Heterogeneous integration to simplify many-core architecture simulations *

Raphael Poss
University of Amsterdam
The Netherlands
r.c.poss@uva.nl

Mike Lankamp
University of Amsterdam
The Netherlands
m.lankamp@uva.nl

M. Irfan Uddin
University of Amsterdam
The Netherlands
mirfanud@uva.nl

Jaroslav Sýkora
Institute of Information Theory and Automation
Czech Republic
sykora@utia.cas.cz

Leoš Kafka
Institute of Information Theory and Automation
Czech Republic
leos.kafka@utia.cas.cz

ABSTRACT

The EU Apple-CORE project[1] has explored the design and implementation of novel general-purpose many-core chips featuring hardware microthreading and hardware support for concurrency management. The introduction of the latter in the cores ISA has required simultaneous investigation into compilers and multiple layers of the software stack, including operating systems. The main challenge in such *vertical approaches* is the cost of implementing simultaneously a detailed simulation of new hardware components and a complete system platform suitable to run large software benchmaks. In this paper, we describe our use case and our solutions to this challenge.

Categories and Subject Descriptors

B.4.3 [**Interconnections**]: Interfaces; B.4.4 [**Performance Analysis and Design Aids**]: Simulation; C.0 [**General**]: System architectures; C.1.3 [**Other Architecture Styles**]: Heterogeneous (hybrid) systems; C.1.4 [**Parallel Architectures**]: Distributed Architectures; D.4.7 [**Organization and Design**]: Distributed systems

General Terms

System design, Vertical approach, Hardware / Software codesign

Keywords

hardware multithreading, many-core architecture, system-on-chip design, simulation, system evaluation

*This work is supported by the European Union, under grant FP7-ICT-215216.

[1] `http://www.apple-core.info`

RAPIDO '12, January 23 2012, Paris, France

1. INTRODUCTION

Future architecture research towards many-core architectures will answer fundamentally different problem statements. Instead of starting with sequential programs and wondering how to make them run faster, the starting point is concurrency: how architecture research should be organized when starting with the assumption that *concurrency is the norm*, both in hardware and software. In this context, the research questions are transformed: the focus shifts away from time to results, towards higher throughputs, higher efficiency and predictability.

The adoption of such a different research path should provide a fertile ground for radical innovation. In particular it provides an opportunity to explore simultaneously issues of latency tolerance within core designs and issues of system-level scalability, which are tightly related.

Unfortunately, until now multi-core system-on-chip (SoC) designers have relied on existing core IPs as basic building blocks, and have been reluctant to innovate *simultaneously* with individual core designs and system-level integrations. This causes an increasing divergence between the design of scalable SoCs and the design of high-performance chips.

The Computer Systems Architecture group at the University of Amsterdam attempts to bridge this gap by designing a general-purpose chip architecture which demonstrates both new core designs and best practices from SoC and NoC design. In a preliminary phase, this research was carried out via detailed simulations of individual components based on their potential behavior on silicon, over simple artificial microbenchmarks. The initial results were encouraging [2, 5, 4], and motivated the definition of a publicly funded project, called Apple-CORE, to carry out a more extensive realization. The strategy was comprehensive:

- at the hardware level, design and implement a prototype single-core implementation of the new architecture on an FPGA; and simultaneously implement a software emulation of a full multi-core system using the new processor design;
- at the software level, design and implement new operating systems, language tool chains, and a representative set of benchmarks to evaluate the new hardware architecture, both on single core (FPGA prototype) and multi-core systems (software simulation).

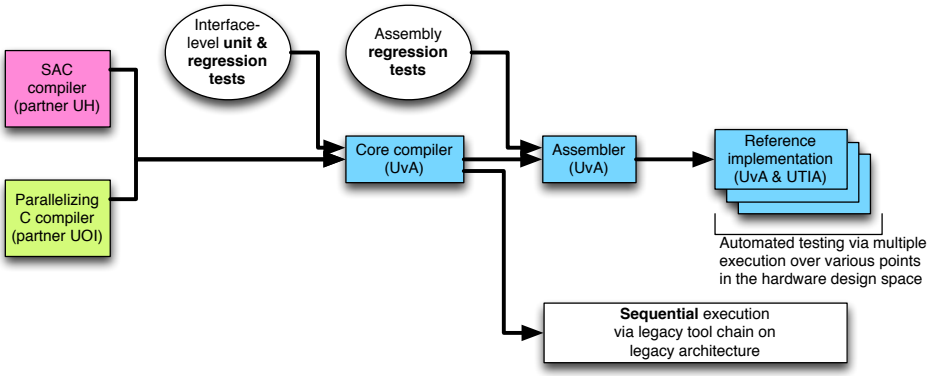


Figure 1: Tool chain interactions for troubleshooting and software validation.

The main challenge of this undertaking was to establish an infrastructure which enables simultaneously the separate validation of hardware and software aspects of the project, and the evaluation of the architecture parameter space. In particular, the need for accurate simulation and the introduction of new experimental core designs conflicted with the need for fully-featured mature operating system and library services in benchmark software.

In this paper, we show how we mastered this challenge by introducing *heterogeneous integration as a means to simplify evaluation*. Heterogeneous integration consists of mixing the new component designs with legacy designs in the architecture model, so that legacy components can support workloads not relevant to evaluation, such as system services. In a simulation environment the legacy workloads can run natively on off-the-shelf hardware. We first detail in section 2 our problem statements and requirements. Then we explain in section 3 our general strategy, and we detail in section 4 our resulting evaluation platforms. We summarize the benefits in section 5 and finally in section 6 we relate our work to other efforts in the field.

2. EVALUATION REQUIREMENTS

To determine the extent of the use of system services by existing code, we have examined the source code of our benchmark applications, as well as commonly used benchmarks from the literature (*e.g.* bzip, yacc, GCC, etc.). We found that most benchmark applications use pre-existing system APIs. For example, file-oriented applications like bzip use POSIX file access directly (`open`, `read`, `write`, `close`) to bypass the buffering offered by the C library. Benchmarks commonly also use system APIs to manipulate file system directory structures and gather precise monitoring data (*e.g.* for self-measurement of execution time), which are defined outside of the standard C library. Moreover, all common C library implementations suitable for porting to a new chip architecture, including the BSD library we eventually selected, assume the existence of a standard POSIX-like system interface to implement the API in charge of memory management, input-output, date & time and interactions with the environment.

In short, a preliminary investigation shows that the availability of services from mature operating systems are a prerequisite to the *evaluation* of a new architecture via commonly accepted benchmarks.

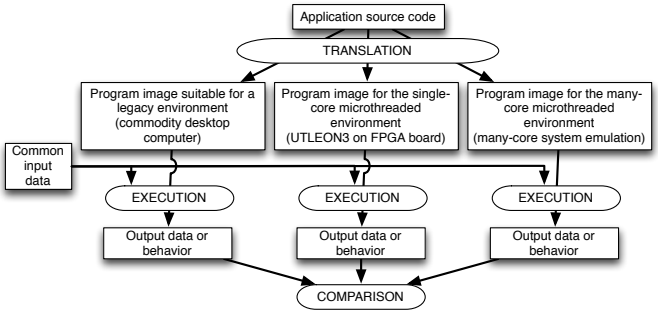


Figure 2: Vertical validation process.

Furthermore, prior to evaluation the infrastructure must be *validated*. If the research project involves both new component designs and new programming methodologies, as in our case, any *baseline* must be established on a separate system. If benchmarks are to be reusable between the baseline and the new system, the program source code must be translatable both to the new architecture and to a legacy environment for comparison. In our setting, we established the infrastructure described in fig. 1. Next to code generation paths from C and SAC [11] benchmarks towards the new chip architecture, we also established a compilation path towards existing legacy architectures for comparison purposes.

From a process perspective, this requires that the same source code and the same input data can be processed via the various target platforms and the final output data / behavior compared against each other, as depicted in fig. 2. While this general validation process may seem at first irrelevant to issues of system integration, it actually implies a fundamental constraint: that *the same system services are available on the legacy reference platform as on the new architecture implementation*. Otherwise, the same application code cannot be used for both, which would defeat the validation strategy.

Two design directions were possible to address this constraint: either the existing system services from the legacy platform are ported for use by applications on the new architecture, which allows to use most existing benchmark appli-

cations without changes; or a different, architecture-specific operating system is developed for the new architecture, and its system services are then emulated on the legacy platform, and the C library implementation is modified to use the new system interfaces, and the existing benchmark applications are modified to use the new interfaces as well.

The latter seemed at first exciting as it could be the starting point for long-lasting partnerships with state-of-the-art operating system research for parallel hardware, such as Barrelfish [18], Helios [17] or fos [23, 24]. However, there is an immense practical obstacle to this latter strategy: not only must we port the new operating system to our platform; the corresponding application-level implementation work, required to make the C library and application programs compatible with the new system, must be compounded with the portability requirement that the code should also run on commodity hardware.

Indeed, when considering the foreground technology produced by state-of-the-art operating research, we noticed that this research is still primarily focused on developing new system principles and much less to provide comprehensive application compatibility layers with legacy systems. We could thus not expect to be able to reuse their technology as-is without a significant investment on our side, which was not budgeted in our effort.

Instead, we considered the exploration of new operating systems to be future work, and we opted for the former strategy instead: port existing system services from a legacy platform to our new environment.

3. SYSTEM DESIGN STRATEGY

As described in the introduction the research focus of the project was a processor chip architecture providing a novel form of hardware multithreading over many cores. Since the first technology produced was an implementation of the architecture (both on FPGA and system emulation in software), a naive approach would have been to directly follow up on this work and port an existing software operating system to run on the new processor. However, we found two obstacles to this, one practical and one conceptual.

The practical issue is that porting an existing operating system to a new hardware platform constitutes a significant undertaking. While we could not find any academic quantification of the effort required, the mere existence of entire businesses dedicated to this task[2] and our own experience suggested that the work required would also exceed our effort budget.

The more fundamental conceptual issue is that all existing reusable operating system codes we could find, from embedded to server platforms and from monolithic kernels to distributed systems, require support for *external control flow preemption* (traps and interrupts) at the pipeline level for scheduling, and either *inter-processor interrupts* or *on-chip programmable packet networks* for inter-processor control and system-level communication. Meanwhile, our core micro-architecture does not support preemption to promote massive hardware scheduled multithreading instead, and does not support inter-process interrupts to favor the use of a custom on-chip network supporting active messages [21] instead. Because of this mismatch, any effort

[2]For example Technologic Systems, AZ, USA, `http://www.embeddedarm.com/services/embedded-os-porting.php`.

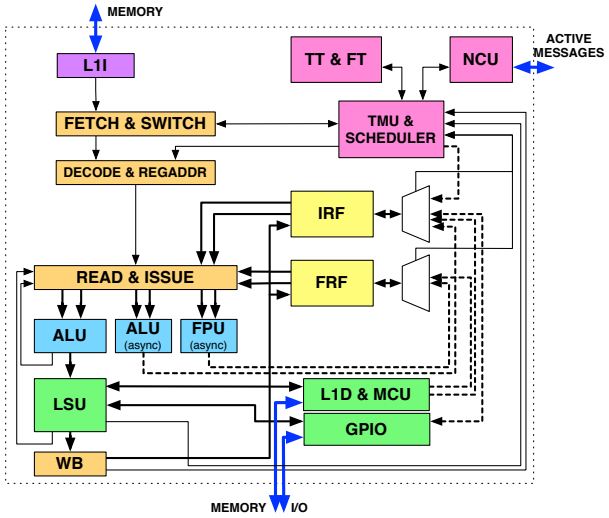


Figure 3: Core micro-architecture.

to port an existing operating system would have required to either introduce new features into the micro-architecture, with the risk that these would impact performance negatively and add complexity to the machine interface, or redesign the operating system components, which would add another significant research expenditure.

Instead, we took a transversal approach: add one or more processors implementing a legacy architecture to the overall system design, and use them to run one or more instances of an existing operating system in software. This is what we call *heterogeneous integration*. Once this is done, it becomes possible to *delegate* any uses of system services from application code running on the new architecture, through the NoC, to the legacy processor. With this approach, the porting effort becomes minimal, since the only new implementation required is a set of "wrapper" API on the microthreaded processors that serve as proxy for a syscall interface implemented on the remote legacy core.

Although we developed this approach independently, we later recognized it in the "multikernel" approach proposed by the designers of Barrelfish [1].

4. REALIZATION

We have realized three different integrations of the proposed new core design, to study its behavior at different levels of abstractions. The FPGA implementation, described in section 4.2, focused on area costs and gate-level interactions inside cores, and thus features only one multithreaded core. The component-level software simulation framework described in section 4.3 was developed to study component-level interactions within and across many cores on chip. The high level software simulation framework described in section 4.4 was realized to make quick and reasonably accurate design decisions at the system level. All three are based on the same single core design, outlined briefly in section 4.1.

4.1 Microthreaded cores

The proposed new core design introduces massive hardware multithreading, dataflow scheduling and hardware sup-

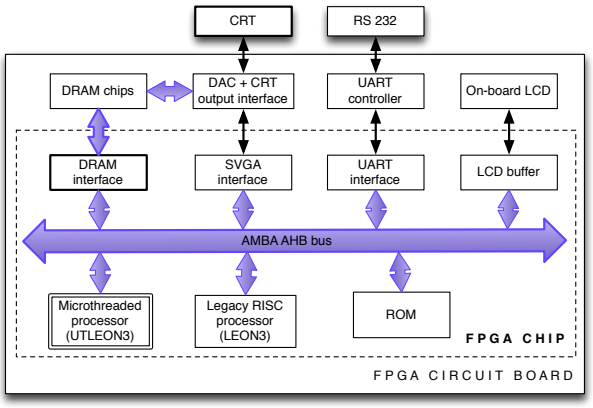


Figure 4: Components on the FPGA platform.

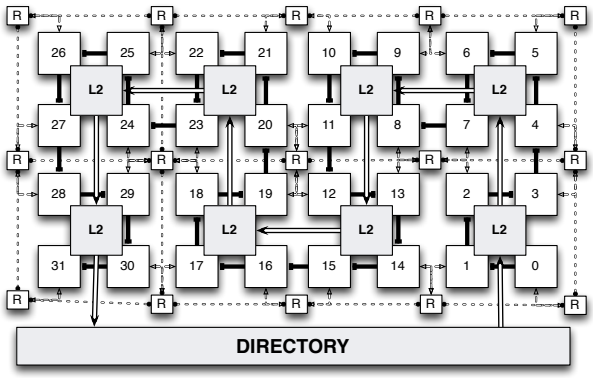


Figure 5: 32-core tile from the Apple-CORE many-core chip design.

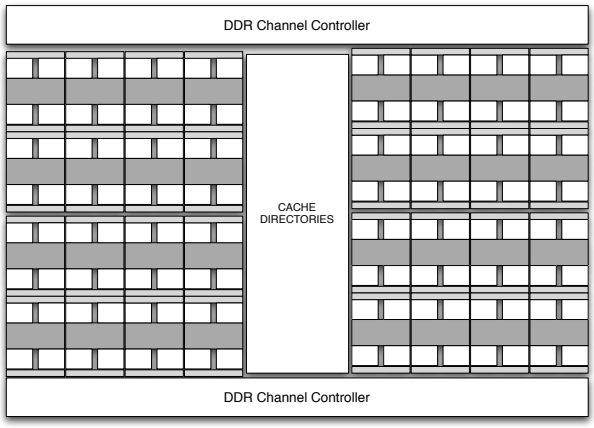


Figure 6: Example many-core chip of 128 microthreaded processors.
Small white tiles represent individual cores. Each dark gray tile represents a L2 cache. Two cores share a FPU.

port for concurrency management as means to tolerate on-chip latencies and optimize throughputs with minimal area requirements. A block diagram is given in fig. 3. A 6-stage, single-issue, in-order multithreaded RISC pipeline implementing either the SPARC or DEC / Alpha ISA, either 32-bit or 64-bit word sizes and shared FPUs is extended with a dataflow scheduler and a dedicated thread management unit. Dataflow scheduling means that all registers are equipped with state bits which cause an instruction to suspend if its operands are not yet computed. Compiler-provided control bits allow to switch preemptively to another thread at the fetch stage for every instruction that may suspend, reducing pipeline bubbles. The fetch unit obtains PCs from a thread active queue in hardware. Register addresses are translated at the decode stage to index different areas of the register files in every thread, possibly overlapping for thread-to-thread communication. Long-latency results like multiplies, floating-point or memory load completions are written asynchronously to the register file and "wake up" waiting threads by placing them back on the active queue. The hardware Thread Management Unit operates on dedicated structures (Family and Thread Tables), and can be controlled remotely via active messages on the NoC. New instructions are introduced in the ISA to create, communicate with and wait on threads.

4.2 FPGA platform

In fig. 4, we illustrate our FPGA-based platform. Using the modular GRLIB [10] component library, we assembled the new processor design UTLEON3 [7, 19], based on the design from section 4.1, together with the original unmodified LEON3 design [9] around an AMBA [8] bus instance, together with a DRAM controller, a system ROM, and various I/O devices (UART, SVGA adapter, on-board LCD).

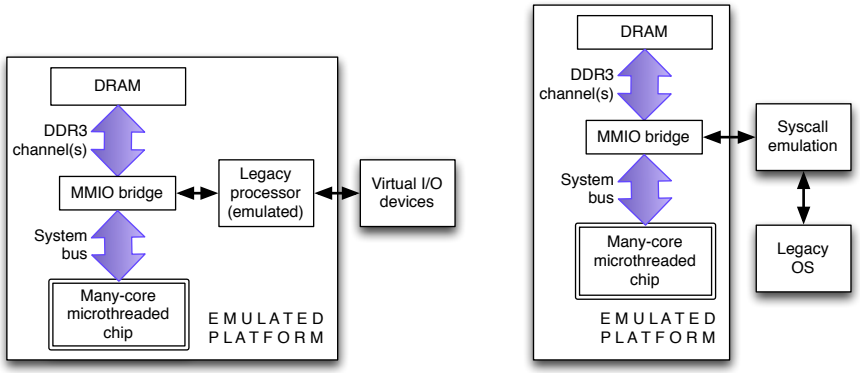
In this design, heterogeneous integration allowed us to use the unmodified LEON3 to run the existing operating system μCLinux [16], a port of Linux for systems without a MMU. Meanwhile, the UTLEON3 processor could run microthreaded code generated with our tool chain. Communication between the two processors occurs via the shared bus: code running on the UTLEON3 can place a request to the system service in memory, then notify the other processor by programming an AMBA interrupt, then wait for the result using a busy loop (the UTLEON3 cannot receive interrupts).

This is the platform that was eventually used to demonstrate the feasibility of a 32-bit configuration of the new architecture and perform single-processor experiments. However, we highlight that this platform was only intended as a proof-of-concept: the shared bus would become a bottleneck if the number of processors was increased keeping the same overall system design, and the busy waiting required by the lack of an asynchronous completion notification mechanism for system services is power-inefficient.

4.3 Many-core system simulation

Our component-level software simulation framework is close in intent and implementation to the M5 simulator [3] from the University of Michigan, to which it is contemporary. Our work differs from M5 in focus: our framework is dedicated to optimizing cycle-accurate simulation of hundreds of cores featuring novel component designs and the accompanying on-chip networks, whereas M5 focuses more on compatibility with legacy systems.

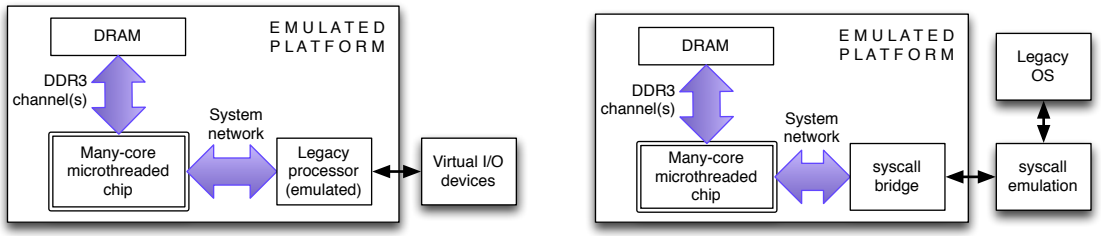
Like M5 however, we strived for maximal configurability.



(a) Legacy processor emulated within the environment.

(b) Bridge interface between host and guest systems.

Figure 8: Shared RAM and memory-mapped I/O interface.



(a) Legacy processor emulated within the environment.

(b) Bridge interface between host and guest systems.

Figure 9: Separate RAM and system bus interfaces.

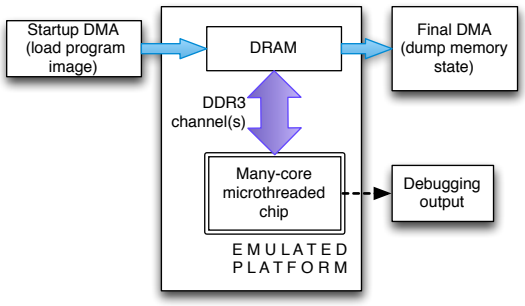


Figure 7: Initial software emulation platform.

Our key parameters include the number and sizes of caches, the cache line sizes, the types of processors, the caching strategies, the system topology, individual component parameters within cores, buffer sizes for hardware FIFOs, the individual frequencies of all components, and the cache coherency protocols. We are able to estimate chip area requirement for memory structures by deriving automatically CACTI [25] parameters from the simulator configuration.

An example simulated 32-core tile is provided in fig. 5. Our typical configurations included arrangements of 64 to 1024 cores following this pattern (*e.g.* fig. 6). The key features of our simulated chip architecture illustrated here are core clusters sharing a common L2 cache; a fast linear network to distribute homogeneous work between adjacent processors, organized in a space-filling curve to maximize locality at multiple scales; a ring-based cache network implementing a COMA protocol [26] with dedicated DDR3 controllers; a low-bandwidth chip-wide mesh network for heterogeneous work distribution.

When we started to consider system integration issues, the overall emulated system design was limited to processor and memory emulation, as in most contemporary architecture research projects. No particular attention was given to the integration of the chip into a *system*: only virtual RAM banks with a simple address translation MMU were emulated outside of the microthreaded chip (connected to the on-chip DDR channel controllers), with a process to load initial data from file to RAM upon startup and dump the RAM contents upon termination (fig. 7).

To support system-level benchmarks as described in section 3, we had to connect the many-core emulation to a legacy operating system somehow. To achieve this we faced two orthogonal design choices, with four possible platform designs:

- *system completeness*: whether to implement a legacy processor fully within the emulated platform (figs. 8a

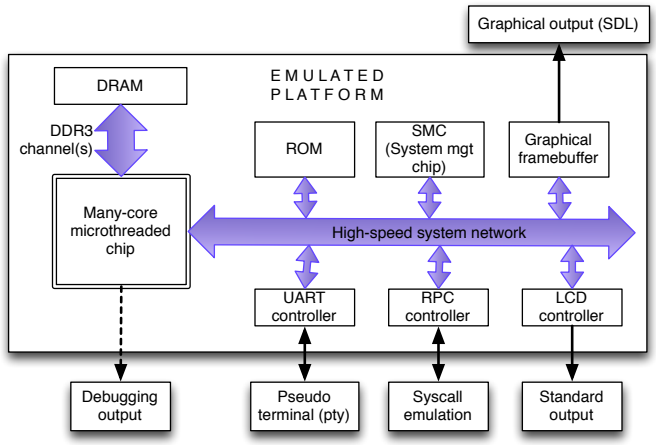


Figure 10: Final software emulation platform.

and 9a), or whether to interface the system services of the *host* platform, where the emulator is running, with the *guest* platform, where the emulated programs are running (figs. 8b and 9b);

- *topology*: whether the interface between the microthreaded processors and the legacy systems hang off the same memory interface (figs. 8a and 8b) or whether to implement a separate dedicated high-speed system network (figs. 9a and 9b).

Our first choice was to select a dedicated system bus for I/O and system services, instead of a memory-mapped interface bridge shared with the DRAM banks. We motivate this choice with two general and two technical arguments.

Our general arguments stem from the observation that the external memory interface is a highly contended resource during computations. To enable higher throughput in streaming applications, it is thus desirable to separate the bandwidth of external input/output from the internal bandwidth of computational data structure accesses using separate physical links. Also, the addition of an interface component between the chip boundary and the DRAM banks would necessarily introduce extra latencies. We consider these arguments appropriately corroborated by contemporary industry trends; for example Intel provides separate DDR channels and a dedicated Quick Path Interconnect interface at the chip boundary in its Core i7 architecture [14].

Our technical arguments are specific to the many-core chip design implemented in the emulation platform. First the provided on-chip COMA protocol does not provide control to processors over the caching strategy, which would prevent cache bypassing, a feature necessary for memory-mapped I/O. Also, the platform also interleaves the address space over the multiple DDR controllers with the granularity of a cache line, which prevents the mapping of a contiguous range of addresses to an external interface. Since this interleaving is not advertised, it would make the development of our wrapper API impractical.

Our next choice was to implement a virtual bridge interface for system services and redirect any request from within the guest platform to the native operating system of the host platform, instead of emulating an entire legacy platform within the emulation environment. This choice was motivated by two practical considerations. First, we could not find off-the-shelf processor emulators suitable for direct inclusion in the emulation platform; without the ability to reuse existing software, we would have had to expend a significant implementation effort. We also considered that the evaluation activities of the Apple-CORE project were primarily focused on processor and memory performance, and thus that the accuracy of I/O performance measurements was not required and did not justify the extra effort.

Our resulting emulation environment, corresponding to the general design in fig. 9b, is illustrated in fig. 10. We implemented a dedicated packet switched system network with a protocol similar to PCI Express and HyperTransport [13], and a bridge interface for system services, labeled "RPC controller" in the diagram. For symmetry with the FPGA platform, we also implemented a ROM chip, a UART interface, a virtual LCD device, a graphical framebuffer and an RTC (not pictured). We implemented the interface between the system network and the processors on the chip using concepts inspired from related work in our research group [12]. Finally, an additional SMC component is in charge of initializing the system upon startup of the emulator, by copying the ROM contents to the chip's memory system via DCA and then triggering activation of the first hardware threads.

The heterogeneous integration here consists of the simultaneous use of the simulation environment to emulate the new core design, and the host system of the simulator to run system services.

4.4 High-level simulation

Next to the low-level platforms described above we have also implemented a high-level simulator to observe the behavior of benchmarks which can consist of billion of instructions executions. This also allows us to investigate mapping strategies of large workloads to different core clusters while developing operating system components.

This environment is illustrated in fig. 11. The architecture model simulates the chip design, whereas the application model executes the workload of simulated cores as native software threads on the host architecture with no detailed simulation of pipeline, instruction issue mechanism or load and store queues.

The mapping function is implemented as automated instrumentation of the native code that generates events relevant to the architecture model. Compared to the cycle-accurate simulator where threads can interleave at every cycle showing a fine-grained interleaving of threads, the high-level simulator evaluates threads based on some time step showing a discrete event simulation of the workload of threads. The time step is computed to have the longest possible step in the execution time between synchronizing events over all executing threads and is explained in [20].

On this platform, system services from the host system are exploited directly. Dedicated software emulators implement the hardware features not present on the host system, such as LCD displays or a system management unit. The heterogeneous integration consists in bypassing the machine model for API calls to library and system services.

5. EVALUATION

During the Apple-CORE project, the new architecture was evaluated across a range of benchmarks, including cryp-

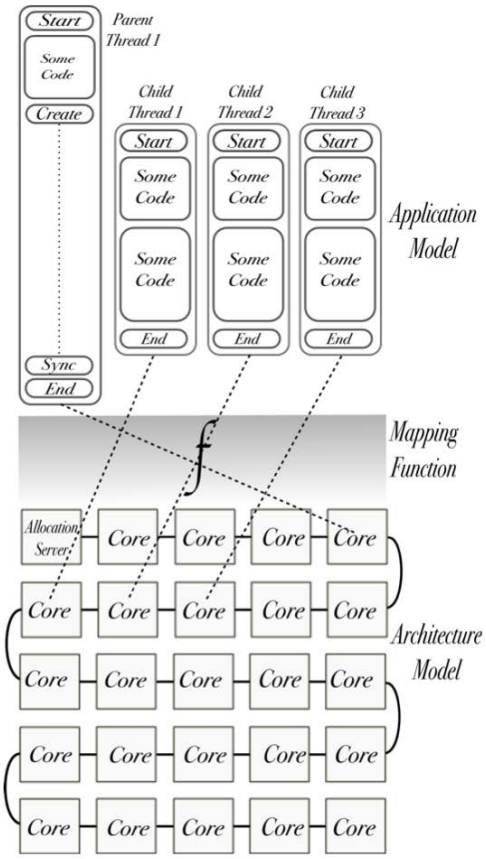


Figure 11: High-level simulation of the microthreaded architecture.

tographic functions, signal processing kernels and interactive applications. The full evaluation results are available publicly[3] and are outside of the scope of this paper. We summarize the performance of our simulators in table 1.

Thanks to our heterogeneous integration strategy, we were able to compile and run legacy code predating the Apple-CORE architecture without changes, and without the burden to integrate entire existing software stacks onto our target new architecture design. Our strategy also allowed us to reduce the efforts dedicated to system integration and software compatibility to 6 man-months, out of a budgeted 350 man-months project.

6. RELATED WORK

Truly our contribution is not as much an improvement to simulation technology, as it is an application of known strategies in system architecture to the field of simulation. The key here is to recognize that we propose to take advantage of a opportunity for spatial heterogeneity in the target (simulated) architecture, to *partially simulate the target system*. This is possible because heterogeneous concurrent applications running on a many-core system, or a SoC, have loosely coupled sub-systems which can be simulated in isolation. Our strategy is to simulate one sub-system (compute

[3] `http://www.apple-core.info/deliverables/`

Simulator	ISA	Speed
FPGA	SPARC	20MIPS, 1 core
Component-level, software	SPARC, Alpha	1-10MIPS shared by all simulated cores
Higher-level, software	any	100MIPS-1GIPS, scalable to multiple host cores

Table 1: Simulator performance

workloads running on the proposed new core design) while letting the function of the rest of the system run on "real" hardware, outside of (and invisible to) the simulation.

This strategy comes in contrast to efforts that achieve *completeness* in the simulated system. We acknowledge the increasing interest to *simulate entire, large, heterogeneous systems* efficiently. RAMP [22] is a recent example of this: the task to implement simulators for individual parts is distributed across multiple organizations, and the resulting individual simulators are run in parallel and cross-synchronized to obtain a single complete simulated system.

We denounce completeness as an inadequate requirement on simulators to demonstrate new component designs. Take as example the Cray XMT [15], which inspired us. In the XMT, the focus of attention of the system designer is the *compute node* with its MTA processor. This runs a lightweight, non-standard microkernel to support local application execution. Any work not directly relevant to the performance workload, like console interactions with users, is delegated to a *service node* running Linux on and AMD Opteron. While it is theoretically possible to port a large operating system like Linux to the MTA, or extend the MTA to run a legacy system, this would *divert design efforts* away from the overall optimization of the MTA to performance workloads. Arguably, a simulation of the Cray XMT would not need to include a simulation of the service nodes to be tremendously useful to the system designers.

More generally, our claim is that research projects would be *simplified* by using heterogeneous integration and reusing existing hardware to avoid the burden of porting entire software ecosystems to new component designs. To our knowledge, no related work has yet exploited this opportunity as a deliberate strategy choice.

Finally, note that our proposed strategy is mostly orthogonal to simulation efficiency and accuracy. In particular, "heterogeneous integration" is not related to *heterogeneous simulation*, where the type of simulation used for a component depends on how much information is desired. An example of the latter is FAST [6], which uses functional models for the overall system behavior, but replays behaviors to be scrutinized onto a detailed timing model on FPGA. This can be used in combination with heterogeneous integration, and we are considering this approach for future work.

7. CONCLUSION

In this paper, we have explained our methodology to resolve the apparent conflict between the architect's requirement of accurate simulations of new component designs and the software engineer's requirements for system completeness. This relies on the exploitation of "companion" legacy cores support system services which cannot be directly implemented on the new hardware components.

While traditional SoC designers acknowledge heterogeneity as an unavoidable externality, we have deliberately introduced heterogeneity as a means to externalize the implementation of legacy system services from the architecture research. This has enabled us to explore radically new principles of processor design while reducing the cost of supporting complex benchmarks from existing evaluation suites.

8. REFERENCES

[1] A. Baumann, P. Barham, P.-E. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, and A. Singhania. The multikernel: a new os architecture for scalable multicore systems. In *Proc. ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP'09, pages 29–44, New York, NY, USA, 2009. ACM.

[2] I. Bell, N. Hasasneh, and C. Jesshope. Supporting microthread scheduling and synchronisation in CMPs. *International Journal of Parallel Programming*, 34:343–381, 2006.

[3] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt. The M5 simulator: Modeling networked systems. *IEEE Micro*, 26:52–60, 2006.

[4] K. Bousias, L. Guang, C. Jesshope, and M. Lankamp. Implementation and evaluation of a microthread architecture. *Journal of Systems Architecture*, 55(3):149–161, 2008.

[5] K. Bousias, N. Hasasneh, and C. Jesshope. Instruction level parallelism through microthreading – a scalable approach to chip multiprocessors. *The Computer Journal*, 49(2):211–233, March 2006.

[6] D. Chiou, D. Sunwoo, J. Kim, N. A. Patil, W. Reinhart, D. E. Johnson, J. Keefe, and H. Angepat. FPGA-accelerated simulation technologies (FAST): Fast, full-system, cycle-accurate simulators. In *Proc 40th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 40, pages 249–261, Washington, DC, USA, 2007. IEEE Computer Society.

[7] M. Danek, L. Kafka, L. Kohout, and J. Sykora. Instruction set extensions for multi-threading in LEON3. In Z. K. et al., editor, *Proc. 13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS'2010)*, pages 237–242. IEEE, 2010.

[8] D. Flynn. AMBA: enabling reusable on-chip designs. *IEEE Micro*, 17(4):20–27, jul/aug 1997.

[9] J. Gaisler. A portable and fault-tolerant microprocessor based on the SPARC v8 architecture. In *Proc. International Conference on Dependable Systems and Networks (DSN'02)*, pages 409–415. IEEE, 2002.

[10] J. Gaisler, E. Catovic, and S. Habinc. *GRLIB IP Library User's Manual*. Gaisler Research, 2007.

[11] C. Grelck and S.-B. Scholz. SAC: a functional array language for efficient multi-threaded execution. *International Journal of Parallel Programming*, 34(4):383–427, Aug 2006.

[12] M. A. Hicks, M. W. van Tol, and C. R. Jesshope. Towards Scalable I/O on a Many-core Architecture. In *International Conference on Embedded Computer Systems: Architectures, MOdeling and Simulation (SAMOS)*, pages 341–348. IEEE, July 2010.

[13] HyperTransport Consortium. The future of high-performance computing: Direct low-latency peripheral-to-CPU connection, September 2005.

[14] Intel Corporation. Intel® Core™ i7-900 desktop processor extreme edition series and Intel® Core™ i7-900 desktop processor series Datasheet, Volume 1, document # 320834-00, February 2010.

[15] P. Konecny. Introducing the Cray XMT. In *Proc. Cray User Group meeting (CUG'07)*, 411 First Avenue South, Seattle, WA 9810, USA, May 2007. Cray Inc.

[16] D. McCullough. uClinux for Linux programmers. *Linux Journal*, (123):34–36,38, July 2004.

[17] E. B. Nightingale, O. Hodson, R. McIlroy, C. Hawblitzel, and G. Hunt. Helios: heterogeneous multiprocessing with satellite kernels. In *Proc. ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, pages 221–234, New York, NY, USA, 2009. ACM.

[18] A. Schüpbach, S. Peter, A. Baumann, T. Roscoe, P. Barham, T. Harris, and R. Isaacs. Embracing diversity in the barrelfish manycore operating system. In *Proceedings of the Workshop on Managed Many-Core Systems*. ACM, June 2008.

[19] J. Sykora, L. Kafka, M. Danek, and L. Kohout. Analysis of execution efficiency in the microthreaded processor UTLEON3. volume 6566 of *Lecture Notes in Computer Science*, pages 110–121. Springer, 2011.

[20] M. I. Uddin, M. W. van Tol, and C. R. Jesshope. High level simulation of SVP many-core systems. *Parallel Processing Letters*, 21(4):413–438, December 2011.

[21] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser. Active messages: a mechanism for integrated communication and computation. In *ISCA '92: Proc. 19th annual International Symposium on Computer Architecture*, pages 256–266, New York, NY, USA, 1992. ACM.

[22] J. Wawrzynek, D. Patterson, M. Oskin, S.-L. Lu, C. Kozyrakis, J. Hoe, D. Chiou, and K. Asanovic. RAMP: Research accelerator for multiple processors. *IEEE Micro*, 27(2):46–57, March-April 2007.

[23] D. Wentzlaff and A. Agarwal. Factored operating systems (fos): the case for a scalable operating system for multicores. *SIGOPS Oper. Syst. Rev.*, 43:76–85, April 2009.

[24] D. Wentzlaff, C. Gruenwald, N. Beckmann, K. Modzelewski, A. Belay, L. Youseff, J. Miller, and A. Agarwal. A unified operating system for clouds and manycore: fos. Technical Report MIT-CSAIL-TR-2009-059, Computer Science and Artificial Intelligence Lab, MIT, November 2009.

[25] S. Wilton and N. Jouppi. Cacti: an enhanced cache access and cycle time model. *Solid-State Circuits, IEEE Journal of*, 31(5):677–688, may 1996.

[26] L. Zhang and C. R. Jesshope. On-Chip COMA Cache-Coherence Protocol for Microgrids of Microthreaded Cores. In Bouge and et al., editors, *Euro-Par Workshops*, volume 4854 of *LNCS*, pages 38–48. Springer, 2007.