

Loop parallelization using hardware-based inter core communication

Eduardo F. Barbosa (IC)

Abstract

As parallelization and parallel programs become more common, techniques were created to help the compiler perform this task automatically. A common target for parallelization are program loops, however most of these techniques rely only on software mechanisms to manage inter core communication, and consequently a high overhead is incurred. In this paper we investigate the benefits of having a dedicated hardware module to support inter core communication, and the results achieved.

Key words: computer architecture, parallelization, FPGA.

Introduction

Techniques have been proposed to parallelize these loops, but as data synchronization is needed, some authors also proposed a hardware support to help implementation.^[1] As there is no hardware support on processors, these techniques are usually implemented using shared memory to synchronize this data, but being memory accesses too slow, it compromises the overall speedup achieved. A possible solution to this would be to use a Network on a Chip (referred as NoC from now on), a network connecting all cores making them able to send and receive data from one another, and this communication is much faster than talking to the memory.

So this works goal is to validate that a hardware support is essential to achieve good speedups. The validation consists of synthesizing a processor with that hardware on a FPGA, then running some benchmarks parallelized with those techniques.

Results and Discussion

Complications were encountered during this experiment, and only recently it was solved and data started being collected. The experiment consists of a LEON processor equipped with a NoC, and a memory latency module (to bring the simulation system closer to reality). It was chosen 4 benchmarks for this test, and they were parallelized using software techniques^{[1][2]} and with the hardware support. The tests would execute for memory latencies configurations varying from 0 to 64 cycles of delays (delay added to memory response of read requests).

As of right now, data was collected from 0 to 8 added latency cycles, and due to them being a little increase, they still don't affect the results.

Image 1 show the speedups achieved for a 8 cycle delay configuration, which was almost the same for the other configurations (speedups measured comparing to the serial version).

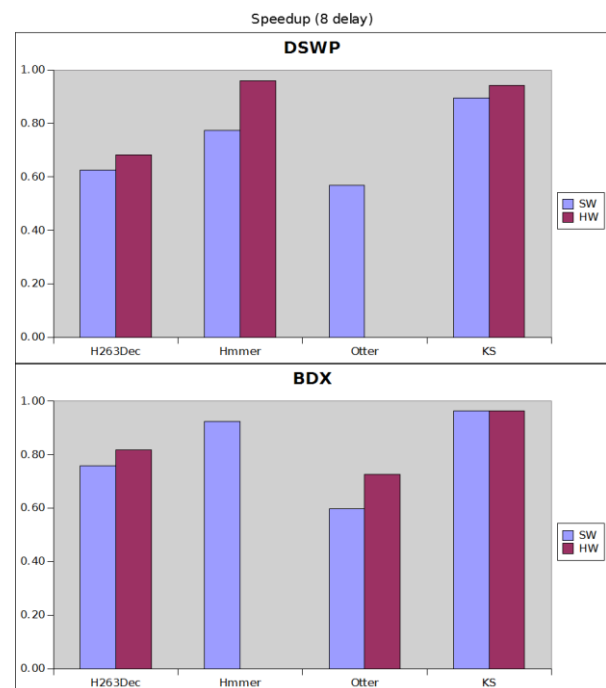


Image 1. Speedups with 8 cycles added.

Conclusions

For a fast memory, a hardware approach is a bit better than the software, but they are both worse than the serial and not worth it.

We believe results will appear for higher delays, as memory access on this test environment are very fast.

The tests will continue until final deadline.

Acknowledgement

¹ Rangan, R.; Vachharajani, N.; Vachharajani, M. e August., D. I. *Decoupled Software Pipelining with the Synchronization Array*. 2004.

² Lucas, D. C. S., Araujo, G. *The Batched DOACROSS Loop parallelization Algorithm*. 2015.