From Parallel Systems to Wireless Sensor Networks and Back

Jana Traue, Reinhardt Karnapke, Jörg Nolte
Distributed Systems/Operating Systems Group
Brandenburg University of Technology
Cottbus, Germany
{jtraue,karnapke,jon}@informatik.tu-cottbus.de

***Abstract*—Many researchers working in the field of wireless sensor nodes have their roots in parallel and distributed computing. Coming from parallel systems, they saw wireless sensor networks as low end parallel systems, with lots of additional challenges. In this paper we propose taking the step back, from wireless sensor networks to parallel systems. In our opinion, the lessons learned from wireless sensor networks should be applied to a special case of parallel systems: Future many-core systems.**

I. Introduction

Parallel and distributed systems have been used for many decades, while wireless sensor networks have been a research focus for little more than one decade. In the beginning of research on wireless sensor networks, many researchers saw them as an extremely challenging type of parallel system, with the sensor nodes as cores and their radio modules as interconnection network. In this representation, the cores are slow processors with a small amount of memory. The interconnection network has a very low bandwidth, high latency and is prone to message losses, presenting the researchers with a lot of additional challenges.

The increase in performance seen in single processor systems in the last decades will not continue much longer. For this reason, multi-core systems have been developed, which currently feature two to four, sometimes 8 cores. But these multi-core systems will be replaced by many-core systems with dozens or even hundreds of cores in the near future. Many-core systems with 48, 64 or 100 cores are already available to researchers and will probably find their way into standard issue desktop computers soon.

Operating systems for these many-core systems will experience many of the problems currently found in operating systems for wireless sensor networks. Therefore we propose a return to the roots: Take the lessons learned in the resource constrained wireless sensor networks back to the world of parallel programming, namely to the many-core systems.

The next section takes a closer look at the development from parallel systems to wireless sensor nodes before a look on single processor systems and their development to multi- and many-cores is taken in section III. The problems that will be experienced when using many-core systems are described in section IV, followed by a short description of current challenges in wireless sensor networks in section V. The similarities of the problems experienced in both worlds are discussed in section VI before the concept of taking a step back is described in section VII. Concluding remarks are given in section VIII.

II. From Parallel Systems to Sensor Networks (history)

Wireless sensor networks (WSN) are envisioned to consist of thousands of nodes, making the usage of low-cost parts necessary. This low-cost requirement results in slow processors with a little bit of memory, cheap transceivers with small bandwidth and lossy communication. Even though these restrictions make working with wireless sensor networks harder than working in traditional parallel systems, they attracted a lot of attention from researchers around the world. Many of those researchers came from the world of parallel systems, and envisioned wireless sensor networks to be a special case of parallel system. While parallel systems often use shared memory as means of communication, sensor nodes use messages, but methods have been invented to build one on top of the other. An example for this is TinyDSM [1], where a distributed shared memory is realized for wireless sensor networks.

III. Single-core and Multi-core versus Many-core (definitions, now)

A. Single-core to Multi-core

In the last decades, the number of transistors on integrated circuits have doubled every two years or faster, in accordance with Moore's Law. While this increase in number of transistors has resulted in ever faster processors for a long time, recent years have seen a paradigm shift from single core CPUs to multi-core CPUs. The single core CPUs are no longer the focus of research due to two major problems: First, doubling the number of transistors on a single core does not double the processor's speed. Instead, it only increases the performance by 40-70 %. Second, the power consumption grows with the number of transistors, resulting in problems for power - and heat dissipation. For these reasons, current desktop and laptop computers feature between two and four cores, with some featuring 8 or 16 (using hyper threading).

The change from single core to multi core CPUs has posed a huge problem for operating system developers. Previously used locking mechanisms like interrupt locks were suddenly not sufficient anymore. Still, concurrent access to the kernel could lead to inconsistency in kernel data structures and had to be avoided at any cost. In Linux, this cost was paid at first by the introduction of the big kernel lock, which prevented concurrent access by allowing only a single process to enter the kernel, independent of the number of cores.

While this approach was effective, it was not efficient. When a process running on one core accessed the kernel, all other processes that wanted to access the kernel had to wait, drastically reducing the gain of parallelism. The solution for this problem was a replacement of the big kernel lock with lots of individual locks. Now, each data structure in the kernel is protected by an individual lock, making parallel access to the kernel possible as long as different processes want to use different services and work on different data structures. But replacing the big kernel lock with individual locks took 10 years[1], and increased the number of different locks drastically. Therefore, the danger of implementing a deadlock has risen.

B. Multi-core to Many-core

In 2007, several publications predicted the production of many-core processors. While multi-cores contained up to eight cores at this time, many-cores were supposed to consist of up to thousand cores [2], [3]. One of the first many-cores that was commercially available, the Tile64 [4] that contained 64 cores, was released by Tilera in the second half of 2007. A distributed cache coherence protocol assured coherence of the caches by using one of five fast on-chip networks. The other networks were dedicated to a special purpose, like serving memory requests from one of the four memory controllers, too.

While Tilera's processors were commercially available, Intel announced its 80 core Terascale [5] processor that was intended for research. Its successor, the Single Chip Cloud Computer (SCC) [6] that comprises 48 cores, was released in 2010. Each of the SCC's modified Pentium cores had its own 16 KB L1 caches and a 256 KB L2 cache. Unlike the Tile64, the hardware did not manage coherence of the caches. Sharing memory between multiple cores was still possible, but consistency had to be assured by software. If these software solutions had to rely on the external main memory modules, communication would have been very slow. In order to avoid such a problem, the SCC featured a fast, on-chip SRAM that was intended for message passing (see figure 1).

IV. Problems of future Many-core Systems

Problems that already arose with multi-cores become even more important with the increasing number of cores. In a current NUMA architecture, the memory access time ranges from 258 to 336 cycles depending on the distance between

[1]http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=commitdiff;h=4ba8216cd90560bc402f52076f64d8546e8aefcb, last visited July 2012

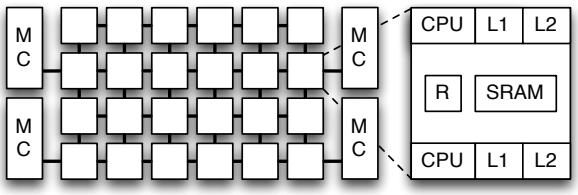

Fig. 1. Schematic of the SCC. 24 tiles are arranged in an 8x6 grid. Each tile comprises two P54C cores, a router and on-chip SRAM. Four memory controllers allow to connect up to 64 GB of main memory.

core and memory controller [7]. This difference is expected to grow and, consequently, a memory bound process should be placed next to a memory controller rather than in the middle of the grid. Choosing cores for processes, a task that is typically handled by the scheduler, is tightly coupled to the memory management. A nearby memory controller is useless if the process frequently accesses data that is hosted by a different controller. Such a usage pattern has to be detected and in some cases processes have to be migrated in order to achieve a better performance. Both of the many-core processors described above are connected to four memory controllers which might become a bottleneck. Therefore, and because of the difference in access times, it is of utmost importance to utilize the caches.

Current systems, like Linux, run a single instance of the operating system (OS) on the whole system. Several research projects that develop operating systems for many-cores (e.g. Corey [7], Barrelfish [8] and fos [9]) agree that a minimalistic kernel should run on each core of a many-core processor. These systems' scalability is expected to benefit from kernel instances that do not share data and communicate explicitly via message passing. With the currently used point-to-point on-chip communication networks, messages that take the same route may cause congestion.

In addition to the above mentioned challenges, hardware developers have to face the problems of power consumption and heat distribution. Recent work has shown that software that halts idle cores may halve the SCC's power consumption [10] and, hence, further research into energy efficient software is worthwhile.

V. Current Problems in Wireless Sensor Networks

Currently available sensor nodes are stronger than first generation sensor nodes, featuring more memory, faster CPUs and better transceivers. Still, they remain within the category of embedded devices. Common examples include the Tmote Sky from Mote IV Corporation (8MHz microcontroller, 10kB RAM and 48kB Flash) or the eZ430-Chronos from Texas Instruments (20MHz microcontroller, 4kB RAM, 32kB Flash). Especially the small amount of RAM which must house the data structures of application, communication protocols and operating system remains a big challenge for sensor network application developers.

TABLE I
CHALLENGES SHARED BY WIRELESS SENSOR NETWORKS AND MANY-CORE SYSTEMS

Challenge	Problem in WSN	Problem on Many-cores
Reduce energy consumption	Battery lifetime	Heat dissipation
Improve communication patterns	Energy cost of transmissions	Network congestion
Handle message loss	Lossy medium and changing links	Network congestion
OS memory consumption	Small available memory	Avoid off-chip memory access

However, it is not the only problem. A single node that senses data is normally not sufficient for application requirements and nodes need to cooperate. This cooperation is done by exchanging messages. Those messages must be transmitted over lossy wireless links, which are not stable over time, resulting in the need for adaptive communication protocols.

The third big problem is energy: Sensor nodes are usually powered by batteries. These batteries provide an upper bound for the time a node can spend in active mode. However, lifetime goals for senor networks are often envisioned in the scope of years, making energy conservation techniques necessary.

VI. SIMILARITIES BETWEEN PROBLEMS IN MANY-CORE SYSTEMS AND WSNS

While single core processors have reached a speed of some GHz, currently available many-core systems use a much reduced speed. The development of future many-cores will probably be built upon slower, smaller cores, but use huge amounts of them on a single die [2]. At the same time, wireless sensor nodes are increasing in performance and memory supply.

While both are still worlds apart, there are a number of similar problems in both worlds, even though the reasons for them are different.

Table I shows some of the similarities between the problems experienced by wireless sensor networks and many-core systems. Both systems must be designed with a close look at the energy consumption: sensor networks need to be careful about the lifetime reachable with a set of batteries while many-core systems get problems with the heat distribution. In sensor networks, the radio module is one of the main consumers of energy, therefore the communication patterns must be designed with the energy constraints in mind. In many-core systems with message passing, the interconnect network between cores becomes a bottleneck and network congestion should be avoided. Both systems suffer from message loss and must use appropriate recovery mechanisms. In WSN, the losses are most often due to the lossy medium and changes of the logical topology, meaning the changing links between nodes. While the medium (wire) is not lossy and the connections do not change in many-core systems, messages still get lost due to congestion. Sensor nodes need a small operating system that fits into their memory, leaving enough room for communication algorithms and application. In many-core systems, the most basic parts of the operating system should fit into the caches, which are not much larger than the RAM of the sensor nodes. Otherwise, the latencies caused by main memory accesses slow the system down and, even worse, the memory controllers become a bottleneck.

VII. FROM SENSOR NETWORKS (BACK) TO PARALLEL SYSTEMS

As the last section has shown, there are many similarities between the problems experienced in wireless sensor networks and those for future many-core systems. Finding new solutions to those problems of many-core systems seems unnecessary, as we have already worked on those same problems for more than a decade, only in a different environment. This research on wireless sensor networks has led to a huge number of solutions for different problems, situations and applications.

One of the more obvious solutions seems to be to use a miniature operating system which will be present on each of the cores of the many-core systems. This is already done in wireless sensor networks, where each of the nodes has its own copy of the operating system for obvious reasons. Operating systems for many-cores that already exist like Corey, Barrelfish and fos agree in using a minimalistic kernel on each core.

The next step seems to be identifying more applicable solutions from wireless sensor networks and moving them to the many-core systems. For example, the heat dissipation problems can be tackled by the usage of duty cycling in combination with power aware scheduling, where scheduling refers to the selection of cores on which a certain process will be run.

VIII. CONCLUSION

Many researchers working on wireless sensor networks have a background in parallel computing, some have started their careers in the world of parallel systems. While there is still a lot of research needed before wireless sensor networks can be made usable for the broad public on a large scale, we argue that it is time to take a break and reflect on what has already been achieved. With this reflection it is possible to identify those solutions from wireless sensor networks, that can be taken back to the world of distributed and parallel systems. Especially operating systems for the upcoming many-core systems can benefit a lot from the experiences already gathered.

IX. ACKNOWLEDGMENTS

The authors wish to thank Intel Research Braunschweig for granting us access to the SCC and the Many-core Application Research Community (MARC) program.

REFERENCES

[1] K. Piotrowski, P. Langendoerfer, and S. Peter, "tinydsm: A highly reliable cooperative data storage for wireless sensor networks," in *Proceedings of the 2009 International Symposium on Collaborative Technologies and Systems*, ser. CTS '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 225–232. [Online]. Available: http://dx.doi.org/10.1109/CTS.2009.5067485

[2] S. Borkar, "Thousand core chips: a technology perspective," in *Proceedings of the 44th annual Design Automation Conference*, ser. DAC '07. New York, NY, USA: ACM, 2007, pp. 746–749. [Online]. Available: http://doi.acm.org/10.1145/1278480.1278667

[3] A. Agarwal and M. Levy, "The kill rule for multicore," in *Proceedings of the 44th annual Design Automation Conference*, ser. DAC '07. New York, NY, USA: ACM, 2007, pp. 750–753. [Online]. Available: http://doi.acm.org/10.1145/1278480.1278668

[4] T. Corporation, "Tile64 processor - product brief," Tilera Website, http://www.tilera.com/sites/default/files/productbriefs/PB010_TILE64_Processor_A_v4.pdf.

[5] T. G. Mattson, R. Van der Wijngaart, and M. Frumkin, "Programming the intel 80-core network-on-a-chip terascale processor," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, ser. SC '08. Piscataway, NJ, USA: IEEE Press, 2008, pp. 38:1–38:11. [Online]. Available: http://portal.acm.org/citation.cfm?id=1413370.1413409

[6] J. Howard, S. Dighe, Y. Hoskote, S. Vangal, D. Finan, G. Ruhl, D. Jenkins, H. Wilson, N. Borkar, G. Schrom, F. Pailet, S. Jain, T. Jacob, S. Yada, S. Marella, P. Salihundam, V. Erraguntla, M. Konow, M. Riepen, G. Droege, J. Lindemann, M. Gries, T. Apel, K. Henriss, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, R. V. D. Wijngaart, and T. Mattson, "A 48-core ia-32 message-passing processor with dvfs in 45nm cmos," in *Proceedings of the International Solid-State Circuits Conference*, 2010.

[7] S. Boyd-Wickizer, H. Chen, R. Chen, Y. Mao, F. Kaashoek, R. Morris, A. Pesterev, L. Stein, M. Wu, Y. Dai, Y. Zhang, and Z. Zhang, "Corey: an operating system for many cores," in *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, ser. OSDI'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 43–57. [Online]. Available: http://portal.acm.org/citation.cfm?id=1855741.1855745

[8] P. of the 12th Workshop on Hot Topics in Operating Systems, Ed., *Your computer is already a distributed system. Why isn't your OS?*, 2009.

[9] D. Wentzlaff and A. Agarwal, "Factored operating systems (fos): the case for a scalable operating system for multicores," *SIGOPS Oper. Syst. Rev.*, vol. 43, pp. 76–85, April 2009. [Online]. Available: http://doi.acm.org/10.1145/1531793.1531805

[10] R. Rotta, T. Prescher, J. Traue, and J. örg Nolte, "In-memory communication mechanisms for many-cores experiences with the intel scc," TACC-Intel Highly Parallel Computing Symposium, http://www.tacc.utexas.edu/ti-hpcs12/program, April 2012.