From Simulation to Practice:

Cache Performance Study of a Prolog System.

Ricardo Lopes[1], Luís Fernando Castro[2] and Vítor Santos Costa[3]

[1]DCC-FC and LIACC, Universidade do Porto, Portugal
rslopes@ncc.up.pt

[2]Dept. of Computer Science, SUNY at Stony Brook
luis@cs.sunysb.edu

[3]COPPE-Sistemas, Universidade Federal do Rio de Janeiro, Brasil
vitor@cos.ufrj.br

Motivation

- Our work aims at addressing several important questions in the implementation of logic programming systems (Yap Prolog System in particular).
- Prolog applications can often spend hundreds of megabytes of data, but there was little work on understanting and quantifying the interactions between Prolog programs and the memory architecture of modern computers.
- Memory performance is a critical issue in the design of a Prolog system, both in terms of total memory usage and locality in memory accesses.
- Most modern Prolog systems are emulator-based. Best performance thus requires both good emulator design and good memory performance.
- Determine through comparative analysis to what extend memory performance depends on the WAM design and on actual implementation.

Introduction

- Declarative languages such as Prolog have been successfully deployed to complex applications such as knowledge base management, machine learning, and natural language processing.
- Such applications are often very demanding, both in terms of CPU and memory usage.
- On the other hand, progress in Prolog applications requires ever better performance and scalability from Prolog implementation technology.

Modern Prolog Systems

- Most modern Prolog systems are based on David H. D. Warren's WAM instruction set.
- Often, the Prolog system includes a compiler that generates WAM code.
- The WAM code is then executed by a specialized emulator, usually written in C.
- Emulators are popular because they are easier to implement and maintain than native-code systems.
- Emulators provide flexibility and portability, at some cost in performance.
- Recent work in obtaining best performance for Prolog has shown that good emulator design can in fact obtain good performance, not so distant from what one would expect from a compiler.

Studying Memory Performance

- The main goal of this work was to understand the memory characteristics of the Yap Prolog system.
- Modern CPUs strongly rely on caches to obtain good memory performance, thus our work focuses on cache performance.
- One may follow several strategies to understand cache performance:
  - obtain detailed information using a simulator;
  - use a CPU profiler to obtain the number of misses in real-time.
- We used the two approaches.

Studing Memory Performance through Simulation

- We used the SimpleScalar simulator toolkit for our simulation studies.
- This toolkit can simulate a multi-scalar CPU (MIPS) and related memory management.
- Simulators can provide data on how many misses stem from each line of code.
- In this work, we were only interested in the memory characteristics of applications, namely on cache performance. We simulate a simple CPU, concentrating on cache organization, and more specifically on the number of misses.
- To perform our simulations we decided to use a cache design based on a standard Intel Pentium-III (16Kb+16Kb + 256Kb), and on an AMD Athlon processor (64Kb+64Kb + 256Kb).

Studing Memory Performance through Profiling

- To validate our results, we compare simulated results to those obtained from profiling of actual benchmark execution.
- We used the Intel Pentium-III and the AMD Athlon performance counters.
- We relied on Linux's `perfctr` driver for access to the performance counters in Linux, and on the `lperfex` tool for easy analysis of the counters.
- Both the Athlon and the Pentium-III execute the same code.

The Benchmark set

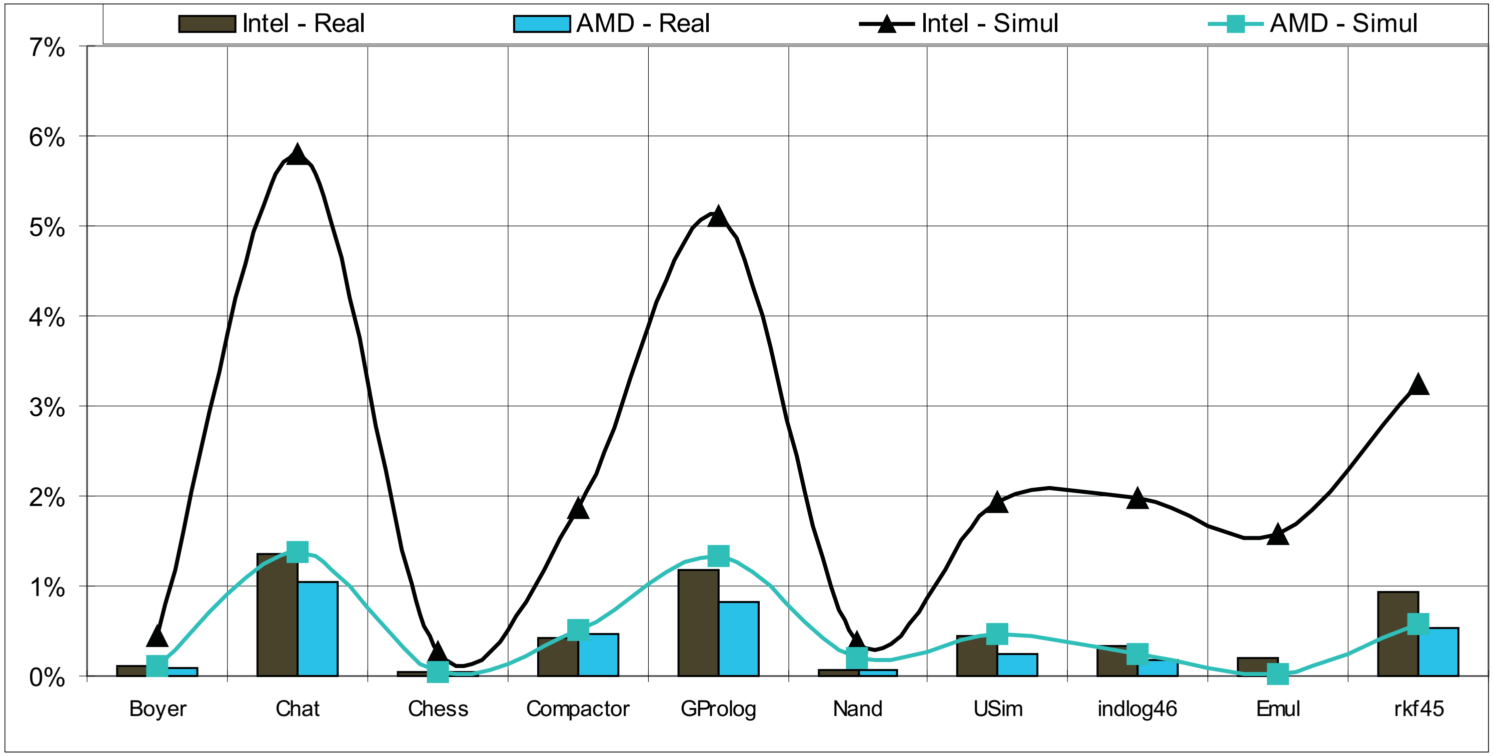
- Our previous experience in studying Prolog systems has shown that Prolog applications can be very diverse and unique.
- In our study the choice of benchmark set was based on three considerations:
  - we wanted to study both deterministic and non-deterministic applications;
  - we wanted both pure Prolog applications and applications that would frequently execute built-ins;
  - simulator limitations mean we should study medium scale applications.

Cache performance for Boyer, a deterministic application

	Simulated				Actual			
	P-III		Athlon		P-III		Athlon	
	Accesses	Misses	Accesses	Misses	Accesses	Misses	Accesses	Misses
I-L1	73,343,933	0.44%	73,385,200	0.11%	89,575,345	0.09%	47,879,772	0.10%
D-L1	33,945,988	0.66%	33,947,744	0.26%	52,277,504	0.44%	45,062,629	0.19%
U-L2	661,774	16.4%	218,404	29.27%	800,822	7.7%	134,536	51.8%

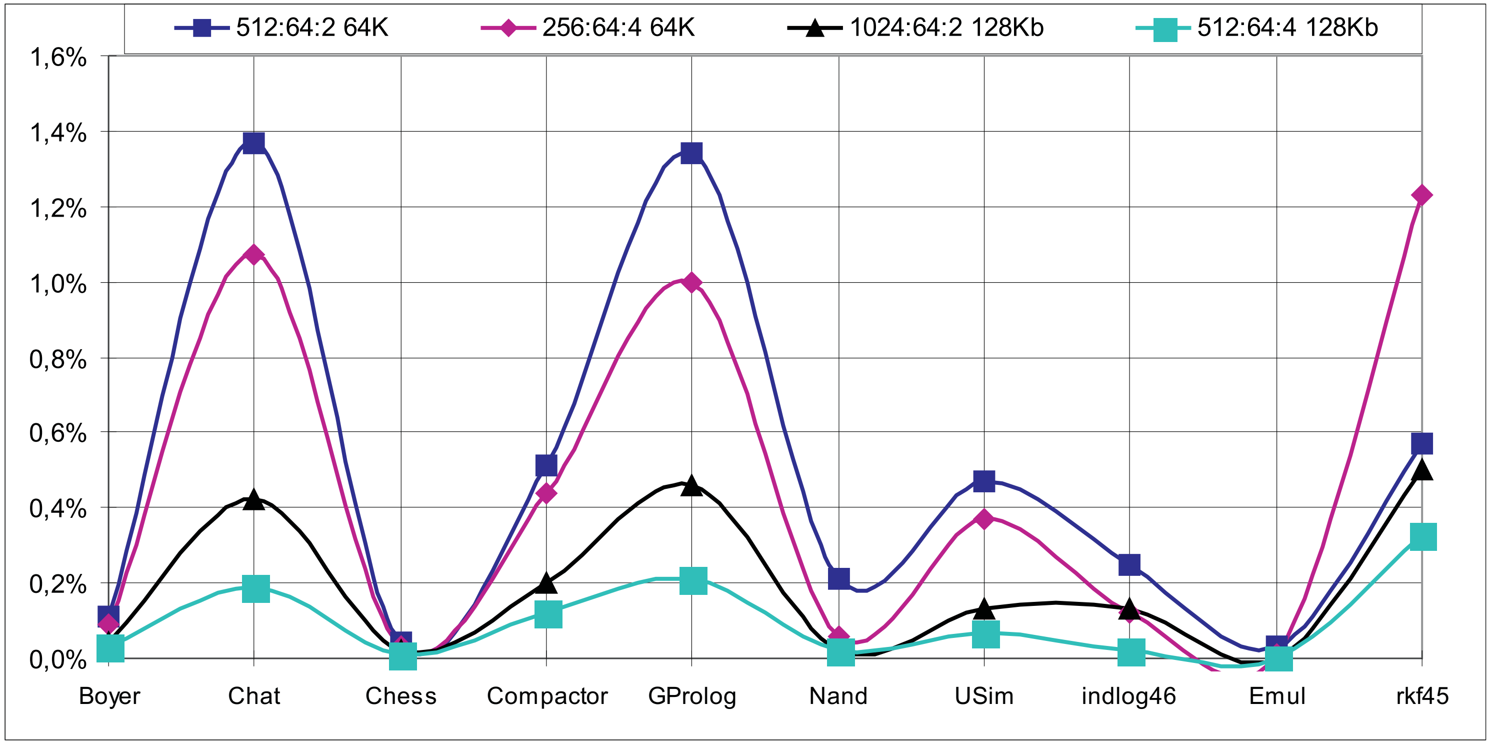
- Simulation results have shown that Boyer has outstanding locality: it uses a few WAM instructions, and most data accesses are to the top of the stack.
- The Pentium III processor provides counters for number of instructions fetched, whereas the Athlon counts the number of memory words accessed (several instructions per word).
- The number of L2 misses is about the same for both processors.
- L2 miss rate for the Athlon is higher due to the high efficiency of its L1 caches.

I-L1 cache Misses - Real vs. Simulated



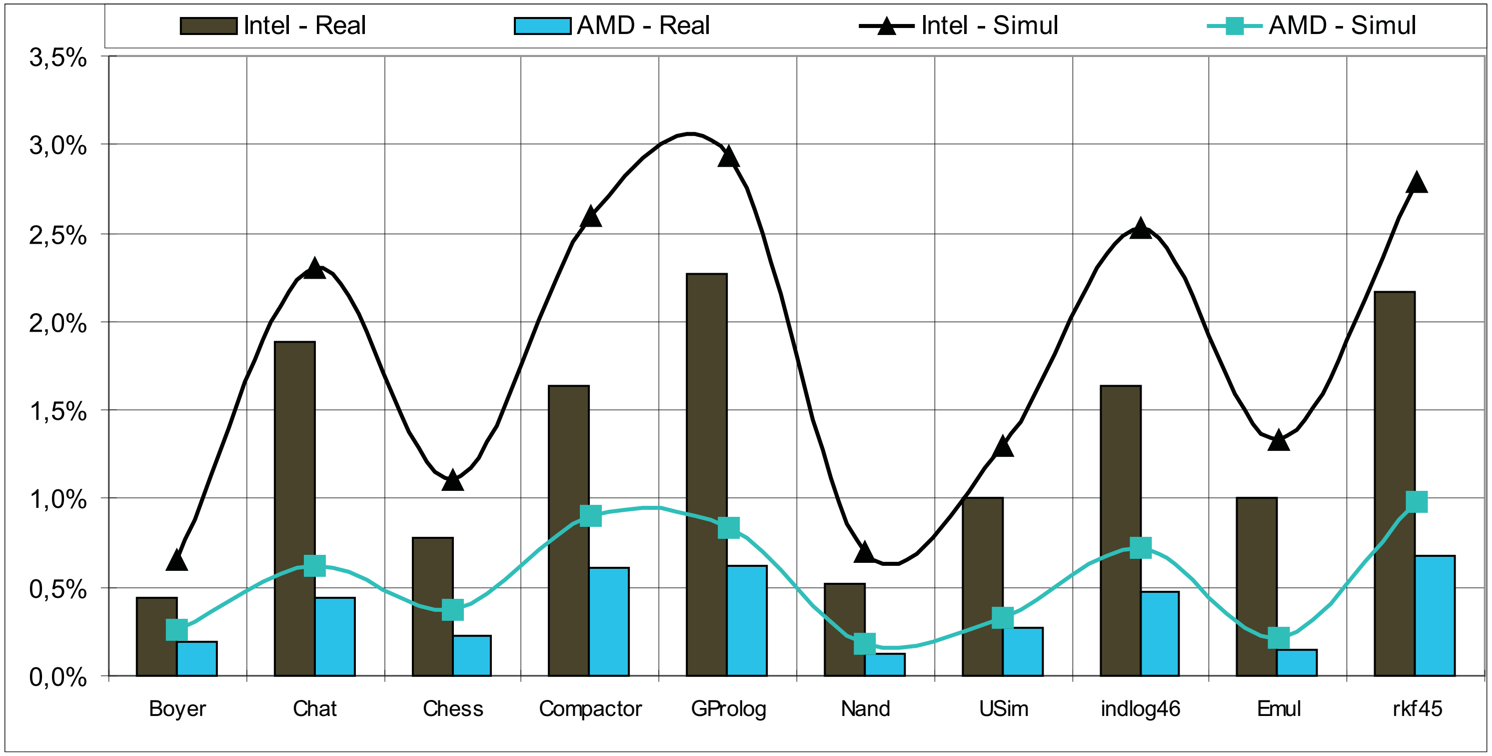
- The I-L1 cache presents the most interesting differences between simulation and actual execution, due to the differences in the ISA between the x86 architecture and the MIPS-like one used by the simulator.

I-L1 cache Misses - Simulation



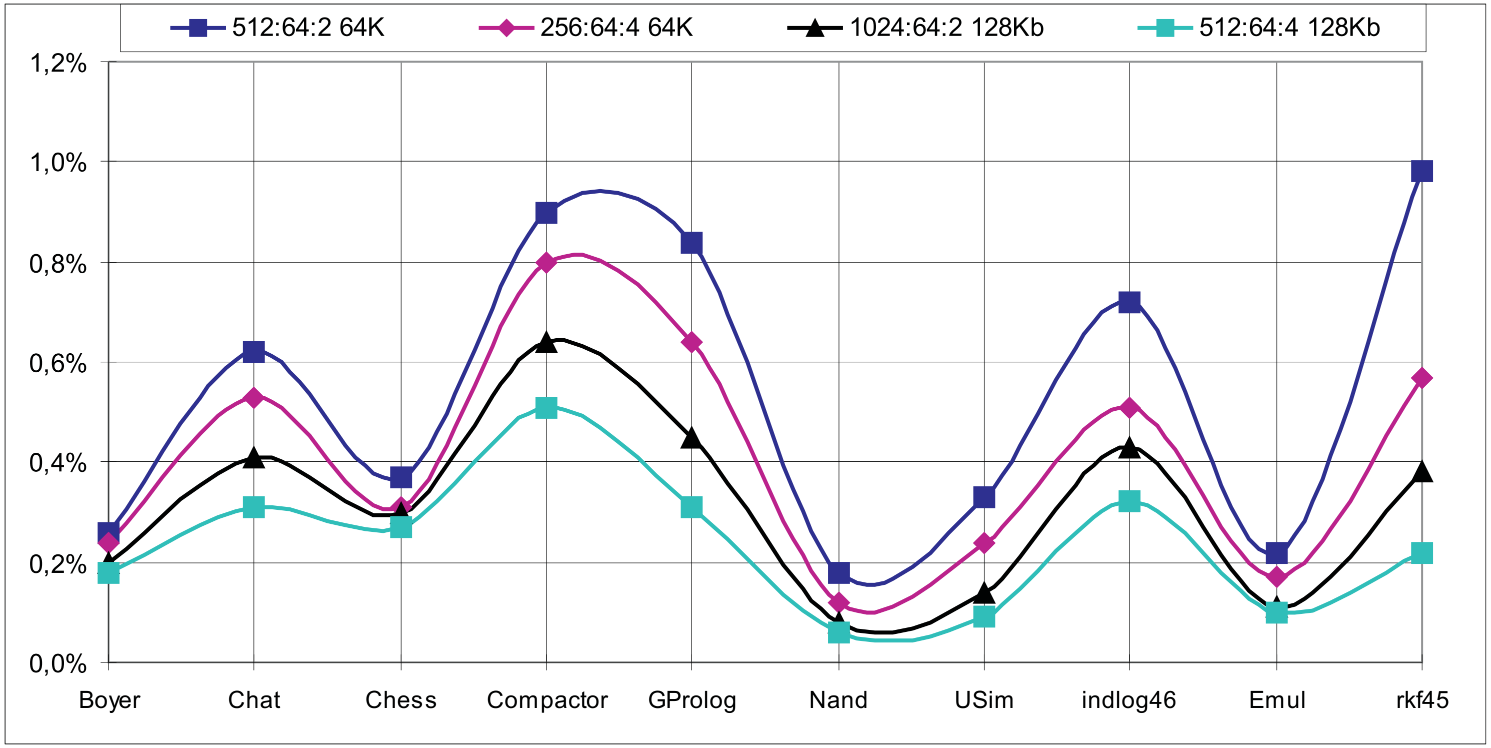
- The results from simulation miss rates for different configurations of the I-L1 cache show that Yap clearly benefits from larger caches, and from moving from 2 way (as used in AMD) to 4 way associativity (used on Intel).

D-L1 cache Misses - Real vs. Simulated



- One expected difference between simulation and actual execution is due to the smaller register set of the x86 architecture, which inhibits Yap’s optimization of keeping WAM registers in actual machine registers.

D-L1 cache Misses - Simulation

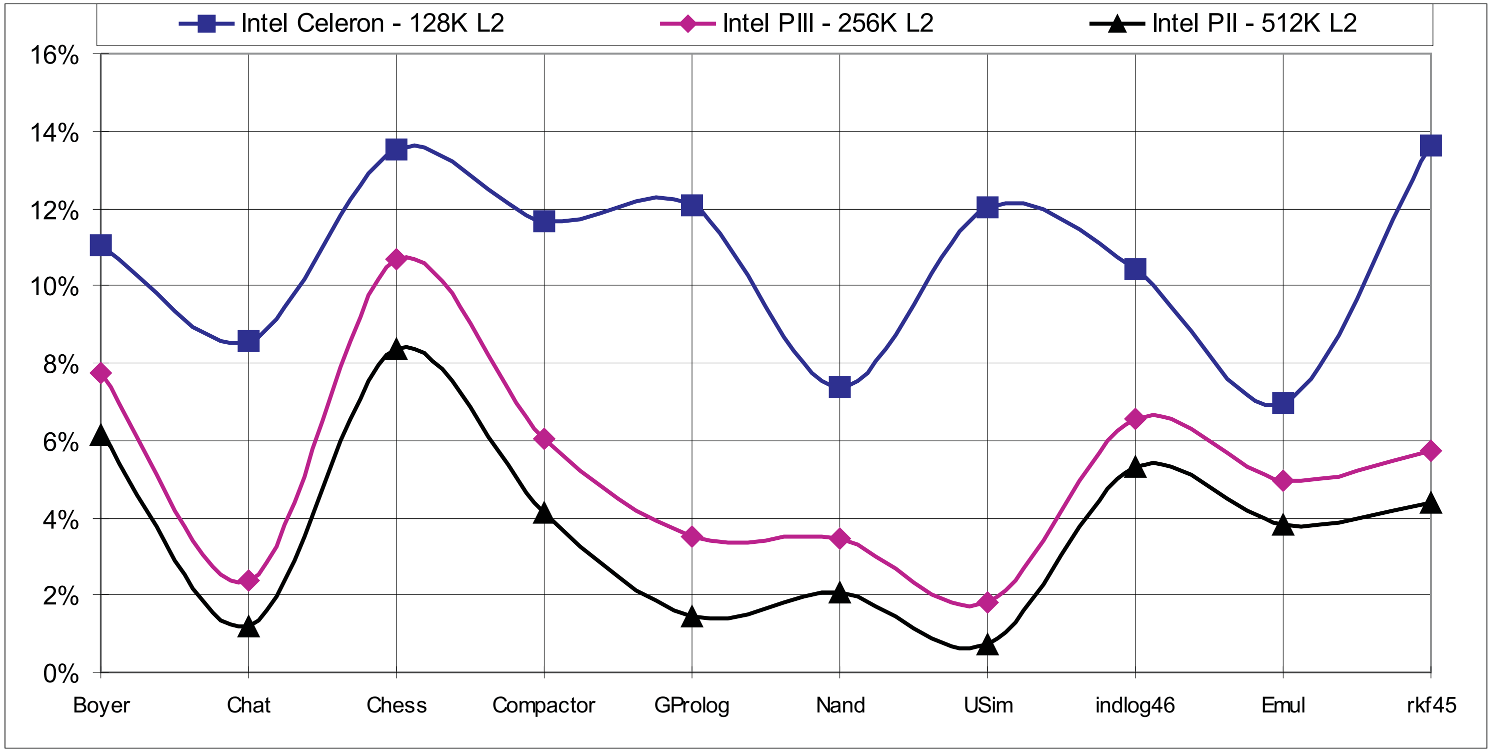


- the results from simulation for different configurations of the D-L1 cache show that, once again, Yap benefits from larger caches, and from moving from 2 way to 4 way associativity.

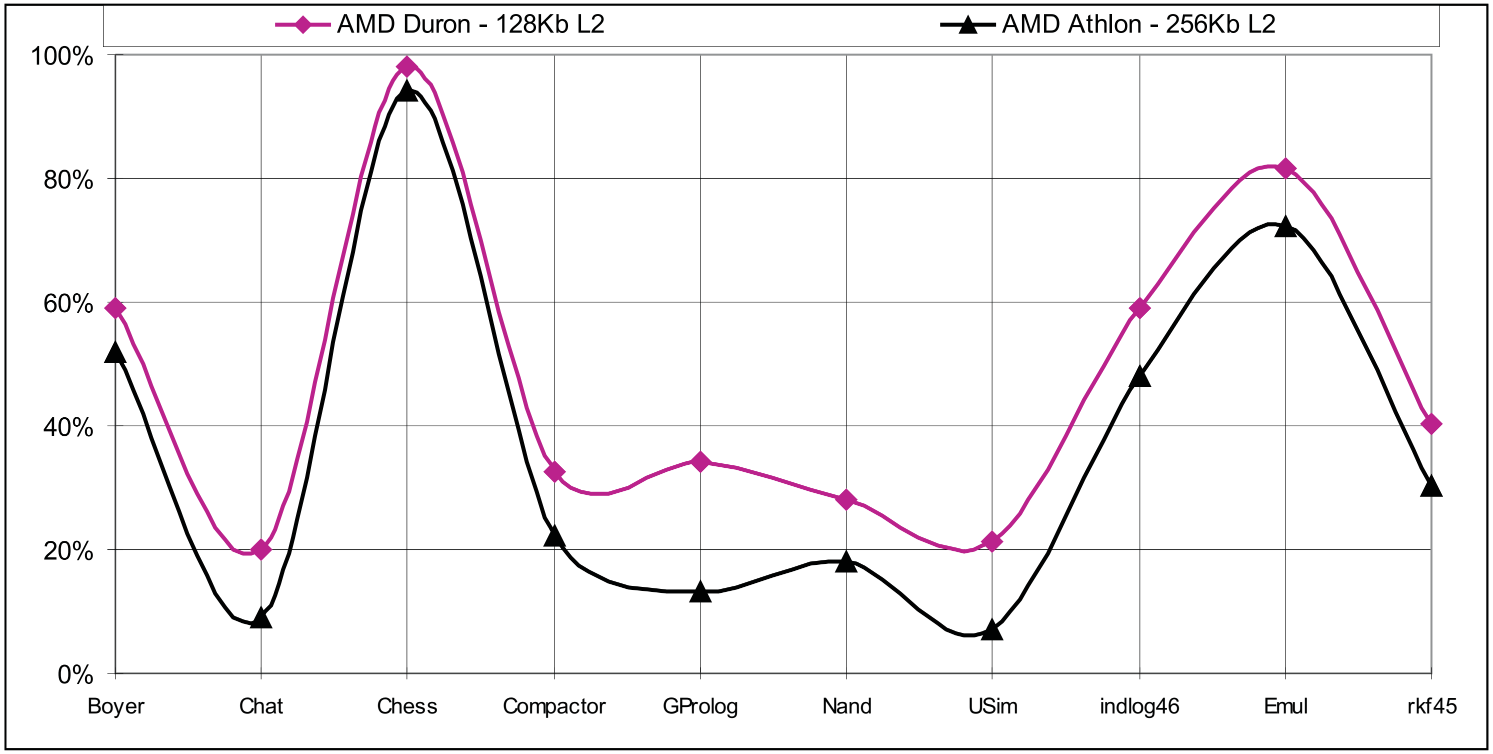
L2 Cache Misses

- Analyzing L2 performance is harder, because its usage is very dependent on the efficiency of the L1 caches.
- The number of L2 accesses in the actual Pentium-III is very close to that predicted by the simulator for most benchmarks.
- The picture is quite different for the Athlon processor. The number of accesses is usually much smaller than that given by the simulator, mostly due to Athlon’s excellent D-L1 performance.

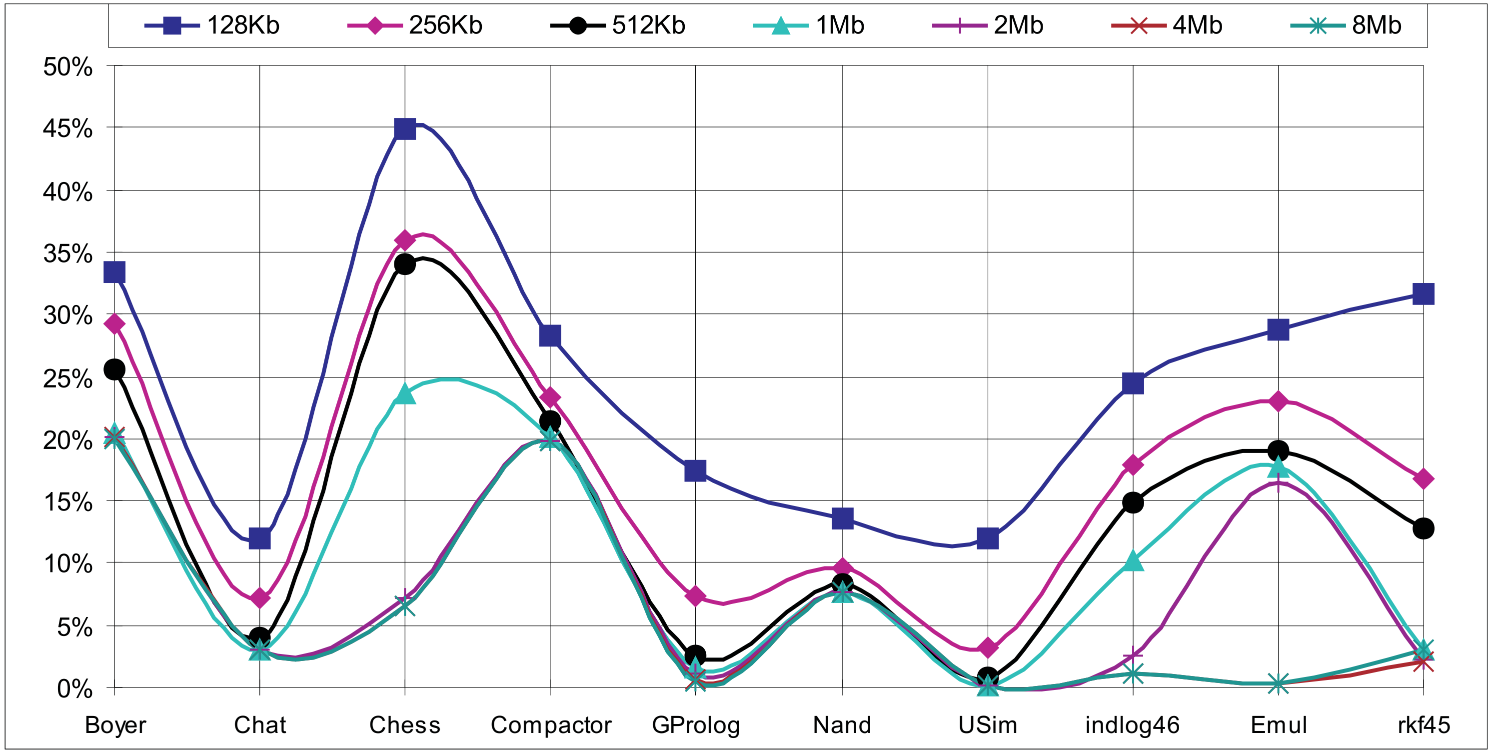
L2 Cache Misses - Celeron vs. Pentium II vs. Pentium III



L2 cache Misses - AMD Duron vs. AMD Athlon



L2 cache Misses - Simulation



- As we were unable to obtain profiling results on processors with larger L2-caches, we have used the simulation to understand if YAP could benefit from larger caches.

More Results

- The L2 simulation results show that on most benchmarks having a cache with 1Mb or more doesn't especially improve the miss-rates, except for Emul and Chess, where YAP actually benefits from a cache larger than 4Mb.

- We need to relate the impact that these cache misses impose to processor pipelines, and instruction throughput.
- Although these processors are capable of performing instruction-level parallelism on average each instruction took more than one CPU clock to execute.
- The best result is for the Athlon, on the `Chess` program, with a factor of 0.96 cycles per instruction retired. All other have CPI larger than one, the worst being for the `Rkf45` benchmark on the Pentium-III, which takes 1.56 cycles per instruction, on average.
- The AMD Duron has up to 10% more cycles than its brother Athlon.
- The differences are stronger between a Pentium-III and a Celeron. Up to 15% more cycles on the Celeron.

Conclusions

- We have presented a detailed study of the cache performance of Yap.
- Our results indicate that Yap usually achieves good miss-rates at the L1 caches, although it tends to perform badly at L2 cache accesses.
- Profiling essentially confirmed the main patterns we had found in the simulation-based analysis.
- We found the Pentium-III I-L1 cache performance to be a lot better than expected.
- Almost every application benefits from a larger cache size, as what is found on the AMD Athlon line of CPUs.
- L2 analysis is harder, partly because miss-rates largely depend on how many accesses are caught by the L1 caches.
- Our results confirm that L2 miss-rates are usually higher than L1 miss-rates, and particularly high when the mark-sweep garbage collector runs (up to 30%).
- Note that in a few applications miss-rates at the L1 caches are so good as to make L2 performance negligible.

Future Work

- Profiling has shown to be a most useful tool in understanding Yap memory performance.
- In the future we plan to study large Prolog applications by profiling them first and then using the simulators to obtain detailed understanding of the core kernel components.
- To do so we will take advantage of constant improvement in profiling tools, and do work on obtaining the profiling information to within the very system.