Resource Allocation in a High Clock Rate Microprocessor

Michael Upton, Thomas Huff, Trevor Mudge & Richard Brown

Department of Electrical Engineering and Computer Science
University of Michigan
email: upton@eecs.umich.edul, thuff@eecs.umich.edu

Abstract

This paper discusses the design of a high clock rate (300MHz) processor. The architecture is described, and the goals for the design are explained. The performance of three processor models is evaluated using trace-driven simulation. A cost model is used to estimate the resources required to build processors with varying sizes of on-chip memories, in both single and dual issue models. Recommendations are then made to increase the effectiveness of each of the models.

Keywords: *Pipelining, decoupled architecture, prefetching, non-blocking cache, resource allocation, superscalar, floating point latencies.*

1 Introduction

This paper describes the decisions made in the design of an experimental high-performance general purpose processor suitable for a workstation or a high end PC system. The goals include integer performance of 200 SPECint and floating point performance of 300 SPECfp, with power dissipation such that the system can be air cooled. Fast switching speeds led us to choose Gallium Arsenide Direct Coupled FET Logic (DCFL) combined with MCM technology to build a compact multiple-chip processing element. The MIPS R3000 ISA was chosen because it is a relatively clean architecture with good architectural development tools.

Figure 1 shows single chip microprocessor clock frequencies for processors presented at the International Solid State Circuit Conference. Over the past 10 years the frequency of microprocessors has increased at a rate of about 40% per year. The fastest chip introduced each year is usually at least twice as fast as the slowest chip for that year, and the gap has been widening. The fastest clock rate processors do not necessarily have the highest system level performance, but the overall trend reflected in this data strongly suggests that processor clock frequencies will continue to increase.

In contrast, the rate of increase in main memory speed has been lower than that of CPUs. The core-based main memory access time of the CDC6600 was 1000ns [15]. Main memory operations on the 1978-vintage VAX 11-780 required 6 cycles of 200ns, or 1200ns. Access times in current high-end workstations such as the HP 9000/735 have decreased to around 150ns, but this may be as many as 20 cycles. Over the past decade, main memory speeds have improved by only a factor of 10, while processor speeds have increased by a factor of 100. As cycle times continue to improve, the primary cache miss penalties will rise from their current levels to as many as 100 clock cycles.

The increasing performance difference between processor and memory speeds, combined with large but finite transistor budgets, presents many interesting trade-offs for computer architects. Given a limited resource budget, the designer must choose the allocation of resources needed to optimize performance. With a limited chip area, resources could be devoted to increasing the on-chip cache size, adding other memory structures, or adding execution units. At high clock rates, additional execution units may be starved for data because of long memory latencies.

This paper examines the resource allocation in superscalar multi-chip microprocessor design. The architecture is briefly described and the goals for the design are explained. The performance of three processor configurations is evaluated using trace-driven simulation. A cost model is used to estimate the resources required to build processors with varying sizes of on-chip memories in both single and dual issue versions. Recommendations are then made to increase the effectiveness of each of the models.

2 System Overview

The Aurora III microprocessor is the culmination of three years of research on implementing pipelined microprocessors using GaAs technology. A block diagram of the Aurora III system is shown in Figure 2. The system is composed of four custom GaAs chips: three logic chips and a 32 K-bit SRAM used for building a 64 K-byte data cache. The logic chips are the Floating Point Unit (FPU), Integer Processing Unit (IPU) and Memory Management Unit (MMU). This partitioning is similar to the SGI R8000 TFP processor [6].

For the system level performance of our GaAs chipset to be competitive with that of contemporary CMOS processors the GaAs system must overcome with increased clock speed the CMOS advantage of much higher integration density, which allows increased parallelism and larger caches. We have explored processor microarchitectures that allow parallel instruction issue without adversely impacting system cycle time.

The IPU consists of five functional modules that operate semi-autonomously to fetch, decode, execute and retire instructions.

ASPLOS VI- 10/94 San Jose, California USA
© 1994 ACM 0-89791-660-3/94/0010..$3.50

This work was supported by the Denfese Advanced Research Projects Agency under Contract Number DAAL03-90-C-0028.

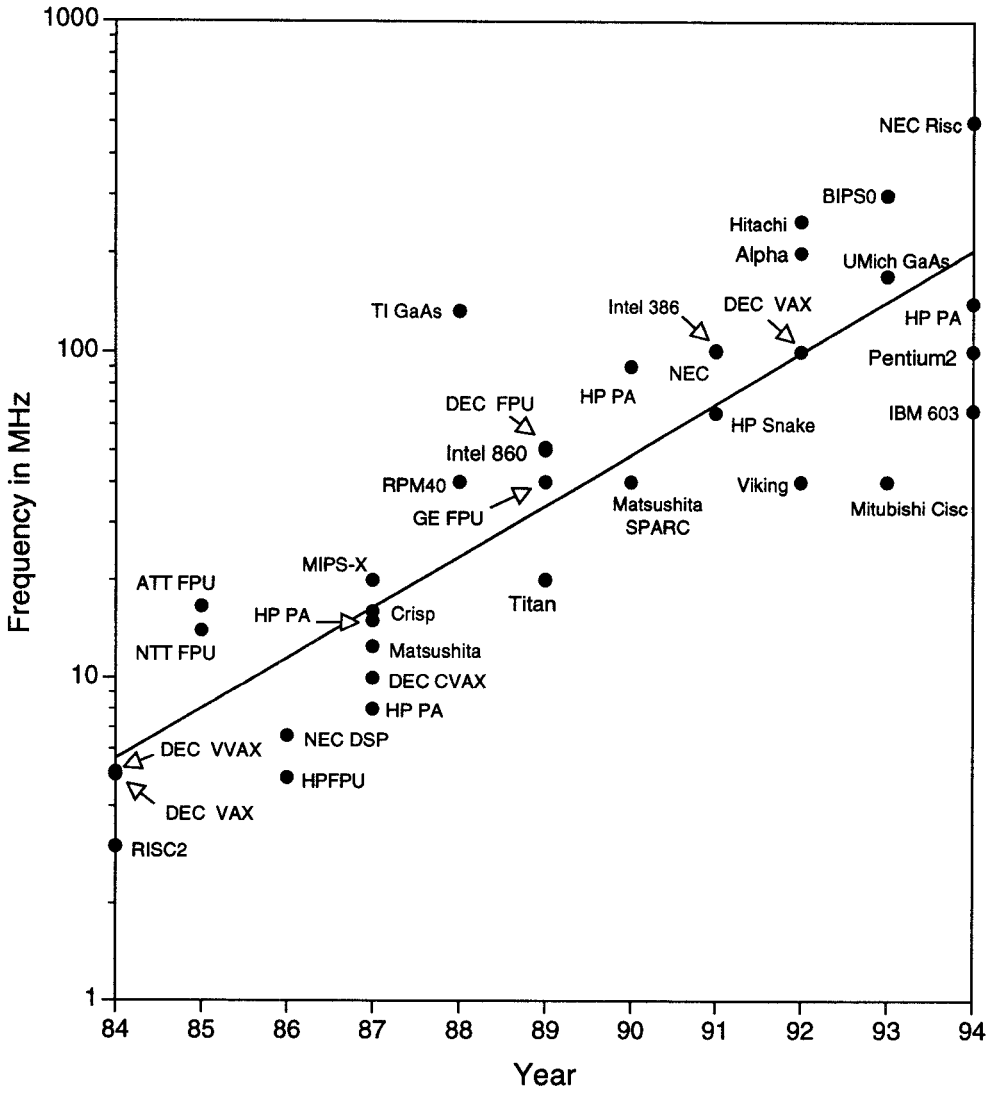


Figure 1: Microprocessor Clock Frequencies

Single chip microprocessor clock frequencies for processors presented at the last eleven International Solid State Circuit Conferences. The line shown represents about a 40% increase per year in frequency.

The IPU is similar to the IBM-Motorola PowerPC 603 and 604 processors [2] in that it includes a Bus Interface Unit (BIU), an Integer Execution Unit (IEU), an Instruction Fetch Unit (IFU), and a Load Store Unit (LSU). In addition to these the IPU has a dedicated Prefetch Unit (PFU) for data and instructions.

Bus Interface Unit The Aurora III memory system is designed to provide a large memory bandwidth from main memory through the MMU to the primary caches. Sustained transfer rates of over 1.5 G-bytes per second have been demonstrated in the BIU performance model for typical workloads [14]. Long memory latencies require high bandwidth to support prefetching and nonblocking caches [9]. The asynchronous BIU interface allows external communication and internal computation to proceed independently.

The IPU is connected to the MMU by a bidirectional 32-bit bus. To tolerate the transaction latency in the system, multiple pending requests are buffered in separate transmit and receive queues. A split-transaction collision-based protocol allows either the IPU or MMU to immediately send data without arbitrating for the bus. Using a Rambus-like signaling scheme [10], the clock is generated on-chip and sent with the data to minimize clock skew. Data is transferred on both edges of the clock to maximize IO bandwidth.

Instruction Fetch Unit The IFU fetches instructions and maintaining the state of the on-chip Instruction Cache. Icache misses are detected in the IFU, and the missing instructions are requested from the MMU via the BIU. The front of the IEU pipeline stalls until the needed instructions arrive, but the LSU continues to process pending data cache misses, and the reorder buffer continues to retire completed instructions.

An on-chip instruction cache was required to support a two-instruction-per-cycle issue rate. Too few IO pins were available to fetch 64 instruction bits each cycle. There are few restrictions on which instructions can issue together. Other than true instruction dependencies, in which an instruction uses the result of the immediately preceding instruction, the primary issue constraint is that only a single memory access instruction can be executed in a given cycle.

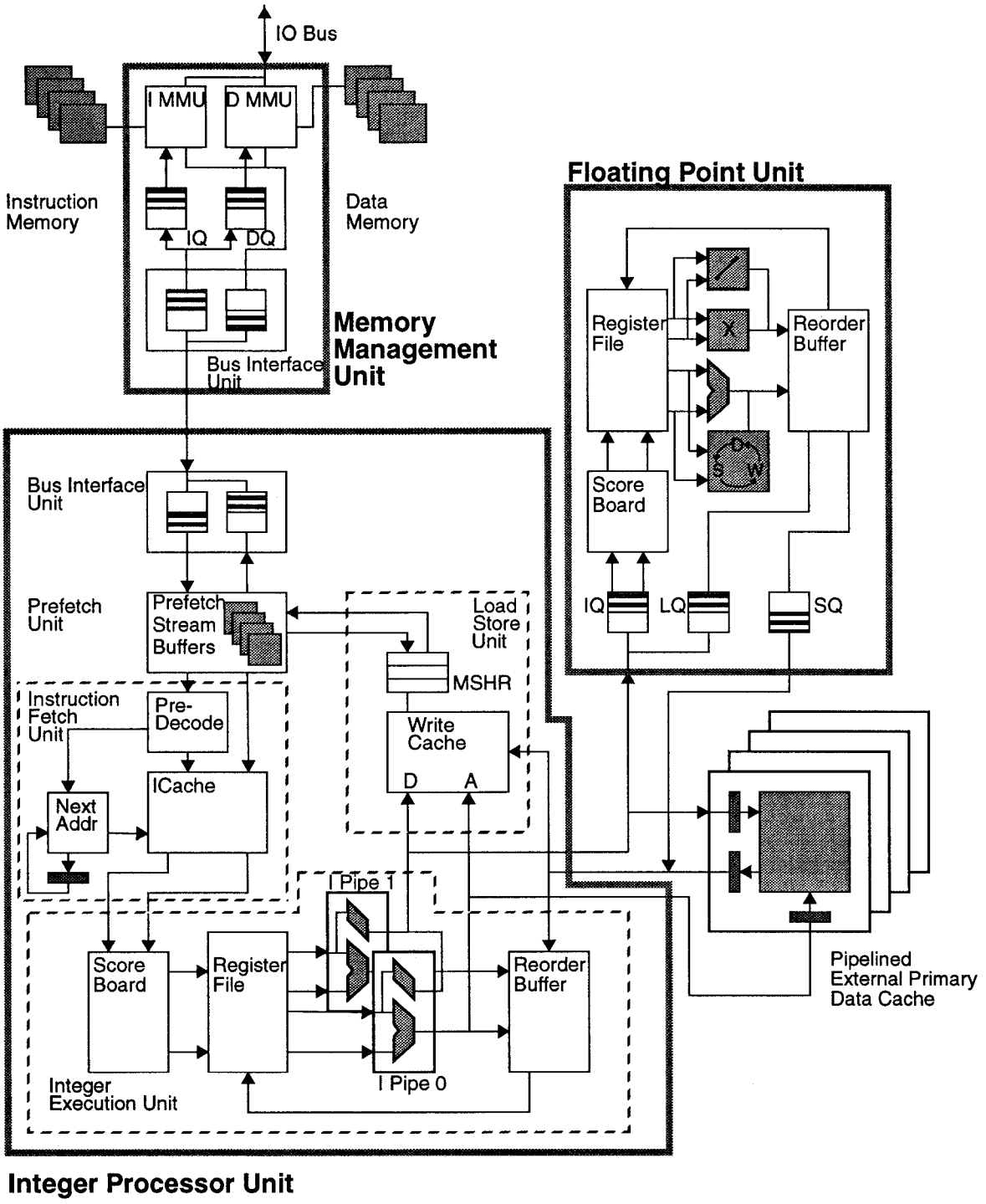


Figure 2: Processor Block Diagram

To speed the instruction issue logic, instructions are predecoded before being inserted into the instruction cache. Figure 3 shows the arrangement of a decoded instruction. All instructions are grouped into pairs, with the EVEN instruction occupying the lower of two consecutive addresses, and the ODD occupying the next sequential address. The DI bit indicates whether an instruction pair dependency prohibits dual issue. The CONT field indicates whether the instruction pair includes a control flow instruction, such as a jump or branch. The MIPS ISA prohibits a branch instructions in a branch delay slot, so there will be a maximum of one control flow instruction in each pair. If the instruction pair contains a control flow instruction, the NEXT field contains the cache index of the target instruction. This branch folding [3] reduces the critical path for a dual issue machine by eliminating

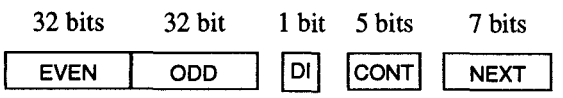


Figure 3: Decoded Instruction Cache Fields

the branch pipeline bubble. The target of the branch can be fetched on the next cycle, without needing to compute a target address from the address of the branch instruction.

2.1 Instruction Execution Unit

The key elements of the IEU are the 32x32 integer register file and two execution pipelines, which consist of an ALU and a six-entry reorder buffer [13]. A register file scoreboard [15] detects instruction dependencies and stalls execution until the needed operands are ready.

To achieve high system clock frequencies, both the area and the number of logic levels needed to implement a logic function must be minimized. Among the main contributors to die area in our previous GaAs microprocessors were the pipeline latches. These latches increase the operating frequency by breaking long operations into smaller stages, but they also increase the layout area. In addition, every pipeline stage after the ALU, such as load delay slots, requires a forwarding path back to the ALU inputs. The state latches and forwarding network for one of our earlier prototypes account for 50% of the execution pipeline area[16].

One way to minimize the area penalty for state latches is to have a short execution pipeline. Like the Motorola 88K [1], we have adopted variable length pipelines, reducing the length of the integer pipelines to four cycles. Memory instructions require an additional two or three cycles to produce their results. Virtual addresses are generated in either of two identical integer ALUs. The address and any store data are transmitted to the LSU and data cache in one cycle. Forwarding paths allow the ALU output and the reorder buffer contents to be used as inputs for the next instruction, avoiding pipeline bubbles.

2.2 Prefetch Unit

A set of hardware prefetch buffers is included to minimize long memory system latencies caused by the fast clock rate and multiple chip partitioning. The prefetch buffers predict future memory requests and bring the data on chip before it is referenced by the IPU.

Jouppi proposed the addition of a small set of associative prefetch buffers, called stream buffers, to fetch sequential lines ahead of the current program counter [7]. The stream buffer consists of a tag register, a tag comparator, a set of status bits and a number of prefetch cache lines for instructions and data. If a memory reference misses in the primary cache, the stream buffers are checked to see whether the required data have already been requested. Jouppi showed that stream buffers can be highly effective for small caches, reducing the cache miss rate by up to half. Since we have limited space for on-chip caches, these are an ideal solution.

On each instruction or data cache miss, a stream buffer is allocated and initialized to fetch the next sequential line. This buffer initially fetches only a single line. If a subsequent request hits in a prefetch buffer, additional sequential lines are fetched until the buffer is filled.

2.3 Load-Store Unit

To simplify the execution unit pipelines, a separate function unit processes all memory operations. The IEU computes the memory address in its ALU and checks for any illegal address exceptions. The address and store data for store instructions are passed to the LSU. The IEU also provides a 3-bit tag value indicating the reorder buffer destination slot.

To accommodate a data cache large enough to achieve good system-level performance, the cache RAMs must be external to the CPU. The system supports external caches of 16 K-, 32 K-, and 64 K-bytes. The external data cache is direct-mapped, pipelined, and has a three-cycle latency. A new access can be initiated each cycle. The cache is accessed using two unidirectional 64-bit data busses and a 14 bit address bus.

To amortize the long primary cache miss penalty, the processor supports multiple outstanding cache misses. As long as the target register of a load instruction is not referenced, other instructions can issue, execute and complete, leaving their results in the reorder buffer. Even more importantly, the access delay of multiple cache misses can be overlapped. A number of Miss Service Holding Registers (MSHRs) maintain the state of pending cache misses. An MSHR is reserved for each memory instruction active in the LSU pipeline, and if no MSHRs are available, the processor stalls until one is free. A machine with only one MSHR cannot overlap memory operations, and must process each load or store sequentially.

Write Cache The LSU contains a 32-word coalescing write buffer called the Write Cache [8]. The 32 data words are organized as four cache lines of eight words per line. The four lines are fully associative.

The write cache groups multiple memory references into a single BIU transaction. Memory write behavior has two characteristics that are effectively exploited by the write cache. Multiple writes will often occur to the same address, as would happen in an inner loop during the updating of the loop index. After the first write to the index address, subsequent writes would hit in the write cache, replacing the previous value. Thus fewer BIU transactions are required to keep the memory system coherent. The second memory access pattern that benefits from a write cache is vector-like operations such as memory copies or floating-point intensive code. In these operations, each entry in the write cache is written in succession, but only one BIU transaction is needed to retire the eight words.

Write Validation Because the MMU is off-chip, the processor cannot retire store instructions until it is known that the address accessed will not cause a memory fault. Receiving a response from the MMU requires many cycles, so simply querying the MMU about each write address is an unacceptable solution. Instead the write cache divides the address tags into page and offset fields. If the page field of the current write address matches any of the valid page fields contained in the write cache, then a write or access fault is not possible. In effect, the write cache operates as a four entry micro-TLB.

Floating Point Support In this architecture, floating point memory instructions transfer data directly between memory and the integer and floating point register files. Unlike integer stores, floating point store data is not immediately ready at the time the instruction is transferred to the LSU. Thus, write cache eviction and data cache writeback must wait for the data. This adds a certain degree of complexity to the synchronization of the floating point data and the cache line within the write cache. In order to meet package pin constraints, all floating point instruction and data transfers occur via the input and output busses of the primary data cache.

2.4 The Branch Delay Slot Problem in Superscalar Architectures

Although modern RISC machines are designed to have few implementation artifacts, a notable exception is the architectural branch delay slot. The branch delay slot was added to allow the processor to execute an instruction while the instruction at the branch target was being fetched, avoiding bubbles in the pipeline. Because architectural branch delay slots significantly complicate the design of superscalar machines, they have been eliminated from more recent architectures such as the DEC Alpha [4]. The branch delay slot causes problems for a superscalar machine in many of the same ways that variable length instructions cause problems for CISC machines. The delay slot may lie on the next cache line. If this line causes a cache miss, the delay slot address and the target address must both be saved until the cache line for the delay slot is fetched. The delay slot must then be issued, and instruction fetching must resume at the target of the branch. Even worse problems occur when the branch and the delay slot span virtual memory page boundaries.

3 Floating Point Unit Design

3.1 Overview

Limited integration levels in GaAs constrain the amount of functionality that can be placed on a single chip. One obvious partitioning, shown in Figure 2, includes separate FPU, IPU, and data cache chips. Floating point operations tend to have longer latencies than integer instructions, which are made worse when interchip communication is necessary. To support double precision numbers, floating point datapaths are wider than their integer counterparts. The time needed to drive control signals across these wide datapaths places an upper limit on clock frequency and results in different maximum operating frequencies for the IPU and FPU. The resources of the IPU and FPU can be used more efficiently if the architecture decouples their operation; one way to achieve this is through the use of instruction and data queues.

The use of an instruction queue instead of a single entry instruction buffer allows the IPU to slip ahead of the FPU during program execution. Instead of stalling for FPU data dependencies, the IPU is able to transfer floating point instructions and continue execution. The IPU will be forced to stall due to the FPU only when the queue becomes full or when it has to wait for the result of an FPU operation. The use of an instruction queue does impact the precise handling of exceptions, since the IPU may be many instructions ahead by the time a floating point exception is detected. One solution involves having both a precise execution mode and a higher performance mode. For the former, instructions are not transferred to the FPU unless they are guaranteed never to cause an exception; examination of operand exponents and the exception flags can be used to predict whether an exception can occur.

The use of an instruction queue necessitates a corresponding load queue to store load data prior to its being written into the floating point register file. However, the depth of this queue does not need to be as large as that of the instruction queue. Similarly, a store queue is used to hold the results of floating point store and move-to-IPU instructions.

The FPU also includes a 32x64 entry register file, reorder buffer, scoreboard, and separate functional units for addition, multiplication, division and conversion. Up to two floating-point instructions from the queue can issue per cycle, to any two of the functional units. Two independent busses are available to write results from the various functional units to the reorder buffer. The add unit is pipelined with a latency of three cycles. The multiply and divide units are iterative and can produce a result in five and 19 cycles, respectively. The conversion unit has a latency of two cycles and performs conversions between single, double, and integer formats. Each functional unit supports the IEEE-754 specification for rounding modes and exceptions.

4 Architectural Evaluation

4.1 Benchmarks

Architectural performance was evaluated using the integer and floating point Spec92 benchmarks. Time constraints imposed by other phases of the design process limited the length of the benchmarks that could be run. The integer benchmarks used the "small test" input file, and the floating point benchmarks were limited to their first 90 million cycles. In all, about 176 million instructions were run from the integer suite, and about 810 million from the floating point suite. All benchmarks were compiled using GCC with no additional code rescheduling.

4.2 Architectural alternatives

Three machine models, referred to as the large, baseline and small models, were chosen to evaluate the effects of resource allocation and memory latency. The hardware resources allocated to each are listed in Table 1. Each of the three models is evaluated with one or two execution pipes, and with secondary memory system average latencies of 17 and 35 cycles, corresponding to medium and fast clock rates. Thus, results for 12 configurations are reported.

The register bit equivalent (RBE) model of Mulder [11] was used to evaluate the implementation cost in chip area of the different configurations. The RBE model provides a normalized measure for the area cost of different microarchitectural components. For our machines the RBE is the area required to implement a 1-bit static latch. For GaAs DCFL, one static latch requires about 16

Model	I Cache Size	D Cache Size	Write Cache Size	Reorder Buffer Entries	Prefetch Buffers	MSHR Entries
Small	1 Kbyte	16 Kbyte	2 lines	2	2	1
Baseline	2 Kbyte	32 Kbyte	4 lines	6	4	2
Large	4 Kbyte	64 Kbyte	8 lines	8	8	4

Table 1: The Three Machine Models and their Associated Resources

IPU Element	Cost in RBE	FPU Element	Cost in RBE
1 Kbyte Cache Block	8,000	1 Data Resource Block (RF, SB)	4,000
2 Kbyte Cache Block	12,000	1 Queue Entry (instruction/data)	50/80
4 Kbyte Cache Block	20,000	1 Add Unit (1-5 cycles)	5,000-1,250
1 Write Cache Line	320	1 Multiply Unit (1-5 cycles)	6,875-2,500
1 Prefetch Line	320	1 Divide Unit (10-30 cycles)	2,500-625
1 Reorder Buffer Entry	200	1 Conversion Unit (1-5 cycles)	2,500-1,250
1 MSHR Entry	50		
1 Integer Execution Pipeline	8192		

Table 2: Processor Element Cost in RBE Units

transistors and corresponds to an area of about 3600 square microns. Static RAM elements are denser, with a single bit requiring an area equal to 0.5 RBE. However, RAM blocks have additional overhead devoted to decoding and sensing. This overhead is a significant fraction of total area for small RAM blocks. The cost of each microarchitectural element is listed in Table 2. These figures are based on layout obtained during chip design. In addition to memory elements, the cost of an execution pipeline is also estimated. An important assumption in this analysis is that the cost of interconnecting microarchitectural elements is an overhead. that scales with the sum of their areas.

The cost of data cache is not included in our analysis because limits on die size placed the data cache in chips separate from the IPU and FPU. A more comprehensive cost analysis would optimize a system consisting of the IPU, FPU and data cache chips on the MCM in a way analogous to the intra-chip design problems considered in this paper. Finally, it is important to note that increases in area will slow the clock cycle. This issue is addressed in [12].

5 Study Results

The size and quantity of both on-chip memory structures and execution pipelines were varied to study the trade-offs between memory structures and execution logic. The Cycles Per Instruction (CPI) for each benchmark were measured, and can be compared to the implementation cost to evaluate the effectiveness of various memory and pipeline structures. The base model instruction cache hit rate is 96.5% and data cache hit rate is 95.4%; these numbers agree with those published in [5].

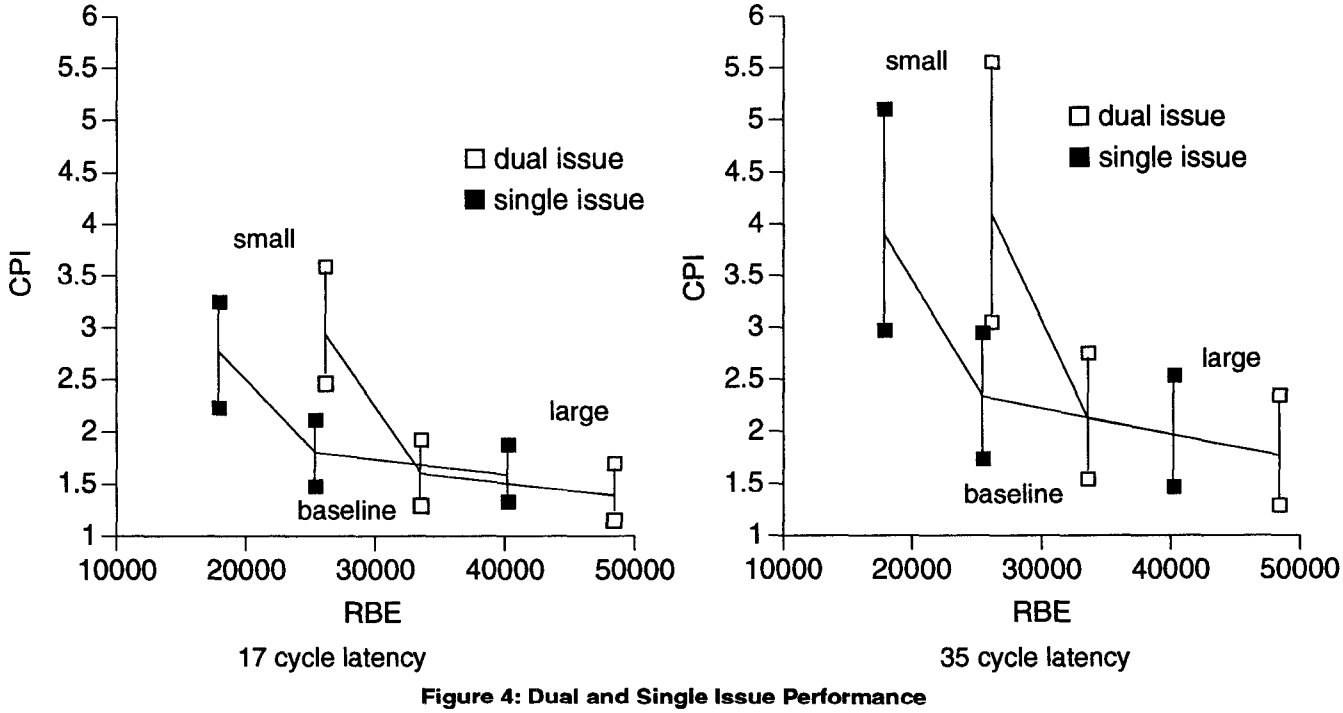


Figure 4: Dual and Single Issue Performance

Performance vs. cost in RBEs is shown for systems having17 cycle latency and 35 cycle latencies. In each graph, the costs of 6 systems are shown. Three are single issue and three are dual issue. The vertical lines with square caps show the range of CPI for each configuration. The upper cap is the maximum CPI, the lower cap is the minimum CPI, and the line is the average CPI.

model	espresso	li	eqntott	compress	sc	gcc
small	56.26	50.79	94.89	50.73	45.97	58.89
baseline	61.02	45.33	88.34	53.13	49.01	57.75
large	57.33	40.56	51.41	53.75	48.05	56.10

Table 3: Integer I Prefetch Hit Rate Percentage

model	espresso	li	eqntott	compress	sc	gcc
small	7.06	9.80	2.85	8.33	22.34	7.84
baseline	8.95	14.41	2.29	13.13	27.42	8.63
large	7.82	14.57	1.53	17.16	30.00	10.02

Table 4: Integer D Prefetch Hit Rate Percentage

5.1 Model Performance Evaluation

We first examine the performance of each of the three models. Figure 4 shows the results for dual and single issue for the 3 baseline models with average secondary latencies of 17 and 35 cycles. For each graph the cost difference between the two curves is caused by the addition of a second execution pipeline, which has a cost of 8192 RBEs.

With 17-cycle latencies, the addition of the second pipe results in higher performance with the baseline and large models. The single issue base model has a similar cost and much better performance than the dual issue small model.The large model with dual issue achieves the best performance by 12.7%, but with a hardware cost increase of 20.4%. With 35-cycle secondary latencies, the dual and single issue machines have similar cost-performance curves; the dual issue model achieves a 9.9% CPI improvement over the single issue model.

5.2 Instruction and Data Prefetching

Table 3 and Table 4 show the prefetch buffer hit rates for the instruction and data streams. A prefetch hit occurs if the data misses in the primary cache and hits in one of the prefetch stream buffers. Average hit rates for the integer SPEC benchmarks are 58% for the instruction stream and about 12% for the data stream. Floating-point prefetch data hit rates average 14.4% for the base

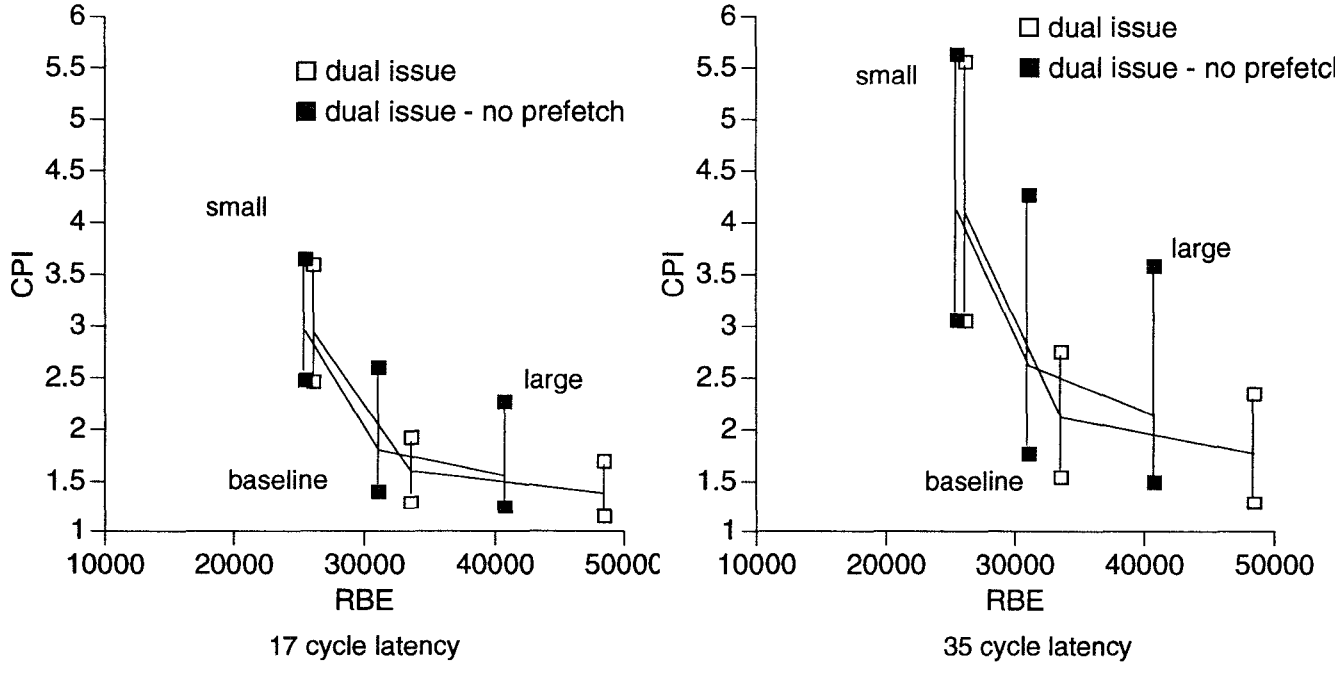


Figure 5: Effects of Prefetch Removal

Performance in CPI vs. cost in RBEs is shown for systems having 17 and 35 cycle latencies. In each graph, the costs of 6 systems are shown. All are dual issue. The vertical lines with square caps show the range of CPI for each configuration. Filled caps indicate no prefetch and hollow caps prefetch. The average CPIs for the three systems with prefetch are connected by a line. The average CPI for the three without prefetch are also connected by a line.

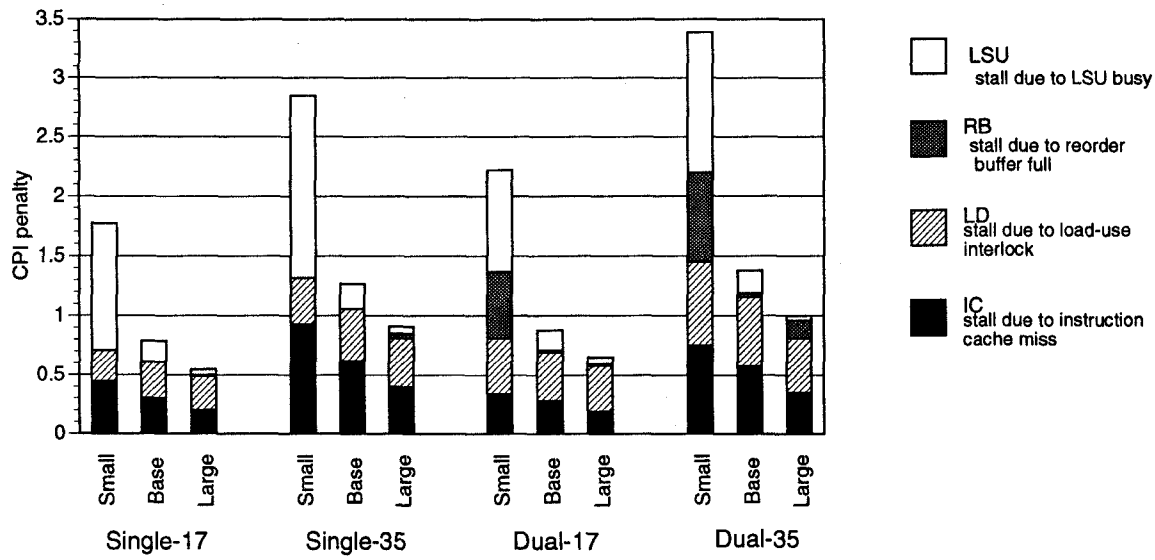


Figure 6: Break Down of Stall Penalties

model, but the peak hit rate on some of the benchmarks is as high as 60%.

Figure 5 shows the effects of removing the prefetch buffers from each of the dual issue models. Prefetching is of little benefit in the small model for a number of reasons. There are only two buffers in the small model, which leads to thrashing between instruction and data references. In addition, the cache miss rates are high, leaving little spare bandwidth for prefetching.

Prefetching is much more successful in the base model, resulting in an average improvement of 11% in the case of 17-cycle latency. Prefetching helps even more in the case of 35-cycle latency, improving performance for the base model by 19%. The large model also sees a significant improvement: 11% for the 17-cycle latency and 17% for the 35-cycle latency. In addition to the average performance, prefetching substantially improves worst case performance, reducing the CPI for the short latency cases in our study by 25% and in the long latency case by 35%. The cost of adding prefetch buffers is fairly small; for the baseline configuration, the prefetch buffers are only 20% of the instruction cache size.

5.3 Execution Unit Stall Distribution

The IPU has 4 major stall conditions: instruction cache stall while waiting for instructions, load stall when the result of a load instruction is referenced before it has been returned by the LSU, Reorder Buffer full stall, and LSU stall when the LSU is full or is using the data busses to fill the cache. Figure 6 shows the CPI penalty contributed by each of these stall conditions.

Except for the small model case, most stalls are caused by instruction misses and data cache accesses. In the small model, most cycles are spent waiting for data from the LSU. In the base and large models, performance is not very sensitive to the size of the IPU reorder buffer because the processor often stalls when it references the result of a load instruction before the reorder buffer fills. In the large model, the small percentage of LSU-Busy stalls indicates that most of the data hits in the cache, the large percentage of Load stalls is caused by the three-cycle latency of the pipelined data cache.

5.4 Degree of Non-blocking Data Cache

One of the largest contributors to high performance is the ability to support multiple outstanding load instructions simultaneously. The number of outstanding data cache misses is set by the number of MSHR registers. Figure 7 shows the effect of changing the number of possible outstanding memory references for the different cache models. The small model shows a dramatic performance increase when additional MSHR registers are allowed, and the base model shows a small improvement when adding an MSHR. The large model show some performance decrease when the number of MSHR registers is reduced. from four. All models get highest performance when 4 MSHR entries are available.

5.5 Write Cache Size

Table 5 gives hit-rate statistics for the on-chip write cache of the different models. The write cache size varies from two lines in the small model to eight lines in the large model. The hit rate includes both load and store data accesses. The write cache has a surprisingly high hit rate, even for the small model.

This high hit rate helps to reduce significantly the write traffic off chip. The number of store transactions is reduced to 44% of the number of store instructions for the small case, to 30% for the base model and to only 22% for the large model. This is more than a two-fold reduction in write traffic for the small model and nearly a five-fold reduction for the large model.

5.6 Analysis of Integer Data and Recommendations

Figure 8 presents all of the simulation data points for the latency-17 data sets for the espresso benchmark. The other benchmarks and latencies give similar performance curves.This graph demonstrates the trade-offs between cost and performance for our machine model. There are three points of note: first, the points labeled A all have only a single MSHR and lie well above other

model	espresso	li	eqntott	compress	sc	gcc
small	29.22	35.97	31.84	37.82	46.61	42.82
baseline	37.17	49.17	48.34	46.29	52.53	54.93
large	43.73	60.52	60.03	59.87	59.56	63.17

Table 5: Integer Write Cache Hit Rate Percentage

configurations of equivalent cases, showing the negative impact of blocking caches. The points labeled B correspond to the large model. A performance plateau exists here that yields little gain in performance even if significant additional cost is expended. The benefits of prefetching are demonstrated by comparing points C and D. They differ only in that D adds prefetching. In general, the small model shows a large performance increase with additional resources, the base model shows a small increase, and the large model shows little increase.

The previous data suggest some obvious changes to our baseline machines. All models benefit from increased MSHR entries, because the cost is low, adding MSHR entries provides price-performance benefit.

A write cache larger than in the baseline model has little performance benefit. In addition, the prefetch performance of the large model is no better than that of the baseline model. The point labeled E on the graph suggest reducing the resources for the large model to a 4 Kbyte Icache, a 4-entry write cache, a 6-entry reorder buffer and 4 MSHR entries. Dual issue is reasonable only if supported by sufficient memory resources and becomes less attractive as memory latency increases.

5.7 Allocation of Floating Point Resources

As with the IPU, the FPU has a large design space to consider when allocating resources. The options include the number of entries for the instruction, load, and store queues, the size and organization of the reorder buffer and register file, and the complexity of the floating point functional units. More sophisticated algorithms can be used by functional units to reduce latencies, and pipelining can be used to reduce critical paths, all at the expense of adding more transistors. One must determine how to use the available resources most effectively. Dual-issue of floating point instructions adds several issue constraints to that for integer instructions. The usefulness of this feature is dependent on the amount of parallelism in the floating-point workload.

5.8 Floating Point Issue Policy

Two issue policies were investigated: 1) in-order issue with in-order completion of instructions, and 2) in-order issue with out-of order completion of instructions. The former approach does not allow multiple issue of instructions or instructions active in multiple functional units. Single issue and dual issue are both possible for the latter policy. Floating point dual issue is constrained by

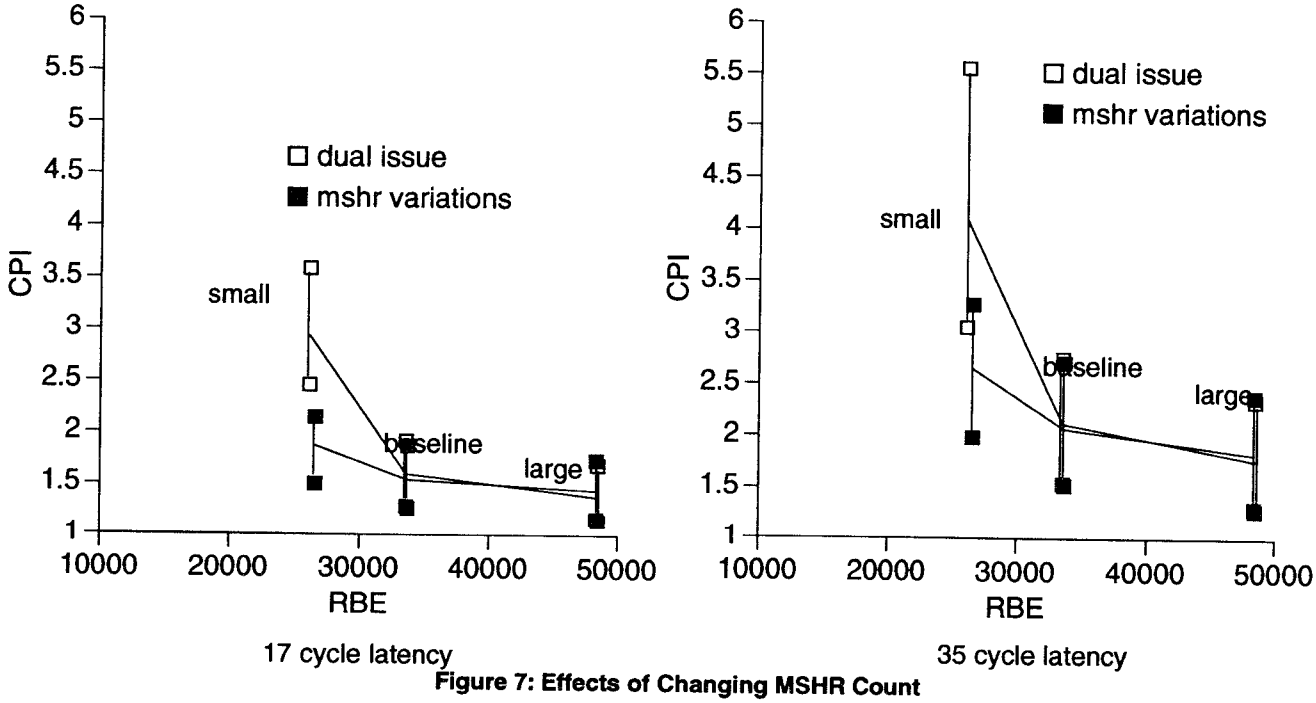


Figure 7: Effects of Changing MSHR Count

The line with open cap bars corresponds to the three standard dual issue configurations. The "mshr variations" line was obtained by increasing the number of MSHRs for the small and baseline models from one and two respectively, to two and four. The large model was modified by reducing the number of MSHRs from four to two.

data dependencies, reorder buffer stalls, busy functional units, result bus conflicts, and by fewer than two entries active in the instruction queue.

Benchmark	In Order Issue and Completion	Single Issue	Dual Issue
alvinn	2.113	2.1110	2.107
doduc	1.957	1.7821	1.671
ear	1.299	1.1549	1.022
hydro2d	1.298	1.1234	0.999
mdljdp2	1.344	1.1357	0.948
nasa7	1.702	1.2939	0.957
ora	1.906	1.7800	1.701
spice2g6	1.219	1.2037	1.203
su2cor	1.973	1.7057	1.557
Average	1.577	1.4012	1.248

Table 6: CPI Figures for Three FPU Issue Policies

Dual issue of floating point instructions incurs some hardware cost, including two additional read ports on the register file and reorder buffer. A sense amplifier-based register file provides additional read ports without a large penalty in speed or area. The instruction queue needs an additional read port to allow access to the instruction immediately below the head of the queue. Additional source operand busses are needed, as well as more complex control for instruction decoding and issue. Much of the instruction decoding is done in advance of the actual issue stage of the FPU.

Results for the various policies are summarized in Table 6. A 12% improvement in performance for single issue and a 21% gain for dual issue are realized, compared to the in-order issue in-order completion policy.

5.9 Floating Point Memory Resources

Figure 9 (a), (b), and (c) show performance metrics for a single instruction issue policy with various sizes of instruction queue, load data queue, and reorder buffer, respectively. The instruction and load queues reflect sizes of 1 to 5 entries, while the reorder buffer varies in size from 3 to 11 entries. For the instruction queue, the performance improvement becomes quite flat out at 3 entries. Dual issue places a greater demand on the instruction queue; simulations (not shown) suggest that design with five entries is optimal. For load instructions, the results indicate that two entries are needed. This result is not surprising, since most of the benchmark applications utilize double precision numbers and two 32-bit loads are needed per operand. The FPU being implemented also supports double-word loads and stores, which should improve performance, since on average 15% of floating point instructions executed in the SPEC benchmarks are loads. The sensitivity to reorder buffer size decreases for more than six entries, suggesting that this is the average number of floating point instructions active in the FPU pipeline.

5.10 Floating Point Functional Unit Latencies

Numerous floating point addition optimizations could be used to reduce latency, all at the expense of transistor count and implementation complexity. These include parallel paths for alignment and normalization, fast generation of the sticky bit, and leading one prediction for normalization. A two cycle add unit incorporating these approaches occupies the most area, primarily due to the use of two 53-bit mantissa adders. Compared to this implementation, a three cycle unit can result in an area reduction of 20%. Further reduction of resources devoted to addition will result in four to five cycle latencies.

Conventional approaches to multiplication involve a partial product array (3-2 or 4-2 carry-save adders) followed by a carry-propagate mantissa adder. Booth recoding of the input operands

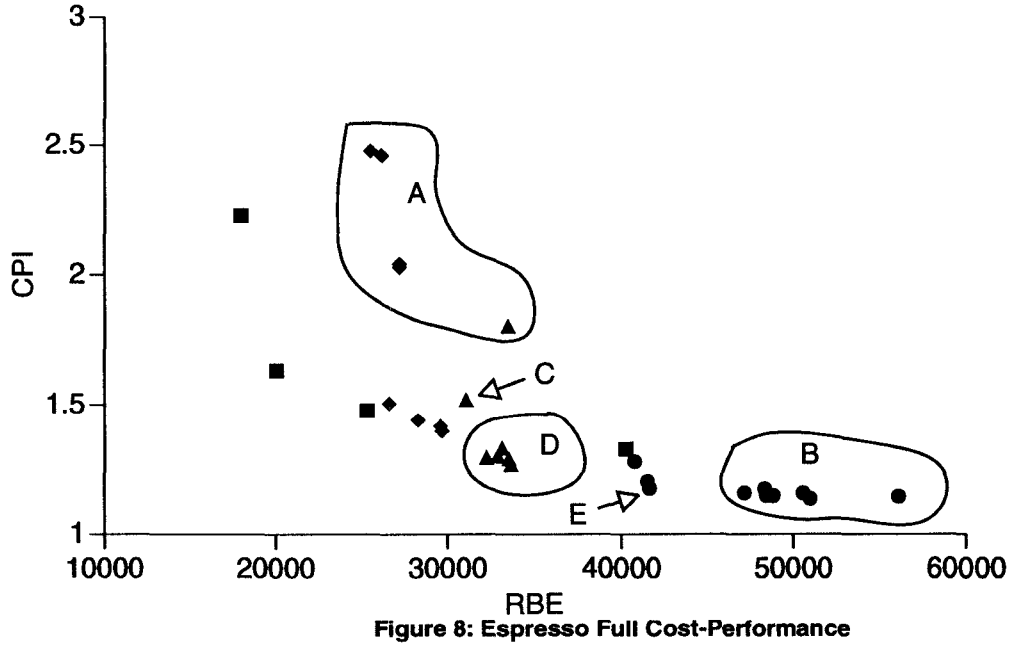
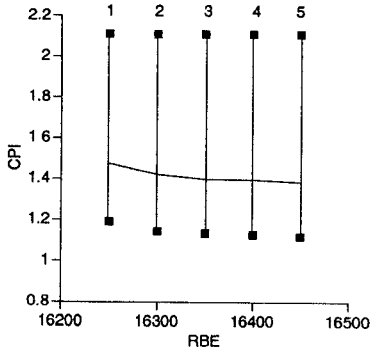
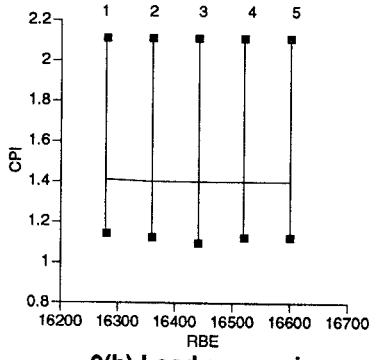


Figure 8: Espresso Full Cost-Performance

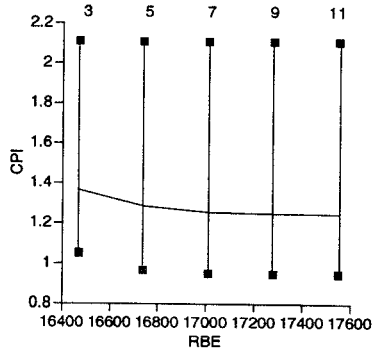
Four classes of systems are shown. The four squares represent single issue systems of various cache size. The eight diamonds represent dual issue systems with a 1K instruction cache and a variety of sizes of other memory elements. Likewise the triangles are 2K systems and the circles are 4k systems



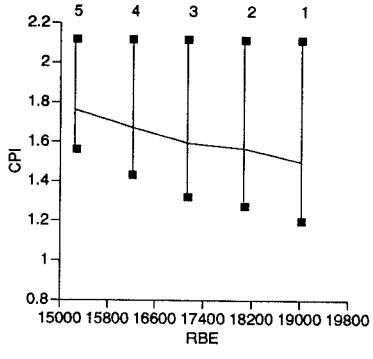
9(a) Instruction queue size



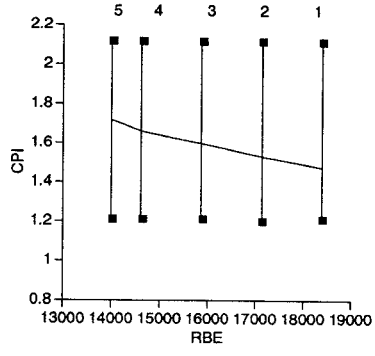
9(b) Load queue size



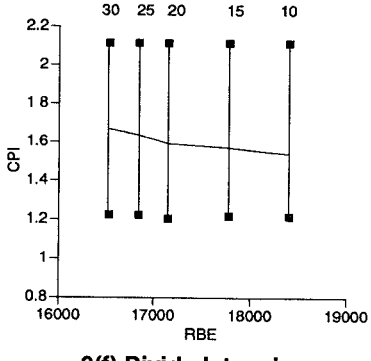
9(c) Reorder buffer size



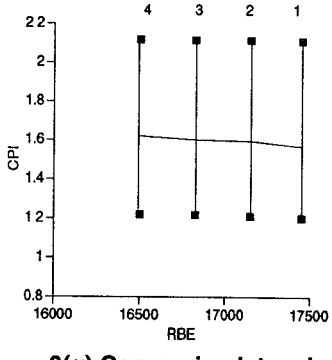
9(d) Add latencies



9(e) Multiply latencies



9(f) Divide latencies



9(g) Conversion latencies

Figure 9: FPU Cost Studies

The numbers above the vertical bars indicate queue sizes for (a), (b), and (c); and latencies in clock cycles for (d), (e), (f), and (g).

can be used to reduce the number of levels in the array by one, at the expense of adding recoding multiplexors. Analysis of these two approaches in GaAs DCFL showed that the area savings of Booth-recoding are small when compared to the increase in the complexity of the design. Increases in capacitance along critical paths of a Booth recoded multiplier tend to offset the reduction in logic depth. Another alternative involves the iterative use of a smaller array. This approach reduces the size of the multiplier considerably, however five cycles are needed to produce a result. Furthermore, the multiplier is not pipelined, forcing subsequent multiply instructions to wait for the current instruction to complete.

Non-restoring division algorithms can be enhanced by representing intermediate results in a higher radix redundant form. SRT-2, SRT-4, and SRT-8 approaches return one, two, and three bits per cycle, respectively. Latencies vary in the range of 20 to more than 50 cycles. Additional techniques can be employed to perform on-the-fly conversion from redundant form to sign-magnitude form and on-the-fly rounding of the result. A square root instruction can be mapped easily to the same hardware used for division, with little additional cost.

Figure 9 (d), (e), (f), and (g) show data for various latencies in the four execution units. Addition and multiplication both show a 17% change in CPI for latencies ranging from one to five cycles. For addition, latencies of two, three, and four correspond to interesting realistic designs. An add unit with a latency of two results in only a 2% improvement in performance over one taking three cycles, while a latency of 4 is 5% worse than a latency of 3. Similarly, each additional cycle of latency in the multiplier reduces performance (as measured by CPI) by 4%. Performance is relatively insensitive to latency; in the division unit, over a latency range of 10 to 30 cycles latency performance changes by 8%.

Conversion instructions occur very infrequently and have little impact on overall performance.

The same simulations were repeated for non-pipelined addition and multiplication units. Interestingly, the degradation in performance is less than 5%. The latches used for pipelining these two units account for approximately 25% of the area of each unit. Not pipelining these units can result in significant savings in area and power dissipation.

5.11 Floating Point Recommendations

The FPU architecture being implemented as a result of these studies has the following features:

- dual issue,
- 5 entry instruction queue,
- 2 entry load data queue,
- 6 entry reorder buffer,
- 3 cycle add unit,
- 5 cycle multiply unit,
- 19 cycle division unit, and
- 2 result busses.

6 Conclusion

The performance contributions of various architectural features were evaluated for a high clock rate superscalar machine. A register bit equivalence model for the cache structures in the processor was utilized, and the performance sensitivity to the cache sizes was evaluated. We found that without special scheduling in the compiler, large memory latencies reduce the benefit of superscalar-issue, and that when memory latencies are long, a less expensive machine can provide performance similar to a more complicated superscalar machine. At shorter memory latencies, both the baseline and large models showed an improvement in performance for superscalar issue.

Nonblocking data caches were shown to be important; additional Miss Status Holding Registers can greatly increase performance for machines having a small transistor budget. With a single MSHR, the pipeline blocks at each cache access, and LSU stalls contribute the majority of stall cycles. Prefetching was also shown to be beneficial.

Increasing the size of the reorder buffer, prefetch buffers and write cache in the large model provided little improvement in performance. A machine deviating from the baseline only in instruction cache size (4 K-byte) achieved nearly the same performance as the large model at a much lower cost. In the large machines, most stalls were caused by the three-cycle latency of the pipelined data cache. Better compiler scheduling could possibly remove some of this penalty.

Floating-point performance can be enhanced by the use of inexpensive decoupling queues. The latencies of floating point functional units have a modest impact on performance, but a larger impact on area. These functional units can be further reduced in size by removing pipeline latches. Dual issue achieves a reasonable gain in performance for the additional resources required.

Acknowledgments The authors would like to thank the other members of Aurora group: Dave Putti, Dave Kibler, Patrick Sherhart, Ajay Chandna, Tim Stanley, and Dave Nagle.

7 References

[1] K. Diefendorff and M. Allen, "Organization of the Motorola 88110 Superscalar RISC Microprocessor," *IEEE MICRO*, April, 1992, pp. 40–63.

[2] K. Diefendorff, R. Oehler, R. Hochsprung, "Evolution of the PowerPC Architecture," *IEEE MICRO*, April 1994, pp 34-49.

[3] D. Ditzel and H. McLellan, "Branch Folding in the CRISP Microprocessor: Reducing Branch Delay to Zero," *14th Annual Symposium on Computer Architecture*, 1987, pp. 2–16.

[4] D. Dobberpuhl et al., "A 200-MHz 64-b Dual-Issue CMOS Microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 27 no. 11, pp. 1555-1567, Nov. 1992

[5] J. Gee, M. Hill, D. Pnevmatikatos et al., "Cache Performance of the SPEC 92 Benchmark Suite," *IEEE MICRO*, August 1993 pp 17-27.

[6] P. Hsu, "Designing the TFP Microprocessor," *IEEE MICRO*, April 1994, pp 23-33.

[7] N. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," *17th International Symposium on Computer Architecture*, 1990, pp. 364–373.

[8] N. Jouppi, *Cache Write Policies and Performance*, WRL Research Report 91/12 DEC Western Research Laboratory

[9] D. Kroft, "Lockup-free Instruction Fetch/Prefetch Cache Organization," *8th International Symposium on Computer Architecture*, 1981, pp. 81–87.

[10] N. Kushiyama et al., "A 500-Megabyte/s Data-Rate 4.5M DRAM," *IEEE Journal of Solid-State Circuits*, vol. 28 no. 4, pp. 490-498, Apr. 1993

[11] J. Mulder et al., "An area model for on-chip memories and its application," *IEEE Journal of Solid-State Circuits*, vol. 26, no. 2, 1991, pp. 98–106.

[12] O.A. Olukotun, T.N. Mudge, and R.B. Brown, "Performance optimization of pipelined primary caches," *Proc. of the 19th Ann. Int. Symposium on Computer Architecture*, May 1992, pp. 181-190.

[13] J. Smith and A. Pleszkun, "Implementing precise interrupts in pipelined processors," *IEEE Transactions on Computers, C-37*, no. 5, May, 1988, pp. 562–573.

[14] T. J. Stanley, M. D. Upton, P. J. Sherhart, T. N. Mudge, R. B. Brown, "A microarchitectural performance evaluation of a 3.2 GB/s microprocessor bus," *MICRO-26, The Ann. ACM/IEEE Int. Symposium on Microarchitecture*, Austin, TX, Dec. 1993, pp. 31-40.

[15] J. Thornton, *The Design of a Computer, the Control Data 6600*, Scott, Foresman and Company, 1970.

[16]M. Upton, T. Huff, P. Sherhart, P. Barker, R. McVay, T. Stanley, R. Brown, R. Lomax, T. Mudge, and K. Sakallah, "A 160,000 transistor GaAs microprocessor," *Int. Solid-State Circuits Conf. Digest of Technical Papers*, vol. 36, Feb. 1993, pp. 92-93.