Efficient On-Chip Pipelined Streaming Computations on Scalable Manycore Architectures

Nicolas Melot\textsuperscript{\ast,1}, Christoph Kessler\textsuperscript{\ast,1}, Jörg Keller\textsuperscript{\dagger,2}

\textsuperscript{\ast} Linköpings Universitet, Dept. of Computer and Inf. Science, Linköping, Sweden
\textsuperscript{\dagger} FernUniversitat in Hagen, Fac. of Math. and Computer Science, Hagen, Germany

ABSTRACT

Performance of manycore processors is limited by programs’ use of off-chip main memory. Streaming computation organized in a pipeline limits accesses to main memory to tasks at boundaries of the pipeline to read or write to main memory. The Single Chip Cloud computer (SCC) offers 48 cores linked by a high-speed on-chip network, and allows the implementation of such on-chip pipelined technique. We assess the performance and constraints provided by the SCC and investigate on on-chip pipelined mergesort as a case study for streaming computations. We found that our on-chip pipelined mergesort yields significant speedup over classic parallel mergesort on SCC. The technique should bring improvement in power consumption and should be portable to other manycore, network-on-chip architectures such as Tilera’s processors.

1 Introduction

Scaling up the number of cores integrated in a single chip is made difficult by the mechanisms necessary to implement shared memory and keep high performance thanks to caches. In a shared memory system, all caches maintain a coherent view of a shared memory, which requires complex hardware that is hard to extend to serve many cores. The Single Chip Cloud computer (SCC) issued from Intel’s Tera-scale research program eliminates the hardware overhead required to maintain a consistent view of shared memory. Instead, consistent view of shared memory in caches is maintained through software protocols with the help of a non-uniform, low-latency shared on-die memory accessible by all 48 cores through a high-speed on-chip network.

Multicore and manycore processors can reduce energy consumption while providing better computation power, but their parallel nature makes them very challenging to program efficiently. Many

\textsuperscript{1}E-mail: \{nicolas.melot, christoph.kessler\}@liu.se
\textsuperscript{2}E-mail: joerg.keller@FernUni-Hagen.de
contributions address automatic or compiler-assisted parallelization of sequential programs, scheduling or synchronization issues for example. Some other contributions extend or create programming languages and provide constructs to help programmers to express parallelism, for example OpenMP or Streamit [TKA02]. Streamit organizes computation in a succession of filters linked by channels from which filters can read input data and write the corresponding output. Depending on their characteristics, they can be mapped to different cores or accelerators in order to achieve task or pipeline parallelism [UGT09].

The heavy use of main memory impacts both speed and energy performance of multi and many-core processors [HKK10]. On-chip pipelining [HKK10] reorganizes computations into a task graph where tasks can be compared to Streamit filters. Tasks are mapped to cores and forward intermediate results to the next tasks in the graph through on-chip network and distributed memory, or via cores’ local memory if mapped to the same core. This reduces accesses to main memory strictly to nodes having to fetch input data or write final computation results, and thus frees intermediate communications from costly latencies thanks to the high-speed on-chip network. Such property would benefit to software running on MPSoC, whose design is heavily constrained by energy consumption issues and that already makes heavy use of streaming computations; for instance to process image sequences or Fast Fourier Transform.

We use mergesort as a simple streaming computation case study to demonstrate the usefulness of the on-chip pipelined approach to parallel computation. We investigated performance and limitation of the SCC and we implemented a non-pipelined parallel version of mergesort as a comparison basis to the on-chip pipelined variant. We introduce an ILP (Integer Linear Programming) formulation to solve the task-to-processor mapping problem, taking into account the constraints measured in previous work. Finally, we implemented on-chip pipelined mergesort and compare its performance with non-pipelined parallel mergesort for SCC.

The rest of the paper is structured as follow: section 2 introduces the SCC architecture and the performance constraints we observed. Section 3 introduces the on-chip pipelining approach and its implementation on the SCC. Finally, section 4 concludes and introduces future work.

## 2 The Single Chip Cloud computer

The SCC provides 48 independent Intel x86 cores, organized in 24 tiles. Figure 1(a) provides a global schematic view of the chip. Tiles are linked together through a $6 \times 4$ mesh on-chip network. Each tile embeds two cores as well as a common message passing buffer (MPB) of 16KiB (8KiB for each core); the MPB supports direct core-to-core communication.

The cores are IA-32 x86 (P54C) cores which are provided with individual L1 and L2 caches of size 32KiB (16KiB code + 16KiB data) and 256KiB, respectively, but no SIMD instructions. Each link of the mesh network is 16 bytes wide and exhibits a 4 cycles crossing latency, including the routing activity.

Each core is attributed a private domain in main memory whose size depends on the total memory available (max 64 GiB). Six tiles (12 cores) share one of the four memory controllers to access cores’ private memory. Cores can access an off-chip shared memory whose size can vary up to several hundred megabytes. Note that private memory is cached on cores’ L2 cache but caching for shared memory is disabled or offers no coherency among cores’ caches. Intel provides the RCCE library which contains MPI-like routines to synchronize cores and allows them to communicate data to each other. RCCE also allows the management of voltage and frequency scaling.

Previous work assessing the SCC shows a significant performance penalty from the use of shared
main memory, compared to using private memory or the on-chip network [AMKK11]. It also indicates that the distance to the memory controller has an influence on round-trip time. Further tests indicate the difficulty to actually saturate the memory controllers using all cores to run read and write operations. We also see that there is more read than write memory bandwidth available to the cores [MAKK11]. Finally, we observe memory bandwidth drops with cache-unfriendly memory access patterns and even more drops with random access patterns [MAKK11].

Figure 1: (left) A schematic representation of the SCC die. (right) An on-chip pipelined task graph.

3 On-chip pipelined mergesort

On-chip pipelined mergesort organizes mergesort’s merging tree in a pipeline, with all tasks running concurrently on an arbitrary number of cores (see Fig.1(b) for a 6 cores example). Producer and consumer tasks mapped to the same core communicate through its local memory; those mapped to different cores use the mesh network and distributed on-chip memory. The leaf tasks start the computation by reading data from memory and they forward their intermediate merging results towards their consumer tasks at the next level in the task graph. These tasks can then start the same process, and so on, until all the tasks in the pipeline are active. This schema restricts memory writes to the root node. As the merge operation is done blockwise, follow-up tasks can start as soon as leaf tasks have produced their first block of intermediate results.

On-chip pipelined mergesort implemented on SCC follows the same process. Because of the high amount of cores available (48), the chip is divided into 8 groups of 6 cores each, all of them using the same memory controller but located on a different tile. This restricts the task mapping problem to a 6-level merging tree placed on 6 cores, simplifying significantly the work of the ILP solver to compute an optimal task mapping [MKA+12]. The symmetry of the SCC allows 8 replicas of this merging tree with no or marginal performance losses. The mergesort implementation runs 8 pipelined mergesort and merges the 8 sorted subsequences using a classic parallel mergesort and shared memory [MKA+12].

The on-chip pipelined phase of our implementation yields a speedup between 1.6 and 1.7, depending on the adaptations of the classic parallel mergesort we compare to [MKA+12]. Furthermore, we observed a speedup of 143% from a previously implemented on-chip pipelined mergesort over CellSort for the Cell B.E. [HKK10].
4 Conclusion

Although multi and manycore processors improve computation power and energy consumption over sequential processors, their performance is limited by main memory’s latencies. Organizing computation in a pipeline not only enables pipeline parallelism, but it can also reduce off-chip memory accesses, yielding better speedup. Our experiments, although still using suboptimal task mappings, show that this technique can provide good runtime improvements over classic parallel implementations. We believe these results can be further improved with a refined implementation and the use of optimal, ILP-generated task mapping. On-going work includes porting the on-chip pipelining technique to other calculations such as Fast Fourier Transform or other streamed computations. We expect the reduced usage of off-chip memory also yields better energy consumption. Furthermore, reduced and easy to control accesses to main memory yields determinism that enables the use of on-chip pipelining in high-performance real-time systems. The technique should be portable to other manycore, network-on-chip architectures such as Tilera’s processors or Adapteva’s Epiphany architecture.

Acknowledgments

The authors are thankful to Intel for providing the opportunity to experiment with the “concept-vehicle” many-core processor “Single-Chip Cloud Computer”.

This research is partly funded by the Swedish Research Council (Vetenskapsrådet), project Integrated Software Pipelining, and the CUGS graduate school at Linköping University.

References


