=0,R=0.3	2618438	1125175	472780	2781188	271642	4398145	6462	1515527	142754	11705637
I= 4,s=1,R=0.3	2618438	1124175	472780	2633988	271642	4398145	6462	1505527	142754	11705637
I= 4,s=2,R=0.3	2618438	1123175	472780	2486788	271642	4398145	6462	1495527	142754	11705637
I= 6,s=0,R=0.3	2618438	1124175	472780	2633988	271642	4398145	6462	1505527	142754	11705637
I= 6,s=1,R=0.3	2618438	1123175	472780	2486788	271642	4398145	6462	1495527	142754	11705637
I= 6,s=2,R=0.3	2618438	1122175	472780	2339588	271642	4398145	6462	1485527	142754	11705637
I= 8,s=0,R=0.3	2618438	1123175	472780	2486788	271642	4398145	6462	1495527	142754	11705637
I= 8,s=1,R=0.3	2618438	1122175	472780	2339588	271642	4398145	6462	1485527	142754	11705637
I= 8,s=2,R=0.3	2618438	1121175	472780	2192388	271642	4398145	6462	1475527	142754	11705637
I= 1,s=0,R=0.9	1481625	902331	249681	4309466	139716	3224231	4807	1451129	94883	6764972
I= 1,s=1,R=0.9	1481625	901331	249681	4162266	139716	3224231	4807	1441129	94883	6764972
I= 1,s=2,R=0.9	1481625	900331	249681	4015066	139716	3224231	4807	1431129	94883	6764972
I= 2,s=0,R=0.9	1481625	894331	249681	3131866	139716	3224231	4807	1371129	94883	6764972
I= 2,s=1,R=0.9	1481625	893331	249681	2984666	139716	3224231	4807	1361129	94883	6764972
I= 2,s=2,R=0.9	1481625	892331	249681	2837466	139716	3224231	4807	1351129	94883	6764972
I= 4,s=0,R=0.9	1481625	890331	249681	2543066	139716	3224231	4807	1331129	94883	6764972
I= 4,s=1,R=0.9	1481625	889331	249681	2395866	139716	3224231	4807	1321129	94883	6764972
I= 4,s=2,R=0.9	1481625	888331	249681	2248666	139716	3224231	4807	1311129	94883	6764972
I= 6,s=0,R=0.9	1481625	889331	249681	2395866	139716	3224231	4807	1321129	94883	6764972
I= 6,s=1,R=0.9	1481625	888331	249681	2248666	139716	3224231	4807	1311129	94883	6764972
I= 6,s=2,R=0.9	1481625	887331	249681	2101466	139716	3224231	4807	1301129	94883	6764972
I= 8,s=0,R=0.9	1481625	888331	249681	2248666	139716	3224231	4807	1311129	94883	6764972
I= 8,s=1,R=0.9	1481625	887331	249681	2101466	139716	3224231	4807	1301129	94883	6764972
I= 8,s=2,R=0.9	1481625	886331	249681	1954266	139716	3224231	4807	1291129	94883	6764972

Table 14
Execution Time for Silicon Offchip Multiplier with Unoptimized Code
(in terms of instruction fetches)
Offchip Delay of 2

bench	ackp	bubblep	fibp	intmmp	permp	puzzlep	queenp	quickp	sievep	towersp
baseline	3101439	3171079	602121	3852427	431167	19313270	39286	1984792	397121	3585189
I= 1,s=0	3101480	3229102	602121	11308450	431190	46935361	40047	2699365	397162	7707212
I= 1,s=1	3101480	3228102	602121	10908450	431190	45401185	40047	2689365	397162	7478212
I= 1,s=2	3101480	3227102	602121	10508450	431190	43867009	40047	2679365	397162	7249212
I= 2,s=0	3101480	3221102	602121	8108450	431190	34661953	40047	2619365	397162	5875212
I= 2,s=1	3101480	3220102	602121	7708450	431190	33127777	40047	2609365	397162	5646212
I= 2,s=2	3101480	3219102	602121	7308450	431190	31593601	40047	2599365	397162	5417212
I= 4,s=0	3101480	3217102	602121	6508450	431190	28525249	40047	2579365	397162	4959212
I= 4,s=1	3101480	3216102	602121	6108450	431190	26991073	40047	2569365	397162	4730212
I= 4,s=2	3101480	3215102	602121	5708450	431190	25456897	40047	2559365	397162	4501212
I= 6,s=0	3101480	3216102	602121	6108450	431190	26991073	40047	2569365	397162	4730212
I= 6,s=1	3101480	3215102	602121	5708450	431190	25456897	40047	2559365	397162	4501212
I= 6,s=2	3101480	3214102	602121	5308450	431190	23922721	40047	2549365	397162	4272212
I= 8,s=0	3101480	3215102	602121	5708450	431190	25456897	40047	2559365	397162	4501212
I= 8,s=1	3101480	3214102	602121	5308450	431190	23922721	40047	2549365	397162	4272212
I= 8,s=2	3101480	3213102	602121	4908450	431190	22388545	40047	2539365	397162	4043212
I= 1,s=0,R =0.3	2429734	2918617	479951	10941229	348160	44113371	32516	2373219	368312	7201185
I= 1,s=1,R =0.3	2429734	2917617	479951	10541229	348160	42579195	32516	2363219	368312	6972185
I= 1,s=2,R =0.3	2429734	2916617	479951	10141229	348160	41045019	32516	2353219	368312	6743185
I= 2,s=0,R =0.3	2429734	2910617	479951	7741229	348160	31839963	32516	2293219	368312	5369185
I= 2,s=1,R =0.3	2429734	2909617	479951	7341229	348160	30305787	32516	2283219	368312	5140185
I= 2,s=2,R =0.3	2429734	2908617	479951	6941229	348160	28771611	32516	2273219	368312	4911185
I= 4,s=0,R =0.3	2429734	2906617	479951	6141229	348160	25703259	32516	2253219	368312	4453185
I= 4,s=1,R =0.3	2429734	2905617	479951	5741229	348160	24169083	32516	2243219	368312	4224185
I= 4,s=2,R =0.3	2429734	2904617	479951	5341229	348160	22634907	32516	2233219	368312	3995185
I= 6,s=0,R =0.3	2429734	2905617	479951	5741229	348160	24169083	32516	2243219	368312	4224185
I= 6,s=1,R =0.3	2429734	2904617	479951	5341229	348160	22634907	32516	2233219	368312	3995185
I= 6,s=2,R =0.3	2429734	2903617	479951	4941229	348160	21100731	32516	2223219	368312	3766185
I= 8,s=0,R =0.3	2429734	2904617	479951	5341229	348160	22634907	32516	2233219	368312	3995185
I= 8,s=1,R =0.3	2429734	2903617	479951	4941229	348160	21100731	32516	2223219	368312	3766185
I= 8,s=2,R =0.3	2429734	2902617	479951	4541229	348160	19566555	32516	2213219	368312	3537185
I= 1,s=0,R =0.9	1086242	2297646	235612	10206787	182099	38469392	17454	1720927	310614	6189132
I= 1,s=1,R =0.9	1086242	2296646	235612	9806787	182099	36935216	17454	1710927	310614	5960132
I= 1,s=2,R =0.9	1086242	2295646	235612	9406787	182099	35401040	17454	1700927	310614	5731132
I= 2,s=0,R =0.9	1086242	2289646	235612	7006787	182099	26195984	17454	1640927	310614	4357132
I= 2,s=1,R =0.9	1086242	2288646	235612	6606787	182099	24661808	17454	1630927	310614	4128132
I= 2,s=2,R =0.9	1086242	2287646	235612	6206787	182099	23127632	17454	1620927	310614	3899132
I= 4,s=0,R =0.9	1086242	2285646	235612	5406787	182099	20059280	17454	1600927	310614	3441132
I= 4,s=1,R =0.9	1086242	2284646	235612	5006787	182099	18525104	17454	1590927	310614	3212132
I= 4,s=2,R =0.9	1086242	2283646	235612	4606787	182099	16990928	17454	1580927	310614	2983132
I= 6,s=0,R =0.9	1086242	2284646	235612	5006787	182099	18525104	17454	1590927	310614	3212132
I= 6,s=1,R =0.9	1086242	2283646	235612	4606787	182099	16990928	17454	1580927	310614	2983132
I= 6,s=2,R =0.9	1086242	2282646	235612	4206787	182099	15456752	17454	1570927	310614	2754132
I= 8,s=0,R =0.9	1086242	2283646	235612	4606787	182099	16990928	17454	1580927	310614	2983132
I= 8,s=1,R =0.9	1086242	2282646	235612	4206787	182099	15456752	17454	1570927	310614	2754132
I= 8,s=2,R =0.9	1086242	2281646	235612	3806787	182099	13922576	17454	1560927	310614	2525132

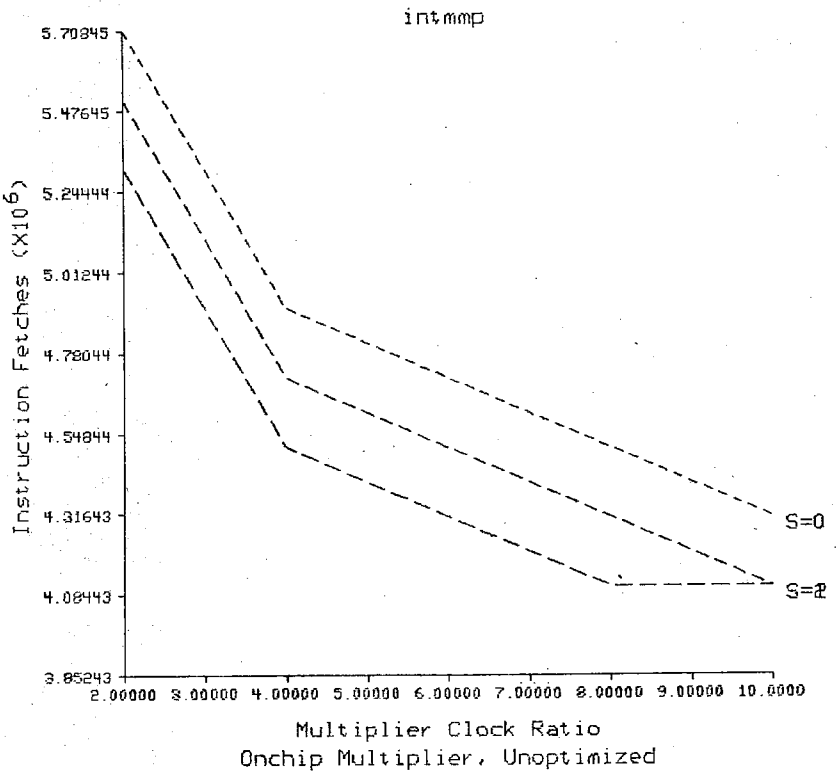


Figure 5.2 Execution Time of *Intmm* with an On-chip Bit-serial Multiplier with GaAs Parameters

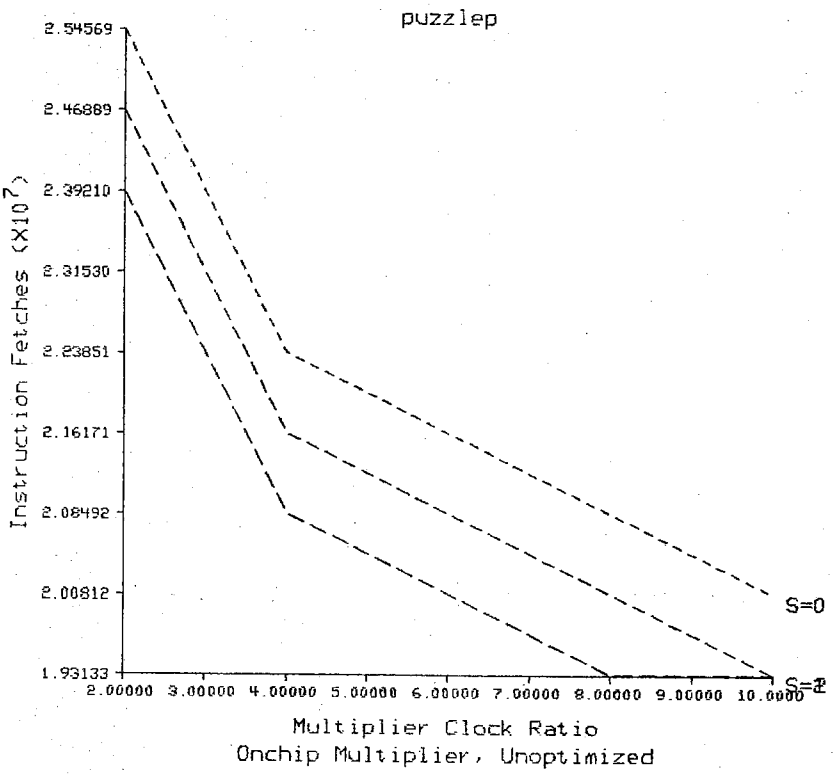


Figure 5.3 Execution Time of *Puzzle* with an On-chip Bit-serial Multiplier with GaAs Parameters

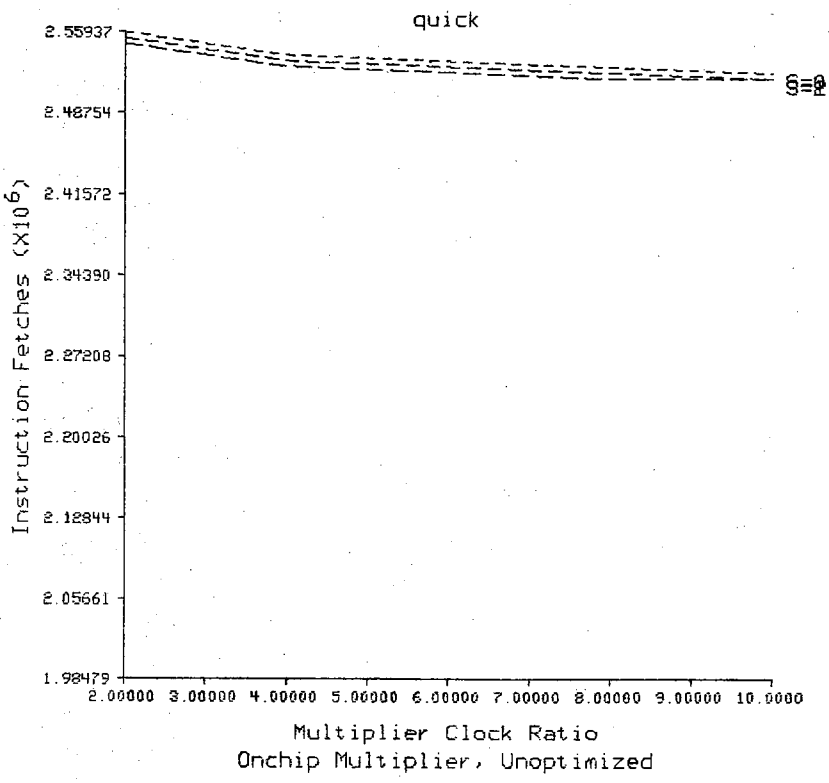


Figure 5.4 Execution Time of *Quick* with an On-chip Bit-serial Multiplier with GaAs Parameters

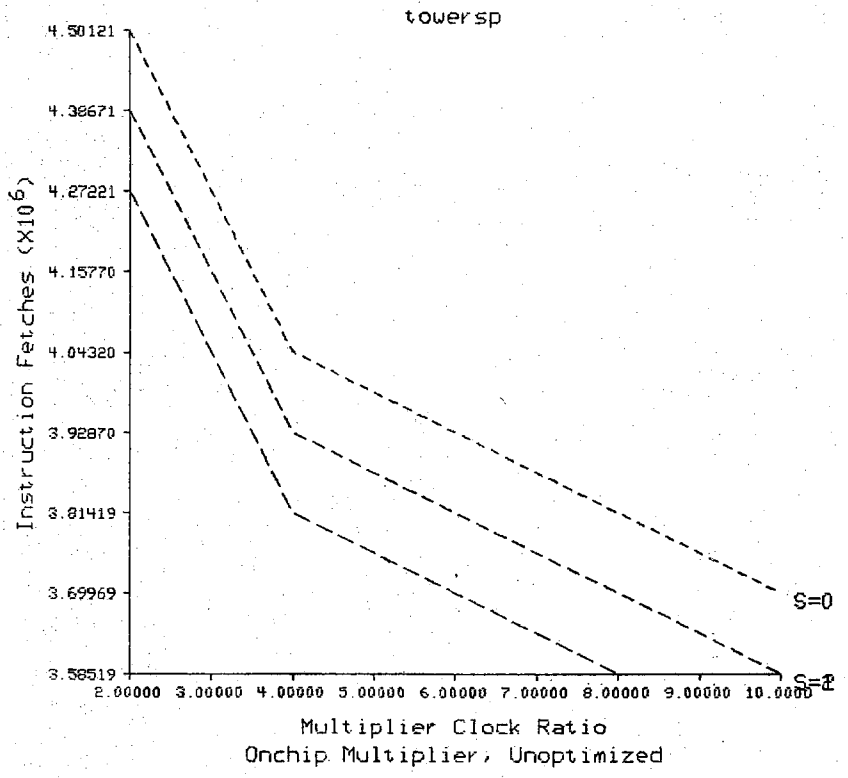


Figure 5.5 Execution Time of *Towers* with an On-chip Bit-serial Multiplier with GaAs Parameters

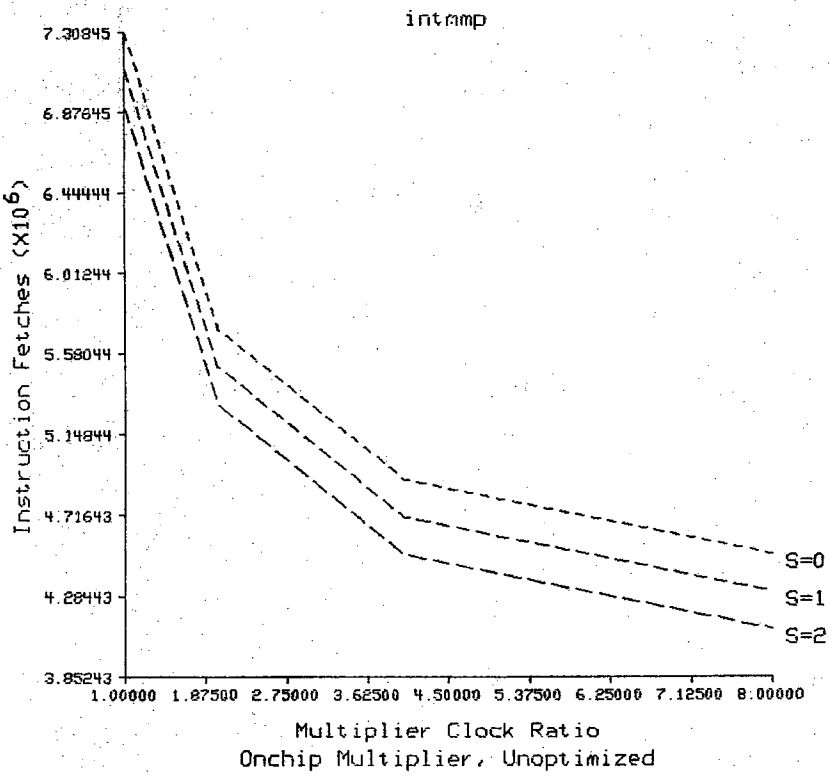


Figure 5.6 Execution Time of *Intmm* with an On-chip Bit-serial Multiplier with Silicon Parameters

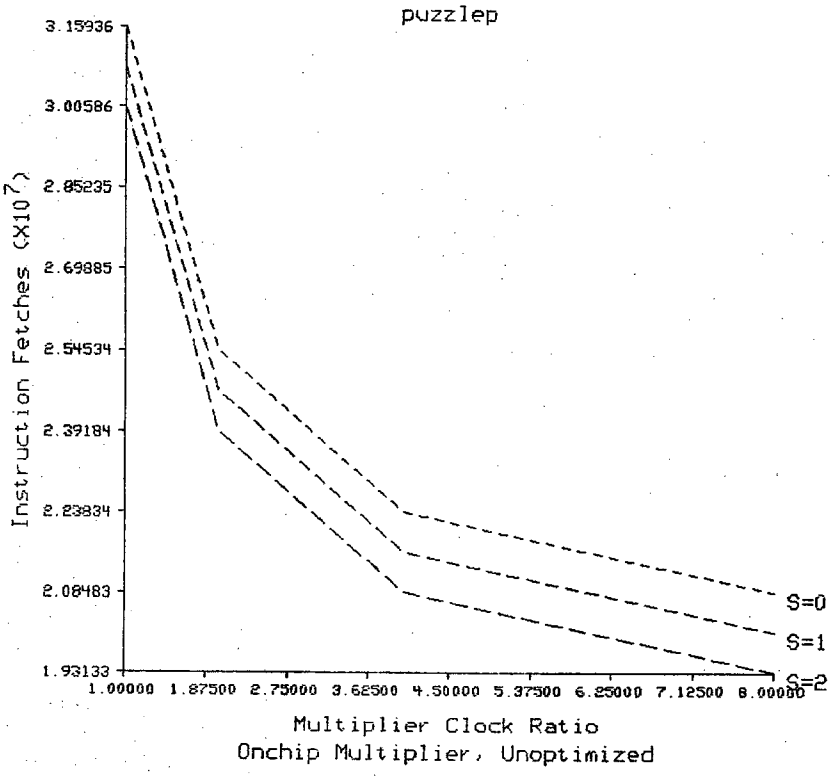


Figure 5.7 Execution Time of *Puzzle* with an On-chip Bit-serial Multiplier with Silicon Parameters

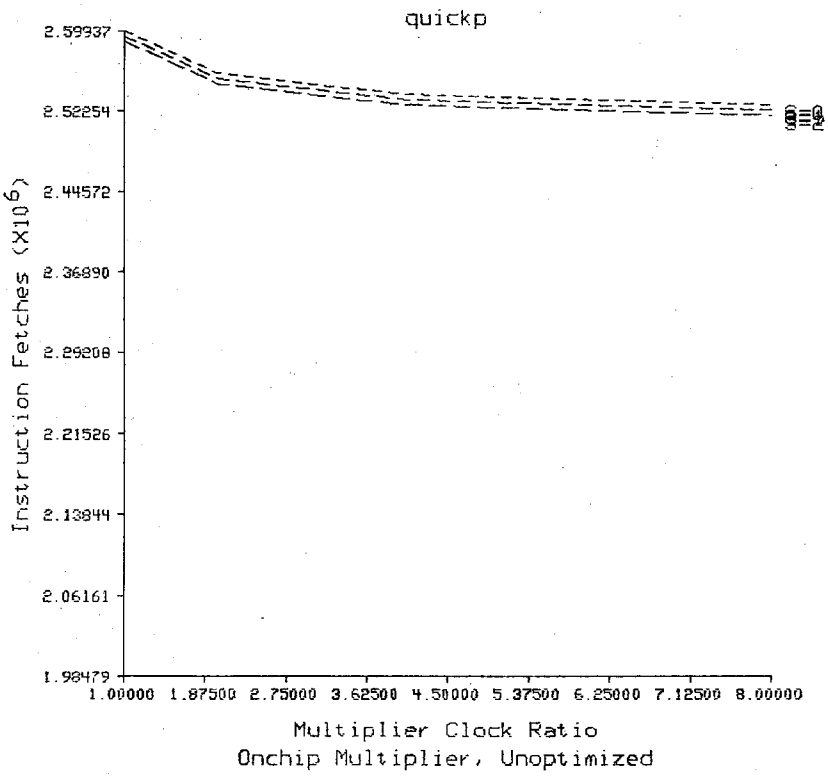


Figure 5.8 Execution Time of *Quick* with an On-chip Bit-serial Multiplier with Silicon Parameters

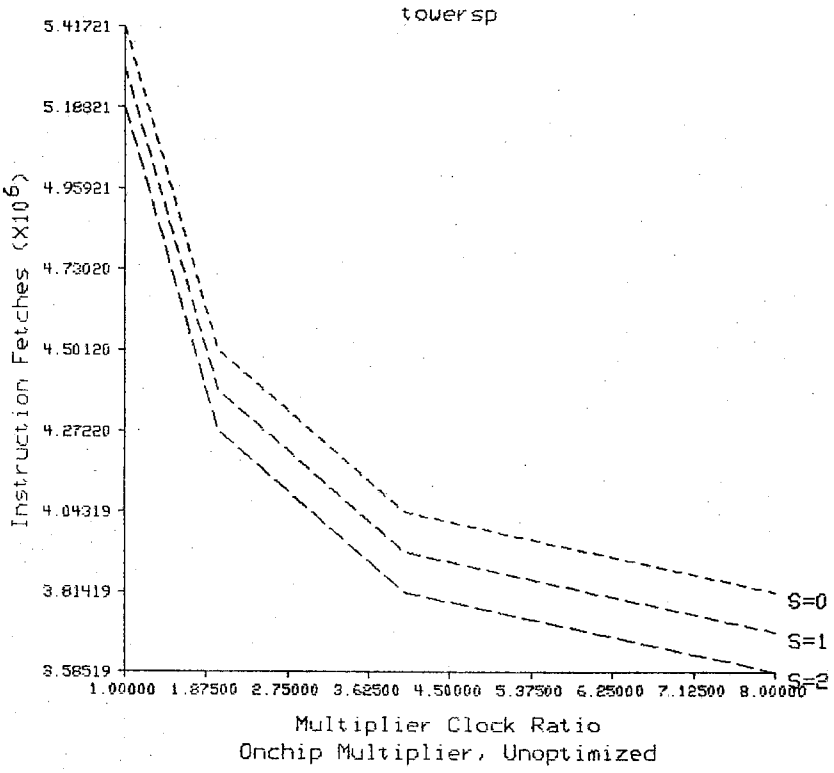


Figure 5.9 Execution Time of *Towers* with an On-chip Bit-serial Multiplier with Silicon Parameters

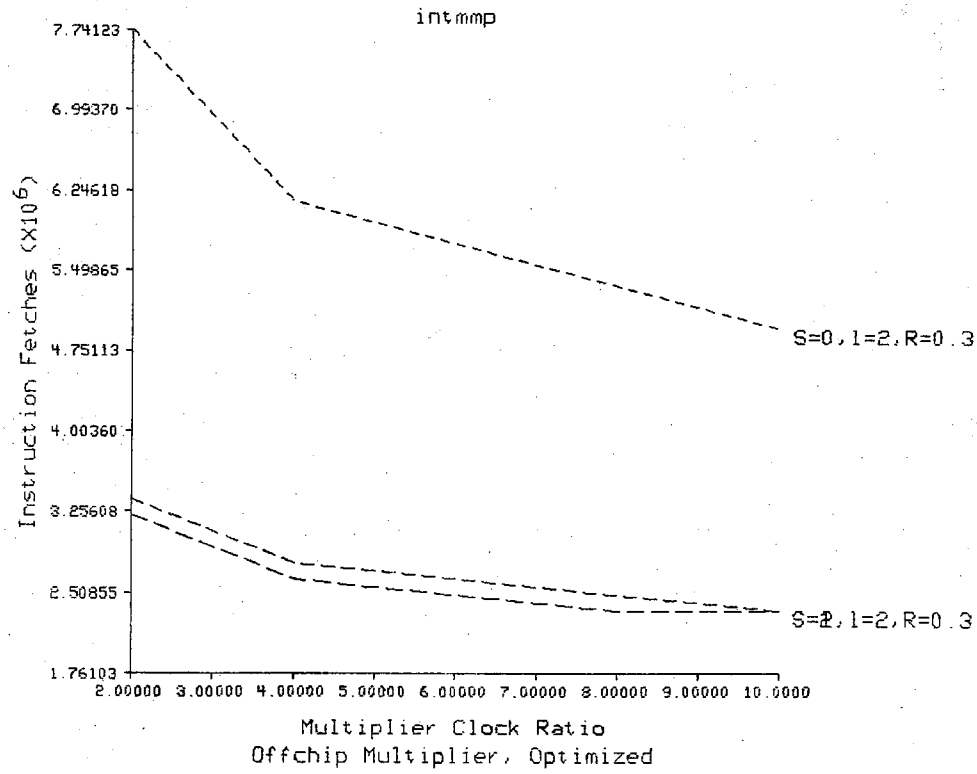


Figure 5.10 Execution Time of *Intmm* with an Off-chip Bit-serial Multiplier with 30% Register Overflow and GaAs Parameters

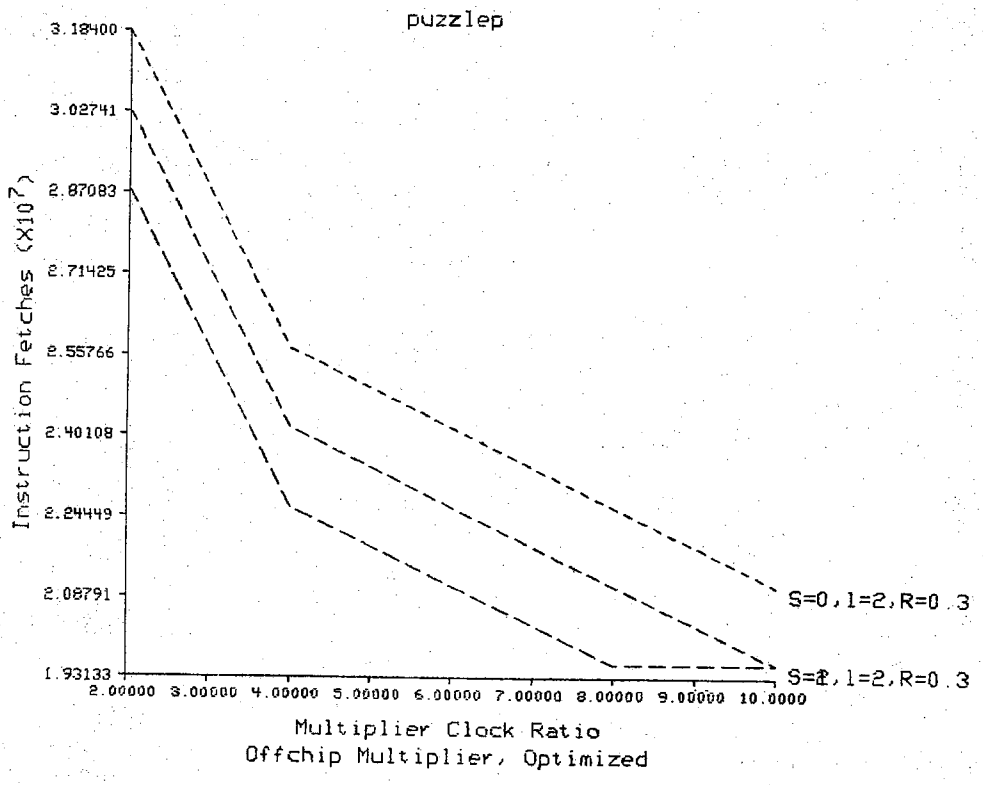


Figure 5.11 Execution Time of *Puzzle* with an Off-chip Bit-serial Multiplier with 30% Register Overflow and GaAs Parameters

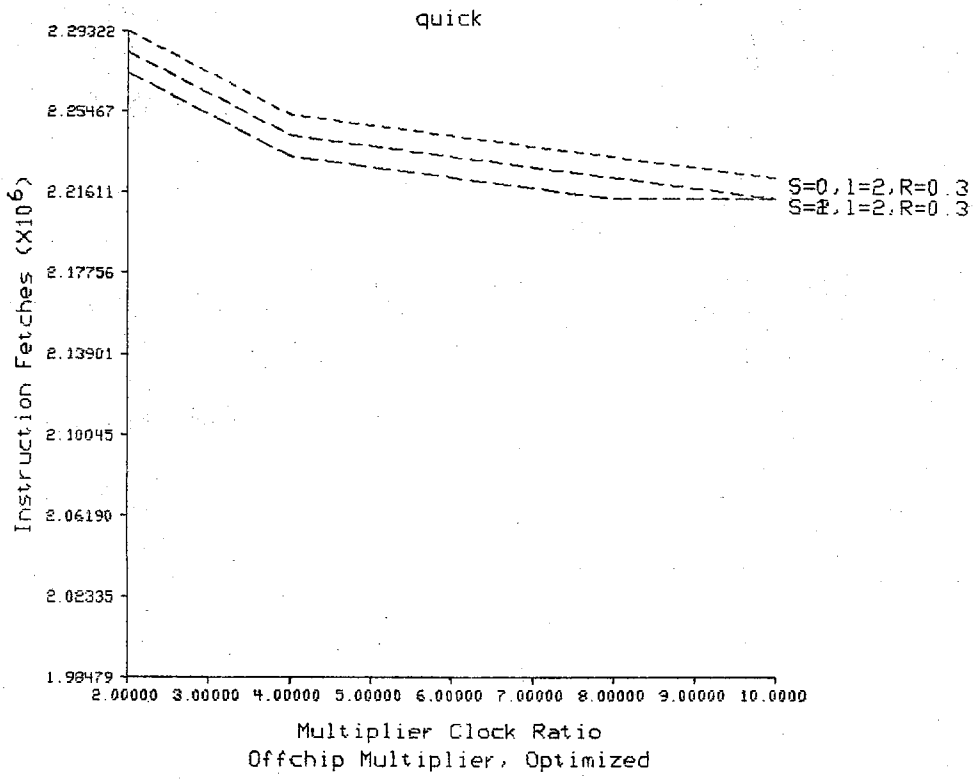


Figure 5.12 Execution Time of *Quick* with an Off-chip Bit-serial Multiplier with 30% Register Overflow and GaAs Parameters

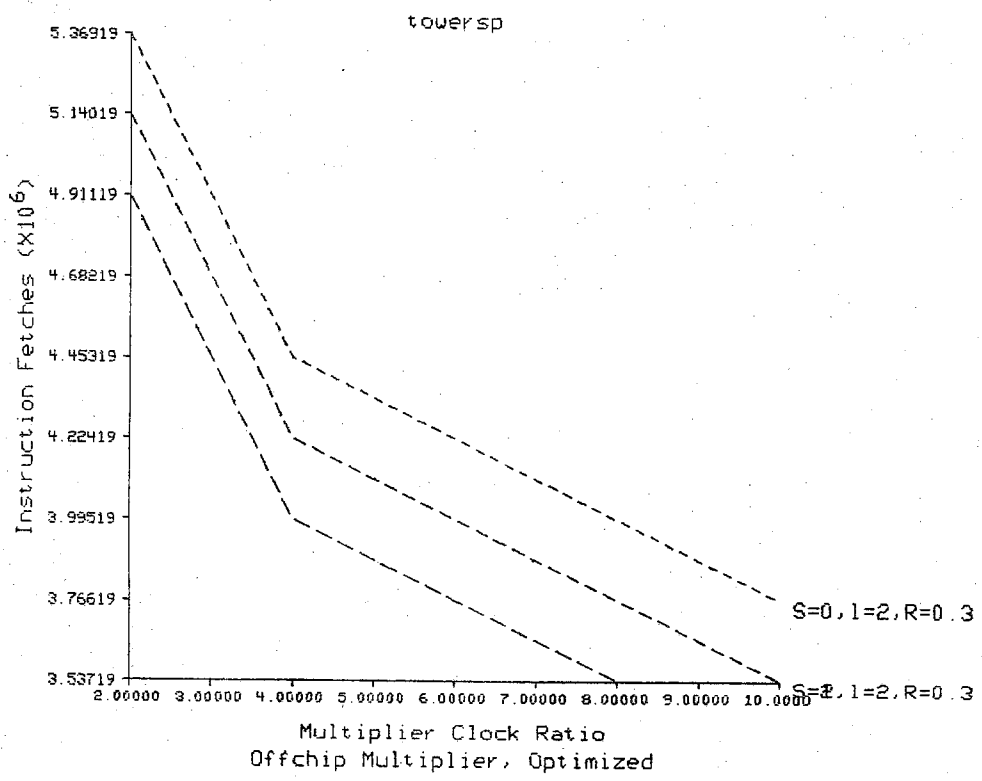


Figure 5.13 Execution Time of *Towers* with an Off-chip Bit-serial Multiplier with 30% Register Overflow and GaAs Parameters

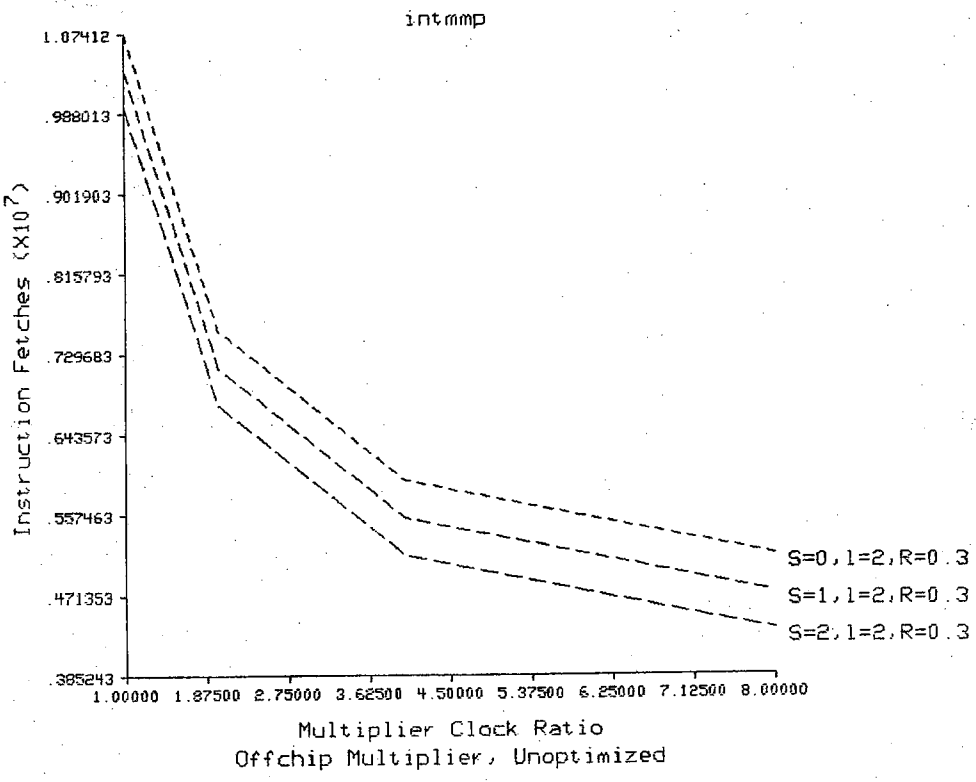


Figure 5.14 Execution Time of *Intmm* with an Off-chip Bit-serial Multiplier with 30% Register Overflow and Silicon Parameters

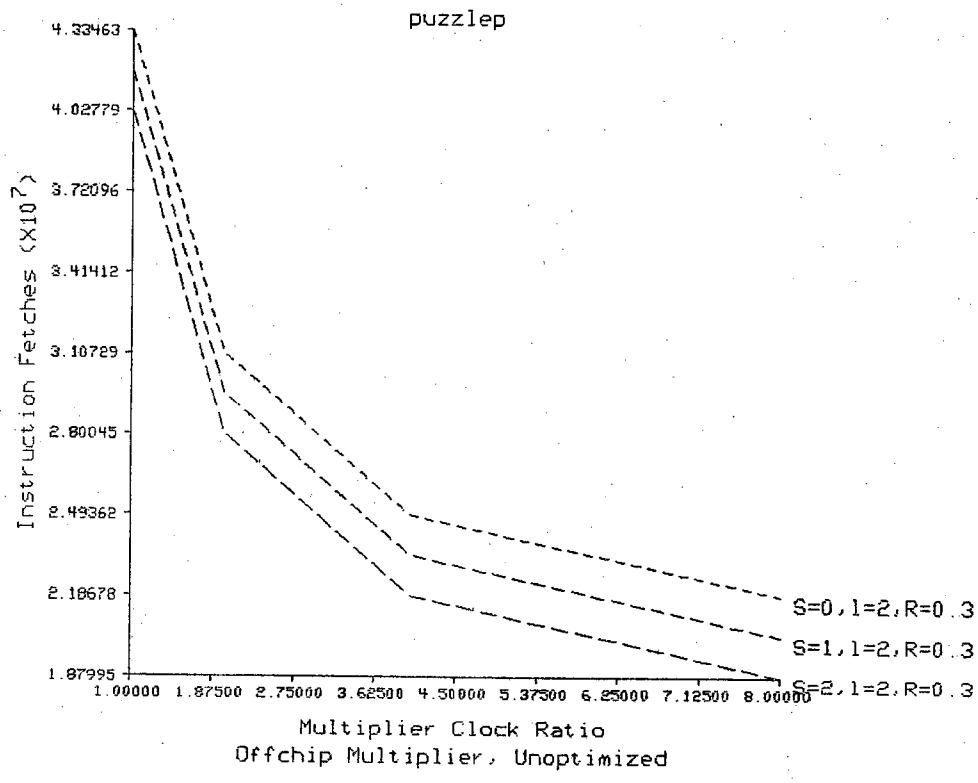


Figure 5.15 Execution Time of *Puzzle* with an Off-chip Bit-serial Multiplier with 30% Register Overflow and Silicon Parameters

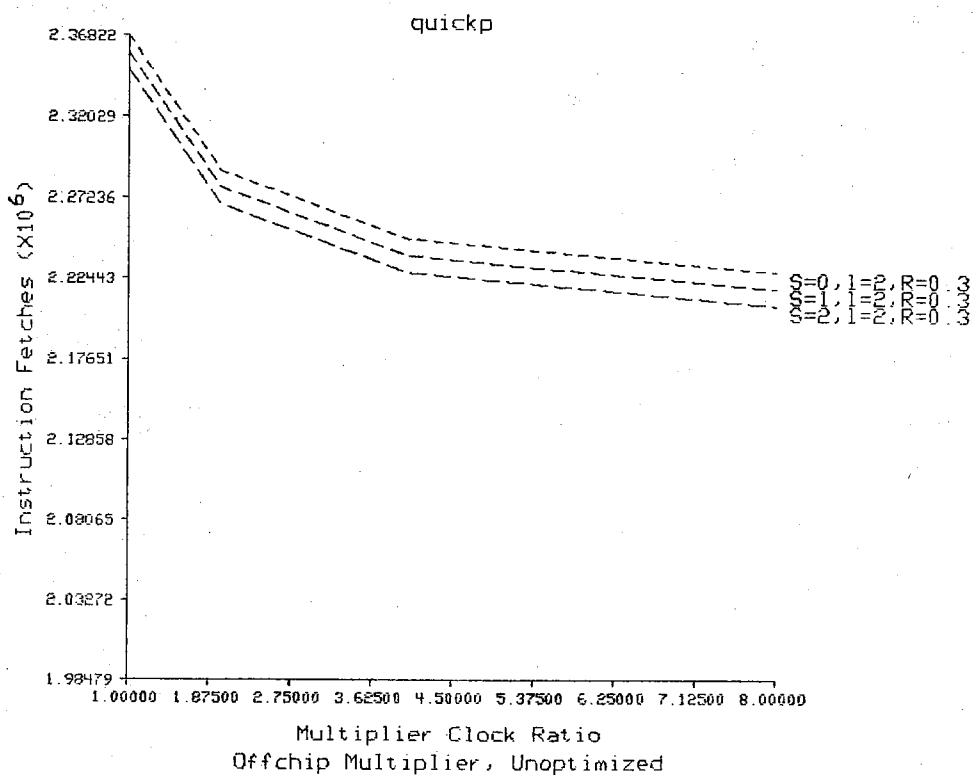


Figure 5.16 Execution Time of *Quick* with an Off-chip Bit-serial Multiplier with 30% Register Overflow and Silicon Parameters

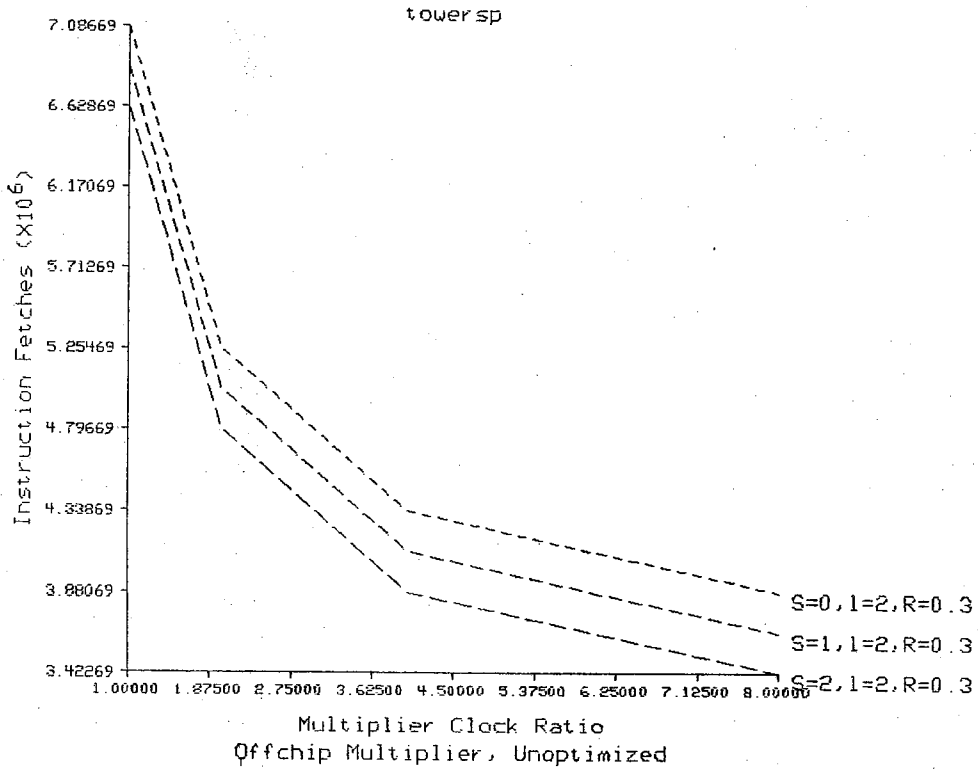


Figure 5.17 Execution Time of *Towers* with an Off-chip Bit-serial Multiplier with 30% Register Overflow and Silicon Parameters

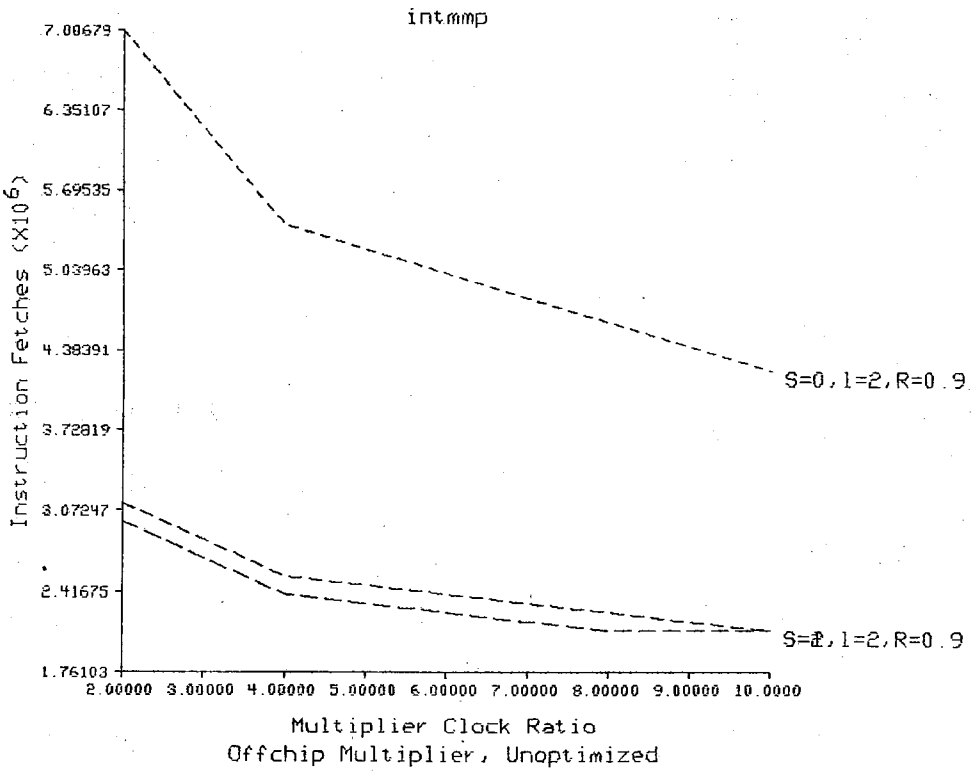


Figure 5.18 Execution Time of *Intmm* with an Off-chip Bit-serial Multiplier with 90% Register Overflow and GaAs Parameters

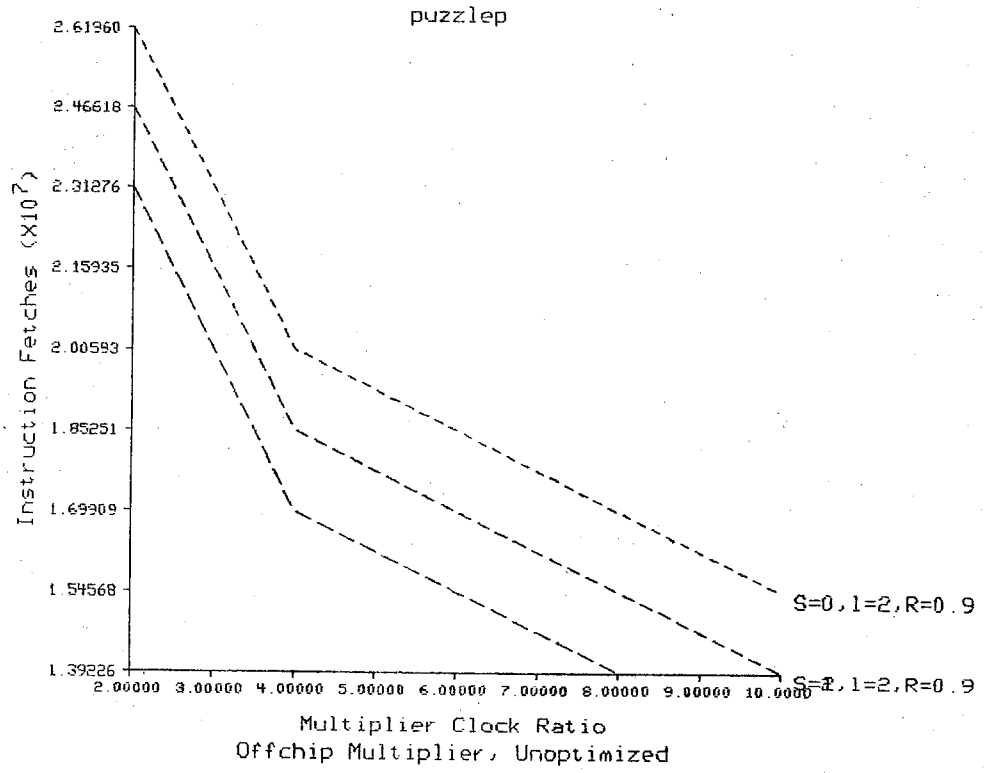


Figure 5.19 Execution Time of *Puzzle* with an Off-chip Bit-serial Multiplier with 90% Register Overflow and GaAs Parameters

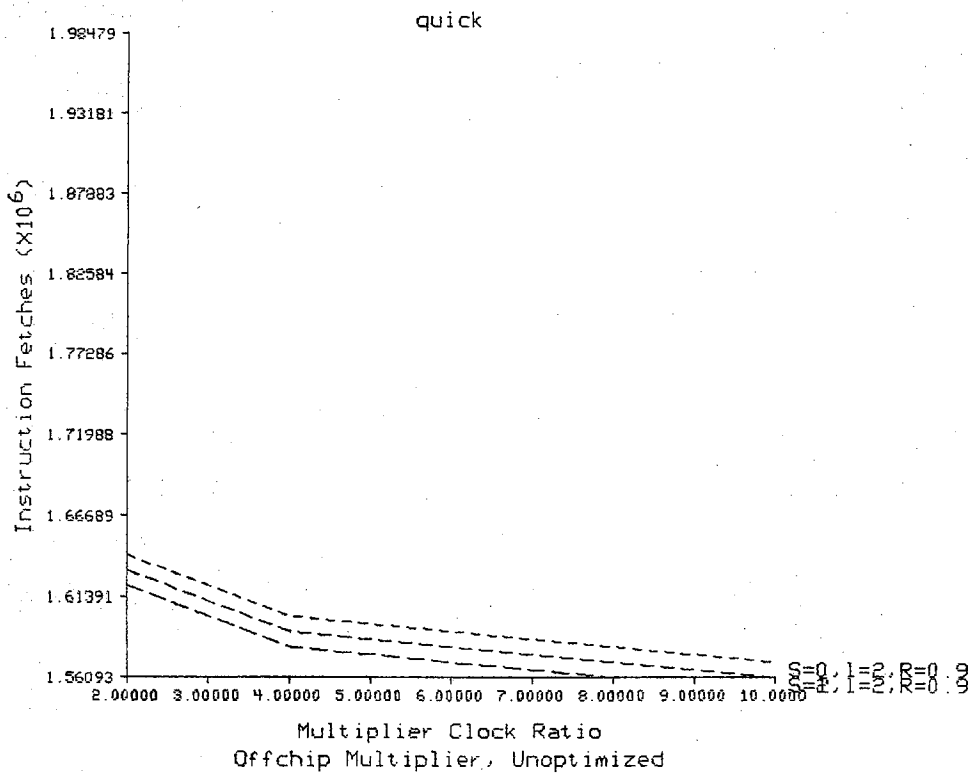


Figure 5.20 Execution Time of *Quick* with an Off-chip Bit-serial Multiplier with 90% Register Overflow and GaAs Parameters

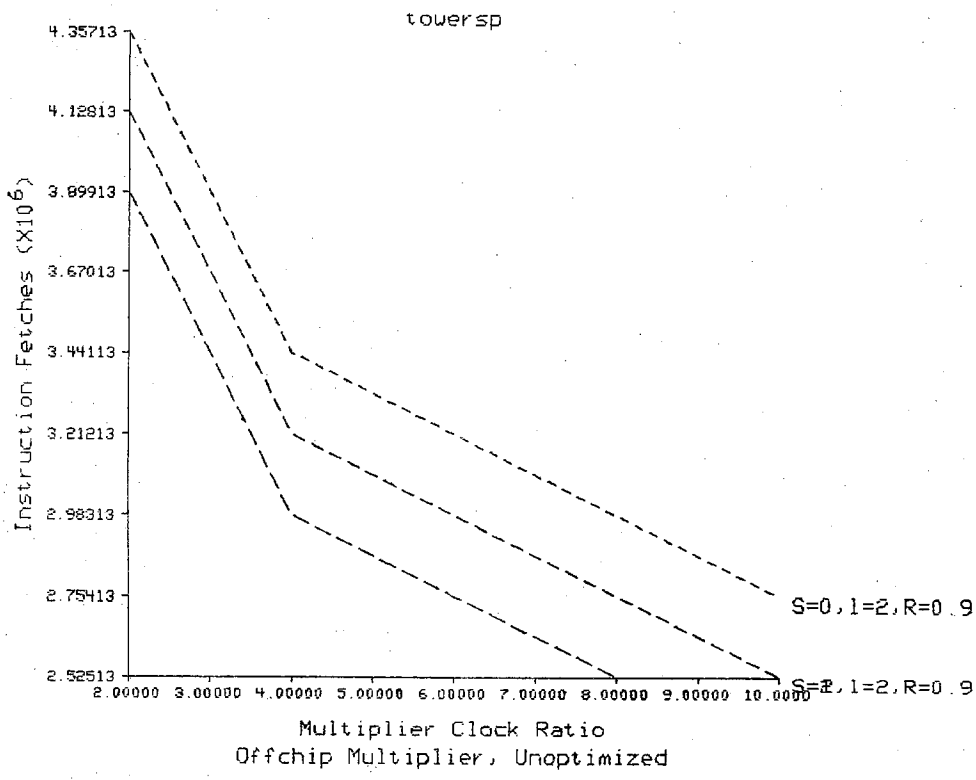


Figure 5.21 Execution Time of *Towers* with an Off-chip Bit-serial Multiplier with 90% Register Overflow and GaAs Parameters

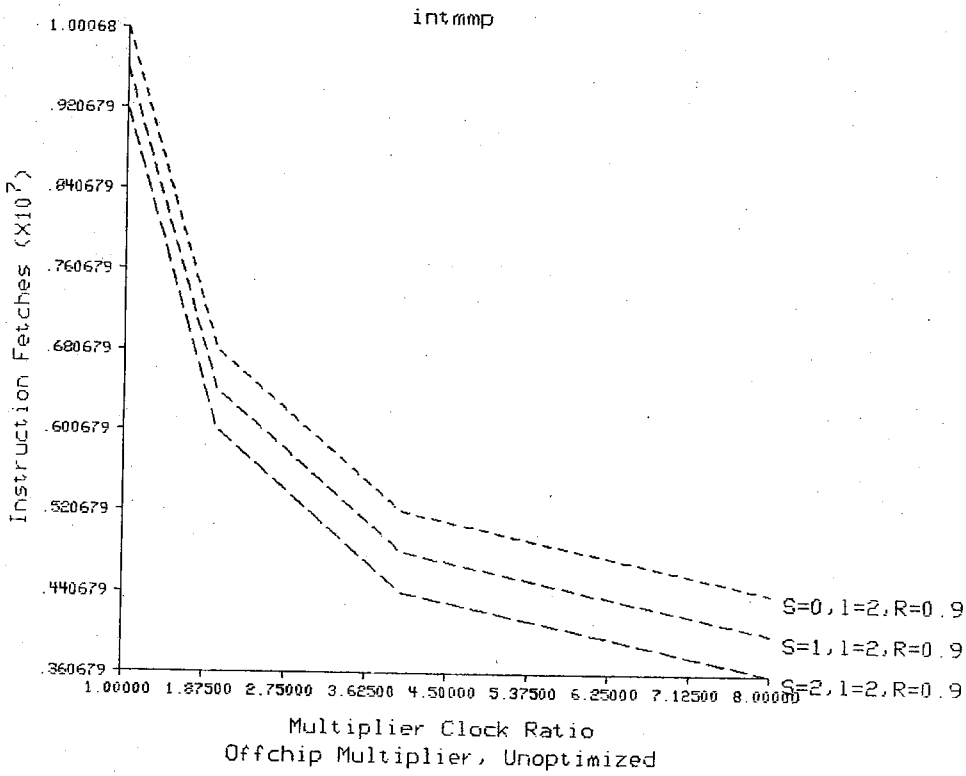


Figure 5.22 Execution Time of *Intmm* with an Off-chip Bit-serial Multiplier with 90% Register Overflow and Silicon Parameters

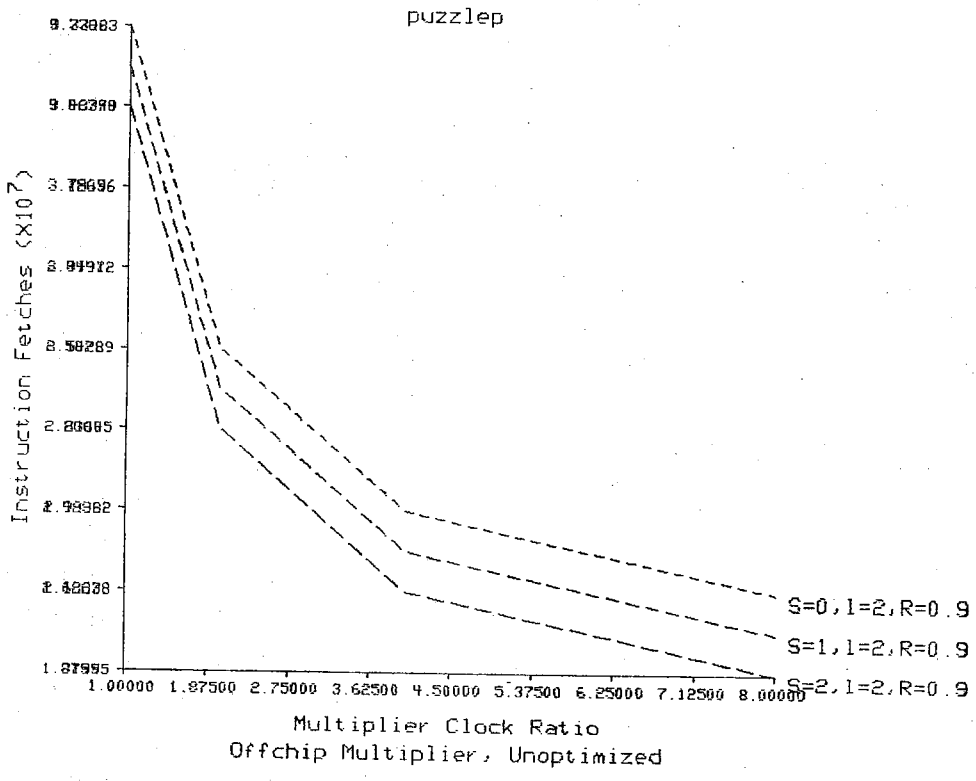


Figure 5.23 Execution Time of *Puzzle* with an Off-chip Bit-serial Multiplier with 90% Register Overflow and Silicon Parameters

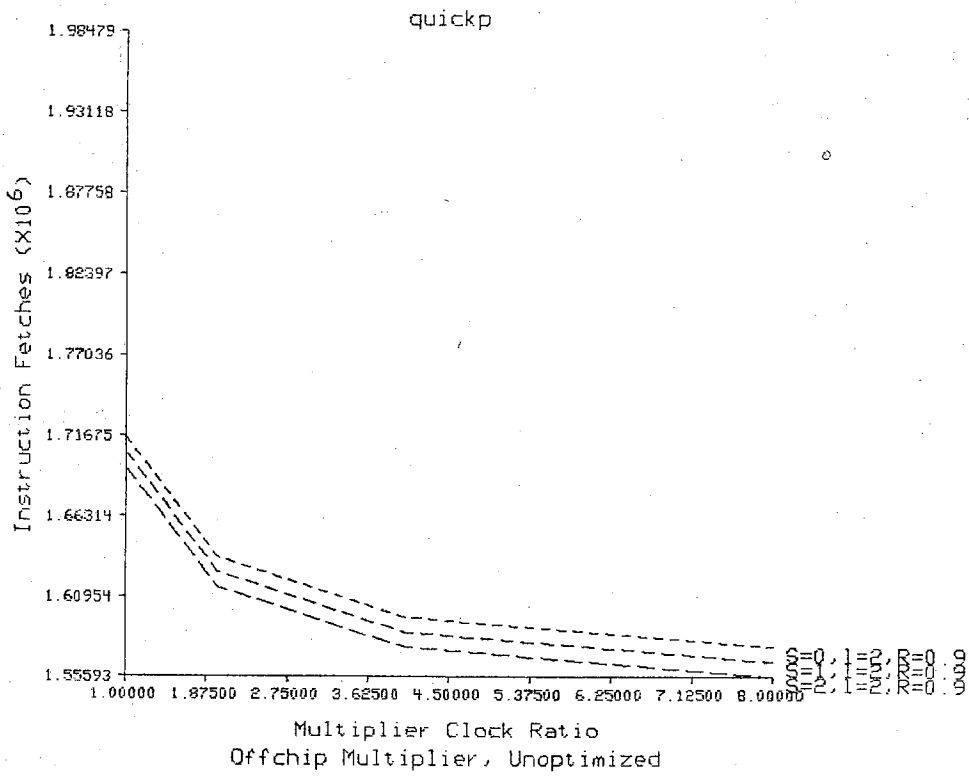


Figure 5.24 Execution Time of *Quick* with an Off-chip Bit-serial Multiplier with 90% Register Overflow and Silicon Parameters

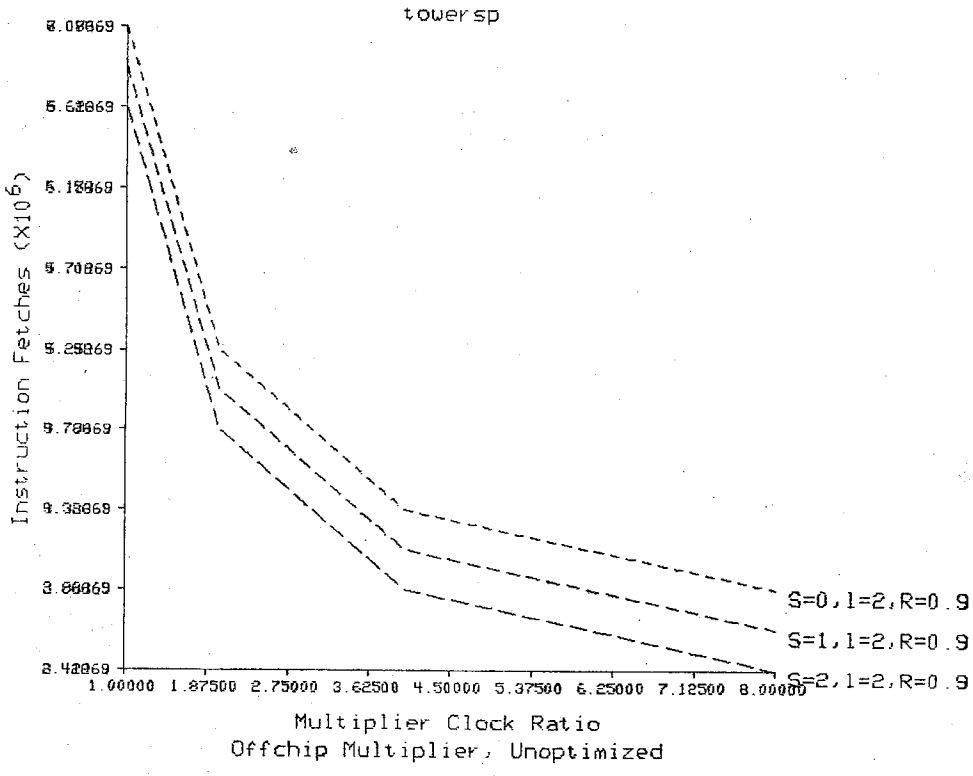


Figure 5.25 Execution Time of *Towers* with an Off-chip Bit-serial Multiplier with 90% Register Overflow and Silicon Parameters

CHAPTER 6
CACHE DESIGN

6.1. Introduction

Processor performance in any technology is greatly dependent on memory access time. This makes cache memory an important consideration in processor design. However, in the case of a problem, cache design is still of secondary importance behind the design of adder and multiplier units. The information is important when considering multiplication units since the Booth-algorithm and delay fillin use memory more frequently than many other operations. Therefore, this chapter is presented to give information on how cache affects performance in order to better choose the optimal multiplier for each application. There are papers which do cover cache in much greater detail [Kabak86], [Smith82].

Since system performance in any technology is still dependent on memory access time, cache memory is still a consideration in system design. Cache memory is placed between the CPU and the main memory to reduce memory fetch delays. Although cache memories are smaller than the main memory, they are also faster. Therefore, frequently accessed data can be kept in the cache to decrease total execution time. The penalty for fetching an item which is not in the cache is greater than that for fetching a data item directly from memory. The tradeoff of the penalty versus the increased speed must be examined closely to determine how the cache must be designed. When the memory access ratio exceeds 40, the penalty may be large enough to warrant a two level cache [SilMi85].

In the silicon environment, cache memory plays an important role in the performance of a system because the access ratio is significantly greater than one. In the GaAs environment, however, the ratio of off-chip to on-chip delays is much larger than in the Si environment and the increased memory fetch delay makes cache memory still more promising as an alternative. GaAs E/D MESFET technology has a ratio of five to ten which makes cache design more critical than in silicon technologies but not critical enough to justify a two level cache.

Standard assumptions in Si cache design need to be reevaluated for a GaAs environment. For example, set or fully associative placement policies are promoted as the best cache organizations for Si; yet a direct mapped cache was shown to be a better cache organization for GaAs [SilMi85]. After the issue of available area, the time for communication with the rest of the system is most affected by the technology switch. Therefore, in addition to considering parameters closely associated with the inner cache organization, some system parameters were considered to determine if their effect was changed by switching from Si to GaAs technology. Consideration was also given to choosing parameters to help improve cache efficiency.

6.1.1. Parameter Selection

The selection of parameters was guided by work already done in [Smith82]. Obviously, some choices of parameters, such as placement and replacement policy, are forced upon us by the choice of direct mapped cache. We did not look at the advantages of using split system/user caches because we did not have multi-user or multi-tasking programs that could provide us with a sufficient base of system code. Since previous experiments were run with the SU-MIPS simulation package, we continued to use it to be consistent. In addition, being bound by the MIPS-like architecture precluded experiments on architectures other than load-store architectures.

The cache size limitations were affected by transistor count limitations. Since the largest memory so far is a 16K bit SRAM with 102,300 gates [IsInI84], we could not realistically work with caches any larger than 4K by 32 bits. Even a 4K SRAM may not be implementable with the space limitations around the processor chip. This limit may be reduced further as the control logic is implemented. Typical silicon caches lose 25% of their area to control and the fanin fanout limitations of GaAs increase this loss to as much as 40% [SilMi85]. The maximum size of the silicon cache was kept to the maximum of the GaAs cache to provide a correlation for the same benchmarks and for other reasons explained later in this chapter.

6.2. Cache Evaluation Methodology

The cache design parameters examined included cache size, block size, and use/nonuse of one block look ahead prefetch for a direct mapped cache. The more important system parameters which we felt were modifiable included the fetch time for non-cache fetch times and cache miss times. The miss time included a constant delay plus an additional delay based on the block size. We

considered only off-chip on-package and off-chip off-package solutions because transistor count limitations would not allow any cache on the CPU chip.

6.3. Cache Experiment Procedure

Each test required running the SU-MIPS simulator for each benchmark for a set of parameters. Each benchmark was relatively small and generated under 4000 instructions. The original simulator was left unchanged except for memory references. Each memory reference goes through a "cache filter" which does an accurate simulation of the memory with cache. The cache filter is called in place of all memory references. For each memory reference, the cache filter updates the tags associated with each block of cache memory and then fetches the appropriate data or writes the data given to it. A block diagram of the data flow in the cache filter is shown in figure 6.1. If a cache miss occurs, the filter also adds on the appropriate miss delays. This allows us to add delays for any function of the cache for memory fetches. We added delays at two points: (1) whenever non-cache fetches are done, and (2) whenever the data must be fetched from main memory. All data fetches are considered non-cache fetches if only an instruction cache is being used. This also means that all instruction fetches are non-cache fetches if only a data cache is being used. This is shown graphically in figure 6.2. The execution times were calculated by summing the execution time of the benchmarks with the delays added by the "cache filter". The delay parameters used by the "cache filter" are set by the user during initialization of each simulator run.

The experiment was composed of three tests. The first test was to determine the impacts of instruction cache, data cache, or a combined instruction/data cache. GaAs area limitations do not permit the efficient placement of both instruction and data cache and therefore, we did not consider that option for GaAs or CMOS/SOS. The second test was to check the effect of varying the fetch time on the overall execution time. The last one was to determine the relative importance of the overhead delay and the delay per word transferred for a cache miss. The effect of the total miss time was also measured.

We did not run tests specifically to determine cache size because of our choice of benchmarks. The benchmarks that were available were not large enough to exercise the cache enough to accurately determine the performance of the cache for different cache sizes. Therefore these results may be less accurate for large benchmarks. The thesis by Kabakibo [Kabak86] does more extensive testing of cache designs with large benchmarks. Our choice of cache

sizes was not made smaller for fear of the block size reaching the same order of magnitude as the cache size.

Each set of data taken was for four different cache sizes, (256, 512, 1024, and 4096 words), and four block sizes, (two, four, eight, and sixteen), for each cache size. One of the remaining four parameters, cache type, miss time, fetch time, and prefetch policy, was varied for each test. Miss times are displayed for each curve and include both the transfer time for each word and the overhead incurred for each block transfer.

6.4. Presentation of Results

For each of the tests, only the interesting information has been presented as part of the thesis. The benchmarks *ack*, *intmm*, *queen*, and *sieve* were deemed interesting. The curves for the remaining benchmarks have a relatively flat profile and, therefore, are included only in the appendix.

Figures 6.3 through 6.8 show how cache organization affects execution time for each cache size for E/D-MESFET technologies, while figures 6.9 through 6.14 show the same information for CMOS/SOS technologies. The block size is denoted BS, the miss time is denoted M, and the fetch time is F for each of the curves. As expected, the results show decreasing execution time with increasing cache size, and due to the small benchmark size, the execution time levels off as the cache is filled with the working information. This is true for each of the block sizes we dealt with. The longest execution time is for the simulation with only data cache; the instruction cache and the combined cache both have superior execution times.

The relatively large difference in execution times for the various block sizes shows that the combined cache is more sensitive to changes in block size than the instruction cache; the smaller the blocks, the better the execution time. The asymptotic nature of the curves show that very little information is being swapped out due to memory pollution. Therefore, the wide spacing of the curves may be due to shorter miss times with small blocks rather than less memory pollution. The data also supports using instruction cache or a combined instruction/data cache for GaAs.

Figures 6.15 through 6.18 show the execution time against varying cache size for GaAs parameters while figures 6.19 through 6.22 show the execution time against varying cache size for silicon parameters. The individual curves within each family of curves represent different block sizes. The two families are identified by different non-cache fetch times. Data fetches use non-cache fetch times if data cache is not being used. Analogously, instruction fetches

use non-cache fetch times if instruction cache is not being used. The large gap in execution times between the two fetch values with very little difference between the block sizes shows that changing the fetch time has the greatest impact on execution time. As the fetch time is decreased, the execution time decreases much more quickly than any change in block size or other parameters accounts for. By observing plots for other experiments, the only parameter which forces such a large change is the cache organization.

Additional plots of execution time against cache size are displayed in figures 6.23 through 6.30 for GaAs parameters and figures 6.31 through 6.38 for silicon parameters. Here, however, the differences in execution times are caused by changing the miss times. Each family of curves is equated to a different miss time where the miss time is the total of the overhead of the transfer time added to the time to transfer the block. As expected, the times are not greatly dependent on the base miss time. The greatest dependency is the time added per word transferred because the total delay per transfer is usually greater than the transfer overhead. This dependence on transfer delay emphasizes the need to reduce the block transfer time rather than the overhead associated with each block transfer.

Reviewing the results, judgments should be influenced by the knowledge that the benchmarks did not provide a flawless basis for evaluation of the cache parameters. After a certain time, cache misses became rare due to the program size being insignificant compared to the cache size. Therefore, one of the biggest influences on the execution time was the cold start cache that was used. Larger benchmark programs would have increased the reliability of our results.

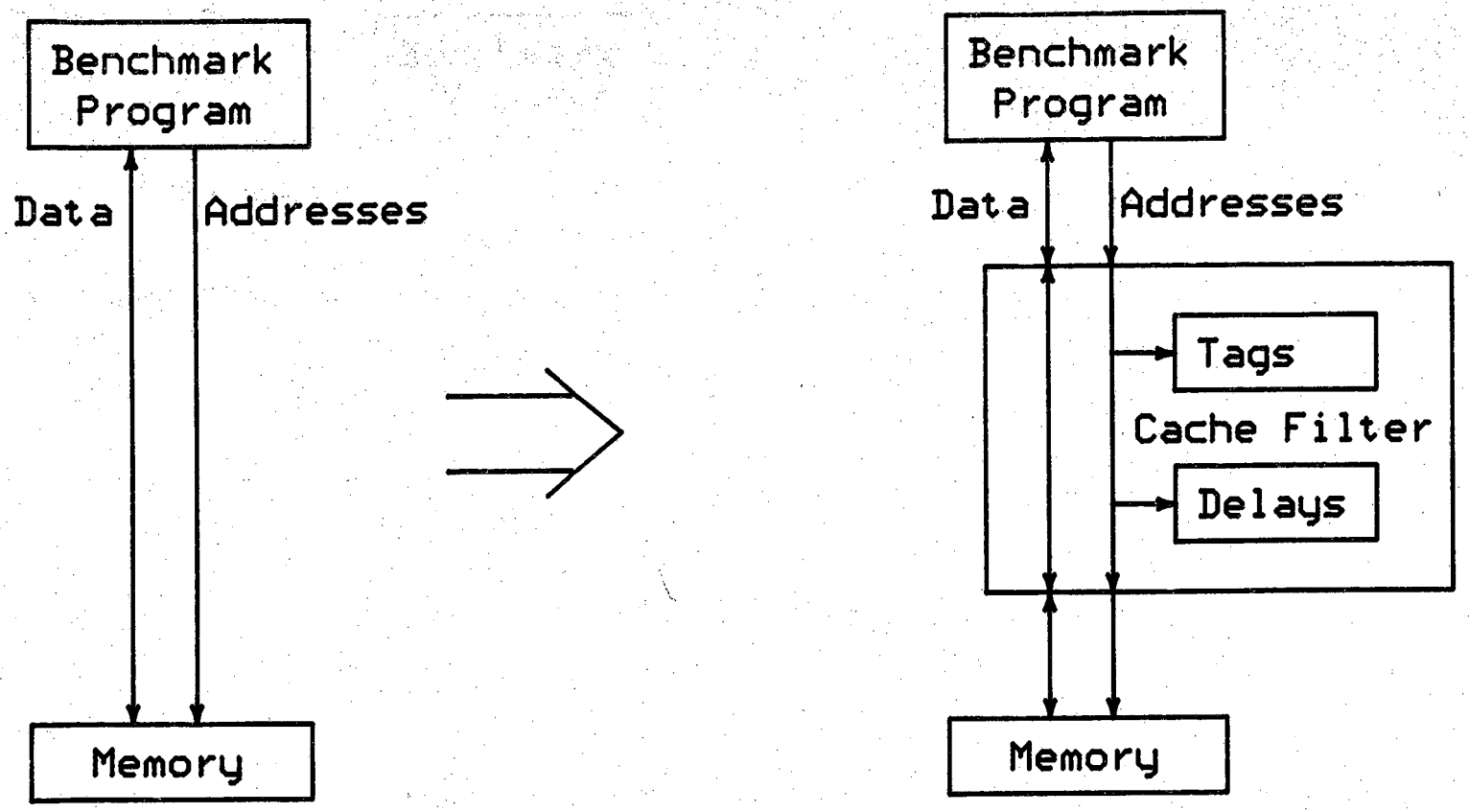


Figure 6.1 Block Diagram of "Cache Filter" Data Flow

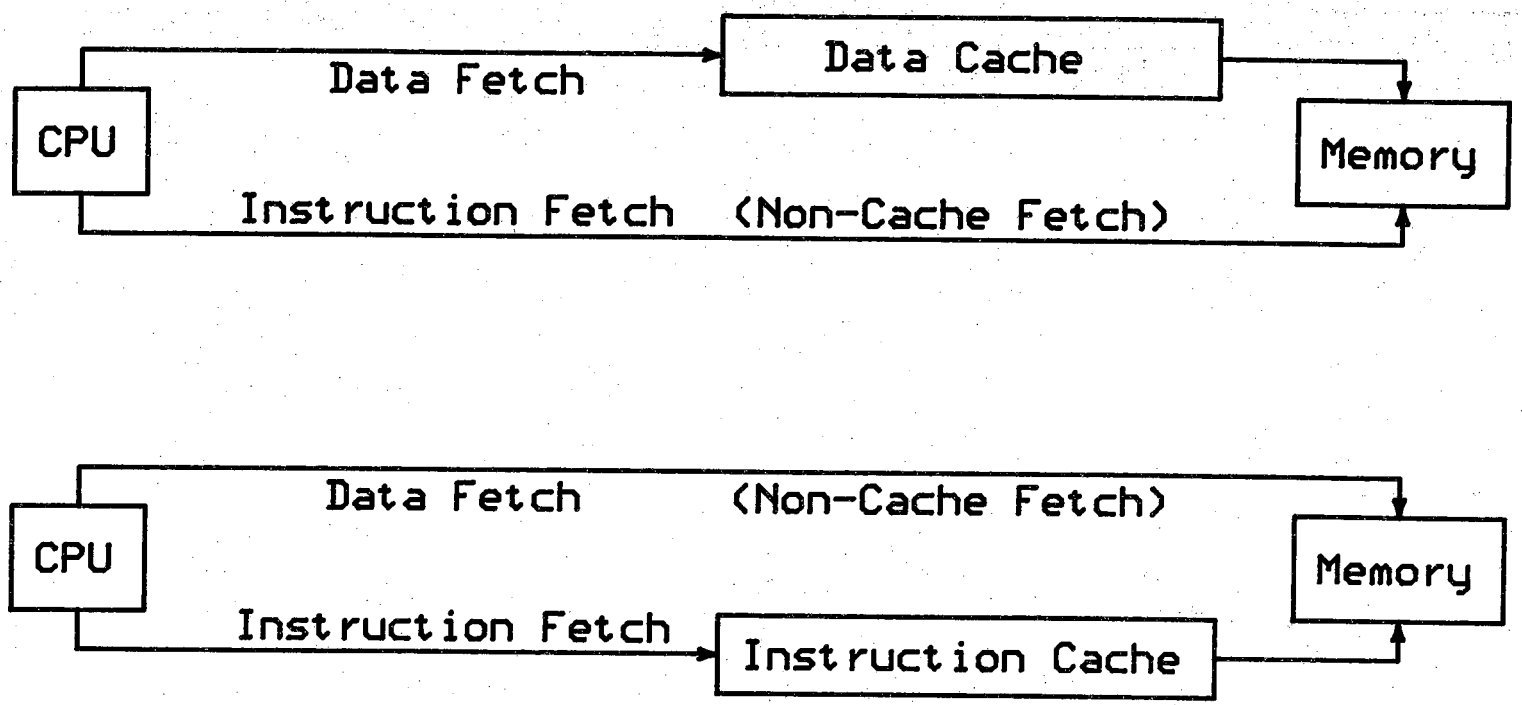


Figure 6.2 Explanation of Non-cache Fetch Time

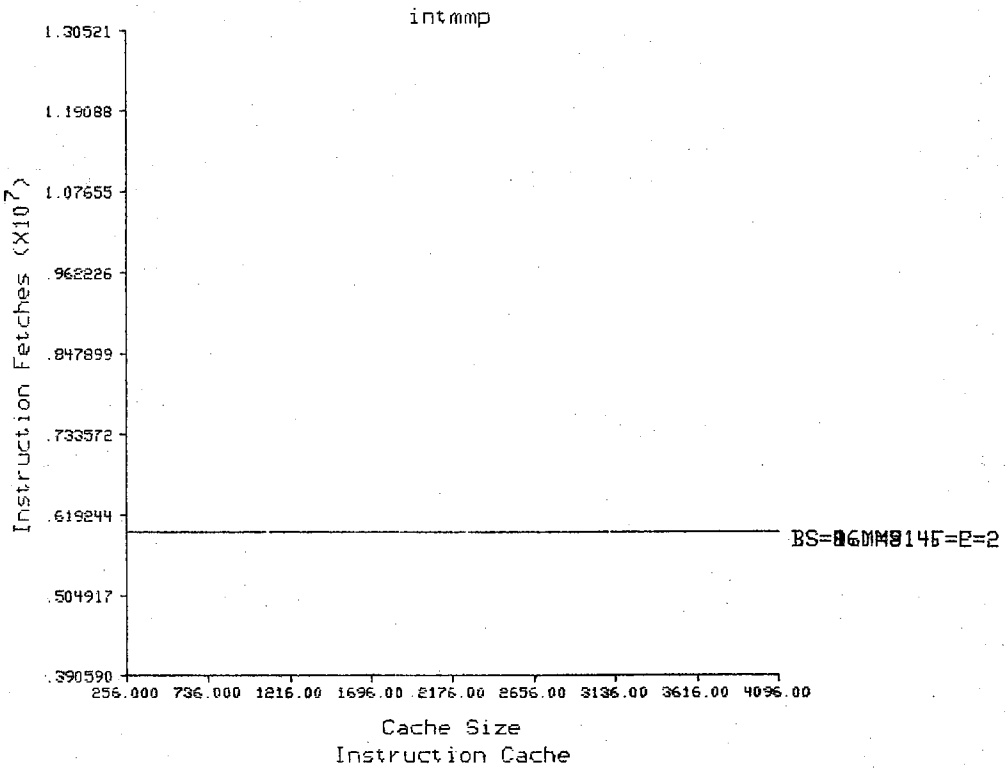


Figure 6.3 Execution Time of *Intmm* with an Instruction Cache with GaAs Parameters

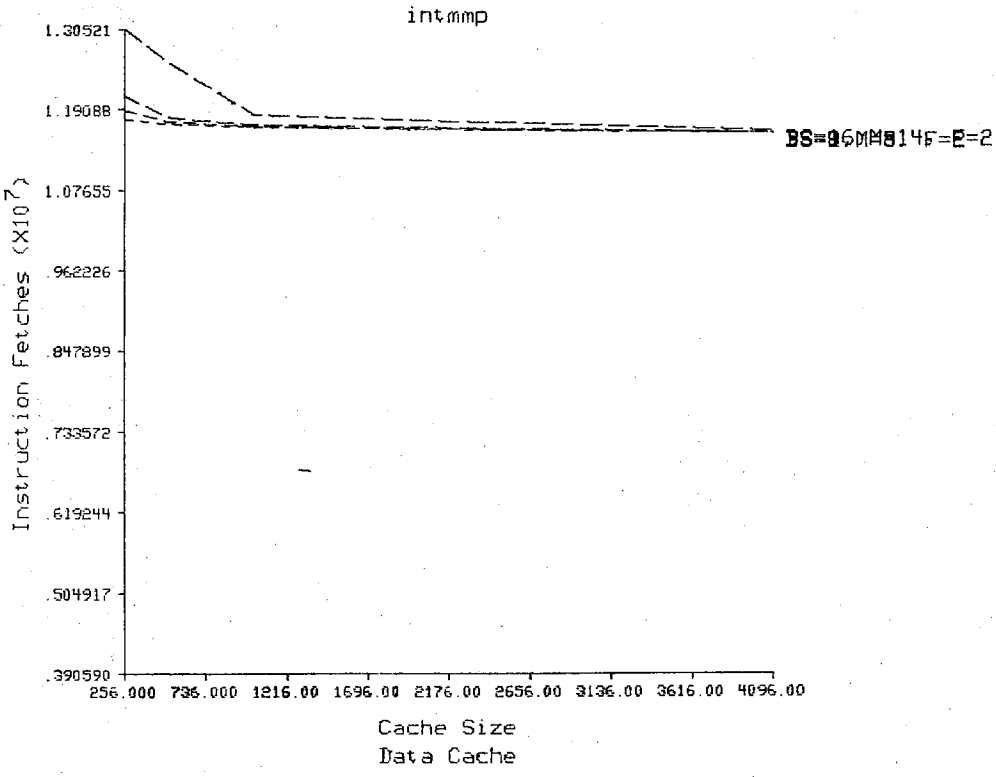


Figure 6.4 Execution Time of *Intmm* with a Data Cache with GaAs Parameters

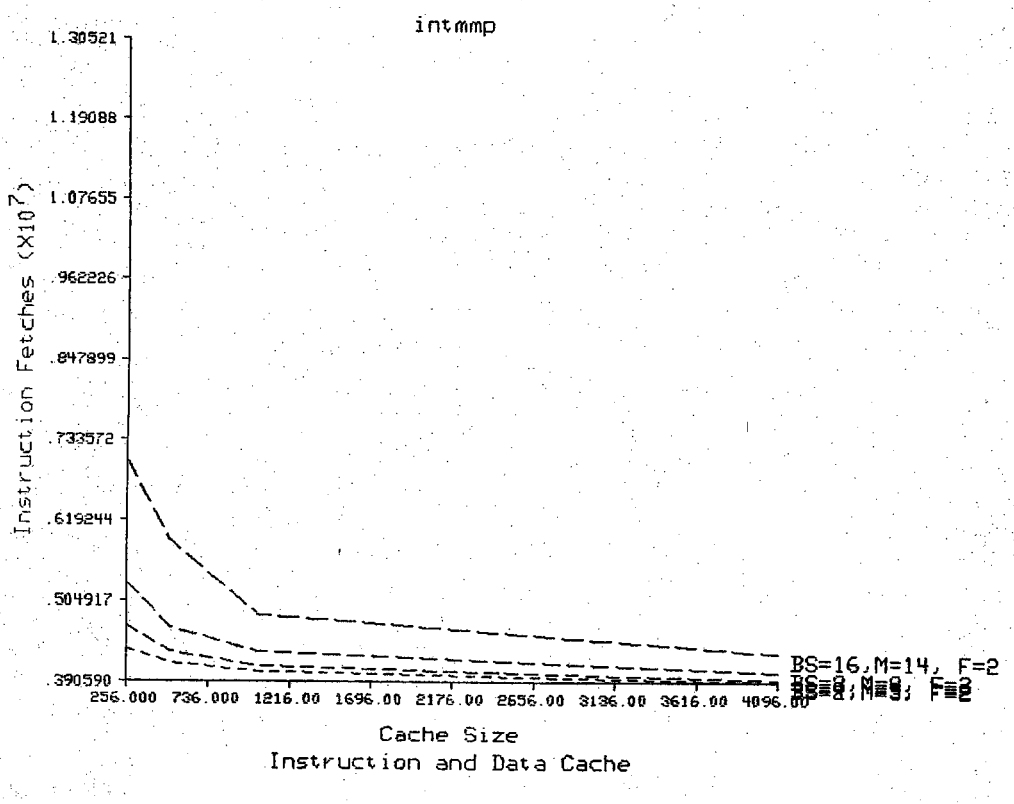


Figure 6.5 Execution Time of *Intmm* with a Combined Instruction/Data Cache with GaAs Parameters

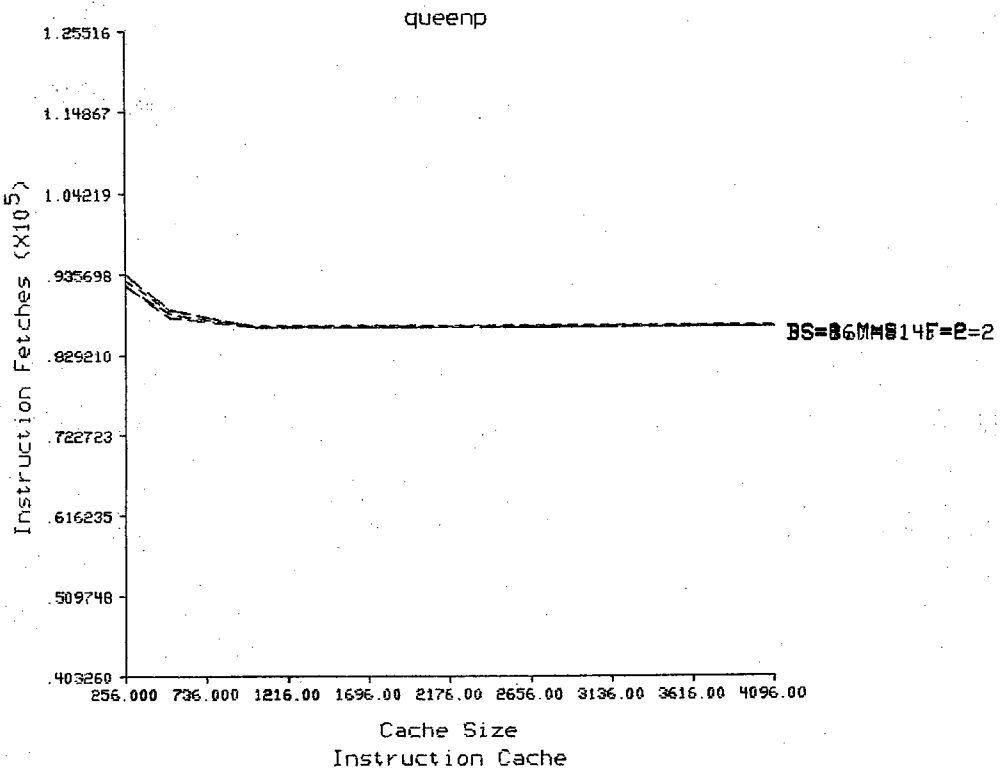


Figure 6.6 Execution Time of *Queen* with an Instruction Cache with GaAs Parameters

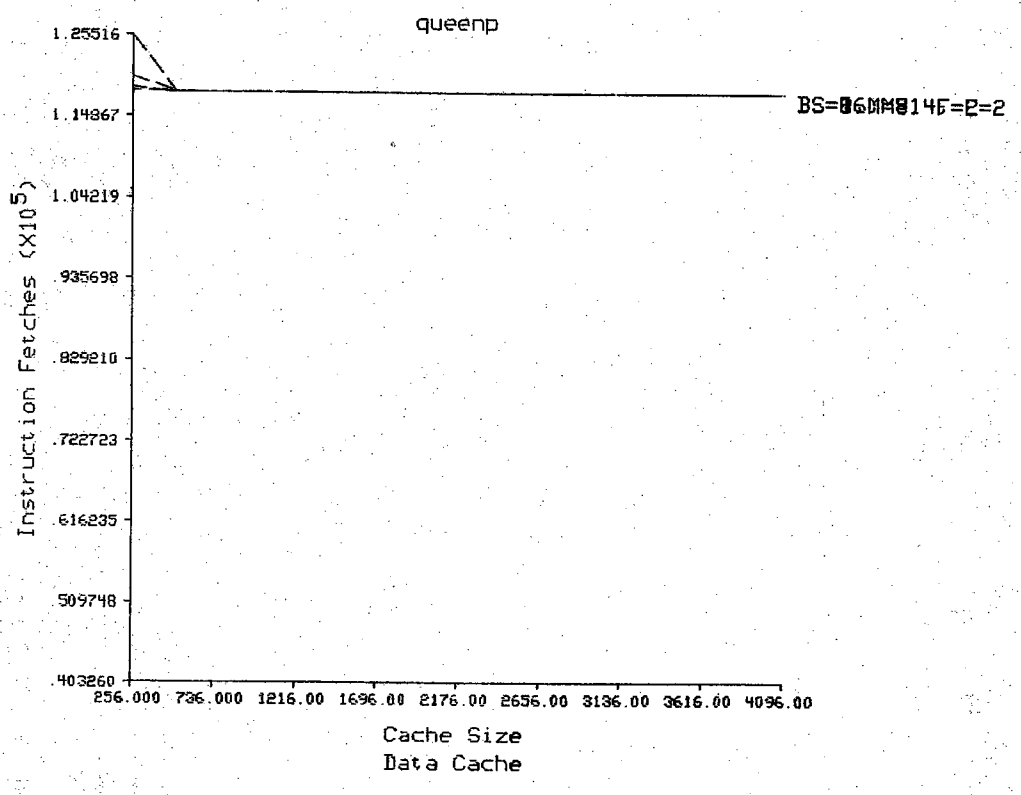


Figure 6.7 Execution Time of *Queen* with a Data Cache with GaAs Parameters

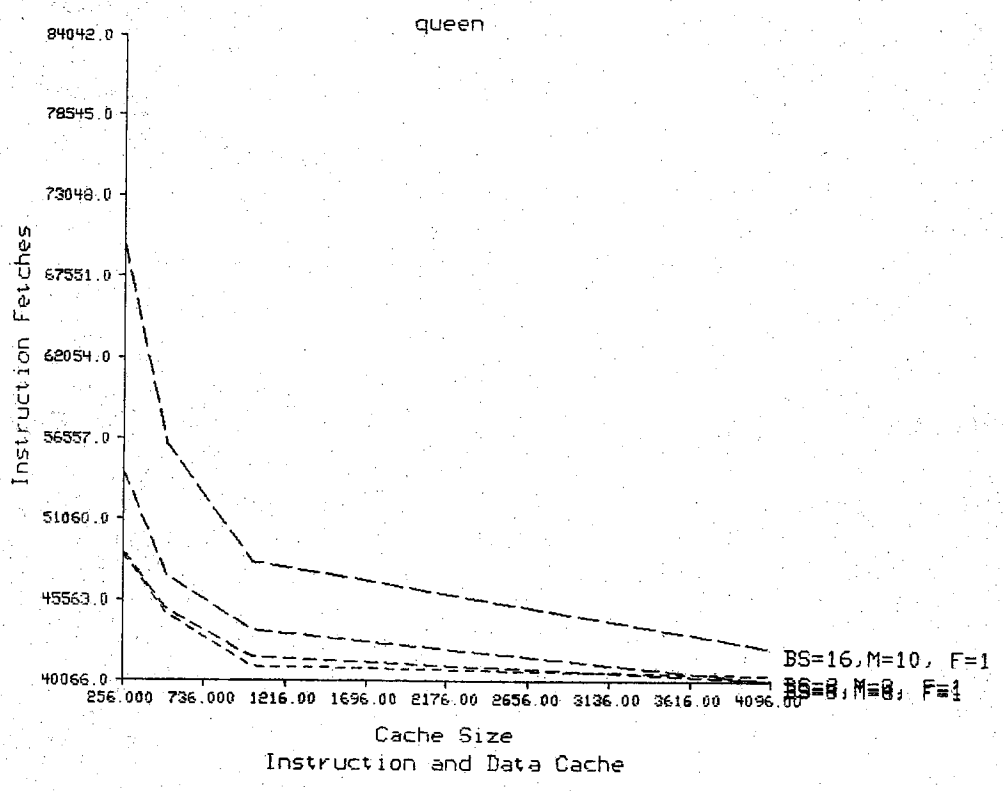


Figure 6.8 Execution Time of *Queen* with a Combined Instruction/Data Cache with GaAs Parameters

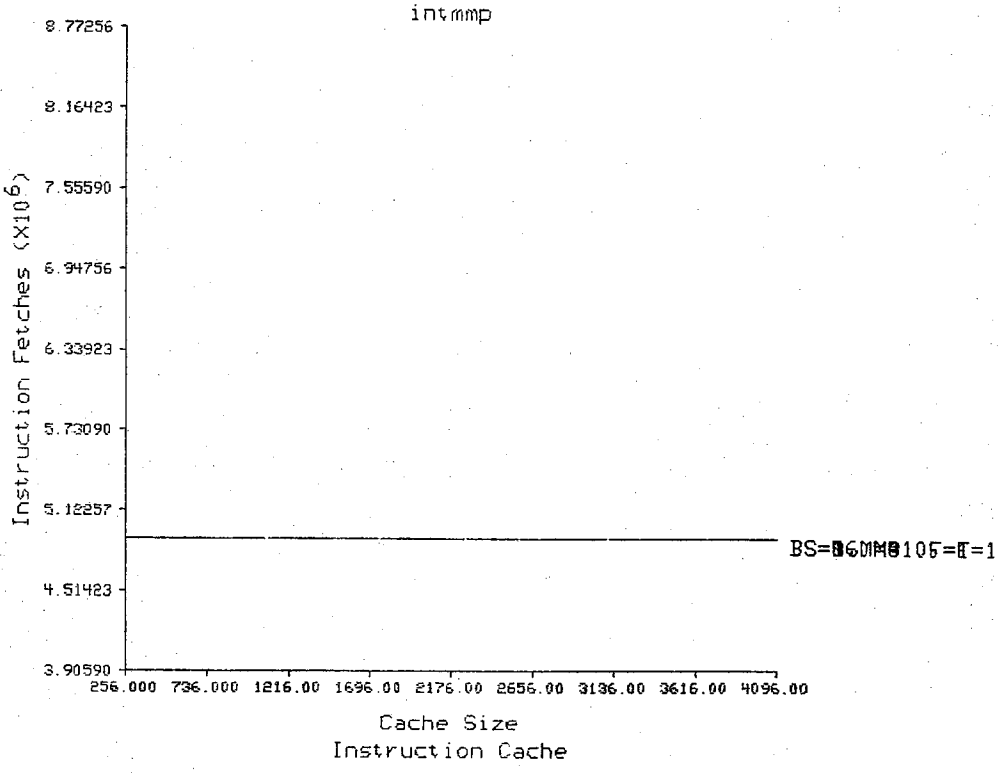


Figure 6.9 Execution Time of *Intmm* with an Instruction Cache with Si Parameters

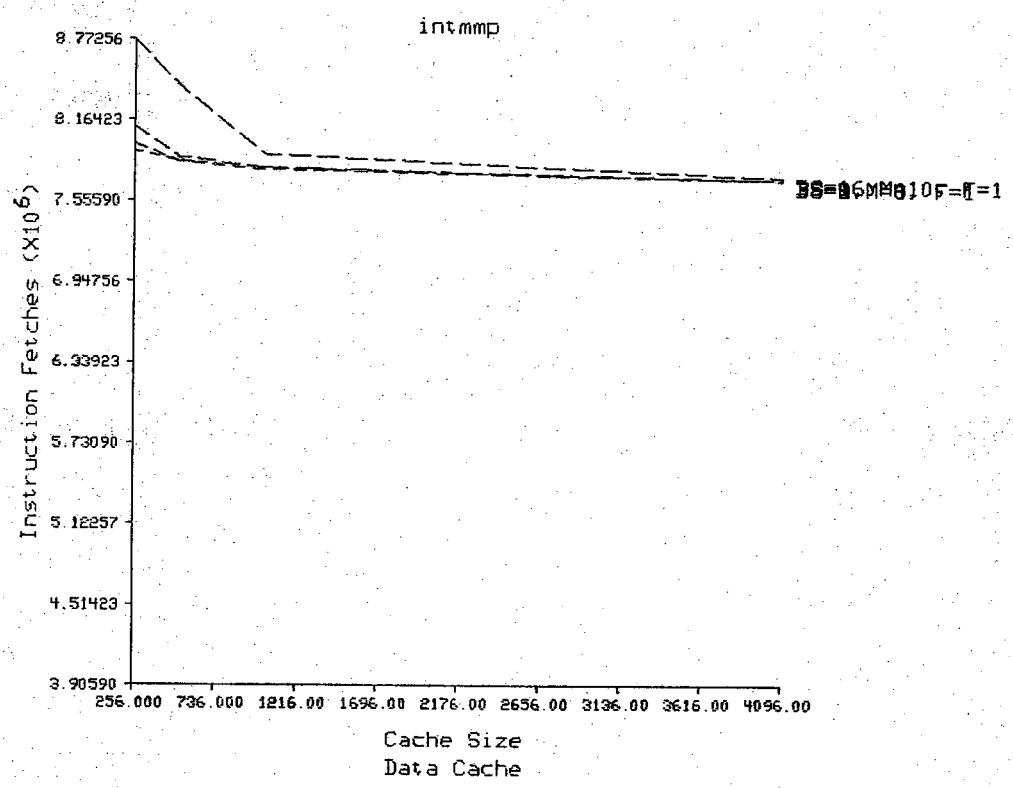


Figure 6.10 Execution Time of *Intmm* with a Data Cache with Si Parameters

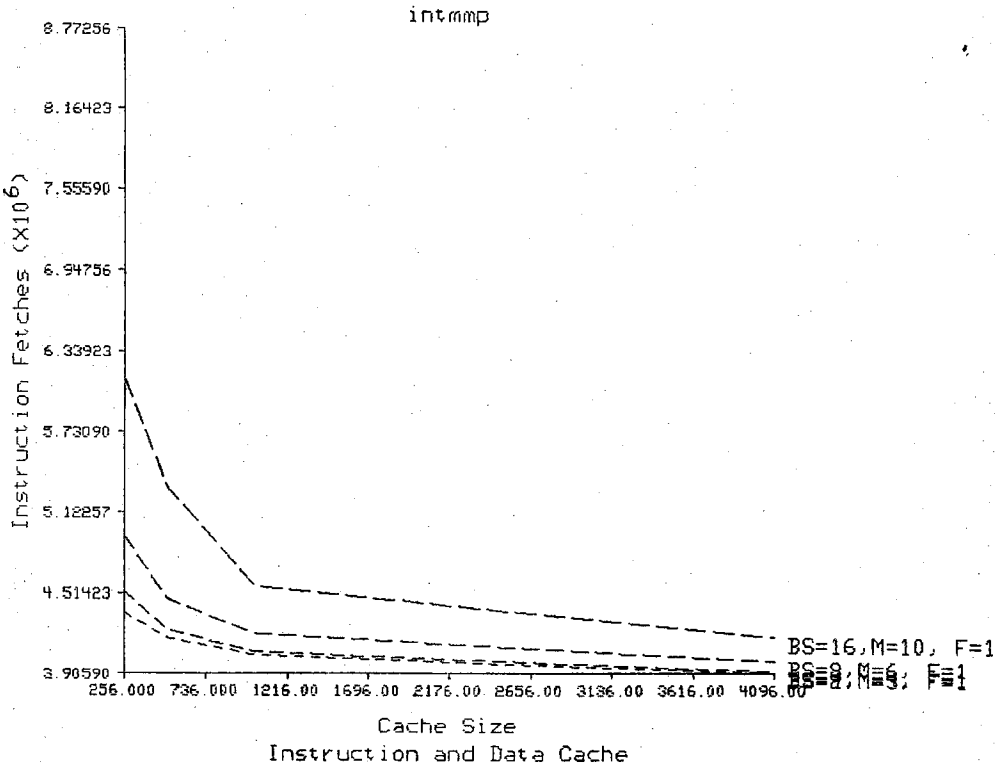


Figure 6.11 Execution Time of *Intmm* with a Combined Instruction/Data Cache with Si Parameters

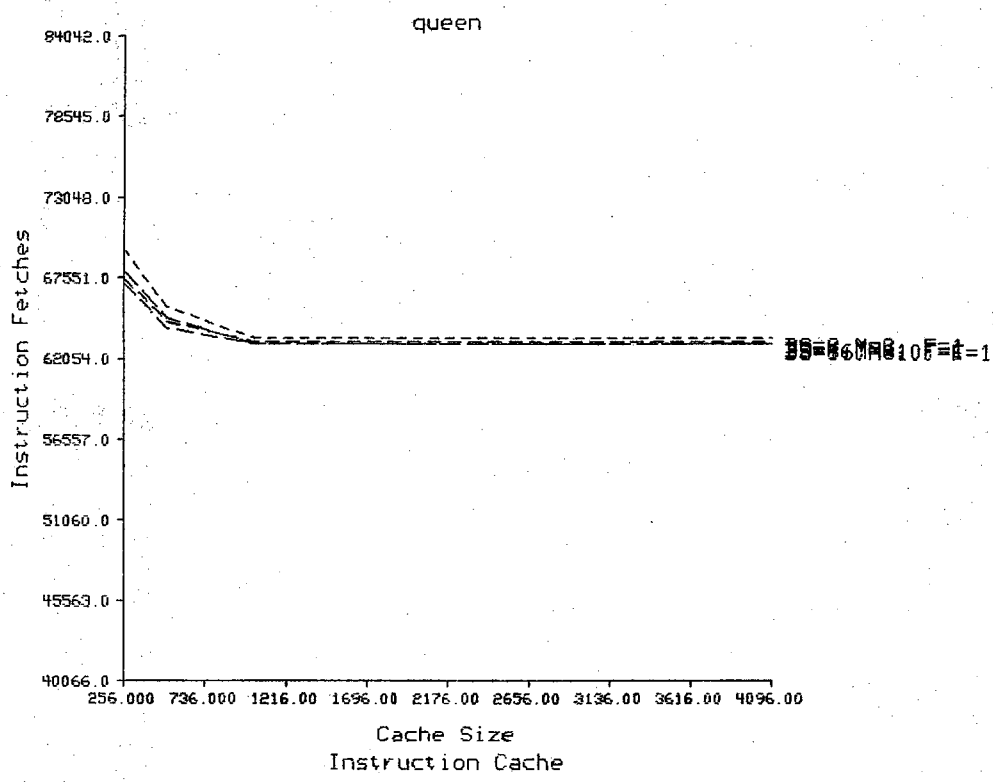


Figure 6.12 Execution Time of *Queen* with an Instruction Cache with Si Parameters

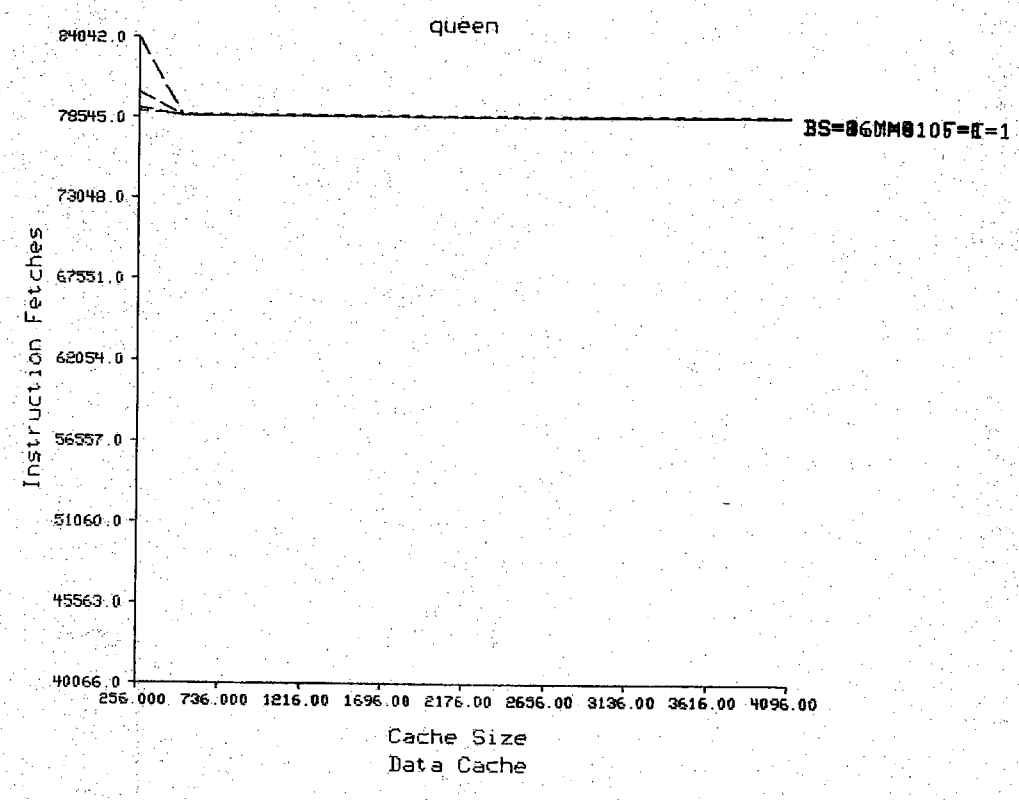


Figure 6.13 Execution Time of *Queen* with a Data Cache with Si Parameters

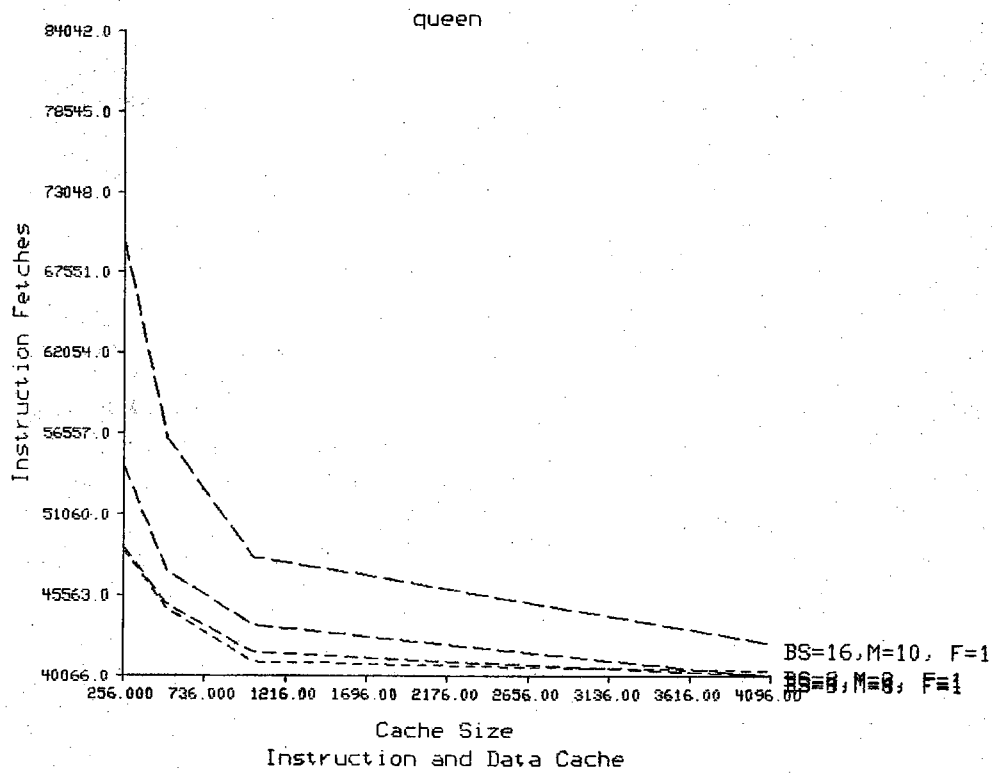


Figure 6.14 Execution Time of *Queen* with a Combined Instruction/Data Cache with Si Parameters

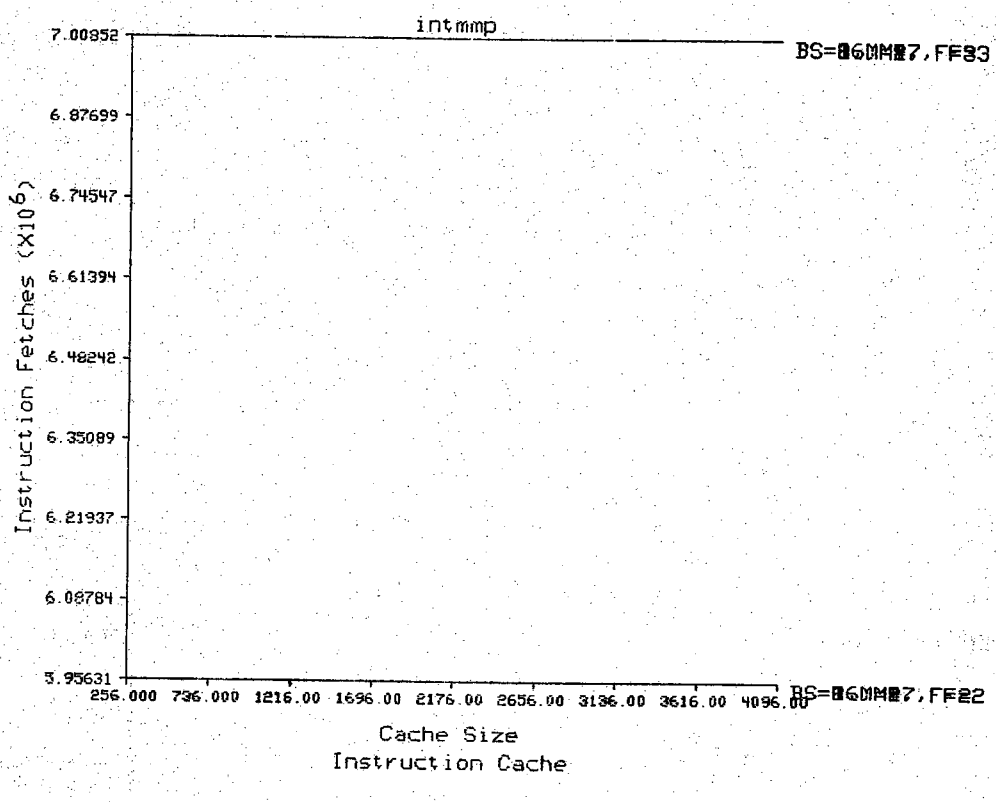


Figure 6.15 Execution Time of *Intmm* for GaAs Fetch Delays with a Slow Cache

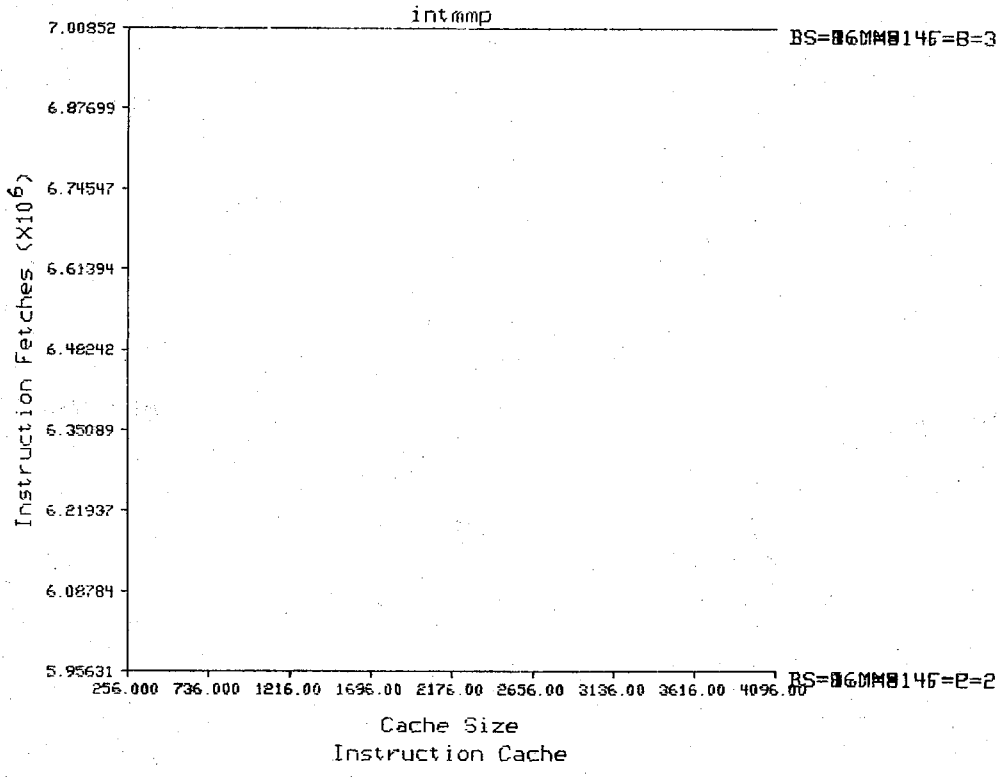


Figure 6.16 Execution Time of *Intmm* for GaAs Fetch Delays with a Fast Cache

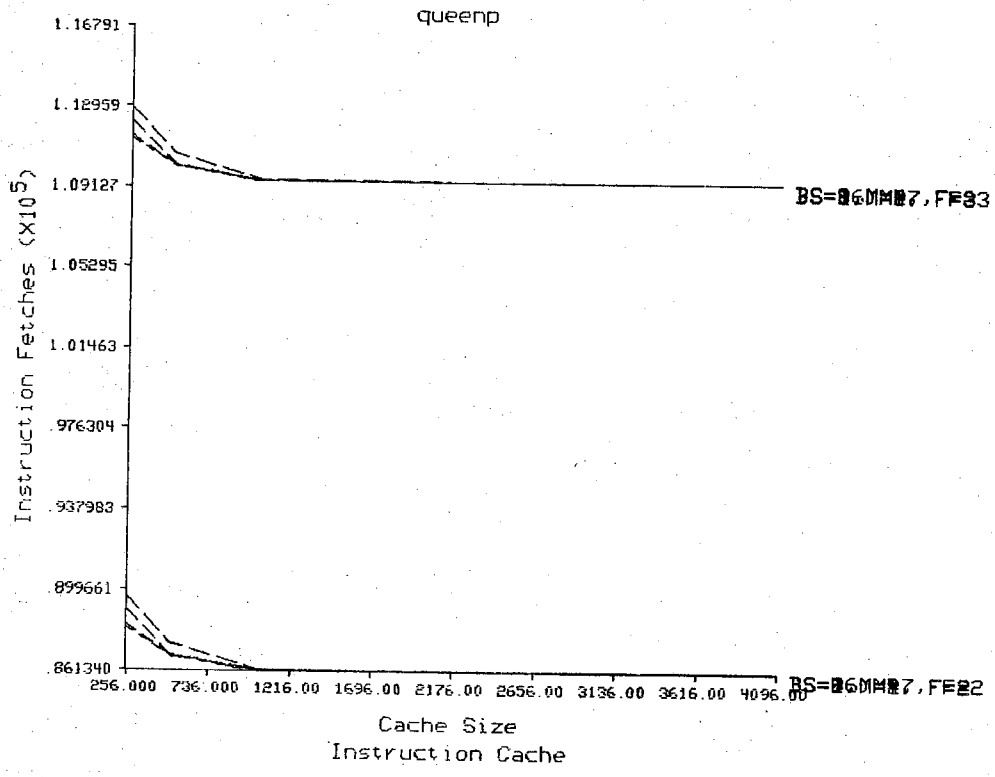


Figure 6.17 Execution Time of *Queen* for GaAs Fetch Delays with a Slow Cache

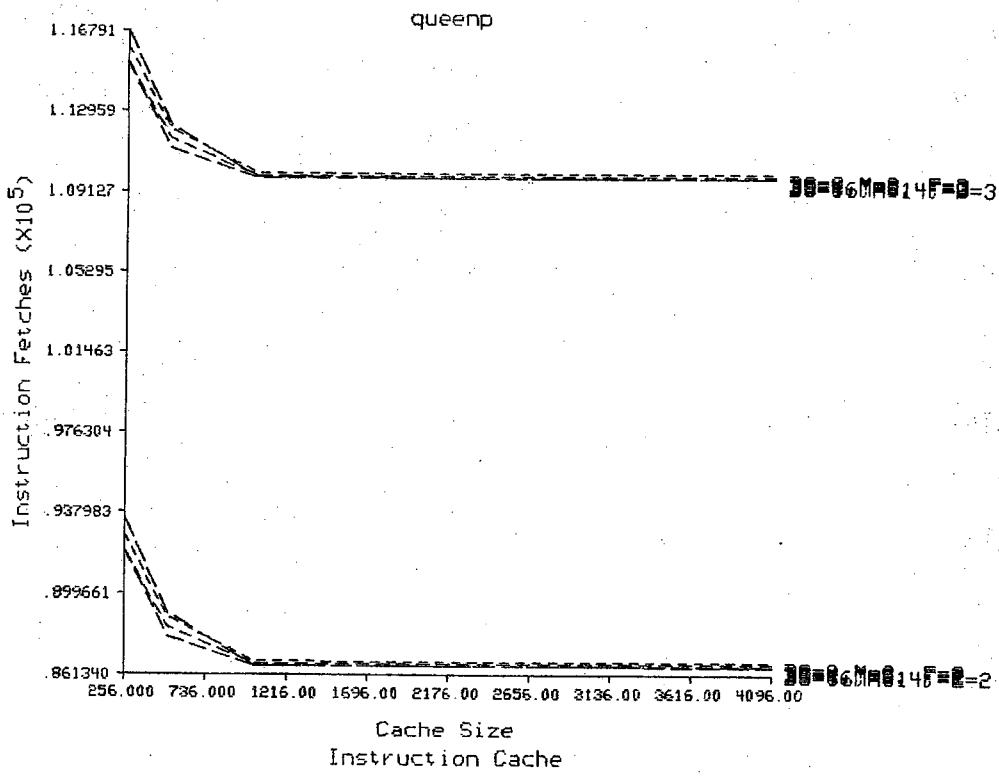


Figure 6.18 Execution Time of *Queen* for GaAs Fetch Delays with a Fast Cache

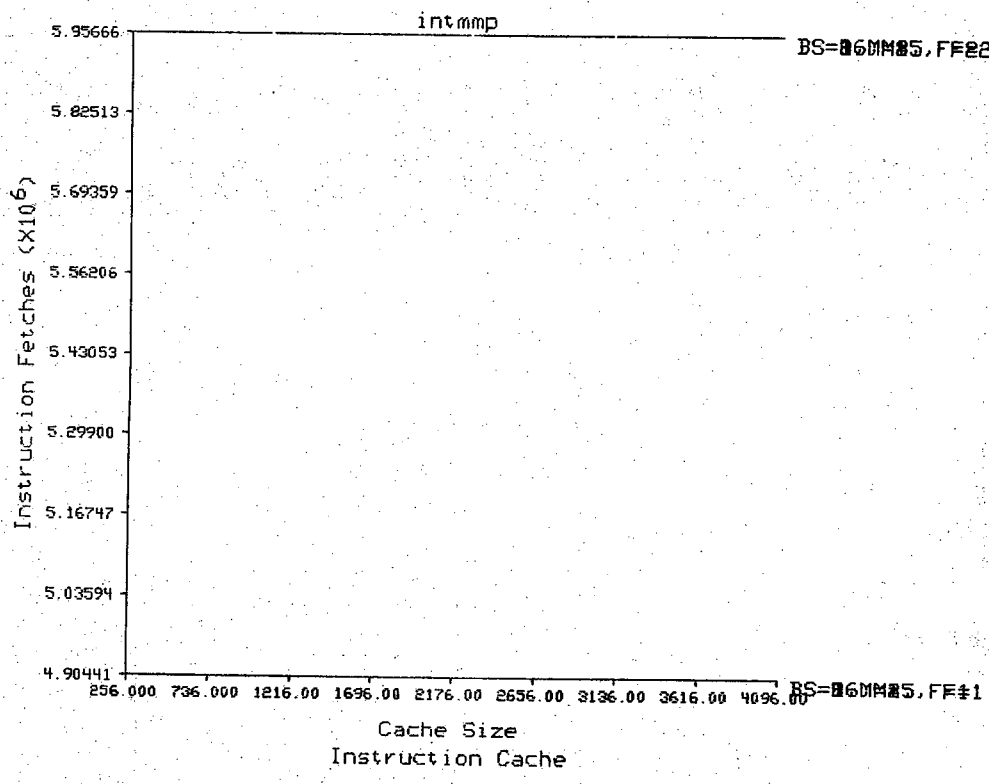


Figure 6.19 Execution Time of *Intmm* for Si Fetch Delays with a Slow Cache

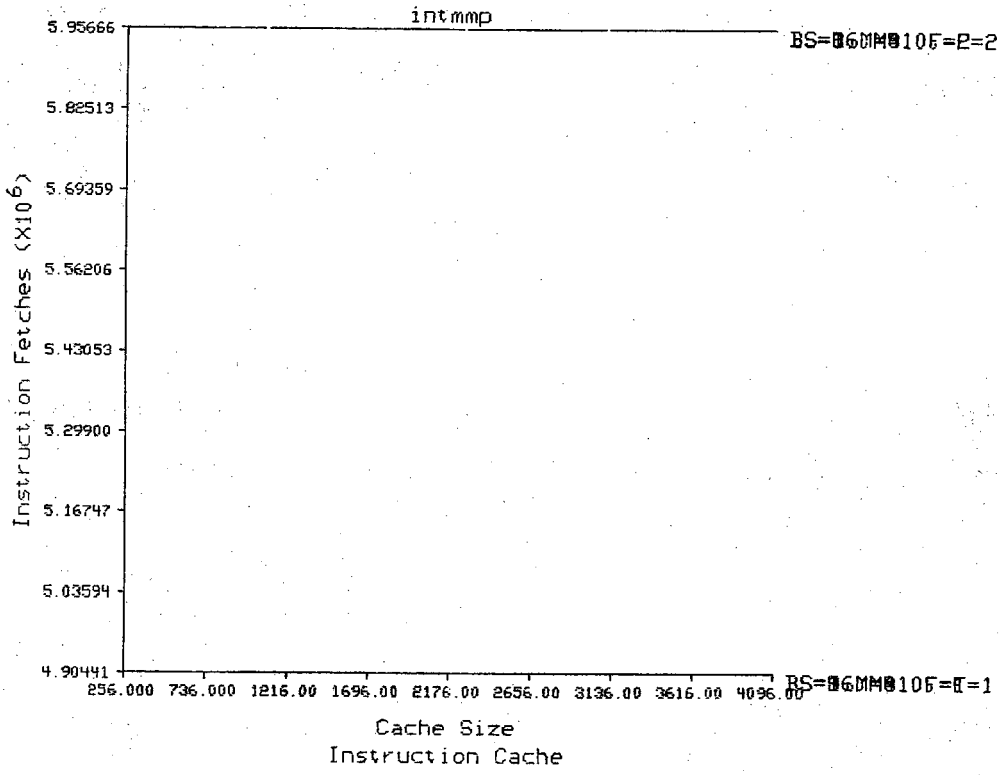


Figure 6.20 Execution Time of *Intmm* for Si Fetch Delays with a Fast Cache

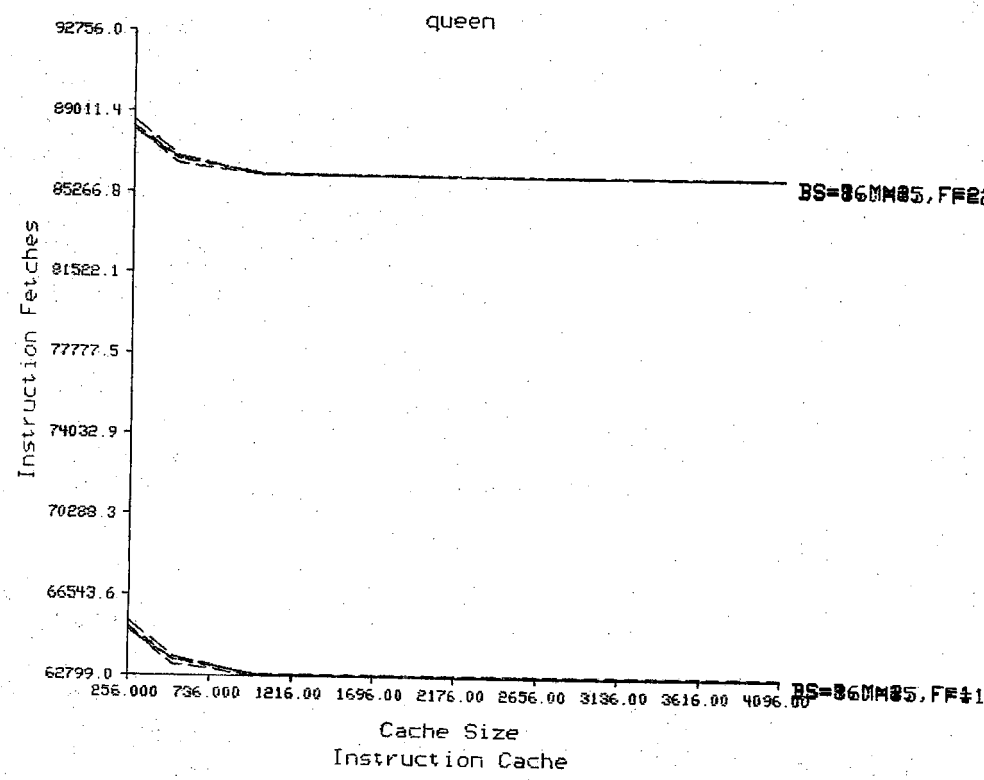


Figure 6.21 Execution Time of *Queen* for Si Fetch Delays with a Slow Cache

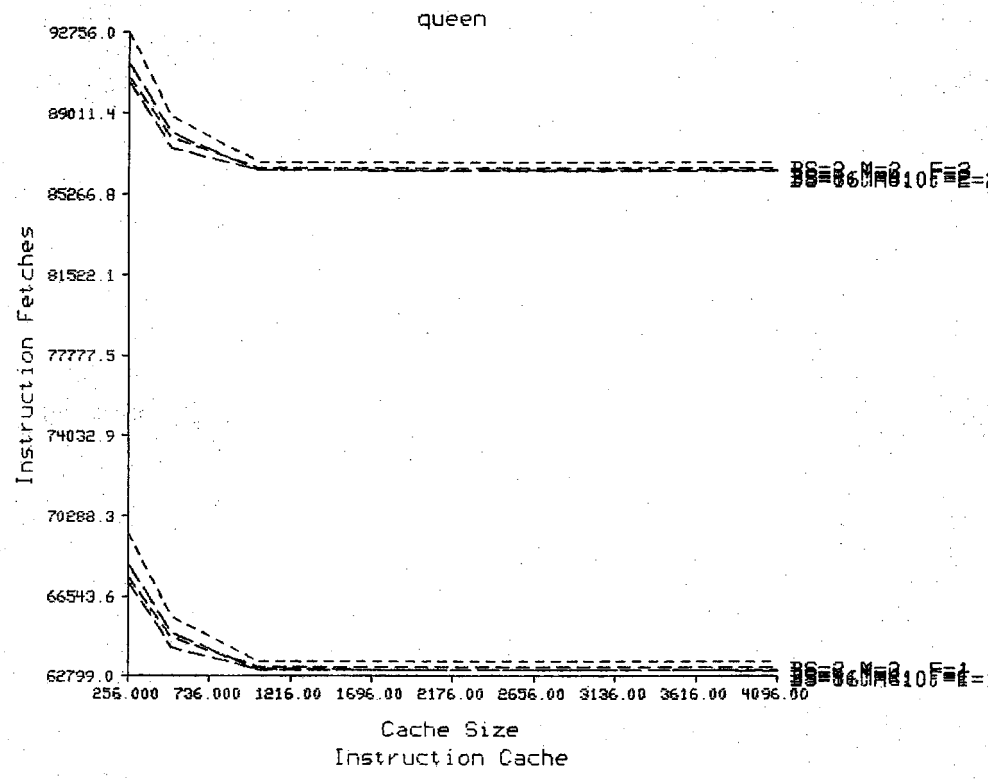


Figure 6.22 Execution Time of *Queen* for Si Fetch Delays with a Fast Cache

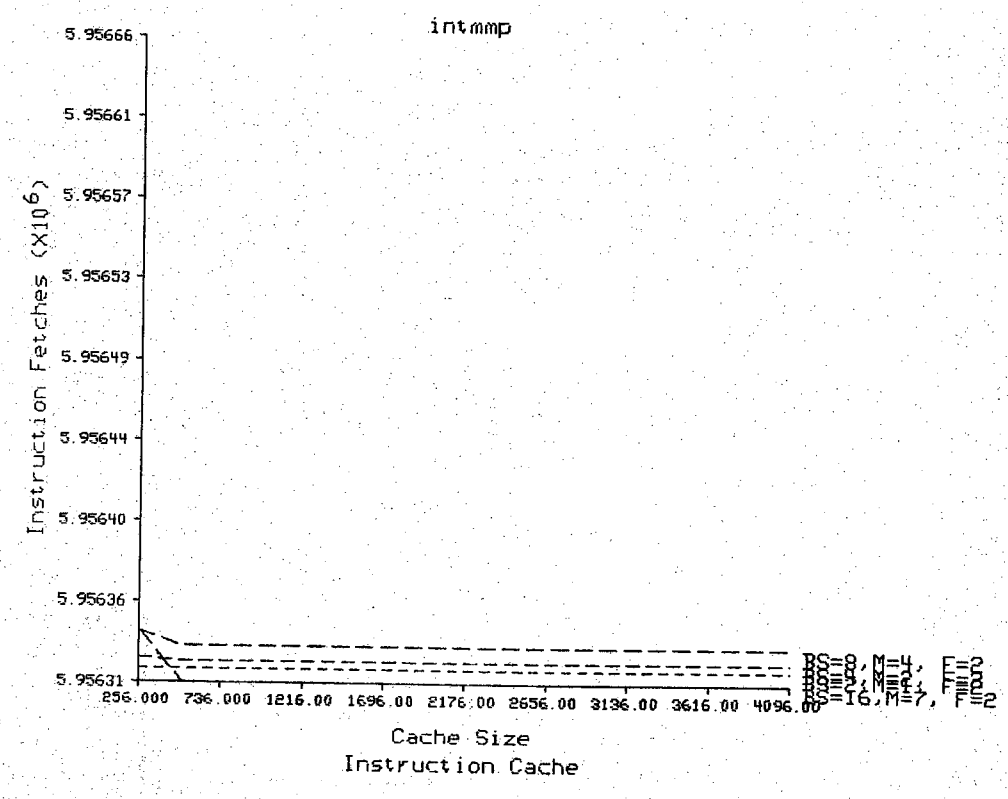


Figure 6.23 Execution Time of *Intmm* with Small Base Delays and Small Transfer Delays for GaAs Parameters

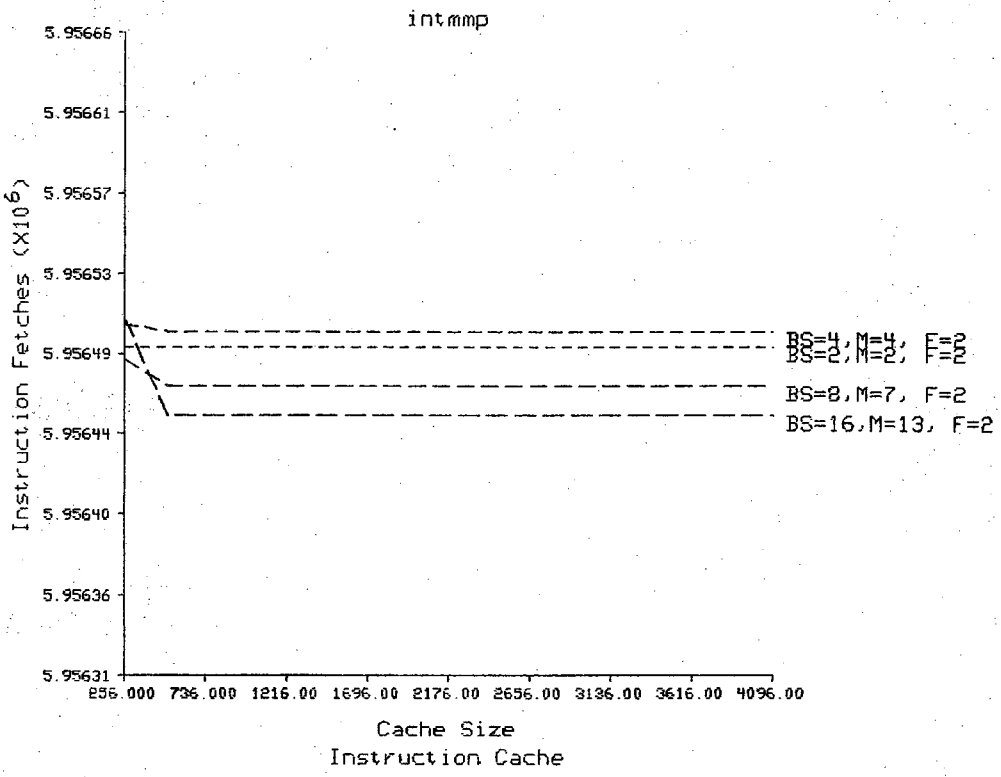


Figure 6.24 Execution Time of *Intmm* with Small Base Delays and Large Transfer Delays for GaAs Parameters

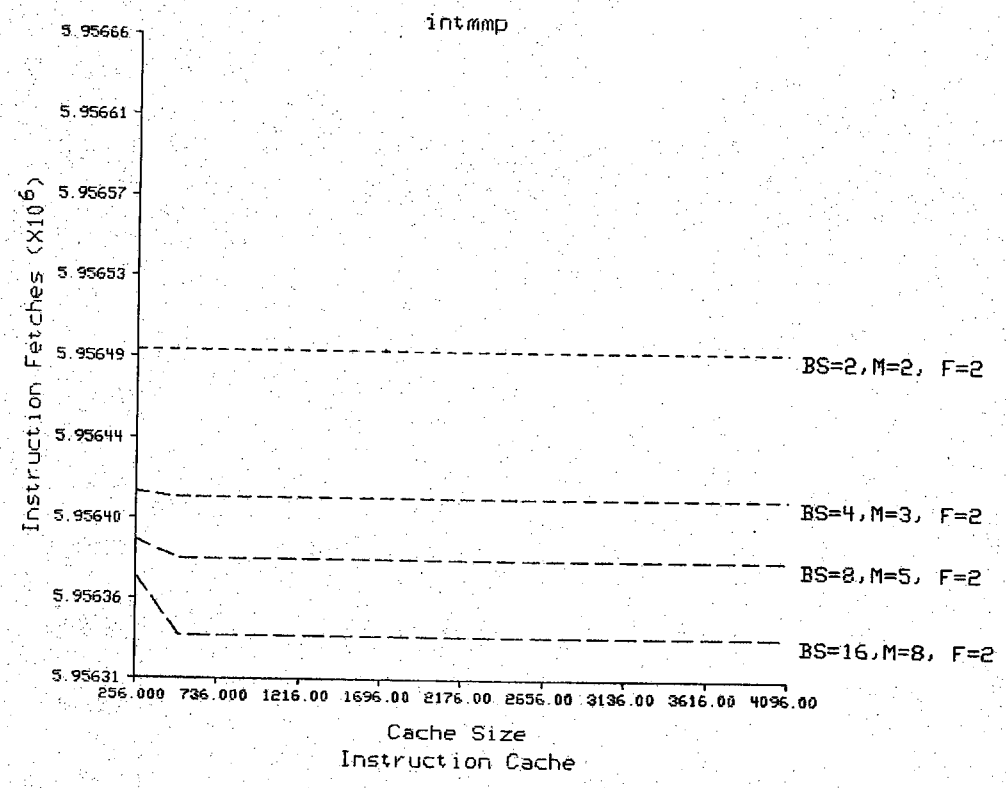


Figure 6.25 Execution Time of *Intmm* with Large Base Delays and Small Transfer Delays for GaAs Parameters

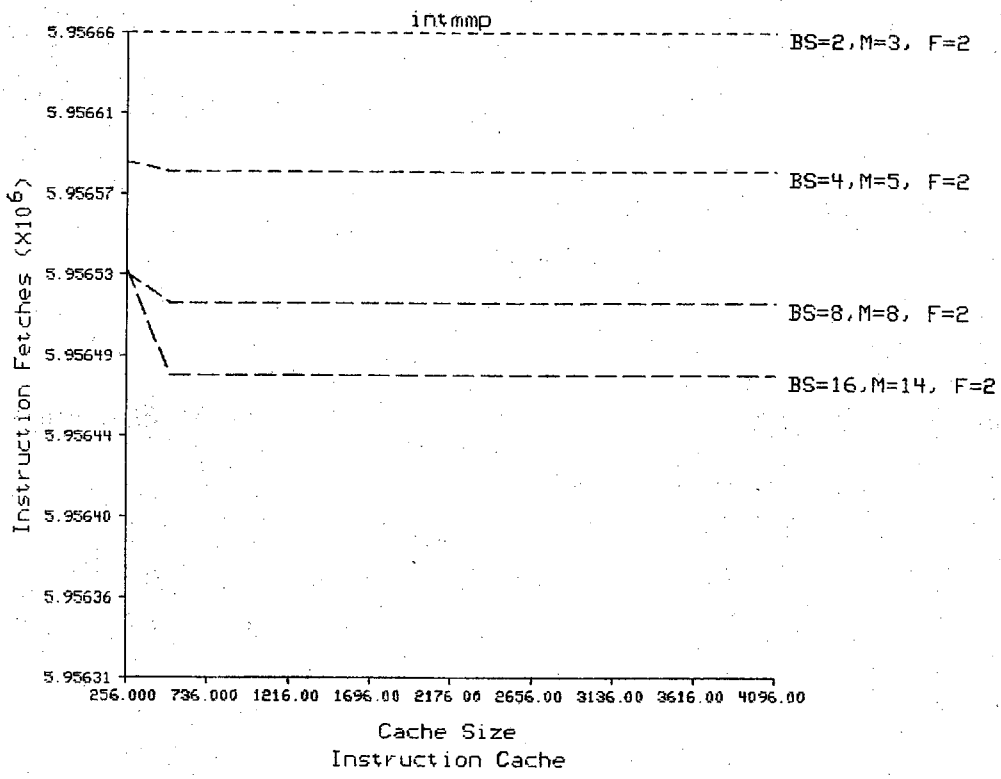


Figure 6.26 Execution Time of *Intmm* with Large Base Delays and Large Transfer Delays for GaAs Parameters

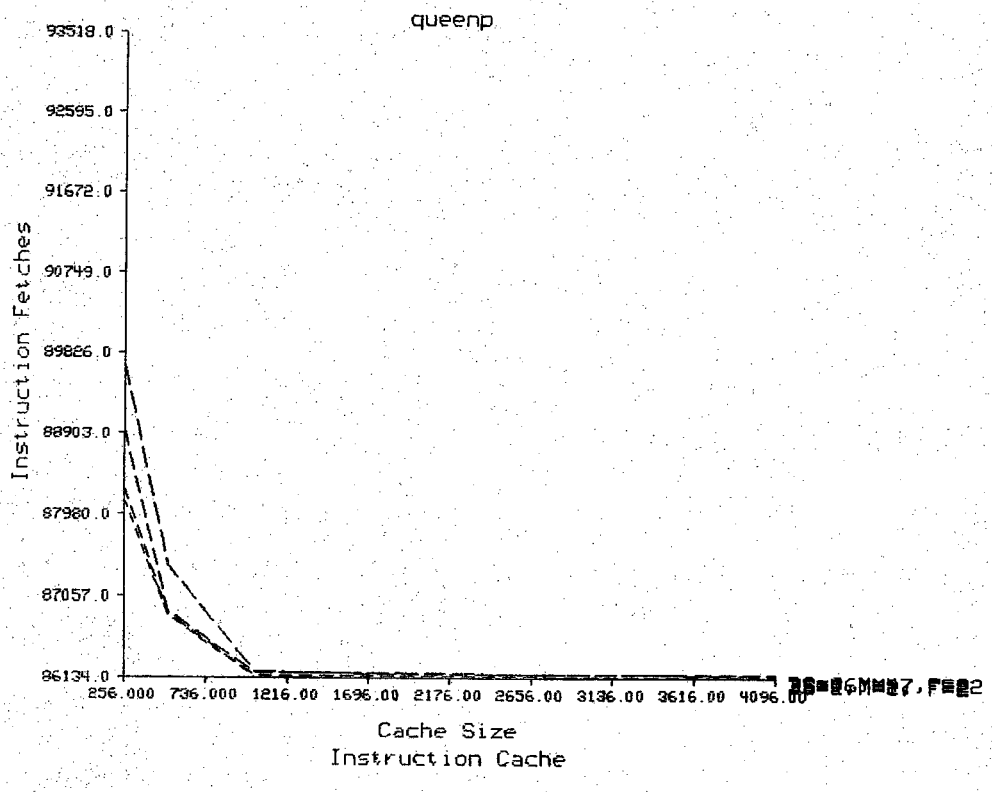


Figure 6.27 Execution Time of *Queen* with Small Base Delays and Small Transfer Delays for GaAs Parameters

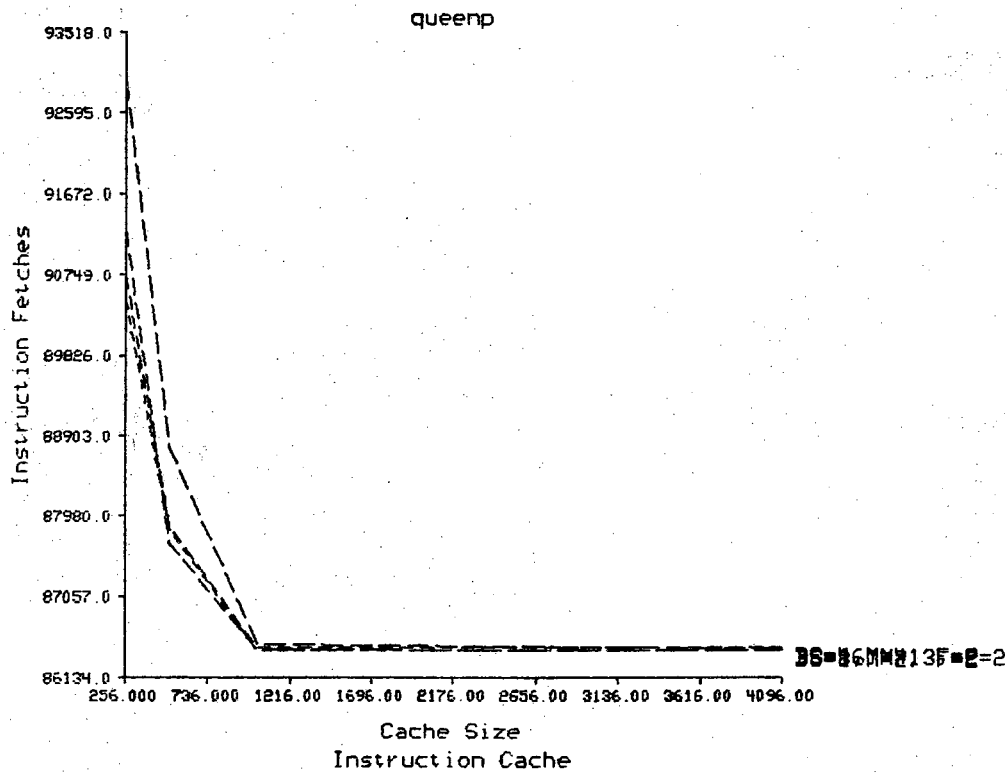


Figure 6.28 Execution Time of *Queen* with Small Base Delays and Large Transfer Delays for GaAs Parameters

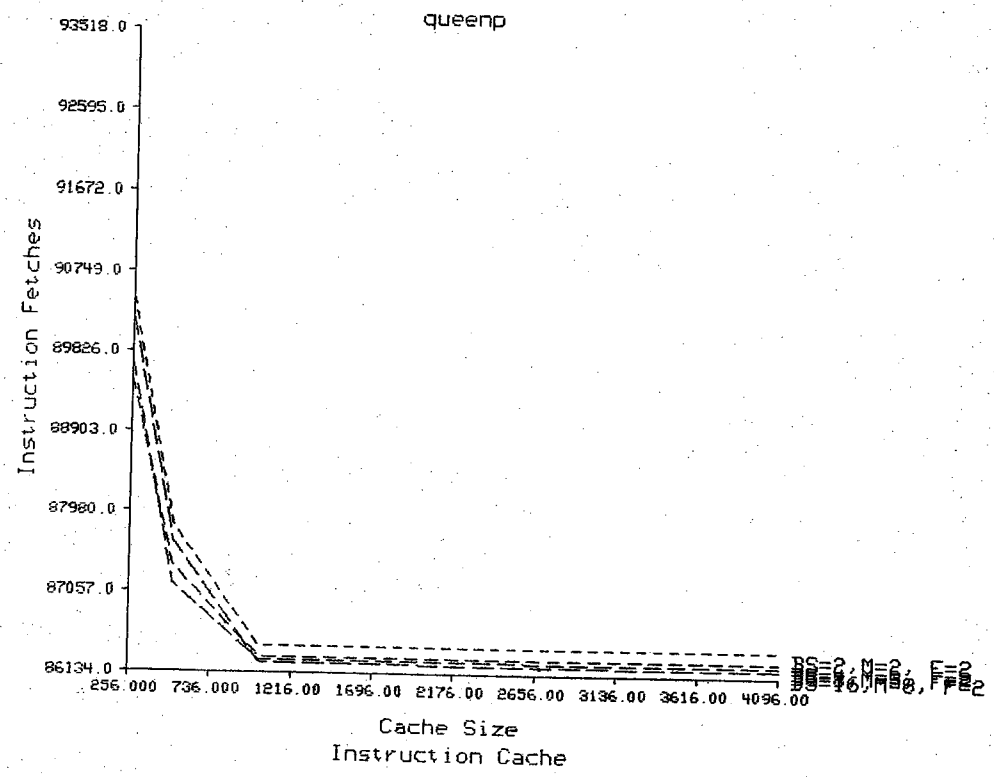


Figure 6.29 Execution Time of *Queen* with Large Base Delays and Small Transfer Delays for GaAs Parameters

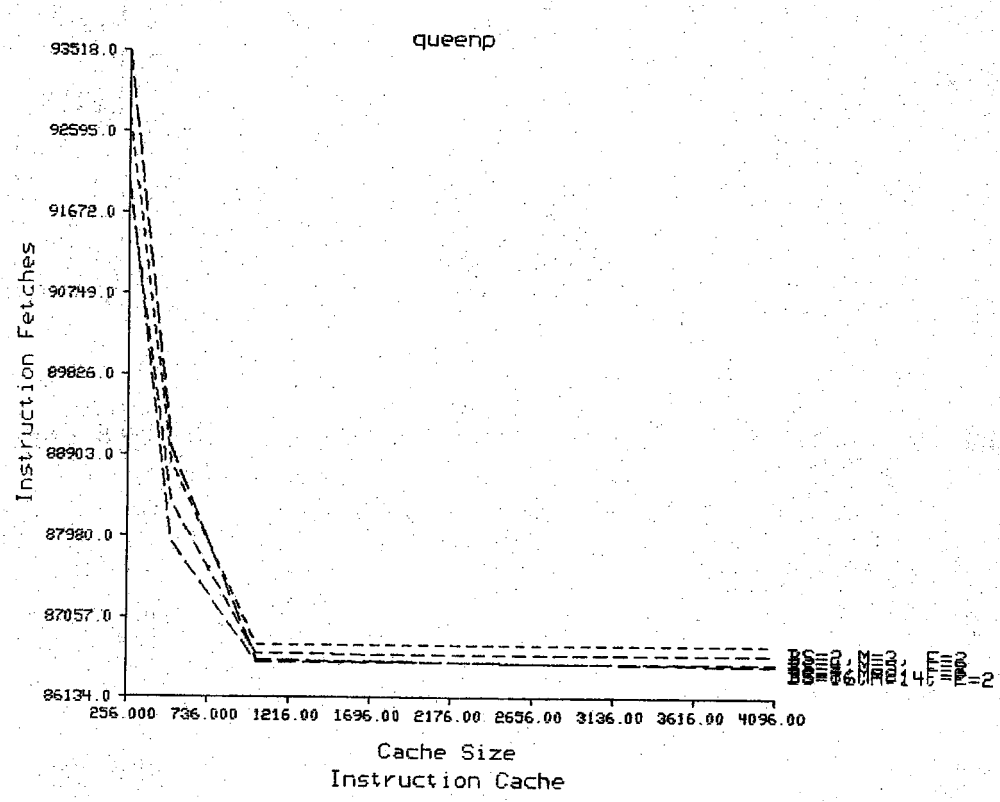


Figure 6.30 Execution Time of *Queen* with Large Base Delays and Large Transfer Delays for GaAs Parameters

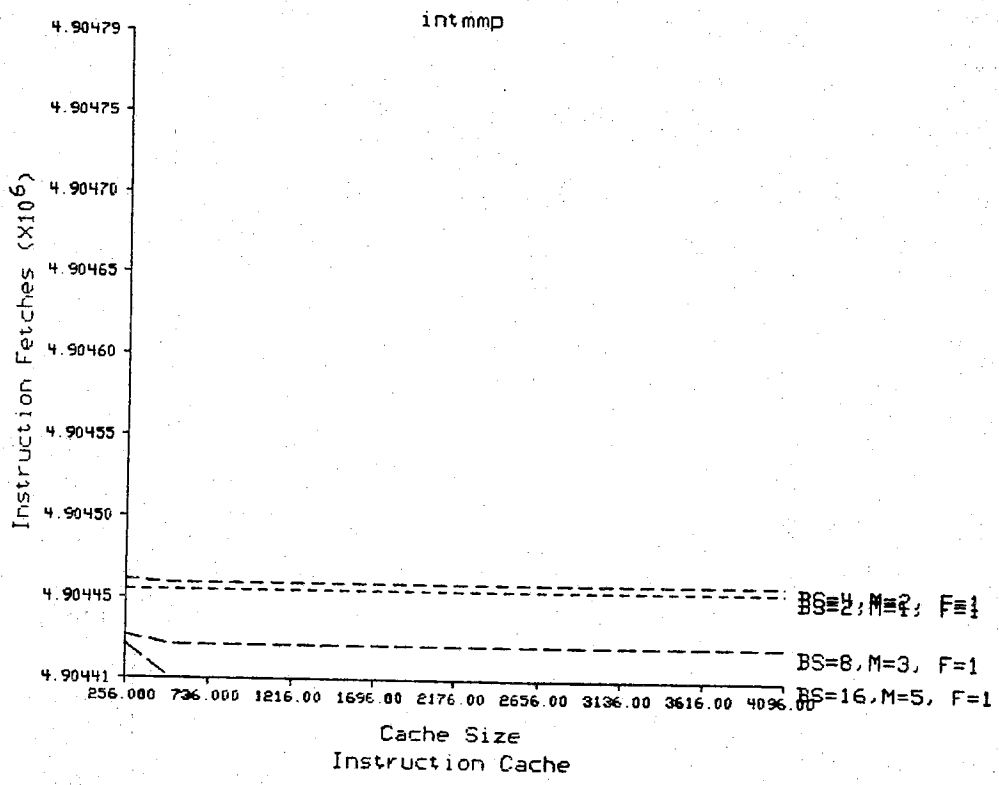


Figure 6.31 Execution Time of *Intmm* with Small Base Delays and Small Transfer Delays for Si Parameters

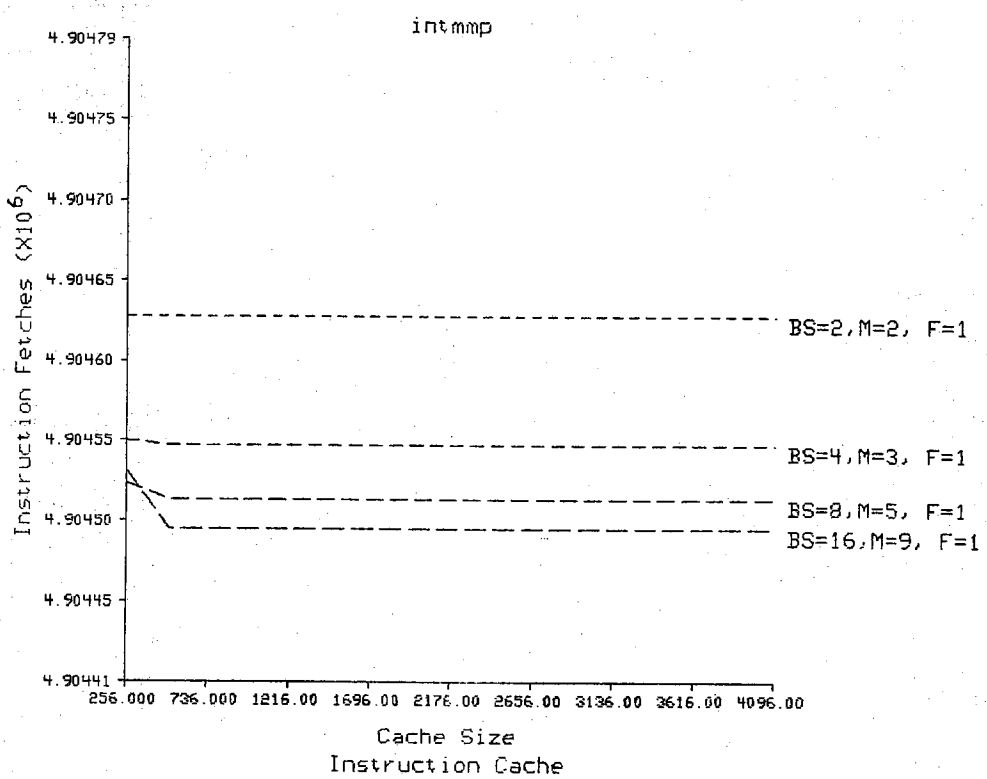


Figure 6.32 Execution Time of *Intmm* with Small Base Delays and Large Transfer Delays for Si Parameters

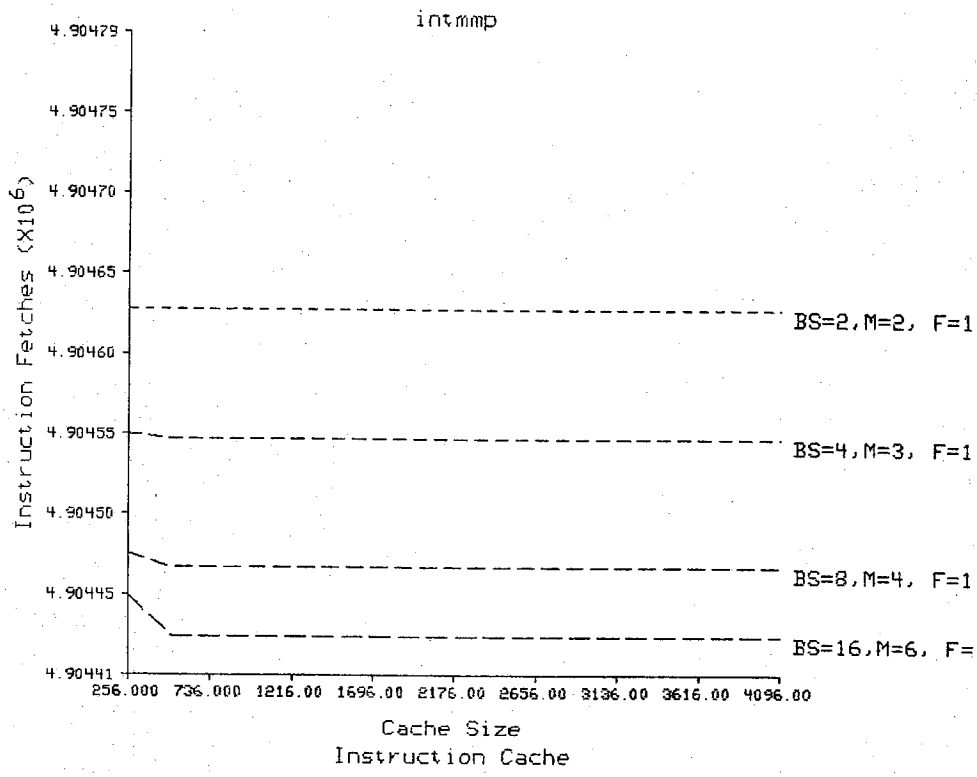


Figure 6.33 Execution Time of *Intmm* with Large Base Delays and Small Transfer Delays for Si Parameters

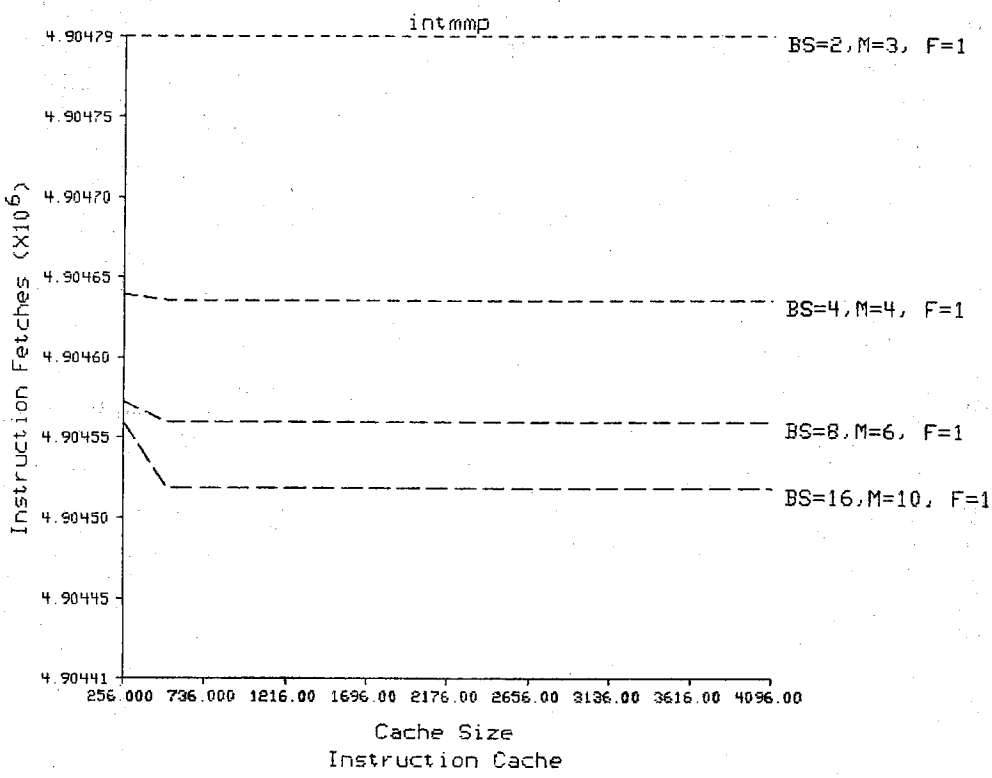


Figure 6.34 Execution Time of *Intmm* with Large Base Delays and Large Transfer Delays for Si Parameters

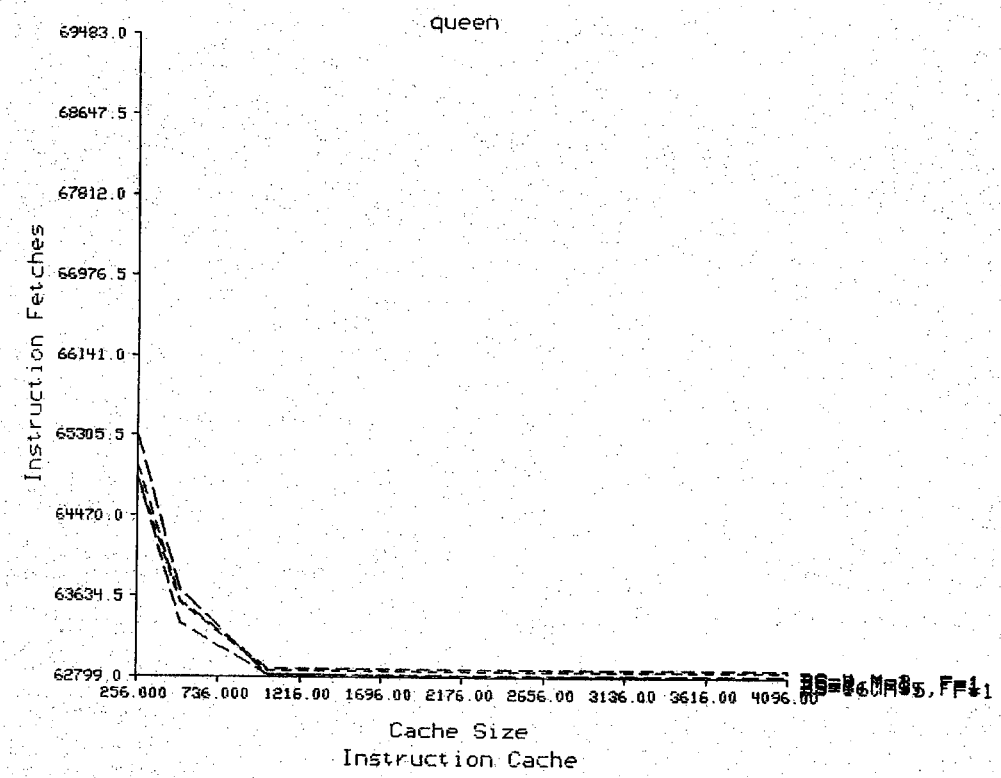


Figure 6.35 Execution Time of *Queen* with Small Base Delays and Small Transfer Delays for Si Parameters

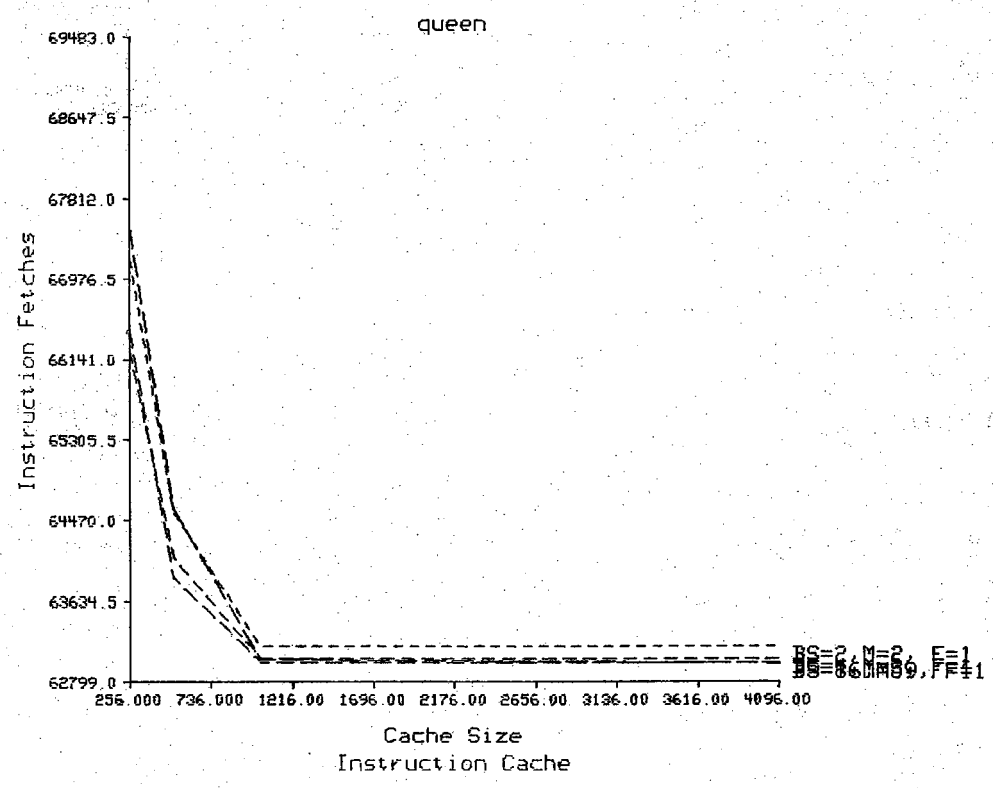


Figure 6.36 Execution Time of *Queen* with Small Base Delays and Large Transfer Delays for Si Parameters

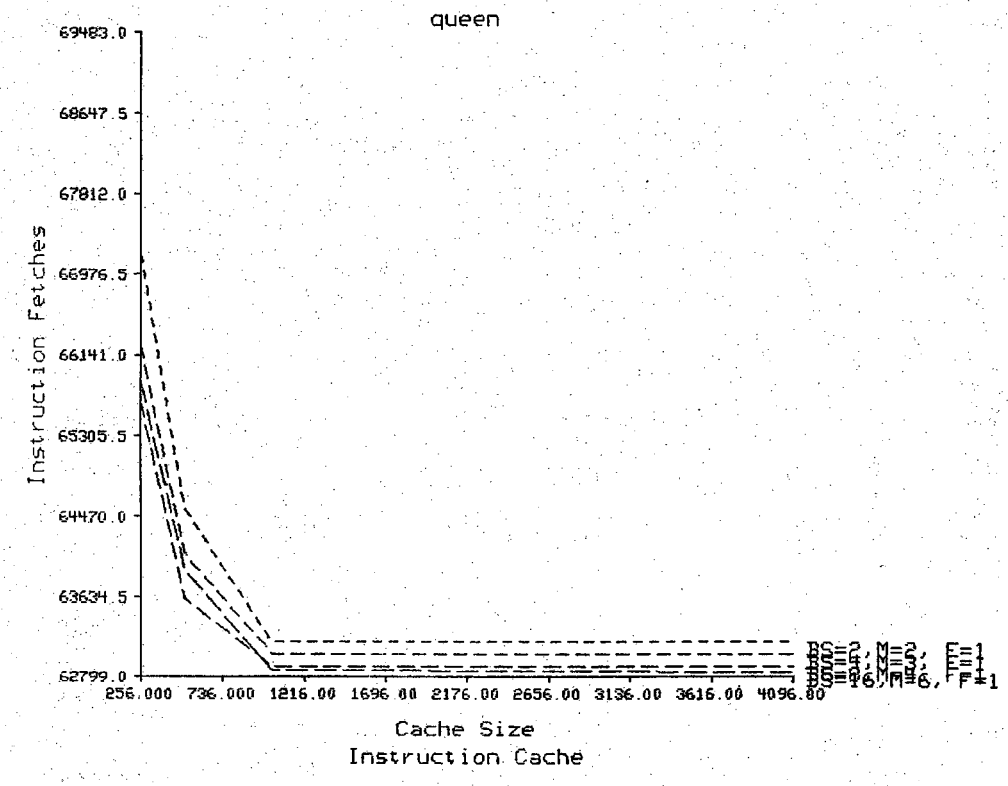


Figure 6.37 Execution Time of *Queen* with Large Base Delays and Small Transfer Delays for Si Parameters

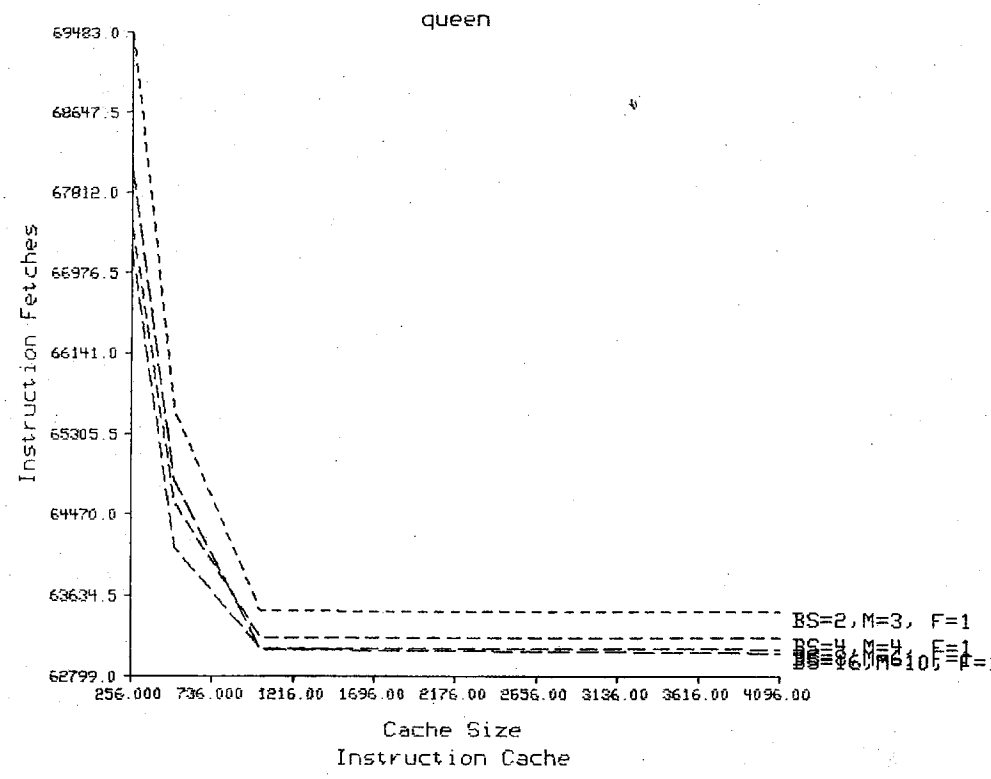


Figure 6.38 Execution Time of *Queen* with Large Base Delays and Large Transfer Delays for Si Parameters

CHAPTER 7
SUMMARY

This thesis has described the technology factors affecting the design of three VLSI structures: adders, multipliers, and cache memory. After a brief discussion of the testing choices, a general description of the work will conclude this chapter.

Each of the three structures was examined to determine how it was affected by the technology. The three structures were then analyzed to determine how each should be implemented. Some of the possible options were rejected for one of several reasons. Some options were discarded because they were impossible to implement. Other options were chosen because previous work with GaAs technologies had shown that these were better for the restrictions imposed by GaAs E/D-MESFET technology.

The adders examined are ripple-carry adders, carry-select adders, and full carry look-ahead adders. I also chose a full range of bit lengths from 1 to 40 bits. The fanin and fanout maximums examined are fanin=2 fanout=2, fanin=2 fanout=5, and fanin=5 fanout=5.

The multiplier options examined are the booth-step algorithm with a full barrel shifter, on-chip bit-serial multiplier with 3 position barrel shifter, and off-chip bit-serial multiplier with 3 position barrel shifter and a larger register file.

The cache options were presented to provide more information for the multiplier choice. The cache options discussed are a direct-mapped cache with block sizes of two, four, eight, and sixteen, for cache sizes of 256, 512, 1024, and 4096 blocks. The choices of instruction cache, data cache, or combined data/instruction cache were made in conjunction with delays induced by various fetch times, various miss times.

Tests were then devised that measured the effect of changing each parameter individually. The range of parameters for each test was varied for both silicon CMOS/SOS technologies and GaAs E/D-MESFET technologies. These results can be used to determine the parameters that will yield the best

performance for each set of options. A range of options and associated parameters can then be used to improve the design of GaAs structures.

7.1. Adders

As expected, each adder ran ran faster in GaAs than in silicon. Although the change in adder type changed the performance by up to 50 percent, the change was less in GaAs than in silicon. A greater distinction between the adders was provided by the area computations. The carry select and the ripple carry adders both consume about the same chip area, while the full carry look-ahead adders consume much more area for adders longer than 20 bits. This suggests that the GaAs environment cannot support full carry look-ahead adders while the silicon environment can. The ripple carry adder is almost as good as the carry select adder when basing the choice solely on minimum area and maximum speed. High speed applications with a little free area can use carry select adders, while applications with no extra area can use the ripple carry adder with little penalty. This is different from the silicon environment where the ripple carry adder is much slower than either of the other two adders which makes it unusable for all but the slowest applications.

7.2. Multiplication

The proposed modifications to the multiplier to enhance performance did not give the expected results. The limits of the simulation tools did not allow us to propose architectures which may have improved performance and verify that improvement. The conclusion is that the original SU-MIPS architecture was the best architecture for the supplied benchmarks. The instruction mix of the benchmarks caused a degradation of performance when the full barrel shifter was replaced by a limited barrel shifter. Since the bit-serial multiplier was unable to compensate for the loss of execution time, the overall performance suffered.

Moving the multiplier off-chip and replacing the barrel shifter might still be a good alternative, but the possibilities could not be tested without more flexible software. The compiler and reorganizer available did not use the register file well enough to make a larger register file worthwhile.

The conclusion is that in either the GaAs or silicon environment, the original SU-MIPS performs the best. The analysis also showed that the E/D-MESFET architecture benefited more from the bit-serial multiplier than the CMOS/SOS architecture. Therefore, such strategies should not yet be discarded for GaAs designs.

7.3. Cache

The results of the cache experiments can be used to help determine the effect of the choice of multipliers. The multiplier section can be used to help determine the increase in the amount of code. This information can then be used with the cache performance to determine the resulting execution time. This can be used to match the cache to the multiplier based on the cache size, block size, and transfer times.

The execution time of the benchmarks based on cache type clearly shows that an instruction cache improves performance more than a data cache. When common data memory and instruction memory are used, the combined instruction/data cache performs better than the instruction cache. The performance for each benchmark also improved with reduced block size. The change in performance between the cache types was more pronounced for the GaAs parameters than for the silicon parameters.

When comparing the effects of the miss parameters, the cache size did not affect the performance in a significant manner. The block size produced the biggest difference in performance. As the block size varied, the time per word transferred (transfer time) had a bigger effect for GaAs parameters than for silicon parameters while the base delay had a bigger effect for silicon parameters than for GaAs parameters. Overall, the transfer time was the dominant factor for GaAs caches while both parameters had similar effects for silicon caches. Although this information is useful, its prime importance is in determining how to implement multiplication.

7.4. Conclusion

New design methodologies for GaAs E/D-MESFET technologies have been presented for a specific architecture and specific problems. The advantages and disadvantages of GaAs E/D-MESFET technologies and how they relate to silicon technology have also been presented as they relate to actual problems. This information has been given as it applies to adder design, multiplier design, and cache effects on multiplier design. With this information, a digital designer should now have a better insight into the choices for optimal GaAs E/D-MESFET designs, both modified silicon designs and unique GaAs designs.

LIST OF REFERENCES

LIST OF REFERENCES

[AsKuH83] Asai, K., Kurumada, K., Hiriyama, M., Ohmori, M., *1Kb Static RAM using Self-Aligned FET Technology,* Proceedings of the 1983 IEEE International Solid-State Circuits Conference, New York City, New York, February 1983, pp. 46-47.

[Barne85] Barney, C., "DARPA Eyes 100-mips GaAs Chip for Star Wars," *ElectronicsWeek* , Vol. 58, No. 20, May 20, 1985, pp. 22-23.

[BasNu84] Bass, S., Neudeck, G., "VLSI Transistor Count and Basic Delays," *Internal Report*, Purdue University, 1984.

[Betti85a] Bettinger, M. K., "Comparison of System Issues between CMOS Silicon and GaAs," Purdue University, Dec. 1985

[Betti85b] Bettinger, M. K., "Adder Design Issues," Research Report, Purdue University, March. 1985

[BeDoF81] Beyers, J., Dohse, L., Fucetola, J., Kochis, R., Lob, C., Taylor, G., Zeller, E., *A 32-Bit VLSI CPU Chip,*" IEEE Journal of Solid-State Circuits, Vol. SC-16, pp.537-541, Oct. 1981.

[Coope84a] Cooper, J.A. Jr., PEEII Lecture, Purdue University, 1984.

[Coope84b] Cooper, J.A. Jr., Private Communication, Purdue University, 1984.

[EdLiW83] Eden, R.C., Livingston, A.R., Welch, B.M., *Integrated Circuits: the Case for Gallium Arsenide,* IEEE Spectrum, Vol. 9, No. 12, December 1983, pp.30-37.

[EdWeZ79] Eden, R.C., Welch, B.M., Zucca, R., Long, S.I., *The Prospects for Ultrahigh-Speed VLSI GaAs Digital Logic,* IEEE Journal of Solid-State Circuits, Vol. sc-14, No. 2, April 1979, pp. 221-239.

[Fura85] Fura, D., "Architectural Approches for Gallium Arsenide Exploitation in High-Speed Computer Design," MSEE Thesis, December, 1985.

[FuTaI84] Furutsuka, T., Takahashi, K., Ishikawa, S., Yano, S., Higashisaka, A., "A GaAs 12 x 12 Bit Expandable Parallel Multiplier LSI Using Sidewall-Assisted Closely-Spaced Electrode Technology," *Proceedings of the International Electron Devices Meeting*, San Francisco, California, December 1984, pp. 344-347.

[GiGrH83] Gill, J., Gross, T., Hennessy, J., Jouppi, N., Przybylski, S., and Rowen, C. *Summary of MIPS instructions,* Technical Note 83-237, Stanford University, November, 1983.

[GroHe82] Gross, T.R., Hennessy, J.L., *Optimizing Delayed Branches,* Proceedings of Micro-15, IEEE, Octover 1982.

[GroGi83] Gross, T., Gill, J. *A Short Guide to MIPS Assembly Instructions,* Technical Note No. 83-236, Stanford University, November, 1983.

[Gross83] Gross, T., *Code Optimization of Pipeline Constraints,* PhD dissertation, Stanford University, September, 1983.

[Gross84] Gross, T. *MIPS-SIM: A MIPS Simulator,* Stanford University, April, 1984.

[Heage85] Heagerty, B., Private Communication, RCA-ATL, 1984.

[HeJoG82] Hennessy, J., Jouppi, N., Gill, J., Baskett, F., Strong, A., Gross, T., Rowen, C., Leonard, J., "The MIPS Machine," *Digest of Papers, Spring COMPCON 82,* San Francisco, California, February 1982, pp. 2-7.

[HeJoP83] Hennessy, J.L., Jouppi, N., Przybylski, S., Rowen, C., Gross, T., "Design of a High Performance VLSI Processor," Stanford University Technical Report No. 236, Feb. 1983.

[HeScZ85] Helbig, W.A., Schellack, R.H., Zieger, R.M., "The Design and Construction of a GaAs Technology Demonstration Microprocessor," *Proceedings of Midcon/85,* Chicago, Illinois, September 1985, pp. 23/1.1-23/1.6.

[HiInM84] Hiriyama, M., Ino, M., Matsuoka, Y., Suzuki, M., *A GaAs 4Kb SRAM with Direct Coupled FET Logic,* Proceeding of the 1984 Ieee International Solid-State Circuits Conference, San Francisco, CA, February 1984, pp. 46-47.

[Hoeff84] Hoefflinger, B., Private Communication, Purdue University, 1984.

[HwaBr84] Hwang, K., Briggs, F.A., *Computer Architecture and Parallel Processing,* McGraw-Hill, 1984.

[Hwang84] Hwang, K., Private Communication, Purdue University, 1984.

[IkToM84] Ikawa, Y., Toyoda, N., Mochisuki, M., Terada, T., Kanazawa, K., Hirose, M., Mizoguchi, T., Hojo, A., *A 1K-Gate GaAs Gate Array,* Proceeding of the 1984 Ieee International Solid-State Circuits Conference, San Francisco, CA, February 1984, pp. 40-41.

[IsInI84] Ishii, Y., Ino, M., Idda, M., Hirayama, M., Ohmori, M., "Processing Technologies for GaAs Memory LSIs," *Proceedings of the GaAs IC Symposium*, Boston, Massachusetts, October 1984, pp. 121-124.

[Kabak86] Kabakibo, A., "A Comparison of E/D-MESFET GaAs and CMOS Silicon for High-Speed Cache Design," MSEE Thesis, December, 1986.

[Katev83] Katevenis, M.G.H., "Reduced Instruction Set Computer Architectures for VLSI," *Report No. UCB/CSD 83/141*, University of California at Berkeley, October 1983.

[Knuth71] Knuth, D. E., "An Empirical Study of FORTRAN Programs," Software-Practice and Experience, 1971, pp. 105-133.

[LeKaW82] Lee, F.S., Kaelin, G.R., Welch, B.M., Zucca, R., Shen, E., Asbeck, P., Lee, C.P., Kirkpatrick, C.G., Long, S.I., Eden, R.C., *A High-Speed LSI GaAs 8x8 Bit Parallel Multiplier,* IEEE Journal of Solid-State Circuits, Vol. sc-17, No. 4, August 1982, pp. 638-647.

[MiFuH86] Milutinović, V., Fura, D., Helbig, W., "An Introduction to GaAs Microprocessor Architecture for VLSI," *IEEE Computer*, Vol. 19, No. 3, March 1986.

[MiSiF86] Milutinović, V., Silbey, A., Fura, D., Bettinger, M., Keirn, K., Helbig, W., Heagerty, W., Zieger, R., Schellack, R., Curtice, W., "Design Issues in GaAs Computer Systems," *IEEE Computer,* Vol. 19, No. 10, October 1986.

[Namor84] Namordi, M.R., GaAs Seminar presented at Purdue University, October 1984.

[NaSuS83] Nakayama, Y., Suyama, K., Shimizu, H., Yokoyama, N., Ohnishi, H., Shibatomi, A., Ishikawa, H., *A GaAs 16x16 Bit Parallel Multiplier,* IEEE Journal of Solid-State Circuits, Vol. sc-18, No. 5, October 1983, pp. 599-603.

[NuBeD81] Nuzillat, G., Bert, G., Damay-Kavala, F., Arnodo, C., *High-Speed Low-Power IC's Using Quasi-Normally-Off GaAs MESFET's,* IEEE Journal of Solid-State Circuits, Vol. sc-16, No. 3, June 1981, pp. 226-232.

[NuPeB82] Nuzillat, G., Perea, E.H., Bert, G., Damay-Kavala, F., Gloanec, M., Ngu, T.P., Arnodo, C., *GaAs MESFET IC's for Gigabit Logic Applications,* IEEE Journal of Solid-State Circuits, Vol. sc-17, No. 3, June 1982, pp. 569-584.

[Patte85] Patterson, D.A., "Reduced Instruction Set Computers," *Communications of the ACM,* Vol. 28, No. 1, January 1985, pp. 8-21.

[PeDaN83] Perea, E.H., Damay-Kavala, F., Nuzillat, G., Arnodo, C., *A GaAs Low-Power Normally-On 4 Bit Ripple Carry Adder,* IEEE Journal of Solid-State Circuits, Vol. sc-18, No. 3, June 1983, pp. 365-369.

[Radin83] Radin, G., "The 801 Minicomputer," *IBM Journal of Research and Development,* Vol. 27, No. 3, May 1983, pp. 237-245.

[Sherb84] Sherburne, R.W. Jr., "Processor Design Tradeoffs in VLSI," *Report No. UCB/CSD 84/173,* University of California at Berkeley, April 1984.

[SilMi85] Silbey, A. A., Milutinovic, V., *System Design Considerations in a GaAs Microprocessor System,* Purdue University Research Report, May, 1985.

[Smith82] Smith, A. J., *Cache Memories,* Computing Surveys, Vol. 14, No. 3, Sep. 1982, pp. 473-530.

[TI84] Texas Instruments, *HDL Design/Simulator User's Manual,* Version 2.0, Dec. 12, 1984.

[VanLi74] Van Tuyl, RR.L., Liechti, C.A., *High-Speed Integrated Logic with GaAs MESFET's,* IEEE Journal of Solid-State Circuits, Vol. sc-9, No. 5, October 1974, pp. 269-276.

[VuRoN84] Vu, T.T., Roberts, P.C.T., Nelson, R.D., Lee, G.M., Hanzal, B.R., Lee, K.W., Zafar, N., Lamb, D.R., Helix, M.J., Jamison, S.A., Hanka, S.A., Brown, J.C. Jr., Shur, M.S., *A Gallium Arsenide SdFl Gate Array with On-Chip RAM,* IEEE Journal of Solid-State Circuits, Vol. sc-19, No. 1, February 1984, pp. 10-22.

[Walle84] Waller, L., *GaAs ICs Bid for Commercial Success,* Electronics, Vol. 57, No. 12, June 14, 1984, pp. 101-102.

[YaHiA83] Yamamoto, R., Higashisaka, A., Asai, S., Tsuji, T., Takayama, Y., Yano, S., *Design and Fabrication of Depletion GaAs LSI High-Speed 32-Bit Adder,* IEEE Journal of Solid-State Circuits, Vol. sc-18, No. 5, October 1983, pp. 592-599.