International Journal of Parallel Programming, Vol. 34, No. 3, June 2006 (© 2006)
DOI: 10.1007/s10766-006-0016-z

A Case for Chip Multiprocessors based on the Data-Driven Multithreading Model

Pedro Trancoso,[1,3] Paraskevas Evripidou,[1] Kyriakos Stavrou,[1] and Costas Kyriacou[2]

Received May 10, 2006; accepted June 2, 2006

Current high-end microprocessors achieve high performance as a result of adding more features and therefore increasing complexity. This paper makes the case for a Chip-Multiprocessor based on the Data-Driven Multithreading (DDM-CMP) execution model in order to overcome the limitations of current design trends. Data-Driven Multithreading (DDM) is a multithreading model that effectively hides the communication delay and synchronization overheads. DDM-CMP avoids the complexity of other designs by combining simple commodity microprocessors with a small hardware overhead for thread scheduling and an interconnection network. Preliminary experimental results show that a DDM-CMP chip of the same hardware budget as a high-end commercial microprocessor, clocked at the same frequency, achieves a speedup of up to 18.5 with a 78–81% power consumption of the commercial chip. Overall, the estimated results for the proposed DDM-CMP architecture show a significant benefit in terms of both speedup and power consumption making it an attractive architecture for future processors.

KEY WORDS: Chip multiprocessor; multithreading; parallel processing; data-driven execution.

1. INTRODUCTION

To deliver performance as predicted by Moore's Law, computer architects have relied on the advancements of process technology, as well as

[1]Department of Computer Science, University of Cyprus, Nicosia, Cyprus.
[2]Computer Engineering Department Frederick Institute of Technology, Nicosia, Cyprus.
[3]To whom correspondence should be addressed. E-mail: pedro@cs.ucy.ac.cy

improvements of the computer architecture and organization. While this approach has worked well in the past, it is currently only resulting in diminishing returns.(1) This is due to the inability of traditional architectures in surpassing two major obstacles: the *memory* and *power walls*. Both walls can be traced back to the von Neumann model of execution that has dominated the computer architecture field since the advent of digital computers. The memory wall problem is due to the imbalance between the speed of microprocessors and that of main memory, while the power wall is due to the high frequencies and complexity in modern microprocessors.

Current microprocessors have a high transistor density, execute at very high frequencies, include large cache memories and rely heavily on out-of-order and speculative execution. For the implementation of these techniques, multiported and even replicated devices are required. This however, leads to an exponential increase at the gate level whereas getting the power consumption out of hand.(2) As such, major microprocessor manufacturers have shifted their strategy to multicore chips in order to avoid the memory and power walls.

It appears that the current trends in architecture and compiler design might not be able to sustain the performance improvements achieved in the past.(1) Consequently, we propose to use Data-Driven Multithreading (DDM),(3,4) as it does not suffer from the previously mentioned limitations. This is due to the fact that DDM is not based on the von Neumann model of execution but instead on the dataflow model of execution, which is side-effect free and scheduling is based on data availability. Thus, the memory latencies can be tolerated without the huge performance penalty of the von Neumann model. Furthermore, the data-driven scheduling does not require the complexity of the multiple issue and out-of-order execution. DDM is a non-blocking multithreading model based on the Decoupled Data-Driven (D^3) model of execution.(5,6) This model decouples the synchronization from the computation portions of a program allowing them to execute asynchronously. In this model a thread is scheduled for execution in a dataflow manner, i.e., whenever all of its required data have been produced. As a consequence, no synchronization or communication latencies are experienced at the execution stage. The performance improvement is achieved at the expense of extra off-chip hardware: the Thread Synchronization Unit (TSU). This hardware is responsible for the dynamic scheduling of threads in the multiprocessor system. In addition, the DDM implementation may be used with conventional off-the-shelf microprocessors. Therefore, it has the obvious benefit that a system may combine both DDM and the latest microprocessor technology. DDM can also benefit from *CacheFlow*, a cache management policy that uses the information provided by the DDM scheduling together with data prefetching.

In this paper we present a Chip Multiprocessor implementation using the Data-Driven Multithreading model (DDM-CMP).(7) This is the extension of the previously proposed DDM implementation, which was based on the Network-of-Workstations model. In the current study we analyze two alternative DDM-CMP configurations designed with the same hardware budget as the baseline Intel Pentium 4 high-end microprocessor: a configuration with 4 medium-complexity Pentium III cores and a configuration with 8 simpler cores (Pentium III with reduced cache size). An estimate of the power consumption shows that with equal frequency, technology and hardware budget as the baseline Pentium 4 chip, the DDM-CMP chips with 4 and 8-cores consume less power than the baseline, 22% and 19%, respectively. For a scenario where we allow the DDM-CMP chip to consume the same power as the baseline processor, we observe a performance speedup ranging from 2.3 to 11.7 for the 4-core DDM-CMP and 2.3 to 22.6 for the 8-core DDM-CMP, for the benchmarks studied. Also, we can achieve large savings in the power consumption if we limit the performance to be equal to the one achieved by the baseline. For this scenario we observe power savings larger than 65% for the 4-core DDM-CMP and 63% for the 8-core DDM-CMP. Overall the results are very encouraging for the proposed DDM-CMP architecture.

This paper is organized as follows: Section 2 presents the data-driven multithreading model and implementation, Section 3 describes the advantages of the chip multiprocessor architecture, and Section 4 shows how the chip multiprocessor may benefit from using the data-driven multithreading model. Section 5 presents the power consumption analysis for the DDM-CMP chip, while in Section 6 we analyze the experimental results obtained. Finally, in Section 7 we discuss the related work and in Section 8 we present the conclusions and future work.

2. DATA-DRIVEN MULTITHREADING (DDM)

DDM provides effective latency tolerance by allowing the computation processor produce useful work, while a long latency event is in progress. This model of execution has been evolved from the dataflow model of computation. In particular, it originates from the dynamic dataflow D^3 graphs,(5,6) where the synchronization part of a program is separated from the computation part. The computation part represents the actual instructions of the program executed by the computation processor, while the synchronization part contains information about data dependencies among threads and is used for thread scheduling.

A program in DDM is a collection of re-entrant code blocks. A code block is equivalent to a function or a loop body in the high-level program text. Each code block comprises of several threads. A thread is a sequence of instructions equivalent to a basic block. A producer/consumer relationship exists among threads. In a typical program, a set of threads, called the *producers*, create data used by other threads, called the *consumers*. Scheduling of code blocks, as well as scheduling of threads within a code block is done dynamically at runtime according to data availability. The instructions within a thread are fetched by the CPU sequentially in control-flow order. Nevertheless, the CPU can reorder the sequence of instructions internally to exploit the advantages of out-of-order execution.

At compile time a program is partitioned into a *data-driven synchronization graph* and *code threads*. Each node of the graph represents one thread associated with its synchronization template. Each thread is identified by the thread number (*Thread#*) consisting of the triplet (*Context, Block, ThreadID*). The *Context* field is set at runtime to distinguish between multiple invocations of the same code block or thread. This is useful for the implementation of multiple invocations of functions and loop bodies. The *Block* field identifies the code block, while the *ThreadID* identifies the thread within the code block. The synchronization template of each thread contains the following information: *Ready Count*, *Instruction Frame Pointer (IFP)*, *Data Frame Pointer (DFP)* and *Consumer threads* (*Consumer1* and *Consumer2*). The *Ready Count* is set by the compiler and corresponds to the number of input values, i.e., producers to the thread. This value is decremented at runtime and a thread is enabled, i.e., it is ready for execution when its Ready Count reaches zero. Whenever the thread completes its execution, it uses the *Consumer thread pointers* to decrement their ready count.

As for the code of the threads, the compiler identifies certain blocks of code where threads may be created. Examples of such code are the loop body and function instructions. Assuming that there are no dependences, for a fixed number of instructions, the compiler may opt between creating a large number of threads with few instructions or a small number of threads with a large number of instructions each. The former is favorable in exploiting more parallelism but the latter is favorable in reducing the thread execution overheads (metadata maintenance and scheduling). Therefore, the compiler uses heuristics to statically determine the number of instructions to be added to a thread. It is relevant to notice that the compiler creates the threads according to the application characteristics and not the architecture configuration. If a large number of threads is created, while a DDM-CMP configuration with more cores will result in better performance, the same code will be able to execute in any other DDM-CMP configuration.

Regarding the DDM memory model, it assumes the existence of a shared memory address space. Nevertheless, it does not require an explicit cache coherence protocol to assure correct execution. The protocol is avoided by forwarding all the shared data from the producer to the cache of the consumer threads.

2.1. TSU: Hardware Support for DDM

The purpose of the *Thread Synchronization Unit* (TSU) is to provide hardware support for data-driven thread synchronization on conventional microprocessors. The TSU is made out of three units: the *Thread Issue Unit* (TIU), the *Post Processing Unit* (PPU) and the *Network Interface Unit* (NIU). When a thread completes its execution, the PPU updates the *Ready Count* of its consumer threads, determines whether any of those threads became ready for execution, and if so, it forwards them to the TIU. The function of the TIU is to schedule and prefetch threads deemed executable by the PPU. The NIU is responsible for the communication between the TSU and the interconnection network. The internal structure of the TSU is shown in Fig. 1.

The TIU provides the computation processor with all information related to the thread scheduled to be executed. This information can be found in two different queues: the *Waiting Queue* (*WQ*) and the *Firing*

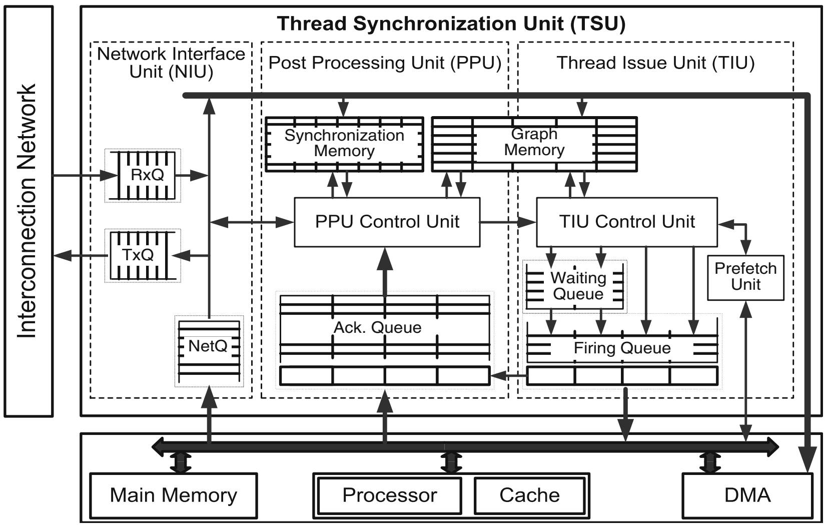


Fig. 1. Thread Synchronization Unit (TSU).

Queue (*FQ*). The thread identification number (Thread#) and iteration number of ready threads are first placed in the WQ. The Thread# is used to determine the thread's IFP and the DFP from the Graph Memory (GM).

The computation processor passes information about the completed threads to the PPU through the Acknowledgment Queue (AQ). This information is the thread number (*Thread#*), the iteration number (*AqIndx*) and the status of the completed thread (*AqStat*). An extra field is provided by the AQ, the *AqData* that is used to pass the data produced by threads that send single values to other nodes. The Thread# is used as an index to the GM to determine the Thread# of the consumers of the completed thread. The Synchronization Memory (SM) is indexed with the Thread# number of the consumer threads. If a consumer thread becomes ready, it is shifted to the WQ of the TIU. If a ready thread belongs to a remote node, it is forwarded to the NIU for further processing.

2.2. CacheFlow

Although DDM can tolerate communication and synchronization latency, scheduling based on data availability may have a negative effect on locality. To overcome this problem we use the scheduling information to implement efficient short-term optimal cache management policies, which we named *CacheFlow*.(8)

The basic CacheFlow implementation employs hardware prefetching to fetch into the cache the data needed by the threads scheduled for execution in the near future. To achieve this, two extra fields are added in the thread's template pointing to the data needed by the thread. These fields are loaded into the Graph Memory at runtime as Data Offset Pointer 1 (DOP1) and Data Offset Pointer 2 (DOP2). If a thread has only one input, then DOP1 points to that value, while DOP2 is set to 0. If a thread has more than two inputs, then DOP1 is set to zero, while DOP2 is a pointer to the DOP list, a memory block that contains a list of DOPs.

The implementation of the CacheFlow policies is directly related to the TIU. When a thread becomes ready, its identification number and context are placed in the WQ, by the PPU. The TIU uses the thread identification number to determine the threads starting address and data pointers by reading its IFP, DFP and DOPs from the Graph Memory. The addresses pointed by the DOPs relative to the DFP are prefetched before the thread is shifted from the Waiting Queue to the Firing Queue.

In addition to the basic CacheFlow policy presented, we explored two optimizations to further improve its performance. The first optimization, called the *false conflict avoidance*, prevents the prefetcher from replacing cache blocks required by the threads waiting in the FQ. This is achieved with the use of a table, called the *Reserved Address Table* (RAT), that contains the addresses of all cache blocks prefetched for the threads waiting in the FQ, as well as the thread currently running. All addresses required by a ready thread are determined using the information from the Graph Memory and are placed in the Tag Queue. These addresses are then compared with the contents of the RAT to determine if prefetching would cause a false cache conflict. A thread is shifted into the FQ if none of its addresses would result in a cache conflict. If it is detected that an address would result in a false cache conflict, then the thread is placed temporarily in a buffer and the next thread from the WQ is tested. Threads waiting in the temporary buffer have precedence over the threads in the WQ. This is essential to avoid thread starvation, as a thread waiting in the temporary buffer is blocking its consumers from executing.

The second optimization, called the *thread reordering*, attempts to exploit locality by reordering the threads in the WQ so that threads with the same identification number (Thread#) are placed near each other in the WQ. This increases the possibility that the code of a thread will be used many times before it is replaced from the cache, thus exploiting temporal locality. Furthermore, threads with the same Thread# are ordered according to their index (iteration number), thus exploiting spatial data locality. The thread reordering mechanism operates in parallel and asynchronously with the rest of the TIU and consequently it does not add any extra delays in the datapath of the TIU.

3. CHIP MULTIPROCESSOR

Until recently, increasing the processor's performance was achieved by adding more features in order to exploit more instruction level parallelism. Nevertheless, this resulted in larger, more complex microprocessors which are neither easy to scale any further, nor may execute at high clock frequencies.(1) Therefore, the solution adopted by all major manufacturers is to exploit parallelism by using multiple simpler processors packed on the same chip. This is known as the Chip Multiprocessor or CMP.

One of the key advantages of the CMP architecture is that it can achieve very high throughput, thus making this architecture ideal for server systems.(9)

Another major advantage is the fact that CMPs are more power-performance efficient architectures. For example, although the performance of the dual-core AMD Athlon 64 × 24800+ (2.4 GHz) processor is higher than the one of the single-core AMD Athlon 64 4000+ (2.4 GHz), the power consumption increases only by 24% (110 W vs. 89 W).(10) This is due to the fact that some of the components of the chip infrastructure are shared in a multi-core chip such as, the clock distribution network, the power distribution network and in the Athlon's case, the memory and hyper-transport controllers, which are shared among the cores.(11) Also, as multi-core architectures are a more efficient architecture, their frequency may be scaled down without loss of performance, thus resulting in a decrease in power consumption. This is the strategy currently followed by Intel who have released dual core processors clocked at a lower frequency than their single core ones.

A larger number of cores on the same chip offers some flexibility for the assignment of tasks to processors. This advantage may be used by a scheduler in order to avoid chip hot-spots and therefore, assign the next task to a core that is far from the hottest point of the chip. A design of a thermal-aware scheduler has been recently proposed.(12,13) In an extreme case, cores may be shut down altogether(14) in order to cool down a certain area or the overall chip. Notice that even in the extreme case, the user would not be dramatically affected as applications will still execute normally but on a smaller number of cores.

Related to the last topic is the fact that if one of the cores of a multi-core chip is detected to be faulty, either during production or regular execution, that core may simply be disabled. This has only a small impact on the execution of the applications as it results only in a reduction of its performance due to the reduced available units for exploiting parallelism. Also, the multiple cores may be used to execute redundant processes in order to avoid soft errors that happen during the execution of an application. Both techniques mentioned make the CMP an architecture that is more fault-tolerant compared to monolithic microprocessor.

Overall, the characteristics of the CMP architecture offer special benefits which have been recognized by all major processor manufacturers and consequently, will become the standard technology for future microprocessors.

4. DDM-CMP

In spite of the advantages mentioned in the previous section, one challenging issue for the CMPs is the configuration of the core to be used. Although smaller cores will result in a worse single-threaded performance, the smaller the core, the larger the number of cores that may be included in the same area resulting in an increased ability to exploit parallelism.(7) DDM helps toward this goal as it may achieve high performance independently of the core features. For example, Instruction Level Parallelism (ILP) is an added benefit but not a requirement. In addition, CacheFlow performs deterministic prefetching of the data used in the threads before their execution. Therefore, there is no need for large caches to capture temporal locality. As such, the cores may be simpler and smaller and consequently more cores may fit in the same area resulting in a potential higher degree of parallelism offered to the applications.

Finally, DDM is designed to use commodity CPUs. As we may use directly the existing core IPs for the processor to be replicated on the chip, a DDM-CMP chip should be implemented without major effort other than the one-time design of the TSU and the interconnection network. Fig. 2 depicts four different alternatives for the DDM-CMP architecture.

The first proposed CMP architecture (Fig. 2-(a)) is the integration of the previously proposed D^2NOW(15) into a single chip. While having one TSU per processor was required in the NOW system, when all processors are on the same chip it is possible to optimize the use of the TSU structure and share it among two or more processors (Fig. 2-(b)). Ultimately, we may consider the extreme case where one TSU is shared among all on-chip CPUs (Fig. 2-(c)). Notice that by saving hardware with the sharing of the TSUs it may be possible to increase the number of on-chip CPUs or alternatively add internal shared cache (Fig. 2-(d)). In this paper we will only evaluate the first architecture.

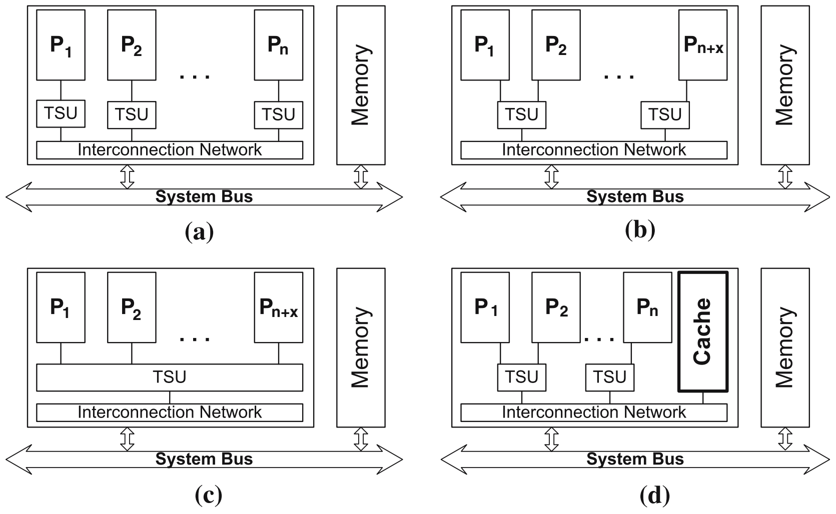


Fig. 2. Several alternatives of the DDM-CMP architecture. (a) Each microprocessor has its own TSU. (b) One TSU is shared among two microprocessors and the number of cores increases. (c) One TSU serves all the microprocessors of the chip and the number of cores increases. (d) Saved space is used to implement on-chip shared cache.

4.1. Design 1: DDM-CMP with Pentium III Cores

The objective of the proposed DDM-CMP architecture is to achieve better performance and/or lower power consumption than a current high-end microprocessor, given the same hardware budget, i.e., the same number of transistors. For our analysis we consider the Intel Pentium 4 as the baseline for the high-end microprocessor. As mentioned before, DDM-CMP is build out of simpler cores. For the purposes of our analysis we consider Intel Pentium III as a representative of such a core. The number of transistors used in implementing Intel Pentium 4 3.2 GHz 90 nm technology is approximately 125 million, while the number of transistor used in implementing the Intel Pentium III 800 MHz 180 nm technology is 22 million.(16) Therefore, the Pentium 4 requires approximately 5.7 times more transistors than what is needed to build the Pentium III.

In addition to the processors, other hardware structures are needed to implement the DDM-CMP architecture: the TSUs and the interconnection network. Regarding the TSU, a preliminary study of its implementation on FPGA lead to an estimate budget of approximately 25 K transistors per TSU.(3) This results in only 100 and 200 K transistor requirement for the TSUs for the 4- and 8-core DDM-CMP chips, respectively. Regarding the interconnection network, several studies report a 10% of the total chip area devoted to its implementation. Considering the case above where we use the area of a Pentium 4 chip in order to implement four Pentium III processors, about 37 million transistors will be left unused. This number of transistors is more than enough to implement the four TSUs and the appropriate interconnection network. Therefore, a DDM-CMP architecture with four Pentium III processors can be implemented with the same number of transistors needed to build a Pentium 4.

4.2. Design 2: DDM-CMP with Minimal Configuration Cores

As the DDM model does not require complex microprocessors or large cache space, the Pentium III core used in the previous design may not be the most efficient core for the DDM-CMP. For example, with DDM's CacheFlow one may expect that the need for cache space is reduced. As such, the challenge was to find out the minimal specifications required for a core, without significant single-process performance penalty, thus fitting more cores into the same chip. As a result, a DDM-CMP chip with increased number of cores will be able to offer a higher degree of parallelism.

In order to achieve our goal, we performed several experiments to observe the impact on the application performance of using different configurations for the core processor. In particular, we used the SimpleScalar(17) simulator to test three SPLASH-2(18) application (FFT, Radix, and LU) using different cache configurations: first-level data cache (*L1 D*\$) ranging from 4 KB to 32 KB, first-level instruction cache (*L1 I*\$) ranging from 4 KB to 512 KB and second-level unified cache (*L2 U*\$) ranging from 64 KB to 4 MB. We also tested the support for instruction-level parallelism by changing the issue width from 1 up to 4 and by selecting both in-order and out-of-order execution. In addition, we tested the configurations for the execution of the application with small and large input data set size.

As scaling down the original Pentium III core to include fewer functionality is not a trivial matter, we focused on the cache requirements of the applications.

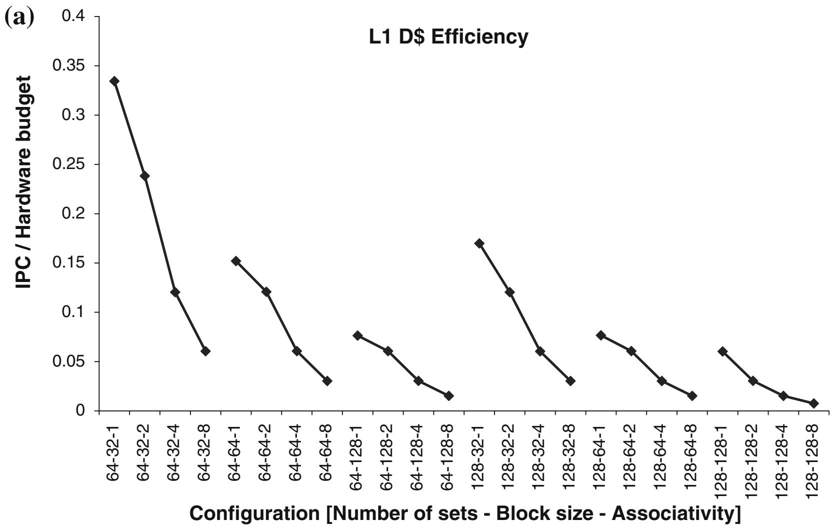
To evaluate the best cache size configuration for the core we used a metric that takes into account both the performance and the die area occupied by the corresponding cache. We named this metric as *efficiency* and we defined it as follows:

$$\mathrm{Eff} = \frac{\mathrm{IPC}}{\mathrm{Area}}$$

In Fig. 3-(a) and (b) we present the results for the efficiency of the *L1 D*\$ and *L1 I*\$ caches, respectively. In the x-axis we show different cache configurations, depending on their characteristics (number of sets, block size, and associativity).

It is relevant to notice from the results that the most efficient cache configurations are those where the cache is small. This is justified by the fact that although larger caches achieve lower miss rates, the reduction is not enough to justify the increase in area required. In addition, larger caches also have larger access times which contribute negatively toward the IPC.

Overall, the experimental results lead us to conclude that an efficient CMP core should include small caches and be able support out-of-order execution. Based on these results, the core configuration chosen for this second DDM-CMP design includes an 8 KB *L1 I*\$, an 8 KB *L1 D*\$ and a 32 KB *L2 U*\$. This core supports out-of-order execution and is able to issue 4 instructions per cycle. Using this configuration we estimate the area, in terms of number of transistors, for each processor configuration, using the floor plan of the Pentium III. The new area is determined by scaling down the area of the original caches by a factor equal to the cache size reduction. Finally, the reduction in the total area of the original Pentium III is such that it allows us to implement 8 of these smaller cores into the same area budget of the baseline Intel Pentium 4 microprocessor.



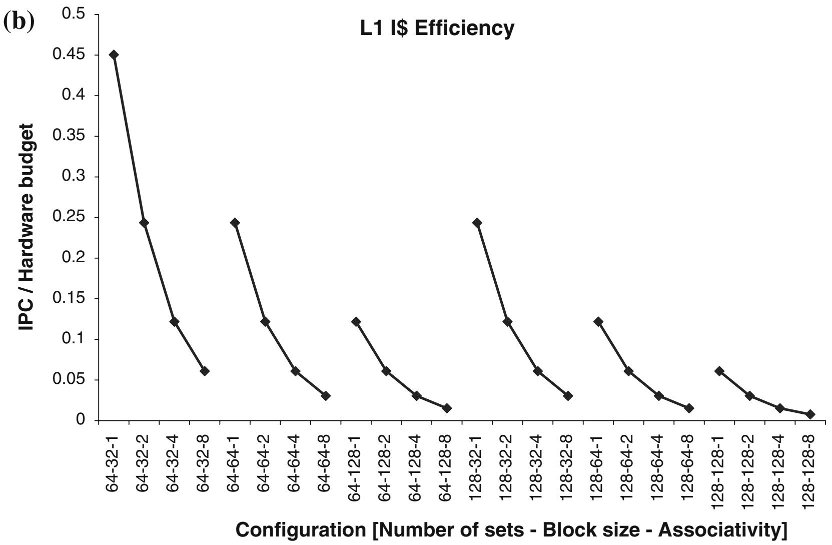


Fig. 3. Cache Efficiency: (a) L1 D$ and (b) L1 I$.

5. POWER ANALYSIS FOR THE DDM-CMP

In this work we present an estimate of the benefits in terms of power consumption for the DDM-CMP architectures. We need to remind that both DDM-CMP Design 1 and 2 as presented in the previous section, occupy the same hardware budget, i.e., they have approximately the same number of transistors, as the baseline Intel Pentium 4 microprocessor. As such, the only factors that will result in a different power consumption are: the support for *Hyper-Threading*, the cache memory size and the percentage of transistors devoted to logic as opposed to memory. Notice that *Hyper-Threading*, or *HT*,(19) is Intel's implementation of the Simultaneous Multithreading technology.(20)

To estimate the DDM-CMP power consumption we developed a mathematical model, which determines the power consumption using the equation shown below:

$$\begin{aligned}\text{Power} = \; & a_0 + a_1 \times (\#\text{Trans})^{e_1} + a_2 \times (\text{HT})^{e_2} \\ & + a_3 \times (\text{Tech})^{e_3} + a_4 \times (L1)^{e_4} + a_5 \times (L2)^{e_5}\end{aligned}$$

In this equation, term #Trans represents the number of transistors of the processor in millions, HT takes the value 1 if the processor supports Hyper-Threading and 0 if not. Tech is the technology point in nm, whereas $L1$ is the size of L1 cache in KB (both instruction and data) and $L2$ the size of L2 cache in KB.

The parameters of the equation ($a0 \cdots a5, e1 \cdots e5$) are estimated using the *multiple polynomial regression* method. Specifically, the model is trained using characteristics of 186 different processors and achieves maximum relative error of 18.3% and average absolute error of 8.8%.

The training samples used include mobile processors (mobile Pentium III and mobile Celeron), high-end server processors (Xeon), low-end desktop processors (Celeron) and different models of high-end desktop processors (Pentium III and Pentium 4). Additionally, the training set contains processors with and without the Hyper-Threading technology, clocked at frequencies ranging from 400 to 3800 MHz, with implementation technologies of 180 and 90 nm and L1 caches ranging from 12 Kops/8 KB to 12 Kops/16 KB and L2 caches from 128 KB to 2 MB. The data was collected from.(10) The processors for which we have estimated the power consumption in our experiments are covered by this range of configurations.

In Table I we show the memory sizes, the estimated power consumption, and the *Iso-Power Frequency* for the Pentium 4 and the two DDM-CMP designs (4 and 8 cores). The *Iso-Power Frequency* is the frequency that results in the same power consumption as the one for the baseline Pentium 4 processor. Both the estimated power consumption and iso-power frequency are obtained using the power model as described above. Note that from hereafter we call the DDM-CMP Design 1 the *DDM-CMP4* and the Design 2 the *DDM-CMP8* due to the number of cores in each design.

Table I. Memory Size and Percentage of Logic and Memory Transistors Per Chip

	Pentium4	DDM-CMP4	DDM-CMP8
L1 I$	12 Kops (32 KB)	4×16 KB = 64 KB	8×8 KB = 64 KB
L1 D$	16 KB	4×16 KB = 64 KB	8×8 KB = 64 KB
L2 U$	1 MB	4×256 KB = 1 MB	8×32 KB = 256 KB
Est. Power (3.2 GHz)	91.4 W	71.3 W	74.0 W
Rel. Power (3.2 GHz)	100%	78%	81%
Iso-Power Freq.	3200 MHz	4009 MHz	3901 MHz

The results of the model for the power consumption of the different architectures show that the DDM-CMP8 has slightly higher power consumption than the DDM-CMP4 and both have lower power consumption than the baseline Pentium 4. Once the results are for the case where all three architectures are clocked at the same frequency, the major factor that determines the difference between the power consumption of the Pentium 4 and the DDM-CMPs is due to the hardware required to support Hyper-Threading, which exists only in the Pentium 4. As for the difference between DDM-CMP4 and DDM-CMP8, this is due to the ratio between logic and memory transistors. We call *memory transistors* the transistors used to implement the caches, whereas *logic transistors* the rest of the transistors on a chip, i.e., the transistors used to implement the functional units. Although both DDM-CMP4 and DDM-CMP8 use approximately the same number of transistors for their implementations, in the DDM-CMP8 the strategy was to exchange memory space for computational units, i.e., we reduced the cache sizes and increased the number of cores with the space that was made free. Therefore, the ratio of logic-to-memory transistors in the DDM-CMP8 is higher than in the DDM-CMP4. Due to the fact that memory transistors have smaller activity factors (percentage of time a transistor switches) compared to logic transistors,(21) the former consume less power than the latter. As such, it is expected that the DDM-CMP8 consumes more power than the DDM-CMP4, as shown in the results.

It is relevant to notice that, at this point, in the power consumption estimate, we do not directly account for either the TSUs or the interconnection network. Instead, we use all the transistors from the baseline Pentium 4 in our estimate. This should be a conservative estimate as both TSUs and interconnection network do not account for such a large number of transistors.

Another relevant piece of information that we extracted from the model was the frequency scaling that can be done in order to match the

Table II. DDM-CMP8 Performance Scaling Factors

FFT	LU	Radix	Mmult	Trapez
0.98	0.99	1.00	0.51	1.00

power consumption of the DDM-CMP architectures to the one reported for the baseline Pentium 4. We call this the *Iso-Power Frequency*. As presented in Table I, DDM-CMP4 consumes the same power as the baseline Pentium 4 when clocked at 4009 MHz, while for DDM-CMP8 this frequency is 3901 MHz.

6. EXPERIMENTAL RESULTS FOR THE DDM-CMP

In this section we present three types of results: (a) Speedup and power consumption for equal frequency, (b) Speedup for equal power consumption and (c) Power consumption for equal speedup. These results are presented for the Pentium 4, DDM-CMP4 and DDM-CMP8 architectures.

6.1. Experimental Setup

For this proof-of-concept analysis the workload considered to test the proposed architecture is composed of three kernels from the SPLASH-2 benchmark suite,(22) LU, FFT and Radix and two applications that represent standard algorithms used in large scientific applications such as the block matrix multiply (*Mmult*) and the trapezoidal method of integration (*Trapez*).

As for the baseline high-end processor we have selected the Pentium 4 3.2 GHz. To obtain the results for this setup we measure the execution time of the application's native execution on that system. The execution time is determined by measuring the number of processor cycles consumed in the execution of the main function of the program, i.e., we ignore the initialization phase. The processor cycles are measured by reading the contents of the hardware program counter of the processor.(23) Notice that as this is native execution, in order for the results to be statistically significant we execute the same experiment several times and we report the average result after having excluded the largest and smallest measurements.

The performance results for the DDM-CMP architecture are derived from the results obtained by Kyriacou et al.(24) for the D^2NOW implementation. In this case, we will use the results for the D^2NOW architecture configured with 4 and 8 Pentium III 800 MHz processors including all architecture optimizations as a base for the DDM-CMP4 and DDM-CMP8 setups. Notice that the D^2NOW results are conservative for the DDM-CMP architecture, as the on-chip interconnection network has both larger bandwidth and smaller latency than the D^2NOW interconnect. In addition, we adjust the DDM-CMP8 results using a factor obtained by simulating the same application in SimpleScalar(17) using the different processor configurations. The values of these factors are presented in Table II.

Notice that the performance scaling factors are all close to 1 except for *Mmult*, which is 0.51. This is due to the fact that this particular application has a working set larger than the cache space available in the DDM-CMP8. Therefore, there is a large penalty in its performance due to an increase in the cache misses observed.

6.2. Speedup and Power Consumption for Equal Frequency Scenario

In Fig. 4 we present the projected speedup and the estimated power consumption for the DDM-CMP4 and DDM-CMP8 systems for the five applications, for the two DDM-CMP alternative designs (DDM-CMP4 and DDM-CMP8). The speedup, reported in the first two groups of five bars presented in the Figure, is relative to the execution time of the applications on the Intel Pentium 4 3.2 GHz baseline high-end processor. The last group of two bars presents the power consumption for the DDM-CMP architecture compared to the Pentium 4. In this setup, the frequency of both the DDM-CMP4 and DDM-CMP8 systems are set to 3.2 GHz, the same as the baseline processor.

From the chart in Fig. 4-(a) we observe that, depending on the characteristics of the application, such as their working set size and data

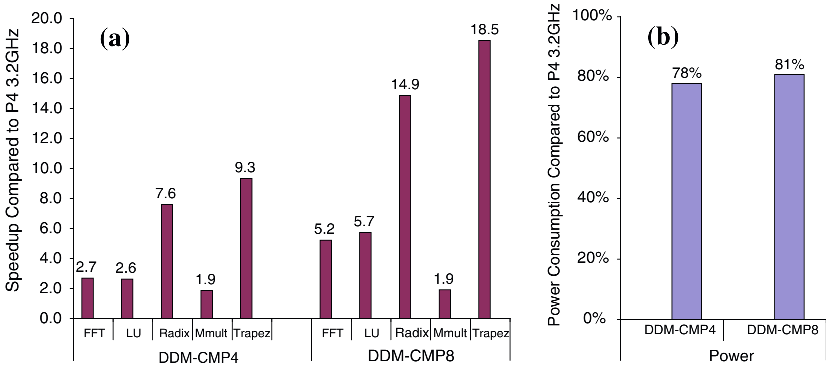


Fig. 4. (a) Speedup and (b) power consumption for frequency equal to the one of the baseline processor.

dependencies between threads, the speedup values observed range from a moderate 1.9 for *Mmult* on the DDM-CMP4 to 18.5 for *Trapez* on the DDM-CMP8. In the next section we analyze in more detail the different characteristics of applications that justify these results. One interesting fact observed with the results in Fig. 4-(a) is that although in DDM-CMP8 the cores have smaller cache space, the speedup obtained across almost all tested applications doubles from the one obtained for the DDM-CMP4. The other result, observed in Fig. 4-(b), is that the power consumption of these two chips is smaller than the one of the baseline chip. For the DDM-CMP4 we observe the power savings to be 22%, while for the DDM-CMP8 the power savings is 19%. The justification and discussion of these results was already presented in Section 5.

6.3. Speedup Results for Equal Power Consumption Scenario

In Fig. 5 we present the projected speedup for the DDM-CMP4 and DDM-CMP8 systems for the five tested applications, compared to Intel Pentium 4 3.2 GHz baseline high-end processor for equal power consumption. The frequency of both the DDM-CMP4 and DDM-CMP8 systems has been set based on the fact that all three systems (Pentium 4, DDM-CMP4, and DDM-CMP8) should consume the same power. We call this frequency the *Iso-Power Frequency* and we determined it using the power model as described in Section 5. According to this model the DDM-CMP4 should be clocked at 4009 MHz and the DDM-CMP8 at 3901 MHz.

From the results in Fig. 5 it is possible to observe that there is a speedup larger than two for all applications and the two different DDM-CMP setups. The speedup ranges from 2.3 to 11.7 for the DDM-CMP4 setup and from 2.3 to 22.6 for the DDM-CMP8. It is also interesting to observe that the results can be of three types. The first type of result is a moderate sub-linear and non-scalable speedup as observed for the *Mmult* application. The speedup for this application is relatively low and does not increase for the DDM-CMP configuration with larger number of cores. This is a consequence of the fact that the characteristics of this application include a large exchange of data between the different cores and also the fact that the data handled is considerably large and so there is a penalty when changing to the DDM-CMP8 configuration where the cache memory space is smaller. A small experiment where we monitored the number of misses observed during the execution of *Mmult* showed that while the *L1 I*\$ and *L1 D*\$ misses are constant, the *L2 U*\$ misses increase by a factor of 60 when changing from the Pentium 4 to the Pentium III core. With

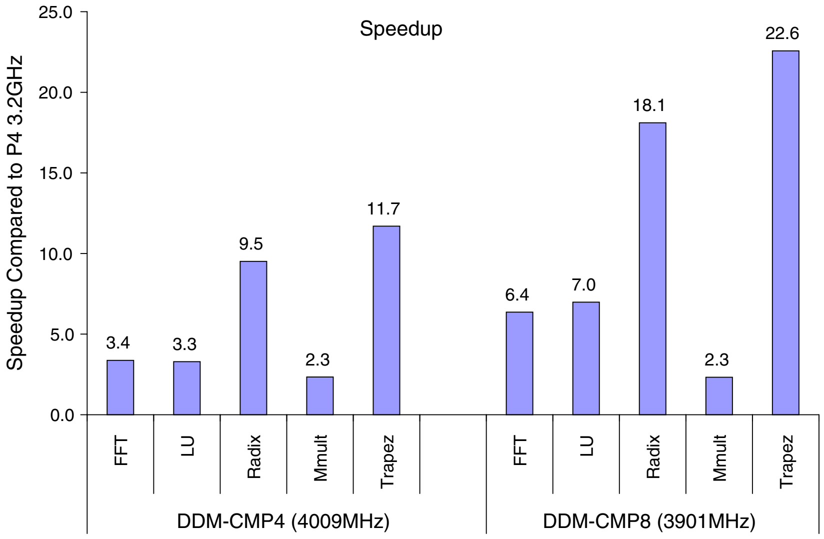


Fig. 5. Speedup when frequency and core scaling is taken into account.

the smaller core, the *L2 U*$ misses increase by a factor larger than 2000 when comparing to the Pentium 4 baseline.

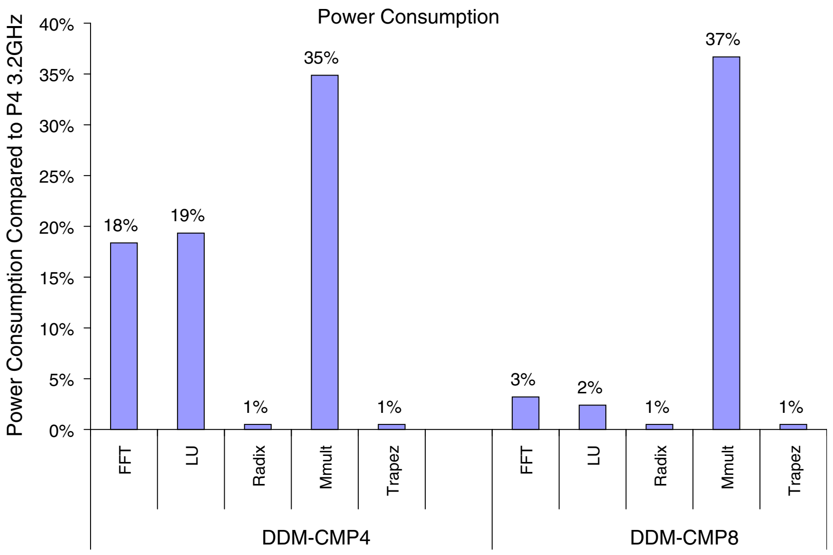
The second type of result is a moderate sub-linear but scalable speedup as observed for *FFT* and *LU*. For these applications the data exchanges is still an important factor. Nevertheless, the working set is smaller and is not affected by the decrease of the cache memory space offered by the DDM-CMP8 configuration. In order to better understand the behavior of these two applications, we performed a brief analysis of their execution and we were able to determine that the main function that performs the calculations for the algorithm accounts for approximately only 50% of the total execution in both FFT and LU. This is an indication that in order to obtain more reliable results for both FFT and LU we will need to use larger data set sizes. This issue will be covered in the near future as we complete the DDM-CMP simulator. Nevertheless, at this point we do not consider this result to be a problem as we expect that there will always be applications that will show such moderate performance improvement when executing on the DDM-CMP.

The third type of results is a large super-linear and scalable speedup as observed for *Radix* and *Trapez*. These results are justified by the fact that these applications are easier to parallelize and include fewer data exchanges between the different threads of the application and their working set is not affected by the smaller cache setup of DDM-CMP8. In addition, for these two applications, the CacheFlow feature offered by DDM-CMP results in more successful data prefetching for the application threads, therefore leading to very efficient execution of the applications. It is important to notice that the speedup is determined in comparison with the baseline, which is a Pentium 4. The two architectures (Pentium III and 4) have some differences which justify part of the super-linear speedup as the fact that the Pentium III L1 caches are larger than the Pentium 4. In addition, it is relevant to add that the speedup presented should be taken as an indication of the possible upper bound and not as their absolute value as they are a result of an estimation. On the one hand, the speedup results are over-estimated if we consider that the frequency scaling was applied to the complete execution time, including memory access time and not just the compute time. Nevertheless, it is relevant to state that the memory access is limited due to the success of the CacheFlow prefetching policies as it reduces the L2 cache misses to the orders of 1%. On the other hand, the speedup results are under-estimated if we consider that the intra-chip interconnection network has a smaller latency than the one on the NOW and the accesses to the TSU are faster as they are on-chip accesses.

6.4. Power Savings Results for Equal Speedup Scenario

After confirming in the last section that DDM-CMP can achieve a high speedup for the equal power consumption scenario, next we would like to test how much power savings DDM-CMP can achieve while keeping the same speedup as the baseline architecture. Therefore, in this section we present the expected power consumption for the two DDM-CMP configurations, normalized to the Pentium 4, when the speedup is the same for all the three systems: Intel Pentium 4, DDM-CMP4, and DDM-CMP8. In order to achieve the same speedup we had to scale the frequency for the two DDM-CMP configurations. This scaling is performed in a per-application basis. Figure 6 depicts the power consumption of the different configurations for the different applications, compared to the Intel Pentium 4 3.2 GHz baseline. The table on the bottom of the chart contains the frequencies used for each architecture and application in order to achieve the constant speedup. These frequency values are an estimate and they were determined by simply assuming that the application performance scales linearly with the frequency.

The results in Fig. 6 show that the DDM-CMP configurations achieve the same speedup at a fraction of the power consumption for the original system. For the DDM-CMP4 we observe savings in the power consumption larger than 65% while for the DDM-CMP8 the savings are larger



	FFT	LU	Radix	Mmult	Trapez
DDM-CMP4 Freq[MHz]	1189	1219	421	1712	343
DDM-CMP8 Freq[MHz]	613	559	215	1677	173

Fig. 6. Power savings of the multicore DDM-CMP comparing with the baseline Pentium 4.

than 63%. As some applications achieve very high speedup (10–20×) for the equal power consumption scenario, the performance can be the same as the one achieved with the Pentium 4 results with a considerable reduction of the frequency. As the power consumption changes with the cube of the frequency, this frequency reduction results in a dramatic decrease in the power consumption. Therefore, it is possible to observe the 1% values in Fig. 6. For this work, although the power model gave very low power consumption for certain frequencies, we kept the minimum power consumption at 0.5 W, the same as achieved by an Intel XScale.(25)

7. RELATED WORK

The architecture proposed in this paper combines two important fields of research: multithreading and chip multiprocessor.

Parallel architectures often suffer from large synchronization and communication latencies. Data-Driven Multithreading (DDM)(15,24) is an execution model that aims at tolerating the latencies by allowing the computation processor to produce useful work, while a long latency event is in progress. The previously proposed DDM implementation uses regular off-the-shelf microprocessors. This is a contrast with previous DDM implementations that required either special design processors(26–29) or modified existing processors.(30–33) Alternatively, software-only approaches such as TAM(34) suffer from performance degradation, in comparison to our proposed solution.

As presented in this work, an alternative design that achieves parallelism but avoids the complexity is the Chip Multiprocessor (CMP).(2) Several research projects have proposed CMP architectures.(2,35–37) In addition, commercial products have also been proposed (e.g., Intel Pentium D, AMD Athlon X2, IBM Power5,(38) and SUN Niagara(39)).

A preliminary study of the efforts into combining these two technologies, data-driven multithreading and chip-multiprocessors was presented by Stavrou et al.(7) In that work the conceptual DDM-CMP idea was presented along with very preliminary results. In this current paper, in addition to extending our analysis to a larger set of applications, we developed a power model and present estimates for the power consumption for the proposed DDM-CMP architecture.

8. CONCLUSIONS

In this paper we presented DDM-CMP, a Chip Multiprocessor architecture that uses the Data-Driven Multithreading model. This architecture aims in overcoming both the memory and power walls. DDM-CMP achieves a higher speedup and/or lower power consumption compared to an equal hardware budget high-performance single-chip uniprocessor.

An estimate of the power consumption shows that with the same frequency, technology and hardware budget as the baseline Pentium 4 chip, the DDM-CMP chip may consume 22% less power than the baseline. For a high-performance scenario where the DDM-CMP chip is allowed to consume the same power as the baseline processor, we observe a speedup increase up to 22.6 for an 8-core DDM-CMP. As for a low-power scenario where the performance is set to be the same as the baseline, we observe a significant reduction in the power consumption. Overall, the results are very encouraging for the proposed DDM-CMP architecture.

The results presented in this paper motivated us to continue this work. In the near future we will complete the DDM-CMP execution-driven simulator, as well as the DDM compiler in order to obtain more accurate results, analyze different points in the design space, and broaden the applications used to evaluate this architecture. Also, we are working on a hardware prototype of the DDM-CMP architecture based on the Xilinx Virtex II-Pro chip.

REFERENCES

1. D. Patterson, Research Accelerator for Multiprocessing (2005).
2. K. Olukotun, B. Nayfeh, L. Hammond, K. Wilson, and K. Chang, The Case for a Single Chip Multiprocessor, in *Proc. of the 7th ASPLOS*, pp. 2–11 (1996).
3. C. Kyriacou, Data Driven Multithreading using Conventional Control Flow Microprocessors. Ph.D. Thesis, Dept. of Computer Science, University of Cyprus (2005).
4. C. Kyriacou, S. Evripidou, and P. Trancoso, Data Driven Multithreading Using Conventional Microprocessors. *IEEE Transactions on Parallel and Distibuted Systems* (in press, 2006).
5. P. Evripidou, D3-machine: A Decoupled Data-Driven Multithreaded Architecture with Variable Resolution Support, *Parallel Computing*, **27**: 1197–1225 (2001).
6. P. Evripidou, and Gaudiot, J.L., A Decoupled Graph/Computation Data-Driven Architecture with Variable-Resolution Actors, in *Proc. of the 1990 International Conference on Parallel Processing (ICPP)*, pp. 405–414 (1990).
7. K. Stavrou, P. Evripidou, and P. Trancoso, DDM-CMP: Data-Driven Multithreading on a Chip Multiprocessor. *Proceedings of the 5th International Workshop on Embedded Computer Systems: Architecture, MOdeling, and Simulation (SAMOS-V)*, pp. 364–373 (2005).
8. C. Kyriacou, P. Evripidou, and P. Trancoso, Cacheflow: A Short-Term Optimal Cache Management Policy for Data Driven Multithreading, in EuroPar-04, pp. 561–570 (2004).
9. P. Kongetira, K. Aingaran, and K. Olukotun, Niagara: A 32-Way Multithreaded Sparc Processor, *IEEE Micro*, **25**: 21–29 (2005).
10. Server, T. B., CPU Types. http://balusc.xs4all.nl/ned/har-cpu.html (2005).
11. I. Gavrichenkov, AMD Athlon 64 × 24800+ Dual-Core Processor Review. http://www.xbitlabs.com/articles/cpu/display/athlon64-x2.html (2005).
12. K. Stavrou, and P. Trancoso, Thermal-Aware Scheduling for Chip Multiprocessors. Technical Report TR 05-16, Department of Coumputer Science, University of Cyprus (2005).
13. K. Stavrou, and P. Trancoso, TSIC: Thermal Scheduling Simulator for Chip Multiprocessors, in *Proceedings of the 10th Panhellenic Conference on Informatics* (PCI 2005), (2005).
14. M. Nikitovic, and M. Brorsson, A Low Power Strategy for Future Mobile Terminals, in *Proceedings of the Design, Automation and Test in Europe Conference (DATE'04)*, (2004).
15. P. Evripidou, and C. Kyriacou, Data Driven Network of Workstations (D2NOW), *Journal of UCS*, **6**: 1015–1033 (2000).
16. Intel: Intel Microprocessor Quick Reference Guide. (http://www.intel.com/pressroom/kits/quickreffam.htm).
17. D. Burger, and Austin, T.M., The Simplescalar Tool Set Version 2. Technical Report TR-1342, University of Wisconsin-Madison (1997).
18. S. Woo et al., The SPLASH-2 Programs: Characterization and Methodological Considerations, in *Proc. of the 22nd Annual International Symposium on Computer Architecture (ISCA)*, pp. 24–36 (1995).
19. D. Marr, F. Binns, D. Hill, G. Hinton, D. Koufaty, J. Miller, and M. Upton, Hyper-Threading Technology Architecture and Microarchitecture. *Intel Technology Journal*, **6**: 4–15 (2002).

20. D. Tullsen, S. Eggers, and H. Levy, Simultaneous Multithreading: Maximizing On-Chip Parallelism, in *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pp. 392–403 (1995).
21. D. Brooks, V. Tiwari, and M. Martonosi, Wattch: A Framework for Architectural-Level Power Analysis and Optimizations, in *Proceedings of the 27th Annual International Symposium on Computer Architecture*, (2005).
22. S. Woo et al., The SPLASH-2 Programs: Characterization and Methodological Considerations, in *Proc. of 22nd ISCA*, pp. 24–36 (1995).
23. PCL, The Performance Counter Library Version 2.2 (2003).
24. C. Kyriacou, P. Evripidou, and P. Trancoso, Data Driven Multithreading Using Conventional Microprocessors. Technical Report TR-05-4, University of Cyprus (2005).
25. I. Corp., Intel XScale Technology Overview. http://www.intel.com/design/intelxscale/ (2005).
26. D. Burger et al., Scaling to the End of Silicon with EDGE Architectures, *IEEE Computer*, **37**: 44–55 (2004).
27. G. Papadopoulos, and D. Culler, Monsoon: An Explicid Token Store Architecture, in *Proc. of the 17th Annual International Symposium on Computer Architecture (ISCA)*, 82–91 (1990).
28. G. Papadopoulos, and K. Traub, Multithreading: A Revisionist View of Dataflow Architectures, in *Proc. of the 18th Annual International Symposium on Computer Architecture (ISCA)*, 342–351 (1991).
29. K. Kavi, R. Giorgi, and J. Arul, Scheduled Dataflow: Execution Paradigm, Architecure, and Performance Evaluation. *IEEE Transactions on Computers*, **50**: 834–846 (2001).
30. B. Ang, Arvind, and D. Chiou, StarT the Next Generation: Integrating Global Caches and Dataflow Architecture, in *Proceedings of the International Conference on Computer Systems and Education*, IISc, (1994).
31. B. Ang, D. Chiou, D. Rosenband, M. Ehrlich, L. Rudolph, and Arvind, The StarT-Voyager: A Flexible Platform for Exploring Scalable SMP Issues, in *Proc. of Super-Computing 98*, (1998).
32. B. Ang, D. Chiou, L. Rudolph, and Arvind, The StarT-Voyager Parallel System, in *Proc. of the International Conference on Parallel Architectures and Compilation Techniques (PACT98)*, (1998).
33. H. Hum et al., A Design Study of the EARTH Multiprocessor, in *Proc. of the International Conference on Parallel Architectures and Compilation Techniques (PACT95)*, pp. 59–68 (1995).
34. D. Culler et al., TAM: A Compiler Controlled Threaded Abstract Machine, *JPDC*, **18**: 347–370 (1993).
35. L. Hammond et al., The Stanford Hydra CMP, *IEEE Micro*, **20**: 71–84 (2000).
36. L. Barroso et al., Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing, in *Proc. of the 27th ISCA*, pp. 282–293 (2000).
37. M. Taylor et al., Evaluation of the Raw Microprocessor: An Exposed Wire Delay Architecture for ILP and Streams, in *Proc. of the 31st ISCA*, pp. 2–13 (2004).
38. R. Kalla, B. Sinharoy, and M. Tendler, IBM POWER5 Chip: A Dual-Core Multithreaded Processor. *IEEE Micro* **24**: 40–47 (2004).
39. P. Kongetira, A 32-Way Multithreaded SPARC Processor, in *Proc. of Hot Chips 2004*, (2004).