DESIGN CHALLENGES IN MULTIPROCESSOR SYSTEMS-ON-CHIP

Wayne Wolf
Department of Electrical Engineering, Princeton University

Abstract: A multiprocessor system-on-chip is an integrated system that performs real-time tasks at low power and for low cost. The stringent requirements on multiprocessor systems-on-chips force us to use advanced design methods to create these systems. Both hardware and software design must be taken into account. In this paper, we will survey some important challenges in the design of these systems.

1. INTRODUCTION

A multiprocessor system-on-chip (MPSoC) is an integrated system designed with multiple processing elements. MPSoCs are already in common use; over the next several years very large MPSoCs will come onto the market.

MPSoCs are interesting examples of complex embedded systems. A variety of techniques must be used to successfully create MPSoCs and their embedded applications. These techniques must take into account real-time performance, power/energy consumption, and cost.

In this paper, we will survey some design challenges in MPSoCs. We will start by reviewing the requirements on these systems. We will then look at several aspects of these systems: processors, multiprocessor architecture, programs, and task-level scheduling.

2. REQUIREMENTS AND APPROACHES

MPSoCs must provide high levels of performance for applications like video compression and high-speed data communication. But that performance must meet real-time, not just average performance requirements. Real-time computing constraints are deadlines that must be met. Average high performance is not satisfactory if one of the system's tasks does not meet its deadlines.

Many multiprocessor SoCs also operate under tight power and energy requirements. Most embedded applications are somewhat power and heat sensitive. Battery-powered systems must make the best use of available battery energy.

Embedded systems are generally cost-sensitive. The cost of the hardware platform, including processors and memory, must be kept down while meeting the real-time performance requirements.

One traditional approach to meeting these requirements is to make use of knowledge of the application. General-purpose computers are designed around benchmarks but not highly tuned to a particular application. If we know details of the application to be run, we can customize the MPSoC to provide features where needed and eliminate them where they are not necessary.

An implication of this approach is that many MPSoCs are heterogeneous, with multiple types of CPUs, irregular memory hierarchies, and irregular communication. Heterogeneity allows architects to support necessary operations while eliminating the costs of unnecessary features. Heterogeneity is also important to real-time performance---limiting access to part of the system can make that part more predictable.

3. PROCESSOR SELECTION

The choice of processor is one of the basic decisions in the design of an embedded system. The decision may be made on several grounds, both technical and non-technical: software performance, I/O system configuration, support tools, setup costs, etc. The designer may also choose between an existing processor and a customized CPU.

Customized processor architectures can be implemented on systems-on-chips or using FPGAs. When choosing a CPU based on software performance, two major alternatives exist: hardware/software partitioning and custom instruction sets. While these techniques have not traditionally been seen as alternatives, both have a similar goal, namely the cost-effective speedup of a program. Hardware/software partitioning works at coarser

granularity while custom instruction sets find speedups at finer levels of granularity.

Hardware/software partitioning builds a custom heterogeneous system with a CPU and a hardwired accelerator, based on program characteristics and performance requirements. A variety of hardware/software partitioning systems have been developed, including COSYMA [7], Vulcan [10], CoWare [25], the system of Eles et al. [Ele97], Lycos [17], and COSYN [5]. These co-synthesis algorithms work on fairly large blocks of program behavior, generally either loop nests or task graphs. They choose units to implement in hardware based in part on the cost of communication between the processor and the accelerator.

The goal of instruction set design is to find operations that can be profitably packaged as instructions. These are generally combinations of more basic instructions or perhaps work on specialized registers, and so show coarser granularity than standard instructions, but finer granularity than accelerators. Instruction sets can be designed by hand, and CPU microarchitectures that are designed to be customized are known as configurable processors. Several companies offer configurable processors. Several university configurable processors also exist, including LISA [11] and PEAS-III [13, 23].

The largest cost in using configurable processors, when compared to accelerators, is the overhead of instruction interpretation. Instruction fetch, decode, and execution on a CPU are more expensive than dedicated logic on an accelerator. However, a configurable processor has two advantages over an accelerator. First, it can be used for many tasks and so may be more heavily utilized. Second, we can eliminate the cost of communication between the accelerator and the processor (so long as we have the registers required to perform the operations). As MPSoCs become larger and the area cost of CPUs becomes less critical, we may see more configurable processors.

4. MULTIPROCESSOR CONFIGURATION

Many embedded systems are multiprocessors, so we need to configure a multiprocessor for use as a platform for the application. Embedded multiprocessors are often heterogeneous, in order to meet real-time performance requirements as well as power and cost requirements.

Hardware/software co-design algorithms have been developed to synthesize multiprocessor architectures, for example systems by Dave and Jha [4] and Wolf [26], but system-on-chip multiprocessors have often been designed by hand. Hand-designed architectures may be satisfactory for

multiprocessors with a few processors, but as we move to systems-on-chips with a dozen or more large processors, we may see MPSoC designers rely more heavily on design space exploration tools.

As in general-purpose computing systems, busses don't scale to large multiprocessors. As a result, MPSoCs are starting to use on-chip networks that route packets between processors and memory. A number of networks-on-chips have been developed, including Nostrum [15], SPIN [9], Slim-spider [16], OCCN [3], QNoC [1], xpipes/NetChip [20,14], and the network of Xu et al. [27].

5. PROGRAM DESIGN

When optimizing embedded programs for the target platform, memory system behavior is a prime target for optimization. Many embedded systems are memory intensive and the program's interaction with memory helps to determine both the system performance and energy consumption.

Code placement was originally developed for general-purpose machines but is also useful in embedded systems. Code placement determines the addresses for sections of code to minimize the cost of cache interactions between instructions. Hwu and Chang [12] extracted information from traces and placed code using a greedy algorithm. McFarling [18] used a combination of trace data and program behavior to determine how to place code.

A variety of methods for optimizing the cache behavior of data have been developed, both by the scientific computing and embedded communities. Panda et al. [21] used a cluster interference graph to optimize the placement of data in memory. Panda et al. [22] developed algorithms for placing data in scratch pads, which are software-controlled memories at the same level of memory hierarchy as level-one caches.

Overall methodologies for memory-intensive systems have also been developed, most notably by Catthoor et al. [2]

6. PROCESS-LEVEL DESIGN

Traditional scheduling algorithms treat jobs as atomic; in some cases, jobs are assumed to arrive dynamically so their characteristics are not known to the scheduler. While embedded systems tasks may use some dynamic tasks, the critical code is often known in advance. Knowledge of the task allows us to combine scheduling algorithms with memory hierarchy analysis, power management, and other aspects of the system

When we build embedded systems on multiprocessor platforms, we often rely on middleware to manage the multiprocessor. Single-processor management is handled by an operating system, while middleware negotiates resource requests across the multiprocessor platform. One approach to building embedded system middleware is to rely on existing standards, such as CORBA [19, 24]. An alternative is to design custom middleware services. Thanks to the tight performance/energy/cost constraints of embedded systems, we should expect to see at least customized versions of middleware standards.

7. SUMMARY

Embedded systems must provide very high levels of performance, but under much more serious power and cost constraints than general-purpose systems. MPSoC designers need to take advantage of the knowledge of computer system design gained over the past several decades, but embedded computing is developing additional techniques to solve its unique problems. Optimizations of both hardware and software are often necessary to achieve the strict requirements of multiprocessor systems-on-chips.

REFERENCES

[1] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "QNoC: QoS architecture and design process for network on chip," *The Journal of Systems Architecture*, 50(2-3), February 2004, pp. 105-128.

[2] Francky Catthoor, Sven Wuytack, Eddy De Greef, Florin Balasa, Lode Nachtergaele, and Arnout Vandecappelle, *Custom Memory Management Methodology: Exploration of Memory Organization for Embedded Multimedia System Design*, Norwell MA: Kluwer Academic Publishers, 1998.

[3] Marcello Coppola, Stephane Curaba, Miltos D. Grammatikakis, Giuseppe Maruccia, and Francesco Papariello, "OCCN" a network-on-chip modeling and simulation framework," in *Proceedings of the Conference on Design Automation and Test in Europe*, vol. 3, IEEE Computer Society Press, 2004, p. 30174.

[4] Bharat P. Dave and Niraj K. Jha, "COHRA: hardware-software cosynthesis of hierarchical heterogeneous distributed embedded systems," *IEEE Transactions on CAD/ICAS*, 17(10), October 1998, pp. 900-919.

[5] Bharat P. Dave, Ganesh Lakshminarayana, and Niraj K. Jha, "COSYN: hardware-software co-synthesis of heterogeneous distributed embedded systems," *IEEE Transactions on VLSI Systems*, 7(1), March 1999, pp. 92-104.

[6] Petru Eles, Zebo Peng, Krzysztof Kuchcinski, and Alexa Doboli, "System level hardware/software partitioning based on simulated annealing and tabu search," *Design Automation for Embedded Systems*, 2, 1996, pp. 5-32.

[7] Rolf Ernst, Joerg Henkel, and Thomas Benner, "Hardware-software cosynthesis for microcontrollers," *IEEE Design and Test of Computers,* 10(4), December 1993, pp. 64–75.

[8] G. Goossens, "Application-specific networks-on-chips," in A. Jerraya and W. Wolf, eds., *Multiprocessor Systems-on-Chips*, Morgan Kaufman, 2004.

[9] Pierre Guerrier and Alain Greiner, "A generic architecture for on-chip packet-switched interconnections," in *Proceedings of the Conference on Design Automation and Test in Europe*, ACM Press, 2000, pp. 250-256.

[10] Rajesh K. Gupta and Giovanni De Micheli, "Hardware-software cosynthesis for digital systems," *IEEE Design and Test of Computers*, 10(3), September 1993, pp. 29–40.

[11] Andreas Hoffman, Tim Kogel, Achim Nohl, Gunnar Braun, Oliver Schliebusch, Oliver Wahlen, Andreas Wieferink, and Heinrich Meyr, "A novel methodology for the design of application-specific instruction-set processors (ASIPs) using a machine description language," *IEEE Transactions on CAD/ICAS*, 20(11), November 2001, pp. 1138-1354.

[12] Wen-Mei W. Hwu and Pohua P. Chang, "Achieving high instruction cache performance with an optimizing compiler," in *Proceedings of the 16th Annual International Symposium on Computer Architecture*, ACM, 1989, pp. 242-251.

[13] Makiko Itoh, Shigeaki Higaki, Jun Sato, Akichika Shiomi, Yoshinori Takuchi, Akira Kitajima, and Masaharu Imai, "PEAS-III: an ASIP design environment," in *International Conference on Computer Design: VLSI in Computers and Processors*, IEEE Computer Society Press, 2000, pp. 430-436.

[14] A. Jalabert, S. Murali, L. Benini, and G. De Micheli, "xpipesCompiler: a tool for instantiating application specific networks-on-chips," in *Proceedings of Design Automation and Testing in Europe Conference*, IEEE, 2004, pp. 884-889.

[15] Sashi Kumar, Axel Jantsch, Juha-Pekka Soininen, Martti Forsell, Mikael Millberg, Jonny Oberg, Kari Tiensyrja, and Ahmed Hemani, "A network on chip architecture and design methodology," in Proceedings

of the IEEE Computer Society Annual Symposium on VLSI, IEEE Computer Society Press, 2002.

[16] Se-Joong Lee, Kangmin Lee, and Hoi-Jun Yoo, "Analysis and implementation of practical, cost-effective networks-on-chips," *IEEE Design & Test of Computers*, 22(5), September/October 2005, pp. 422-433.

[17] J. Madsen, J. Grode, P. V. Knudsen, M. E. Petersen, and A. Haxthausen, "LYCOS: the Lyngby Co-Synthesis System," *Design Automation for Embedded Systems*, 2, 1997, pp. 195-235.

[18] Scott McFarling, "Program optimization for instruction caches," *Proceedings of the Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ACM, 1989, pp. 183-191.

[19] Object Management Group, CORBA Basics, 2006, http://www.omg.org/gettingstarted/corbafaq.htm.

[20] M. D. Osso, G. Biccari, L. Giovanni, D. Bertozzi, and L. Benini, "xpipes: a latency insensitive parameterized network-on-chip architecture for multi-processor SoCs," in *Proceedings of the 21st International Conference on Computer Design*, IEEE Computer Society Press, 2003, pp. 536-539.

[21] Preeti Ranjan Panda, Nikil D. Dutt, and Alexandru Nicolau, "Memory data organization for improved cache performance in embedded processor applications," ACM Transactions on Design Automation of Electronic Systems, 2(4), October 1997, pp. 384-409.

[22] Preeti Ranjan Panda, Nikil D. Dutt, and Alexandru Nicolau, "On-chip vs. off-chip memory: the data partitioning problem in embedded processor-based systems," *ACM Transactions on Design Automation of Embedded Systems*, 5(3), July 2000, pp. 682-704.

[23] Toshiyuki Sasaki, Shinsuke Kobayashi, Tomohide Maeda, Makkiko Itoh, Yoshinori Takeushi, and Masahiru Imai, "Rapid prototyping of complex instructions for embedded processors using PEAS-III," in *Proceedings, SASIMI 2001*, October, 2001, pp. 61-66.

[24] Douglas C. Schmidt and Fred Kuhns, "An overview of the real-time CORBA specification," *IEEE Computer*, 33(6), June 2000, pp. 56-63.

[25] S. Vercauteren, B. Lin, and H. De Man, "A strategy for real-time kernel support in application-specific HW/SW embedded architectures," in *Proceedings, 33rd Design Automation Conference*, ACM Press, 1996, pp. 678-683.

[26] Wayne Wolf, "An architectural co-synthesis algorithm for distributed embedded computing systems," *IEEE Transactions on VLSI Systems*, 5(2), June 1997, pp. 218-29.

[27] Jiang Xu, Wayne Wolf, Joerg Henkel, and Srimat Chakradhar, "A design methodology for application-specific networks-on-chips," *ACM Transactions on Embedded Computing Systems*, to appear in the Special Issue on Multiprocessor Systems-on-Chips.