A Holistic Dataflow-Inspired System Design

Stéphane Zuckerman, Haitao Wei
Guang R. Gao
Dept. of Electrical. and Computer. Eng.
University of Delaware
Newark DE, 19716, USA
Email: {szuckerm, hwei, ggao}@udel.edu

Howard Wong
Jean-Luc Gaudiot
The Henry Samueli School of Engineering
University of California, Irvine
Irvine, CA 92697-2625, USA
Email: {hwwong, gaudiot}@uci.edu

Ahmed Louri
University of Arizona
1230 E. Speedway Blvd.
P.O. Box 210104
Tucson, AZ 85721-0104, USA
Email: louri@ece.arizona.edu

***Abstract*—Computer systems have undergone a fundamental transformation recently, from single-core processors to devices with increasingly higher core counts within a single chip. The semi-conductor industry now faces the infamous power and utilization walls. To meet these challenges, heterogeneity in design, both at the architecture and technology levels, will be the prevailing approach for energy efficient computing as specialized cores, accelerators, *etc.*, can eliminate the energy overheads of general-purpose homogeneous cores. However, with future technological challenges pointing in the direction of on-chip heterogeneity, and because of the traditional difficulty of parallel programming, it becomes imperative to produce new system software stacks that can take advantage of the heterogeneous hardware. As a case in point, the core count per chip continues to increase dramatically while the available on-chip memory per core is only getting marginally bigger. Thus, data locality, already a must-have in high-performance computing, will become even more critical as memory technology progresses. In turn, this makes it crucial that new execution models be developed to better exploit the trends of future heterogeneous computing in many-core chips. To solve these issues, we propose a cross-cutting cross-layer approach to address the challenges posed by future heterogeneous many-core chips.**

***Index Terms*—Dataflow; Codelets; Streaming; Heterogeneous architecture**

I. Introduction

Computer systems have undergone a fundamental transformation from the single processor devices of the early 2000s to today's (and future) devices with increasingly higher core counts connected within a single chip. Parallelism is ubiquitous and found at many levels of the entire hardware-software stack. However, because of the physical limits faced by the semiconductor industry, and because our ability to achieve predictable performance improvements through traditional processor techniques has ended, there is a concrete need for a comprehensive rethinking of our approach to algorithm design, programming models, architecture design, and system software design if we are to successfully exploit parallelism and deliver scalability. In other words, we need to bring breakthrough technology to face a new era in parallel computing.

We take the position that future extreme-scale systems must be rethought from the ground-up, taking into consideration the program execution model, the underlying hardware architecture, and the system software which bridges both. In this context, we consider applications whose computations can be naturally expressed as *streams*. To meet our objectives of efficiently exploiting parallelism and delivering scalability in stream processing, a number of challenges must be overcome, including exploiting multi-grain parallelism, heterogeneous hardware and workloads, and developing efficient resource management mechanisms. There are three main problems to solve. First, We need to exploit both coarse-grain and fine-grain parallelism. Second, we need to handle the problem of heterogeneity of the computation load, and heterogeneity of the computation type. Finally, We need to maximize locality and minimize data movement. They have an impact on both performance and energy efficiency: (a) Some data are continuously streaming through the data channels in the dataflow graph; and (b) Some data are not streaming; they should be placed in the shared memory hierarchy in order to exploit locality and minimize data movement. To address these challenges, several inter-related research aspects must be explored in a new framework, SPARTA (Stream-based Processor and Run-Time Architecture), and which includes: (a) a novel program execution model (PXM), (b) a novel architecture model, and (c) the resulting system software stack.

The remainder of this paper is as follows: Section II describes our proposed novel program execution model (PXM) for streaming applications for extreme-scale systems; Section III describes our targeted architecture; Section IV describes the system software required to bridge the PXM and its underlying architecture; Section V describes work related to our research; and we conclude in Section VI.

II. The SPARTA Program Execution Model

Figure 1 presents an overview of a SPARTA system, from the expression of a program to its execution on the type of chip we envision in order to run highly efficient stream programs. The remainder of this section decribes the PXM, system architecture, and system software we believe is required to efficiently run stream programs on future extreme-scale machines.

A. Extending the Codelet Model to Streams

Overall, we need to smoothly and optimally expose parallelism by leveraging the power of asynchrony and developing strategies to exploit its use in the Codelet Model into a *Streaming Codelet Model* (SCM). Streaming Codelets are codelets [1] with some key additional properties: An interface must describe buffer sizes and latencies to ensure steady state scheduling preferences in terms of resources. This is required to implement software pipelining. The (Streaming) Codelet Model favors "determinacy-by-default." However, it also allows for explicitly non-determinate behavior to deal with inherently non-determinate situations, *e.g.*, transaction processing. In addition, an interface must be defined to express interconnections between codelets, since some may be mapped to different hardware parts in the system.

As a result, this PXM must help hide bulk data transfer latency and scheduling channels, and reduce the complexity

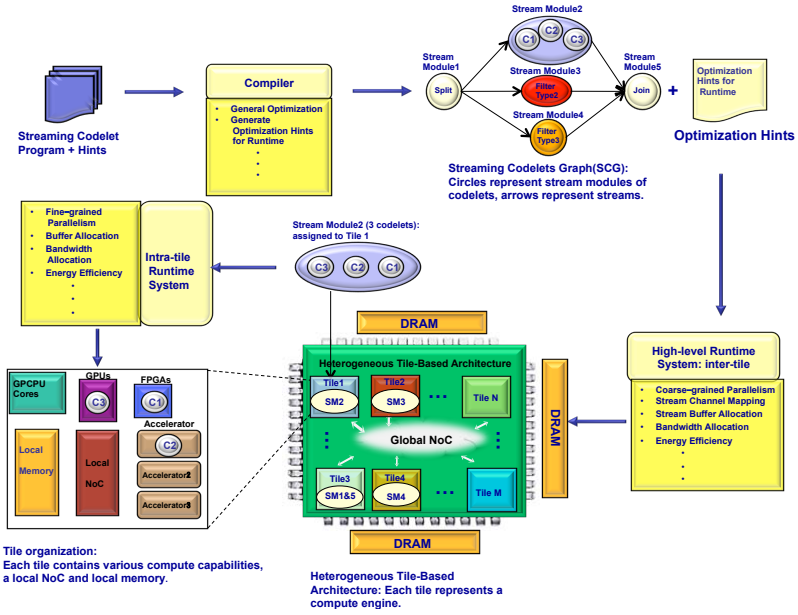

Fig. 1. An overview of the SPARTA system. A program is written along with its compiler hints. The compiler generates the corresponding streaming codelet graph, along with hints and goals for the runtime system (RTS). The graph is compiled into machine code. The RTS selects streaming codelet sub-graphs to be mapped to specific hardware tiles. Within each tile, the RTS chooses how to allocate the streaming codelets that are contained in the module.

of the job scheduler. To this end, the appropriate abstractions for hardware models must be defined, in order to simplify the scheduler and runtime design. It results that other features must augment the initial Codelet Model: a SCM program is partitioned into modules which are connected by channels. A module can be seen as a group of codelets connected by *intra-module* stream channels within a module. Modules are themselves connected through *inter-module* stream channels. Each module contains at least one streaming codelet. A stream channel is modeled as an abstract FIFO queue. Each stream module is an autonomous computation unit which consumes data at a given rate from the input channel and produces data at a given rate to its output channel. At execution time, each stream computation module is ready to run only if there are enough data items in the input channels and enough buffer space in the output channels.

Both the modules and the streaming codelets they contain are event-driven, with the arrival of data as the primary event to satisfy, thereby potentially exploiting both coarse-grained and fine-grained parallelism. At the module level, pipelining and task parallelism can be exploited between stream computation modules. The channels are directly tied to the consuming codelets that require them, and thus modules are only a logical grouping of streaming codelets. Further, each module may also contain any degree of parallelism. Streaming codelets belonging to the same module can be mapped to different compute engines within a given cluster. The need to pass data between codelets may induce unreasonably long latencies, thus requiring the use of intra-module stream channels to specify buffer sizes, *etc.* On the other hand, within a given portion of the machine, latencies will be essentially non-existent, thus only requiring that codelets signal the availability of new data, much like the original codelet model proposes.

B. The Streaming Codelet Abstract Machine Model

To make our PXM reflect the pervasive heterogeneity we envision in future extreme-scale architectures, we propose a two-level abstract machine model (AMM). The original Codelet Model relies on an AMM which is hierarchical and distributed, and provides two types of engines: the *computing units* (CUs, which perform the actual work) and the *synchronization units* (SUs, which ensure the correct scheduling and resource allocation across the machine). This AMM must be extended, adding to it concepts such as Memory/Compute Decoupled architectures [2]. The high-level layer of our proposed AMM is visible to the application programmer, and consists of clusters of (heterogeneous) compute engines on which streaming codelets are to be mapped. Depending on the state of the AMM at run time (*e.g.*, if there are faulty components, highly-contended ones, *etc.*), the available parallelism will vary. Additional properties must be added to the initial codelet AMM to reflect these states. Specific SUs are dedicated to the distribution of work among the clusters. The second layer of the AMM is a low-level abstract machine model, visible to both the compiler and the runtime system. It must identify the type of capabilities embedded in clusters. This allows it to generate the adequate code variants. The compute engines contained in a cluster range from specific functions provided by a low-level component of the cluster to fully general purpose CUs. They also feature SUs to map streaming codelets to portions of a module to the available CUs.

III. The SPARTA System Architecture

The system architecture implements an abstract machine on the hardware for the PXM. While architectural trends tend to favor multi/many-core chips as a commodity architecture, their current design does not scale well due to severe power limitations, which may require that parts of the chip may have to remain inactive [3]. This results in the utilization wall [4] which limits the fraction of a chip that can be used. In order to improve performance/power efficiency by an order of magnitude, we propose a cross-layer *cooperatively designed system* that includes a structure consisting of a heterogeneous set of adaptive computational resources, a hybrid memory

organization, a high-bandwidth, low-power and scalable reconfigurable interconnect, and compiler and operating system directives to harness the promises of heterogeneity.

A. Heterogeneous Computing Engines for Streaming Codelets

To achieve end-to-end energy efficiency, we must optimize both on- and off-chip memory systems that feed the compute engines and the communication structure that transports the streams. These two elements introduce energy overhead orders of magnitude higher than the overhead of the compute engines. To lay the foundation for further research, we propose a tile-based architecture with a wide range of customizable computing elements, from coarse-grain cores to fine-grain accelerators and field-programmable circuit fabrics, and scalable reconfigurable high-performance interconnects. At the highest level, different tiles are interconnected by a reconfigurable global NoC (GNoC). The tiles represent the compute engines. Each consists of customizable accelerators, general-purpose cores, GPUs, a tile shared-memory (TSM), a Bulk Memory Transfer Unit (BMTU), and an intra-tile local NoC (LNoC). The combination of tiles, memory structure, and a hierarchical NoC will provide adequate heterogeneous hardware support for the proposed Streaming Codelet Model.

B. Exploiting Dataflow Concepts with Heterogeneous Memory

In order to fetch an entire codelet and its associated data all at once, we need bulk memory access as opposed to word-at-a-time access throughout the memory hierarchy, hence requiring a change in the ISA. We therefore propose TSM as a scratchpad memory (to store codelet contexts) and BMTU to speed up bulk transfer. The BMTU will act much like a DMA engine, albeit with a much simpler implementation. Other memory models will be explored. For instance, split-phase operations are important primitives of message passing operations: they allow a request to be asynchronously placed by a consumer in the memory. Should the data element in question already be present where it was expected, the memory can simply satisfy the request. Otherwise, the request is held in the memory itself. Eventually, the codelet producing the element writes it into the memory system. Upon recognizing the existing pending request(s), the memory system forwards the data to (all) pending consumer(s), thus enabling *producer-consumer parallelism*. Hence designing the right memory system for streaming is crucial, and could be based on efficient implementations of I-Structures [5]. The presence of a TSM and LNoC enforces locality of computation within the tile needed for the proposed streaming codelet model. On the technology side, technology scaling of SRAM and DRAM is increasingly constrained by fundamental technology limits to somewhat alleviate the problems due to the power and memory walls Emerging non-volatile memory (NVM) technologies have combined the speed of SRAM, the density of DRAM, and the non-volatility of Flash memory [6]. Thus, there is a need to design a memory hierarchy using different technologies so that it is more power-efficient and provide higher performance than a hierarchy based on conventional technologies.

C. Binding It All with Reconfigurable Interconnect Fabric

On top of the above, the system architecture must provide low-latency, high-bandwidth, and reconfigurable interconnects for data sharing within and between the tiles: On- and off-chip communications are projected to become a major bottleneck in terms of performance, energy consumption, and reliability when hundreds of heterogeneous compute engines are integrated [7]. Compiler analysis can be used to (partly) address communications optimization in message passing and shared memory systems [8]. Through hardware sensing, traffic loads and communication behavior can be dynamically monitored at run time, as well as network resource usage, power consumption, and resource health. Such information can be used to devise dynamic reconfiguration algorithms to adapt the hardware to the current application needs. thus improving performance and energy efficiency. These algorithms must also provide robustness and fault-tolerance by avoiding faulty channels and faulty routers. On the architecture side, the reconfigurable NoC should be augmented with the hardware capability of establishing fast and energy efficient channels between frequently communicating sources and destinations [9]. Such architecture must be built upon energy-efficient and fault-tolerant NoC architectures for multicore systems, which includes exploring the use of novel interconnect technologies, in particular 3-D stacking, optical interconnects, and RF interconnects [10].

IV. The SPARTA System Software

The SPARTA runtime system (RTS) should manage parallelism, memory, energy usage, network traffic, *etc.* As Figure 1 shows, the SPARTA RTS schedules the stream modules of a given application to the tiles and creates stream channels for the communications between the modules. Then, the RTS needs to schedule the codelets to the CUs in order to exploit the fine-grained parallelism. The outcome in terms of load balance, throughput, and communication contention must also be considered. The allocation of stream modules to the appropriate tile in the system must be based upon a set of compiler "hints:" Indeed, the compiler should be aware of the requirements of each task before it can allocate it to a tile with the proper computational capabilities. As a results important program patterns such as data parallelism, pipelining, *etc.*, must be considered, along with computation, memory allocation and communication optimization. Based on this information, the runtime code will perform dynamic resource management and optimization to ensure that processing and memory resources are not left idle when there are tasks/codelets available for execution. We believe that extremely significant gains in performance and energy savings are possible by building the most critical features of the runtime API into the hardware of a massively parallel heterogeneous chip.

A. Establishing an Efficient Communication Scheme

We must address the important challenge of runtime allocation and utilization of bandwidth provided by the hierarchical interconnection networks in a SPARTA machine: both within a chip and across compute nodes. Resolving this adds two more challenges: channel assignment to support stream communication and bandwidth allocation within a (potentially shared) channel. A balance must thus be found between the hardware and system software in the SPARTA system. Further, we must be capable of identifying the parameters of computation and communication requirements: The compiler should be able to "direct" the various pieces of codes to the computationally most efficient engine. This would result in better load balancing, code placement according to detected or otherwise statically set communication patterns, *etc.*

B. The Need for Fast and Efficient Queues

As we explained earlier, a trade-off must be found between the system software and the hardware. Among critical components that are part of it are queues, which are a fundamental data structure to support scheduling in parallel environments. Most likely, software queues relying solely on `compare-and-swap` instructions won't be scalable enough given the expected amount of tasks that will be generated in such systems: Hardware queues may be the only efficient way to handle such massive parallelism.

V. Related Work

Program execution models centered on streams have been studied by many researchers and have been an active field of research for the past 30 years [11], [12]. The most relevant early work on streams is the dataflow execution model pioneered by Dennis [13], and the synchronous dataflow model of Lee [14]. Other work includes software pipelining for streaming programs [15]. However, these models do not address the main problems we are addressing in this proposal, such as targeting a highly heterogeneous and hierarchical chip.

Heterogeneous multicore architectures and specialization of the functions are being heralded as the solutions to the dark silicon and utilization wall issues and provide the most compelling architectural path to mitigate these problems, while providing continued performance scaling [16], [17]. Heterogeneous architectures have been studied by a number of researchers but many of these efforts are only targeted at isolated dimensions of heterogeneity, either the cores, the memory, or the interconnects in isolation. We believe that heterogeneity can be harnessed at a much larger cross-cutting scope, spanning heterogeneous cores, heterogeneous hybrid memory hierarchies of different technologies, heterogeneous interconnect technologies and architectures, and software.

Several compiler and runtime techniques have been proposed in the past to optimize stream programs on specific architectures, *e.g.,* on special stream processors [18]. Further, several techniques were proposed to map streaming languages to multithreaded processors [19], [20]. However, none of these solutions target a highly heterogeneous system, nor were they designed with additional constraints such as power and energy efficiency in mind. As a consequence, we developed a strategy to solve the thorny problem of trading off performance *vs.* power and energy efficiency through the SPARTA runtime system.

VI. Conclusion and Future Work

This paper presents the SPARTA framework, and takes the position that extreme-scale systems require a complete overhaul at the compute node level in order to handle extreme parallelism. This in turn requires that novel dataflow-based programe execution models be specified, along with heterogeneous hardware. A trade-off must be found between software and hardware, and the resulting co-design effort will be bridged by some efficient system software, combining compiler and runtime systems.

VII. Acknowledgements

This material is based upon work supported by the National Science Foundation, under awards CCF-1439142, CCF-1439165, and CCF-1439097.

References

[1] S. Zuckerman, J. Suetterlein, R. Knauerhase, and G. R. Gao, "Using a "Codelet" program execution model for exascale machines," in *In Proceedings of the 1st International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era, EXADAPT '11*, 2011, pp. 64–69.

[2] W. Ro and J.-L. Gaudiot, "A complexity-effective microprocessor design with decoupled dispatch queues and prefetching parallel computing," *Parallel Computing*, vol. 35, no. 5, pp. 255–268, May 2009.

[3] S. Borkar, "Thousand Core Chips: A Technology Perspective," in *Proceedings of the 44th Annual Design Automation Conference*, ser. DAC '07. New York, NY, USA: ACM, 2007, pp. 746–749. [Online]. Available: http://doi.acm.org/10.1145/1278480.1278667

[4] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson, and M. B. Taylor, "Conservation Cores: Reducing the Energy of Mature Computations," *SIGPLAN Not.*, vol. 45, no. 3, pp. 205–218, Mar. 2010. [Online]. Available: http://doi.acm.org/10.1145/1735971.1736044

[5] W.-Y. Lin, J.-L. Gaudiot, J. Amaral, and G. R. Gao, "Performance Analysis of the I-Structure Software Cache on Multi-Threading Systems," in *Proceedings of the 19th IEEE International Performance, Computing and Communication Conference, IPCCC2000*, Phoenix, Arizona, February 2000, pp. 83–89.

[6] S. A. Wolf, J. Lu, M. Stan, E. Chen, and D. Treger, "The Promise of Nanomagnetics and Spintronics for Future Logic and Universal Memory," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2155–2168, Dec 2010.

[7] R. Weiss, "NSF Workshop on Emerging Models and Technologies in Computing: Bio-Inspired Computing and the Biology and Computer Science Interface." Ph.D. dissertation, United States Naval Academy, 2008.

[8] S. Shao, A. Jones, and R. Melhem, "A compiler-based communication analysis approach for multiprocessor systems," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, April 2006, pp. 10 pp.–.

[9] A. K. Kodi, A. Louri, and J. M. Wang, "Design of Energy-Efficient Channel Buffers with Router Bypassing for Network-on-Chips (NoCs)," in *ISQED*, 2009, pp. 826–832.

[10] D. DiTomaso, A. Kodi, and A. Louri, "QORE: A Fault-Tolerant Network-on-Chip Architecture with Power-Efficient Quad Function Channel (QFC) Buffers," in *Accepted to appear in 20th IEEE International Symposium on High-Performance Computer Architecture (HPCA), Orlando, FL, February 15-19, 2014*, February 15-19 2014.

[11] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with CUDA," *Queue*, vol. 6, no. 2, pp. 40–53, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1365490.1365500

[12] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, and K. Skadron, "A performance study of general-purpose applications on graphics processors using CUDA," *Journal of Parallel and Distributed Computing*, vol. 68, no. 10, pp. 1370–1380, Oct. 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731508000932

[13] J. B. Dennis, "First version of a data flow procedure language," in *Programming Symposium, Proceedings Colloque sur la Programmation*. London, UK: Springer-Verlag, 1974, pp. 362–376.

[14] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Trans. Computers*, vol. 36, no. 1, pp. 24–35, 1987.

[15] R. Govindarajan, G. Gao, and P. Desai, "Minimizing Memory Requirements in Rate-optimal Schedules," in *In ASAP '94: Proceedings of the 1994 International Conference on Application Specific Array Processors*, 1994, pp. 75–86.

[16] M. Suleman, O. Mutlu, J. Joao, K. Khubaib, and Y. Patt, "Data Marshaling for Multicore Systems," *Micro, IEEE*, vol. 31, no. 1, pp. 56–64, Jan 2011.

[17] H. Najaf-abadi and E. Rotenberg, "Architectural Contesting," in *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, Feb 2009, pp. 189–200.

[18] U. J. Kapasi, P. Mattson, W. J. Dally, J. D. Owens, and B. Towles, "Stream Scheduling," in *in Proceedings of the 3rd Workshop on Media and Streaming Processors*, 2001, pp. 82–92.

[19] P. Mattson, W. J. Dally, S. Rixner, U. J. Kapasi, and J. D. Owens, "Communication scheduling," *SIGARCH Comput. Archit. News*, vol. 28, no. 5, pp. 82–92, Nov. 2000. [Online]. Available: http://doi.acm.org/10.1145/378995.379005

[20] M. Kudlur and S. Mahlke, "Orchestrating the execution of stream programs on multicore platforms," *SIGPLAN Not.*, vol. 43, no. 6, pp. 114–124, Jun. 2008. [Online]. Available: http://doi.acm.org/10.1145/1379022.1375596