Runahead Threads: Reducing Resource Contention in SMT Processors

Tanausú Ramírez[1], Alex Pajuelo[1], Oliverio J. Santana[2], Mateo Valero[1,3]
[1]UPC, email:{tramirez,mpajuelo,mateo}@ac.upc.edu. [2]ULPGC, email:ojsantana@dis.ulpgc.es. [3]BSC.

Abstract

In this work, we propose Runahead Threads as a valuable solution for both exploiting memory-level parallelism and reducing resource contention in simultaneous multithreaded processors.

1 Introduction

In the SMT [2] scope, threads share and compete for important resources in the processor core. The different features and requirements of threads can unbalance the resource allocation. Some threads can hold more resources than others, degrading performance seriously and hindering the benefit of multithreaded execution.

To overcome this, different fetch policies and resource schedulers have been proposed. A fetch policy decides which threads can feed the processor with new instructions to get the opportunity of using the available resources. A resource scheduler controls the resource allocation among threads, trying to avoid resource monopolization. These resource control policies stall or flush threads under determined conditions to prevent resource overuse.

The most harmful case occurs with memory-bounded threads. A memory-bounded thread can block the ROB due to long-latency operations, since following instructions cannot be issued nor committed. Besides, a lot of critical resources may have been assigned to this thread, which is not making any progress. If this thread holds too many resources, it can starve other threads of the required resources, that is, it prevents them from advancing too. Consequently, this effect leads to global performance degradation in SMT processors.

Current resource policies make drastic actions, either stalling or flushing threads, which degrade memory-bounded thread performance to unfairly benefit fast threads. Besides, they can sometimes produce a resource under-utilization situation, preventing a stopped thread from using resources that no other thread requires.

2 Proposal

We propose **Runahead Threads** (RaT) to both exploiting the memory-level parallelism and reducing the resource contention in SMT processors. *Runahead* [1] execution is a mechanism whose goal is bringing speculatively data and instructions into the caches. Our proposal transforms an eager resource thread into a light-consumer thread with fast instruction stream execution.

When a thread undergoes a long-latency load, it switches to RaT. Then, it enters into an speculative light mode in which the thread uses the different resources during short time without limiting the available resources for other threads. At the same time, the issued prefetches make possible to increase the memory level parallelism improving its own performance. With this ability, RaT's allow memory-bounded threads to speculatively going in advance, doing something beneficial to itself without disturbing the other threads. RaT's improve the total performance by increasing the memory level parallelism available and by alleviating resource contention among threads.

Therefore, RaT's are much less aggressive than normal threads with the valuable SMT resources, taking and releasing them in short periods of time. According to our results, RaT uses much less than half the registers used in normal execution.

3 Conclusions

Although RaT's does not have knowledge of direct resource allocation among threads, its advantage comes from the right interaction between the fast runahead threads and normal threads, which makes possible that both memory-bounded threads and the remainder threads get improvements using the available resources. Our results prove that it is preferable to exploit memory-level parallelism under overpressure in memory than strictly limiting the resources or even stalling or flushing the threads as some previous work does.

We compare an SMT architecture using RaT's to both state-of-the-art static fetch and dynamic resource control policies. Our results show that RaT's performs better, in terms of throughput and fairness, than any of those policies.

References

[1] O. Mutlu *et. al.* Runahead execution: An alternative to very large instruction windows for out-of-order processors. *HPCA'03*, Washington, 2003.
[2] D. M. Tullsen, *et. al.* Exploiting choice: instruction fetch and issue on an implementable simultaneous multithreading processor. *ISCA-23*, New York, 1996.

Acknowledgments.This work has been supported by the Ministry of Education of Spain under contract TIN-2004-07739-C02-01 and grant AP2003-3682, the HiPEAC European Network of Excellence, and the Barcelona Supercomputing Center (BSC-CNS).