A First Glance at Kilo-instruction Based Multiprocessors

Marco Galluzzi
DAC, UPC
Barcelona, Spain
galluzzi@ac.upc.es

Valentín Puente
ATC, UC
Santander, Spain
vpuente@atc.unican.es

Adrián Cristal
DAC, UPC
Barcelona, Spain
adrian@ac.upc.es

Ramón Beivide
ATC, UC
Santander, Spain
mon@atc.unican.es

José-Ángel Gregorio
ATC, UC
Santander, Spain
jagm@atc.unican.es

Mateo Valero
DAC, UPC
Barcelona, Spain
mateo@ac.upc.es

ABSTRACT

The ever increasing gap between processor and memory speed, sometimes referred to as the *Memory Wall* problem [42], has a very negative impact on performance. This mismatch will be more severe in future processor's generation. Modern cache organizations and prefetching techniques will not be able to solve this problem. A very novel and promising technique to deal with the *Memory Wall* consists on designing processors able to maintain thousands of in-flight instructions. An example of this kind of processors has been denoted as *Kilo-instruction processors* [8]. When running numerical applications, *Kilo-instruction processors* have demonstrated its ability to effectively maintain high values of IPC while increasing memory latencies.

In this paper, we will study for the first time, the influence of *Kilo-instruction processors* on the performance of small-scale CC-NUMA multiprocessors. Our first results, using an ideal network, show the enormous potential of the *Kilo-instruction processors*, when using them as computing nodes, not only for hiding local DRAM latencies but also for the remote ones. A deeper analysis, using realistic networks, reveals the existence of heavy demands on packet throughput required by each node, since larger reorder buffers translate on higher density of remote accesses. Next, we show that current interconnection networks cannot cope with this high traffic levels, so newer and faster networks have to be designed. In short, our results show dramatic performance gains over multiprocessors based on current microprocessors and dictate a possible way to build future shared-memory multiprocessor systems.

Categories and Subject Descriptors

C.1 [**Computer Systems Organization**]: PROCESSOR ARCHITECTURES

CF'04, April 14–16, 2004, Ischia, Italy.

General Terms

Design, Measurement, Performance

Keywords

Kilo-instruction Processors, Memory Wall, Shared-Memory Multiprocessors, CC-NUMA, ROB, Instruction Window, In-flight Instructions

1. INTRODUCTION

The gap between processor speeds and DRAM speeds is increasing, making the problem of the *Memory Wall* harder and harder [42]. Modern processors have to wait hundreds of cycles for data coming from main memory and the impact in the processor performance is being studied [19]. This problem is also present in multiprocessors and is accentuated by the need of a memory consistency model and by one of the two major hurdles of parallel processing: the relatively high cost of communications between processors [16].

Many techniques have been proposed and used in the past to reduce the negative effects due to memory latency in uniprocessors. Cache memory [37] exploits the spatial and temporal locality exhibited by most programs. Prefetching fetches data from memory before it is requested, and can be done by software [21] [24], by hardware [18] [2] or even by a combination of both [20] [27]. Later, the concept of having another thread analyzing more complex prefetching, bringing data from memory to L2 cache, has been introduced in [11] and continued in [5] [43] [34] [6] [38].

In multiprocessors, we can use successfully the same techniques, also to hide the communication latency. However, to reduce the memory latencies we also have to take care of the memory consistency model, where the use of a relaxed model can be useful [14]. In this way the use of a large reorder buffer seems to be very helpful [13][33][15]. To reduce communication latency new interconnection network models and new routing algorithms and artefacts are being studied [31].

Recently, a novel approach aimed to hide memory latency has been proposed [10] [8]. This approach lies in multi-checkpointing long latency instructions allowing thousands of instruction to be in flight; in fact we can keep the processor busy executing more instructions since we can virtu-

ally have a big Re-Order Buffer. This kind of processors, called *Kilo-instruction processors*, has been shown to effectively maintain the performance while increasing memory latency. With this study we want to prove that the use of *Kilo-instruction processors* is also beneficial for multiprocessor systems as we can also hide communication latency.

The results of our study come from simulations made with RSim [28], a multiprocessor simulator coming from the Computer Architect group at Rice University. This simulator environment emulates a complete CC-NUMA machine with out-of-order processors. We also have added a detailed network simulator called SICOSYS [30] that allows us to produce detailed simulations.

The first results we obtained with an *ideal network* and a memory latency of 500 cycles show us that the use of *Kilo-instruction processors* can hide this latency. In later simulations we use different kinds of real networks, and we can see how the latency coming from network communication can be also hidden. Besides, from these results we always get the same answer: the network has contention problems due to the pressure that the *Kilo-instruction processors* impose to the multiprocessor system.

The behaviour of the multiprocessor system together with the high contention of the interprocessor network takes us to some statements: the *Kilo-instruction processors* are very suitable to achieve sustained performance results when the memory latency is incremented, and the increase in the network contention opens a new research area in building new and better packet routers capable to support this amount of traffic.

The paper is organized as follows. In section 2, we summarize what are the *Kilo-instruction processors* and how they work. In section 3, we describe the simulator used and the benchmarks involved in our study. We show and analyze the results obtained in our simulations in section 4. In the last section we give some conclusions.

2. KILO-INSTRUCTION PROCESSORS

As we have explained, the gap between processor speed and memory latency is continuously widening. This increasing difference, called the *Memory Wall* effect [42], affects performance by stalling the processor pipeline while waiting for memory accesses. Superscalar out-of-order processors cope with these increasing latencies by having more in-flight instructions from where to extract ILP. At current trends, however, processors cannot keep up with this growing disparity, and as a result, long-latency operations are increasingly more taxing on performance.

Figure 1 is a motivating example of this problem. It shows average IPCs attained by Spec2000 floating-point applications using a 4-way processor, and supposing different memory latencies and ROB sizes. In order to understand the problem we have to pay attention just to the first four bars, those resulting from having 128 in-flight instructions, the number a modern processors can have. In this figure we can observe how the increasing memory latency can degrade the performance. With a 500 cycles latency we achieve around the 50% of degradation from a baseline memory latency of 100 cycles. With a 1000 cycles latency the 66% of degradation is reached from the same baseline. These are significant losses of performance we cannot either ignore or admit.

The explanation of what is exactly occurring in these processors is as follows. When a load instruction misses in L2 cache, the latency of the overall operation will eventually make this instruction the oldest in the processor. Since instructions commit in order, this instruction will disallow the retirement of newer instructions, which will fill entirely the re-order buffer (ROB) of the processor and halt the fetching of new instructions. In this case the performance drops because the amount of independent instructions to issue is limited due to the lack of new instructions.

The results showed in Figures 1 and 2 comes from experiments carried out in [9] and they have been obtained from a processor simulator with typical configuration, simulating basically an out-of-order 4-way processor running 300 million representative instructions from the SPEC2000 fp benchmarks. Starting from this basic configuration, other artefacts have been added to the simulator in order to implement some *Kilo-instruction processor* features.

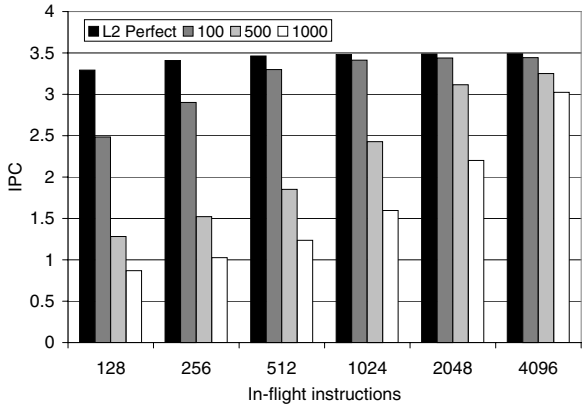


Figure 1: Effect on IPC of large memory latency for SPEC2000 fp applications

2.1 Breaking the memory wall

The solution we can think of is obvious: increase the maximum amount of in-flight instructions that the processor can handle. More instructions means that the halting will start later, thus affecting less overall performance, or not happen at all. Besides, increasing the availability of instructions to issue, increases the probability of finding independent instructions, thus also increasing IPC [16].

Numerical applications follow this behaviour very well, whereas integer applications have some problems due to branch misprediction and pointer chasing references that avoids these applications reaching a high IPC.

In order to increase the amount of in-flight instructions we must increase the capability of several resources that directly depend on their number, the most important ones being the ROB, the instruction queues and the physical registers.

Re-Order Buffer (ROB) The ROB will have as many entries as in-flight instructions we need. In the case of a 1000 cycles memory latency and a 4-way processor, and assuming we would like not to halt the processor, this would mean at least a four thousand entries ROB. Unfortunately, simply up-sizing this structure is not clear to be feasible technologically.

Instruction Queues (IQ) In-flight instructions also live

in the IQs. The number of entries in the IQ must be equal to the quantity of instructions that can be waiting for another instruction to finish due to some dependency. This number can be huge if think of instructions waiting for several long-lasting missing loads. Due to its complexity, having big IQs is also an impracticable idea.

Physical Registers The need for a renaming process [40] point to an amount of physical registers proportional to the number of in-flight instructions, since we must ensure at least a free physical register for every different destination logical register. This is, in the worst case, so many registers as ROB entries we have. This leads us to a big amount of physical registers, and this is neither simple nor inexpensive.

In Figure 1 we can see why we must not disregard the performance benefits that this increase brings. In this figure we can observe how beneficial to the performance is having more in-flight instructions. The results obtained with larger ROBs indicate us that higher memory latencies can be hidden, resulting in having similar IPCs than nowadays processors, those with little ROBs and small memory latencies. Moreover, the increasing number of in-flight instructions is capable of achieving nearly perfect memory behaviour.

2.2 Making it possible

At this point of our analysis it must be clear that simply up-sizing critical structures to tolerate memory latency is not a feasible solution. So we need a different approach to this problem.

Below, we explain the techniques employed to get an *affordable* approach. Such an approach is possible because critical resources are underutilized in present out-of-order processors as it has been showed in [7].

Checkpointing The main purpose of the ROB is to enforce in-order commit. It also allows the processor to recover the state after each specific instruction, and fortunately, recoveries are rare: they just may happen due to exceptions or mis-speculations. We can think of improving the common case [16] and execute normal instructions as if they would never except nor be part of any mis-speculation. This motivation led to [10], which proposes the first approach to substitute the need for a large ROB with a small set of microarchitectural checkpoints (*multi-checkpointing*). A different work, previously done, exploits checkpointing just to provide repair mechanisms [17]. The central idea of [10] is to commit instructions in an *out-of-order* fashion, having the possibility of freeing more resources than in normal processors. At certain points in execution checkpointing is made, so the processor is acting as if the state of the machine changed every group of executed instructions, instead of after each instruction. Returning to a precise state of the processor after an interrupt is still available; it needs only to return to the previous checkpoint to the error and proceed after that.

Slow Line Instruction Queuing There are instructions with different latency. Long latency instructions, i.e. L2 missing loads, are those that make the IQ to fill. In [9] is proposed a scalable and complexity-affordable solution to face this problem. First, with a simple mechanism long latency loads are detected. Second, these long latency instructions, and its dependent instructions, are moved into a *Slow Lane Instruction Queue (SLIQ)* where they will wait until the data comes from main memory. This mechanism allows us to keep the IQ small and fast.

Load/Store Queue Special attention must be paid to Load/Store Queues since the management of a great amount of loads and stores can be a hard problem due to the need for memory disambiguation. In [35] [29] [1] new proposals have been presented in order to overcome this problem. They essentially propose some kind of load/store filtering in conjunction with the use of two-levels structures, with the aim of storing most or all the instructions in one big structure while maintaining a shorter one to easily check the dependencies.

Ephemeral Registers In order to maintain thousands of in-flight instructions we need a lot of physical registers. A way to reduce this requirement is modifying the renaming strategy to limit the amount of live registers at any moment and therefore decrease the long-term necessities of the processor. In this way, the *Ephemeral Registers* mechanism has been developed [22]. It proposes an aggressive register recycling mechanism which combines delayed register allocation and early register recycling and, in conjunction with checkpointing, it allows the processor to non-conservatively deallocate resources. However, even though the amount of physical registers can be reduced, a mechanism to maintain the dependencies between this huge amount of in-flight instructions is indispensable. This is exactly what *Virtual Tags* [23] are used for, helping to achieve *Ephemeral Registers*.

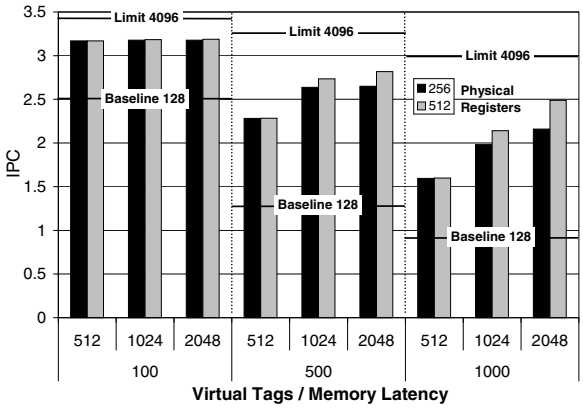


Figure 2: IPC achieved using reasonable amount of resources

In Figure 2 we show IPC results obtained from the combination of three mechanisms, those above explained, with respect to the amount of *Virtual Tags*, the memory latency and the amount of physical registers. The figure is divided

into three zones, each of them comprising the results for different memory latencies: 100, 500 and 1000 cycles. Besides, each zone has two lines that represent the performance obtained with the baseline (with 128 ROB entries) and the performance obtained by a Limit microarchitecture where all the resources have been up-sized with no constraints, the latter acting as an upper bound. Figure 2 helps us to understand how well the proposed mechanisms are behaving. First of all it allows seeing how the combination of these three orthogonal techniques work. Another conclusion that we can see is that there is still room for improvement with 1000 cycles or more. We can notice how the mechanisms nearly saturate at 500 cycles. For this latency having 1024 Virtual tags is nearly as having 2048, while in the 1000 latency zone we can see that the growth trend is far from saturating. This enforces us in the belief that *Kilo-instruction processors* with more in-flight instructions will still yield benefits.

3. SIMULATION ENVIRONMENT

In this section we are going to explain the details of the simulation environment used. First, we will describe the tools and the benchmarks we employed, and second, we will focus on the interconnection network models, since they are an important parameter in the analysis we make.

3.1 Experimental Tools and Benchmarks

An execution-driven simulator has been employed to study the impact of the processor window size in the performance of multiprocessor systems. The integration of a detailed network simulator, SICOSYS [30] together with the RSIM simulation environment [28] provides a powerful tool to emulate a complete CC-NUMA machine with state-of-the-art processors. SICOSYS allows us to take into consideration most of the VLSI network implementation details with high precision, but with much lower computational requirements than hardware-level simulators. The maximum error observed with respect to a standard hardware simulator is less than 4%, providing in all cases pessimistic estimations [30].

The microarchitecture modelled by RSIM is close to the MIPS R10000 processor. In order to simulate a *Kilo-instruction processor* we have simply up-sized the required resources: ROB, Instruction Queue, Load/Store Queue, Physical Registers, etc. Since this up-sizing lead us to reasonable results in our first approach, the effort of modelling the proposed *Kilo-instruction processor* as in [8] has not been taken into account. These results are just a little optimistic as we can see from the difference between Figure 1 and Figure 2.

Due to the limitations imposed by the complexity of the execution-driven simulated system, 16KB L1 cache and 128-KB L2 cache have been used. The benchmarks employed in this study have been tuned according to these cache sizes. Other parameters of the simulated system are shown in Table 1.

As Table 1 shows, the memory consistency model used is Release Consistency. This relaxed consistency model offers good performance results because it allows relaxing program order between all operations to different locations. This characteristic allows extensive buffering and pipelining, which can hide part of the latency of memory accesses [14].

To carry out a realistic evaluation, we fed our simulation platform with three applications selected from the SPLASH-2 suite: Radix, FFT and LU, and two applications from the SPLASH-1 suite: Mp3d, and Water. The problem size for FFT is 256K complex data points. The problem size for Water is 50.000 particles under the default geometry. In the case of Mp3d, the problem size is 43 molecules using again the default geometry. These values correspond to the problem sizes established in [36] and [41] respectively. Due to the high demand for computational resources the problem size of LU has been reduced from its default size of 512x512 to 256x256. The problem size for Radix has been also reduced from one million integer keys to a half-million using the default radix of 1024.

RSIM Processor Microarchitecture	
Issue policy	Out-of-order
Fetch/Decode/Commit width	4/4/4
Branch Predictor	2-bit agree, 2048 entries
Shadow Mappers	64
I-L1	not modeled
D-L1 size	16KB 4-way, 64B line
D-L1 latency	3 cycles
D-L2 size	128KB 4-way, 64B line
D-L2 latency	20 cycles, tag 4 cycles
Memory latency	250 and 500 cycles
Mem. & Dir. interleaving	128
Directory Buffer size	2048 entries
Coherence Protocol	MSI
Consistency Protocol	RC
Bus size/latency	512 bits/3 cycles
MSHR size/coalesc.	32K/16
Integer General Units	4 (lat/rep 1/1)
Integer Mult/Div	(lat/rep 3/1 and 13/13)
FP Functional Units	2 (lat/rep 3/1)
Address Generation Units	2
IQ, LQ, Physical Registers	proportional to ROB

Table 1: Computation node parameters of the simulated system

3.2 Interconnection Network Models

In the multiprocessor design arena the *Memory Wall* reveals as an exacerbated problem. Remote memory accesses exhibit larger latencies and they have to share network bandwidth with coherency and synchronization traffic. Modern medium-scale CC-NUMA multiprocessors use large caches, directory-based coherency protocols and switched interconnection networks. Small-scale UMA multiprocessors also use local caches but, in contrast, snooping coherency protocols. A common shared bus have been traditionally considered sufficient to manage the data traffic generated by these small systems. Nevertheless, the CC-NUMA architecture success together with scalability and expandability issues are imposing a distributed shared-memory style even in small multiprocessors [25] [12]. Thus, switched networks are nowadays present in multiprocessor systems of any size.

We are going to consider several interconnection networks within this research. Firstly, an unrealistic network design will be used in order to establish an upper bound achievable performance. This network, which we will denote as *ideal network*, will consist of a two-stages pipelined cross-

bar operating at the processor frequency. Secondly, a set of realistic network designs will be considered for establishing practicable performance gains.

Topology, wire technology and router architecture will define a realistic network design. Multiple interconnection topologies have been proposed in last decades. Meshes, Tori and Hypercubes are among the most popular. Hypercube architectures have progressively been replaced by lower degree topologies and nowadays, Tori and Meshes still compete in the design of the most recent high-end multiprocessors [26] [3]. Although Meshes are easier to implement, at least from a functional point of view, Tori clearly outperform Meshes. Hence, we focus our attention on bi-dimensional Tori in which four input/output links are used for communicating network nodes.

A quick look at wire technology will configure a first snapshot of the interconnection network. The physical characteristics of current wire technologies establish the network operation cycle, which is constrained by signal propagation delays. Channel pipelining, consisting on having multiple in-flight data units over the wire has been used for increasing network throughput. Nevertheless, the actual network frequency is dictated by the inherent physical phenomena associated with the employed wire technology. For example, the upper bound on network frequency operation for Alpha 21364 based multiprocessors has been established at 800MHz [25]. These systems are physically highly-coupled architectures because they are backplane-based designs and their processors integrate its packet router on-chip. Other more flexible architectures such as SGI Origin [12] or last IBM SP series [39], suffer from heavier restrictions. The *NUMALink4* employed by SGI operates at 200 MHz and the wire technology employed by IBM SP series imposes a network operation of 125 MHz.

Besides topology and wire technology, router architecture completes an interconnection network design. We will employ two realistic router architectures in this paper. Let us to describe their most important features. The simplest router uses deterministic Dimension Order Routing (DOR) and Bubble Flow Control for deadlock avoidance [4]. This router, which is denoted as BDOR, consists of a five-stages pipeline and employs FIFO queues located at the input ports. The second router under study also relies on Bubble Flow Control for deadlock avoidance but, in contrast, it uses adaptive routing. This router, denoted as BADA, has a six-stages pipeline and two virtual channels (implemented as FIFO queues) per input port [32]. It should be noticed that this routing mechanism has been implemented in the Torus network used by the IBM BlueGene/L supercomputer [26].

Taking into account the above mentioned wire technology restrictions we have set the router operation frequency at 666 MHz in all the cases. In addition, links will be bidirectional supporting 32 data bits in each direction. It means in turn, that the phit size managed by the network (number of bits transmitted in parallel per port and network cycle) is set to 32 bits. Coherency protocol packets have been implemented to be 16 bytes long.

4. RESULTS AND ANALYSIS

In this section, we show and analyze the results obtained from our simulations. The first subsection introduces the idea of having *Kilo-instruction* based multiprocessors, and the second subsection describes what is going on in this kind of systems.

Our study is based on assuming memory latencies of 250 and 500 cycles. The former because is a reasonable number for current systems, and the latter because we want to go a step further on and take into account future latencies. In each case we will indicate what latency is used.

4.1 Kilo-instruction Based Multiprocessors

At this point, we know that *Kilo-instruction processors* report improvements to uniprocessor systems, but now is the moment to analyze what *Kilo-instruction processors* can contribute to the performance of a multiprocessor system. Hence, we are going to evaluate a *Kilo-instruction* based multiprocessor system. This system will be made of 16 processors. Each processor employed will be that described in Table 1 and the network will be the *ideal network* cited in section 3.2. These first results will also put light on the suitability of the benchmarks we are going to use, those taken from the SPLASH and the SPLASH-2 suites and depicted in section 3.1.

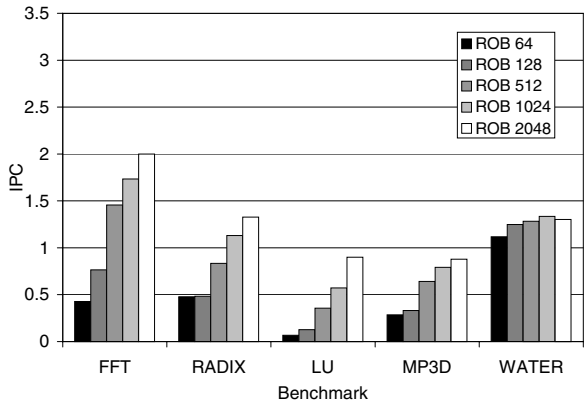
Figures 3a and 3b gives us the results for the experiment above described, first using just one processor and then using the mentioned 16 processors system. The memory latency involved is 500 cycles. In this figure, we can observe, for each benchmark, how the increasing amount of in-flight instructions, starting from 64 (the baseline), influences the resulting IPC.

The results are clear, as in the results with one processor we also get performance improvements by using 16 *Kilo-instruction processors*. These improvements are significant in both cases. Observing the results for one processor, there is an improvement between the 170% and the 370% approximately for all the benchmarks, using a 2048 entries ROB over the baseline. The exceptions are the LU benchmark that reach a 1200% of improvement, and the Water benchmark that reach only a 17%.

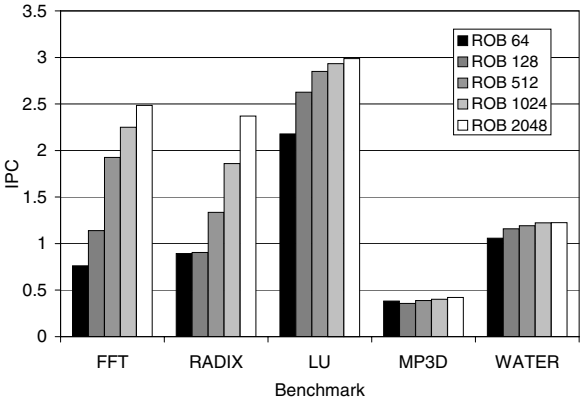
Having a look to the results for 16 processors we can also appreciate improvements. We have limited improvements for LU, Mp3d and Water benchmarks, between 10% and 37%, and still considerable improvements for Radix and FFT benchmarks, 166% and 227% respectively. The reason of having minor improvements when running on 16 processors lies in the fact that there are less L2 cache missing loads, since we divide the same problem size between more processors. Another consequence is that we obtain higher IPCs running the benchmarks on 16 processors than running them on one processor.

Furthermore, in Figures 3a and 3b we can observe that two benchmarks behave in a different way. The Mp3d benchmark shows poor improvements running on 16 processors and using big ROBs. This happens due to an unfortunate implementation of the application. Mp3d uses a simple block to access the global counter of particle collisions, and employs many synchronization barriers, so the potential performance is degradated by waits. On the other hand, the Water benchmark have little improvements running on 1 processor and running on 16 processors, due to the high branch misprediction rate (more than 10%) and to the size of its basic block (20 instructions). In some way Water is acting as an integer application, and this is not what we expected.

Once studied the set of proposed benchmarks, we have



(a) 1 processor



(b) 16 processors

Figure 3: IPC obtained by having different number of in-flight instructions for some SPLASH and SPLASH-2 programs, assuming an ideal network and a memory latency of 500 cycles

demonstrated that the performance increase obtained by *Kilo-instruction processors* in uniprocessor systems is also achieved in multiprocessor systems, which is the expected result. Besides, we have demonstrated that the benchmarks Mp3d and Water are not useful to the purposes of our study.

4.2 The Impact of the Interconnection Network

In this subsection we analyze the behaviour of *Kilo-instruction* based multiprocessors when using realistic interconnection networks. In particular, we will consider BDOR and BADA routers arranged as bi-dimensional 4x4 Torus networks. We will focus only on Radix, FFT and LU applications as they exhibit natural characteristics to be conveniently exploited by a parallel computer. In order to further illustrate our analysis, we will also run these applications on a multiprocessor having and ideal network. In this way, we can clearly establish a performance upper bound.

We start our analysis considering a high-end microprocessor running at 4 GHz with a state-of-the-art 64 entries ROB and DRAM memory latency of 500 processor cycles. We consider four experiments: ideal network and memory (zero latency), only ideal network, BADA and BDOR networks. The different performance results obtained are shown in Figure 4.

Under these conditions, the impact on system performance of the interconnection network is almost negligible. By using an ideal network, it is only possible to achieve a performance gain under 10% in the most favouring case. This is an expected behaviour. On the one hand, the DRAM latency dominates over the remote accesses. Actually, the base-latency in a realistic network is only around 35 processor cycles. On the other hand, the network does not suffer from congestion because there are few in-flight remote access instructions and hence, there is a low volume of traffic to be managed.

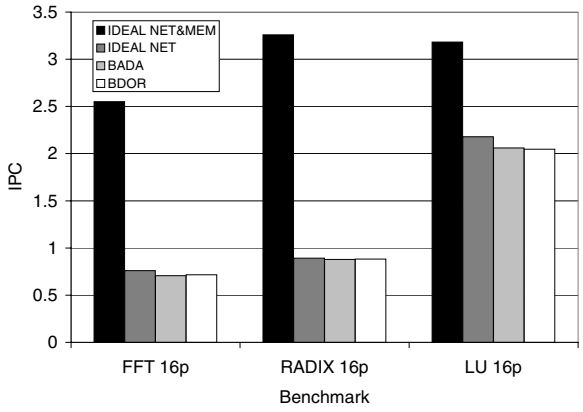


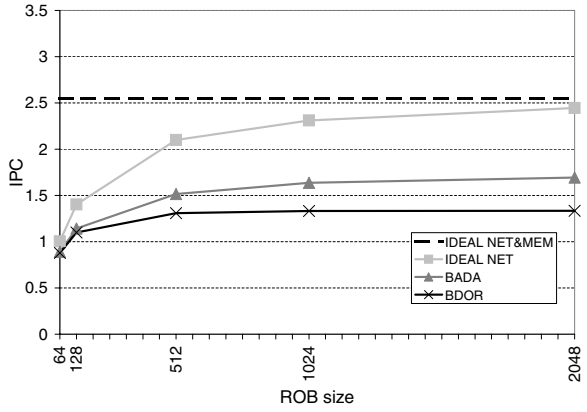
Figure 4: How the type of network impact the performance with 64 in-flight instructions and a memory latency of 500 cycles

The previous scenario totally changes when the number of ROB entries is progressively incremented. Figures 5a, 5b and 5c show the performance achieved for ROB sizes of 128, 512, 1024 and 2048 entries and memory latency of 250 processor cycles. Figures 6a, 6b and 6c show the same experiment using a memory latency of 500 processor cycles, so we can compare later the utilization of different latencies.

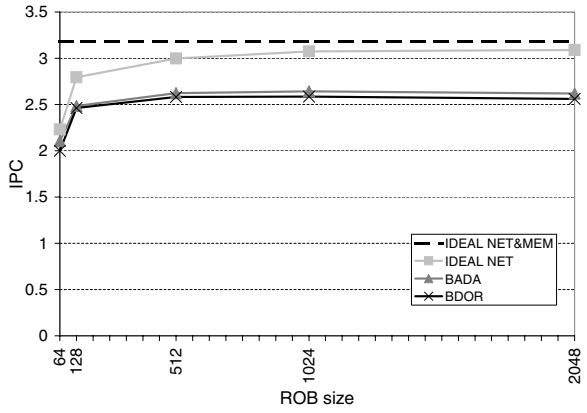
Although the results are quite similar, we will pay attention just to Figures 6a, 6b and 6c, those with 500 cycles of memory latency, for the following analysis.

We can see that the system performance for a 2048 entries ROB with an ideal network is quite close to that achieved when using a combination of an ideal network and an ideal memory. At first glance, this observation proves two facts. First, the suitability of big instruction windows for tolerating memory latencies. Second, the interconnection network becomes in the system bottleneck when augmenting the ROB size.

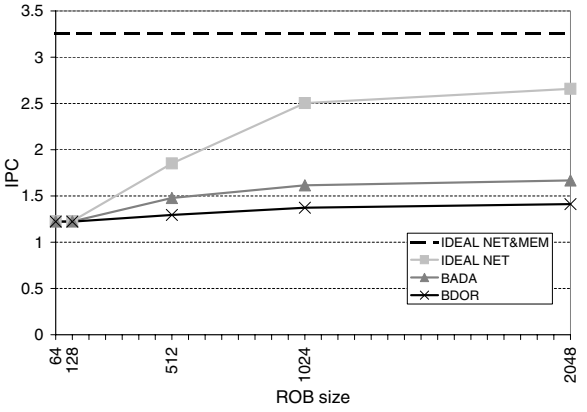
Under a realistic scenario, the ability to achieve higher values of IPC by using big ROBs will depend on the interconnection network effectiveness. With medium ROB sizes the performance gains provided by BDOR and BADA networks, although noticeable, are clearly below of those obtained by using an ideal network. Augmenting the ROB size above 512 entries returns nearly negligible profits. When using a 2048 entries ROB, the gap between BADA and BDOR networks is around 15%, which shows how adaptive routing translates into better performance. Nevertheless, the gap



(a) FFT

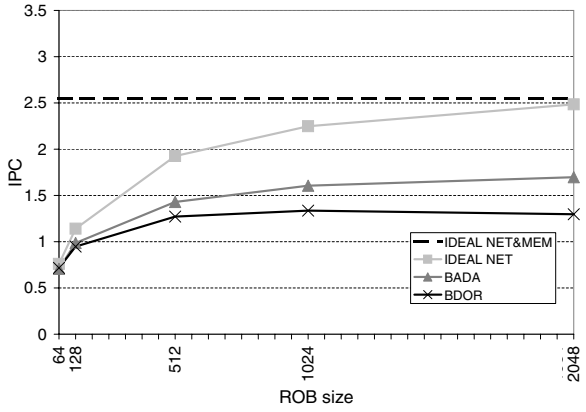


(b) LU

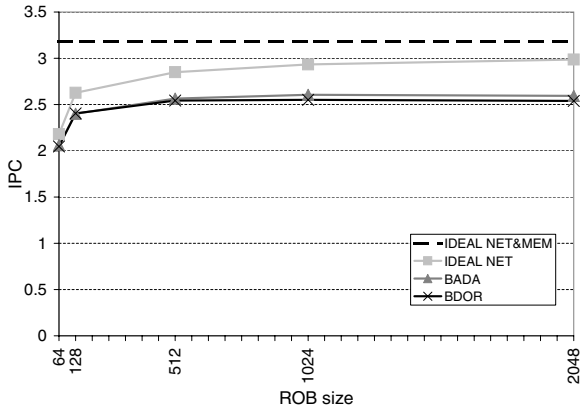


(c) Radix

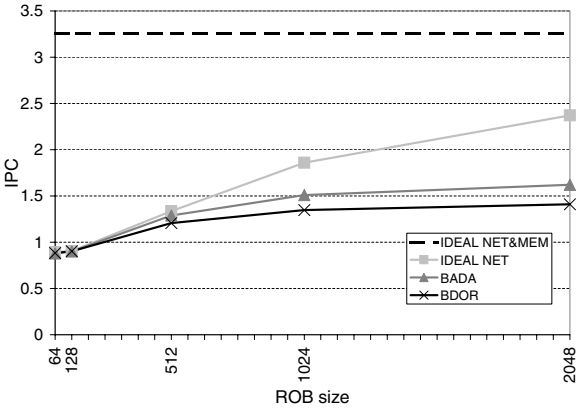
Figure 5: Performance of three SPLASH-2 benchmarks using different types of network, assuming a memory latency of 250 cycles



(a) FFT



(b) LU



(c) Radix

Figure 6: Performance of three SPLASH-2 benchmarks using different types of network, assuming a memory latency of 500 cycles

between BADA and an ideal network is still around 40%, which tell us that, by using more evolved interconnection networks, there are opportunities for important performance gains.

The previous analysis clearly shows how the interconnection network constitutes, in our case, the system bottleneck, which avoids achieving higher performance. As stated before, the use of standard instruction windows generates a relatively low traffic volume and the interconnection network can cope with it. Nevertheless, bigger instruction windows implies higher rates of remote accesses which in turn, brings a higher pressure over the network. In such situations, contention among packets arises and consequently, remote accesses exhibit longer latencies. During some ex-

ecution phases, computing nodes inject such a volume of traffic that the network enters on a saturated state being it unable to drain packets at the same rate as they are generated. In this degraded scenario, remote access latencies exponentially grow up. In consequence, although there are more in-flight remote accesses they suffer from enormous latencies. In short, we can see that augmenting the instruction window size can beneficiate local memory accesses but is tremendously dangerous for the remote ones. This is the reason why BADA and BDOR show different behaviour although they have similar base latencies. BADA clearly outperforms BDOR because adaptive routing is able to mitigate, up to some extent, the negative effects of network contention.

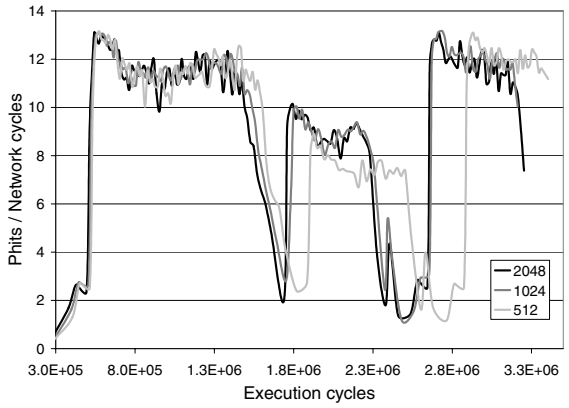
Now, comparing set of Figures 5 with Figures 6 we can clearly deduce that with our *Kilo-instruction based multiprocessor* we can see more defined IPC increments when using a latency of 500 cycles. In most cases, with 500 cycles and little ROBs the IPC is lower than using 250 cycles, and that is logical. With bigger ROBs the IPC is practically the same for both latencies using realistic interconnection networks, and that is a good result if we think that memory latency has been doubled. Only with ideal network the IPC for bigger ROBs is greater for 250 cycles than 500 cycles, but the difference is little so the results are fairly reasonable.

In the rest of the section, from Figure 7 to Figure 9, the memory latency of 250 cycles is used despite of losing some benefit from having a higher latency as we have seen above.

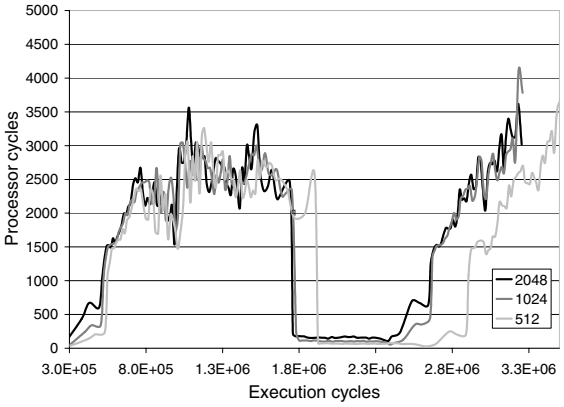
In order to have a better insight of this phenomenon we focus on the network behaviour when a Radix application is executed for ROB sizes of 512, 1024 and 2048 entries. Figure 7 dynamically shows the traffic load and packet latencies supported by the network. In some execution phases, as the first and third ones, which correspond to random communication patterns, the network throughput is around 14 phits consumed every network cycle. It must be noticed that the maximum achievable throughput corresponds to 16 phits per network cycle. In such execution phases the network is clearly saturated regardless the window size. Nevertheless, in the central phase which employs an all-to-one communication pattern, the network is able to manage a slightly higher load when using a 1024 entries ROB instead of a 512 entries ROB which translates into a superior performance when consider the whole application. Notwithstanding, no further gains are observed by using a bigger ROB.

Having a look to Figure 7b we can observe how the network contention negatively impacts on the remote access latencies. As we increment the number of in-flight instructions, remote accesses are also incremented but the network is unable to cope with them. In such cases, some part of the traffic generated by the computing nodes cannot be injected in the network provoking a clear increment of the latency experienced by packets. For a best observation of this effect, Figure 8 records the average packet latency at the beginning of the application execution. The bigger the window size the higher the number of in-flight instructions and hence, a superior tolerance of the slower memory accesses. Nevertheless, this effect is counterbalanced by a high packet population count, which saturates the network avoiding the achievement of the expected performance gains.

To end this section, we are going to have a quick look at the performance behaviour when scaling up the system size. Figure 9 shows a FFT running on an 8x8 BADA Torus.



(a) Throughput



(b) Network latency

Figure 7: Load and latency caused by messages running Radix and using different ROB sizes and a memory latency of 250 cycles

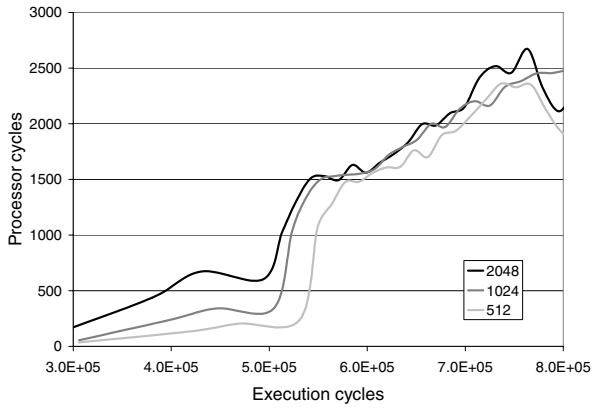


Figure 8: Detailed network latency of the initial execution phase of Radix

When comparing a realistic network with an ideal one there is a noticeable 50% degradation even when using standard ROBs of 64 entries. This gap grows up when increasing the ROB because when we scale the system size the maximum

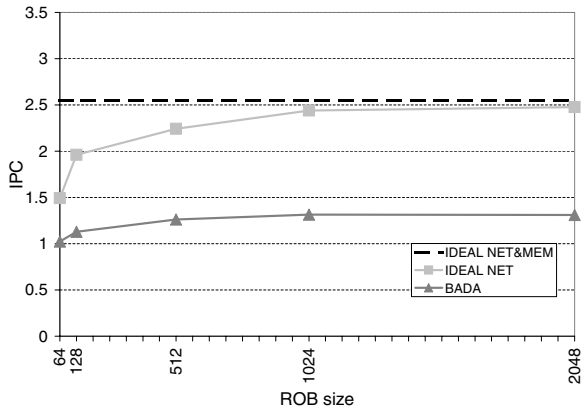


Figure 9: The network impact with 64 processors and running FFT, assuming a memory latency of 250 cycles

achievable throughput decreases and the average latency increases. This decreasing in throughput is not manifested in terms of phits managed by network cycle but in terms of a lower rate of phits injected by individual computing nodes. In this case, the use of ROBs having more than 512 entries does not provide further performance gains due to the higher network contention.

5. CONCLUSIONS

In order to tolerate increasing memory latencies, a large number of in-flight instructions must be maintained. This is also true for multiprocessors, where, besides, we can encounter more sources of latency: those coming from memory accesses and those coming from network communication.

This paper provides another evidence of the potential of the *Kilo-instruction processors*, added to those provided in previous works that we have already cited along this paper. Its application in the multiprocessors environment is not just viable but also beneficial in two ways. On the one hand, we have demonstrated that *Kilo-instruction processors* are an appropriate mechanism to hide the latencies coming from different sources. On the other hand, we show the relevance of the interconnection network and, therefore, that we have to put more effort in creating packet routers oriented to support high traffic levels.

This work is a first evaluation of what we call *Kilo-instruction based multiprocessors*. We provide evident results of how beneficial can be combining knowledge coming from *Kilo-instruction processors* and *Multiprocessors*. This way, we are opening a new direction where more research is needed and known and new ideas can be applied both at the processor and interconnection network levels.

6. ACKNOWLEDGMENTS

This work was supported by the Ministry of Science and Technology of Spain under contracts TIC-2001-0995-C02-01 and TIC-2001-0591-C02-01. The authors would like to thank Francisco J. Cazorla and Ayose Falcón for their valuable comments.

7. REFERENCES

[1] H. Akkary, R. Rajwar, and S. Srinivasan. Checkpoint Processing and Recovery: Towards Scalable Large Instruction Window Processors. *Proceedings of the 36th Annual ACM/IEEE Intl. Symposium on Microarchitecture*, pages 423–434, December 2003.

[2] J.-L. Baer and T.-F. Chen. An Effective On-chip Preloading Scheme to Reduce Data Access Penalty. *In Proceedings of Supercomputing '91*, pages 176–186, November 1991.

[3] W. Camp and J. Tomkins. The Red Storm Computer Architecture and its Implementation. *Conference on High-Speed Computing*, April 2003.

[4] C. Carrion, R. Beivide, J. Gregorio, and F. Vallejo. A Flow Control Mechanism to Avoid Message Deadlock in K-ary N-cube Networks. *Fourth International Conference on High Performance Computing*, pages 322–329, December 1997.

[5] R. Chappell, J. Stark, S. Kim, S. Reinhardt, and Y. Patt. Simultaneous Subordinate Microthreading (SSMT). *Proceedings of the 26th Annual Intl. Symposium on Computer Architecture*, pages 186–195, May 1999.

[6] J. D. Collins, D. M. Tullsen, H. Wang, and J. P. Shen. Dynamic Speculative Precomputation. *Proceedings of the 34th Annual ACM/IEEE Intl. Symposium on Microarchitecture*, pages 306–317, December 2001.

[7] A. Cristal, J. F. Martinez, J. Llosa, and M. Valero. A Case for Resource-conscious Out-of-order Processors. *In IEEE TCCA Computer Architecture Letters*, 2, October 2003.

[8] A. Cristal, D. Ortega, J. Llosa, and M. Valero. Kilo-instruction Processors. *Proceedings of the 5th International Symposium on High Performance Computing (invited paper)*, pages 10–25, October 2003.

[9] A. Cristal, D. Ortega, J. Llosa, and M. Valero. Out-of-Order Commit Processors. *Proceedings of the 10th Intl. Conference on High Performance Computer Architecture*, February 2004.

[10] A. Cristal, M. Valero, A. Gonzalez, and J. Llosa. Large Virtual ROBs by Processor Checkpointing. Technical Report UPC-DAC-2002-39, Universidad Politécnica de Cataluña, July 2002.

[11] M. Dubois and Y. Song. Assisted Execution. Technical Report CENG 98-25, Department of EE-Systems, University of Southern California, October 1998.

[12] M. Galles. Spider: A High-Speed Network Interconnect. *IEEE Micro*, 17(1):34–39, Jan.-Feb. 1997.

[13] K. Gharachorloo, A. Gupta, and H. Hennessy. Hiding Memory Latency Using Dynamic Scheduling in Shared-memory Multiprocessors. *Proceedings of the 19th Annual Intl. Symposium on Computer Architecture*, pages 22–33, May 1992.

[14] K. Gharachorloo, A. Gupta, and J. Hennessy. Performance Evaluation of Memory Consistency Models for Shared-memory Multiprocessors. *Proceedings of the 4th Intl. Conference on Architectural Support for Programming Languages and Operating Systems*, pages 245–257, April 1991.

[15] C. Gniady, B. Falsafi, and T. Vijaykumar. Is SC + ILP = RC? *Proceedings of the 26th Annual Intl.*

Symposium on Computer Architecture, pages 162–171, May 1999.

[16] J. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, San Mateo, California, 3rd edition, 2003.

[17] W. Hwu and Y. Patt. Checkpoint Repair for Out-of-Order Execution Machines. pages 18–26, December 1987.

[18] D. Joseph and D. Grunwald. Prefetching Using Markov Predictors. *Proceedings of the 24th Annual Intl. Symposium on Computer Architecture*, pages 252–263, June 1997.

[19] T. Karkhanis and J. Smith. A Day in the Life of a Cache Miss. *2nd Annual Workshop on Memory Performance Issues (WMPI)*, June 2002.

[20] M. Karlsson, F. Dahlgren, and P. Stenstrom. A Prefetching Technique for Irregular Accesses to Linked Data Structures. *Proceedings of the 6th Intl. Conference on High Performance Computer Architecture*, pages 206–217, January 2000.

[21] A. Klaiber and H. Levy. An Architecture for Software-Controlled Data Prefetching. *Proceedings of the 18th Annual Intl. Symposium on Computer Architecture*, pages 43–53, May 1991.

[22] J. Martinez, A. Cristal, M. Valero, and J. Llosa. Ephemeral Registers. Technical Report CSL-TR-2003-1035, Cornell Computer Systems Lab, 2003.

[23] T. Monreal, A. Gonzalez, M. Valero, J. Gonzalez, and V. Vinals. Delaying Physical Register Allocation Through Virtual-Physical Registers. *Proceedings of the 32nd Annual ACM/IEEE Intl. Symposium on Microarchitecture*, pages 186–192, November 1999.

[24] T. Mowry, M. Lam, and A. Gupta. Design and Evaluation of a Compiler Algorithm for Prefetching. *Proceedings of the 5th Intl. Conference on Architectural Support for Programming Languages and Operating Systems*, pages 62–73, October 1992.

[25] S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb. The Alpha 21364 Network Architecture. *In Proceedings of Hot Interconnects 9*, August 2001.

[26] N.R. Adiga et al. An Overview of the BlueGene/L Supercomputer. *In Proceedings of Supercomputing '02*, November 2002.

[27] D. Ortega, E. Ayguade, J.-L. Baer, and M. Valero. Cost-Effective Compiler Directed Memory Prefetching and Bypassing. *Proceedings of the 11th Intl. Conference on Parallel Architectures and Compilation Techniques*, pages 189–198, September 2002.

[28] V. Pai, P. Ranganathan, and S. Adve. RSIM: An execution-Driven Simulator for ILP-Based Shared-Memory Multiprocessors and Uniprocessors. *IEEE TCCA Newsletter*, 35(11):37–48, October 1997.

[29] I. Park, C. Ooi, and T. Vijaykumar. Reducing Design Complexity of the Load/Store Queue. *Proceedings of the 36th Annual ACM/IEEE Intl. Symposium on Microarchitecture*, pages 411–422, December 2003.

[30] V. Puente, J. Gregorio, and R. Beivide. SICOSYS: An Integrated Framework for Studying Interconnection Networks in Multiprocessor Systems. *Proceedings of the 10th Euromicro Workshop on Parallel and Distributed Processing*, pages 360–368, January 2002.

[31] V. Puente, J. Gregorio, R. Beivide, and C. Izu. On the Design of a High-Performance Adaptive Router for CC-NUMA Multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 14(5), May 2003.

[32] V. Puente, C. Izu, J. Gregorio, R. Beivide, and F. Vallejo. The Adaptive Bubble Router. *Journal on Parallel and Distributed Computing*, 61(9):1180–1208, September 2001.

[33] P. Ranganathan, V. Pai, and S. Adve. Using Speculative Retirement and Larger Instruction Windows to Narrow the Performance Gap between Memory Consistency Models. *In Proceedings of the 9th Symposium on Parallel Algorithms and Architectures*, June 1997.

[34] A. Roth and G. S. Sohi. Speculative Data-Driven Multithreading. *Proceedings of the 7th Intl. Conference on High Performance Computer Architecture*, pages 37–50, January 2001.

[35] S. Sethumadhavan, R. Desikan, D. Burger, C. Moore, and S. Keckler. Scalable Hardware Memory Disambiguation for High ILP Processors. *Proceedings of the 36th Annual ACM/IEEE Intl. Symposium on Microarchitecture*, pages 399–410, December 2003.

[36] J. Singh, W. Weber, , and A. Gupta. SPLASH: Stanford Parallel Applications for Shared-Memory. *Computer Architecture News*, 20(1):5–44, March 1992.

[37] A. Smith. Cache Memories. *Computing surveys*, 14(3):473–530, September 1982.

[38] Y. Sohilin, J. Lee, and J. Torrellas. Using a User-Level Memory Thread for Correlation Prefetching. *Proceedings of the 29th Annual Intl. Symposium on Computer Architecture*, pages 171–182, May 2002.

[39] C. Stunkel, J. Herring, B. Abali, and R. Sivaram. A New Switch Chip for IBM RS/6000 SP Systems. *In Proceedings of Supercomputing '99*, November 1999.

[40] R. Tomasulo. An Efficient Algorithm for Exploiting Multiple Arithmetic Units. *IBM Journal of Research and Development*, (11):25–33, January 1967.

[41] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. *Proceedings of the 22nd Annual Intl. Symposium on Computer Architecture*, pages 24–36, June 1995.

[42] W. Wulf and S. McKee. Hitting the Memory Wall: Implications of the Obvious. *Computer Architecture News*, 23(1):20–24, March 1995.

[43] C. Zilles and G. Sohi. Execution-based Prediction using Speculative Slices. *Proceedings of the 28th Annual Intl. Symposium on Computer Architecture*, pages 2–13, July 2001.