

Performance and Power Impact of Issue-width in Chip-Multiprocessor Cores

Magnus Ekman and Per Stenstrom

{mekman, pers}@ce.chalmers.se

Department of Computer Engineering

Chalmers University of Technology

SE-412 96 Göteborg, Sweden

Abstract

In chip-multiprocessors (CMPs), the number of cores and the issue width of each core presents an important design trade-off to balance the amount of TLP and ILP between multi-threaded and single-threaded applications.

This paper explores the trade-off between issue-width of the cores and the number of cores on a chip by exploring design points with comparable area with respect to both performance and energy. Despite the fact that several simple cores are advantageous for well-behaved parallel applications, e.g. SPLASH-2, we show that these applications can be run as efficiently and with comparable power consumption on a CMP with fewer, but wider-issue cores. In fact, four-issue cores strike a good balance between ILP and TLP. This is attributable to the inherent ILP and the fact that fewer cores result in less performance and power consumption losses in the on-chip memory hierarchy. Thus, our study shows that opting for moderate issue widths may strike a good compromise between single-threaded and multi-threaded applications.

1. Introduction

The difficulty in exploiting more instruction-level parallelism (ILP) along with a shift in the application landscape towards multi-threaded implementations of e.g. database, web, and scientific/engineering applications has led to the emergence of processors that support thread-level parallelism. Chip-multiprocessors [17] use the real-estate to trade off a single wide-issue superscalar processor for many cores with narrower issue width.

An interesting question that has motivated our research is to seek deeper insight into the trade-off between selecting the number of cores versus the issue width of each core. Starting with the arguments for picking many narrow-issue cores, they are expected to provide a better match for multi-threaded applications. In fact, in the Piranha project [2], they opted for single-issue cores acknowledging the hard-to-exploit ILP in database applications [19]. Secondly, they can yield shorter time-to-market, and finally, when it comes to power consumption, which has a growing impact on design decisions [16], simpler cores may have an advantage because of less misspeculation, smaller centralized structures such as register files and power consuming instruction windows [1] [9], as evidenced in previous studies [7].

As for arguments for picking fewer but wider-issue cores, they are a better target for single-threaded applications with exploitable ILP. In addition, for multi-threaded applications that have a decent amount of exploitable ILP, they might run faster on fewer cores as inter-thread communication that goes through the (on-chip) memory system may present more performance and power

losses than running them on the same core where the same communication takes place between registers or through the first-level cache. Recent advances in thread-level data dependence speculation support can make it possible to widen the scope of single-threaded applications that can be parallelized with parallelizing compiler frameworks. However, not many such codes to this date have shown a decent speedup on chip-multiprocessors [18][20]. So, balancing the number of cores with a decent issue-width for each core is a delicate design decision.

Huh *et al.* [11] partly addressed this issue by conducting a study aiming at exploring architectural trade-offs for chip-multiprocessors. Their study focused on the trade-offs between whether the cores should be in-order or out-of-order, the number of cores, and the amount of on-chip cache under a fixed area constraint. Their study was driven by sequential applications. They found that while increasing the number of cores can maximize job throughput, limited pin bandwidth may force designers to use bigger caches. While they acknowledged the importance of having out-of-order cores, their study did not provide insights into the trade-off between issue-width and the number of cores. In addition, they provided no data on multi-threaded applications – a key target for chip-multiprocessors. Zyuban and Kogge [24][25] have studied the optimization of a single superscalar processor core with respect to power. They found that there were significant gains to do by partitioning the core into clusters, but they did not consider multiple cores.

In this paper our goal is to provide insights into selecting the issue width and the number of cores in a chip multiprocessor from both an application performance as well as power consumption perspective. As a base for our study, we use multi-threaded applications from the SPLASH-2 benchmark suite [23]. These applications have been tuned to exhibit a linear speedup in the range of up to sixteen cores. In addition, significant efforts have been invested to exploit locality which would de-emphasize the potential disadvantage that increasing the number of cores would lead to performance losses in the on-chip memory system. Thus, one would expect them to benefit from many simple cores.

Our study is based on a detailed simulation model of a chip-multiprocessor in which micro-architectural features such as issue-width, instruction window size, and number of functional units can be parameterized apart from cache hierarchy organizational parameters. We have used SimWattch [6] which utilizes the complete-system simulation support of Simics [14] with the accurate performance and power modeling features of Wattch [3] and SimpleScalar [5]. We adopt a scaling methodology in which we compare the performance and power consumption of a number of design points consuming about the same chip resources in which we change the number of cores and their individual issue width. Based on this, we can build intuition into which factors are important in order to balance the inherent thread-level and instruction-level parallelism in multi-threaded workloads such as SPLASH-2.

While one would expect that the applications from SPLASH-2 would benefit from as many cores as possible, we actually find that as few as four four-issue cores perform almost as well as sixteen single-issue cores. The main reason for this is that SPLASH-2 has a fair amount of instruction-level parallelism so less thread-level parallelism can be traded for more instruction-level parallelism. In addition, even if the performance losses in the memory system are quite modest, they are enough to speak in favor of fewer cores where much of the inherent and artificial communication can be carried out at the first-level caches. Finally, while the system with few and wider-issue cores will spend more power in the cores, they will spend less power in the on-chip memory hierarchy. The net result is that power consumption does not vary much across the design points. Overall, our data suggests that opting for moderate issue widths, say four, strikes a good compromise to exhibit a high performance across single- and multi-threaded applications.

As for the rest of the paper, we establish the architectural framework and the scaling methodology in the next section. We then introduce the experimental methodology in Section 3 followed by the experimental results in Section 4. We end the paper by presenting our conclusions in Section 5.

2. Architectural Framework & Scaling Methodology

This section first presents the architectural framework in Section 2.1 and then introduces the scaling methodology in Section 2.2.

2.1 Architectural Framework

We consider a chip-multiprocessor consisting of a number of multiple-issue processor cores, each associated with a private L1 instruction and data cache that interface to a shared L2 cache via a shared internal bus according to Figure 1(A). The L1 data caches are lockup-free and are kept coherent with a MOESI snoopy cache protocol.

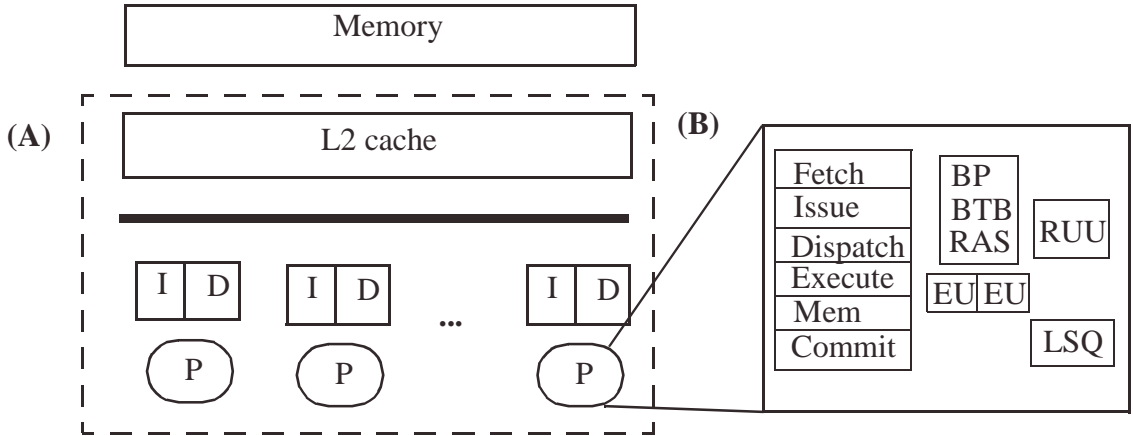


FIGURE 1. Chip-multiprocessor (A) and core (B) architecture.

Each core is an out-of-order multiple issue processor pipeline, modeled according to the pipeline organization ‘sim-outorder’ from SimpleScalar [5]. The pipeline and its functional units are displayed in Figure 1(B). The fetch stage includes a gshare branch predictor (BP) with two-bit counters, and the targets are predicted with a branch-target-buffer (BTB). Additionally, a return address stack (RAS) predicts return addresses from function calls. Register renaming and reordering of instructions are done with a register update unit (RUU). Loads and stores are tracked by the load/store-queue (LSQ). Finally, the instructions are executed by the different execution units (EU). More details regarding the experimental assumptions are provided in Section 4.

This chip-multiprocessor architectural framework enables us to study architectures similar to most of the recently proposed organizations. The first proposed chip-multiprocessor was the Hydra from Stanford [17]. It consists of four single-issue cores and also has support for thread-level speculation. The Piranha chip multiprocessor proposal [2] consists of eight single-issue cores and is optimized for OLTP-like workloads which are characterized as containing very little ILP but have an abundance of TLP. The IBM Power4 [21] consists of two very aggressive eight-issue cores with

a sustained completion rate of five instructions per cycle. Finally, the MAJC [22] consists of two cores executing VLIW-instructions that can contain up to four instructions in each word.

TABLE 1. Different existent and proposed CMPs.

CMP	Cores	Issue-width
Hydra	4	1
Piranha	8	1
Power4	2	8
MAJC	2	4

Table 1 shows the number of cores and issue-width of these very different designs which were suggested in different time frames and for different market segments. We note that while some of these implementations favor many simple cores, others opt for a few cores supporting as many as eight instructions to be issued each clock cycle. This gives concrete evidence as to the difficulty of balancing the issue-width and number of cores. In order to study this trade-off, we have used a scaling methodology which is described in the next section.

2.2 Scaling Methodology

We consider four design points that differ in the number of cores and their issue widths ranging from sixteen single-issue cores to a processor consisting of two eight-issue cores. In between these extreme points we also study eight-way dual-issue and four-way four-issue cores. All designs use out-of-order cores as advocated in [11]. As we scale the cores from a single-issue to an eight-issue processor, we adjust the following architectural resources inside each core, with the goal that they consume about the same amount of real estate on chip.

- The instruction window
- The load/store queue
- The branch prediction mechanism
- Number and type of functional units
- L1 data caches
- L1 instruction caches

We scale all these parameters linearly. For instance, if we double the number of cores we decrease the size of the structures by a factor of two. We also scale the number of execution units. However, the total number of execution units differ in the various systems. The reason for this is that in the systems with wider issue cores, it is possible to utilize the execution units better. For example, in an eight-issue core there is hardly any meaning having eight floating-point dividers, but each single-issue core needs at least a single floating-point divider to be able to execute a floating-point division. Therefore the number of execution units tends to increase as we move towards a CMP with more cores.

The systems all have a shared L2-cache, and private separate instruction and data caches. The caches are write-allocate and copy-back and are kept coherent by a snoop-based MOESI protocol. The L2-caches in the different systems all have the same size. For the private caches we have assumed a fixed size (in kilobytes) which is equally divided among the processors, that is, the cores in the 16-way CMP have caches that are eight times smaller than in the two-way CMP.

It can be argued that the increase in area when going from single issue to dual issue is not a factor of two. For example, all cores need a set of registers in order to hold the context, while only a part of the register file logic must be replicated in order to increase the issue width. Similarly, all the execution units don't have to be replicated. Partly, this is also true when going from a dual-issue to a four-issue core even though adding more ports to the shared structures (issue-window, load/store queue and register file) will increase the area of them more than linearly with respect to their increase in size. In order to go from four-issue to eight-issue, however, some structures will need to be doubled since it is not realistic with that many ports. A decent estimate is thus that an eight-issue core should occupy about twice as much area as a four-issue core

It can thus be argued that a more fair comparison would be to e.g. consider systems with two, four, seven, and thirteen cores. However, our benchmarks can only be run on systems where the number of cores is a power of two. Therefore we choose two, four, eight and sixteen cores in the systems that we evaluate. Note, however, that we give a performance advantage to the systems with many cores. As we will show, this will strengthen the main observations in the quantitative analysis.

In this study we have considered rather basic systems without many power optimizations. We use clock-gating for the structures that are not used each cycle but we don't use any of the techniques to reduce the power consumption, that have been proposed for the past few years, for example dynamic instruction window resizing [1][9] and leakage reduction techniques for caches [8][12]. By this, we don't mean that they should not be used if these systems were to be built. Rather, we find these techniques irrelevant to our comparison as they can be orthogonally applied to each design point. It should be noted however, that which techniques to apply highly depends on the design point. For the wide-issue cores one should aim for micro-architectural techniques such as dynamic window resizing, while for the multiple-core systems one should aim for techniques that address power dissipation in the on-chip memory system, for example inefficiencies in the cache coherence protocol [4].

3. Simulation Set-up, Architectural Assumptions & Workloads

We first discuss the simulation infrastructure we have developed to conduct this study. Then we present in detail all the assumptions we have made as far as architectural performance parameters and workloads.

3.1 Simulation Infrastructure

We have used a recently developed simulation tool – SimWattch [6] – to implement a simulation model of a chip-multiprocessor. This tool is built around Simics [14], a complete-system simulation infrastructure that can functionally model the execution of a program on top of a single-issue processor that runs a commodity operating system (e.g. Solaris). SimWattch integrates Simics with Wattch [3] to enable detailed cycle-level performance and power consumption modeling of micro-architectural features such as the impact of issue width which is at the core of this study.

The approach taken in SimWattch to integrate Simics with Wattch is to let Simics functionally simulate the execution of the program. Each executed instruction is fed into a FIFO buffer, called Instruction History Queue (IHQ), along with its memory-bound operands. The IHQ is consumed by Wattch by letting it fetch instructions as well as memory-bound operands from it. As long as there is no misspeculation, Simics executes ahead of Wattch. However, when Wattch executes along a mispredicted path, Simics is stalled, and Wattch will pick instructions as well as operands

from the memory rather than from the IHQ aided by the built-in support in Simics to do virtual-to-physical address translation and accesses to the physical memory.

SimWattch was originally developed to model a single processor. We have modified its implementation to model a chip-multiprocessor. Apart from implementing the on-chip cache hierarchy along with an intra-chip cache coherence protocol, we faced some non-trivial issues.

Because of the functional-first approach taken by SimWattch, there is a lag between the time at which an instruction is completed in Simics and the time at which it is committed in Wattch. For example, if a load instruction experiences a miss, the stall for that load will be experienced several cycles after it was committed by Simics. This may affect the global ordering of events in the multiprocessor as discussed in [15]. While we have not yet explored the absolute performance errors that this may cause, it can be argued that it will not systematically give neither a performance advantage nor a disadvantage to a system with more cores than another. Thus, we feel that this effect should not affect the relative performance much between the simulated systems.

Another issue is that it is not possible to control how many instructions Simics issue each cycle. We have solved this by assuming that we want a certain issue width, e.g. four. We then buffer four instructions in an intermediate structure between Simics and Wattch. These instructions are then fed into Wattch and increase our internal clock cycle counter by one. However, this creates a subtle error. Simics believes that four cycles, in spite of one, have elapsed. As a result, the time-triggered interrupt which invokes the operating system happens four times as often as it should. This is solved by assuming that the simulated cores run at a higher frequency. For example, by doubling the frequency when the issue width is doubled, the user program is interrupted by the operating system after the same wall clock time.

3.2 Architectural Model and Workload Assumptions

We have used the cc3 power model from Wattch. This implies aggressive clock gating and static power consumption as follows: If a multiported unit is used during a clock cycle its maximal power consumption is linearly scaled according to how many of the ports that are used. If a unit is not used at all during a cycle it consumes 10% of its maximal power consumption as static power.

In [17], Olukotun *et al.* argue that a simpler core should be possible to clock faster. While this may have a significant effect for cores with an issue-width much bigger than we see today, there is no evidence that it is difficult to scale up the frequency for issue widths up to four. Therefore, we have assumed that all our design points run at the same frequency. All numbers shown are for the 0.18 process in Wattch. Based on the architectural model described in Section 2, we assume the architectural parameters in Table 2 for the design points we study. Apart from the latency for accessing the L2-cache (or foreign L1-cache in the case of cache-to-cache transfers) on the L1-misses, there is also a latency for bus arbitration which depends on the current load of the on-chip memory system.

TABLE 2. Baseline parameters for the systems

	2	4	8	16
Number of cores	2	4	8	16
Fetch/Issue/Commit width	8	4	2	1
Integer ALU	8	4	2	1
Integer Mult/Div	4	2	1	1
FP ALU	8	4	2	1
FP Mult/Div	4	2	1	1
Instr. window (RUU)	128	64	32	16
LSQ Entries	64	32	16	8
Fetch Queue	8	4	2	1
Mispred. Penalty	7	7	7	7
L1 I-Cache	64K, 2-way	32K, 2-way	16K, 2-way	8K, 2-way
L1 D-Cache	64K, 4-way	32K, 4-way	16K, 4-way	8K, 4-way
L2 Cache	2M, 8-way, 12 cycles	2M, 8-way, 12 cycles	2M, 8-way, 12 cycles	2M, 8-way, 12 cycles
Memory	128 cycles	128 cycles	128 cycles	128 cycles
Branch Pred.	16 K-entries	8 K-entries	4 K-entries	2 K-entries
BTB	4 K-entries	2 K-entries	1 K-entries	512 entries

On top of the simulator we run Sun Solaris 5.8. We boot this without the architectural model since this would be a very time consuming assuming the detailed micro-architectural simulator. We have evaluated our systems with applications and kernels from SPLASH-2 [23]. The benchmarks are run with the default input data sets which are listed in Table 3. The benchmarks are all compiled with optimization turned on. The statistics are collected during the parallel section which is run to its end.

TABLE 3. Benchmarks and data sets

Benchmark	Input data set
FFT	256K integers
Radix	1M keys
Water-sp	512 molecules
Raytrace	Car
Ocean	258x258
Cholesky	tk.15.O

4. Results

In this section we present our simulated results. Section 4.1 describes the performance results. Thereafter the power and energy effects are studied in Section 4.2.

4.1 Performance Results

The execution times for the different benchmarks are plotted in Figure 2 (left). It is normalized to the execution time for the 2-way system. Since the SPLASH-2 applications have been shown to scale linearly with the number of processors, at least on the scale we consider, one would expect that execution time drops a factor of two as we double the number of cores. While this happens when we move from two to four cores, performance seems to flatten out and even deteriorate as we move to more cores. We next consider the underlying reasons for this key observation by considering in detail what happens when we double the number of cores.

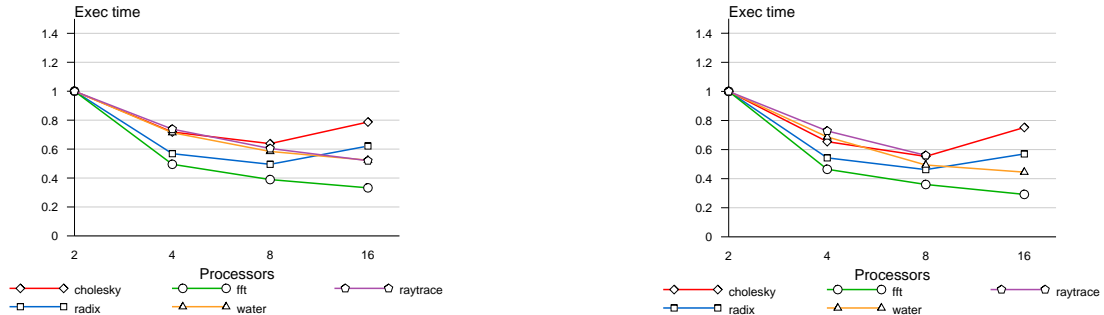


FIGURE 2. Normalized execution time for baseline system (left). Normalized execution time for a system with longer memory latency (right).

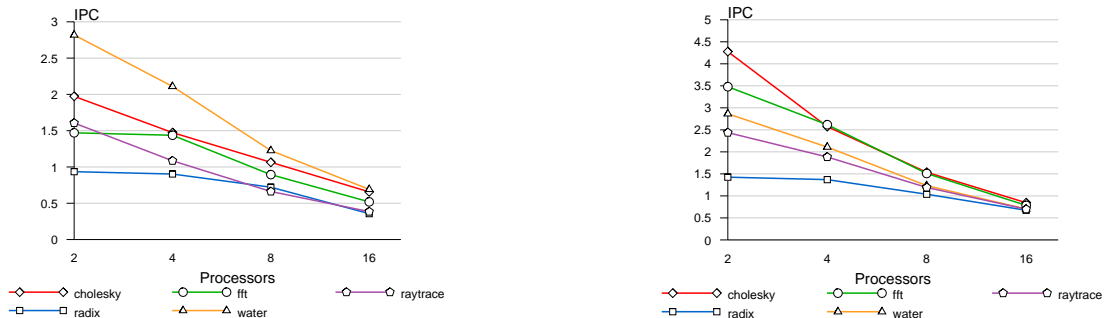


FIGURE 3. Instructions per cycle for baseline system (left). Instructions per cycle with a perfect memory system (right).

4.1.1 2-Way to 4-Way Performance Trade-off

First, when moving from a 2-way to a four-way system, the complexity of the cores is reduced from eight-issue to four-issue. As already said, we would expect the execution time to decrease by a factor of two if the issue width was left unchanged. If we consider FFT and Radix, this is indeed what happens. However, the rest of the benchmarks (Raytrace, Water and Cholesky) don't show the same speedup. The explanation of this behavior is that the four-issue cores cannot exploit the same amount of ILP as the eight-issue cores. This is confirmed by Figure 3 (left) which shows the IPC-rate for each application. We see that for FFT and Radix, both the eight-issue and the four-issue core exploit the same amount of ILP, while the other benchmarks lose a fair amount of exploited ILP, which explains the difference in speedup for the different benchmarks.

To get more insight into the available ILP in the applications we have studied them with a perfect memory system. This is shown in Figure 3 (right) which shows the IPC-rate. The only benchmark that is not affected by this is Water. All the other benchmarks are affected but in different ways. The absolute value of IPC for Radix is changed but the exploited ILP for the two-core system still equals the exploited ILP for the four-core system. For FFT on the other hand, the eight-issue cores would exploit more ILP than the four-issue cores. This would shift the performance curve a little but it is still clear that the system with four four-issue cores is a better choice than the two-core system for the studied benchmarks.

4.1.2 4-Way to 8-Way Performance Trade-off

As we go from a 4-way to an 8-way system, the performance difference is not as big as in the previous region according to Figure 2 (left). Both FFT and Radix levels off radically, since they as well as the other benchmarks lose a fair amount of IPC in this region according to Figure 3 (left). The eight-core system performs about 20% better than the four-core system. Remember, however, that as we mentioned in Section 2.2 we tend to favor the systems with eight and 16 cores area-wise. If the system with eight cores just had seven cores, the two systems would perform at about the same level.

4.1.3 8-Way to 16-Way Performance Trade-off

In the last region, Radix and Cholesky lose performance, while the other ones gain some performance. We see that all of the benchmarks lose a great deal of IPC in this region but there are unexpected effects. The IPC for Radix decreases by a factor of two. At the same time, the number of processors is doubled. While one would expect a small performance change, we clearly notice a performance loss. The same is true for Cholesky. The reason is that neither of these two benchmarks scale perfectly with the number of processors when we reach 16 processors. Serial regions in the benchmarks make some of the processors being idle, executing useless instructions, which leads to that the total number of instructions executed is greater for the 16-core system.

4.1.4 Variations of Memory Latency

The numbers we have shown assume a memory latency (128 cycles) that may not be a good estimate for the systems we build in the future. As we have seen before, the exploited amount of ILP depends on the memory system. To make sure that the results in the previous sections do not depend on this we have studied the performance for a system with three times bigger memory latency. The amount of parallelism that the processor can uncover can depend on this.

The results are shown in Figure 2 (right). At first it seems like the results are exactly the same as for a shorter memory latency, but the graphs actually differ a little bit. Remember that the execution times are normalized so one can only compare absolute performance within each diagram. Table 4 shows the relative slowdown between the two-core system with long memory latency and the baseline system. The memory latency clearly affects performance, but it does not seem to affect the relative performance between the number of cores and their issue-width.

In summary, we have shown that while intuition says that multi-threaded applications would benefit from increasing the number of cores, our data actually shows effects that can counter this expectation. First, as we move to more but narrower issue-width cores, the amount of ILP to be exploited is reduced. Second, register- and L1-cache-level communication will be converted into

communication across L1-caches. Both these effects tend to offset the gains of having a larger number of cores.

TABLE 4. Relative slowdown between the systems with different memory latency.

Benchmark	Increased execution time
Cholesky	114%
Radix	112%
FFT	103%
Water	61%
Raytrace	94%

4.2 Power and Energy Consumption

4.2.1 Power Consumption Results

Power and energy issues are becoming as important to consider as performance. Power consumption is important mostly for cooling and package consideration while energy today is mostly a concern for battery operated devices. However, within a not too distant future, the energy issue might also be a constraint in desktop and server systems due to the energy cost. The power consumption for the systems for all the benchmarks is shown in Figure 4.

It is normalized against the power consumption for the two-core system. Interestingly, the power consumption is rather constant across the systems, even though the power distribution within the systems is changed. In the wide-issue system, more power is consumed within the processor core, while the memory system consumes more in the system with more cores. In short we can see the following when moving towards a system with more cores:

- L2-Cache power increases
- L1-ICache power increases
- Result bus, Window and LSQ power decreases
- ALU power increases

The increase in L2 cache power has two reasons. Firstly, as we go towards a system with more cores, the performance is increased. However, if the number of accesses to the L2 cache is constant the power consumption will be increased since these accesses are performed in a shorter amount of time. This is seen if we compare the systems with two and four cores for FFT. As mentioned in previous sections, the execution time decreased with a factor of two in this region. As we see in the graph, this causes the power consumption to increase by a power of two. An unexpected effect is seen in the region between 8 and 16 cores however. We don't increase performance as much as power consumption. The reason here is the number of accesses to the L2 cache. There are 56% more accesses in the 16-core system than in the 8-core system. The rest of the benchmarks follow the same trend, but the amount of the total power consumption depends on if there are few or many accesses to the L2 cache.

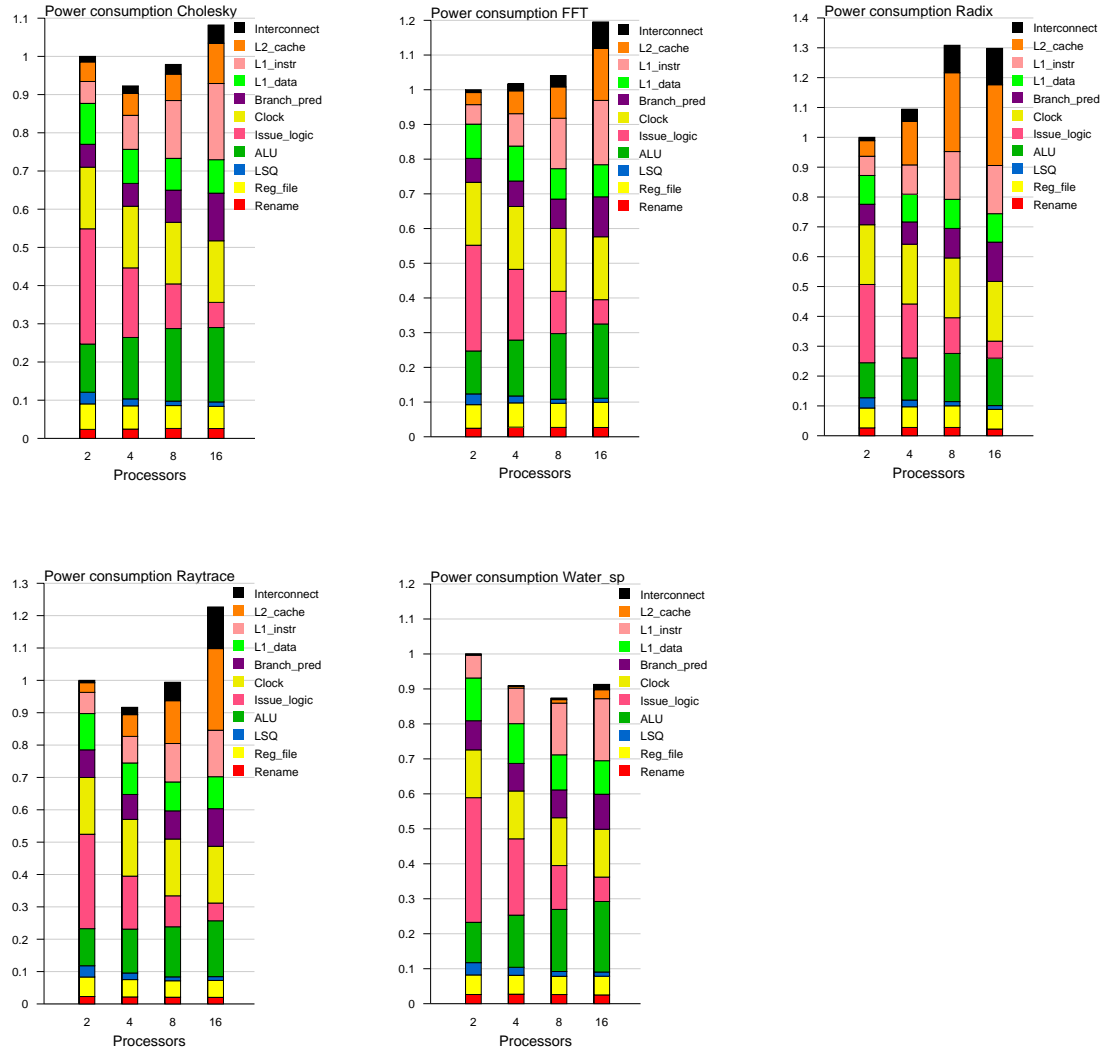


FIGURE 4. Power consumption for the different systems for all benchmarks.

The trend for the instruction caches is a complex mix of four effects. Firstly, we have the same effect as for the L2 cache, that the execution time is shorter for the systems with more cores and therefore the power consumption is bigger. Secondly, the accesses to the instruction caches tend to have a higher spatial locality than accesses to data caches. This means that in a wide-issue core, one access to a word line in the cache can supply several words, while a narrow-issue core gets fewer words per access. This leads to an increase in the total number of accesses to the instruction caches. This could be optimized by reading the line into a buffer and then read words from this buffer in the following cycles, but we have not assumed that in our calculations. These two effects both increase the power consumption for the systems with more cores.

The third effect counteracts these two. As the sizes of the caches are scaled we get a different banking of the caches. It turns out that the chosen banking for the systems with more cores makes each access cheaper energy-wise. The final effect is the number of speculative instructions that also favors the systems with many cores. The system with few cores executes more speculative instructions that never are committed than the other systems. This is shown in Figure 5 (left). For Raytrace, the two-core system executes 20% more instructions than the four-core system. In

Figure 5 (right), the total number of committed instructions is shown. We see here that neither Cholesky, nor Radix scale very well with the number of cores.

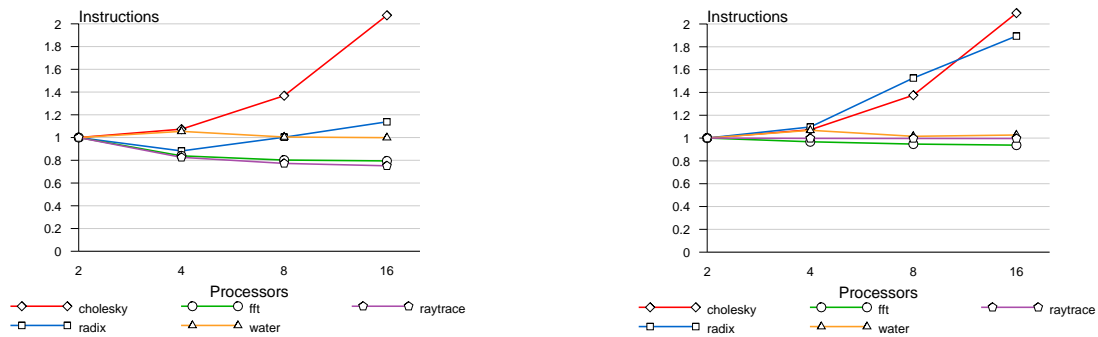


FIGURE 5. Normalized number of executed instructions (left). Normalized number of committed instructions (right).

The data cache shows a different behavior. The power consumption is almost constant across the systems. All the effects for the instruction cache, except for the contiguous accesses, also apply to the data caches. Since the systems with fewer cores don't have the advantage of reading contiguous blocks virtually for free, the power consumption does not differ very much.

The power consumed by the issue-logic in each core clearly decreases with the number of cores. This is because power consumption scales super-linearly with issue width. The number of comparators for the issue-window, for example, is proportional to both the issue width and the window size. This leads to that even if the total number of entries in all the instruction windows of the cores on the chip is kept constant, the power consumption is still lower in the systems with simple cores.

Finally, the power consumption in the execution units (ALU in the graph) increases with the number of cores. The reason behind this is mostly that the execution time is shorter. Overall, however, we see that there is not much difference in power consumption across the different configurations.

4.2.2 Energy Consumption Results

Figure 6 shows the energy consumption results of all benchmarks. The bars are normalized to the two-core system. We see the combination of the power consumption from Figure 4 and the execution time from Figure 2 (left). Since the power consumption is rather constant for the different systems (except for 16 processors, and for Radix 8 processors), the energy will be proportional to the execution time for the three first bars. The data suggests that an eight-core system is the most energy-effective one.

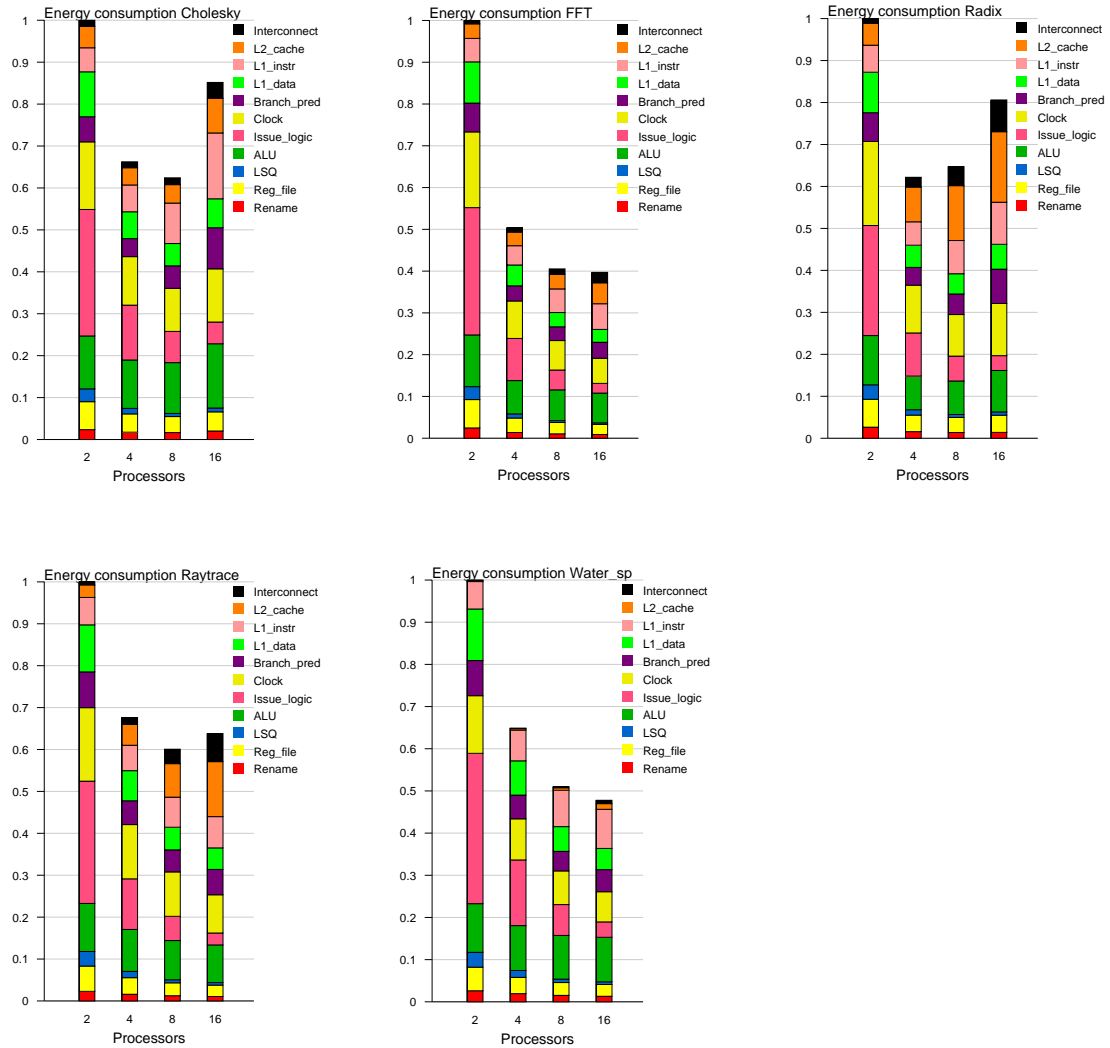


FIGURE 6. Energy results for all benchmarks.

5. Conclusion

We have studied four different systems consisting of two eight-issue cores, four four-issue cores, eight dual-issue cores and 16 single-issue cores. Our scaling methodology that aim at comparing design points with comparable chip resources, however, gives the narrow-width systems a slight performance advantage as they would consume more chip resources.

Our results show that we get best performance with the eight-core and 16-core systems. However, the performance difference between these systems and the four core system is fairly small, especially if we take into account that these systems require more area than the four core system. With only slightly fewer cores on the two systems with the highest number of cores, these three systems would give about the same performance. With respect to power we see that the four-core system often seems to be best, but the eight-core and 16-core systems would consume less power if they would have only slightly fewer cores.

The applications studied have an abundance of thread-level parallelism and thus would favor a system with highest number cores. Despite this, we have found the following:

- A system built from four four-issue cores performs equally well as the eight-core and 16-core systems, while a system built from two eight-issue cores performs worse.
- All the systems that we have studied have approximately the same power requirements.

Given this, a natural choice would be to build a system consisting of four four-issue cores, since this will also give good single-thread performance.

This paper also provides insights into the relative power consumption in trading issue width with the number of cores. We found the following trends as we increase the number of cores and reduce each core's issue width:

- The power consumption in the on-chip memory system is increased.
- The power consumption in the cores is decreased.
- These above two changes offset each other leading to that the total power requirements remain constant.

Overall, this study shows that even if a chip-multiprocessor is optimized for multi-threaded applications, thread-level parallelism can in many cases be traded for instruction-level parallelism. Therefore, it appears that opting for moderate issue widths for each core provides a robust platform for single-threaded as well as multi threaded applications.

Acknowledgement

This research has been supported by a grant from the Swedish Foundation for Strategic Research under the ARTES/PAMP program. Additionally, we are grateful to Fredrik Dahlgren at Ericsson Mobile Platforms for valuable input and ideas for this study. Finally, thanks to Jianwei Chen for providing the SimWattch environment.

References

- [1] R. I. Bahar, S. Manne: Power and energy reduction via pipeline balancing. *International Symposium on Computer Architecture, 2001*, 218-229
- [2] L. Barosso, K. Gharachorloo, R. McNamara, A. Nowatzky, S. Qadeer, B. Sano, S. Smith, R. Stets, B. Verghese. Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing. In *Proc. of International Symposium on Computer Architecture*, pages 282-293, 2000.
- [3] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In *Proc. of International Symposium on Computer Architecture*. pages 83-94, June, 2000.
- [4] M. Ekman, F. Dahlgren and P. Stenstrom. TLB and Snoop Energy-Reduction using Virtual Caches in Low-Power Chip Multiprocessors, In *Proceedings of International Symposium on Low Power Electronics and Design*, August 2002.
- [5] D. Burger and T. Austin. *The SimpleScalar Tool Set Ver. 2.0*. University of Wisconsin-Madison, Computer Sciences Department, Technical Report #1342, 1997.
- [6] J. Chen, M. Dubois, and P. Stenstrom: SimWattch: An Approach to Integrate Complete-System with User-Level Performance/Power Simulators. In *Proc of IEEE ISPASS-2003*, March 2003.

- [7] R. Gonzales and M. Horowitz. Energy Dissipation In General Purpose Microprocessors. *IEEE Journal of Solid-State Circuits*, pages 1277-1284, September 1996.
- [8] K. Flautner, N. Kim, S. Martin, D. Blaauw, T. Mudge. Drowsy Caches: Simple Techniques for Reducing Leakage Power. *International Symposium on Computer Architecture*, 2002.
- [9] D. Folegnani and A. Gonzales. Energy-Effective Issue Logic. In *Proc. of International Symposium on Computer Architecture*, 2001, pages 230-239.
- [10] L. Hammond, B. Hubbert, M. Siu, M. Prabhu, M. Chen, K. Olukotun. The Stanford Hydra. In *IEEE Micro*, pages 71-84, 2000.
- [11] J. Huh, S. W. Keckler and D. Burger. Exploring the Design Space of Future CMPs. In *Proc. of International Conference on Parallel Architectures and Compilation Techniques*, 2001.
- [12] S. Kaxiras, Z. Hu and M. Martonosi: Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power. In *Proc. of International Symposium on Computer Architecture*, 2001, pages 240-251.
- [13] J. Lo, L. Barosso, S. Eggers, K. Gharachorloo, H. Levy, and S. Parekh. An Analysis of Database Workload Performance on Simultaneous Multithreaded Processors. In *25th Annual International Symposium on Computer Architecture*, pages 94-105, 1998.
- [14] P. S. Magnusson, F. Dahlgren, H. Grahm, M. Karlsson, F. Larsson, F. Lundholm, A. Moestedt, J. Nilsson, P. Stenstrom and B. Werner. SimICS/sun4m: A Virtual Workstation. In *Proc. Usenix Annual Technical Conference*, Jun. 1998, pp. 119-130.
- [15] C. Mauer, M. Hill, and D. Wood. Full System Timing-First Simulation. In *Proc. of SIGMETRICS*, June 2002.
- [16] T. Mudge. Power: A first class design constraint. In *Computer*, vol. 34, no. 4, pages 52-57, April 2001.
- [17] K. Olukotun, B. Nayfeh, L. Hammond, K. Wilson, and K.-Y. Chang. The Case for a Single-Chip Multiprocessor. In *Proc. of 7th International Conference on Architectural Support for Programming Languages and Operating Systems*. 1996.
- [18] K. Olukotun, L. Hammond, and M. Willey. Improving the Performance of Speculatively Parallel Applications on the HYDRA CMP. In *Proc. of the 1999 Int. Conf on Supercomputing*, pages 21-30, June 1999.
- [19] P. Ranganathan, K. Gharachorloo, S. Adve, and L. Barosso. Performance of Database Workloads on Shared-memory Systems with Out-of-Order Processors. In *Proc. of 8th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 307-318, Oct. 1998.
- [20] J. Steffan, C. Colohan, A. Zhai, and T. Mowry. The Potential for Using Thread-Level Data Speculation to Facilitate Automatic Parallelization. In *Proc. of the 8th Int. Symp. on High-Performance Computer Architecture*, pages 2-13. Feb 2002.
- [21] J. M. Tendler, J. S. Dodson, J. S. Fields Jr, H. Le and B. Sinharoy. POWER4 System Microarchitecture. *IBM Journal of Research and Development*, Vol. 46 No. 1, January 2002.
- [22] M. Tremblay, J. Chen, S. Chaudry, A. Conigliaro, and S-S Tse. The MAJC Architecture: A Synthesis of Parallelism and Scalability. In *IEEE Micro*, pages 12-25, 2000.
- [23] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proc. of the 22th International Symposium on Computer Architecture*, pages 24-36, June 1995.
- [24] V. Zyuban and P. Kogge. Optimization of High-Performance Superscalar Architectures for Energy Efficiency. In *Proc. of International Symposium on Low Power Electronics and Design*, August 2000.
- [25] V. Zyuban and P. Kogge. Inherently Lower-Power High-Performance Superscalar Architectures. *IEEE Transactions on Computers*, Volume: 50, Issue: 3, March 2001.