Architecture Emulation and Simulation of Future Many-Core Epiphany RISC Array Processors

David A. Richie[1] and James A. Ross[2]

[1] Brown Deer Technology, MD, USA
drichie@browndeertechnology.com
[2] U.S. Army Research Laboratory, MD, USA
james.a.ross176.civ@mail.mil

Abstract. The Adapteva Epiphany many-core architecture comprises a scalable 2D mesh Network-on-Chip (NoC) of low-power RISC cores with minimal uncore functionality. The Epiphany architecture has demonstrated significantly higher power-efficiency compared with other more conventional general-purpose floating-point processors. The original 32-bit architecture has been updated to create a 1,024-core 64-bit processor recently fabricated using a 16nm process. We present here our recent work in developing an emulation and simulation capability for future many-core processors based on the Epiphany architecture. We have developed an Epiphany SoC device emulator that can be installed as a virtual device on an ordinary x86 platform and utilized with the existing software stack used to support physical devices, thus creating a seamless software development environment capable of targeting new processor designs just as they would be interfaced on a real platform. These virtual Epiphany devices can be used for research in the area of many-core RISC array processors in general.

Keywords: RISC, Network-on-Chip, Emulation, Simulation, Epiphany.

1 Introduction

Recent developments in high-performance computing (HPC) provide evidence and motivation for increasing research and development efforts in low-power scalable many-core RISC array processor architectures. Many-core processors based on two-dimensional (2D) RISC arrays have been used to establish the first and fourth positions on the most recent list of top 500 supercomputers in the world [1]. Further, this was accomplished without the use of commodity processors and with instruction set architectures (ISAs) evolved from a limited ecosystem, driven primarily by research laboratories. At the same time, the status quo in HPC of relying upon conventional commodity processors to achieve the next level of supercomputing capability has encountered major setbacks. Increasing research into new and innovative architectures has emerged as a significant recommendation as we transition into a post-Moore era [2] where old trends and conventional wisdom will no longer hold.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_22

At the same time, there is increasing momentum for a shift to open hardware models to facilitate greater innovation and resolve problems with the ecosystems that presently provide the majority of computing platforms. Open hardware architectures, especially those based on principles of simplicity, are amenable to analysis for reliability, security, and correctness errata. This stands in stark contrast to the lack of transparency we find with existing closed architectures where security and privacy defects are now routinely found years after product deployment [3]. Open hardware architectures are also likely to spark more rapid and significant innovation, as was seen with the analogous shift to open-source software models. Recognition of the benefits of an open hardware architecture can be seen in the DARPA-funded RISC-V ISA development, which has recently lead to the availability of a commercial product and is based on a BSD open source licensed instruction set architecture.

Whereas the last decade was focused mainly on using architectures provided by just a few large commercial vendors, we may be entering an era in which architectures research will become increasingly important to define, optimize, and specialize architectures for specific classes of applications. A reduction in barriers to chip fabrication and open source hardware will further advance an open architecture model where increasing performance and capability must be extracted with innovative design rather than a reliance on Moore's Law to bring automatic improvements.

More rapid and open advances in hardware architectures will require unique capabilities in software development to resolve the traditional time lag between hardware availability and the software necessary to support it. This problem is long standing and one that is more pragmatic than theoretical. Significant software development for new hardware architectures will typically only begin once the hardware itself is available. Although some speculative work can be done, the effectiveness is limited. Very often the hardware initially available will be in the form of a development kit that brings unique challenges, and will not entirely replicate the target production systems. Based on our experience with Epiphany and other novel architectures, the pattern generally follows this scenario. Efforts to develop hardware/software co-design methodologies can benefit development in both areas. However in this work we are proposing an approach that goes further.

Modern HPC platforms are almost universally used for both development and production. With increasing specialization to achieve extreme power and performance metrics for a given class of problems, high-performance architectures may become well designed for a specific task, but not well suited to supporting software development and porting. An architecture emulation and simulation environment, which replicates the interfacing to real hardware, could be utilized to prepare software for production use beyond the early hardware/software co-design phase. As an example, rather than incorporate architectural features into a production processor to make it more capable at running compiler and development tools, the production processor should be purpose-built, with silicon and power devoted to its specific production requirements. A more general-purpose support platform can then be used to develop and test both software and hardware designs at modest scale in advance of deployment on production systems.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_22

The focus of this research has been on the Epiphany architecture, which shares many characteristics with other RISC array processors, and is notable at the present time as the most power-efficient general-purpose floating-point processor demonstrated in silicon. To the best of our knowledge, Epiphany is the only processor architecture that has achieved the power-efficiency projected to be necessary for exascale. The Adapteva Epiphany RISC array architecture [4] is a scalable 2D array of low-power RISC cores with minimal un-core functionality supported by an on-chip 2D mesh network for fast inter-core communication. The Epiphany-III architecture is scalable to 4,096 cores and represents an example of an architecture designed for power-efficiency at extreme on-chip core counts. Processors based on this architecture exhibit good performance/power metrics [5] and scalability via a 2D mesh network [6,7], but require a suitable programming model to fully exploit the architecture. A 16-core Epiphany-III processor [8] has been integrated into the Parallella mini-computer platform [9] where the RISC array is supported by a dual-core ARM CPU and asymmetric shared-memory access to off-chip global memory. Most recently, a 1024-core, 64-bit Epiphany-V was fabricated by DARPA and is anticipated to have much higher performance and energy efficiency [10].

The overall motivation for this work stems from ongoing efforts to investigate future many-core processors based on the Epiphany architecture. At present we are investigating the design of a hybrid processor based on a 2D array of Epiphany-V compute cores with several RISC-V supervisor cores acting as an on-die CPU host. In support of such efforts, we need to develop a large-scale emulation and simulation capability to enable rapid design and specialization by allowing testing and software development using simulated virtual architectures. In this work, a special emphasis is placed on achieving a seamless transition between emulated architectures and physical systems. The overall design and implementation of the proposed emulation and simulation environment will be generally applicable to supporting more general research and development of other many-core RISC array processors.

The main contributions presented here are as follows: we present a description of the design and implementation of an Epiphany architecture emulator that can be used to construct virtual Epiphany devices on an ordinary x86 workstation for software development and testing. Early results from testing and validation of the Epiphany ISA emulator are presented.

2 Background

The Adapteva Epiphany MIMD architecture is a scalable 2D array of RISC cores with minimal uncore functionality connected with a fast 2D mesh Network-on-Chip (NoC). The Epiphany-III (16-core) and Epiphany-IV (64-core) processors have a RISC CPU core that support a 32-bit RISC ISA with 32 KB of shared local memory per core (used for both program instructions and data), a mesh network interface, and a dual-channel DMA engine. Each RISC CPU core contains a 64-word register file, sequencer, interrupt handler, arithmetic logic unit, and a floating point unit. The fully memory-mapped architecture allows shared memory access to global off-chip

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_22

memory and shared non-uniform memory access to the local memory of each core. The Epiphany-V processor, shown in Figure 1, was extended to support 64-bit addressing and floating-point operations. The 1,024-core Epiphany-V processor was fabricated by DARPA at 16nm.

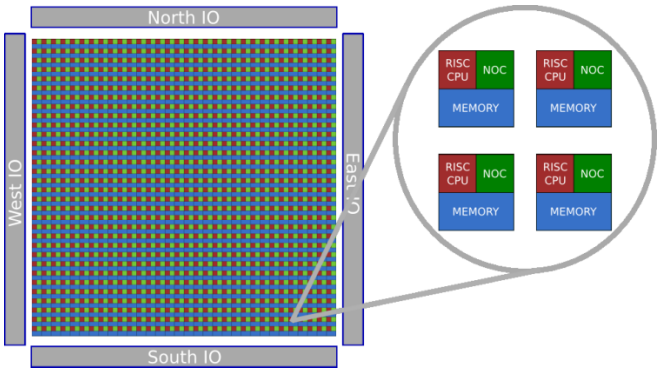

Fig. 1. The Epiphany-V RISC array architecture. A tiled array of 64-bit RISC cores are connected through a 2D mesh NoC for signaling and data transfer. Communication latency between cores is low, and the amount of addressable data contained on a mesh node is low (64 KB). Three on-chip 136-bit mesh networks enable on-chip read transactions, on-chip write transactions, and off-chip memory transactions.

The present work leverages significant research and development efforts related to the Epiphany architecture, and which produced the software stack to support many-core processors like Epiphany. Previous work included investigating a parallel programming models for the Epiphany architecture, including threaded MPI [11], OpenSHMEM [12,13], and OpenCL [14] support for the Epiphany architecture. In all cases the parallel programming model involved explicit data movement between the local memory of each core in the RISC array, or to/from the off-chip global DRAM. The absence of a hardware cache necessitated this movement to be controlled in software explicitly. Also relevant to the present work, progress was made in the development of a more transparent compilation and run-time environment whereby program binaries could be compiled and executed directly on the Epiphany co-processor of the Parallella platform without the use of an explicit host/coprocessor offload model [15].

3 Simulation Framework for Future Many-Core Architectures

There are several technical objectives addressed in the design and implementation of a simulation framework for Epiphany-based many-core architectures. First and foremost, the ISA emulator(s) must enable fast emulation of real compiled binaries since they are to be used for executing real application code, and not merely for targeted testing of sub-sections of code. This will require a design that emphasizes efficiency and potential optimization. An important application will be the use of virtual devices

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_22

operating at a level of performance that, albeit slower than real hardware, is amenable to executing large applications.

Cycle-accurate correctness of the overall system is not an objective of the design, since the goal is not to verify the digital logic of a given hardware design; sufficient tools already exist for this purpose as part of the VLSI design process. The goal instead is to ensure that the emulation and simulation environment is able to execute real applications with correct results and with the overall performance modeled sufficiently well so as to reproduce meaningful metrics. Thus, performance modeling is done by way of directly executing compiled binary code rather than employing theoretical models of the architecture. The advantage of this approach is that it will simultaneously provide a natural software development environment for proposed architectures and architecture changes without the need for physical devices. The software development and execution environment should not appear qualitatively different between simulation and execution on real hardware.

3.1 Epiphany Architecture Emulator

The design and implementation of an emulator for the Epiphany architecture is initially focused on the 32-bit architecture since physical devices are readily available for testing. The more recent extension of the ISA to support 64-bit instructions will be addressed in future work. The emulator for the 32-bit Epiphany architecture is implemented as a modular C++ class, in order to support the rapid composition and variation of specific devices for testing and software development. Implementing the emulator directly in C++, and without the use of additional tools or languages, avoids unnecessary complexity and facilitates modifications and experimentation. In addition, the direct implementation of the emulator in C++ will allow for the highest levels of performance to be achieved through low-level optimization. The emulator class is primarily comprised of an instruction dispatch method, implementations of the instructions forming the ISA, and additional features external to the RISC core but critical for the architecture functionality, such as the DMA engines.

The present design uses an instruction decoder based on an indirect threaded dispatch model. The Epiphany instruction decode table was analyzed to determine how to efficiently dispatch the 16-bit and 32-bit instructions of the ISA. Examining the lowest 4 bits of any instruction can be used to differentiate 16-bit and 32-bit instruction. For 16-bit instructions, it was determined that the lower 10 bits could efficiently dispatch the instruction by way of a pre-initialized call table for all 16-bit instructions. For 32-bit instructions, it was determined that a compressed bit-field of {b19...b16|b9...b0} could efficiently dispatch instructions by way of a larger pre-initialized call table that extends the table used for 16-bit instructions. The instruction call table is sparse, representing a balance of trade-offs between table size and dispatch efficiency.

The instruction dispatch design will allow for any instruction to stall in order to support more realistic behaviors. Memory and network interfaces are implemented as separate abstractions to allow for different memory and network models. Initially, a simple memory mapped model is used, and the incorporation of more complex and

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_22

accurate memory models will be introduced in future work. The emulator supports the Epiphany architecture special registers, dual DMA engines, and interrupt handler. The DMA engines and interrupt support are based on a direct implementation of the behaviors described in the Epiphany architecture reference, and are controlled by the relevant special registers.

As will be described in more detail below, the emulator was validated using applications developed in previous work and has been demonstrated to correctly execute complex code that included interrupts, asynchronous DMA transfers, and host-coprocessor synchronization for host callback capabilities and direct Epiphany program execution without supporting host code.

3.2 Virtual Epiphany Devices

Rather than incorporate the emulator into a stand-alone tool, the chosen design allows the use of the emulator to create virtual Epiphany devices that present an interface identical to that of a physical coprocessor and is indistinguishable from a user application. This is accomplished by creating a nearly identical interface to that which is found on the Parallella boards. On this platform, the dual-core ARM host and the Epiphany-III device share 32 MB of mapped DRAM, and the Epiphany SRAM and registers are further mapped into the Linux host address space. The result is that with one exception of an ioctl() call intended to force a hard reset of the device, all interactions occur via reads and writes to specific memory locations. Further, the COPRTHR-2 API uses these mappings to create a unified virtual address space (UVA) between the ARM host and Epiphany coprocessor so that no address translation is required when transferring control from host to coprocessor.

Low-level access to the Epiphany coprocessor is provided by the device special file mounted on the Linux host file system at /dev/epiphany/mesh0. The setup of the UVA described above is carried out entirely through mmap() calls of this special file from within the COPRTHR software stack. Proper interaction with the Epiphany device requires nothing more than knowing the required mappings and the various protocols to be executed via ordinary reads and writes to memory. In order to create a virtual Epiphany device, a shared memory region is mounted at /dev/shm/e32.0.0 that replicates the memory segments of a physical Epiphany device, as shown in **Error! Reference source not found.**.

The emulator described in Section 3 is then used to compose a device of the correct number of cores and topology, and then run "on top" of this shared memory region. By this, it is meant that the emulator core will have mapped its interfacing of registers, local SRAM, and external DRAM to specific segments of the shared memory region. By simply redirecting the COPRTHR API to map /dev/shm/e32.0.0 rather than /dev/epiphany/mesh0, user applications executing on the host see no difference in functionality between a physical and virtual Epiphany device.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_22

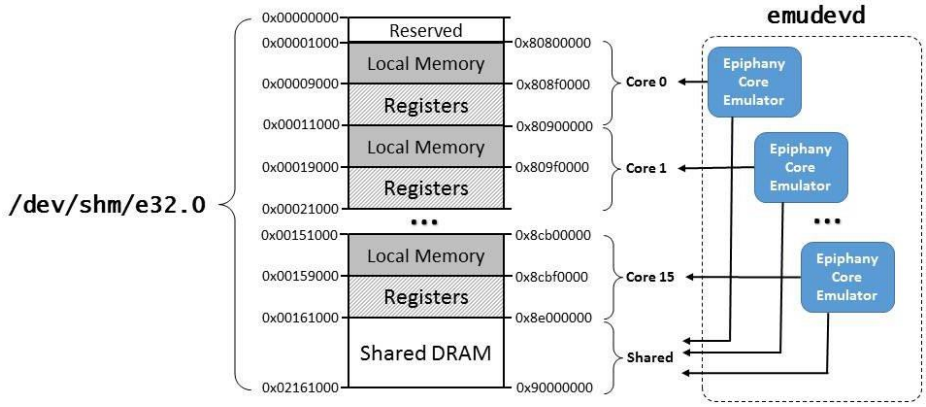

Fig. 2. The shared memory region replicates the physical memory segments of an Epiphany processor. Each emulated core has virtual local and global addresses which match the physical addressing.

The only real distinction is the replacement of the ioctl() call mentioned above with a direct back-channel mechanism for forcing the equivalent of a hard reset of the virtual device. In addition, whereas the device special file is mapped as though it represented the full and highly sparse 1 GB address space of the Epiphany architecture, the shared memory region is stored more compactly to optimize the storage required for representing a virtual Epiphany device. This is achieved by removing unused segments of the Epiphany address space for a given device, and storing only the core-local memory, register files, and global memory segments within the shared memory region. As an example, for a 256-core device with 32 MB of global memory, the compressed address space of the device will only occupy 42 MB rather than a sparse the sparse 1 GB address space.

The Linux daemon process emudevd creates this shared memory region and then operates in either active or passive mode. In active mode, an emulator is started up and begins executing on the shared memory region. If subsequently the user executes a host application that utilizes the Epiphany coprocessor, it will find the virtual device to be active and running, just as it would find a physical device.

The result of fully decoupling the emulator and user applications has an interesting benefit. Having a coprocessor in an uncertain state is closer to reality, and there is initially a low-level software requirement to develop reliable initialization procedures to guarantee that an active coprocessor can be placed in a known state regardless of the state in which it is found. This was the case during early software development for the Epiphany-III processor and the Parallella board. Issues of device lockup and unrecoverable states were common until a reliable procedure was developed. If a user application were executed through a "safe" emulator tool placing the emulated device in a known startup state, this would be overly optimistic and avoid common problems encountered with real devices. The decoupling of the emulator and user application replicates realistic conditions and provides visibility into state initialization that was previously only indirectly known or guessed at during early software development.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_22

It is worth emphasizing the transparency and utility of these virtual Epiphany devices. The Epiphany GCC and COPRTHR tool chains are easily installed on an x86 platform, and with which Epiphany/Parallella application code can be cross-compiled. By simply installing and running the emudevd daemon on the same x86 platform, it is possible to then execute the cross-compiled code directly on the x86 platform. The result is a software development and testing environment equivalent to that of a Parallella development board. Furthermore, the virtual device is configurable in terms of the number of cores and other architectural parameters. It is also possible to install multiple virtual devices appearing as separate shared memory device special files under /dev/shm. Finally, through modifications to the (open-source) Epiphany emulator, researchers can explore "what-if" architecture design modifications. At the same time, the user application code is compiled and executed just as it would be on a Parallella development board with a physical device. A discussion of the initial testing and verification performed using the Epiphany ISA emulator and virtual devices will be presented in Section 4.

4 Epiphany Emulator Results

Initial results from testing the Epiphany ISA emulator are promising and demonstrated functional correctness in a benchmark application, generating results identical to those generated using a physical Epiphany-III device. Two platforms were used for testing. A Parallella development board was used for reference purposes, and was comprised of a Zynq 7020 dual-core ARM CPU and a 16-core Epiphany-III coprocessor, and with a software stack consisting of Ubuntu Linux 15.04, GCC 4.9.2 for compiling host applications, GCC 5.2.0 for cross-compiling Epiphany binaries, and the COPRTHR-2 SDK for providing software support for the Epiphany coprocessor. Emulation was tested on an ordinary x86 workstation with an eight-core AMD FX-8150 CPU, and with a software stack consisting of Linux Mint 17.3, GCC 5.3.0 for compiling host applications, GCC 5.4.0 for cross-compiling Epiphany binaries, and the COPRTHR-2 SDK for providing software support for the Epiphany coprocessor.

Two test cases were used for initial debugging and then validation of the Epiphany architecture emulator. The first test application involved a simple "Hello, World!" type program that used the COPRTHR host-coprocessor interoperability. This represents a non-trivial interaction between the host application and the code executed on the Epiphany coprocessor. The test code was compiled on the x86 workstation using the COPRTHR coprcc compiler option '-fhost' to generate a single host executable that will automatically run the cross-compiled Epiphany binary embedded within it. We note that the test code was copied over from a Parallella development board and left unmodified. When executing the host program just as it would be executed on the Parallella development platform, the application ran successfully on the x86 workstation using the Epiphany emulator. From the perspective of the host-side COPRTHR API, the virtual Epiphany device appears to be a physical Epiphany coprocessor that was simply mounted at a different location within the Linux file system.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_22

A variation of this "Hello, World!" type program was also tested using an explicit host program to load and execute a function on one or more cores of the Epiphany coprocessor. For this test, the Epiphany binary was first compiled using the GCC cross-compiler on the x86 workstation, with results being very similar to the first successful test case. A cross-compiled Epiphany binary was then copied over from the Parallella platform and used directly on the x86 workstation with emulation. Using the binary compiled on the different platform, no differences in behavior were observed. This demonstrated that Epiphany binaries could be copied from the Parallella platform and executed without modification using emulation on the x86 workstation. Using the COPRTHR shell command coprsh we were able to execute the test program using various numbers of cores up to 16, with success in all cases. From a user perspective, the "look and feel" of the entire exercise did not differ from that experienced with software development on a Parallella development board.

The overall results from the above testing demonstrated that the test codes previously developed on the Parallella platform using the COPRTHR API could be compiled and executed via emulation on an ordinary workstation, seamlessly, and using an identical workflow. For a more demanding test of the emulator, a benchmark application was used that exercises many more features of the Epiphany coprocessor.

The Cannon matrix-matrix multiplication benchmark was implemented in previous work for Epiphany using the COPRTHR API with threaded MPI for inter-core data transfers [11]. This application code was highly optimized and used previously for extensive benchmarking of the Epiphany architecture and provides a non-trivial test case for the emulator for several reasons. The Cannon algorithm requires significant data movement between cores as sub-matrices are shifted in alternating directions. These inter-core data transfers are implemented using a threaded MPI interface, and specifically the MPI_Sendrecv_replace() call which requires precise inter-core synchronization. Finally, the data transfers from shared DRAM to core-local memory are performed using DMA engines. As a result, this test case places significant demands on the architecture emulator and is built up from complex layers of support with the COPRTHR device-side software stack. For a complete and detailed discussion of this Epiphany benchmark application see reference [11].

Error! Reference source not found. shows the actual workflow and output from the command-line used to build and execute the benchmark on the x86 workstation with the emulated virtual Epiphany device. This workflow is identical to that which is used on a Parallella platform, and the benchmark executes successfully without error. It was mentioned above that the application code leverages the COPRTHR software stack; it is important to emphasize again that no changes have been made to the COPRTHR software stack to support emulation. The virtual Epiphany devices create a seamless software development and testing capability, and appear to the supporting middleware to be real devices.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_22

```
] gcc -I$COPRTHR_INC_PATH -c cannon_host.c

] gcc -rdynamic -o cannon.x cannon_host.o \
  -L$COPRTHR_LIB_PATH -lcoprthr -lcoprthrcc -lm -ldl

] coprcc -o cannon_tfunc.e32 cannon_tfunc.c \
  -L$COPRTHR_LIB_PATH -lcoprthr_mpi

] ./cannon.x -d 4 -n 32

COPRTHR-2-BETA (Anthem) build 20180118.0014
main: Using -n=32, -s=1, -s2=1, -d=4
main: dd=0
main: 0x2248420 0x223f3f0
main: mpiexec time 0.117030 sec
main: # errors: 0
```

Fig. 3. Workflow and output from the command-line used to build and execute the Cannon matrix-matrix multiplication benchmark on the x86 workstation using the emulated virtual Epiphany device. The workflow and execution is unchanged from that used on the Epiphany Parallella platform where the benchmark was first developed. This seamless interface to the Epiphany ISA emulator enables a testing and software development environment for new designs that is identical to production hardware.

The idea behind using emulated devices is that they allow for testing and software development targeting future architecture changes. The previously developed matrix-matrix multiplication benchmark allowed command line options to control the size of the matrices and the number of threads used on the Epiphany device. With a physical Epiphany-III, the range of valid parameters was limited to 16 threads, with submatrices required to fit in the core-local memory of the coprocessor core executing each thread. Using emulated Epiphany devices, it was possible to execute this benchmark on 64 and 256 cores, and with larger matrices.

The results from this testing are shown in **Error! Reference source not found.** where for each combination of device, matrix size, and thread count, the total execution time for the benchmark is reported in terms of 1,000s of device clocks and wall-clock time in milliseconds. For each reported result, the numerical accuracy of the calculated matrix satisfied the default error test requiring that the relative error of each matrix element be less than 1% as compared with the analytical result. This criterion was used consistently in identifying coding errors during benchmark development, and is used here in validating the successful executing of the benchmark through emulation.

Data for certain combinations of device, matrix size, and thread count are not shown due to several factors. First, results for larger thread counts require devices with at least as many cores. Additionally, the size of the matrices is limited by core count since the distributed submatrices must fit in core-local memory, which for the

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_22

purposes of testing was kept at 32 KB. Finally, smaller matrices have a lower limit in terms of the number of threads that can be used, and this limit is impacted by a four-way loop unrolling in the optimized matrix-matrix multiplication algorithm.

The overall trend shows that the emulator executes the benchmark in fewer clocks when compared to a physical device. This result is expected, since the instruction execution at present is optimistic and does not account for pipeline stalls. Having such an optimistic mode of emulation is not necessarily without utility, since it allows for faster functional testing of software. The emulator also, as expected, takes longer to execute the benchmark than a physical device. Future work will attempt to address the issue of enabling more realistic clock cycle estimates while also optimizing the emulator for faster execution in terms of wall clock time. Finally, it should be noted that the scaling of wall clock time with the number of emulated cores is expected since the emulator is presently not parallelized in any way.

Of importance is the fact that as a result of this work, the software stack for devices that do not yet exist in silicon may be developed. A case in point can be seen in the results for the 256-core device which does not correspond to any fabricated Epiphany device. The ability to prepare software in advance of hardware will shorten significantly the traditional lag that accompanies hardware and then software development.

Table 1. Performance results for the execution of the Cannon matrix-matrix multiplication benchmark using physical and emulated devices for different matrix sizes and thread counts. Results are shown in terms of 1,000s of device clocks (wall clock time in milliseconds).

		Epiphany-III	Emulated Device		
Matrix	Threads	16-core	16-core	64-core	256-core
16^2	1	104 (2.7)	46 (59)	60 (340)	79 (2667)
	4	90 (2.8)	11 (53)	12 (310)	16 (2485)
	16	109 (2.7)	14 (57)	14 (325)	18 (2288)
32^2	1	201 (3.1)	112 (138)	127 (682)	145 (4032)
	4	155 (3.1)	37 (86)	38 (448)	41 (2712)
	16	145 (3.1)	22 (70)	23 (325)	26 (2311)
	64	-	-	47 (569)	51 (2868)
64^2	4	479 (4.5)	201 (298)	202 (1421)	205 (7679)
	16	311 (4.0)	73 (141)	73 (672)	77 (3773)
	64	-	-	64 (663)	67 (3358)
	256	-	-	-	258 (8773)
128^2	16	1062 (9.4)	400 (561)	400 (2395)	404 (13522)
	64	-	-	165 (1230)	168 (6033)
	256	-	-	-	291 (9831)
256^2	64	-	-	816 (4849)	820 (23651)
	256	-	-	-	490 (15731)

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_22

5 Conclusion and Future Work

An Epiphany 32-bit ISA emulator was implemented that may be configured as a virtual many-core device for testing and software development on an ordinary x86 platform. The design enables a seamless interface allowing the same tool chain and software stack to be used to target and interface to the virtual device in a manner identical to that of real physical devices. This has been done in the context of research into the design of future many-core processors based on the Epiphany architecture. The emulator has been validated for correctness using benchmarks previously developed for the Epiphany Parallella development platform, which work without modification using emulated devices.

Efforts to develop the software support for simulating and evaluating future many-core processor designs based on the Epiphany architecture reflects ongoing work. In the near term, the emulator will be improved with better memory models and instruction pipeline timing to allow for the prediction of execution time for software applications. The emulator will be extended to support the more recent 64-bit ISA which is backward compatible with the 32-bit Epiphany architecture. With direct measurements taken from the Epiphany-V SoC the emulator will be refined to produce predictive metrics such as clock cycle costs for software execution. With this calibration, general specializations to the architecture can then be explored with real software applications.

6 Acknowledgements

This work was supported by the U.S. Army Research Laboratory. The authors thank David Austin Richie for contributions to this work.

References

1. https://www.top500.org/lists/2017/11/. [Accessed 04-Feb-2018].
2. https://www.nitrd.gov/nitrdgroups/images/b/b4/NSA_DOE_HPC_TechMeetingReport.pdf . [Accessed 04-Feb-2018].
3. https://spectreattack.com/spectre.pdf, https://meltdownattack.com/meltdown.pdf. [Accessed 04-Feb-2018].
4. "Adapteva introduction." [Online]. Available: http://www.adapteva.com/introduction/. [Accessed: 08-Jan-2015].
5. A. Olofsson, T. Nordström, and Z. Ul-Abdin, "Kickstarting high-performance energy-efficient manycore architectures with Epiphany," *ArXiv Prepr. ArXiv14125538*, 2014.
6. D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. Brown III, and A. Agarwal, "On-chip interconnection architecture of the tile processor," *IEEE Micro*, vol. 27, no. 5, pp. 15–31, Sep. 2007.
7. M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, W. Lee, A. Saraf, N. Shnidman, V. Strumpen, S. Amarasinghe, and A. Agarwal, "A 16-issue multiple-program-counter microprocessor with point-to-point scalar operand

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_22

network," in *2003 IEEE International Solid-State Circuits Conference (ISSCC)*, 2003, pp. 170–171.
8. "E16G301 Epiphany 16-core microprocessor," Adapteva Inc., Lexington, MA, Datasheet Rev. 14.03.11.
9. "Parallella-1.x reference manual," Adapteva, Boston Design Solutions, Ant Micro, Rev. 14.09.09.
10. "Epiphany-V: A 1024-core processor 64-bit System-On-Chip" [Online]. Available: http://www.parallella.org/docs/e5_1024core_soc.pdf. [Accessed: 10-Feb-2017]
11. D. Richie, J. Ross, S. Park, and D. Shires, "Threaded MPI Programming Model for the Epiphany RISC Array Processor," Journal of Computational Science, Volume 9, July 2015, pp. 94–100.
12. J. Ross and D. Richie, "Implementing OpenSHMEM for the Adapteva Epiphany RISC array processor," International Conference on Computational Science, ICCS 2016, San Diego, California, USA, 6-8 June 2016
13. J. Ross and D. Richie, "An OpenSHMEM Implementation for the Adapteva Epiphany Coprocessor," OpenSHMEM and Related Technologies. Enhancing OpenSHMEM for Hybrid Environments, vol. 10007, pp. 146-159, Dec. 2016, doi:10.1007/978-3-319-50995-2_10
14. D. Richie and J. Ross, "OpenCL + OpenSHMEM Hybrid Programming Model for the Adapteva Epiphany Architecture," OpenSHMEM and Related Technologies. Enhancing OpenSHMEM for Hybrid Environments, vol. 10007, pp. 181-192, Dec. 2016, doi: 10.1007/978-3-319-50995-2_12
15. D. Richie and J. Ross, "Advances in Run-Time Performance and Interoperability for the Adapteva Epiphany Coprocessor," Procedia Computer Science, vol. 80, Apr. 2016, doi:10.1016/j.procs.2016.05.47

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_22