PERFORMANCE ANALYSIS OF THE ALPHA 21264-BASED COMPAQ ES40 SYSTEM

Zarka Cvetanovic and R.E. Kessler
Compaq Computer Corporation

Abstract

This paper evaluates performance characteristics of the Compaq ES40 shared memory multiprocessor. The ES40 system contains up to four Alpha 21264 CPU's together with a high-performance memory system. We qualitatively describe architectural features included in the 21264 microprocessor and the surrounding system chipset. We further quantitatively show the performance effects of these features using benchmark results and profiling data collected from industry-standard commercial and technical workloads. The profile data includes basic performance information – such as instructions per cycle, branch mispredicts, and cache misses – as well as other data that specifically characterizes the 21264. Wherever possible, we compare and contrast the ES40 to the AlphaServer 4100 – a previous-generation Alpha system containing four Alpha 21164 microprocessors – to highlight the architectural advances in the ES40. We find that the Compaq ES40 often provides 2 to 3 times the performance of the AlphaServer 4100 at similar clock frequencies. We also find that the ES40 memory system has about five times the memory bandwidth of the 4100. These performance improvements come from numerous microprocessor and platform enhancements, including out-of-order execution, branch prediction, functional units, and the memory system.

1. INTRODUCTION

The Compaq ES40 is a shared memory multiprocessor containing up to four third-generation Alpha 21264 microprocessors [1][2]. Figures 1 and 2 illustrate the ES40 performance relative to other vendor systems on the SPEC95 benchmarks. Figure 1 compares single processor SPEC95 performance using published results as of March 2000. Figure 2 shows a similar comparison in the multiprocessor SPECrate_fp95. (We use SPECfp95 workloads rather than SPECint95 for the multiprocessor comparison since several of the SPECfp95 benchmarks

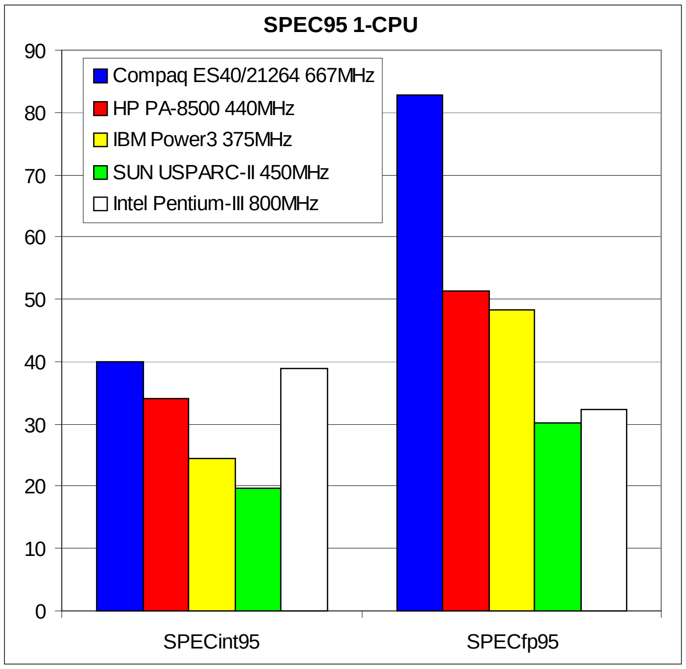

Figure 1 - SPEC95 Comparison

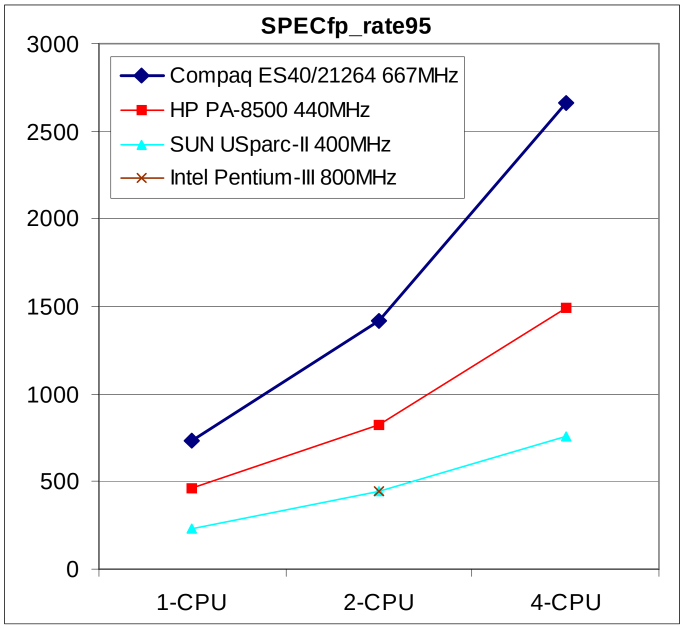

Figure 2 - SPECfp_rate95 Comparison

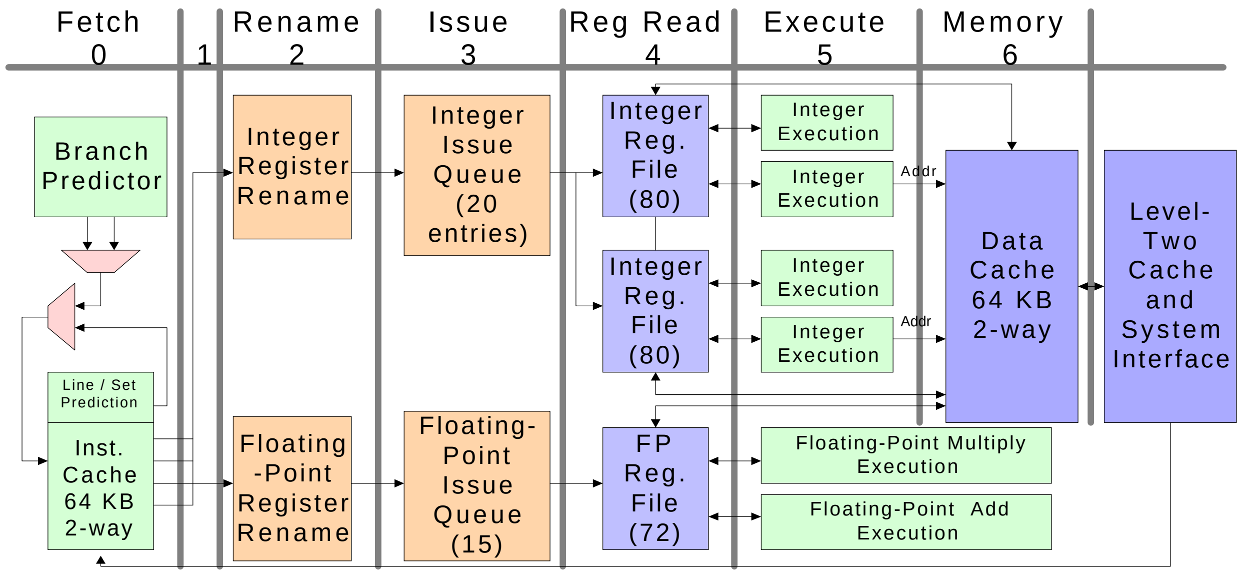

Figure 3 – The Basic Alpha 21264 Instruction Pipeline

stress the memory system, while all SPECint95 benchmarks fit well in 4MB caches and thus are not a good indicator of memory system performance.)

The single processor results show the ES40 between 3% to a factor of three faster than other leading-vendor systems. The multiprocessor results indicate that ES40 scales well in memory-bandwidth intensive workloads and has between 60% and a factor of 3.5 advantage over other vendor platforms. We analyze key performance characteristics of the ES40 in this paper to expose the techniques that were needed to reach these performance levels.

The ES40 contains many architectural advances – both in the microprocessor and in the surrounding memory system - that contribute to its performance. The 21264 includes many techniques to expose instruction-level parallelism: numerous prediction techniques, out-of-order execution, and the hardware to manage many in-flight instructions. The surrounding memory system services many parallel cache misses at high-bandwidth with low latency. We describe and analyze these architectural advances and present key results and profiling data to clarify the analysis. We evaluate the benefits of these architectural techniques. We quantify the performance improvements with performance/profiling data gathered from hardware measurements. We use a previous-generation Alpha four-processor system (the AlphaServer 4100) as a comparison point when possible to highlight the architectural advances made from the 21164 Alpha generation [3][8] to the 21264 [1][2][13].

We include on-line transaction processing (pseudo TPC-C workload [9]) and technical/scientific workloads (SPEC95 [10]) in our results. The transaction-processing workload exercises processor power, but also memory interface, as well as I/O. The SPEC95 benchmarks exercise processor, memory hierarchy, and compiler performance.
We use profile data from the ProfileMe and DCPI tools based on built-in non-intrusive CPU hardware monitors [4][5]. These monitors collect various events including stalls, retired and non-retired instructions, branch mispredicts, replay traps, TB misses, cache misses, etc. The monitors are useful tools for analyzing system behavior with various workloads. We use the data from these tools extensively in our architectural analysis. The detailed program profiles (at image, procedure, or instruction level) generated with these tools can also be used by compiler and application developers for code optimization.

The remainder of this paper is organized as follows: Section 2 describes the architecture of the 21264 microprocessor and the ES40 system. Section 3 describes the fetch and execution improvements in the 21264. Section 4 describes the memory system improvements. Section 5 concludes.

2. SYSTEM OVERVIEW

Figure 3 shows the instruction execution pipeline in the 21264 microprocessor. The 21264 is a superscalar microprocessor that can fetch up to four instructions and execute up to six instructions in any given cycle. Like the previous-generation Alpha processors, the 21064 and

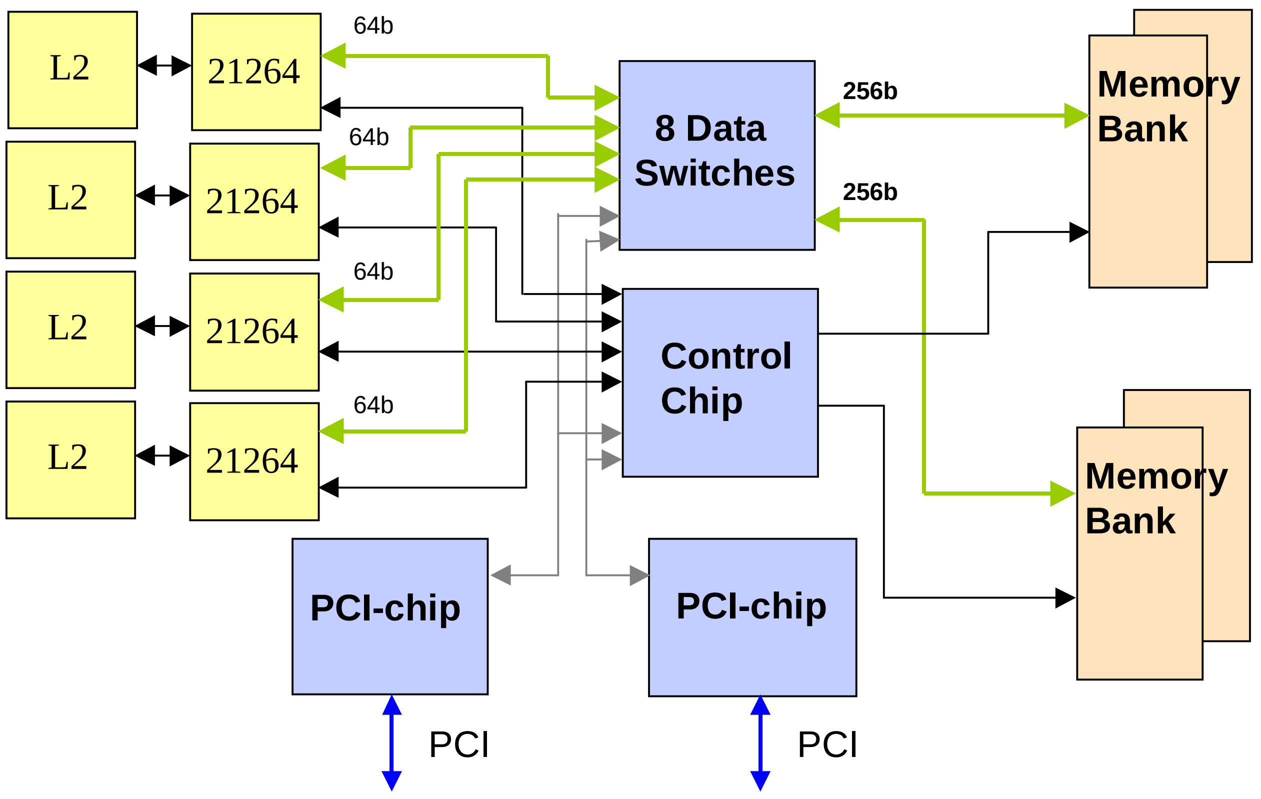

Figure 4 – ES40 Block Diagram

21164, high clock speed is a large contributor to the industry-leading performance of the 21264. But the 21264 also adds out-of-order and speculative execution to expose more instruction parallelism – ultimately improving (retired) instructions per cycle. The 21264 manages up to 80 in-flight instructions – a large number compared to some currently available processors.

Section 3 analyzes many of the architectural features in the early stages of the instruction pipeline. The first stage (stage 0) fetches four instructions per cycle. The large (64 KB) and associative instruction cache, line-and-way prediction, and branch prediction all enable high-bandwidth instruction fetch. Stage 1 is largely consumed transferring instructions from fetch to map. It slots instructions to the integer or floating-point pipelines.

Stages 2 and 3 – the map and queue stages - implement the out-of-order execution in the 21264. Here the user-visible (virtual) registers are remapped into internally-visible (physical) registers. This process removes all except for read-after-write register dependencies. It also allows for more internal registers than are programmer visible (e.g. there are 31 visible floating-point registers and 72 internally). The queues issue up to six instructions. Instructions execute out-of-order from the issue queue, prioritized from oldest to newest. Once they issue, they proceed through pipeline stages 4, 5, and beyond.

Section 4 describes the memory system of the ES40. Pipe stages 5, 6, and beyond in Figure 3 depict some features of the 21264's internal memory system. The queue can issue up to two memory operations per cycle. These two operations simultaneously access the large (64 KB) and associative level-one data cache. References that miss in this cache access the off-chip level-two cache (of 8 MB, direct-mapped). References that miss in the level-two (L2) cache are serviced by the surrounding off-chip memory system.

Figure 4 shows a block diagram of a 4-processor ES40. The control chip manages the system and enforces write-invalidate cache coherence. The 21264's pass the L2 miss requests to the control chip. The control chip then simultaneously forwards the request to read the DRAM and broadcasts the address of the request (i.e. probes) to the other 21264's. The other 21264's check for necessary coherence violations and respond to the probe. The control chip examines the probe responses and responds to the requesting 21264 with data from either the DRAM memory or another 21264, as appropriate. Note that the control chip does not keep a copy of the L2 tags from the four CPU's to determine coherence actions. It shares the 21264's L2 cache tags via the probing mechanism. This coherence implementation is lower cost than an implementation that duplicates the L2 tags. It is also lower latency for requests

		ES40	4100	Ratio
MHz		667	600	1.11
SPECint95	**1-CPU**	40.0	18.8	2.13
SPECfp95	**1-CPU**	82.7	29.2	2.83
SPECfp95	**4-CPU**	147	51.4	2.86
SPECint_rate95	**4-CPU**	1390	657	2.12
SPECfp_rate95	**4-CPU**	2686	858	3.13
Linpack 1000x1000 (MFLOPS)	**4-CPU**	3721	2634	1.41
Transactions Per Minute		31K	16K	1.94

Table 1 - Benchmark Performance of the ES40 and 4100

that must be serviced by data from another 21264's cache. It performs well for the four-processor systems we analyze in this paper but can be limited by the available probe bandwidth (as discussed later).

The ES40 architecture has an address broadcasting coherence scheme that is similar to those commonly used in bus-based multiprocessors, but is a crossbar-based design. The interconnects in Figure 4 are unidirectional point-to-point. This allows for very high bandwidth transfers. The eight switch chips provide the crossbar functionality between the four 21264's, the DRAM memory, and the (PCI) IO interface.

We compare the 21264-based Compaq ES40 to the 21164-based AlphaServer 4100 at many points in this paper. The 4100 is the previous-generation four-processor Alpha server. It is bus-based, unlike the ES40. Table 1 compares the performance results of the two generations from some important benchmarks. The ES40 provides 2 to 3 times the performance of the 4100 server for most benchmarks. The clock speed improvement (667 MHz vs. 600 MHz) provides a maximum 11% potential gain; the majority of the gain comes from 21264 microprocessor and system platform improvements.

The technical-scientific workloads (SPECfp95) benefit the most from memory bandwidth and thus show the highest improvement on ES40 vs. 4100. The small integer workloads (SPECint95) benefit mostly from the processor architectural and compiler improvements. The Linpack benchmark does benefit from processor and Bcache improvements, but the number of floating-point pipelines limits its improvement to less than other benchmarks. The commercial workloads benefited from both system/memory improvements and processor enhancements. In addition to processor and platform-specific enhancements, re-compiling with compilers tuned for 21264 provides additional performance improvement: typically 5-15% (up to 30% in some applications). That is due to better code scheduling and enhanced prefetching.

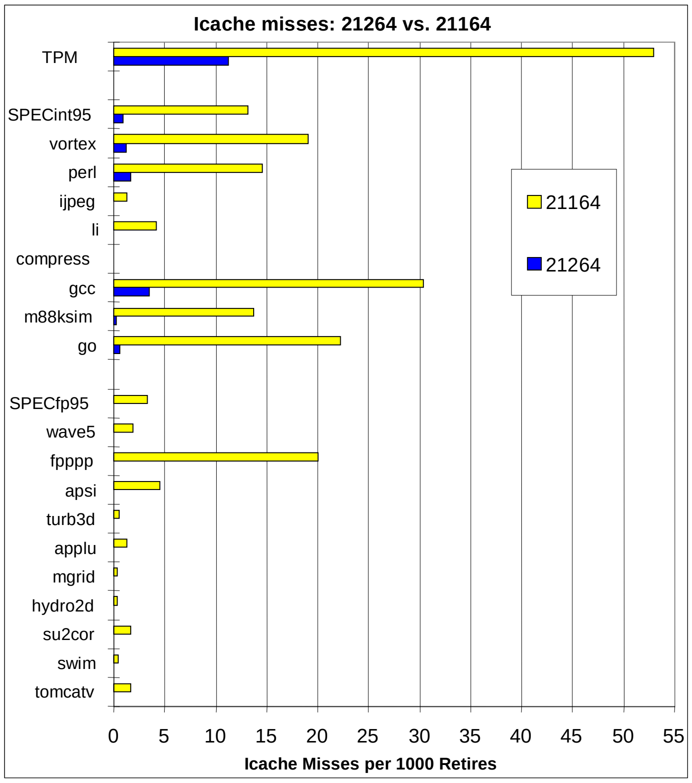

Figure 5 – Instruction Cache Miss Rate Comparison

More detailed descriptions of the 21264 and 21164, and systems based on those chips can be found in [1][2][3][6][7][8][13], and in the remainder of this paper.

3. FETCH AND EXECUTION IMPROVEMENTS IN THE 21264

3.2 INSTRUCTION FETCH

The level one instruction cache in the 21164 is 8 KB direct-mapped and the 21264's is 64 KB two-way associative. Figure 5 compares the instruction cache miss rates in the two cases. The 64KB Icache in 21264 greatly reduces the number of Istream misses compared to 21164. There are almost no remaining misses in SPECfp95 since their code sizes are so small. Increased instruction cache size and associativity is a very important contributor to improved fetch efficiency in the 21264.

The 21264 prefetches more aggressively when there is an instruction cache miss. It can prefetch up to 256 bytes sequentially ahead while the 21164 can only prefetch 96 bytes ahead. Note that the 21264 instruction cache miss counter does not consider a prefetched block to be a miss.

Instruction cache misses still remain high in the transaction processing workload on the 21264, indicating that there is room for further improvement in future systems. The

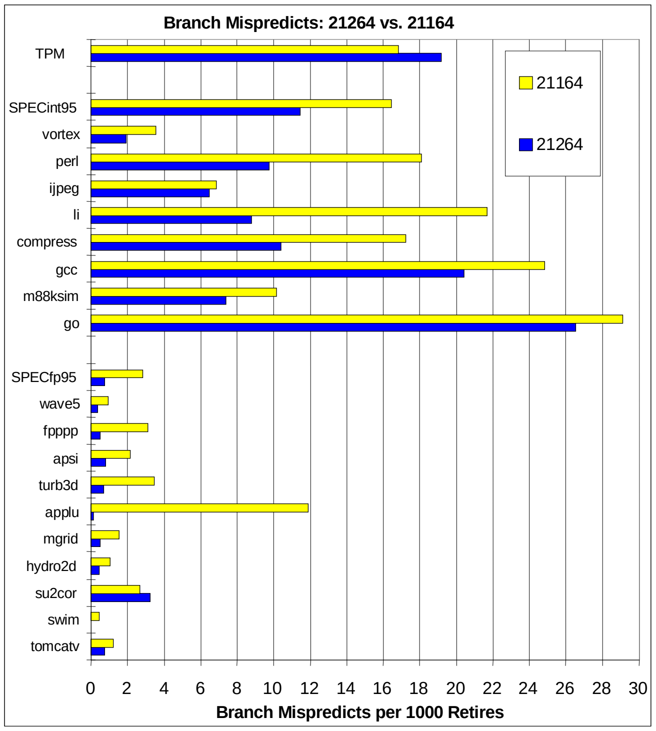

Figure 6 – Branch Mispredicts Comparison

instruction misses in the transaction processing workload are caused by frequent branches, process context switches, TB misses, subroutine calls, run time library, and system calls. Larger and faster caches can provide more benefits in future designs. Even more instruction prefetching may improve performance, but strict linear prefetching may only benefit the highly-sequential code segments.

The 21264 implements line and way prediction to improve instruction fetching. This predictor indicates the next four instructions to be fetched from the instruction cache. It can eliminate branch-taken stalls and improve fetch performance for other non-sequential control flows. It also can predict indirect subroutine jumps.

The 21264 branch predictor is more sophisticated than the 21164's simple two-bit scheme. Branch prediction is very important in speculative-execution architecture like the 21264 since it allows the out-of-order execution engine to speculatively issue the instructions down the predicted path. If the prediction is incorrect, all the instructions executed down the mispredicted path are wasted.

The 21264 branch predictor keeps both local and global history in a tournament branch predictor. The predictor learns complex patterns predicted by the (local) history of individual instructions, learns complex patterns predicted by the (global) history of multiple branch instructions, and chooses the best predictor of the two. This led to a significant reduction in mispredicted branches, as indicated in Figure 6. The branch mispredicts are almost completely eliminated in the floating-point codes, and are considerably reduced in most of the integer benchmarks.

The transaction processing workload is an interesting case since there are slightly *more* branch mispredicts in the 21264 than in the 21164. The instruction footprint of transaction processing workloads is large, with many branches, so it is challenging for the 1024-entry local history file and the 4096-entry global history file to contain history for all the relevant branches. The 21164's much simpler 2048-entry by two-bit predictor, indexed by program counter, provides slightly better branch prediction performance under the same overflow condition. This data hints that there is an opportunity for improved transaction processing branch prediction in future systems.

3.3 INSTRUCTION MAP AND QUEUEING

The map and queue pipeline stages provide much of the out-of-order and speculative execution functionality in the 21264 that was not present in the 21164. Register results are renamed to one of the 80 available integer registers and 72 available floating-point registers. An instruction is considered in-flight from the map stage until the instruction retires. The 21264 can allow up to 80 instructions to be in flight.

Figure 7 shows the average number of in-flight instructions on the 21264 for the applications we analyze. The results show that few applications average more than half the 80 maximum in-flight instructions. This indicates that the average number of in-flight instructions is less often limited by the *maximum* in-flight count. Queue overflows, physical register availability, and instruction cache misses more often stop instruction fetch. Branch (and other) mispredicts also limit the number of in-flight instructions because they remove mis-speculated instructions from the in-flight inventory.

The floating-point applications generally have more instructions in flight than the other applications. They tend to have more large loops with fewer control-flow changes and longer-latency instructions. They also have integer instructions, so both the integer and floating-point queues and registers can be utilized. Load and other integer instructions are siloed only in the integer queue, independent of the floating-point operation instructions, providing more buffering. All of these factors make it easier to keep more instructions in flight in the floating-point applications.

The integer and transaction applications have fewer instructions in flight. This is likely limited by branch mispredicts, instruction cache misses, and queue overflows.

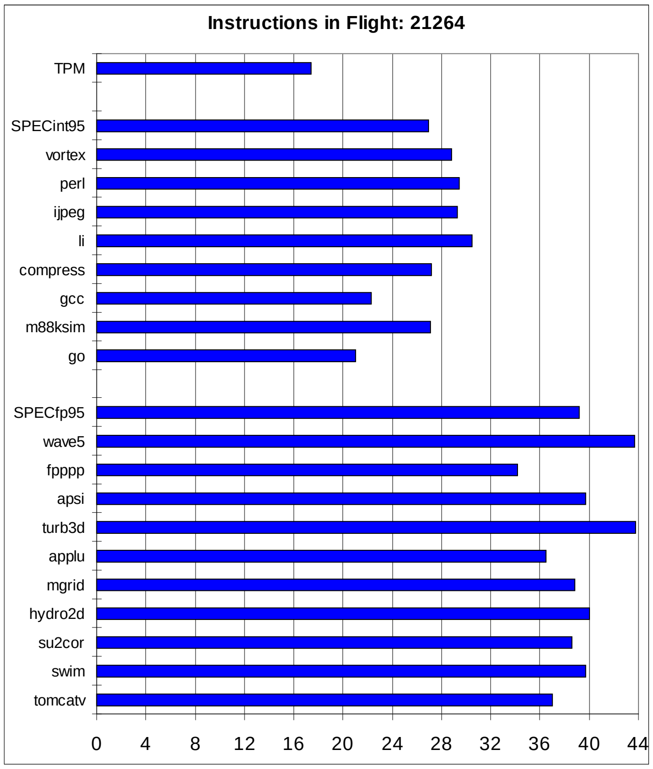

Figure 7 – Average Instructions in Flight

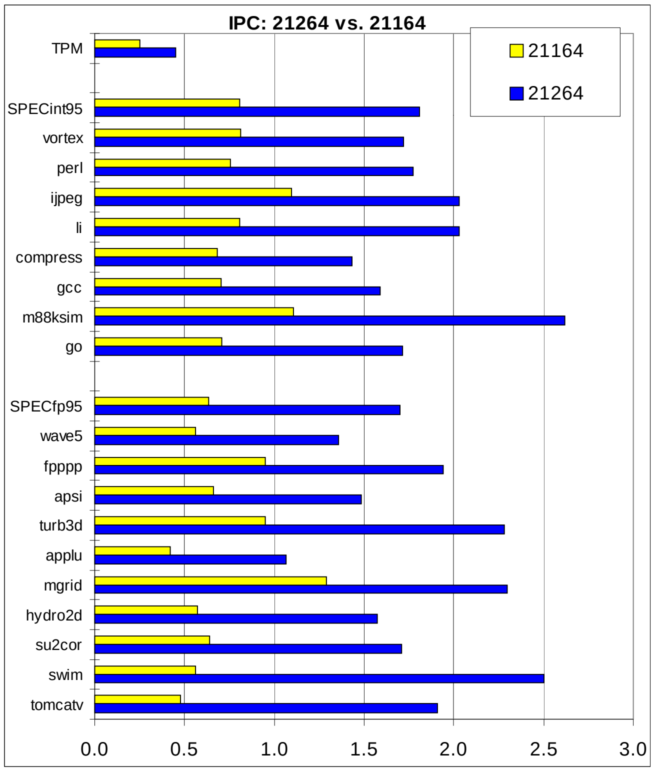

Figure 8 – IPC Comparison

Figure 8 compares the instructions per cycle (IPC) for the ES40 (21264) and the 4100 (21164) systems. Note that IPC in 21264 actually measures number of retired instructions per cycle (the number of retired instructions is lower than the number of issued instructions). The Compaq ES40 results were obtained with code specifically scheduled for the 21264. It also benefited from other generic compiler enhancements not included in the AlphaServer 4100 code.

The 21264 achieves an average IPC in the range of 1.5 to 2.5 for the majority of workloads. This is 2 to 3 times the 21164. Note that IPC improvements exclude gains due to clock speed.

All the architectural improvements discussed in this paper contribute to the IPC improvement of the 21264. Perhaps out-of-order execution – primarily implemented by the map and queue pipeline stages – is the primary contributor. Out-of-order execution reduces processor stalls by executing younger instructions sooner. Earlier execution reduces the latency to execute critical-path computations. For example, the 21264 can execute load instructions earlier, allowing other operations to be overlapped with the data access of the load. This increases functional unit utilization and lowers dependency on the detailed instruction schedule. The obtained 21164 performance is very dependent on the instruction schedule, as are other static compiler-scheduled microprocessors. The 21264 hardware, however, can schedule around stalls that are difficult to predict at compile time, such as many cache misses. It also can improve performance on workloads that have not been tuned for optimum schedule.

The improvement in IPC is higher in floating-point than integer workloads. This is partially due to the architectural advancements discussed so far, and also due to the memory system improvements that will be discussed in the next section.

Though the 21264 substantially improved the transaction-processing IPC, it still requires more than two cycles to execute every instruction. This is substantially higher than in the other workloads we examined. The transaction-processing workload contains many mispredicts and cache misses and is sensitive to memory latency. It presents an opportunity for further improvements in the future-generation Alpha systems.

3.4 INSTRUCTION EXECUTION

The 21264 has four integer execution pipelines while the 21164 has only two. The 21264 has lower floating-point divide latency and new square-root instructions, but otherwise the floating-point functional units are similar. Functional unit improvements are only a small contributor to faster floating-point execution.

A few of the applications (m88ksim in particular) *sustain* integer IPC's on the 21264 that are more than the peak integer performance of the 21164 - two instructions per cycle. Clearly, the extra 21264 integer functional units are necessary for the highest performance in those cases.

The fastest floating-point applications sustain about 50% of the peak floating point performance. Though more floating-point functional units could improve performance in some cases, they are not as advantageous, in general, as the two additional integer pipelines included in the 21264. (An exception to this rule may be the Linpack benchmark, shown in Table 1, which could benefit from additional floating-point pipelines.)

3.5 INSTRUCTION RETIRE AND KILL

As explained earlier, the 21264 fetches and speculatively issues a sequence of instructions – often down the path predicted by the branch predictor. An instruction result is committed to the architectural state at *retire* time. If the wrong path is taken, the instructions are not retired (killed). This speculative execution exposes instruction parallelism and improves performance when the speculation is correct. On the other hand, the penalty for speculative execution is in the overhead for killed instructions.

Figure 9 shows the percentage of all issued instructions that don't retire (killed). The percentages for the integer and transaction processing workloads closely mirror the branch mispredicts. This data shows that the floating-point workloads have fewer killed instructions than the other workloads. This is expected since there are so few branch mispredicts in the floating-point workloads (Figure 6). However the percent of non-retired instructions remains significant for the floating-point applications. That is because memory traps (described in the next section and more prevalent in the memory-intensive floating-point workloads) are another key contributor to non-retired instructions.

Instruction kills reduce performance because they limit the number of in-flight instructions and the exposed instruction parallelism. The issue slots wasted by killed instructions are not a large performance problem, even though more than 40% of the issued instructions are killed in some of the applications. The queues issue older instructions before younger ones so an issue slot used by a killed instruction would be idle in a non-speculative processor anyway. Speculation provides the opportunity to expose more instruction parallelism with little down-side performance risk. A high fraction of kills indicates that this opportunity was not fully realized in some workloads.

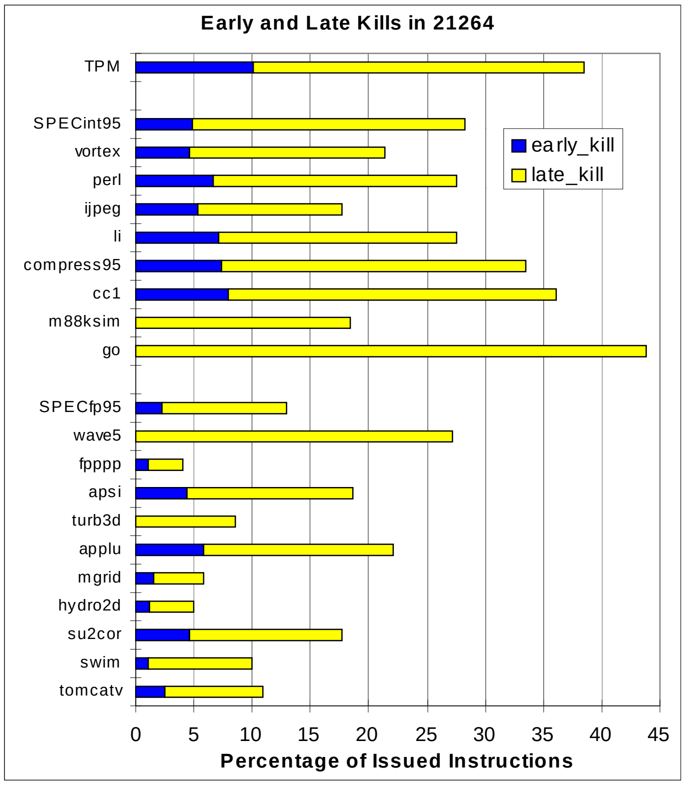

Figure 9 – Non-retired Instructions

Figure 9 also splits the kills into early and late. Early_kill indicates that the instruction was killed before it entered an issue queue (stage 2 or sooner). Late_kill indicates that the instruction was killed later. The data indicates that late_kill is more frequent than early_kill – early kill is typically 25% or less of the total kills. There are simply fewer early pipe stages than there are later ones. For example, the branch and jump/jsr mispredict latencies are seven or more cycles – only three of which are early. Instruction kills can occur long after an instruction enters the issue queue, particularly if the instruction causing the kill stalls.

4. MEMORY SYSTEM IMPROVEMENTS

The memory systems of the 21264 and 21164 are entirely different. Table 2 summarizes the differences. Like the earlier stages in the instruction pipeline, the 21264's memory system is highly parallel and out-of-order. It can manage up to 32 in-flight loads and 32 in-flight stores as well as 8 outstanding off-chip cache misses and 8 outstanding cache victims. This parallelism is exposed (automatically) throughout the memory system of the ES40. The ES40 crossbar memory system (surrounding the 21264) can manage more than 24 outstanding cache miss requests and cache victims. It also schedules memory operations out-of-order within this 24-entry in-flight window for the highest efficiency.

	ES40	4100
CPU	21264	21164
Outstanding References		
Loads	32	21
Stores	32	21
Cache Misses	8	2
Prefetches	yes	yes
Modify Intent	yes	no
WH64	yes	no
Translation Buffer (TB)		
ITB	128-entry	48-entry
DTB	128-entry	64-entry
On-chip Caches		
1st-level Dcache	64 KB, 2-way	8 KB
Latency	4.4 ns (3 cy)	3.3 ns (2 cy)
Peak Bandwidth	10.6 GB/sec	9.6 GB/s
1st-level Icache	64 KB, 2-way	8 KB
2nd-level unified cache	N/A	96K, 3-way
Latency (min)		12ns (7 cy)
Peak Bandwidth		9.6 GB/s
Off-chip Cache	8 MB	8 MB
Latency	24 ns (16 cy)	42ns (25 cy)
Peak Bandwidth	7.1 GB/sec	1.33 GB/sec
Bus	Dedicated	Shared
Memory Latency (min)	183 ns	210 ns
Peak Bandwidth	5.3 GB/sec	1.1 GB/sec
Copy Bandwidth		
1-CPU MB/sec	1197	263
4-CPU MB/sec	2547	470

Table 2 - Key Memory System Features of the ES40 and 4100

In addition to the automatic hardware techniques, the memory system parallelism can also be explicitly exposed with software cache block prefetches of different flavors: normal (read-only), modify-intent (writeable), and the new Alpha WH64 (write-only). The modify-intent prefetch loads a block into the cache in a state that can be immediately written. The WH64 prefetch is a hint to hardware that the block is write-only; this makes the read of the previous value of the cache block unnecessary on a cache miss.

The 21264 has only a single on-chip cache level while the 21164 has two. Although the total amount of on-chip cache is comparable for the 21264 and 21164, the 21264 benefits from more efficient on-chip caches. The data cache has longer latency than the first-level cache in the 21164, but the 21264’s first-level cache is larger and more associative. The latency-tolerant instruction execution capabilities of the 21264 make the first-level load latency less critical.

The off-chip caches are the same size and associativity in the ES40 and 4100 systems - 8MB, direct-mapped. Consequently, the cache misses per instruction are similar in the two systems. However, the ES40 memory system outperforms the 4100 memory system in other aspects. Some of the benchmarks hit frequently in the off-chip cache. They benefit from the lower latency (about half) and higher bandwidth (about five times) of the ES40 off-chip cache. Some of the workloads we study, like the transaction-processing workload and some of the floating-point benchmarks, have high cache miss rates even with the 8MB caches. These applications benefit from the improved DRAM memory system in the ES40.

4.1 MEMORY BANDWIDTH AND LATENCY IMPROVEMENTS

Table 2 shows the large improvement in memory bandwidth in ES40 vs. 4100. This is largely responsible for the floating-point performance improvement from the 21164 to the 21264. The STREAM benchmark measures sustainable memory bandwidth in megabytes per second (MB/s) across four vector kernels: Copy, Scale, Sum, and SAXPY[12]. We show only the results for the Copy loop (the other loops have similar results). The raw bandwidth improvements of the off-chip caches and the crossbar memory system of the ES40 enabled this capability, as did the memory system parallelism. The 21264 also benefits from the new WH64 prefetch.

Not only does the ES40 have higher memory bandwidth performance for a single processor, it also scales better with multiple processors. With 4 processors the ES40 is more than five times faster than the 4100. The crossbar interconnect of the ES40 has more bandwidth and is more adept at exposing memory system parallelism than the bus-based 4100. This capability is a contributor to the high throughput of the ES40, as evidenced by the SPECfp_rate95 results in Figure 2 and Table 1.

Figure 10 compares memory bandwidth of the STREAM Copy loop across Alpha servers as well as other leading-vendor systems [12]. Note that the ES40/667MHz data is based on preliminary engineering measurements and not yet submitted for publication. This data shows that the memory bandwidth on ES40 is a factor of 1.6 to 3 times higher than all other systems shown.

The comparison between the ES40's of different speed in Figure 10 is interesting. The four-CPU memory bandwidth improves by more than 40% though the processor speed improved by only 33% (500 to 667), and the DRAM bandwidth did not improve. The 667 MHz ES40 off-chip cache has more than twice as much bandwidth as the 500 MHz ES40. This is crucial for the ES40 system memory bandwidth since, at maximum copy bandwidth, the off-chip cache of the 500 MHz system is more than 50% utilized

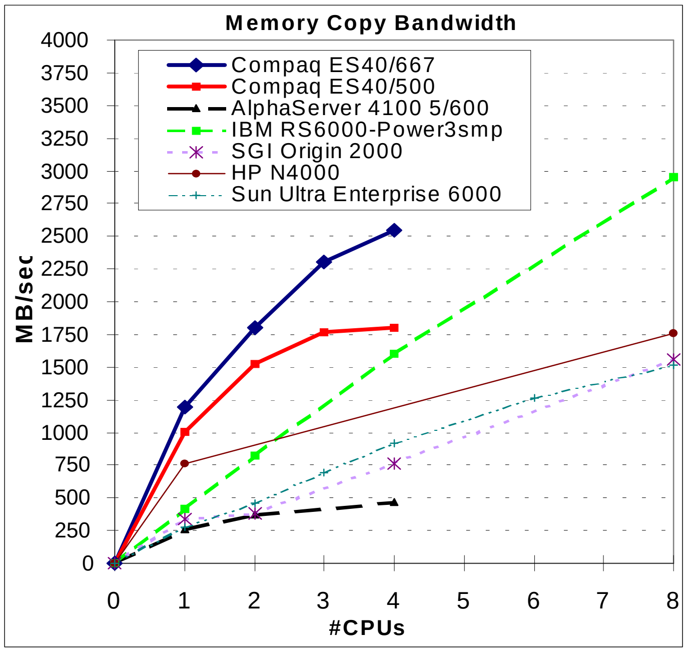

Figure 10 – STREAM Copy Bandwidth

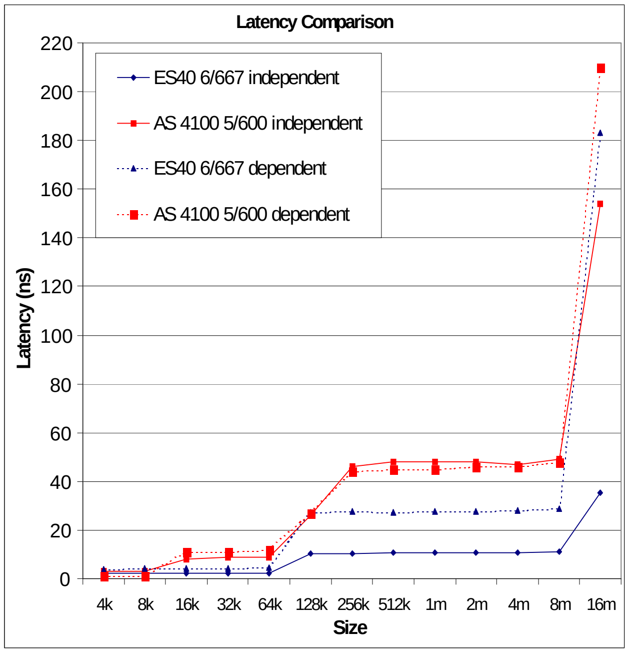

Figure 11 – Load Latency (and Bandwidth) Comparison

servicing probe requests from remote processors. The faster off-chip cache gives the 667 MHz ES40 much more performance. The original 500Mhz ES40 design is not as well optimized for memory performance.

Note that the memory bandwidth improvement from three to four CPUs is still modest in the 667 MHz ES40. This platform may need additional independent paths to memory (to reduce memory conflicts) and additional probe bandwidth to support more than four CPUs at full bandwidth.

Figure 11 compares ES40 and 4100 using both "dependent-load" and "independent-load" latency. The "dependent-load latency" [11] measures load-to-use latency where each load depends on the result from the previous load. It is largely a measure of memory latency. The "independent-load latency" is a measure of bandwidth, since there is no dependency between consecutive loads. It is lower in platforms that allow more simultaneous outstanding requests (which can overlap), like the ES40. The lower axis in Figure 11 varies the referenced data size to fit in different levels of the memory system hierarchy.

The results in Figure 11 show that ES40 is 40% faster in "dependent-load" memory latency (16M size) vs. the 4100. The advantage is much higher (a factor of 4 times) in the "independent-load" memory latency (bandwidth). Note that the 21264 allows eight outstanding read requests while 21164 allows only two, therefore the improvement in the "independent-load latency" on ES40 is so substantial. The memory system of ES40 is designed to expose such memory parallelism. The applications that show the highest improvement on ES40 vs. 4100 are likely to allow multiple outstanding independent memory references.

4.2 FEWER MEMORY TRAPS

A replay trap is a mechanism that kills and restarts all instructions in the 21264. This is needed in situations where a load or store instruction cannot be executed due to a (relatively infrequent) condition that is detected after that instruction is issued, and often only after the memory address is calculated and compared to other in-flight memory references.

The 21264 and the 21164 use memory traps in similar circumstances. Some common 21264 traps are: (same address) store-load order traps, load queue or store queue overflow conditions, and some cache index match conditions. Some common 21164 traps are: write or miss buffer overflows, potential ordering violations, and load-miss-and-use traps. One notable difference is the 21264's store-load order traps. These occur because a later load issued before a prior store to the same address. This can't happen on the in-order 21164. Another difference is the 21164's load-miss-and-use trap. This occurs when the consumer of load data speculatively issues assuming the load would hit, though the load really misses. The 21264 issues a mini-restart in this case rather than a trap.

Figure 12 compares the number of replay traps in the 21264 and the 21164. The data indicates that there are many fewer replay traps in the 21264 compared to the 21164. However, the number of traps in several floating-point workloads and the transaction processing workload remains substantial (around 10 per 1000 instructions). Perhaps the biggest reason for the reduction in replay traps in the 21264 is the more highly-parallel memory system. Write and miss buffer overflow conditions cause many of

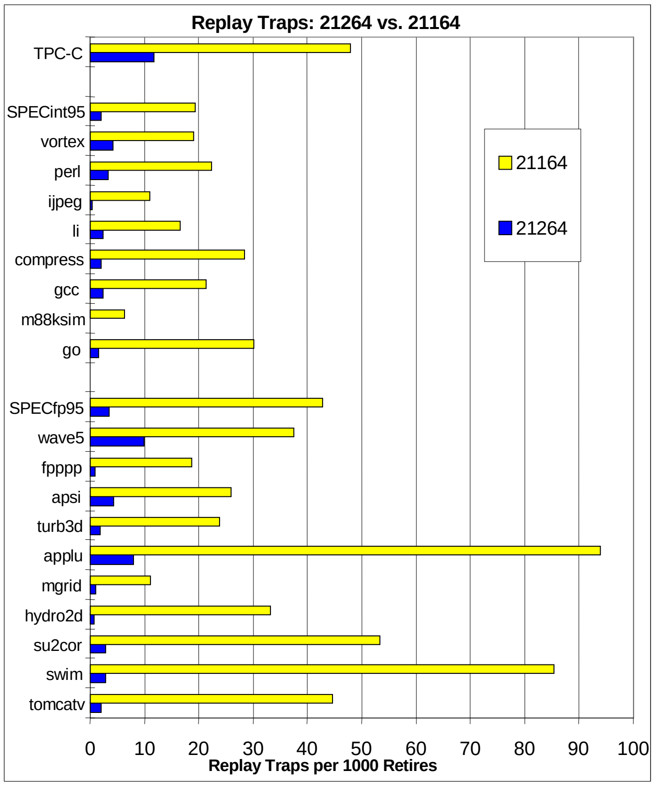

Figure 12 – Replay Trap Comparison

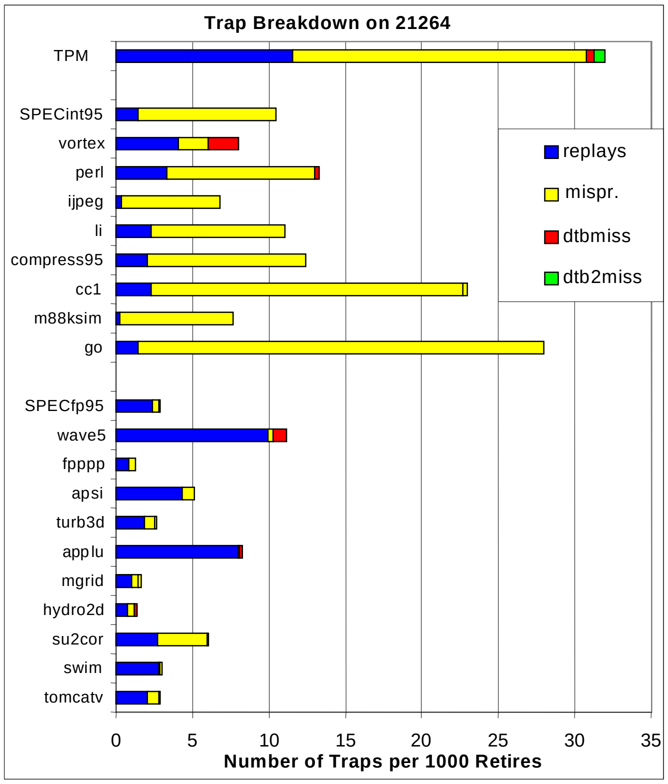

Figure 13 – 21264 Trap Breakdown

the 21164 replay traps. The 32 outstanding loads and stores in the 21264, as well as the compiler use of prefetch instructions, provide ample parallelism for the 21264 to avoid many of the overflows.

The ProfileMe and DCPI tools allow a program to be tuned by identifying areas in the program with a high number of traps and then reducing traps by a variety of techniques, including: memory padding to prevent cache index matches and changing the order of loads/stores. Such simple code/compiler modifications gave 10-15% improvement in several SPEC95 workloads.

Figure 13 shows the breakdown of all traps on the 21264, including memory replay traps, branch mispredict traps, and memory translation buffer miss traps. The number of translation buffer misses (dtb miss, dtb2 miss) is small on all of the workloads we examined. The transaction processing workload shows the highest number of all traps combined. Mispredict traps are prevalent in both integer and transaction processing workloads, but not in floating-point workloads. Replay traps are present in all workload classes, but are more prevalent in the transaction processing and some of the floating-point workloads. Replay traps occur in these floating-point workloads primarily since they stress the memory system. This can cause overflow traps despite 21264's large number of in-flight memory references.

5. CONCLUSIONS

We examined the architecture and performance characteristics of the Compaq Alpha ES40 shared memory multiprocessor – a four processor system that is 3% to three times faster than other similar systems on some important industry standard benchmarks. We described the architecture of the 21264 microprocessor and the surrounding memory system logic included in the ES40 and compared it to the previous-generation four-processor Alpha system. The benchmark results show the ES40 is often 2-3 times the performance of the previous-generation system at nearly the same processor clock rate. We analyzed the architectural techniques that enabled this performance improvement.

The fetch and execution improvements include: a larger instruction cache, a more sophisticated branch predictor, out-of-order and speculative execution, and more functional units. We showed that these improvements contributed to the reduction in cycles per instruction. We also showed that these mechanisms successfully kept many instructions in flight – a key requirement to extract instruction parallelism.

The fetch and execution improvements are complemented by memory system improvements. The (DRAM) memory system in the ES40 is lower latency than the previous-generation system on off-chip cache misses by a small amount – giving a modest performance benefit. But the primary improvement in the ES40 is its higher bandwidth and ability to expose and manage many parallel memory system requests simultaneously. Both the off-chip cache and the crossbar memory system deliver about five times more bandwidth than the previous generation Alpha system. This and the parallel capabilities of all levels in the memory hierarchy makes the ES40 substantially faster than the previous generation on memory-bandwidth intensive workloads.

We use only a sample of commercial and technical workload in our analysis. The SPEC benchmarks do not generate a lot of operating system activity and fit in megabyte-sized caches. Many other real applications have different characteristics. Further study is needed on a broader range of workloads and applications to expand the characterization of the ES40 presented in this paper.

Acknowledgments

The authors would like to thank to Jason Campoli for profiling data collection and to Cynthia Murray, Peter Gilbert, John Shakshober, and Dave Wilson for help with benchmarks. Mark Vandervoorde provided help and support with the performance monitoring tools. Special thanks to Rick Hill and John Murray from the Compaq ES40 team, Dave Webb, Steve Lang, Pete Bannon, Geoff Lowney, Rich Grove, Steve Hobbs, Bill Noyce, Kent Glossop, and many others. The authors are grateful to Masood Heydari for supporting this work, and to the reviewers for numerous helpful comments.

REFERENCES

[1] R. E. Kessler, “The Alpha 21264 Microprocessor”, IEEE Micro, March-April 1999, pp. 24-36.

[2] R. E. Kessler, E.J. McLellan, and D.A Webb, “The Alpha 21264 Microprocessor Architecture”, Proc 1998 IEEE Int’l Conf. Computer Design, Oct. 1998, pp. 90-95

[3] J. H. Edmondson, P. Rubinfeld, R. Preston, and V. Rajagopalan, “Superscalar Instruction Execution in the 21164 Microprocessor”, IEEE Micro, Apr. 1995, pp. 33-43.

[4] DCPI and ProfileMe External Release page: http://www.research.digital.com/SRC/dcpi/release.html

[5] J. Dean, J. Hicks, C. Waldspurger, W. Weihl, G, Chrysos, “ProfileMe: Hardware Support for Instruction-level Profiling on Out-of-Order Processors”, 30th Symposium on Microarchitecture (Micro-30), Raleigh, NC, Dec. 1997.

[6] Z. Cvetanovic and D. Bhandarkar, "Characterization of Alpha AXP Performance Using TP and SPEC Workloads", *The 21st Annual International Symposium on Computer Architecture*, April 1994, pp. 60 - 70.

[7] Z. Cvetanovic and D. Bhandarkar, “Performance Characterization of the Alpha 21164 Microprocessor Using TP and SPEC Workloads,” *The Second International Symposium on High-Performance Computer Architecture* (February 1996), pp. 270–280.

[8] Z. Cvetanovic, and D. D. Donaldson, “AlphaServer 4100 Performance Characterization, *Digital Technical Journal, Vol 8 No. 4, 1996:* pp. 3-20.

[9] Information about the Transaction Processing Performance Council (TPC) is available at http://www.tpc.org

[10] SPEC95 Benchmarks (Manassas, Va.: Standard Performance Evaluation Corporation, 1995) information available at http://www.specbench.org/osg/cpu95/results/

[11] Information about the lmbench available at http://reality.sgi.com/employees/lm_engr/lmbench/whatis_lmbench.html

[12] The STREAM benchmark information available from the University of Virginia, Department of Computer Science (Charlottesville, Va.) at http://www.cs.viginia.edu/stream

[13] “Compiler Writer’s Guide for the Alpha 21264”, Compaq Computer, June 1999, Order Number EC-RJ66A-TE, ftp://ftp.digital.com/pub/Digital/info/semiconductor/literature/dsc-library.html