

# Exploiting Execution Locality with a Decoupled Kilo-Instruction Processor

Miquel Pericàs<sup>1,2</sup>, Adrian Cristal<sup>2</sup>, Ruben González<sup>1</sup>, Daniel A. Jiménez<sup>3</sup>,  
and Mateo Valero<sup>1,2</sup>

<sup>1</sup> Computer Architecture Department, Technical University of Catalonia (UPC)  
Jordi Girona, 1-3, Mòdul D6 Campus Nord, 08034 Barcelona (SPAIN)  
{mpericas,adrian,gonzalez,mateo}@ac.upc.edu

<sup>2</sup> Barcelona Supercomputing Center (BSC)

Jordi Girona, 29, Edifici Nexus-II Campus Nord, 08034 Barcelona (SPAIN)

<sup>3</sup> Department of Computer Science, The University of Texas at San Antonio (UTSA)  
Science Building, One UTSA Circle, San Antonio, TX 78249-1644 (USA)  
djimenez@acm.org

**Abstract.** Overcoming increasing memory latency is one of the main problems that microprocessor designers have faced over the years. The two basic techniques introduced to mitigate latencies are *caches* and *out-of-order* execution. However, neither of these solutions is adequate for hiding off-chip memory accesses in the order of 200 cycles or more. Theoretically, increasing the size of the instruction window would allow much longer latencies to be hidden. But scaling the structures to support thousands of in-flight instructions would be prohibitively expensive.

However, the distribution of instruction issue times under the presence of L2 cache misses is highly correlated. This paper describes this phenomenon of *Execution Locality* and shows how it can be exploited with an inexpensive microarchitecture consisting of two linked cores. This *Decoupled Kilo-Instruction Processor* (D-KIP) is very effective in recovering lost potential performance. Extensive simulations show that speed-ups of up to 379% are possible for numerical benchmarks thanks to the exploitation of impressive degrees of Memory-Level Parallelism (MLP) and the execution of independent instructions in the shadow of L2 misses.

## 1 Introduction

The memory wall problem [1] is one of the main causes for the low instructions-per-cycle (IPC) rates that current architectures are able to achieve. In general, to overcome memory latencies, two very successful techniques were introduced in high performance microarchitectures: Memory Caches and Out-of-Order Processing.

Caches [2,3] exploit the locality in data access patterns exhibited by programs, giving the cost advantage of a large and cheap main memory with the low latency of a small and expensive memory. This technique has proven very successful and is used in all current microprocessors.

Out-of-Order execution allows instructions in the instruction window to execute in dataflow order. The required hardware is expensive and must be designed to be small enough so as to not impair the cycle time. Out-of-order execution allows the processor to continue executing while an earlier instruction is blocked. However, the technique is limited by the size of the instruction queues and, for the case of memory access latencies, this technique is not sufficient.

Recently, much research has proposed replacing the most critical microarchitectural structures with distributed structures that can scale much farther. Kilo-Instruction processors are such an approach. They provide an effective solution for overcoming the memory wall problem. A Kilo-Instruction processor does not need to stall under the presence of a cache miss. This has two benefits:

- Distant Load Misses can be pre-executed. This results in higher exploitation of MLP.
- Independent instructions can be executed in the shadow of a L2 Cache miss. This allows the architecture to take profit of distant parallelism.

However, in its current incarnation, the Kilo-Instruction Processor still has shortcomings in terms of its complexity. For example, the management of registers still lacks a low-complexity solution. It would be desirable to obtain simpler schemes that do not involve large amounts of complexity.

This paper proposes a new complexity-effective implementation of the KILO based on the novel concept of *Execution Locality*. Execution locality is a concept derived from data access locality and memory hierarchies. The idea is that instructions can be grouped at runtime into clusters depending on their `decode`→`issue` distance. Instructions that issue soon are said to have high execution locality. This allows building a kilo-instruction processor using a processor hierarchy in which different processors handle instructions belonging to different locality groups. The architecture presented in this paper proposes using two superscalar processors linked by instruction/register buffers (see Figure 1). The first processor (*Cache Processor*, CP) is small and fast, and it executes instructions that have high execution locality. The second processor (*Memory Processor*, MP) can be simple and wide, and it executes the remaining low-locality instructions. The proposal reuses some concepts recently introduced to scale the processor structures but reorganizes the processor architecture in a novel fashion resulting in a decoupled processor that combines scalability, sustained complexity, and high performance.

The main contribution of this paper is twofold:

1. The introduction of the concept of *Execution Locality* and its evaluation.
2. The proposal of a *Decoupled Kilo-Instruction Processor*, a low-complexity architecture designed to exploit Execution Locality.

This paper is organized as follows. In Sect. 2 a motivation study is presented that will provide background to the possible improvements in execution. Section 3 will provide a discussion of execution locality along with an initial evaluation of execution behavior. Using these concepts it will be possible to begin a description

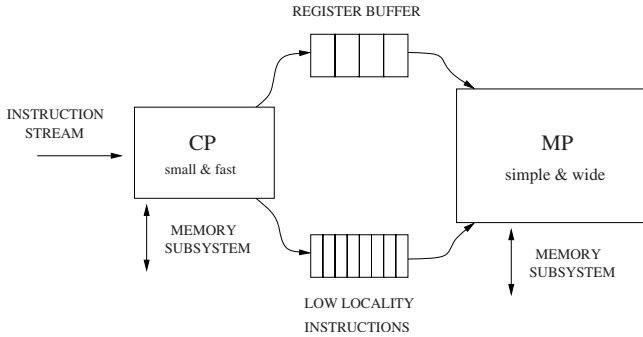


Fig. 1. 2-Level Decoupled Processor

of the decoupled Kilo-Instruction Processor in Sect. 4. The paper continues with a description of the simulation framework and the evaluation results in Sect. 5. Sections 6 and 7 complete the proposal with a summary of related works and the conclusions, respectively.

## 2 Motivation

To evaluate the impact of the memory wall, a workload consisting of all benchmarks of the SPEC<sub>FP</sub>2000 benchmark suite is run on a simulator that models a typical 4-way out-of-order processor. The processor configurations are such that they are constrained only by the size of the ROB. As branch predictor the perceptron predictor [4] is used. Several processor configurations with ROB sizes from 32 to 4096 entries are evaluated. For each configuration, six different memory subsystems are tested. Three of these subsystems contain ideal caches. The IPC results are shown in Fig. 2. The sizes of the caches are 32KB for the first-level cache and 512KB for the second-level cache. Associativity is 4 in both cases and the access latencies are 1 and 10 cycles, respectively.

The figure shows how for numerical codes the possible gains of using a large-window processor are very large. Instructions are almost never discarded due to misspeculation and many independent instructions can be executed in the shadow of a cache miss, as predicted by [5]. This allows a large instruction window to much better hide the memory latencies. It can be seen that, at 4096 in-flight instructions, the processor configuration is almost insensitive to the memory latency. Only for the large 1000-cycle latency can a substantial IPC drop be observed. The reason is that the ROB is too small to hide latencies derived from two-level load chains, which is not the case for the 400-cycle configuration, where the cumulative latency of 800 cycles can still be hidden. The idea here is that performance close to perfect memory is feasible if an instruction window that supports thousands of in-flight instructions can be built.

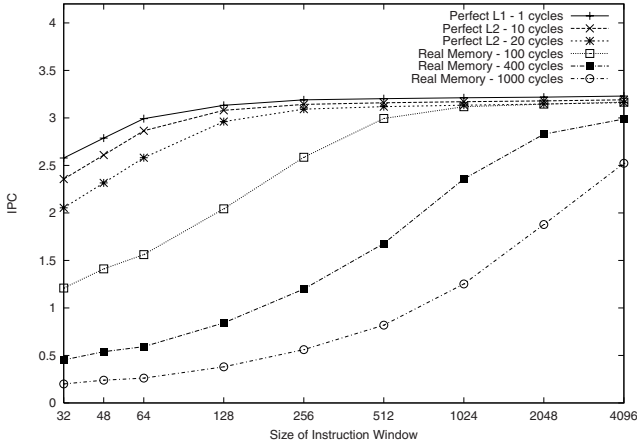


Fig. 2. Impact of ROB size and Memory Subsystem on SPEC FP

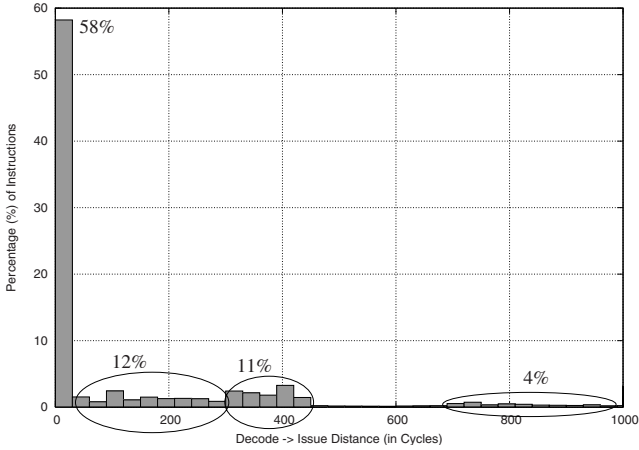
### 3 Execution Locality

In a modern microprocessor, execution is data-driven. Once an instruction has all its operands computed, this instruction will execute within a small amount of cycles. Therefore, it should be expected that a large correlation exists between the number of cycles an instruction waits until *issue* and the presence of long-latency events in its dependency tree.

Figure 3 depicts the distribution of cycles that instructions remain in the instruction queues in a 4-way processor with 400 cycles of main memory access latency. These values have been obtained for all committed instructions in SPEC FP. The distribution shows that most instructions (about 70%) do not depend on outstanding L2 cache misses while the remaining do depend directly on one or multiple L2 cache misses. A large part of these (11%) depend on a single cache miss and remain around 400 cycles in the queues. A smaller quantity (4%) depend on two caches misses. The remaining 15% of instructions belong either to issue clusters that depend on more than two L2 cache misses or are irregular instructions whose issue times do not belong clearly to one cluster or another.

This figure illustrates the major concept that this paper introduces: the *Execution Locality*. Similarly to data accesses, it is possible to classify all instructions using locality concepts. The phenomenon of execution locality classifies instructions based on their issue latency. Those instructions that will issue soon have a high degree of execution locality, while those that depend on long-latency memory accesses are classified as having low execution locality.

As will now be seen, execution locality can be used to reorganize the internal microarchitecture in a very efficient way such that a different core processes instructions belonging to different execution locality groups.



**Fig. 3.** Average distance between decode and issue for SPECFP

## 4 A Decoupled Kilo-Instruction Processor

The concepts of *execution locality* and the analysis presented in the previous section enable the efficient implementation of a *Kilo-Instruction Processor* using a heterogeneous decoupled dual core approach, a simplified version of which has been already shown in Fig. 1.

Some techniques used in the Kilo-Instruction processor are used in the proposal. They will be briefly summarized in Section 6. The next section will focus on the new concepts introduced by the Decoupled Kilo-Instruction Processor (D-KIP).

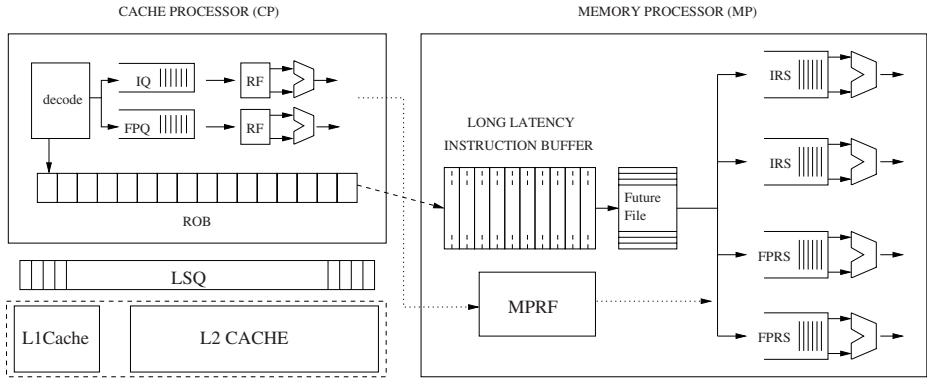
### 4.1 Heterogeneous Dual Core for Kilo-Instruction Processing

This section describes the basic microarchitecture of a D-KIP. The D-KIP is the first proposal designed from the ground up to provide an efficient solution that exploits *execution locality*. Here we will be giving a conceptual description of the D-KIP. A more detailed description is outside the scope of this paper.

Using the concepts of decoupling and execution locality this paper proposes a new and interesting approach to kilo-instruction processing: a two-level processor hierarchy. This proposal is, to the best of our knowledge, the first heterogeneous dual-core architecture aimed at single-thread performance, and the first to exploit execution locality concepts.

The D-KIP consists of two simple superscalar cores linked by a long-latency instruction buffer (LLIB). The LLIB is a FIFO structure that temporarily holds instructions that have low execution locality (ie, depend on a miss) while they are waiting for the L2 cache miss to complete. In combination both processors act as a Kilo-Instruction Processor. The first core will be referred to as the *cache processor* (CP) and the second core is the *memory processor* (MP). The reason to

call the first processor *cache processor* is that its structures are sized to deal only with latencies resulting from cache hits, no further. A more detailed microarchitecture is shown in Fig. 4. There are actually many possible implementations. We will present only the most straight-forward implementation here.



**Fig. 4.** Microarchitecture of a decoupled Kilo-Instruction Processor

The CP uses a conventional microarchitecture with a merged register file and separate instruction queues, similar to the MIPS R10000 [6]. This first-level processor is intended to process operations with short issue latency. All long issue-latency instructions are to be executed in the memory processor (MP). Therefore, the CP can be designed with a perfect L2 cache in mind. This means that all structures in the processor can be quite small. The resulting processor is simpler than current generation processors and can be very fast. The CP also contains a ROB structure to perform simple rollback of mis-predicted branches.

If an instruction depends on a L2 cache miss then it is classified as long-latency and sent to the memory processor.

Instructions that have executed in the CP are removed from the ROB without later consideration after they reach the head of the ROB. This means that all the information related to this instruction is discarded. This is not incorrect because once an instruction leaves the CP ROB it is associated with a checkpoint. In case recovery has to start, the architecture will rollback to this previous checkpoint and re-execute the instructions. As a consequence, there is no necessity for a ROB structure in the MP, which instead relies on checkpoints [7].

The memory processor (MP) requires a somewhat more innovative design, but the microarchitecture is basically a reservation station based back-end with the addition of some special structures to implement a Future File and the long-latency instruction buffer (LLIB).

Instructions at the head of the CP ROB are inserted into the LLIB when they have long issue-latency. The LLIB is a buffer that stores sequences of dependent instructions in program order. The load that starts a sequence is itself not stored in the LLIB, but in the Load/Store processor. Once a missing load returns to

the MP, if this load is the oldest off-chip access in-flight, then the following instructions until the next unfinished load in the LLIB are awakened and sent to the reservation stations to be executed. To simplify the LLIB and the register management, the LLIB structure is strictly in-order. It is thus implemented as a buffer that stores the instruction opcode and the necessary data to compute and honor the dependences. The in-order nature reduces IPC by only 5% in the worst case. This agrees with our observations that the MP can tolerate large latencies and that most loads complete in-order anyway. In addition, it is necessary that the scanning is in-order so that the future file can be implemented in the memory processor.

**Register Management.** Register management in the *cache processor* works as with the R10000 [6] as long as instructions have short latency issue. Once an instruction is determined to have long-latency issue it is inserted into the LLIB. At this point, its source registers can be in two states: READY (ie, with value) or NOT READY (ie, long-latency register without value). In any case, only a single source can be READY as otherwise the instruction could never have long issue-latency. As instructions are extracted from the *cache processor*'s ROB, READY registers are retrieved and inserted into a dedicated register file, the MPRF. It is important to understand that no register sharing exists within the MPRF. Thus, two instructions in the LLIB that source the same READY register will see different entries once they are in the LLIB. This dramatically simplifies the register management, because all registers in the MPRF now have a *single consumer* and can be freed right after they are read.

NOT READY registers are inserted together with the instructions into the LLIB using only the logical register descriptor. Their purpose is to maintain the dependencies between the instructions in the LLIB.

## 5 Evaluation

### 5.1 Simulation Infrastructure

The decoupled kilo-instruction processor is evaluated using an execution-driven simulator consisting of the *simplescalar-3.0* [8] front-end (executing Alpha ISA) and a new back-end that has been rewritten from scratch to simulate kilo-instruction processors. The focus of this work is on numerical and memory intensive applications. The workload is composed of all 14 benchmarks of the SPEC FP2000 benchmark suite. The benchmarks have been compiled with *cc* version DEC C V5.9-008 on Digital UNIX V4.0 and using the `-O2` optimization level. Of each benchmark we simulate 200 million of representative instructions chosen using SimPoint [9].

Several architectures are simulated:

1. **BASE-256:** This baseline is a speculative out-of-order superscalar processor. It features structures (ROB, IQ, RF) that are somewhat larger than today's most aggressive implementations. The parameters are summarized in Table 1. The sizes are so that it is only constrained by the size of the ROB.

2. **BASE-92:** A smaller and cheaper baseline is also used. It is also limited only by the ROB which in this case has only 92 entries. This configuration has the same parameters as the CP in the decoupled kilo-instruction model. The remaining parameters are also shown in Table 1
3. **KILO:** A first-generation Kilo-Instruction Processor as described in [7] is simulated. The pseudo-ROB of this processor has 92 entries and the Slow Lane Instruction Queue has 1024 entries. These parameters make it more similar to the D-KIP model.
4. **D-KIP:** An implementation of the Decoupled Kilo-Instruction Processor is included. The parameters of the simulated D-KIP are shown in Table 2.

**Table 1.** Parameters of the two baseline configurations: BASE-256 and BASE-92

Fetch/Issue/Commit Width	4 instructions/cycle
Branch Predictor	Perceptron
I-L1 size	32 KB, 1 cycle latency
D-L1 size	32 KB, 4-way, 2 rd/wr ports, 1 cycle latency
D-L2 size	512 KB, 4-way, 2 rd/wr ports, 10 cycle latency
Memory Width / Latency	32 bytes / 400 cycles
Reorder Buffer Size	256 (BASE-256) / 92 (BASE-92)
[Integer/FP] Physical Registers	288 (BASE-256) / 124 (BASE-92)
Load/Store Queue	256 / 92 entries
[Integer/FP] Queue Size	256 entries (BASE-256) / 92 entries (BASE-92)
Integer & FP Functional Units	4 Adders / 1 Multiplier

The *Cache Processor* of the D-KIP configuration has a 92-entry ROB. This size is a result of the design guidelines presented in section 4. A 92-entry ROB is enough to recover most of the lost performance due to memory when using a configuration with a perfect L2 cache. This value is extrapolated from the IPC analysis shown in Fig. 2.

The amount of registers in the MPRF is as large as the LLIB. Having 1024 registers may seem excessive, but due to the regular nature of insertion/extraction in the LLIB it can be implemented as a banked memory structure where each bank has a single read/write port. The cost of this is very small.

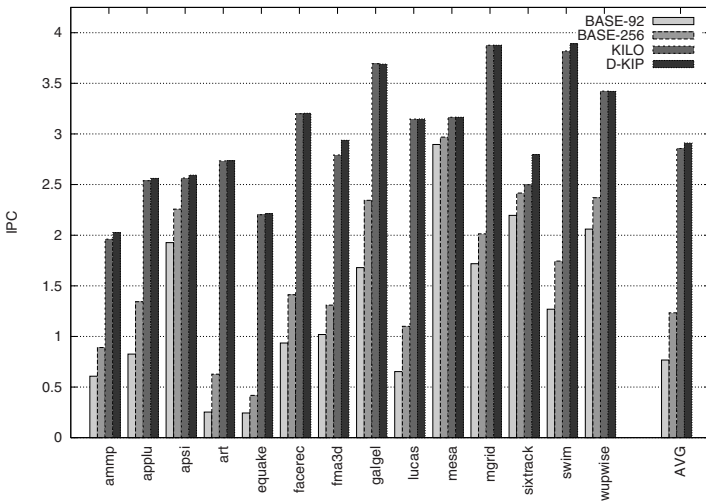
On the other hand, the size of the instruction queues may seem a bit large. These sizes have been chosen so that the ROB size limits the number of in-flight instructions. The real utilization of these instruction queues is likely to be quite small, almost never exceeding 32-40 instructions.

## 5.2 Instruction Level Parallelism

Using the same set of simulation and configuration parameters that has just been presented, the IPC achieved by all configurations has been measured for all benchmarks of SPEC FP2000. The result is shown in Fig. 5. The last column represents the average IPC for the whole benchmark set.

**Table 2.** Parameters of the decoupled KILO processor

<i>Common parameters</i>	
ICache (L1) size	32 KB, 1 cycle latency
DCache (L1) size	32 KB, 4-way, 2 rd/wr ports, 1 cycle latency
Unified Cache (L2) size	512 KB, 4-way, 2 rd/wr ports, 10 cycle latency
Memory Width / Latency	32 bytes / 400 cycles
Load/Store Queue	unlimited
<i>First Level Processor</i>	
Fetch/Issue/Commit Width	4 instructions/cycle
Branch Predictor	Perceptron
Ports to the Register File	8 Read & 4 Write
Reorder Buffer Size	92
Integer/FP Physical Registers	128 / 128
Integer/FP Queue	92 entries / 92 entries
Integer & FP Functional Units	4 Adders / 1 Multiplier
<i>Second Level Processor</i>	
LLIB Size	1024 entries
LLIB extraction rate	4 instructions/cycle
Number of Checkpoints	16 stack entries
Registers in L2RF	1024 entries
Integer & FP Functional Units	4 Adders / 1 Multiplier

**Fig. 5.** IPC values for SPECfp2000

The speed-ups are very impressive for this set of numerical benchmarks. As a result of the accurate branch prediction in numerical codes most of the instructions in the large instruction window are useful for the execution and are not discarded. This results in an average speed-up of 236% compared to BASE-256 and an impressive 379% when compared to BASE-92. The memory-bound benchmarks are the ones that obtain the largest speed-ups. This should be no surprise as memory-bound applications are those that would spend a higher number of cycles stalled due to unavailability of ROB slots. The D-KIP with its non-blocking ROB mechanism provides tremendous speed-ups for these applications.

The decoupled model achieves a small speed-up of 1.02 versus the traditional KILO model [7]. This is thanks to the separate issue queues and additional functional units. However, the goal of decoupling is not to beat the KILO model, but to perform comparable with the benefit of sustainable design complexity.

Overall the FP performance is similar to that achieved by BASE-92 with an ideal L2 cache with a latency between 10 and 20 cycles (see Fig. 2).

## 6 Related Work

Processor behavior in the event of L2 Cache misses has been studied in detail by Karkhanis *et al.* [5]. They showed that many independent instructions can be fetched and executed in the shadow of a cache miss. This observation has fueled the development of microarchitectures able to overcome the memory-wall problem [1].

Many suggestions have been proposed for overcoming the ROB size and management problem. Cristal *et al.* propose virtualizing the ROB by using a small sequential ROB combined with multicheckpointing [10,7,11]. Akkary *et al.* have also introduced a checkpointing approach [12] which consists in taking checkpoints on low-confidence branches.

The instruction queues have also received much attention recently. The *Waiting Instruction Buffer* (WIB) [13] is a structure that holds all the instructions dependent on a cache miss until its data returns from memory. The *Slow Lane Instruction Queue* (SLIQ) [7] is similar in concept to the WIB but is designed as an integral component of an overall kilo-instruction microarchitecture. In addition, it contains a pseudo-ROB structure to detect which instructions are long-latency and which are not. Recently, Akkary *et al.* have proposed the *Continual Flow Pipelines* (CFP) [14] architecture in which they propose an efficient implementation of a two-level instruction queue.

Several proposals try to improve the efficiency of physical register allocation. *Virtual Registers* [15] is a technique to delay the allocation of physical register until the issue stage. On the other hand, *Early Release* [16] tries to release registers earlier. An aggressive technique consists in combining both approaches. This technique is known as *Ephemeral Registers* [17].

Finally, several techniques have been proposed to attack the scalability problem of the load/store queues. There are several recent proposals for dealing with

the scalability problems of store queues, including two-level queue proposals [12,18] or partitioned queues [19].

The main difference between these proposals and the technique presented so far is that the D-KIP proposes a complete solution from the point of view of *Execution Locality*.

*Decoupling* is an important technique that allows simplification of the design of loosely coupled microarchitectures while increasing its performance. Smith first proposed decoupling the memory system from the processor in the Decoupled Access-Execute computer architecture [20].

## 7 Conclusions

This paper has presented a new concept in program execution termed *execution locality*. This concept describes the execution behavior of instructions inside the instruction window. It has been shown how, based on the instruction issue times, instructions can be grouped into clusters. Based on this observation a new implementation of kilo-instruction processors called the *Decoupled Kilo-Instruction Processor* has been proposed. This processor consists of three subprocessors linked by queues: the *Cache Processor*, the *Memory Processor* and a *Load/Store processor*. The decoupling offers a complexity-effective approach to the design of kilo-instruction processors.

Extensive simulations conducted using the SPEC FP2000 benchmarks have shown that this decoupled architecture fully exploits the possibilities of kilo-instruction processing. It even wins 2% of IPC when executing numerical codes. Overall, this represents a speedup of 379% compared to a single-windowed processor with the same structures as the *Cache Processor* when running numerical codes. But the important observation here is that this speed-up is obtained with only a small increase in the complexity of the design.

## Acknowledgments

This work has been supported by the Ministry of Science and Technology of Spain under contract TIN-2004-07739-C02-01 and the HiPEAC European Network of Excellence under contract IST-004408. Daniel A. Jiménez is supported by NSF Grant CCF-0545898.

## References

1. Wulf, W.A., McKee, S.A.: Hitting the memory wall: Implications of the obvious. *Computer Architecture News* (1995)
2. Wilkes, M.V.: Slave memories and dynamic storage allocation. *IEEE Transactions on Electronic Computers*, 270–271 (1965)
3. Smith, A.J.: Cache memories. *ACM Computing Surveys* 14(3), 473–530 (1982)

4. Jimenez, D.A., Lin, C.: Dynamic branch prediction with perceptrons. In: Proc. of the 7th Intl. Symp. on High Performance Computer Architecture, pp. 197–206 (2001)
5. Karkhanis, T., Smith, J.E.: A day in the life of a data cache miss. In: Proc. of the Workshop on Memory Performance Issues (2002)
6. Yeager, K.C.: The MIPS R10000 superscalar microprocessor. *IEEE Micro* 16, 28–41 (1996)
7. Cristal, A., Ortega, D., Llosa, J., Valero, M.: Out-of-order commit processors. In: Proc. of the 10th Intl. Symp. on High-Performance Computer Architecture (2004)
8. Austin, T., Larson, E., Ernst, D.: SimpleScalar: an infrastructure for computer system modeling. *IEEE Computer* (2002)
9. Perelman, E., Hamerly, G., Biesbrouck, M.V., Sherwood, T., Calder, B.: Using SimPoint for accurate and efficient simulation. In: Proc. of the Intl. Conf. on Measurement and Modeling of Computer Systems (2003)
10. Cristal, A., Valero, M., Gonzalez, A., Llosa, J.: Large virtual ROBs by processor checkpointing. Technical report (2002), Technical Report number UPC-DAC-2002-39 (2002)
11. Cristal, A., Santana, O.J., Martinez, J.F., Valero, M.: Toward kilo-instruction processors. *ACM Transactions on Architecture and Code Optimization (TACO)*, 389–417 (2004)
12. Akkary, H., Rajwar, R., Srinivasan, S.T.: Checkpoint processing and recovery: Towards scalable large instruction window processors (2003)
13. Lebeck, A.R., Koppanalil, J., Li, T., Patwardhan, J., Rotenberg, E.: A large, fast instruction window for tolerating cache misses. In: Proc. of the 29th Intl. Symp. on Computer Architecture (2002)
14. Srinivasan, S.T., Rajwar, R., Akkary, H., Gandhi, A., Upton, M.: Continual flow pipelines. In: Proc. of the 11th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (2004)
15. Gonzalez, A., Valero, M., Gonzalez, J., Monreal, T.: Virtual registers. In: Proc. of the 4th Intl. Conf. on High-Performance Computing (1997)
16. Moudgill, M., Pingali, K., Vassiliadis, S.: Register renaming and dynamic speculation: an alternative approach. In: Proc. of the 26th. Intl. Symp. on Microarchitecture, pp. 202–213 (1993)
17. Cristal, A., Martinez, J., Llosa, J., Valero, M.: Ephemeral registers with multichunking. Technical report(2003), Technical Report number UPC-DAC-2003-51, Departament d'Arquitectura de Computadors, Universitat Politecnica de Catalunya (2003)
18. Park, I., Ooi, C.L., Vijaykumar, T.N.: Reducing design complexity of the load/store queue. In: Proc. of the 36th Intl. Symp. on Microarchitecture (2003)
19. Sethumadhavan, S., Desikan, R., Burger, D., Moore, C.R., Keckler, S.W.: Scalable hardware memory disambiguation for high ILP processors. In: Proc. of the 36th Intl. Symp. on Microarchitecture (2003)
20. Smith, J.E.: Decoupled access/execute computer architectures. In: Proc. of the 9th annual Intl. Symp. on Computer Architecture (1982)