A Study of Single-Chip Processor/Cache Organizations for Large Numbers of Transistors

Matthew Farrens
Gary Tyson

Computer Science Department
University of California, Davis
Davis, CA 95616
(farrens@cs.ucdavis.edu)
(tyson@cs.ucdavis.edu)

Andrew R. Pleszkun

Department of Electrical and Computer Engineering
University of Colorado-Boulder
Boulder, CO 80309-0425
(arp@tosca.colorado.edu)

Abstract

This paper presents an examination of different cache and processor configurations assuming transistor densities will continue to increase as they have in the past. While in the short term any additional transistors should clearly be put into increasing the size of on-chip first-level caches, as transistor counts increase above a certain level this is no longer holds true. As cache sizes increase, so does the difficulty of accessing the cache in a single-cycle. Context switches also present a potential problem.

A trace-driven simulation-based study of a wide range of cache configurations and processor counts was performed, and the results are presented here. In order to compare different configurations, the concept of an *Equivalent Cache Transistor* is presented. We found that the access time of the first-level data cache proved critical; configurations with a small single-cycle access data cache consistently outperformed much larger 2-cycle access caches. In addition, it appears that once approximately 15 million transistors become available, a two processor configuration is preferable to a single processor with correspondingly larger caches.

Keywords: Cache, Multiple Processor, VLSI

1. Introduction

Advances in VLSI technology are reducing the minimum feature sizes for semiconductor devices at a tremendous rate. Transistor densities are currently measured in the millions per die, with densities of tens of millions on the horizon. It is therefore important to begin an examination of how best to utilize such enormous transistor counts.

In the past, there has not been much question as to the best way to use additional transistors. Transistor count limitations forced on-chip word sizes to be small and floating point, virtual memory and cache hardware to be located off-chip. However, a state has now been reached where existing transistor densities support a 64-bit word and the placement of floating-point, virtual memory and cache hardware on-chip with the processor. For the short term, as transistor densities continue to increase, it is clear that increasing the size of on-chip caches will provide the best performance return on transistor investment. What is not clear is how much longer this will continue to hold.

It is certainly true that as cache sizes increase, the cache hit rate increases. However, this increase is not linear. There comes a point at which doubling the size of the cache leads to only a marginal increase in the hit rate. In addition, there are a number of drawbacks to increasing cache size without bound. For example, once a cache reaches a certain size, accessing it in a single cycle will become impossible. The processor designer will be forced to develop a pipelined cache design, or move to a hierarchical cache configuration (with a smaller faster cache located in front of a larger, slower cache). Placing such a hierarchy of caches on-chip presents a new set of problems, like maintaining cache coherence between 3 levels of cache (assuming another cache exists off-chip to handle on-chip misses.) Such a configuration may make supporting multiprocessor systems very difficult.

Technological constraints will undoubtedly limit off-chip bandwidths as well. This will be due (in part) to the number of pins available for wide off-chip busses and, perhaps more importantly, due to the power needed to drive signals off-chip. This exposes another problem with large on-chip caches; due to context switches, several contexts will have to be stored on-chip. If they are not, the time taken to bring new contexts on-chip and externally store the existing context could lead to significant delays. The effects of context switches must be considered since context switches are a real phenomenon in any system and can aggravate problems that are caused by limited off-chip bandwidth.

At what point does one stop increasing the size of the caches and look at alternate configurations? Since marginal performance improvements are achieved as cache sizes become large, adding other types of performance improving features could have a more significant impact on the system performance. For example, it may make sense to invest some transistors in branch prediction hardware, such as that proposed by Yeh and Patt [YeP91]. Another use of *extra* transistors could be for register renaming schemes, or more hardware support for multiple or out-of-order issue of instructions. Also, at some point, when most of the chip area is used as memory, it becomes reasonable to consider placing more processors on the chip. While this would decrease the area available for on-chip memory, such a move could provide an increase in the throughput of the system as a whole. If multiple processors are placed on a single chip, the results presented later in this paper could be used to decide whether, from the perspective of the number of transistors used, it makes sense to have more than one floating point unit on a chip. It is possible that allocating the transistors used in one of the on-chip floating-point units to a slightly larger cache may result in a significant performance improvement.

The goal of this work is to help designers better deal with such questions by presenting the results of our investigation into the tradeoffs between the usage of silicon real estate and system throughput. Using trace-driven simulations, we have examined the performance of a wide range of cache/processor configurations, including many that are currently unrealizable. These configurations were modeled because we expect the results presented here to apply not only to current technologies but to emerging ones as well.

2. Background

The recent announcements of single-chip processors such as the DEC Alpha and MIPS 4000, each with 1+ million transistors, motivate the need to look at how best to organize systems when 10 or 20 million transistors become available. Both these processors are 64-bit processors, with floating point and virtual memory support located on-chip with the processor. They both also have relatively small on-chip caches.

Given that we are interested in the best way to utilize a given number of transistors, we must determine which combinations of parameters lead to systems that fit a particular transistor budget. For example, how does a single processor system with a 128K byte second level cache and 32K byte instruction and data caches compare to a system with 2 processors, each backed by a 16K byte instruction cache and a 16K byte data cache that share a 128K byte second-level cache? Even though the two systems have identical amounts of memory, they are not equivalent in terms of transistor counts, since the second system must account for the transistors needed for the second processor.

Accounting for the second processor would not be particularly difficult were it not for the fact that, due to the more random nature of processor logic as compared to cache logic, the transistor densities in a cache are much higher than those found in a processor. To account for these different densities we developed the concept of an **Equivalent Cache Transistor** (ECT). Each transistor used to build a cache is equal to 1 ECT. Since transistors in the processor are less densely placed, one real transistor in the processor will correspond to more than 1 ECT.

To approximate the number of ECTs needed to implement a processor we will use the following approach. First, we approximate the number of transistors that are required to implement the caches in an existing single-chip processor. (This transistor count is in terms of ECTs.) In our case, we used the DEC Alpha processor as our model. This processor has 16K bytes of on-chip cache and uses a 6 transistor standard cache cell design [DWAA92, Site93]. Then, using this transistor count in conjunction with an observation of the amount of on-chip space that is occupied by the caches, we can compute the number of transistors (in ECTs) needed to implement the processor.

Consider the 8K byte direct-mapped caches of this processor; given a 32 byte line size, there are 256 lines in the cache. Since these are virtual addressed caches and the virtual address is 64 bits, 52 bits are needed per tag. With 256 lines and 52 tag bits per line (plus the dirty and valid bits), roughly 14K bits are need for the tag memory, or $14K \times 6 = 84K$ transistors. The data memory array for the cache requires:

$$8k\ bytes \times \frac{8\ bits}{byte} \times \frac{6\ transistors}{bit}\ =\ 384K\ transistors$$

The cache with the data and tag arrays requires roughly $(84K + 384K)\ =\ 468K$ transistors.

Since our model processor uses both an instruction cache and a data cache, each 8K bytes in size, the caches require approximately 936K transistors. Observation of the model processor die indicates that the caches appear to require approximately 1/4 (25%) of the chip area. If the entire chip were allocated only to caches, we would be able to place $(936K \times 4) = 3,744K$ ECTs on the die. Since the die only uses 936K of these for caches and the remainder of the chip is occupied by the processor, the processor requires roughly $(3,744K - 936K) = 2,808K$ ECTs. Since this is an approximation, we can say that a processor uses approximately 3M ECTs. (It should be noted at this point that since we define the processor as everything that is not cache, the 3 Million ECTs required by the processor includes the busses, VM support, floating point, control, I/O, etc.)

In order to examine the effects of varying cache sizes and processor counts, we need an equation in terms of these parameters that computes the total number of transistors. The number of transistors needed in the cache includes both the transistors in the data array and the transistors in the tag array. The number of transistors for the cache can be expressed as a function of only the size of the cache if a multiplier term is used to account for the tag array size. For the model processors described above, the 8K byte caches required

$$384K + 84K \ = \ 468K\ ECTs$$

A term we call the *overhead multiplier,* **om**, is computed by dividing the total number of ECTs for the cache by the number of ECTs needed in the data memory array.

$$\frac{468K}{384K} \ = \ 1.21$$

This roughly accounts for the tag arrays in any size cache we look at that has 32 byte lines. Thus, the total transistor count (TTC) can be expressed as:

$$TTC \ = \ P_{num}[P_{tct} + ((ics + dcs) \times 8 \times 6 \times om)] + (scs \times 8 \times 6 \times om)$$

where

P_{num}	is the number of processors
P_{tct}	is the number of non-cache transistors per processor (3M ECTs)
ics	is the instruction cache size in bytes
dcs	is the data cache size in bytes
scs	is the second-level cache size in bytes
om	is the overhead multiplier of 1.21
8	is the number of bits per byte
6	is the number of transistors per bit

This function approximates the number of transistors, in ECTs, that are needed to implement a particular system configuration. While some readers may be uncomfortable with such an approximation, the number of ECTs needed to implement a processor can be treated as just another parameter when analyzing the simulation results.

3. The Simulation Environment

As stated in the introduction, our goal is to evaluate the performance of different cache/processor configurations, in order to help designers make transistor allocation decisions. Quantifying the effectiveness of varying cache sizes and numbers of processors on both processor and system throughput requires a simulation model, a simulator, and some traces over which to evaluate the different configurations. These are presented below.

3.1. The Simulation Model

The simulation model used in our study assumes between 1 and 4 processors, each with its own instruction and data cache, sharing a single second-level cache via a shared bus (See Figure

1). (There are two shared busses between the first-level and second-level caches, one for addresses and one for data.) Bus conflicts that occur are arbitrated in a round-robin fashion. The line sizes for the first-level caches are fixed at 4 words, while the second-level cache has a line size of 8 words. We chose a second-level cache size of 8 words instead of the 16 word line size used on the Alpha to reflect the difference in register word size between the 64bit Alpha and the 32 bit MIPS R3000 used to generate the traces. Numerous cache studies have also validated this as a reasonable size [Smit82, SmGo85].

The processors are single-issue processors with a CPI of 1 (excluding memory references). These can be thought of as non-pipelined processors, since no internal hazards are simulated. The processors can issue both an instruction and a data read on the same clock, and the model assumes there is a single load delay slot following all load instructions.

3.2. The Simulator

The simulator used in this study allows the user to specify each cache size, associativity and access time, as well as the number of processors in a configuration, the number of cycles between first-level cache flushes, the percentage of delay slots after a load that can be filled, the

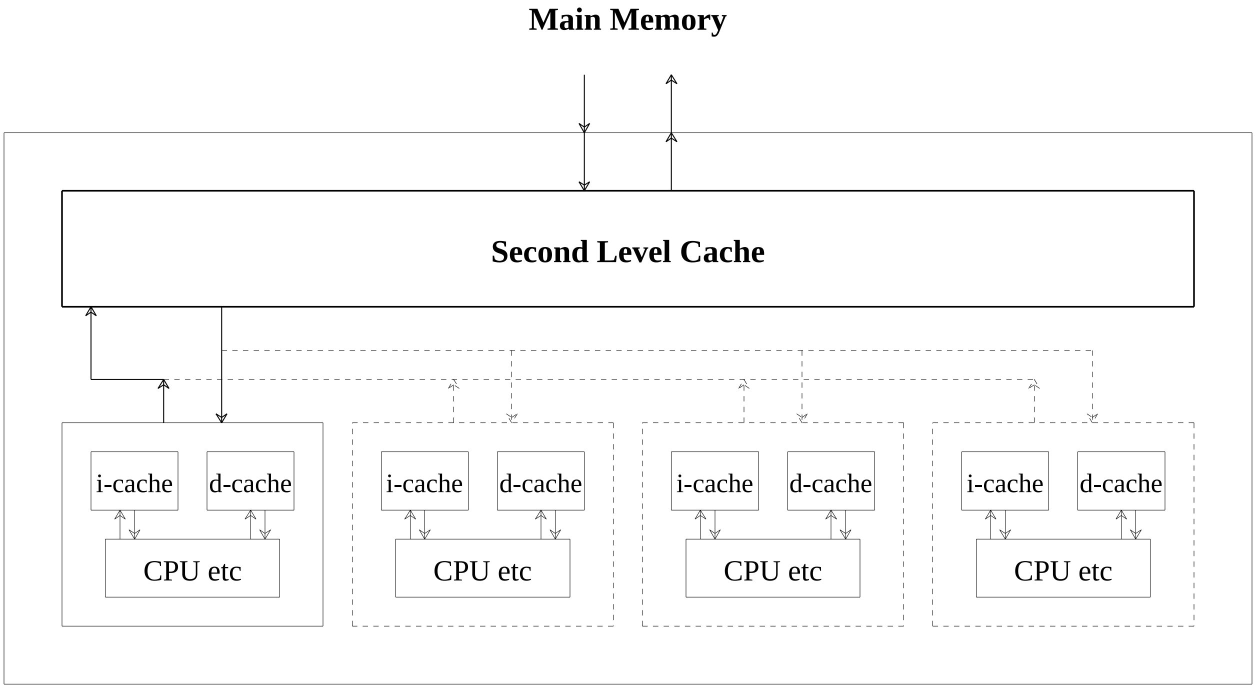


Figure 1.

width of the off-chip read and write busses, the off-chip memory access time, and the amount of pipelining the off-chip memory can support.

3.3. The Traces

Sixteen traces were selected as representative of a typical workload. Nine traces were chosen from the SPEC89 benchmark suite, and augmented by seven additional traces used by Johnson in his book [John91]. The benchmark programs were compiled with the native RISC C compiler (/usr/bin/cc) on a DECstation 5000/240 with optimization level 2. The traces were then gathered using the pixie trace generating facility.

Due to limited time and space resources, it was not possible to use an entire trace as input to our simulator for each configuration modeled. Instead, for each trace we created a randomly sampled trace and used this as input to the simulator. This sampled trace was created by first executing the program to be sampled in its entirety, and tabulating the number of addresses generated. Once that number was known, the program to be sampled was then executed again, with the output piped into a sampling program that created contiguous blocks of a user-specified size that were randomly distributed over the length of the trace. So, for example, in our study we use sampled traces that consist of 100 contiguous blocks of instructions each 75,000 references long. This approach to using shorter traces to represent much longer reference streams is very similar to the approach used in [LaPI88].

In order to get some feel for how representative the sampled traces were, we also calculated the number of unique addresses in each sampled and unsampled trace. These numbers appear in Table 1. There are a number of interesting things to point out in the table. For example, the number of unique data read addresses is often larger in the sampled trace than in the original trace. This is because the sampling often misses the first time a location is written to, so the address gets put in the wrong unique column. In no cases is the actual number of unique data references higher in the sampled trace than in the original trace.

It should also be pointed out that the dnasa7 trace used is only a subset of the entire dnasa7 simulation. The dnasa7 program consists of 7 loops that are executed sequentially. In its original form, it is intractably long. In order to get around this problem, we separated the source code into the 7 different constituent loops, and ran each of them individually. Dnasa7cho was chosen since it had the 3rd highest number of unique instruction references and the 3rd highest number of unique data references.

The table shows that our sampled traces capture on average 58% of the unique instruction addresses and 54% of the unique data addresses. This is notable since our 7.5 million reference long trace is (on average) only .8% of the length of the full trace run. The fact that these

Table 1. Detailed Trace Information

Benchmark Program	Total References	Instructions	Data Reads	Data Writes	Total Number of Unique Insts	Reads	Writes
gcc	72,659,899	55136559	10999366	6523974	67207	6367	306817
sampled	7,499,993	5690254	1133148	676591	51446	18530	67963
espresso	722,494,426	590499826	107037178	24957422	17218	4062	68040
sampled	7,418,953	6047161	1109309	262483	9110	25263	13196
spice	4,706,975,682	3797896227	765764079	143315376	25656	15625	108580
sampled	7500002	6064519	1193310	242173	8786	64827	13101
doduc	369,988,935	272023222	72505623	25460030	25185	4411	20675
sampled	7,423,397	5456924	1465643	500830	15737	3580	4293
xlisp	1,691,475,551	1234262375	290354971	166858148	5437	1535	13534
sampled	7,499,984	5458931	1310690	730363	2262	6037	3890
eqntott	1,459,278,325	1242867748	205009255	11401322	3447	536	428195
sampled	7,499,977	6349398	1081858	68721	741	87581	5473
matrix100	137,784,247	88930970	32590045	16261852	4872	552	60580
sampled	7,499,986	4841305	1776223	882458	500	38450	5099
tomcatv	2,248,356,429	1467535289	611393143	169428117	4385	537	915107
sampled	7,500,031	4885150	2041431	573450	2056	449698	158255
dnasa7cho	2,406,530,343	1528741657	621887005	255901681	5418	635	371624
sampled	7,499,969	4756077	1936422	807470	1118	240512	26071
awk	16,815,613	12211251	2703339	1901023	4584	3133	3222
sampled	7,500,014	5448866	1204599	846549	3903	3017	1573
compress	21,309,967	16672328	2957065	1679732	1726	2098	103272
sampled	7,499,988	5859063	1038731	602194	420	23490	51189
dhrystone	457,005,816	324504414	84000764	48500638	1313	64	217
sampled	7,499,992	5357225	1358413	784354	513	28	43
grep	17,908,191	13590761	2647143	1670287	1151	2087	2184
sampled	7,500,010	5692057	1108287	699666	493	2076	2114
linpacks	104,134,441	73957086	19890676	10286962	7446	651	21037
sampled	7,499,939	5378169	1392494	729276	2822	18881	1564
troff	13,070,251	10302886	1892280	871386	7973	4402	7428
sampled	7,494,842	5911709	1084260	498873	6178	2706	2587
whetstone	15,501,877	11281052	3233402	987423	1075	451	81
sampled	7,500,004	5456901	1576992	466111	893	388	14

sampled traces include so many of the existing unique references in a trace makes them very valuable cache analysis traces. However, the fact that the ratio of unique references to total references is much higher in the sampled traces than in the original traces implies that the CPI for the sampled trace might be affected. In fact, we did find that the CPI for a full run of several different traces was somewhat lower than the corresponding CPI for the sampled trace, as one would expect.

4. The Simulations

Time and resource constraints prevented us from achieving the ideal goal of exhaustively simulating all possible configurations of processors and caches. Thus, we decided to investigate those system configurations (the combinations of P_{num}, *ics*, *dcs*, and *scs*) for which *TTC* is between 4 and 25 million ECTs, with the following restrictions placed on the system configurations:

(1) The data cache size is restricted to be between one-half and four times the instruction cache size, or

$$\frac{ics}{2} < dcs < (4 \times ics)$$

(2) The second-level cache must be at least 4 times the first-level data cache size.

$$scs \geq 4 \times dcs$$

This seems reasonable since the second-level cache has to be large enough to support a sufficiently larger working set than the data cache to make it worthwhile.

The second-level cache associativity is also fixed at 4-way. An argument can be made for making the second-level cache direct-mapped (and we intend to simulate the system with a direct-mapped second-level cache in the future). However, the afore-mentioned resource constraints forced us to chose a single associativity organization. While we realize that a large direct-mapped second-level cache should perform well [Hill87] we chose a 4-way set associative cache since in some of the system configurations, the second-level cache is not much larger than the first-level caches it supports. This is especially true when we look at multiple on-chip processor configurations where the second-level cache is supporting several first-level caches.

The context switch time or time sharing quantum is the number of clock cycles that occur before a context switch occurs, and is fixed at 75000. This parameter causes the first-level caches to be flushed every 75000 cycles (the second-level cache is not affected). We also assume that since off-chip memory requests service second-level cache misses, the off-chip memory will be pipelined to some depth. For this paper we set this to allow fully pipelined operation so that we will not experience delays due to memory conflicts. The data pin size was set to 32 bits requiring multiple data transfers to transfer a cache line.

The associativity of the first-level instruction cache is fixed at direct-mapped, and the data cache can be either direct-mapped or 4-way set associative. The access time to the instruction cache is fixed at 1 cycle. For the data cache, we looked at an access time of 1 or 2 cycles. The rationale here is that once the data cache grows to a certain size, it may not be possible to access the cache in one machine cycle. For the second-level we selected an access time of 2 or 4 cycles, with the rationale for using these times again due to cache size. As the second-level cache grows larger and more processor are added to the chip, it is difficult to imagine a design in which the second-level cache can be accessed in a single processor cycle.

The simulation model also assumes a load delay slot after each load. A simulation parameter allows the user to specify what percentage of these slots are filled during a simulation - if the user specifies 50% filled, for example, half of the time the instruction following the load instruction is allowed to proceed and the other half of the time it is blocked.

The range of parameters that were simulated can be found in table 2. As stated in the introduction, we strove to avoid placing technological limitations on cache configurations; therefore, the cache configurations were chosen to be inclusive, and therefore do not limit performance to current established capabilities. Some configurations are clearly unrealizable (e.g. accessing a 128K, 4-way set associative cache in a single cycle), but may not continue to be in the future.

After computing all the legal combinations of parameters that met our constraints, we found we had 1504 different system organizations to simulate for a single processor on chip. After analyzing the single chip data, we further restricted the runs for multiple processors even more by eliminating the 4-way set associative data cache runs. Even with these restrictions we

Table 2. Range of Parameters used in Simulations

No. of processors (P_{num})	1,2,3,4
First-level I-cache	Cache size - 4K, 8K, 16K, and 32K bytes Associativity - Direct Mapped Access time - 1 cycle
First-level D-cache	Cache size - 2K, 4K, 8K, 16K, 32K, 64K and 128K bytes Associativity - Direct Mapped, 4-way Access time - 1 and 2 cycles
Second-level cache	Cache size - 8K through 256K bytes Associativity - 4-way Access time - 2, 4 cycles
Main Memory	Access time - 25, 75 cycles Pipeline Depth - 75 outstanding requests (fully pipelined)
Context switch time	75,000 cycles

still had 240 runs for each 2-processor simulation, 212 for each 3-processor simulation, and 148 for each 4-processor configuration. While these numbers are significantly lower than the number of runs for a single processor, their execution took at least as long since simulation time is approximately proportional to processor count, and there are many different combinations of traces that need to be run together.

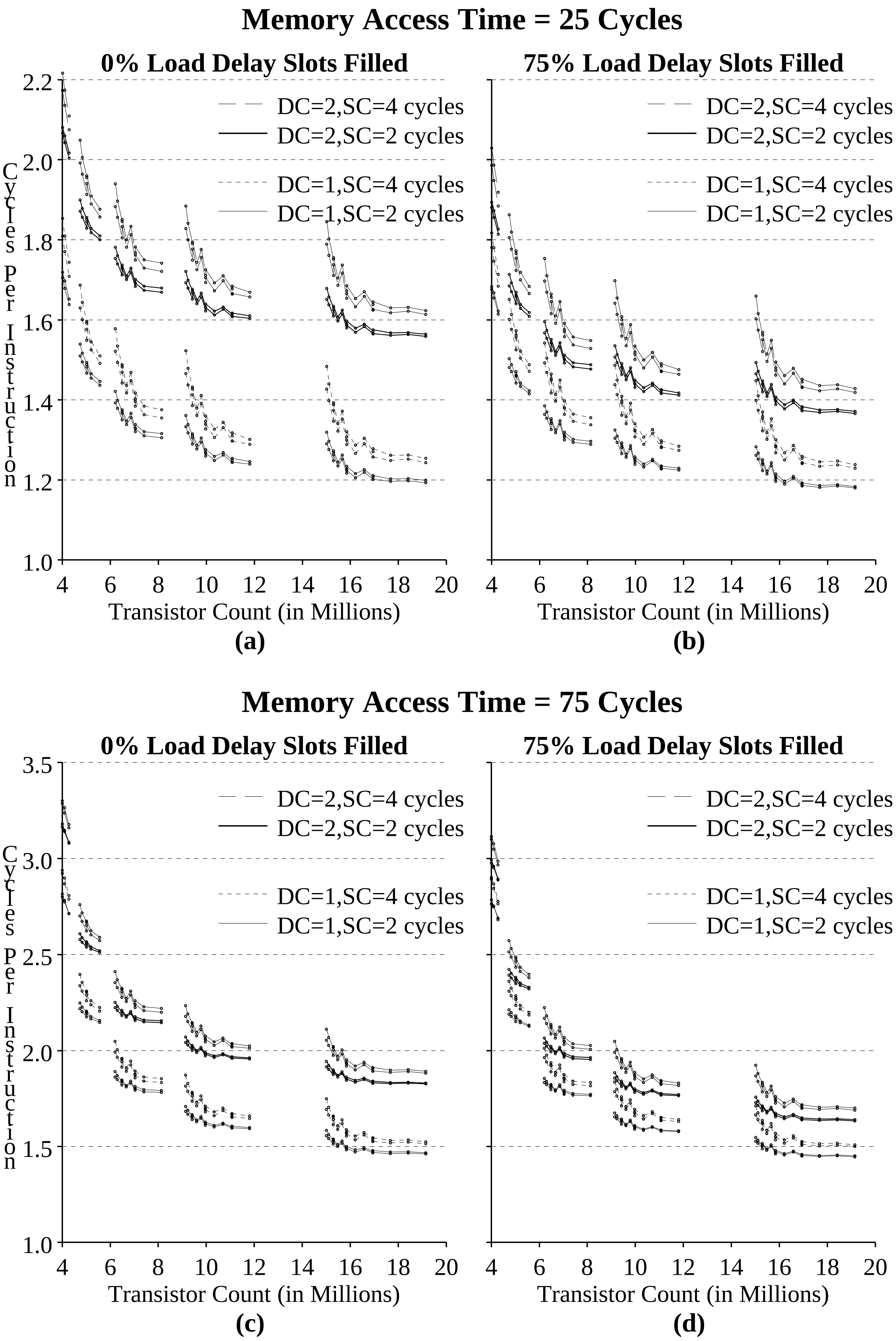
Once the simulations had all been executed, the resulting data for each configuration was averaged over all the runs. It is this average data that will be presented in the following sections.

5. Single Processor Simulation Results

Since there are literally tens of thousands of simulation results, presenting them in a coherent manner presents a challenge, especially in a limited amount of space. We decided to present as many simulation results as possible in the form of figures. However, since some of our interpretations of these results may not be immediately evident from only looking at the figures, or because the reader may wish to analyze the results from a different perspective, we have include a table of all the simulations results in Appendix A.

Before looking at multiprocessor results, we will first present the results for the single processor simulations. Figures 2a through 2d plot performance in terms of clocks per cycle (CPI) versus transistor count. The four plots group results by main memory access times and the percentage of delay slots filled. Within a particular plot there are 40 curves, organized into 5 groups of 8. Each group represents a particular second-level cache size. Reading from left to right, the second-level cache sizes vary from 16K bytes at the left to 256K bytes at the far right, doubling for each group. Within each group of curves there are 4 sets of 2 curves. These curves represent combinations of data cache associativity, data cache access times and second-level cache access times. For each set of 2 curves, the lower curve is for a system with a 4-way set associative data cache while the upper curve is for a direct-mapped data cache. Starting from the bottom, the 4 sets of curves represent data and second-level caches with the following access times: 1:2, 1:4, 2:2 and 2:4 cycles.

Based on these figures, for a given probability of filling a load delay slot, a particular main memory access time and a fixed number of transistors, we find that it is most important to keep the data cache access time to a minimum, while the access time of the second-level cache is of secondary importance. We see that for a given transistor count, while a combination of data cache/secondary cache access times of 1:2 cycles performs the best, systems with access times of 1:4 cycles perform better than those with an access time of 2:2 or 2:4 cycles. How much better the systems with 1 cycle access times for the data cache perform over the systems with a 2 cycle data cache depends on the probability of filling the load delay slots. This can be seen by



Memory Access Time = 25 Cycles
0% Load Delay Slots Filled
75% Load Delay Slots Filled
DC=2,SC=4 cycles
DC=2,SC=2 cycles
DC=1,SC=4 cycles
DC=1,SC=2 cycles
Cycles Per Instruction
Transistor Count (in Millions)
(a)
(b)
Memory Access Time = 75 Cycles
0% Load Delay Slots Filled
75% Load Delay Slots Filled
DC=2,SC=4 cycles
DC=2,SC=2 cycles
DC=1,SC=4 cycles
DC=1,SC=2 cycles
Cycles Per Instruction
Transistor Count (in Millions)
(c)
(d)

Figure 2.

the closer vertical groupings of the curves when comparing Figure 2a with Figure 2b and Figure 2c with Figure 2d. In Figures 2a and 2c, where a load delay slot is never filled, there is an obvious gap between the systems with a single cycle data cache access time and a 2 cycle data cache access time.

The main memory access time has a significant impact on the CPI of the system as can be seen by comparing Figures 2a and 2b with Figures 2c and 2d. In the first of set of curves the CPI ranges from approximately 1.45 to 3.30, whereas in the second set of curves the CPI ranges from approximately 1.18 to 2.15. We chose to look at such large main memory access times because we are assuming that the relative speed of processors vs. memory will continue to increase in the future. As might be expected, the figures show that the second-level cache does a fairly good job of hiding the main memory access times once it gets large enough.

One feature of note on the curves representing systems with more than approximately 6 million transistors is their jagged nature. Since this may not be apparent from the figures due to their small size, we have taken Figure 2b and enlarged it and relabeled it as Figure 3. In Figure 3, one can clearly see the jagged nature of the curves. This lack of smoothness is due to the changes in the relative sizes of the instruction and data caches. The peaks in what one might expect to be smooth curves are generally due to configurations that have too small an instruction cache. For example, from Figure 3 we see that the only system that can be constructed with a transistor budget of 9.61 million transistors is one with an 8K byte instruction cache, an 8K byte data cache and a 128K byte second-level cache. Such a system has an average CPI of 1.26 for a 4-way set associative data cache with an access time of one cycle and a second-level cache with an access time of 2 cycles. By adding approximately 180 thousand transistors, a system with a 4K byte instruction cache and 16K byte data cache can be built. For this second system the CPI climbs to 1.29, a slightly worse performance even though more transistors are being used. However, if the transistor budget can allow an additional 180 thousand transistors (for a total of 9.97 million), a system can be constructed with either a 128K byte second-level cache, an 8K byte instruction cache and a 16K byte data cache, or a 128K byte second-level cache, an 16K byte instruction cache and an 8K byte data cache. These systems have CPI's of 1.26 and 1.25, respectively.

Put another way, an analysis of this section of the graph indicates that an optimal configuration using 9.71 million transistors performs almost as well as the optimal configuration requiring 0.38 million more transistors. If it is possible to implement a chip with the larger transistor count, the designer may want to consider using the slightly smaller configuration, and allocating 0.38 million transistors for some other performance improving feature. There are also the large gaps (between 12 and 15 million transistors, for example) into which no realizable

75% Load Delay Slots Filled, Memory Access Time = 25 Cycles

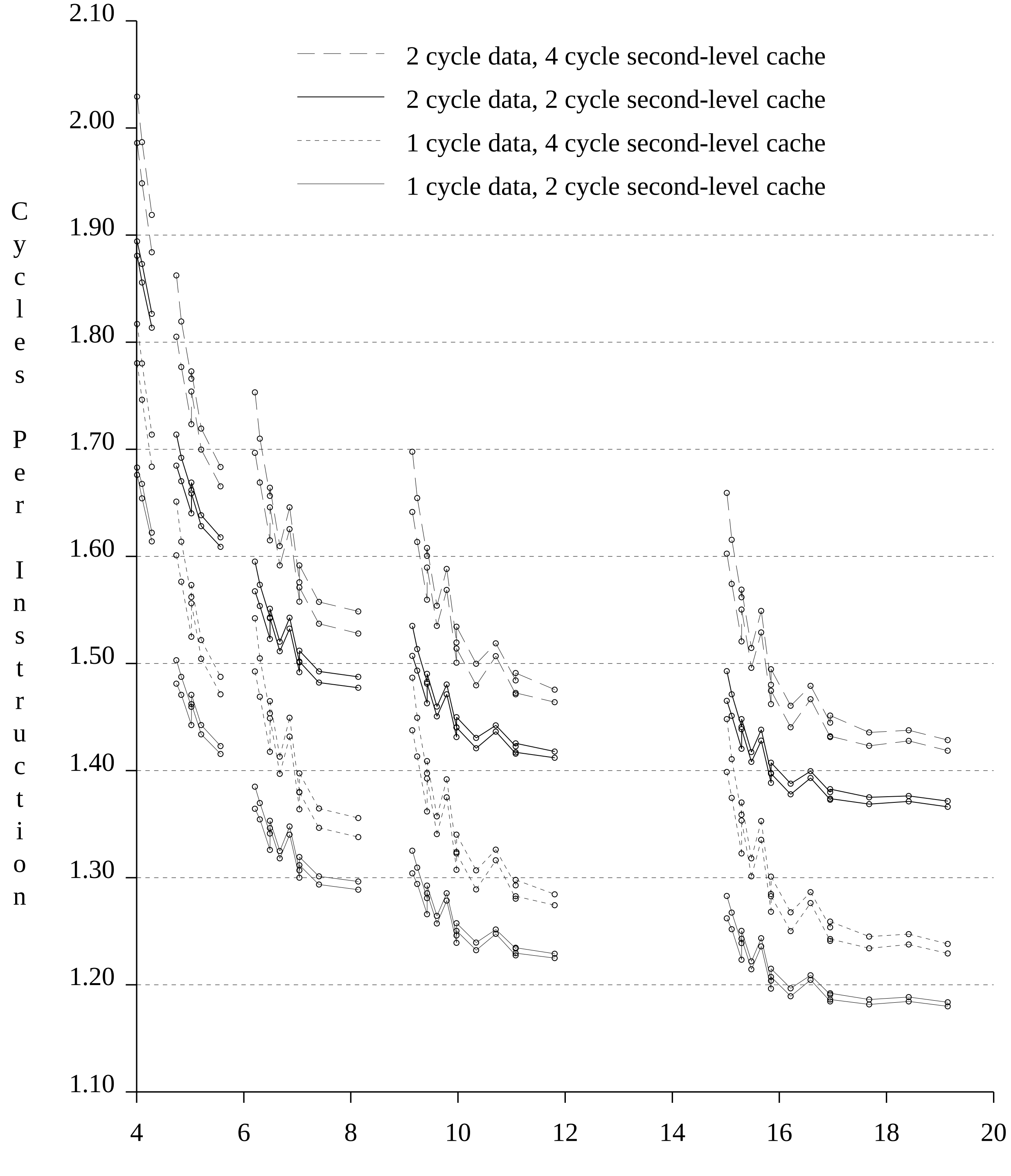


Figure 3.

cache configuration falls. Again, transistor counts that fall into these regions should have transistors available for alternate uses. Such detailed comparisons of other configurations can be made with the help of the data in Appendix A.

If we look at the groups of curves from left to right, as the second-level cache size becomes larger than 64K bytes, some overlap in the performance of the systems becomes apparent. That is, a system with a 128K byte second-level cache may perform better than a system with a 256K byte second-level cache; using another 5 million transistors may not result in a performance improvement. A performance improvement with a larger second-level cache can only be achieved if the first-level caches are made sufficiently large.

6. Multiple Processor Simulation Results

Instead of using available transistors to make larger and larger first and second-level caches, an increase in the number of transistors could be used to place more processors on a die. In a single processor system, after the caches reach a size that captures the working set of the program, further increasing the cache size provides only marginal gains. One might therefore expect a more efficient use of transistors if another processor, with its own caches, were placed on the same chip. As more processors and their first-level caches are placed on the chip, more demands are placed on the second-level cache. The results presented in this section explore the tradeoffs between the number of processors and cache sizes for a given transistor budget.

For these simulations, we have constrained the number of simulations by restricting the first-level data cache to being direct-mapped. As with the single processor results, the instruction cache is single cycle and direct-mapped while the second-level cache is 4-way set associative.

In performing the simulations with multiple processors, we assigned different traces to each of the active processors in the simulation. To avoid skewing the results by simulating traces with similar reference patterns, several permutations of trace assignment were analyzed with average CPI and throughput ratings established. The permutations were selected by analyzing the instruction and data reference patterns and accepting those with 1) the most divergent reference patterns, 2) the most similar reference patterns, and 3) average reference patterns (i.e. in a two processor simulation, the traces would be split into groups of two with each group containing approximately the same number of unique instruction and data references). In Figures 4a and 4b, we plot system throughput versus transistor count. The throughput is computed by taking the total number of instructions executed and dividing by total execution time. For a single processor system, this is equivalent to taking the inverse of the CPI. With a two processor system, ideally one would execute twice as many instructions in the same amount of time as in a

Throughput

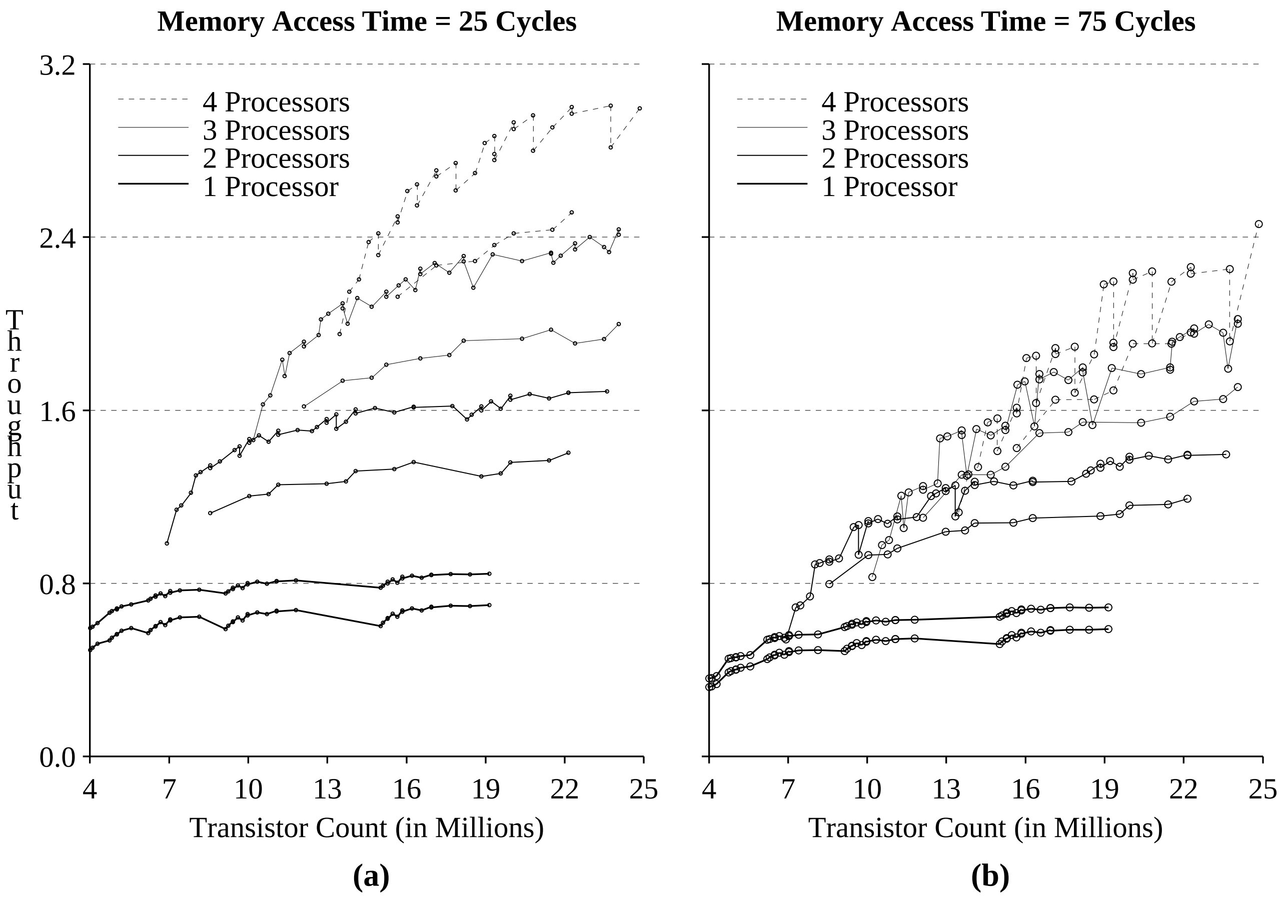


Figure 4.

single processor system; therefore, a two processor system should have twice the throughput of a single processor system. Similarly, an ideal three processor system would have three times the throughput and a four processor system would have four times the throughput of the single processor system.

In Figure 4a we plot the throughput of a system with a 25 cycle main memory, while in Figure 4b the main memory has a 75 cycle access time. For all these simulations, we assume that the load delay slot is filled 75% of the time.

In Figures 4a and 4b, there are four sets of two curves in each figure. For Figure 4b this may not be clear, due to the overlap of performance results for the 3 and 4 processor systems; nonetheless from the bottom, the four sets of curves represent the results for 1, 2, 3 and 4 processor systems. Within a set of curves, the two curves represent the fastest and slowest systems with different data cache and second-level cache access times. From the bottom, the combination of access times for the data cache and the second-level cache are 2:4 and 1:2, thus showing the results for the worst and best case access time combinations. As was seen with the single processor simulations, it is always important to make the first-level data cache as fast as possible.

Figure 4a shows that for a system built with 16 million transistors, a single processor systems offers a best-case throughput of slightly over 0.80. A two processor system offers a throughput of approximately 1.60 for the fastest cache access times, close to the ideal of twice the throughput of the single processor case. Unfortunately, the three and four processor systems do not triple or quadruple the throughput. The inability of the these systems to continue providing improved throughput is due to a second-level cache that is too small. Essentially, each time a processor is added the effective size of the second-level cache is reduced. Two processors each get half of a 128K byte second level cache, whereas in a 4 processor system each processor is only getting 25%. We can see from the figures that a second-level cache size of 32K bytes is too small, so the performance degradation of the 4-processor setup is logically consistent.

If we look at Figure 4b and the throughput for systems with 16 million transistors, the gains in throughput are not nearly as good due to the slow main memory access time. In Figure 4b, while the single processor throughput degrades somewhat compared to that in Figure 4a, the benefits of having 3 and 4 processors are not nearly as great. Limiting oneself to a two processor system, however, still provides close to twice the throughput of the single processor system. Again, this is due to the fact that the second-level cache is not large enough to support the first-level caches of 3 and 4 processor systems. This can be inferred from the figures since the throughput for the 3 and 4 processor systems is increasing while the throughput for the 1 and 2 processor systems has leveled off.

As was the case with a single processor, the jagged nature of the 3 and 4 processor curves reflect the system configurations in which the instruction and data caches are too small to capture the working set of the programs. They must therefore rely on the second-level cache to service their high miss rate.

Figures 5a and 5b plot the average CPI for each processor. The average CPI is not merely the inverse of the throughput. Instead the average of the individual CPIs is found. That is,

$$CPI_{average} = \frac{\sum_{i=1}^{n} CPI_i}{n}$$

Where n is the number of processors.

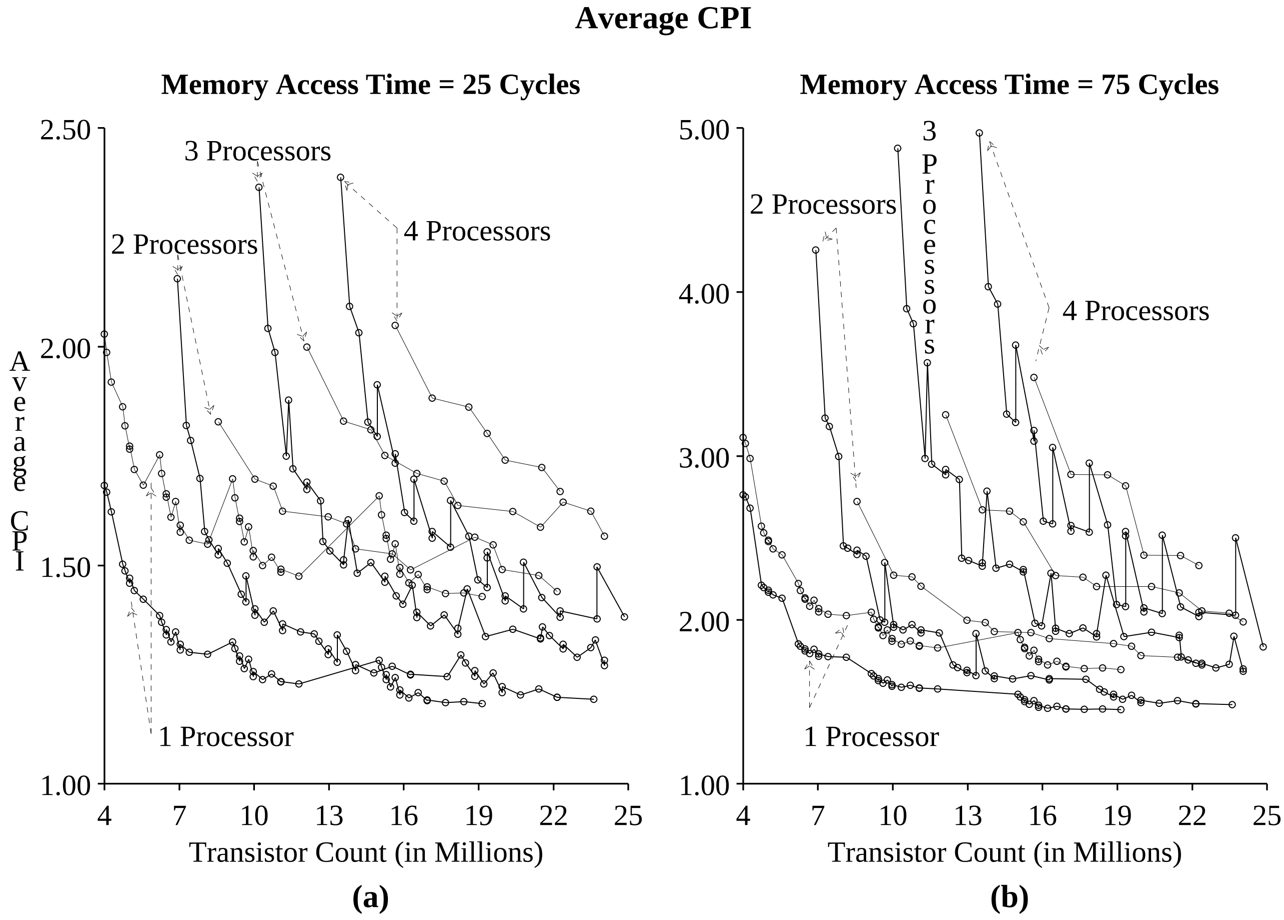


Figure 5

This is a way to normalize the results, since the simulator stops the simulation (in the multi-processor simulations) as soon as one of the traces has completed. Using average CPI also tends to highlight the effects of interference that is caused among processors needing to access the second-level cache to service first-level cache misses. If each processor is making equal progress in executing its program, the average CPI should be equivalent to the inverse of the throughput. On the other hand, if one processor is making slow progress (has a high CPI), the average CPI will be somewhat higher than the inverse of the throughput.

It is important to note that unlike the other graphs presented, the scales are different between figures 5a and 5b. We can still see some similarities, however. These graphs contain 8 curves each, two each for 1, 2, 3 and 4 processor systems. These curves again represent the fastest and slowest access time combinations for the data and second-level caches. If we concentrate on the 1 and 2 processor curves, we see that a 2 processor system with fast cache access times can perform better, in terms of average CPI, than a single processor system with slow cache access times. For the 3 and 4 processor systems, the spaghetti nature of the curves show

how hard it is to reach conclusions based on the average CPI.

The final set of curves shown in Figures 6a and 6b is an attempt to see how well a single instruction stream would run on a multi-processor system. In other words, if only one program is available to run on a multi-processor system for a particular transistor count, it does not have as large an instruction cache, data cache or second-level cache as it would have in a single processor system. The performance of this single stream, even though it is running all by itself, is degraded. The curves in Figures 6a and 6b show the CPI of a single instruction stream. The curve at the bottom is the CPI for a single instruction stream on a single processor. The other curves, for 2, 3 and 4 processors, are almost shifted versions of the single processor curve. The heavier lines represent systems with the fastest cache access times, while the lighter lines represent systems the slowest cache access times. For a main memory access time of 25 cycles, a single stream can run virtually as fast in a multi-processor as in a single processor system once

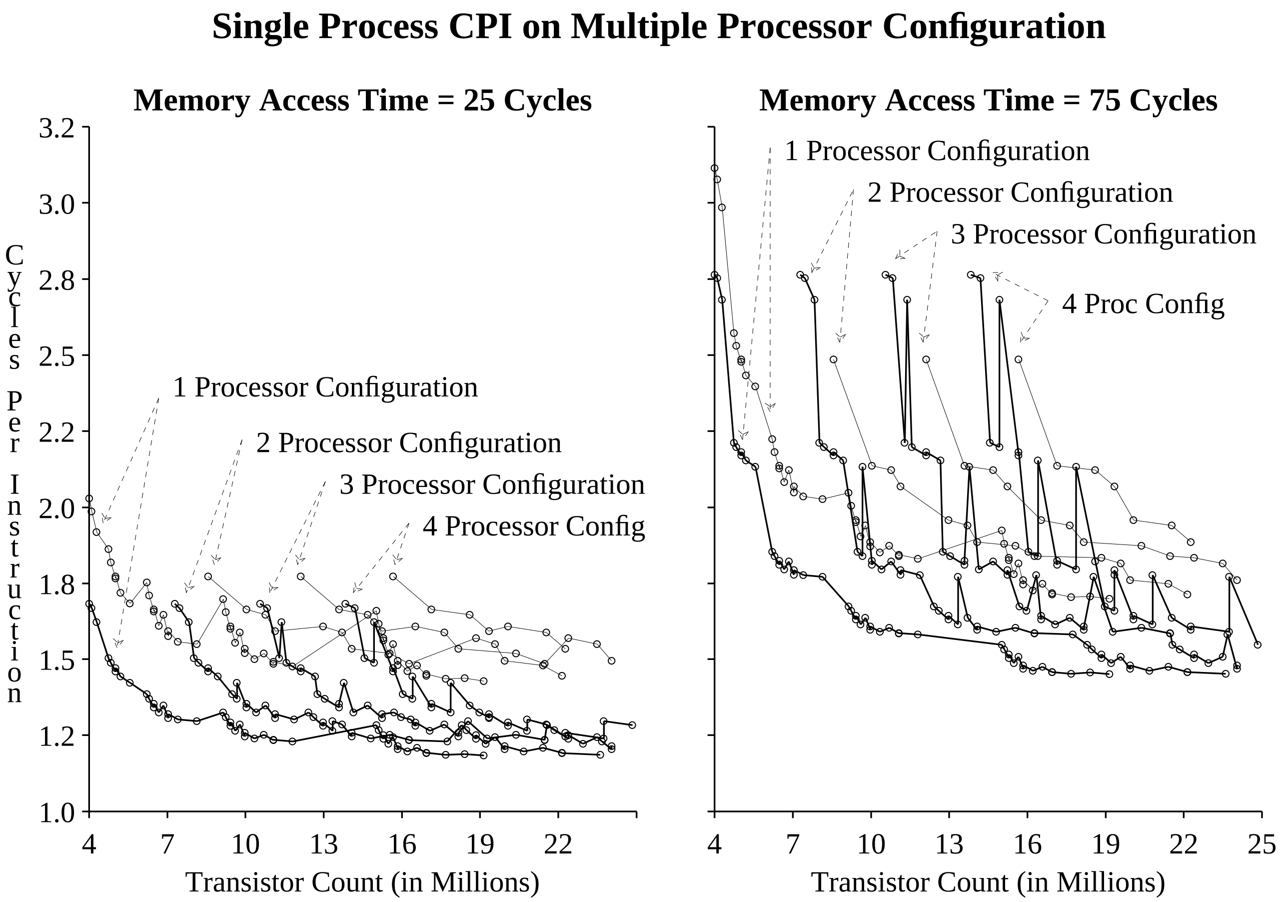


Figure 6.

the second-level cache is made large enough (128K or 256K bytes). With a main memory access time of 75 cycles this does not occur.

7. Conclusions and Future Work

In this paper, we have presented an investigation into how best to allocate many millions of transistors in a single-chip design. For a given transistor budget, we simulated "reasonable" processor and cache organizations and evaluated the performance of these systems in terms of clocks per instruction (CPI). Once we had evaluated single processor systems, we then extended these studies to multi-processor systems. In all simulations, the basic system model consisted of each processor having its own instruction and data cache that were supported by a second-level on-chip cache.

To compare different system organizations, we developed the concept of the equivalent cache transistor (ECT). Because transistors densities for a processor are lower than those found in a cache, we in effect normalized the transistor density for the processor in terms of cache transistor densities. Based on empirical evidence, we found that a high performance processor, such as a DEC Alpha, could be implemented with roughly 3 ECTs (the area used to implement the processor could be used to build a 3 million transistor cache).

Having developed the ECT metric, we used trace-driven simulations to determined the performance of tens of thousands of different processor and cache configurations. In these simulations, we varied cache sizes, cache associativity, cache access times, main memory access times and the number of processors on a single chip. Sixteen different programs were used as benchmark traces, 9 taken from the SPEC89 suite. The results of these simulations were then analyzed with respect to average clocks per instruction or throughput versus transistor count. Based on these results we are able to reach some general conclusions.

First, having a larger transistor budget does not necessarily lead to improved performance. The transistors must be allocated in a proper balance between the instruction and data caches. In particular, it its important to make instruction caches large enough to hold the working set of a program. This is noteworthy since increases in transistor counts do not result in linear increases in available cache sizes - many times the cache configuration realizable for a given increase in transistor count may provide worse performance than that provided by a configuration using fewer transistors, and the increase in transistors should be invested in some alternate hardware.

Second, it is extremely important to have a single cycle access data cache. This may at first seem obvious; however, as the data cache is made larger and larger, it becomes more and more difficult to build a single-cycle access cache. At some point one will be forced to build a 2 cycle data cache. Our simulations show that it is better to build a smaller single cycle data cache than a much larger 2 cycle data cache. This holds true even when 75% of loads do not block the following instruction. Optimizing the speed of the second-level is not as critical. As long as the first-level data cache has a single cycle access time, the the second-level cache may have an access time of 4 cycles without sacrificing a great deal of performance.

For the lower range of transistor counts we investigated (4 to about 8 million) we found that the single processor configurations perform the best. However, as transistor counts climb above 8 million, we found that a 2 processor system offers very good performance per transistor. In fact, if one has a transistor budget of roughly 16 million transistors, a two processor system appears to be an excellent system organization. Our simulation results indicate that one can achieve twice the throughput of a single processor system with this same transistor budget. At the same time, if there is only a single process running on this two processor system, it will run nearly as fast as on a single processor using the same number of transistors. The three and four processor configurations also exhibit improved throughput; however, a point of diminishing returns is quickly reached. This is because the second-level cache becomes effectively too small when shared among three or four processors.

The studies and approach reported in this paper lay the foundation for many more interesting studies that can be performed. For example, what happens when each processor is a superscalar processor, increasing the demands placed on the instruction and data caches? In addition, the results presented here are based on trace driven simulations using sampled traces. It would be of interest to run the simulations on the non-sampled traces to see how the results compare. It may also be possible to develop and validate analytical models based on these studies.

8. References

[DWAA92] D. W. Dobberpuhl, R. T. Witek, R. Allmon, R. Anglin, D. Bertucci, S. Britton, L. Chao, R. A. Conrad, D. E. Dever, B. Gieseke, S. M. N. Hassoun, G. W. Hoeppner, K. Kuchler, M. Ladd, B. M. Leary, L. Madden, E. J. McLellan, D. R. Meyer, J. Montanaro, D. A. Priore, V. Rajagopalan, S. Samudrala and S. Santhanam, ‘‘A 200-MHz 64-b Dual-Issue CMOS Microprocessor’’, *IEEE Journal of Solid-State Circuits,* vol. 27, no. 11 (November 1992).

[Hill87] M. D. Hill, *Aspects of Cache Memory and Instruction Buffer Performance*, Ph.D. Thesis, Department of Computer Sciences,, Berkeley, California, (November 1987).

[John91] M. Johnson, *Superscalar Microprocessor Design*, Prentice Hall, Englewood Cliffs, New Jersey, (1991).

[LaPI88] S. Laha, J. Patel and R. K. Iyer, ‘‘Accurate Low-Cost Methods for Performance Evaluation of Cache Memory Systems’’, *IEEE Transactions on Computers,* vol. 37, no. 11 (November, 1988), pp. 1325-1336.

[Site93] R. L. Sites, ‘‘Alpha AXP Architecture’’, *Communications of the ACM,* vol. 36, no. 2 (February, 1993), pp. 33-44.

[Smit82] J. E. Smith, ‘‘Decoupled Access/Execute Computer Architectures’’, *Proceedings of the Ninth Annual International Symposium on Computer Architecture*, Austin, Texas (April 26-29, 1982), pp. 112-119.

[SmGo85] J. E. Smith and J. R. Goodman, ‘‘Instruction Cache Replacement Policies and Organizations’’, *IEEE Transactions on Computers,* vol. C-34, no. 3 (March 1985), pp. 234-241.

[YeP91] T. Yeh and Y. Patt, ‘‘Two-Level Adaptive Training Branch Prediction’’, *Proceedings of the 24th Annual International Symposium on Microarchitecture*, Albuquerque, New Mexico (November 18-20, 1991), pp. 51-61.