Computing in Science & Engineering, copublished by the IEEE CS and the AIP
vol. 13(6), pp. 79-83, Nov/Dec 2011
DRAFT accepted for publication
see http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6077845 for final version

Editors: Volodymyr Kindratenko, kindr@ncsa.uiuc.edu
Pedro Trancoso, pedro@cs.ucy.ac.cy

SCC: A Flexible Architecture for Many-Core Platform Research

by Matthias Gries, Ulrich Hoffmann, Michael Konow, and Michael Riepen

The Single-chip Cloud Computer (SCC) experimental processor by Intel Labs is a "concept vehicle" aimed at scaling future multicore processors and serving as a software research platform.

To mitigate the power wall's effect on CPU performance, the industry has switched to multicore designs in virtually all mainstream application markets within the last five years. According to Moore's law, increasing transistor integration will let us build many-core designs to further improve energy efficiency of logic operations.[1] Intel Labs' Single-chip Cloud Computer (SCC) experimental processor,[2] a 48-core concept vehicle incorporating technologies intended to scale multicore processors to 100 cores and beyond, was also created as a platform for many-core software research. It exposes many hardware "knobs" to system software and applications to foster outside-the-box experiments for new options in how to organize and coordinate hardware and software. We'll be able to leverage many-core efficiency only if we can match the parallelism in the hardware architecture with concurrency in the software stack.

Here, we'll highlight SCC's capabilities for flexible use of on-die and off-die resources for efficient synchronization and communication among cores. These capabilities expand the design space for exploration of an Intel Architecture (IA) platform significantly. As the "MARC: The Many-Core Applications Research Community" sidebar describes, the synergies within Intel's MARC program underpin SCC's usefulness as a flexible research tool for assessing the possible software and hardware features of future many-core architectures.

SCC's Top-Level Architecture

As Figure 1 shows, SCC is organized in 24 tiles connected by a 6×4 rectangular 2D mesh topology.[2,3] Each tile consists of two IA32 cores based on P54C Intel Pentium® in-order execution cores. Each tile is connected to a mesh router.

The 48 cores can communicate over the on-die network using a message-passing architecture that allows data sharing with software-maintained memory consistency. Four independent double data rate type three (DDR3) memory channels provide main memory capacity. The processor is connected to other platform components by a proprietary system interface (SIF) that's connected to a SCC router port. On the hardware platform, a field-programmable gate array (FPGA) acts as a reconfigurable chipset, providing I/O such as PCI Express (PCIe) end point and Ethernet MACs. A management console PC (MCPC) connected over PCIe provides the framework for initializing, monitoring, and configuring the SCC platform. SCC offers fine-grain Dynamic Voltage and Frequency Scaling (DVFS) infrastructure by controlling voltage on groups of four tiles and frequency on each tile.

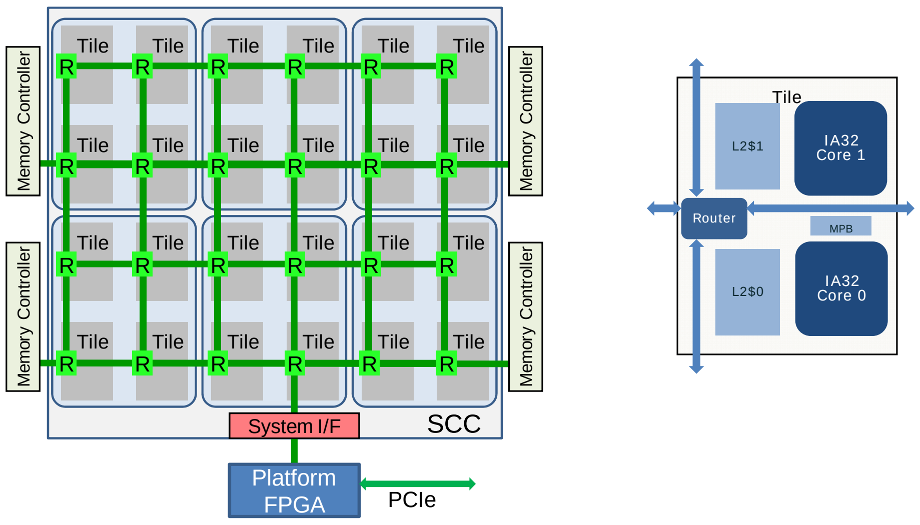

Figure 1. The Single-chip Cloud Computer (SCC). (a) Top-level system overview. (b) Tile structure.

Memory Hierarchy

Each SCC core contains 16 Kbytes L1 data and instruction caches with 32 bytes line size and a 256 Kbytes private L2 cache. Each tile is connected to a mesh router that provides the connectivity to four independent memory channels on the mesh interconnect's periphery. Each DDR3 channel can be populated with up to two 8-Gbyte Dual Inline Memory Modules (DIMMs), leading to up to 64 Gbytes of main memory capacity. Each individual core can access up to 4 Gbytes of memory divided into private and shared sections. Access to shared sections—which are potentially visible to all cores—is uncached by default because SCC doesn't contain hardware mechanisms to maintain coherent caches.

The programmer can distribute the address space for each core over all four memory controllers in 16-Mbyte quantities. For this, there's a 256-entry lookup table (LUT) near each core that maps individual core addresses to system-wide addresses. System addresses enable the definition of shared memory areas seen by some or even all the cores. Configuration registers on each tile are memory mapped. Programmers can access registers—such as those for test and set bits and frequency control of individual tiles from any core—providing a rich fabric for software-managed policies.

The integrated memory controllers (iMCs) are responsible for issuing data transfers in-order while interleaving control sequences for memory banks and ranks. Due to the high number of cores and address mapping offered at tile and controller levels, the locality of memory requests enqueued in front of the iMCs is low. Because the available number of 32 memory banks per channel is high relative to the core count, interleaving of control sequences to different banks and ranks is a decent choice for increasing the achievable bandwidth, combined with closed-page mode.

Figure 2 shows how a memory channel can be configured to model memory-bound scenarios. Different operating points are selected as specified by their frequencies—tile (T), mesh (R) and DDR3 (M). We run the memory-bound stream benchmark on each core while adding more and more cores to one DDR3 controller to saturate the channel; that is, one individual core can't saturate the memory channel.

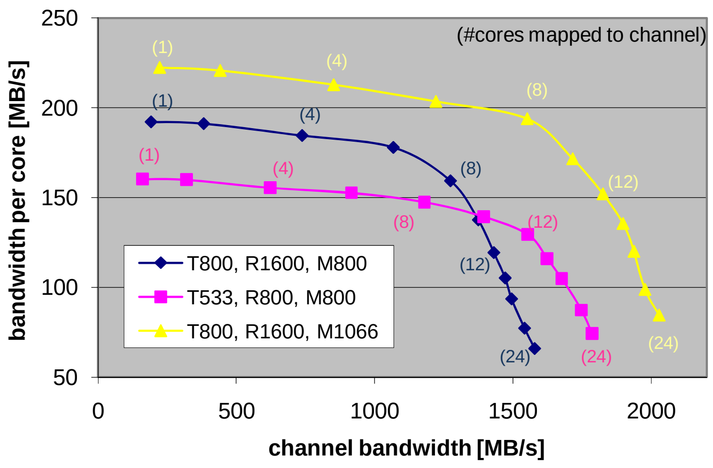

Figure 2. Configurable DDR3 channel operating points. The user has the choice between more compute-bound (T533,R800,M800) or memory-bound scenarios (T800,R1600,M800) to be leveraged by the software stack.

Figure 2 illustrates two types of behavior. Configurations (T533, R800, M800) and (T800, R1600, M1066) provide almost fair service up to 12 cores mapped to one channel, with a relatively flat access latency. Because SCC provides four iMCs for 48 cores, a memory mapping can be achieved where all cores work virtually independent of each other because the average throughput varies only by approximately 10–20 percent. On the other hand, configuration (T800, R1600, M800) better represents the setup that we find in current out-of-order processors. Starting with about six cores, the available memory bandwidth must be split among competing cores, leading to a significant increase in access latency.

Consequently, we have the option to choose an operating point for the memory subsystem such that the memory service visible by individual cores is virtually independent from the workload of other cores if cores are mapped to channels in a balanced way (12 cores per channel). In this "sandboxed" operation mode, individual cores behave more deterministically than current multicore designs, enabling research on algorithmic scalability in parallel software. The DDR3 memory subsystem also provides configurations where SCC behaves as if it were memory-bound, which permits exploration of policies for resource management and sharing.

Communication Architecture

SCC provides mechanisms for fast on-die communication that we can use for explicit inter-core message passing instead of providing hardware-maintained cache coherence for shared memory operations. For message passing, each tile has dedicated 16 Kbytes of fast on-die memory—the *message-passing buffer* (MPB)—which results in a total of 384 Kbytes on die. The distribution of this memory isn't fixed. By default, each core has its private chunk of 8 Kbytes allocated in the local tile.

However, a programmer can also allocate all 384 Kbytes to one core. Any mix in between is possible as well. Because the MPBs are located in the tile, locality matters in terms of latency.

We introduce a new memory type—the *Message-Passing Memory Type* (MPMT)—that lets users differentiate between normal data transfers and message-passing-related memory transfers. The programmer can enable this type as a page-table attribute. The effect is that cache lines belonging to this type are cached only in the level-1 cache and bypass the level-2 cache completely for efficient on-die and off-die communication. A new instruction (`INVDMB`) lets us invalidate all MPMT cache lines in the level-1 cache within a single cycle. In addition, a write-combine buffer assembles partial writes to 32-byte block transfers to reduce the number of messages over the on-die mesh. Because read and write accesses beyond a cache line's size aren't atomic, the architecture also features 48 test and set registers (two registers at each tile) to allow atomic reads and writes in any shared memory area.

Although we originally brought in all of these features to work with the fast on-die MPBs, we can also apply them to memory areas in the DDR3 main memory. During the development of an MPICH2-based SCC-specific MPI library[4] we found out that a combination of MPB-based and DDR3-based transfers results in optimal performance. For large messages, MPB-based communication requires splitting data into several small chunks, leading to lower performance due to associated management overhead. In contrast, small messages, such as collective operations (such as barriers), or any sort of flags profit from the low-latency transmission through the MPBs, in which they easily fit. Our RCKMPI implementation uses part of the MPB to exchange flags and flow control data in a low-latency, lock-free manner. We use the remaining MPB capacity for short messages, while large messages are immediately transferred through DDR3 memory to prevent the serialization overhead that the MPB requires.

As Figure 3 shows, the combination of MPB and shared memory (SHM) DDR3 usage leverages the advantages of both worlds by offering the lowest latency for varying message lengths.

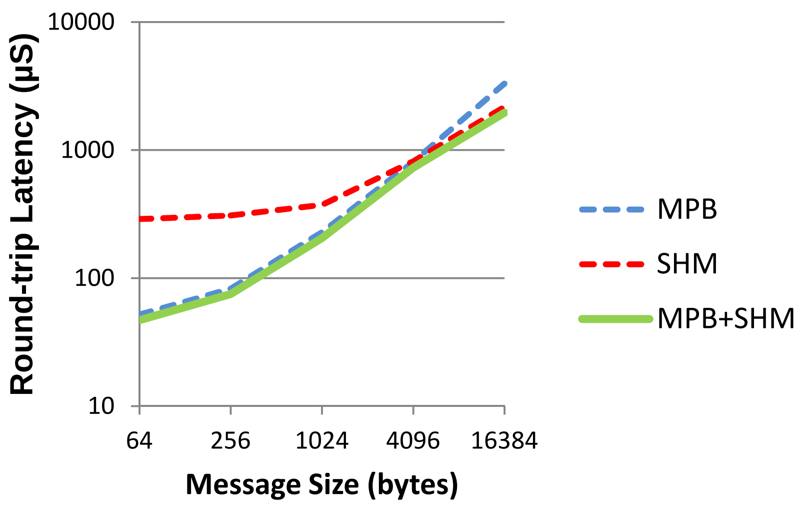

Figure 3. Roundtrip latency for RCKMPI channels on 48 cores (T800, R800, M800). Depending on the message size, either the MPB-based or the shared memory (SHM) DDR3-based MPI implementation performs better. Combining both (MPB+SHM) provides best performance for all message sizes and underpins the usefulness of the new Message-Passing Memory Type (MPMT) applied to both MPB and DDR3 for achieving lowest communication latency.

Exploratory Use of SCC

At first glance, the architecture looks like a cluster-on-a-chip consisting of 48 general-purpose processor cores combined with a high-speed network for communication and data sharing. Given the flexibility built into the architecture, researchers have been tuning SCC's characteristics in directions far beyond its apparent use model as a cluster.

Imagine a use case in which SCC models a heterogeneous architecture that consists of general-purpose cores and special-purpose accelerators. The accelerators are implemented with bare-metal kernels that run on top of some of the general-purpose cores to model a custom device's behavior. Other cores still run operating systems and control the accelerators through an API. SCC provides all required features to realize this approach:

- We can fully isolate cores from each other by restricting access to their private memory areas.
- Each core can have its own boot code.
- Core-to-core interrupts can be sent by addressing memory mapped registers in the tiles.
- Data sharing can be done explicitly through the MPB or DDR3 memory.

Researchers from the University of Amsterdam apply this approach for modeling direct memory access (DMA) engines on SCC to efficiently copy large chunks of memory concurrently to computations.[5] They use an asynchronous message-based protocol to control these "copy accelerators." DMA commands are written to the accelerator core's MPB followed by an interrupt to start the copy process. The kernel's code is small enough to fit into the core's 16 Kbytes instruction cache. This prevents recurring code accesses to memory. Therefore, the model's behavior looks very similar to a custom implementation of a DMA engine in hardware.

We can also express heterogeneity by mixing a few big cores with an array of small cores. Frequency scaling on SCC can be applied statically to create a heterogeneous model by scaling the frequency up for some cores, making them look bigger (execute faster) compared to an array of small cores that run at a lower frequency. In additional, the available memory bandwidth can be adjusted to the cores' size by modifying the frequencies of the attached memory controllers. The configuration of big cores versus small cores can be changed during runtime by dynamically changing the core frequencies. This permits exploration of various compositions of big cores and smalls cores on a many-core architecture.

To make the setup even more asymmetric, we can include the big cores from the management PC's CPU (MCPC) into the model as well. The SCC's system interface exposes the mesh network to the attached FPGA and therefore to the MCPC. By tunneling the mesh packets through PCI Express to the MCPC, we can virtually connect the MCPC's CPU to the SCC's mesh as another execution node. Researchers at the Rheinisch-Westfälische Technische Hochschule (RWTH) Aachen have developed the SCC-MPICH library,[6] which can launch MPI processes on the SCC and the MCPC. The communication between the processes is aware of the hierarchy such that it uses the on-die MPB for communication between SCC cores and shared memory for the communication between SCC and MCPC cores.

A key feature that makes SCC a flexible architecture for software and programming research is the lack of hardware-managed coherence. This adds flexibility to the architecture to explore new ways of maintaining coherence on many-core architectures. Given architectures with hundreds or thousands of cores, software management of hardware-supported coherence might have advantages over software-oblivious hardware-managed coherence across all cores. Managing coherence completely in software gives broad flexibility and enables application-specific optimizations, but might add significant overhead whenever memory areas must remain coherent among many processing cores. A hybrid solution might be the best tradeoff between complexity and efficiency for large-scale many-core architectures. With SCC, we can model different flavors of hybrid coherence models to answer research questions such as

- What is the reasonable size of a coherence domain in many-core architectures?
- How much flexibility is required in creating and managing coherence domains?
- How much support in hardware is required for an efficient implementation?

System software with a software layer that resides below traditional operating system functions can model coherence for a defined number of cores. We'd like to dynamically configure both the number of cores that are part of one coherent domain and the number of independent coherence domains. Between coherence domains, we can enable data sharing explicitly using message-passing protocols. Intel researchers are working on a shared virtual memory framework for SCC that lets cache-coherent domains spawn across multiple SCC cores.[7]

Regarding more software-managed usages of on-die memory resources, the University of Bayreuth is looking into task parallelism on the SCC using its flexible hardware design.[8] To develop an efficient runtime scheduler, they use the MPB as a scratchpad memory instead of a message-passing buffer. This scratchpad holds task counters and double-ended queues (deques) to store tasks descriptors. This enables the creation of an efficient runtime scheduler. When necessary, they use the test and set bits for locking the access to the deque entries.

These are all great examples of exploratory use cases of the SCC platform beyond the "intended use" of the hardware building blocks, opening the door to many new research ideas to come.

Our ongoing and vivid interaction with the MARC community has led to a variety of extensions to the original FPGA design connected to SCC's system interface. This has broaden the design space for experiments—including more I/O connectivity (Ethernet and SATA interfaces) for more embedded setups—providing a global interrupt source and a bank of atomic counters. Because the FPGA is physically connected to SCC's mesh fabric, these extensions appear to be integral parts of SCC. Using high-speed serial I/O connectors on the board, we can virtually expand the on-die network across systems to emulate a CPU with hundreds or thousands of cores.

Intel Labs' experimental SCC has inspired many academic and industrial researchers to create a whole new design space for many-core SCC experiments. As the examples here show, the community is active and creative in using the SCC as a research tool, based on its flexible and software-visible architecture coupled with the platform FPGA's extensibility. SCC's application domain for many-core research ranges from established cluster configurations to exploratory OS and user-managed scenarios previously unseen on Intel Architecture.

Acknowledgements

We thank our colleagues from SCC design teams in the US, India, and Germany for making the SCC a reality.

References

1. S. Borkar, “Thousand Core Chips—A Technology Perspective,” *Proc. 44th Annual Design Automation Conf.* (DAC), ACM Press, 2007, pp. 746–749.
2. J. Howard et al., “A 48-Core IA-32 Processor in 45nm CMOS Using On-Die Message-Passing and DVFS for Performance and Power Scaling,” *IEEE J. Solid State Circuits*, vol. 46, no. 1, 2011, pp. 173–183.
3. P. Salihundam et al., “A 2 Tb/s 6 4 Mesh Network for a Single-Chip Cloud Computer With DVFS in 45 nm CMOS,” *IEEE J. Solid-State Circuits*, vol. 46, no. 4, 2011, pp. 757–766.
4. I.A. Comprés Ureña, M. Riepen, and M. Konow, “RCKMPI—Lightweight MPI Implementation for Intel’s Single-Chip Cloud Computer,” *Proc. European MPI Users' Group Meeting* (EuroMPI). Y. Cotronis et al. (Eds.), LNCS 6960, Springer-Verlag, Sept. 2011, pp. 208--217
5. M.W. van Tol et al., “Efficient Memory Copy Operations on the 48-Core Intel SCC Processor,” *Proc. 3rd Many-Core Applications Research Community Symp.*, KIT Scientific Publishing, 2011
6. C. Clauss et al., “Recent Advances and Future Prospects in iRCCE and SCC-MPICH,” 3rd Many-core Applications Research Community Symp., KIT Scientific Publishing, 2011
7. X. Zhou et al., “A Case for Software Managed Coherence in Many-Core Processors,” *Proc. 2nd Usenix Workshop on Hot Topics in Parallelism*, (HOTPAR), Usenix Assoc., 2010*;* www.usenix.org/event/hotpar10/final_posters/Zhou.pdf.
8. A. Prell and T. Rauber, “Task Parallelism on the SCC,” *Proc. 3rd Many-Core Applications Research Community Symp.*, KIT Scientific Publishing, 2011

MARC: The Many-Core Applications Research Community

In 2010, Intel Labs established the Many-Core Applications Research Community (MARC http://communities.intel.com/community/marc) with the goal of establishing a global community to accelerate the evolution and adoption of many-core software and hardware technologies. MARC’s first Intel research platform is the Single-chip Cloud Computer (SCC).

Initially, more than a hundred active participants started using the community site and the forum. Today, more than 300 researchers from 90 research institutions in more than 20 countries are part of MARC and pursue research on various research topics. Among the most popular ones are parallel programming models, message-passing algorithms, power management, and many-core operating systems. To date, more than 50 SCC-related papers have been submitted to major conferences.

Intel Labs encourages the community to contribute software to the project, have peer discussions, and share ideas. To foster these goals, international MARC symposia are hosted several times a year, with the research community gathering to present their progress and discuss new ideas. Intel Labs has hosted three initial MARC symposia in Braunschweig, Germany; Santa Clara, California; and Beijing, China. In July 2011, the first MARC symposium not hosted and organized by Intel Labs, was presented by a MARC partner—the Fraunhofer-Gesellschaft in Germany—and the proceedings were published for the first time.

Matthias Gries *is a research scientist at Intel Labs Braunschweig in Germany, where he is a member of the design team of SCC’s DDR3 memory subsystem. His research interests include IA platform architectures, benchmarking, and mitigating the memory wall. Gries has a PhD in technical sciences from the Swiss Federal Institute of Technology (ETH) Zürich. Contact him at matthias.gries@intel.com.*

Ulrich Hoffmann *is a Technology Strategist at Intel Labs Braunschweig. He manages the Many-core Applications Research Community (MARC) in Europe and drives technologies from Intel Labs to adoption at internal and external customers. Hoffmann has a diploma degree in electrical engineering from the Braunschweig University of Technology. Contact him at ulrich.hoffmann@intel.com.*

Michael Konow *is an engineering manager at Intel Labs Braunschweig, where his team developed the core* *subsystem, validated the silicon and hardware platform, and implemented the software framework for SCC. His research interests include processor architectures, embedded systems, and FPGA prototyping. Konow received a diploma degree in electrical engineering from the Braunschweig University of Technology and is the recipient of two Intel Achievement Awards. Contact him at michael.konow@intel.com.*

Michael Riepen *is a senior research engineer at Intel Labs Braunschweig, where he worked on SCC pre- and post-silicon validation and created SCC's management software framework. His research interests include many-core programmability, validation methodologies, and exascale computing. Riepen has an MSc in computer science from the University of Applied Sciences Wedel, Germany, and is recipient of an Intel Achievement Award. Contact him at michael.riepen@intel.com.*

The Single-chip Cloud Computer (SCC) experimental processor is a 48-core "concept vehicle" created by Intel Labs incorporating technologies intended to scale future multicore processors and to serve as a platform for many-core software research.

Keywords: Message Passing, Intel Architecture, Scalability, software coherence, many-core, on-die communication