Chip Multiprocessors with Implicit Transactions

Enrique Vallejo[†], Marco Galluzzi[‡], Adrián Cristal[‡], Fernando Vallejo[†], Ramón Beivide[†], Per Stenström[§], James E. Smith[¤] and Mateo Valero[‡*]

[†] *Grupo de Arquitectura de Computadores, Universidad de Cantabria*
[‡] *Departament d'Arquitectura de Computadors, Universitat Politècnica de Catalunya*
**Computer Architecture Group, Barcelona Supercomputing Center (BSC)*
[§] *Dept. of Computer Science and Engineering, Chalmers University of Technology*
[¤] *Dept. of Electrical and Computer Engineering, University of Wisconsin-Madison*

ABSTRACT

Chip Multiprocessors (CMPs) are an efficient way of designing and use the huge amount of transistors on a chip. Different cores on a chip can compose a shared memory system with a very low-latency interconnect at a very low cost. Unfortunately, consistency models and synchronization styles of popular programming models for multiprocessors impose severe performance losses. Known architectural approaches to combat these losses are too complex, too specialized, or not transparent to the software.

In this short paper, we introduce *Implicit Transactions* as a generalized architectural concept to remove such performance losses, which can be implemented at a low complexity by leveraging the multi-checkpoint mechanism of the Kilo-Instruction Processor [Cris05]. By relying on a general speculation substrate, it supports the strictest consistency model – sequential consistency – potentially as effectively as weaker models and it allows multiple threads to concurrently execute critical sections, and speculate beyond barriers and synchronization events.

KEYWORDS: Kilo-Instruction Processors; Multiprocessors; Consistency; Implicit Transactions

1 Introduction

Nowadays, some of the most notable limiting factors related to the design of multiprocessor systems are: the maintenance of a coherent view of memory among concurrently executing cores with private caches – *cache coherence* –; the correct ordering of the accesses to shared data that is consistent with architected rules that the programmer relies on – *memory consistency* –; and the implementation of mechanisms for thread synchronization – *barriers* – and for mutual exclusion in shared data access – *critical sections* –. These problems, of course, also complicate the task of programming parallel systems.

[†] E-mail: {enrique, fernando, mon}@atc.unican.es. These authors thank the support by the Spanish Ministry of Education and Science (Grant AP-2004-6907 and TIN2004-07440-C01-01).

[‡] E-mail: {galluzzi, adrian, mateo}@ac.upc.edu. These authors thank the support by the Spanish Ministry of Education and Science (TIN-2004-07739-C02-01 and grant AP2003-0539), the HiPEAC Network of Excellence and the Barcelona Supercomputing Center.

[§] E-mail: pers@ce.chalmers.se. This author thanks the Swedish Research Council under contract VR 2003-2576.

[¤] E-mail: jes@jkl.ece.wisc.edu. This author thanks the NSF (Grant CCR-0311361).

A proposed method for efficient and accurate data sharing is the use of *Transactional Memory* [Anan05] [Herl93] [Moor06], which also allows to use simple memory consistency models [Hamm04] and to simplify, to a great extent, the programmer task by giving the programmer the *transaction* construct. Transactions are blocks of code that execute atomically, and are aborted if there is a data race with another transaction in a remote core.

In this work, we introduce the concept of *implicit transactions*, i.e. transactions composed and orchestrated solely in hardware, without the need for explicit instructions and programmer support. Each core automatically divides execution into implicit transactions, executing several ones concurrently, and validating them atomically and in thread-program order. This is done by leveraging the multi-checkpoint mechanism in the Kilo-Instruction Processors [Cris05].

Kilo-Instruction Processors have emerged as an efficient proposal to overcome the *memory wall problem* by holding thousands of in-flight instructions, overlapping computation with long latency memory accesses. They are based in a multi-checkpoint system that overrides the need for a big RoB in the processor. Furthermore, holding thousands of in-flight instructions has been showed to efficiently reduce the latency penalty associated to the interconnection mechanisms on multiprocessor systems [Gall04].

The presented multiprocessor system based on our implicit transactions, it is expected to provide a high performance, while at the same time maintaining the most restrictive consistency model – sequential consistency –. Additionally, the proposal can host a number of optimizations, such as concurrent speculative execution of parallel sections and silent store detection and deletion, all of them with a low design cost.

2 Proposed Model: Implicit Transactions

In the proposed model we consider all the instructions comprised between two different checkpoints as a transaction, and in addition we apply a mechanism to provide atomicity, consistency and isolation properties. We call this kind of transactions *implicit transactions*, as they are automatically hardware delimited, by means of the checkpointing mechanism, and the programmer does not need to be aware of it. Therefore, both the programming model and the ISA are unaffected, and current binaries can be directly executed. This mechanism is, due to the architectural relation, similar to the one proposed in [Hamm04].

During the execution, all the in-flight instructions remain speculative until their corresponding checkpoint commits. Memory instructions remain in the processor queues and do not modify the local cache or the global memory, as they are subject to a rollback. The oldest

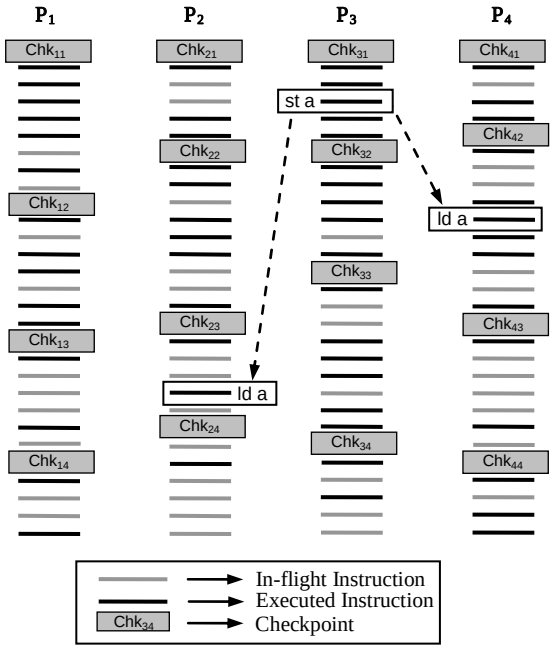

Figure 1: Execution flow for 4 processors

checkpoint in the processor can commit when all of its corresponding instructions are finished. This commit is followed by acquiring the write grant and a logically atomic (not releasing the grant) broadcast of the pending memory updates.

Remote processors snoop the memory update addresses, searching for a conflict with any speculative loads they have in their load queues. In that case, the remote processor is forced to roll back, because it has speculatively used data that, at this point, is discovered to be invalid. Finally, when the broadcast finishes, the checkpoint commits and the speculatively executed instructions are considered to be globally performed. In the example of figure 1, the broadcast of a store to memory location "a" conflicts with two other processors that have already speculatively loaded from location "a". In this example, P2 is rolled back to Chk23, causing instructions from Chk24 to Chk23 to be discarded. Also P4 rolls back to Chk42.

3 Contributions

The strength of our proposal lies in correctly leveraging the multi-checkpointing mechanism that the Kilo-Instruction Processors implicitly offer. We propose using checkpoints for two main reasons:

1. *To perform memory updates in a transactional manner*, by grouping store operations from a single checkpoint and releasing them all together when the checkpoint is ready to commit.
2. *To easily apply speculation mechanisms for critical sections and synchronization points*, by adding a "silent store" [Lepa02] elimination mechanism, and barrier detection mechanisms.

3.1. Transactional behavior

We propose for the first time, to our knowledge, what we call *implicit transactions*. Thanks to this transactional behavior, our proposed design provides desirable features that traditional transactional memory systems implicitly provide:

- *The sequential memory consistency model is preserved*, which is the simplest memory model for a programmer to think about when coding. Furthermore, our design allows memory operations to be reordered and overlapped, similarly to previous works [Gnia04] [Rang97], what provides the opportunity for improving the performance achieved by direct implementations, which normally do not perform as well as relaxed consistency models.
- *The cache coherence protocol overhead is reduced*. Managing several memory updates atomically, similarly to [Hamm04], can reduce the overhead experienced by the interconnect hardware when managing each memory update separately.

Consequently, the first reason to leverage the checkpointing mechanism deals with the "correctness" and "performance" of the system.

3.1. Speculation mechanisms

This point only deals with the "performance" of the system. We consider that *the system is suitable to be improved with speculative mechanisms for constructions like locks and barriers*, such as [Rajw01] or [Mart02], which can reduce the synchronization waits of parallel application when such constructs are conservatively coded. In our case, we propose a generic silent store removal mechanism for memory operations within a checkpoint that elides the lock blocking, and a barrier detection mechanism to allow subsequent speculation. The cited proposals on speculating on locks and barriers constructs need to incorporate some kind of checkpointing mechanisms to be able to return to a safe non-speculated state. Given that we start from a design that natively incorporates

such checkpointing, the implementation cost for these mechanisms is reasonably low. Other interesting features of our speculating mechanisms are:

- *No software support is needed*. Although other proposals are also hardware-based, we can achieve it more easily in certain cases.
- *The size of the critical section is not a problem*. In case of overflow our design divides a large critical section among different implicit transactions without affecting "correctness". This is equivalent to actually locking the critical section, avoiding any further speculation.
- *More opportunity to speculate beyond a barrier*. Since Kilo-Instruction Processors allow thousands of instructions to be in-flight, we can possibly speculate far enough after a barrier to make all processes reach the barrier without waiting.

Finally, we want to remark that using checkpoint-based processors on a multiprocessor system can be useful under different configurations, such as the studied snoopy-bus SMP configuration, or a CMP. Of course, it could be also interesting exploring other configurations such a directory-based DSM, that needs special care due to its unordered interconnect.

References

[Anan05] C. S. Ananian, K. Asanovic, B. C. Kuszmaul, C. E. Leiserson, and S. Lie, "Unbounded Transactional Memory", In Proc. of the 11th HPCA, California, pp 316-327, Feb. 2005.

[Cris05] A. Cristal, O. Santana, Francisco 1, M. Galluzzi, T. Ramirez, M. Pericas, M. Valero. "Kilo-Instruction Processors: Overcoming the Memory Wall," *IEEE Micro*, vol. 25, no. 3, pp. 48-57, May/June 2005.

[Gall04] M. Galluzzi et al., "A First Glance at Kilo-Instruction based Multiprocessors", In Proc. of the 1st Conf. on Computing Frontiers, pp. 212-221, April 2004.

[Gnia04] C. Gniady, B. Falsafi, and T. N. Vijaykumar, "Is SC + ILP = RC?", 26th ISCA, 1999.

[Hamm04] L. Hammond et al., "Transactional Memory Coherence and Consistency", In Proc. of the 31st ISCA, Germany, June 2004.

[Lepa02] K. M. Lepak, M. H. Lipasti. "Temporally silent stores", In 10th ASPLOS, Oct. 2002.

[Herl93] Maurice Herlihy and J. E. B. Moss. "Transactional Memory: Architectural Support for Lock-Free Data Structures". In Proceedings of the 20th ISCA, pages 289-300, May 1993.

[Mart02] J. Martínez, J. Torrellas, "Speculative Synchronization: Applying Thread-Level Speculation to Explicitily Parallel Applications", In Proc. of the 10th ASPLOS, Oct. 02

[Moor06] K. Moore *et al*, "LogTM: Log-Based Transactional Memory", In 12th HPCA, Feb. 2006.

[Rajw01] R. Rajwar, and J. R. Goodman, "Speculative Lock Elision: Enabling Highly-Concurrent Multithreaded Execution", In 34th Int. Symp. on Microarchitecture, Dec. 2001.

[Rang97] P. Ranganathan, V.S.Pai, and S. Adve, "Using Speculative Retirement and Larger Instruction Window to Narrow the Performance Gap Between Memory Consistency Models", In Proc. of the 9th SPAA. June, 1997.