Programming Configurable Multiprocessors

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

Abstract

A new high performance computation technique involving multiple processors on a single silicon die is quickly gaining popularity. This new design approach provides very high performance, excellent power efficiency and a high level of programmability as compared to other existing solutions. This approach also serves to move the design effort away from hardware design and toward software. This results in a faster time to market as well as a lower up-front design cost. This paper discusses the Configurable Multiprocessor design environment from Cmpware, Inc.. This toolkit is used to design ASIC, FPGA and SoC multiprocessor solutions.

1. Introduction

As the number of available transistors on a die has continued to increase toward the one billion mark, traditional processor architectures have arrived at a critical juncture. Microprocessors and FPGAs have become power-limited and are having difficulties increasing both their device sizes and their clock speeds. The circuits currently used to implement these devices consume nearly as much power as can be conveniently dissipated. Microprocessors have quickly turned toward multicore designs – devices containing two or more microprocessor cores. This change has been so dramatic that it appears that from this point forward, no new high performance microprocessors will be announced or built using a single processor core [10] [11][12].

A similar trend has occurred in the FPGA world. Large modern FPGA devices such as the Xilinx Virtex II Pro already contain several standard microprocessor cores [1]. There are also reports from ASIC processor core vendors that their customers are on average using several cores per device. It is expected that as performance and power constraints continue to dominate design, this trend will continue and accelerate.

This paper discusses a new software development tool from Cmpware, Inc. aimed at programming such multiprocessor architectures. This toolkit provides a fast simulation and development environment for a network-connected arrays of processors. This environment gathers and presents a rich variety of system run time performance and execution data which is essential to successful development in a multiprocessor environment.

2. Configurable Multiprocessing

There are three major multiprocessing approaches used today. The first is the multicore approach used by traditional microprocessor vendors such as Intel, IBM and Sun. Here, multiple processor cores share memory and often cache. Several research efforts in the late 1990 pioneered this "Chip Multiprocessing" approach [2][3][6] This tightly coupled arrangement does not address the memory bandwidth issues in increasing processor performance, but it does provide a task level of parallelism which saves a potentially expensive context switch. This is useful in the presence of high rate interrupts, such as a busy network interface. But it is not clear that there are many such tasks available in conventional systems to exploit this type of parallelism.

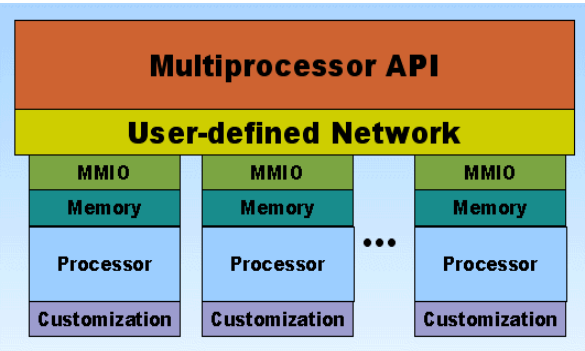

Figure 1: The Cmpware Framework.

The second approach is used by FPGA manufacturers such as Xilinx. Here multiple processor cores are distributed around the FPGA die to be used as the

designer sees fit. Additionally, FPGA designers are increasingly turning to "soft" processor cores such as the Altera NIOS or Xilinx MicroBlaze [14]. Many are increasingly turning to multiple soft processor cores to implement systems. Software support for these hardwired processors, as well as multiprocessing in general, has been largely unavailable.

The last approach is the large-scale use of processor cores in a custom ASIC device [7][8][9][13]. Vendors of processor cores are reporting high levels of multiprocessor development from their customers. Again, these systems have tended to be ad-hoc, and little support is provided to develop such architectures.

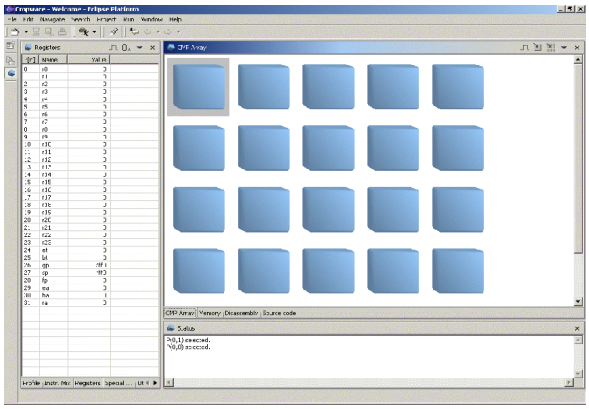

Figure 2: The Cmpware development environment.

In general, none of these approaches have provided much support for the multiprocessor design process. Also, because they tend to be ad-hoc implementations, no particular programming environment addressing the needs of multicore devices has been made available from these vendors. This is in spite of the obvious dedication of both vendors and customers to this approach.

Configurable Multiprocessing (CMP) attempts to put a flexible and useful framework in place that will in turn permit useful tools and supporting intellectual property to be put in place to support these designs. This framework consists of collections of standard processor cores communicating across well-defined communication links.

The Cmpware approach is to treat the processor / compiler as a "black box" capable of implementing algorithms from a high level language. This approach makes the maximum use of existing intellectual property. Existing processor cores and tools are used as well as existing libraries and other software for these processors. This is in contrast to other recent approaches which either define new programming languages or programming language extensions, and / or new processor architectures. Designing and implementing new processors and tools can be an expensive and time-consuming endeavor. The approach taken by Cmpware provides a path to efficiently use existing tools and architectures while also providing the flexibility for integrating new intellectual property where necessary. All of this places the emphasis on designing a solution to the problem at hand, not to designing a support infrastructure to be used to solve the problem at hand.

In the Cmpware approach, the preferred method of communication is a direct, point to point link accessed as a memory mapped IO port. The reasons for this are twofold. Point to point links provide the maximum bandwidth and faster synchronization compared to other approaches. Traditional shared buses and system-level networking such as TCP/IP tend to be large and slow and essentially re-create existing system bottlenecks.

```
void  _start(void) {
   int  node;
   int  ntaps;

   /* Get the parameters */
   node = *west;
   ntaps = *west;

   /* Send to the nextnode */
   *east = (node-1);
   *east = ntaps;

   for (;;) {
      *east = FIR(ntaps, *west);
      *east = shift(ntaps, *west);
      }  /* end for() */

   }  /* end _start() */
```

Figure 3: The multiprocessor FIR code.

Using a memory-mapped IO port as the interface to the communication channel has two very significant benefits. First, it keeps the processor core intact. There is no need to modify either the processor hardware design or its simulation model to use this type of communication link. Second, and perhaps more importantly, it does not require any modifications to the compiler. The memory mapped IO ports appear as an address or "pointer" to be written to or read from. This also provides a very simple and natural interface to the programmer.

Figure 1 gives the general structure of the underlying Cmpware model. Standard components are used for the Processor and the Multiprocessor interfaces. These provide not only generally useful default behavior, but

also all of the machinery necessary to perform multiprocessor simulation. Note that the Processors are uniform objects with their own customization as well as memory and memory mapped IO. These are interfaces to the user-defined interconnection network, which supplies the multiprocessor interface. It is this interface which communicates the the Eclipse Integrated Development Environment (IDE) as well as a simple command line interface.

While these are the suggested approaches for building configurable multiprocessors, the Cmpware toolkit is very flexible in its system modeling offerings. Users may specify custom processor architectures, modify existing architectures and supply custom interconnection models where required.

3. An FIR Filter Example

The Finite Impulse Response (FIR) filter is a very common processing element which is often implemented in both hardware and software. It will make a good candidate for demonstrating some of the features of the configurable multiprocessing approach.

```
int FIR(int  ntaps, int sum) {
   int  i;

   for (i=0; i<ntaps; i++)
      sum += h[i] * z[i];

   return (sum);

   }  /* end FIR() */
```

Figure 4: The FIR code.

In this example, the processing node selected is a NIOS II processor from Altera. The communication network is a 2D grid, with each processor communicating with its four neighbors, although all of these links may not be used in this particular example. The links used by the processors are Shared Registers, which behave like one word synchronous FIFOs. These permit data to be communicated between nodes in a single cycle, while providing the tight synchronization required for high levels of processor utilization.

Figure 2 shows the Cmpware development environment. A 1 x 6 array of NIOS II processors has been allocated and the FIR code loaded into the nodes. Note that this development environment is based on the popular Eclipse IDE [14] and in this case plugs directly into the Altera NIOS II software development environment. In this display, the graphical view of the processor array as well as the IO port status is shown. The other displays include a node source code trace of execution, a memory viewer, a disassembler as well as detailed internal node information such as register values and performance and profiling statistics. These displays are too numerous to detail in this paper, but all are simple to access and interpret and are valuable in debugging and tuning multiprocessor systems.

Figure 3 shows the multiprocessor version of the FIR code. This particular piece of code is parameterized and is run on one or more nodes used to implement the FIR filter. In this case, the number of nodes and the number of taps for the filter is passed in as parameters to configure the filter. In the example displayed in Figure 2, there are six nodes. The first node is a simple data source which sends the input data to the FIR filter. Similarly, the last node is the data sink, which just stores the results for later inspection. So in this particular run, the FIR uses four nodes, each with two taps, for a total of eight taps. The east and west pointer variables in the code represent the IO ports.

Figure 4 shows the code which actually implements the FIR filter. What is perhaps most interesting about this code is that it is a standard "C" function which was taken directly from the literature. There is no particular reference to the parallelism being exploited. All of this is handled at the parameterization level as shown in Figure 3. Another feature of this approach is that the number of taps and the number of processors used is set by two parameters, which can be set dynamically at run time. This permits such processing to dynamically allocate resources at a very fine grain to manage such system parameters such as power consumption and performance.

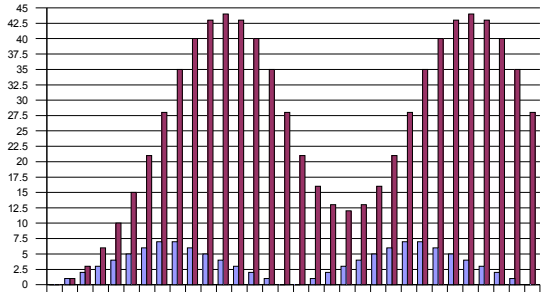

Figure 5: The FIR result.

The graph in Figure 5 shows the input data and the output data, just to verify that a FIR filtering operation was indeed performed. Perhaps more interesting, the graph in Figure 6 shows the speedup as nodes are added. Because the communications and synchronization between nodes is so fast, the speedups as the algorithm is parallelized are fairly dramatic. Note that with no

change in the code, the final 8 processor bar shows a very high degree of parallelization of the algorithm. In this case there is just one filter stage per processor, and there is still an nearly linearly increase in performance.

Unlike other system-level parallel processors, the ability to communicate on-chip permits these efficiencies. Because there is little overhead in communicating, parallelizing of the algorithm can continue at higher levels of efficiency than is possible with other parallel machines. In fact, the fast communication and synchronization make the computation, when it is fully parallelized, resemble hardware-style register transfer language (RTL) design. Unlike hardware design, however, the resources may be redeployed at run time and used in more flexible ways.

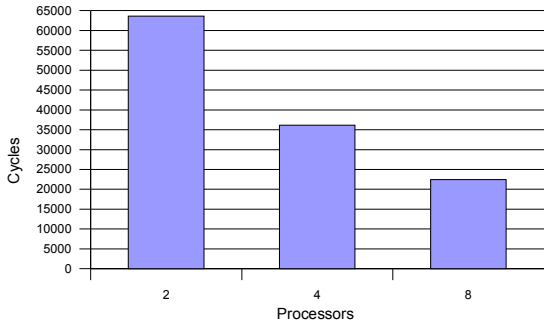

Figure 6: The FIR speedup.

4. Conclusion

Today, it is possible to build and program devices containing thousands of processors. Even reconfigurable logic devices such as FPGAs can easily support hundreds of soft processor cores in a single device. This provides the potential for thousands of MIPS of computing power, programmed in traditional high level languages. To make this approach even more attractive, it operates at a power efficiency that is orders of magnitude higher than the approach used by existing desktop uniprocessors.

Cmpware, Inc. has defined a simple framework for describing a multiprocessor architecture and has provided a fast and flexible programming and simulation environment for such an architecture. This environment supports pluggable processor nodes and a configurable interconnection network. It also provides fast and accurate multiprocessor simulation with a wide variety of run-time data displays. The configurable multiprocessing approach promises to provide fast and flexible high-performance, low-power architectures while continuing to take advantage of increasing circuit densities.

5. References

[1] "Xilinx enhances FPGAs with embedded PowerPCs", http://www.eet.com/semi/news/OEG20020304S0017, March 4, 2002.

[2] Lance Hammond, Basem A. Nayfeh and Kunle Olkotun, A Single-Chip Multiprocessor, Computer, Volume 30, Number 9, September 1997, Pages 79-85.

[3] Lance Hammond, Ben Hubbert , Michael Siu, Manohar Prabhu, Mike Chen , and Kunle Olukotun. The Stanford Hydra CMP, IEEE MICRO Magazine, March-April 2000.

[4] "The Future of Microprocessors", David Patterson, U. California at Berkeley, June 2001. http://www.cs.berkeley.edu/~pattrsn/talks/NAE.ppt

[5] Chip Multiprocessing Resources: http://www.princeton.edu/~jdonald/research/cmp/

[6] Stanford Hydra Project: http://www-hydra.stanford.edu/

[7] John Goodacre, Understanding the Options for Embedded Multiprocessing, ARM IQ Journal, Vol.2, No.2.

[8] "Tensilica clears path to multiprocessor SoCs", http://www.eetimes.com/story/OEG20020826S0024, August 26, 2002.

[9] "Tensilica Introduces Industry's First Integrated Development Environment for Multiple Processor SOC Hardware and Software Design", http://www.tensilica.com/html/pr_2003_06_16a.html, June 16, 2003.

[10] "Intel Demonstrates Breakthrough Processor Design", http://www.intel.com/pressroom/archive/releases/20010828comp.htm, August 28, 2001.

[11] "AMD Announces Technology Milestone With Its Multiple-Core Strategy", http://www.amd.com/us-en/Corporate/VirtualPressRoom/0,,51_104_543~86455,00.html, June 14, 2004.

[12] "Sun Drives Multithreaded Processor Innovation with New UltraSPARC IV+", http://www.sun.com/smi/Press/sunflash/2004-10/sunflash.20041005.2.html, October 5, 2004.

[13] "ARC International's Customers Lead Way in Multiprocessing Design", http://www.us.design-reuse.com/news/news5747.html, June 18, 2003.

[14] "Eclipse Platform Technical Overview", http://www.eclipse.org/whitepapers/eclipse-overview.pdf, February, 2003.

[15] "Literature: NIOS II Processor", http://www.altera.com/literature/lit-nio2.jsp, 2004.