Memory-intensive parallel computing on the Single Chip Cloud Computer: A case study with Mergesort

Nicolas Melot*,[1], Kenan Avdic*,[1],
Christoph Kessler*,[1], Jörg Keller†,[2]

* *Linköpings Universitet, Dept. of Computer and Inf. Science, Linköping, Sweden*
† *FernUniversitat in Hagen, Fac. of Math. and Computer Science, Hagen, Germany*

ABSTRACT

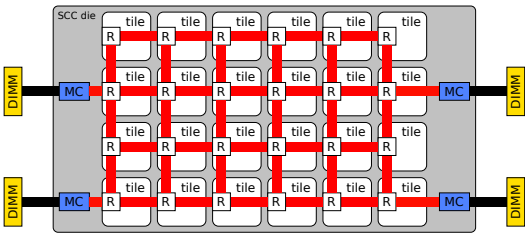
The Single Chip Cloud computer (SCC) is an experimental processor from Intel Labs with 48 cores connected with a 2D mesh on-chip network. We evaluate the performance of SCC regarding off-chip memory accesses and communication capabilities. As benchmark, we use the merging phase of mergesort, a representative of a memory access intensive algorithm. Mergesort is parallelized and adapted in 4 variants, each leveraging different features of the SCC, in order to assess and compare their performance impact. Our results motivate to consider on-chip pipelined mergesort on SCC, which is an issue of ongoing work.

1 Introduction

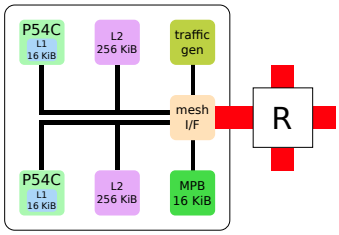
The SCC processor is a 48-core "concept-vehicle" created by Intel Labs as a platform for many-core software research. It features 48 cores in the same chip, which synchronize, communicate and perform off-chip memory operations through a 2D mesh on-chip network. This many-core architecture yields new considerations of classic algorithms such as sorting, which is a representative of the class of memory access intensive algorithms, i.e., with a low compute-communication ratio. Parallelization of sorting algorithms for multi-/many-core systems is an active research area, such as CellSort [GBY07], which leverage the SPEs on the Cell broadband engine. More recent work [HKK10] improves over these parallel sorting algorithms by applying on-chip pipelining, which optimizes off-chip memory accesses by forwarding of intermediate results via the on-chip network, and achieves a speedup of up to 143% over CellSort.

As the SCC and the Cell BE are very different architectures, implementing such on-chip pipelined sorting algorithm would enlighten the impact of these differences on performance. Parallel mergesort is a memory and communication intensive algorithm, and is thus suitable to stress the SCC regarding access to off-chip memory and on-chip communications. This paper[3] considers parallel mergesort as a benchmark for the SCC and defines and evaluates four variants of mergesort adapted to this chip.

[1]E-mail: {nicolas.melot, christoph.kessler}@liu.se, kenav350@ida.liu.se
[2]E-mail: joerg.keller@FernUni-Hagen.de
[3]An extended version of this paper appears in Proc. A4MMC-2011 workshop at ISCA, San Jose, June 2011, ACM.



(a) Global organization of the 24 tiles linked by an on-chip mesh network.



(b) A tile with two cores and their individual memory and MPB.

Figure 1: Organization of SCC.

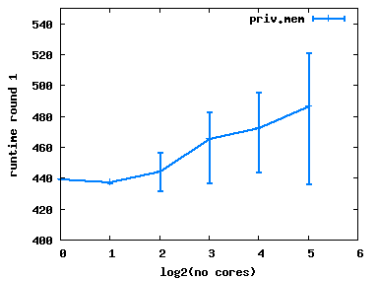
2 The Single Chip Cloud computer

The SCC provides 48 independent IA32 cores organized in 24 tiles. Figure 1(a) provides a global view of the organization of the chip. Each tile embeds a pair of cores and a message passing buffer (MPB) of 16KiB (8KiB per core) used for direct core-to-core communications. A tile is represented in Fig. 1(b). Six tiles share a memory controller and 4 memory controllers are available. Tiles are linked together and access memory controllers through a 6×4 mesh on-chip network working at up to 2GHz, and where each link (16 bytes wide) convey messages with a latency of 4 cycles. The cores can run at up to 800MHz and have their own caches L1 and L2 of size 16KiB and 256KiB, respectively. They do not provide SIMD instructions. Each core is given a private domain in main memory and a small part of this memory is shared among the cores, with a total of 64GiB maximum globally available. When the shared memory is cached, no hardware cache coherency mechanism is provided; it must be implemented in software.

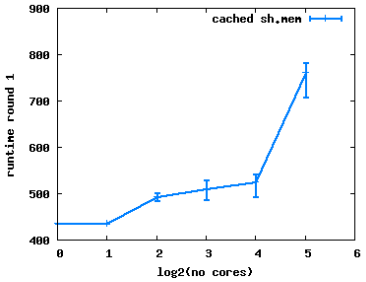
The SCC can be programmed using the framework RCCE provided by Intel, which comes in two versions: baremetal for OS development or on top of an individual Linux kernel per core. RCCE provides chip management routines (frequency and voltage scaling) as well as MPI-like functions to allow communication and synchronization between cores. Programming on the SCC is very similar to programming on a cluster using an MPI framework.

3 Mergesort variants for SCC benchmarking

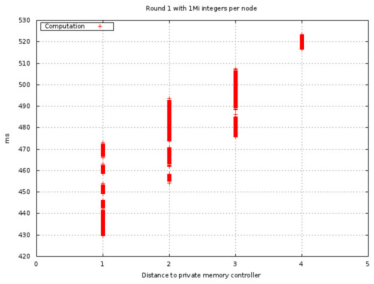
Parallel sorting algorithms have been investigated in the past. Mergesort is a good example of a memory access intensive algorithm, and is easy to parallelize. This makes it an ideal case to benchmark the SCC. All algorithms described here are adaptations of mergesort to the SCC's architecture. The *"private memory mergesort"* exploits cores' private memory to store data and shared off-chip memory to communicate it between cores. The algorithm starts with cores generating two pseudo-random blocks of data of size such that all blocks from all cores can all together fit in half of one core's private memory so it can merge two blocks into the final larger one. Then the algorithm performs a sequence of *rounds* as follows: A core merges its two blocks and sends the larger merged block to another core and turn idle, or receives another block and start a new round. In each round, half of the cores send their merged block and turn idle; the other half receive it and start a new round. In the last round, only one processor merges the final block. The communication process is implemented as follows: cores write a portion of their block in a portion of shared memory, then they send a flag to the recipient core through the on-chip network. The recipient copies this data and then sends a flag to the sender core to



(a) Timing of round 1 for private memory variant.



(b) Timing of round 1 for cached shared memory variant.



(c) Timing of private memory variant depending on distance to memory controllers.

Figure 2: Runtime differences with increasing core count

get the data left. Both cores can determine precisely the portion of shared memory they can use, the core they have to communicate with, and the amount of data to be transferred, from their id and the number of the current round. When the round ends, all non-idle cores cross a barrier and a new round begins until the whole data is merged.

The *"MPB private memory variant"* is similar to the process described above, but it uses the MPB and on-chip network to communicate blocks instead of the shared, off-chip memory. The two last variants *"Uncached shared memory variant"* and *"Cached shared memory variant"* differ by using shared memory only, storing directly their input block in it thus avoiding the need to communicate blocks from one core's private memory to another. The first shared memory variant does not use the cache whereas the second activates it. The cached shared memory variant makes sure that caches are flushed between rounds to guarantee a consistent view of memory in all reading operations.

4 Experimental evaluation

The performance of the SCC is assessed regarding off-chip memory access as well as communication latency and bandwidth and overall performance. Main off-chip memory is tested by running and monitoring the first round of the four variants with 1 to 32 processors, each processor merging blocks of 2^{20} integers. Results are shown in figure 2. It shows a growing processing time with the number of cores, which seems to indicate contention in off-chip memory bandwidth. First round of private memory variant is run on cores at different distance to the memory controller, and the time is depicted in figure 2(c). It demonstrates a growing impact of distance from cores to memory controller on their performance of off-chip memory access.

Communication latency and bandwidth are assessed through messages sent from one core to another, the cores separated from each other by a distance of 0 (same tile) through 8 (opposite corners on the die), as shown in the right part of figure 3. The left part of the same figure shows a slightly linearly growing time with distance between cores. When the size of blocks to be transferred equals the L2 cache size, this measured latency becomes higher and less reliable, due to too large blocks received and stored in L2 together with operating system running data and resulting in cache misses.

Finally, all variants are run several times with a growing number of integers to be merged and different number of cores involved. The results are depicted in figure 4. The private memory variant is clearly the one with the worst performance, and cached shared memory is much better. Shared memory variants have the advantage not to have transmission between rounds, but uncached shared memory variant is not much better than private memory variant. The MPB variant, even though it needs a

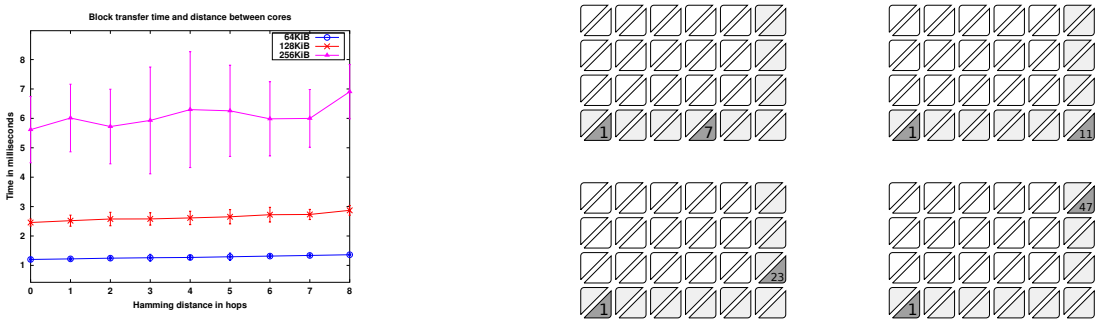
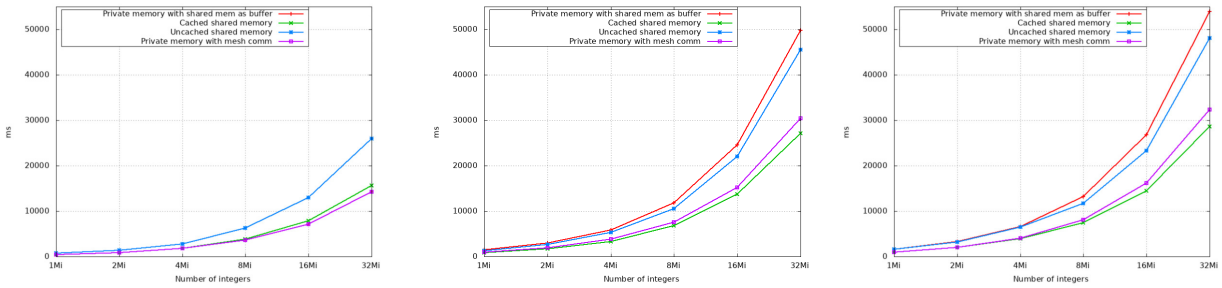


Figure 3: Four different mappings of core pairs with a different and growing distance.



(a) Merging time using 1 core. (b) Merging time using 8 cores. (c) Merging time using 32 cores.

Figure 4: Performance of the four mergesort variants on SCC, sorting integer blocks with randomly generated data of different size and using different numbers of cores.

communication phase, is not much worse than the cached shared memory variant. This denotes a low performance impact from using the on-chip network, which is good for on-chip pipelining.

5 Conclusion and ongoing work

Mergesort is an interesting case of a memory access intensive algorithm, which makes it suitable benchmark to stress the SCC and monitor its performance regarding off-chip memory access and on-chip communication. The experiments enlighten important performance factors of SCC. The results motivate the implementation of on-chip pipelined mergesort, which is an issue of on going work. On-chip pipelining reduces off-chip memory accesses with more on-chip communication, and is expected to bring significant speedup because of both poor performance of off-chip memory and low impact from the use of the on-chip network. Performance may even be improved by optimizing the tasks' placement on cores, considering the distance between cores and distance to the memory controllers.

References

[GBY07] B. Gedik, R.R. Bordawekar, and P.S. Yu. Cellsort: high performance sorting on the Cell processor. In *VLDB '07 Proceedings of the 33rd international conference on Very large data bases*, 2007.

[HKK10] R. Hultén, J. Keller, and C. Kessler. Optimized on-chip-pipelined mergesort on the Cell/B.E. In *Proceedings of Euro-Par 2010*, volume 6272, pages 187–198, 2010.