Software Development Tools for Soft Multiprocessors (Demonstration Proposal)

Steven A. Guccione
Cmpware, Inc.
Austin, TX (USA)
Email: Steven.Guccione@cmpware.com

Introduction

As the density of FPGA devices has increased, it has become possible to configure larger and larger digital circuits covering a broader and broader class of applications. As FPGAs have surpassed the 1 million gate mark they found increasing use in embedded systems with a 'soft' microprocessor core used as the central control. This trend permits various peripherials and interfaces to be all be placed on a single device and programmed using traditional embedded systems programming tools such as compilers and assemblers. This trend is significant in that it not only consumes significant resources inside of the FPGA but is often used in place of other potentially simpler and smaller custom logic designs.

This trend mirrors that of the early use of embedded microprocessors. While they were less efficient and always slower than a custom hardware solution, the ability to program and reprogram these devices eventally trumped all other advantages. This trend was echoed during the 1980s and 1990s in the FPGA world. The large size, high power consumption and high cost of FPGAs were, in the end, no barrier to their acceptance. The ability to program and reprogram the hardware eventually trumped all other concerns.

Because programmabilility appears to be the feature of highest significance in digital hardware, at least over the long term, it is perhaps time to explore the new ways in which silicon area can be sacrified in the service of programmability. One interesting option is to use FPGA resources to configure multiple microprocessors, creating a custom embedded multiprocessor. While an inefficient use of silicon resources by some standards, it presents an architecture that is relatively simple to design, is highly programmable and can achieve relatively high levels of performance given its programmability.

Soft Multiprocessors

Like other repetitive architectural structures, the hardware design of a multiprocessor is not particularly labor-intensive. It is expected that the cores themselves are already designed and verified, and much of the interconnect structure may also be relatively simple and easy to verify. Rather than challenging the hardware designer, such architectures provide more of a challenge to software developers. How the algorithms are partitioned and how they communicate will determine the performance of the system. But getting high performance and high levels of parallelism is often not the first hurdle to overcome. Getting code to function properly at all if often more of a challenge. Only in later refinements is performance typically optimized.

Hardware and software efforts have largely been decoupled historically. This was possible when the performance requirements were heavily dependent on low-level details of the hardware architectures. If embedded multiprocessors are to be made useful, the operation of the application software as well as the operating of the underlying hardware must be taken into account. This will likely require new techniques and almost certainly new approaches to the standard design tools. It should be noted that this process is distinct from so-called Hardware / Software Co-design. These codesign efforts begin by attempting to find the computationally intensive 'kernel' of an application and maps that portion of the software algorithm to some hardware, either fixed or reconfigurable. While this is often a profitable approach, it requires at least one talented engineer with high skill levels in both hardware and software. Additionally, the resulting architecture has usually merely traded flexibility for performance. The new hardware may speed up the selected application, but it likely to be useless for unrelated algorithms.

The appeal of multiprocessors is that the entire design effort remains in the software realm, and the effort tends to be in the parallelizing of the algorithm. It should be noted that this effort is somewhat similar to the search for computationally intensive kernels in Hardware / Software Co-design. In fact, quite often these kernels are themselves the highly parallelizable structures which may also be exploited by a multiprocessor.

The Cmpware CMP-DK

The Cmpware Configurable Multiprocessor Development Kit (CMP-DK) is an Eclipse based multiprocessor simulator used to quickly perform multiprocessor architectural exploration while simultaneously producing a high-performance multiprocessor simulation. This permits actual code from standard compilers to be run on these simulation models. Several features make this design environment particularly useful. These are:

Rapid processor modeling: Models for a processor as complex as a modern RISCcan be completely built in a matter of a few hours, with several processor models, including Altera NIOS, Altera NIOS II, Xilinx MicroBlaze, SPARC-8 (LEON), and MIPS32 included as standard components. Processors may be any hardware which processes input and produces an output, although special support for tradional microprocessors is provided.

Rapid multiprocessor modeling: Processing nodes may be easily connected with standard or custom communication objects. The two basic types of processor communication are serial links and shared memory. These may be configured in a variety of ways to produced synchronized or unsynchronized implementations. Standard modes for FIFOs, shared registers, shared block memories and busses are included. Networks using these components in topologies such as mesh and torus are included. Custom links and network topologies are simple to implement taking on the order of minutes to produce.

Fast simulation: the multiprocessor simulation runs at approximately 2 million cycles per second. This speed permits real interactive software development to be performed on a standard PC.

Powerful IDE: The simulation models for processors, links and networks maintain various types of run-time data which is extracted and displayed by the Cmpware IDE. This data includes standard debugger type interfaces displays for memory, registers, source code, profiling information and instruction trace. Other more multiprocessor specific displays such as processor utilization and power consumption as well as communication channel status are also displayed. This provides one of the most powerful cockpits available for multiprocessor software development.

Several applications have been demonstrated and are availabelecompiled for the default MIPS32 processor architecture. These generally exploit parallelism at the sub-task level and are parameterized to operate on as few as one processor or as many as is limited by the algorithm with no re-compilation. Example applications include an Finite Impulse Response (FIR) filter parallelizable up to the number of taps, a Fast Fourier Transform (FFT) parallelizable up to the number of FFT stages, an Advanced Encryption Standard (AES) algorithm parallelizable to the maximum number of rounds. Because of fast on-chip communications, these algorithms have relatively high speed-up ratios often approaching 1.0 as more processors are added. And futher parallelism may be available in these algorithms. A sub-task level of parallelism of approximately 10 was the goal for these examples.

Conclusions

An emerging trend in hardware design is single-chip multiprocessing. Such an approach holds promise for FPGA devices by providing a highly programmable, high performance approach to design. Such an approach, while following many long term trends toward programmability is not without its challenges. One challenge is developing new techniques and tools to support this style of design. But this approach, in the end, should not be any more complex inherently than hardware design. Hardware designs are currently partitioned spatially, with communication channels defined. And the partitioned pieces are themsleves usually implemented independently. Such an approach is not dramatically different from what is required to program an embedded soft multiprocessor. And such an approach may benefit from the large body of existing software implementations for popular algorithms. Rather than translating these into various hardware components with various design parameters and interfaces, multiprocessing holds out the hope for applications which closely resemble existig code and each other. The Cmpware CMP-DK attempts to simultaneously address several of the challenges facing this style of design. By providing an environment that facilitates multiprocessor modeling and provides a powerful multiprocessor software development environment, many of the first hurdles facing this approach should be more easily overcome.