Heracles 2.0: A Tool for Design Space Exploration of Multi/Many-core Processors

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

***Abstract*—This paper presents *Heracles*, an open-source, functional, parameterized, synthesizable multicore system toolkit. Such a multi/many-core design platform is a powerful and versatile research and teaching tool for architectural exploration and hardware-software co-design. The *Heracles* toolkit comprises the soft hardware (HDL) modules, application compiler, and graphical user interface. It is designed with a high degree of modularity to support fast exploration of future multicore processors of different topologies, routing schemes, processing elements or cores, and memory system organizations. It is a component-based framework with parameterized interfaces and strong emphasis on module reusability. The compiler toolchain is used to map C or C++ based applications onto the processing units. The GUI allows the user to quickly configure and launch a system instance for easy factorial development and evaluation. Hardware modules are implemented in synthesizable Verilog and are FPGA platform independent. *Heracles* tool is freely available under the open-source MIT license at http://projects.csail.mit.edu/heracles/.**

I. Introduction

The ability to integrate various computation components such as processing cores, memories, custom hardware units, and complex network-on-chip (NoC) communication protocols onto a single chip has significantly enlarged the design space and workload space in multi/many-core systems. The design of these systems requires tuning of a large number of parameters in order to find the most suitable hardware configuration, in terms of performance, area, and energy consumption, for a target application domain. This increasing complexity makes the need for efficient and accurate design tools more acute.

There are two main approaches currently used in the design space exploration of multi/many-core systems. One approach consists of building software routines for the different system components and simulating them to analyze system behavior. Software simulation has many advantages: i) large programming tool support; ii) internal states of all system modules can be easily accessed and altered; iii) compilation/re-compilation is fast; and among others iv) less constraining in terms of number of components (e.g., number of cores) to simulate. Some of the most stable and widely used software simulators are Simics [2]–commercially available full-system simulator–GEMS [3], Hornet [4], and Graphite [5]. However, software simulation of many-core architectures with cycle and bit level accuracy is almost time prohibitive, and many of these systems have to trade-off evaluation accuracy for execution speed. Although such trade-off is fair and even desirable in the early phase of the design exploration, making final micro-architecture decisions based on these software models over truncated applications or application traces leads to inaccurate or misleading system characterization.

The second approach used, often preceded by software simulation, is register-transfer level (RTL) simulation or emulation. RTL-level accuracy considerably reduces system behavior mis-characterization and helps avoid late discovery of system performance problems. The primary disadvantage of RTL simulation/emulation is that as the design size increases so does the simulation speed. But this problem can be circumvented by adopting synthesizable RTL and using hardware-assisted accelerators–field programmable gate array (FPGA)–to speed up system execution. Although FPGA resources constrain the size of design one can implement, recent advances in FPGA-based design methodologies have shown that such constraints can be over come. HAsim [6], for example, has shown using its time multiplexing technique how one can model a shared-memory multicore system including detailed core pipelines, cache hierarchy, and on-chip network, on a single FPGA. RAMP Gold [7] is able to simulate a 64-core shared-memory target machine capable of booting real operating systems running on a single Xilinx Virtex-5 FPGA board. Fleming et al [8] propose a mechanism by which complex designs can be efficiently and automatically partitioned among multiple FPGAs.

RTL-level design exploration for multi/many-core systems nonetheless remain unattractive to most researchers because it is still a time-consuming endeavor to build such large designs from the ground up and ensure correctness at all levels. Furthermore, researchers are generally interested in one key system area, such as processing core and/or memory organization, network interface, interconnect network, or operating system and/or application mapping. Therefore, we believe that if there is a platform-independent design framework, more specifically, a general hardware toolkit, which allows designers to compose their systems and modify them at will and with very little effort or knowledge of other parts of the system, the speed versus accuracy dilemma in design space exploration of many-core systems can be further mitigated.

To that end we propose *Heracles*, a functional, modular, synthesizable, parameterized multicore system toolkit. It is a powerful and versatile research and teaching tool for architectural exploration and hardware-software co-design. Without loss in timing accuracy and logic, complete systems can be constructed, simulated and/or synthesized onto FPGA, with minimal effort. The initial framework is presented in [1]. *Heracles* is designed with a high degree of modularity to support fast exploration of future multicore processors–different topologies, routing schemes, processing elements or cores, and memory system organizations–by using a library of com-

ponents, and reusing user-defined hardware blocks between different system configurations or projects. It has a compiler toolchain for mapping C or C++ based applications onto the core units. The graphical user interface (GUI) allows the user to quickly configure and launch a system instance for easy factorial development and evaluation. Hardware modules are implemented in synthesizable Verilog and are FPGA platform independent.

II. Related Work

In [9] Del Valle *et al* present an FPGA-based emulation framework for multiprocessor system-on-chip (MPSoC) architectures. LEON3, 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* [10], 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* [11] 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. Some designs focus primarly on the Network-on-chip (NoC). Lusala *et al* [12], for example, propose a scalable implementation of NoC on FPGA using a torus topology. Genko *et al* [13] 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* [14]. Other designs make use of multiple FPGAs. H-Scale [15], 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.

III. *Heracles* System

Heracles presents designers with a global and complete view of the inner workings of the multi/many-core system at the cycle-level granularity from instruction fetches at the processing core at each node to the flit arbitration at the routers, with RTL level correctness. It enables designers to explore different implementation parameters: 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. It is implemented with user-enabled performance counters and probes.

A. System overview

Figure 1 illustrates the general *Heracles*-based design flow. Full applications–written in single or multithreaded C or C++–can be directly compiled onto a given system instance using

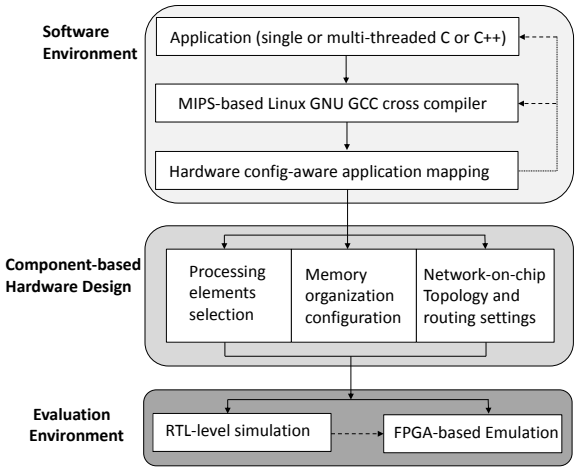

Fig. 1. *Heracles*-based design flow.

the *Heracles* MIPS-based GCC cross-compiler. The detail compilation process and application examples are presented in [1]. For the multi/many-core system, we take a component-based approach by providing clear interfaces to all modules for easy composition and substitutions. The system has multiple default settings options to allow users to quickly get a system running and only focus on the their area of interest. System and application binary can be executed in an RTL simulated environment and/or on an FPGA. Figure 2 shows two different views of a typical network node structure in *Heracles*.

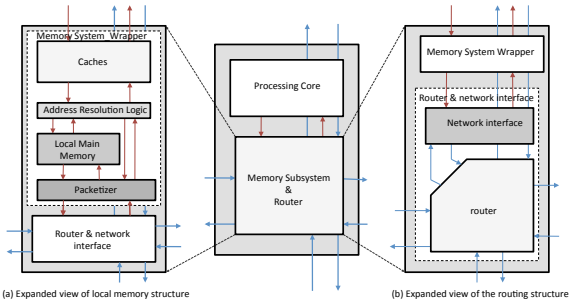

Fig. 2. Network node structure.

B. Processing Units

In the current version of the *Heracles* design framework, users can instantiate four different types of processor cores, or any combination thereof, depending on the programming model adopted and architectural evaluation goals:

1) Injector Core: The injector core (iCore) is the simplest processing unit. It emits and/or collects from the network user-defined data streams and traffic patterns. Although it does not do any useful computation, this type of core is very handy when user is only focusing on the network on-chip behavior. It is a good functionality to use for network congesting evaluation where applications running on real cores fail to produce enough data traffic to saturate the network.

2) Single Hardware-Threaded MIPS Core: There is an integer-based 7-stage 32-bit MIPS–Microprocessor without Interlocked Pipeline Stages–Core (sCore). This RISC architecture is widely used in commercial products and for teaching

purposes [16]. Most users are very familiar with this architecture and its operation, and will be able to easily modify it when necessary. Our implementation is very standard with some modifications for FPGAs. For example, the adoption of a 7-stage pipeline, due to block RAM access time on the FPGA. The architecture is 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. Stall and bypass signals are modified to support the extended pipeline. Instructions are issued and executed in-order, and the data memory accesses are also in-order.

3) Two-way Hardware-Threaded MIPS Core: A fully functional fine-grain hardware multithreaded MIPS core (dCore). There are two hardware threads in the core. The execution datapath for each thread is similar to the single-threaded core above. Each of the two threads has its own context which includes a program counter (PC), a set of 32 data registers, and one 32-bit state register. The core can dispatch instructions from any one of hardware contexts and supports precise interrupt–doorbell type with limited state saving. A single hardware thread is active on any given cycle, and pipeline stages must be drained between context switches to avoid state corruption. User has the ability to control the context switching conditions, e.g., minimum number of cycles to allocate to each hardware thread at the time, instruction or data cache misses.

4) Two-way Hardware-Threaded MIPS Core with Migration: The fourth type of core is also a two hardware-threaded processor but enhanced to support hardware-level thread migration and evictions (mCore). It is the user's responsibility to guarantee deadlock-freedom under this core configuration. One approach is to allocate local memory to contexts so on migration they are removed from the network. Another approach which requires no additional hardware modification to the core, is using Cho et al [17] deadlock-free thread migration scheme.

5) FPGA Synthesis Data: All the cores have the same interface, they are self-containing and oblivious to the rest of the system, therefore easily interchangeable. The cores are synthesized using Xilinx ISE Design Suite 11.5, with Virtex-6 LX550T package ff1760 speed -2, as the targeted FPGA board. Number of slice registers and slice lookup tables (LUTs) on the board are 687360 and 343680 respectively. Figure 3 shows the register and LUT utilization of the different cores. The two-way hardware-threaded core with migration consumes the most resources and is less than 0.5%. Figure 4 shows the clocking speed of the cores. The injector core, which does no useful computation, runs the fastest at $500.92MHz$ where the two-way hardware-threaded core runs the slowest at $127.4MHz$.

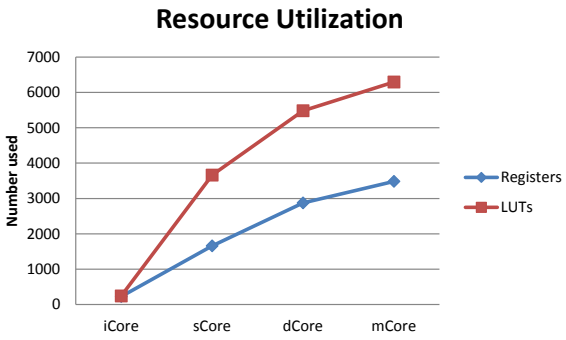

Fig. 3. FPGA resource utilization per core.

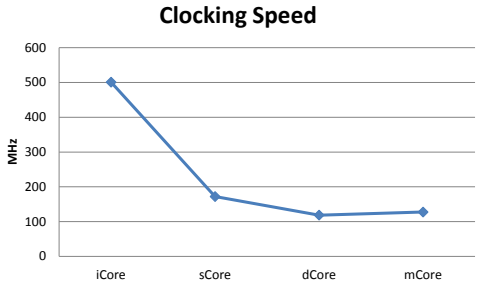

Fig. 4. Clocking speed of the cores.

C. Memory System Organization

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

1) Main Memory Configuration: The main memory is constructed to allow different memory space configurations. For Centralized Shared Memory (CSM) implementation, all processors share a single large main memory block; the local memory (shown in Figure 2) size is simply set to zero at all nodes except one. In Distributed Shared Memory (DSM), where each processing element has a local memory. The local memory is parameterized and has two very important attributes: the size can be changed on a per core-basis, providing support for both uniform and non-uniform distributed memory, and it can service a variable number of caches in a round-robin fashion. The fact that the local memory is parameterized to handle requests from a variable number of caches allows to present to the local memory the traffic coming into a 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. For cache coherence, a directory is attached to each local memory and MSI protocol is implemented as the default coherence mechanism. Remote access (RA) is also supported. In RA mode, the network packetizer directly sends network traffic to the caches. Memory structures are implemented in FPGA using block RAMs. There are 632 block RAMs on the Virtex-6 LX550T. A local memory of $0.26MB$ uses 64 block RAMs or %10.

2) Caching System: User can instantiate direct-mapped level 1 or levels 1 and 2 with the option of making level 2 an inclusive cache. The *INDEX_BITS* parameter defines the number of blocks or cache-lines in the cache where the *OFFSET_BITS* parameter defines block size. By default, cache and memory structures are implemented in FPGA using block

RAMs, but user can instruct *Heracles* to use LUTs for caches or some combination of LUTs and block RAMs. A single $2KB$ cache uses 4 FPGA block RAMs, 462 slice registers, 1106 slice LUTs, and runs at $228.8MHz$. If cache size is increased to $8KB$ by changing the *INDEX_BITS* parameter from 6 to 8, resource utilization and speed remains identical. Meanwhile if cache size is increase to $8KB$ by changing the *OFFSET_BITS* parameter from 3 to 5, resource utilization increases dramatically: 15 FPGA block RAMs, 1232 slice registers, 3397 slice LUTs, and runs at $226.8MHz$. FPGA-based cache design favors large number of blocks of small size versus small number of blocks of large size [1].

3) Network Interface: The Address Resolution Logic works with the *Packetizer* module, shown in Figure 2, 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. The *Packetizer* 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.

4) Hardware multithreading and caching: In this section, we examine the effect of hardware multithreading (HMT) on system performance. We run the 197.parser application from the SPEC CINT2000 benchmarks on a single node with the dCore as processing unit using two different inputs–one per thread–with five different execution interleaving policies.

- In setup 1: threads take turns to execute every 32 cycles, on a context switch, the pipeline is drained before the execution of another thread begins.
- For setup 2: thread switching happens every 1024 cycles.
- In setup 3: thread context swapping is initiated on an instruction or a data miss at the level 1 cache.
- For setup 4: thread interleaving occurs only when there is a data miss at the level 1 cache.
- In setup 5: thread switching happens when there is a data miss at the level 2 cache.

Figure 5 shows the total completion time of the two threads (in terms of number of cycles). It is worth noting that even with fast fine-grain hardware context switching, multithreading is most beneficial for large miss penalty events like level 2 cache missing or remote data accesses.

D. Heracles *network-on-chip (NoC)*

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).

[1]Cache-line size also has traffic implications at the network level

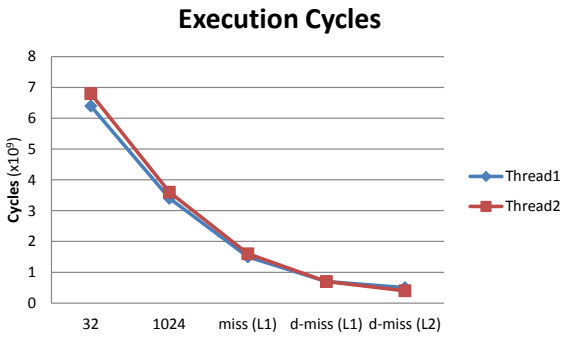

Fig. 5. Effects of hardware multithreading and caching

1) Flow control: Routing in *Heracles* can be done using either bufferless or buffered routers. Bufferless routing are generally used to reduce area and power overhead associated with buffered routing. Contention for physical link access is resolved by either dropping and retransmitting or temporarily misrouting–deflecting–of flits. With flit dropping an acknowledgment mechanism is needed to enable retransmission of lost flits. With flit deflection, a priority-based arbitration, e.g., *age-based*, is needed to avoid livelock. In *Heracles*, to mitigate some of the problems associated with the lossy bufferless routing, namely retransmission and slow arbitration logic, we supplement the arbiter with a routing table that can be statically and off-line configured on per-application basis.

The system default virtual-channel router conforms in its architecture and operation to conventional virtual-channel routers [18]. It has some input buffers to store flits while they are waiting to be routed to the next hop in the network. The router is modular enough to allow user to substitute different arbitration schemes. 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. In the buffered router the number of virtual channels per port and their sizes are controlled through *VC_PER_PORT* and *VC_DEPTH* parameters. Figure 6 shows the register and LUT utilization of the bufferless router and different buffer configurations of the buffered router. Figure 7 shows effect of virtual channels on router clocking speed. The key take-away is that larger number of VCs at the router increases both the router resource utilization and the critical path.

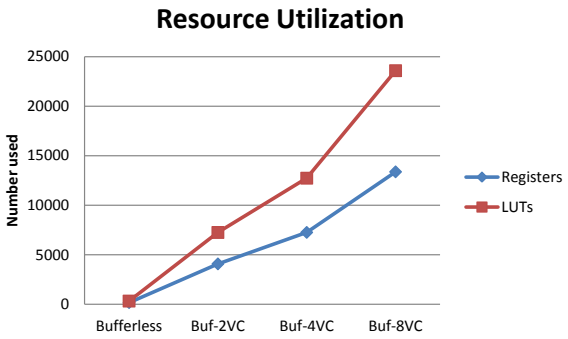

Fig. 6. FPGA resource utilization per router configuration.

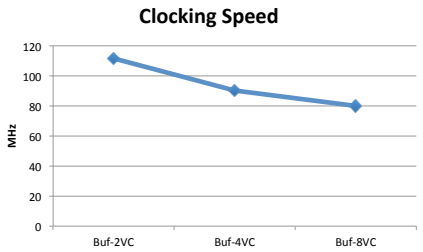

Fig. 7. Effect of virtual channels on router clocking speed.

2) Routing algorithm: Algorithms used to compute routes in network-on-chip (NoC) architectures, generally fall under two categories: *oblivious* and *dynamic* [19]. The default routers in *Heracles* primarily support *oblivious* routing algorithms using either fixed logic or routing table. Fixed logic is provided for dimension order routing (DOR) algorithms, which are widely used and have many desirable properties. 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. Both buffered and bufferless routers can make usage of the routing tables. *Heracles* provides support for both static and dynamic virtual channel allocation.

3) 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. Figure 8 shows the clocking speed of a bufferless router, a buffered router with strict round-robin arbiter (Buf-Arbiter1), a buffered router with weak round-robin arbiter (Buf-Arbiter2), and a buffered router with 7 ports for a 3D-mesh network. The bufferless router runs the fastest at $817.2MHz$, Buf-Arbiter1 and Buf-Arbiter2 run at the same speed ($\sim$ 112), although the arbitration scheme in Buf-Arbiter2 is more complex. The 7-port router runs the slowest due to higher arbitration logic level.

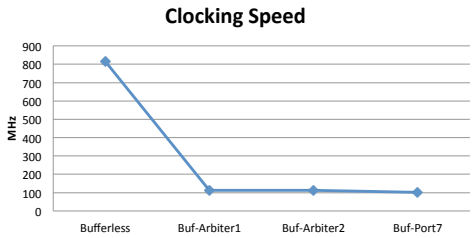

Fig. 8. Clocking speed of different router types.

E. Heracles *graphical user interface*

The graphical user interface (GUI) called *Heracles Designer* helps to quickly configure and launch system configurations. Figure 9 shows a screen shot of the GUI. On the core tab, user can select: (1) the type of core to generate, (2) the network topology of the system instance to generate, (3) the number of cores to generate, (4) traffic type, injection rate, and simulation cycles in the case of an injector core, or (5) different pre-configured settings. *Generate* and *Run* buttons on this tab are used to automatically generate the Verilog files and to launch the synthesis process or specified simulation environment. The second tab–memory system tab–allows user to set: (1) maim memory configuration (e.g., Uniformed Distributed), (2) total main memory size, (3) instruction and data cache sizes, (4) level 2 cache, and (5) FPGA favored resource (LUT or block RAM) for cache structures. The on-chip network tab covers all the major aspects of the system interconnect: (1) routing algorithm, (2) number of virtual channels (VCs) per port, (3) VC depth, (4) core and switch bandwidths, (5) routing tables programming, by selecting source/destination pair or flow ID, router ID, output port, and VC (allowing user-defined routing paths), and (6) number of flits per packet for injector-based traffic. The programming tab is updated when user changes the number of cores in the system, user can: (1) load a binary file onto a core, (2) load a binary onto a core and set the starting address for another core to point to that binary, (3) select where to place the data section or stack pointer of a core (it can be local, on the same core as the binary or on another core), and (4) select which cores to start.

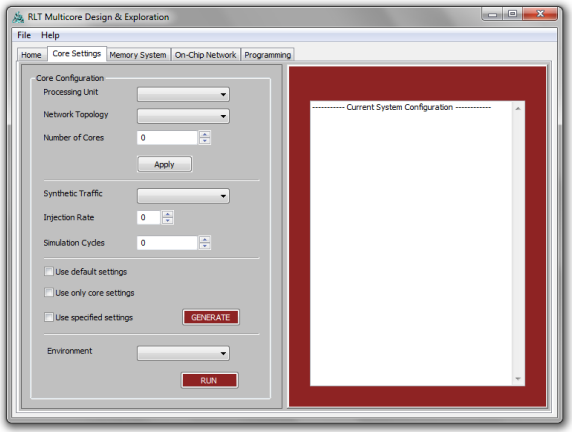

Fig. 9. *Heracles* designer graphical user interface.

F. Programming models

The *Heracles* design tool supports the following programming models:

- Executing a single program on a single core.
- Running the same program on multiple cores (program inputs can be different), in this model the program binary is loaded to one core and the program counter at other cores points to the core with the binary.
- Executing one program across multiple cores.
- Running multiple programs on one core and hardware multithread them.
- Executing multiple programs (both single threaded and multithreaded) on multiple cores.

G. Full 2D-mesh systems

The synthesis results of five multicore systems of size: 2×2, 3×3, 4×4, 5×5, and 6×6 arranged in 2D-mesh topology

are summarized below. Table I gives the key architectural characteristics of the multicore system. All five systems run at $105.5MHz$, which is the clock frequency of the router, regardless of the size of the mesh.

Core	
ISA	32-Bit MIPS
Hardware threads	1
Pipeline Stages	7
Bypassing	Full
Branch policy	Always non-Taken
Outstanding memory requests	1
Level 1 Instruction/Data Caches	
Associativity	Direct
Size	variable
Outstanding Misses	1
On-Chip Network	
Topology	2D-Mesh
Routing Policy	DOR and Table-based
Virtual Channels	2
Buffers per channel	8

TABLE I
2D-MESH SYSTEM ARCHITECTURE DETAILS.

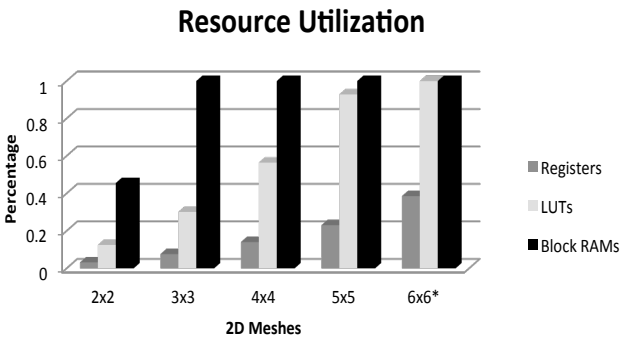

Fig. 10. Percentage of FPGA resource utilization per mesh size.

Figure 10 summarizes the FPGA resource utilization by the different systems in terms of registers, lookup tables, and block RAMs. In the 2×2 and 3×3 configurations, the local memory is set to $260KB$ per core. The 3×3 configuration uses 99% of block RAM resources at $260KB$ of local memory per core. For the 4×4 configuration the local memory is reduced to $64KB$ per core, and the local memory in the 5×5 configuration is set to $32KB$. The 6×6 configuration, with $16KB$ of local memory per core, fails during map and router, due to lack of LUTs.

IV. Conclusion

In this work, we present the new *Heracles* design toolkit which is comprised of the soft hardware (HDL) modules, application compiler, and a graphical user interface. It is a component-based framework that gives researchers the ability to create complete, realistic, synthesizable, multi/many-core architecture for fast and high accuracy design space exploration. In this environment, user can explore design trade-offs at the processing unit level, the memory organization and access level, and the network on-chip level. The *Heracles* tool is open-source and can be downloaded at http://projects.csail.mit.edu/heracles/.

References

[1] M. Kinsy, M. Pellauer, and S. Devadas, "Heracles: Fully synthesizable parameterized MIPS-based multicore system," in *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, sept. 2011, pp. 356 –362.

[2] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, Feb. 2002.

[3] W. Yu, "Gems a high performance em simulation tool," in *Electrical Design of Advanced Packaging Systems Symposium, 2009. (EDAPS 2009). IEEE*, dec. 2009, pp. 1 –4.

[4] M. Lis, P. Ren, M. H. Cho, K. S. Shim, C. Fletcher, O. Khan, and S. Devadas, "Scalable, accurate multicore simulation in the 1000-core era," in *Performance Analysis of Systems and Software (ISPASS), 2011 IEEE International Symposium on*, april 2011, pp. 175 –185.

[5] J. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal, "Graphite: A distributed parallel simulator for multicores," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, jan. 2010, pp. 1 –12.

[6] M. Pellauer, M. Adler, M. Kinsy, A. Parashar, and J. Emer, "Hasim: FPGA-based high-detail multicore simulation using time-division multiplexing," in *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, feb. 2011, pp. 406 –417.

[7] Z. Tan, A. Waterman, R. Avizienis, Y. Lee, H. Cook, D. Patterson, and K. Asanovic and, "Ramp gold: An FPGA-based architecture simulator for multiprocessors," in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, june 2010, pp. 463 –468.

[8] K. E. Fleming, M. Adler, M. Pellauer, A. Parashar, A. Mithal, and J. Emer, "Leveraging latency-insensitivity to ease multiple FPGA design," in *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*, ser. FPGA '12. New York, NY, USA: ACM, 2012, pp. 175–184.

[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," in *Very Large Scale Integration, 2006 IFIP International Conference on*, oct. 2006, pp. 140 –145.

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

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

[12] A. Lusala, P. Manet, B. Rousseau, and J.-D. Legat, "Noc implementation in FPGA using torus topology," in *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, aug. 2007, pp. 778 –781.

[13] N. Genko, D. Atienza, G. De Micheli, J. Mendias, R. Hermida, and F. Catthoor, "A complete network-on-chip emulation framework," in *Design, Automation and Test in Europe, 2005. Proceedings*, march 2005, pp. 246 – 251 Vol. 1.

[14] N. Banerjee, P. Vellanki, and K. Chatha, "A power and performance model for network-on-chip architectures," in *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, vol. 2, feb. 2004, pp. 1250 – 1255 Vol.2.

[15] 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 *VLSI, 2007. ISVLSI '07. IEEE Computer Society Annual Symposium on*, march 2007, pp. 21 –28.

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

[17] M. H. Cho, K. S. Shim, M. Lis, O. Khan, and S. Devadas, "Deadlock-free fine-grained thread migration," in *Networks on Chip (NoCS), 2011 Fifth IEEE/ACM International Symposium on*, may 2011, pp. 33 –40.

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

[19] L. M. Ni and P. K. McKinley, "A survey of wormhole routing techniques in direct networks," *Computer*, vol. 26, no. 2, pp. 62–76, 1993.