2011 21st International Conference on Field Programmable Logic and Applications

Heracles: Fully Synthesizable Parameterized MIPS-Based Multicore System

Michel A. Kinsy, Michael Pellauer, and Srinivas Devadas
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139
Email: mkinsy, pellauer, devadas@csail.mit.edu

***Abstract*—*Heracles* is an open-source complete multicore system written in Verilog. It is fully parameterized and can be reconfigured and synthesized into different topologies and sizes. Each processing node has a fully bypassed, 7-stage pipelined microprocessor running the MIPS-III ISA, a 4-stage input-buffer, virtual-channel router, and a local variable-size shared memory. Our design is highly modular with clear interfaces between the core, the memory hierarchy, and the on-chip network. In the baseline design, the microprocessor is attached to two caches, one instruction cache and one data cache, which are oblivious to the global memory organization. The memory system in *Heracles* can be configured as one single global shared memory (SM), or distributed shared memory (DSM), or any combination thereof. Each core is connected to the rest of the network of processors by a parameterized, realistic, wormhole router. We show different topology configurations of the system, and their synthesis results on the Xilinx Virtex-5 LX330T FPGA board. We also provide a small MIPS cross-compiler toolchain to assist in developing software for *Heracles*.**

I. Introduction

Multicore architectures have become mainstream computing platforms. These systems typically consist of processing elements (PEs or cores), a memory subsystem, and an infrastructure for inter-core communications. Traditionally, buses have been used in establishing communications between cores, but because of the increasing complexity of these designs and the lack of scalability of wired connections between cores, network-on-chip (NoC) architectures have been introduced as an effective data communication infrastructure [7], [11]. It has been shown that the overall performance of multicore systems is often defined by their communication limits in terms of bandwidth, speed and concurrency [4], and not by the individual computation power of the cores. Therefore, simple reduced instruction set computer (RISC) cores are often used in these architectures.

In this paper, we present a new open-source FPGA-based system for designing multicore architectures. *Heracles* is a complete multicore system written in Verilog, fully parameterized, that can be reconfigured into different topologies and sizes. The main contribution of our work is the fact that *Heracles* is designed with a high degree of modularity to support exploration of future multicore processors of different topologies, routing schemes, processing elements or cores, and memory system organizations. Figure 1 shows the top level view of *Heracles* multicore system arranged in 2D-mesh topology, and Figure 2 shows two different views of the network switch local to a node.

There has been a large body of work on implementing multicore architectures on FPGAs. In contrast, there seems to be very little on complete, modular, multicore systems, with reconfigurable network topology, where processing core, memory system, and on-chip network are fully self-contained. *Heracles* presents designers with a global and complete view of the inner workings of a multiprocessor machine cycle-by-cycle from instruction fetches at the microprocessor core at each node to the *flit* arbitration at the routers, with RTL level correctness. A flit is the smallest unit of information recognized by the flow control method [8]. This enables the designer to explore different implementation approaches: core micro-architecture, levels of caches, cache sizes, routing algorithm, router micro-architecture, distributed or shared memory, or network interface, and to quickly evaluate their impact on the overall system performance.

Section II describes an integer-based 7-stage MIPS processing element (PE), and its usage in forming a node in the network. Section III presents our structure for supporting an arbitrary memory organization, and details regarding the network interface. Section IV deals with the router micro-architecture and support for various routing algorithms. Section V shows different *Heracles* topologies and their FPGA utilization. Section VI describes the current software toolchain, and Section VII presents our performance analysis results. Related work is summarized in Section VIII. Section IX concludes the paper.

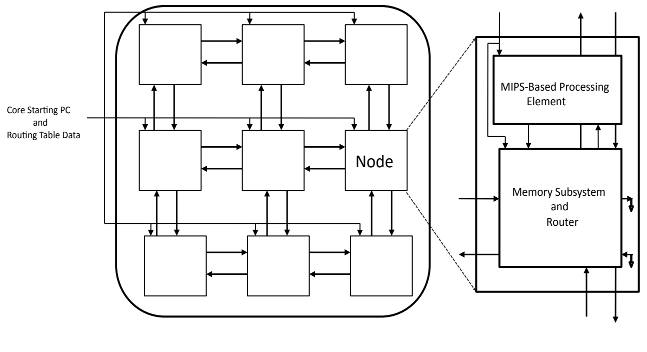

Fig. 1. 2D-Mesh Topology Heracles Architecture.

II. Processing Element Module

The processing element in *Heracles* consists of an integer-based 7-stage MIPS Core. MIPS (Microprocessor without Interlocked Pipeline Stages) is a register based RISC architecture widely used in commercial products and for teaching purposes [19]. Our implementation is a standard Verilog implementation of the micro-architecture described by Patterson and Hennessy [19], with some modifications for FPGAs. For example, the adoption of a 7-stage pipeline, due to block RAM access time on the FPGA.

Figure 3 shows the core architecture, a 7-stage pipeline architecture, fully bypassed, with no branch predictor or branch delay slot, running MIPS-III instruction set architecture (ISA) without floating point. Instruction and data caches are implemented using block RAMs, and instruction fetch and data memory access take two cycles. Instruction address is issued at *I-Fetch 1* stage and on a cache hit, the actual instruction appears in the *I-Fetch 2* stage. Instruction decode and register read stage and the execution stage remain functionally the same as described in [19]. Stall and bypass signals are modified to support the extended pipeline. Data memory (cache) is implemented over *D-Memory 1* and *D-Memory 2* stages. For a read, the memory address is presented to the cache in the *D-Memory 1* stage and the data on a cache hit appears in the *D-Memory 2* stage. On a memory write, we also check in the *D-Memory 2* stage that there is a cache hit before continuing execution. Instructions are issued and executed in-order, and the data memory accesses are also in-order.

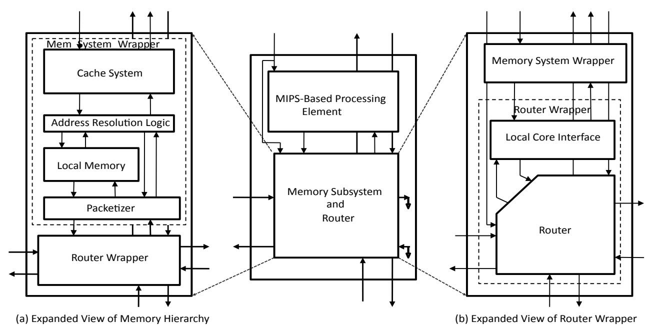

Fig. 2. Network Switch Expanded Views.

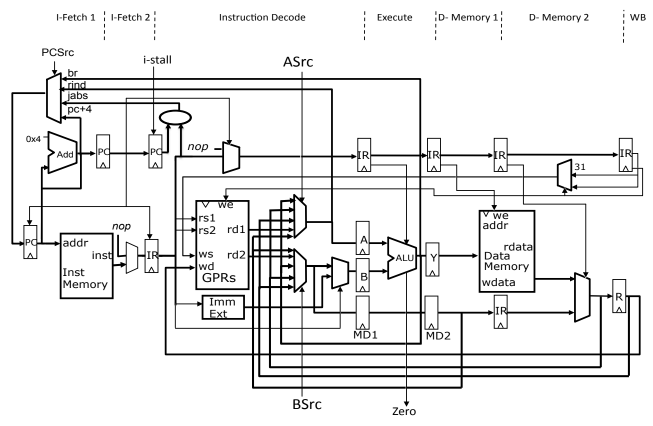

Fig. 3. Integer-based 7-stage MIPS processing element.

	Used	Available	Utilization
Registers	1,635	207,360	under 1%
Lookup Tables	2,529	207,360	1%

Critical Path (ns)	6.151
Clock Rate (MHz)	162.564

TABLE I
Processing Element Synthesis Results

The core is synthesized using Xilinx ISE Design Suite 11.5, with Virtex-5 LX330T as the targeted board. Table I gives a brief summary of the synthesis results and the clocking speed of the design. As shown in Table I, our 7-stage pipeline core architecture runs at 162.5 MHz, and has an FPGA resource utilization of around 1% on the Virtex-5 LX330T. Due to the modular design of *Heracles*, any core with the same memory interface can be plugged into the system.

III. Memory System Organization

The memory system in *Heracles* is completely parameterized, and can be set up in various ways, independent of the rest of the system. The key components are the cache system, the local memory, and the network interface.

A. Cache System

In *Heracles*, we implement a 1-level cache system composed of a direct-mapped instruction cache and a direct-mapped data cache, which can be extended to more cache levels. Each cache can be independently configured. The *INDEX_BITS* parameter controls the number of blocks or cache-line in the cache. The *OFFSET_BITS* parameter determines the cache block size. The direct-mapped cache is implemented using block RAM, where on a hit, the data appears on the output port in the following cycle. Since block RAMs on the FPGA are constrained resources, we also implement a direct-mapped cache using registers and lookup tables, but at a high FPGA resource cost. The cache system, like the core, is oblivious to the system-level memory organization and network topology. This decoupling is achieved through the *Address Resolution Logic*, which sits outside the cache system and interacts with the rest of the memory structure.

B. Local Memory Distribution

The memory system in *Heracles* is constructed to allow different memory space configurations. The local memory is parameterized and has two very important attributes: its size can be changed on a per core-basis, and it can service a variable number of caches in a round-robin fashion. For a Shared Memory (SM) implementation, where all processors share a single large memory block, the local memory size is simply set to zero at all nodes except one. At the nodes with no local memory, the *Address Resolution Logic* directly sends all cache system traffic into the network, where it transits to the target node. In Distributed Shared Memory (DSM), where each processing element has its own private memory, local memory size can be set to be the same across all the nodes or set to different values. For example, in a mesh network, our experiments show that for a large class of routing algorithms locating larger blocks of memory at the center nodes, can improve network congestion. The *LOCAL_ADDR_BITS* parameter is used to set the size of the local memory.

The fact that the local memory is parameterized to handle requests from a variable number of caches allows us to present to the local memory the traffic coming into the node from other cores through the network, as just another cache communication. This illusion is created through the network packetizer. Local memory can also be viewed as a memory controller. *Heracles*, at the moment, provides no cache coherence protocol. However, the design of the system is such that a cache coherence scheme can be supported.

C. Network Interface

The *A*ddress Resolution Logic works with the *Packetizer* module to get the caches and the local memory to interact with the rest of the system. All cache traffic goes through the *Address Resolution Logic*, which determines if a request can be served at the local memory, or if the request needs to be sent over the network. In *Heracles*, an address contains two fields, where the lower order bits represent the real address, and the the higher order bits identify the home core for that particular address. These two fields are automatically identified based on the *LOCAL_ADDR_BITS* and the *ADDRESS_BITS* parameters. If the home core of an address is not the core that generated the address, the *Address Resolution Logic* forwards the request to the network, through the *Packetizer*.

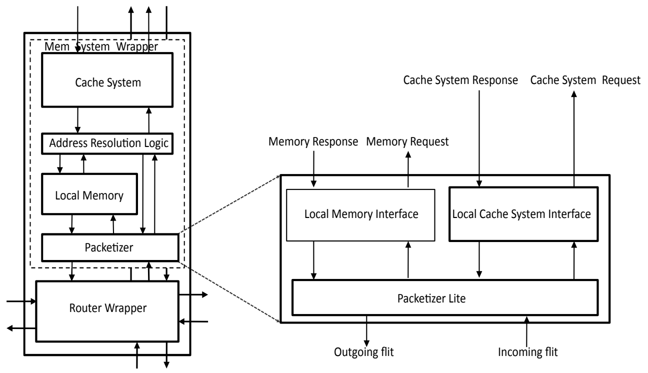

Fig. 4. Network Interface Packetizer.

Inside the *Packetizer* module, there are three submodules as shown in Figure 4. The *Local Memory Interface* uses a cache-like protocol to interact with the local memory. In our baseline design, the *Local Memory Interface* simply acts as a third cache on the local memory side. The *Local Cache System Interface* uses a memory-like protocol to interact with the cache system like a second larger memory block. The *Packetizer Lite* is responsible for converting data traffic, such as a load, coming from the local memory and the cache system into packets or flits that can be routed inside the Network-on-chip (NoC), and for reconstructing packets or flits into data traffic at the opposite side when exiting the NoC. The *Packetizer Lite* directly connects to the network router. Table II gives a brief summary of the synthesis results and the clocking speed of our 7-stage pipeline MIPS core with 2 caches (I-Cache and D-Cache) of 2KB each, and 262KB of local memory.

	Used	Available	Utilization
Registers	2,695	207,360	under 1%
Lookup Tables	5,562	207,360	2%
Block RAM/FIFO	75	324	23%

Critical Path (ns)	6.471
Clock Rate (MHz)	155.825

TABLE II
PROCESSING ELEMENT WITH CACHES AND MEMORY SYNTHESIS RESULTS

IV. ROUTER ARCHITECTURE

To provide scalability, *Heracles* uses a network-on-chip (NoC) architecture for its data communication infrastructure. An NoC architecture is defined by its topology (the physical organization of nodes in the network), its flow control mechanism (which establishes the data formatting, the switching protocol and the buffer allocation), and its routing algorithm (which determines the path selected by a packet to reach its destination under a given application). This section discusses the router micro-architecture, and its support for different network topologies and routing algorithms.

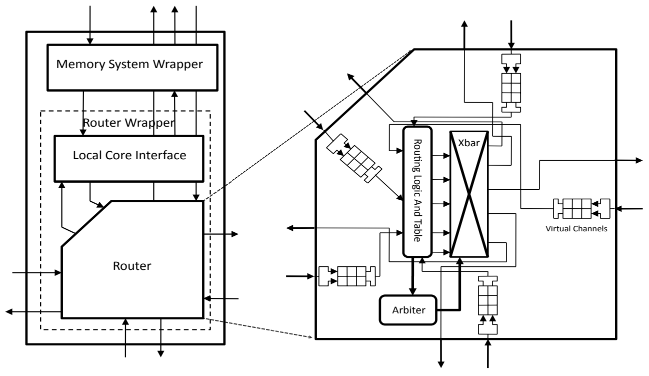

Fig. 5. Router Micro-Architecture.

A. Router Micro-Architecture

Figure 5 illustrates the virtual-channel router used in *Heracles*. The router fairly conforms, in its architecture and operation, to conventional virtual-channel routers [8], [17]. It has some input buffers to store flits while they are waiting to be routed to the next hop in the network. The routing operation takes four steps or phases, namely routing (RC), virtual-channel allocation (VA), switch allocation (SA), and switch traversal (ST), where each phase corresponds to a pipeline stage in our router. When a head flit (the first flit of a packet) arrives at an input channel, the router stores the flit in the buffer for the allocated virtual channel and determines the next hop for the packet (RC phase). Given the next hop, the router then allocates a virtual channel in the next hop (VA phase). Finally, the flit competes for a switch (SA phase), if the next hop can accept the flit, and moves to the output port (ST phase).

The switch allocation (SA) stage is the critical stage in our router design, due to the complexity of the arbiter. During the SA stage, the arbiter grants switch traversal to all input ports requesting output ports for which they have priority. If an input port is requesting an output port, and the priority holder on that outgoing port is either idle or requesting a different output port, it has to compete with all other input ports requesting the same output port. The arbiter is also responsible for adjusting priorities to promote fairness and avoid starvation. The synthesis of the router shows a delay of 14.016 nanoseconds, with 24 levels of logic on the critical path.

B. Route Computation and Virtual Channel Allocation

Algorithms used to compute routes in network-on-chip (NoC) architectures, generally fall under two categories: *oblivious* and *dynamic* [18]. The router implemented in *Heracles* primarily supports *oblivious* routing algorithms using either fixed logic or routing table. Fixed logic is provided for dimension order routing (DOR) algorithms [21], which are popular and have many desirable properties. For example, they generate deadlock-free routes in mesh or hypercube topologies [6]. Either using *XY-Ordered Routing* or *YX-Ordered Routing*, each packet is routed along one dimension in its first phase followed by the other dimension. On the other hand, table-based routing provides greater programmability and flexibility, since routes can be pre-computed and stored in the routing tables before execution. Table-based routing supports both *minimal* and *non-minimal* routing algorithms. In this routing scheme, at the beginning of the program, routing tables are updated; and during execution each packet has a flow ID, which is used to address the routing table to determine the packet's outgoing port. The *RT_ALG* parameter is used to select the proper routing algorithm for a given application and topology. *Heracles* provides support for both static and dynamic virtual channel allocation. When static allocation is used, the routing table stores the outgoing port of the packet along with the virtual channel to be used in the next node. There is no additional hardware cost for supporting static virtual channel allocation, since the entry into the table is also used during dynamic allocation. The number of virtual channels per port and their sizes are variable parameters (*VC_PER_PORT* and *VC_DEPTH*).

C. Network Topology Configuration

The parameterization of the number of input ports and output ports on the router and the table-based routing capability give *Heracles* a great amount of flexibility and the ability to metamorphose into different network topologies; for example, *k*-ary *n*-cube, 2D-mesh, 3D-mesh, hypercube, ring, or tree. A new topology is constructed by changing the *IN_PORTS*, *OUT_PORTS*, and *SWITCH_TO_SWITCH* parameters and reconnecting the routers. In the case of the 3D-mesh, the *IN_PORTS* and *OUT_PORTS* parameters are set to 2 or 3, one to connect the router to the local core and the remainder to connect the router to the third dimension. Table III gives a brief summary of the synthesis results and the clock frequency of our virtual-channel router in 2D-meshes. It runs at 71 MHz, with the limiting factor being logic complexity of the arbiter.

	Used	Available	Utilization
Registers	2,806	207,360	1%
Lookup Tables	2,058	207,360	1%

Critical Path (ns)	14.016
Clock Rate (MHz)	71.345

TABLE III
VIRTUAL-CHANNEL ROUTER SYNTHESIS RESULTS

For a *fat-tree* [15] topology, routers at different levels of the tree have different sizes, in terms of crossbar and arbitration logic. The root node contains the largest router, and controls the clock frequency of the system. Figure 6 shows an unbalanced *fat-tree* topology, and Table IV shows the summary of the synthesis results and the clock frequency of the *fat-tree* multicore system.

	Used	Available	Utilization
Registers	166,726	207,360	80%
Lookup Tables	165284	207,360	80%

Critical Path (ns)	33.246
Clock Rate (MHz)	30.079

TABLE IV
UNBALANCED FAT-TREE TOPOLOGY SYNTHESIS RESULTS

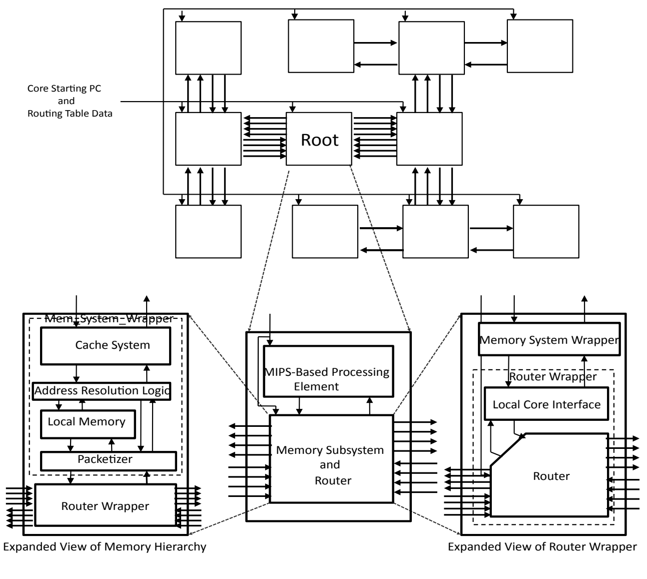
Core Starting PC and Routing Table Data
Root
Mem. System Wrapper
Cache System
Address Resolution Logic
Local Memory
Packetizer
Router Wrapper
MIPS-Based Processing Element
Memory Subsystem and Router
Memory System Wrapper
Router Wrapper
Local Core Interface
Router
Expanded View of Memory Hierarchy
Expanded View of Router Wrapper

Fig. 6. Unbalanced Fat-Tree Topology with Expanded Views of the Root.

V. FPGA 2D-MESH TOPOLOGY SYSTEMS

A large number of cores can be implemented on a modern FPGA. Moreover, having a simple RISC core, MIPS in our case, for the processing element (PE) allows for a good size multicore system. This section presents three different sizes, 2×2, 3×3, and 4×4, of the complete *Heracles* multicore architecture arranged in 2D-mesh topology. Figure 1 shows the 3×3 mesh topology. Table V summarizes the key architectural characteristics of the multicore system. The system is running at 71 MHz, which is the clock frequency of the router, regardless of the size of the mesh. The system speed will increase if a less complex arbitration scheme is adopted.

	Heracles
Core	
ISA	32-Bit MIPS
Multiply/Divide	Software
Floating Point	Software
Pipeline Stages	7
Bypassing	Full
Branch policy	Always non-Taken
Outstanding memory requests	1
Address Translation	None
Level 1 Instruction/Data Caches	
Associativity	Direct
Size	16KB
Outstanding Misses	1
On-Chip Network	
Topology	2D-Mesh
Routing Policy	DOR and Table-based
Virtual Channels	2
Buffers per channel	8

TABLE V
2D-MESH *Heracles* MULTICORE ARCHITECTURE

Resource Utilization

	Registers	Lookup Tables	Block RAM/FIFO
2×2 Mesh	35.5%	39.6%	92.1%
3×3 Mesh	58.1%	60.0%	99.6%
4×4 Mesh	99.9%	94.7%	83.5% *

Total Available Resource

Registers	Lookup Tables	Block RAM/FIFO
207,360	207,360	324

TABLE VI
VIRTEX-5 LX330T FPGA RESOURCE UTILIZATION PER MESH SIZE.

Using a 2-D mesh topology, we are able to fit up to 16 cores on the Virtex-5 LX330T FPGA board. Table VI summarizes the FPGA resource utilization for different network sizes in terms of registers, lookup tables, and RAMs. In the 2×2 configuration, the local shared memory is set to $260KB$ per core. Whereas for the 3×3 configuration the size of the local shared memory is reduced to $64KB$ per core, due to limited FPGA block RAM. The local memory in the 4×4 configuration is set to $32KB$, which lowers the percentage of used block RAM/FIFO down to 83.5%.

VI. SOFTWARE PROGRAMMING TOOLCHAIN

We are releasing *Heracles* with a small open-source software toolchain to assist in developing software for the system, although users can easily build their own. The toolchain for this release supports a single-Thread C programming model. It is built around the GCC MIPS cross-compiler using GNU C version 3.2.1. Figure 7 depicts the software flow for compiling a C program into the compatible MIPS instruction code that can be executed on the system. The compilation process consists of a series of six steps. First, the user invokes *mips-gcc* to translate the C code into assembly language (e.g., *./mips-gcc -S fibonacci.c*). In step 2, the assembly code is then run through the isa-checker (e.g., *./checker fibonacci.s*). The checker's role is to: (1) remove all memory space primitives, (2) replace all pseudo-instructions, and (3) check for floating point instructions. Its output is a *.asm* file. For this release, there is no direct high-level operating system support. Therefore, in the third compilation stage, a small kernel-like assembly code is added to the application assembly code for memory space management and workload distribution (e.g., *./linker fibonacci.asm*). Users can modify the *linker.cpp* file provided in the toolchain to reconfigure the memory space and workload. In step 4, the user compiles the assembly file into an object file using the cross-compiler. This is accomplished by executing *mips-as* on the *.asm* file (e.g., *./mips-as fibonacci.asm*). In the next step, the object file is disassembled using the *mips-objdump* command (e.g., *./mips-objdump fibonacci.o*). Its output is a *.dump* file. Finally, the constructor script is called to transform the dump file into Verilog memory, *.vmh*, file format (e.g., *./dump2vmh fibonacci.dump*). The software toolchain is still evolving.

VII. EXPERIMENTAL RESULTS

We examine the performance of two SPEC CINT2000 benchmarks, namely, 197.parser and 256.bzip2 on *Heracles*. We modify and parallelize these benchmarks to fit into our evaluation framework. For the 197.parser benchmark, we identify three functional units: file reading and parameters setting as one unit, actual parsing as a second unit, and error reporting as the third unit. When there are more than three cores, all additional cores are used in the parsing unit. Similarly, 256.bzip2 is divided into three functional units: file reading and cyclic redundancy check, compression, and output file writing. The compression unit exhibits a high degree of data-parallelism, therefore we apply all additional cores to this unit for core count greater than three. We also present a brief analysis of a simple *Fibonacci* number calculation program. Figures 8 and 10 show 197.parser and 256.bzip2 benchmarks under single shared-memory (SSM) and distributed shared-memory (DSM), using *XY-Ordered* routing. Increasing the number of cores improves performance for both benchmarks; it also exposes the memory bottleneck encountered in the single shared-memory scheme. Figures 9 and 11 highlight the impact of the routing algorithm on the overall system performance, by comparing completion cycles of *XY-Ordered* routing and BSOR [13]. BSOR, which stands for Bandwidth-Sensitive Oblivious Routing, is a table-based routing algorithm that minimizes the maximum channel load (MCL) or maximum traffic across all network links in an effort to maximize application throughput. The routing algorithm has little or no effect on the performance of 197.parser and 256.bzip2 benchmarks, as shown in Figure 9 for 197.parser, because of the traffic patterns in these applications. For the *Fibonacci* application, Figure 11, BSOR routing does improve performance, particularly with 5 or more cores.

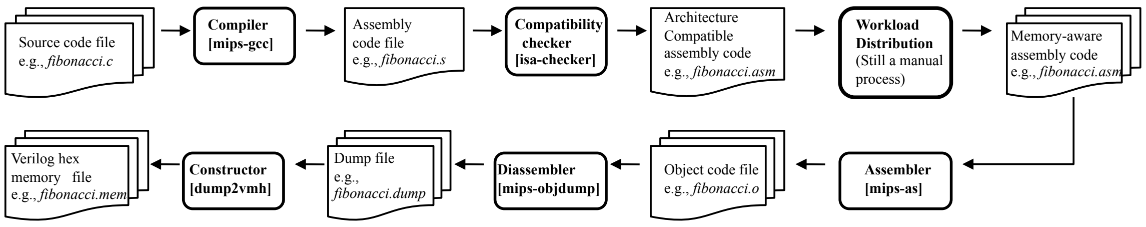
Source code file e.g., *fibonacci.c*
Compiler [mips-gcc]
Assembly code file e.g., *fibonacci.s*
Compatibility checker [isa-checker]
Architecture Compatible assembly code e.g., *fibonacci.asm*
Workload Distribution (Still a manual process)
Memory-aware assembly code e.g., *fibonacci.asm*
Assembler [mips-as]
Object code file e.g., *fibonacci.o*
Diassembler [mips-objdump]
Dump file e.g., *fibonacci.dump*
Constructor [dump2vmh]
Verilog hex memory file e.g., *fibonacci.mem*

Fig. 7. Software Toolchain Flow.

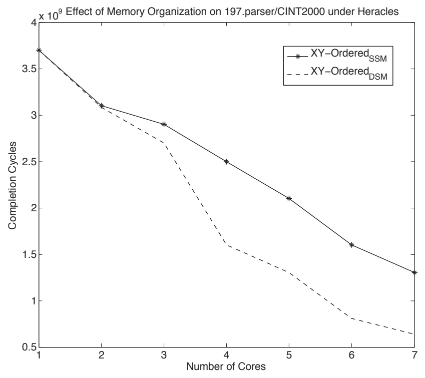

Fig. 8. 197.parser: Effect of Memory Organization on Performance

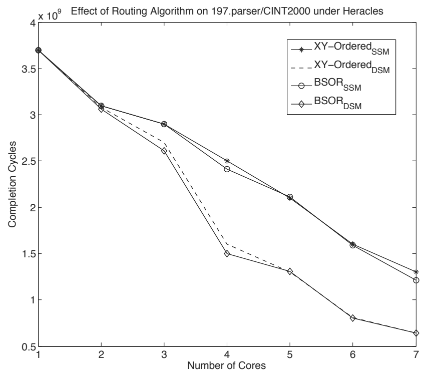

Fig. 10. 197.parser: Effect of Routing Algorithm on Performance in 2D-Mesh

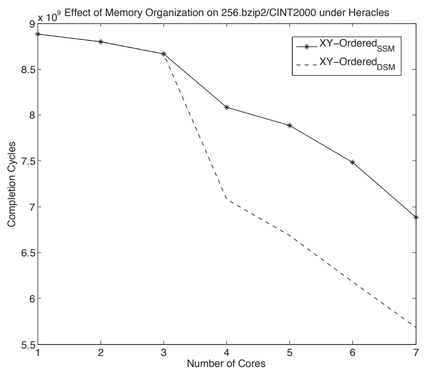

Fig. 9. 256.bzip2: Effect of Memory Organization on Performance

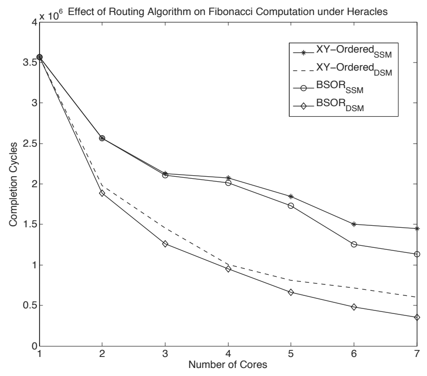

Fig. 11. Fibonacci: Effect of Routing Algorithm on Performance

Heracles Verilog files and software toolchain for building MIPS code to run on the system can be found at: `http://web.mit.edu/mkinsy/Public/Heracles`

VIII. Related Work

Implementation of multicore architecture on FPGAs has been the subject of several research projects. In [9] Del Valle *et al* present an FPGA-based emulation framework for multiprocessor system-on-chip (MPSoC) architectures. LEON3 [1], a synthesizable VHDL model of a 32-bit processor compliant with the SPARC V8 architecture, has been used in implementing multiprocessor systems on FPGAs. Andersson *et al* [2], for example, use the LEON4FT microprocessor to build their Next Generation Multipurpose Microprocessor (NGMP) architecture, which is prototyped on the Xilinx XC5VFX130T FPGA board. However, the LEON architecture is fairly complex, and it is difficult to instantiate more than two or three on a medium size FPGA. Clack *et al* [5] investigate the use of FPGAs as a prototyping platform for developing multicore system applications. They use Xilinx MicroBlaze processor for the core, and a bus protocol for the inter-core communication. James-Roxby *et al* [12] shows similar FPGA design in their proposed architecture for supporting a single program multiple data model of parallel processing.

Other FPGA-based multicore architectures are more application specific. Ravindran *et al* [20] demonstrate the feasibility of FPGA-based multicore systems for high performance applications, through the implementation of IPv4 packet forwarding using Xilinx Virtex-II Pro FPGA. Wang *et al* [24] propose a multicore architecture on FPGAs for large dictionary string matching. Similarly, Tumeo *et al* [23] present FPGA-based multicore shared memory for dual priority scheduling algorithm for real-time embedded systems. Some designs focus primarily on the Network-on-chip (NoC). Lusala *et al* [16], for example, propose a scalable implementation of NoC on FPGA using a torus topology. Genko *et al* [10] also present an FPGA-based flexible emulation environment for exploring different NoC features. A VHDL-based cycle accurate RTL model for evaluating power and performance of NoC architectures is presented in Banerjee *et al* [3]. Other designs make use of multiple FPGAs. H-Scale [22], by Saint-Jean *et al*, is a multi-FPGA based homogeneous SoC, with RISC processors and an asynchronous NoC. The S-Scale version supports a multi-threaded sequential programming model with dedicated communication primitives handled at run-time by a simple operating system. The RAMP Blue project [14] has developed a set of reusable design blocks to emulate multicore architectures on FPGAs. The system consists of 768-1008 MicroBlaze cores in 64-84 Virtex-II Pro 70 FPGAs on 16-21 BEE2 boards.

IX. Conclusion

We have presented a complete, realistic, fully parameterized, synthesizable, modular, multicore architecture. The system, called *Heracles*, uses a component-based design approach, where the processing element or core, the router and the network-on-chip, and the memory subsystem are independent building blocks, and can be used in other designs. The baseline system has a 7-stage integer-based MIPS core, a virtual-channel wormhole router, with support for both shared memory and distributed shared memory, and can be implemented on the Xilinx Virtex-5 LX330T FPGA board. We have introduce a small software toolchain for compiling C programs onto the system.

We have shown a 2D-Mesh topology and an unbalanced *fat-tree* topology implementation of *Heracles*, to demonstrate the flexibility and the robustness of the system. *Heracles* can serve as a simulator in testing routing algorithms, flow control schemes, network topologies, or memory controller organizations, or it can be used as an accelerator when simulating a network-on-chip (NoC) by removing the MIPS cores from the design and placing only the NoC on the FPGA.

Future work will involve adding a small kernel binary code to each core on start up for handling exceptions and proper interrupts for peripheral communications. Multi-threading and dynamic runtime workload management among the cores, via thread migration, will be explored.

Acknowledgment

We thank Joel Emer, Li-Shiuan Peh, Omer Kan, Myong Hyon Cho, and Noah Keegan for interesting discussions throughout the course of this work.

References

[1] A. G. AB. Leon3 processor. *Available at*: http://www.gaisler.com.

[2] J. Andersson, J. Gaisler, and R. Weigand. Next generation multipurpose microprocessor. *Available at*: http://microelectronics.esa.int/ngmp/NGMP-DASIA10-Paper.pdf, 2010.

[3] N. Banerjee, P. Vellanki, and K. Chatha. A power and performance model for network-on-chip architectures. volume 2, pages 1250 – 1255 Vol.2, feb. 2004.

[4] L. Benini and G. De Micheli. Networks on chips: a new soc paradigm. *Computer*, 35(1):70–78, Jan 2002.

[5] C. R. Clack, R. Nathuji, and H.-H. S. Lee. Using an fpga as a prototyping platform for multi-core processor applications. In *WARFP-2005: Workshop on Architecture Research using FPGA Platforms*, Cambridge, MA, USA, feb. 2005.

[6] W. J. Dally and C. L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Trans. Computers*, 36(5):547–553, 1987.

[7] W. J. Dally and B. Towles. Route Packets, Not Wires: On-Chip Interconnection Networks. In *Proc. of the 38th Design Automation Conference (DAC)*, June 2001.

[8] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2003.

[9] P. Del valle, D. Atienza, I. Magan, J. Flores, E. Perez, J. Mendias, L. Benini, and G. Micheli. A complete multi-processor system-on-chip fpga-based emulation framework. pages 140 –145, oct. 2006.

[10] N. Genko, D. Atienza, G. D. Micheli, J. M. Mendias, R. Hermida, and F. Catthoor. A complete network-on-chip emulation framework. In *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, pages 246–251, Washington, DC, USA, 2005.

[11] A. Ivanov and G. D. Micheli. The Network-on-Chip Paradigm in Practice and Research. *Design & Test of Computers*, 22(5):399–403, 2005.

[12] P. James-Roxby, P. Schumacher, and C. Ross. A single program multiple data parallel processing platform for fpgas. In *FCCM '04: Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 302–303, Washington, DC, USA, 2004.

[13] M. Kinsy, M. H. Cho, T. Wen, E. Suh, M. van Dijk, and S. Devadas. Application-Aware Deadlock-Free Oblivious Routing. In *Proceedings of the Int'l Symposium on Computer Architecture*, jun. 2009.

[14] A. Krasnov, A. Schultz, J. Wawrzynek, G. Gibeling, and P.-Y. Droz. Ramp blue: A message-passing manycore system in fpgas. pages 54 –61, aug. 2007.

[15] C. E. Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Trans. Comput.*, 34(10):892–901, 1985.

[16] A. Lusala, P. Manet, B. Rousseau, and J.-D. Legat. Noc implementation in fpga using torus topology. pages 778 –781, aug. 2007.

[17] R. D. Mullins, A. F. West, and S. W. Moore. Low-latency virtual-channel routers for on-chip networks. In *Proc. of the 31st Annual Intl. Symp. on Computer Architecture (ISCA)*, pages 188–197, 2004.

[18] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26(2):62–76, 1993.

[19] D. Patterson and J. Hennessy. *Computer Organization and Design: The Hardware/software Interface*. Morgan Kaufmann, 2005.

[20] K. Ravindran, N. Satish, Y. Jin, and K. Keutzer. An fpga-based soft multiprocessor system for ipv4 packet forwarding. pages 487 – 492, aug. 2005.

[21] H. Sullivan and T. R. Bashkow. A large scale, homogeneous, fully distributed parallel machine, i. *SIGARCH Comput. Archit. News*, 5(7):105–117, 1977.

[22] N. Saint-Jean, G. Sassatelli, P. Benoit, L. Torres, and M. Robert. Hs-scale: a hardware-software scalable mp-soc architecture for embedded systems. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, pages 21–28, Washington, DC, USA, 2007.

[23] A. Tumeo, M. Branca, L. Camerini, M. Ceriani, M. Monchiero, G. Palermo, F. Ferrandi, and D. Sciuto. Prototyping pipelined applications on a heterogeneous fpga multiprocessor virtual platform. In *Proceedings of the 2009 Asia and South Pacific Design Automation Conference*, pages 317–322, Piscataway, NJ, USA, 2009. IEEE Press.

[24] Q. Wang and V. K. Prasanna. Multi-core architecture on fpga for large dictionary string matching. In *FCCM '09: Proceedings of the 2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*, pages 96–103.