2014 Fourth Workshop on Data-Flow Execution Models for Extreme Scale Computing

Toward a Self-Aware Codelet Execution Model

Stéphane Zuckerman, Aaron Landwehr, Kelly Livingston, and Guang Gao
Department of Electrical and Computer Engineering
University of Delaware
Newark DE, 19716, USA
Email: {szuckerm@, aron@, kelly@, ggao@capsl.}udel.edu

***Abstract*—Future extreme-scale supercomputers will feature arrays of general-purpose and specialized many-core processors, totaling thousands of cores on a single chip. In general, many-core chips will most likely resemble a "hierarchical and distributed system on chip." It is expected that such systems will be hard to exploit not only for performance, but will also need to deal with reliability issues, as well as power and energy issues.**

The Codelet Model is a fine-grain dataflow-inspired and event-driven program execution model which was designed to run parallel programs on a combination of such many-core chips into a supercomputer Meanwhile, some on-going work is attempting to take into account user goals as well as resource usage and make the system "self-aware:" By using introspective means, this kind of research tries to have the system software modify the state of the overall system at run-time to satisfy the user goals. It is very likely that future extreme-scale systems will be in constant demand of different kinds of resources, may they be processing elements (general purpose or otherwise), bandwidth, power budget, *etc*.

This paper takes the position that a potential solution to solve the resource management issue at this scale is a hierarchical and distributed self-aware system leveraging the fine-grain event-driven codelet threading model.

***Index Terms*—Dataflow; Codelets; Self-Awareness; Resource Management**

I. Introduction

Today's supercomputers reach a peak performance in the range or the peta-FLOPS. While most parallel applications still require the programmer to use some combination of MPI (for inter-node communication, with specialized inter-connection networks) and on-node shared memory (most of the time, using some OpenMP-like environment), a new trend is to use accelerators (such as GP-GPUs, Xeon Phi, *etc.*) when there is a lot of parallelism exploitable by such devices. This in turn forces the application programmer to learn and combine several programming models as well as program execution models. Overall, these systems do not feature revolutionary intra-node mechanisms, whether one considers the hardware or the software, save for the occasional use of accelerators.

There are plenty of execution models dealing with current supercomputers, ranging from Charm++ [1] to a combination of MPI and OpenMP, some of which even explicitly combine the two in a unified framework [2], [3].

It is not expected that future extreme-scale (exascale and beyond) systems will significantly change their way of communicating across nodes (*i.e.*, Infiniband-like interconnects are expected to make incremental progress and still be in use). However, following the advent of the aforementioned accelerators, intra-node technologies will most likely significantly change, featuring more heterogeneous hardware at the node or even chip level, giving way to new design paths for both hardware and software. Indeed, future extreme-scale supercomputers will very likely feature arrays of general-purpose and specialized many-core processors, totaling thousands of cores on a single chip with deeply nested memory hierarchies, and millions of cores in the whole system. In general, many-core chips will most likely resemble a "hierarchical and distributed system on chip," and themselves will be part of a bigger hierarchical and distributed supercomputer. It is expected that such systems will be hard to exploit using conventional means, not only for performance, but also in terms of dealing with reliability, as well as power and energy issues [4].

Several past and on-going projects to tackle these challenges have been issued by major research agencies across the world, by several United States agencies such as the DOD [5], [6]; the DOE [7]; and the European Community [8]. In the context of such projects, the Codelet Model [9], a fine-grain, event-driven execution model, was proposed to take advantage of the foreseen massive parallelism that future supercomputers will expose at the node level, and beyond. Several implementations of the Codelet Model have been proposed [10], [11] for current supercomputers and many-core systems. They demonstrate good scalability both at the node level and for complete systems. However, the Codelet Model specification as well as its various implementations tends to focus on the expression of parallelism and task scheduling. This is good to target performance, but leaves aside other issues we mentioned above, namely ensuring the correct execution of programs for extreme-scale systems under a given set of constraints, which inevitably lead to fine-grain resource management.

The remainder of this paper is organized as follows: Section II presents the necessary background; Section III describes our proposed solution to run self-aware codelets on extreme-scale systems; and Section IV presents our conclusions and future work.

II. Background

A. The Codelet Model

The Codelet Model [9] is a hybrid von Neumann-dataflow execution model designed for extreme-scale systems in mind. Its quantum of computation is the *codelet*, a sequence of

978-1-4799-8095-6/14 $31.00 © 2014 IEEE
DOI 10.1109/DFM.2014.12

machine instructions that execute preemptively upon availability of specific resources, the primary one being data. Other resources may include bandwidth requirements, maximal power envelope, the use of an accelerator, network access, *etc.* Codelets are grouped into *codelet graphs*. Codelets do not have data of their own (except local data, such as data allocated in a stack for functions). Instead, codelets are parts of a bigger construct, the *Threaded Procedure* (TP). Threaded procedures are asynchronous functions. They act as containers for codelet graphs, and can be invoked in a control-flow manner. All codelets belonging to the same TP can access its frame, and therefore its data. The codelet graph within a TP is static, and the dynamic aspect of the model is conveyed thanks to the use of threaded procedures. TPs are assigned to a specific portion of the target machine, such as a group of cores (or a whole socket). Once assigned there, a TP cannot be migrated: all the codelets it contains will execute in that area of the machine. This constraint allows for a better control of locality at execution time.

The Codelet Model was implemented several times [10], [11], more or less faithfully. It is also the inspiration behind more recent endeavors to exploit future exascale systems [5], [7]. However, while the data-driven aspect is always a prime component, the more general event-driven aspect has generally been left aside as future work.

B. Self-Awareness for Extreme-Scale

Landwehr et al. [12] argued that systems need to become more self-aware and introspective with respect to performance, energy, and resiliency. Toward that end, they presented a Target Exascale Architecture (TEA) and associated toolchain as well discussed hardware requirements needed to enable self-adaptive and introspective system software. Core to concept of self-awareness is a notion of an observe-decide-act (ODA) loop to monitor, make decisions, and to control both hardware and software aspects at various levels of a system. Furthermore, they discussed research venues in terms of fine-grained and coarse-grained adaptation. The former entails the decision making process at the finest level of control within a self-aware system. For example, whether to DVFS components. The latter entails adaptation at a higher level and deals strictly with hierarchal management. It deals not only with the decision making process, but also the communication subsystem in place for communication between control engines. For example, how to minimize communication overhead.

While there have been proposals for self-aware systems prior to the proposed work, most of them focus either on very specific aspects of self-adaptation (*e.g.*, quality of service for network aspects of distributed applications [13], [14], or current multi-core systems (*e.g.*, SEEC [15]).

III. A Self-Aware Codelet Execution Model

Our goal is to bridge a dataflow-inspired fine-grain execution model (the Codelet Model) and a previously proposed self-aware fine-grain resource management system for extreme-scale systems, as shown in Figure 1. In this section we discuss the interactions between fine-grain and coarse-grain task specifications (including resource requirements), their execution, as well as the underlying system software decisions relative to the management of the hardware according to where the tasks are going to execute. We do not discuss task scheduling specifically, except when a decision that affects the hardware will affect program execution per se.

A. Extending the Codelet Specification

So far, codelets and the threaded procedure they belong to have been used solely using the data dependence relations they expose, proving that dataflow-inspired execution models are well-suited for the many-core era. However, certain aspects which used to be explored solely for performance purposes, such as locality, are becoming of paramount importance for other reasons, such as energy efficiency. In addition, fault tolerance is also an important aspect of future HPC systems, as the mean time between failures is expected to get shorter as transistors feature sizes diminish [16].

To deal with such issues, it is imperative to provide as much information to the underlying system (system software and hardware) as possible. We propose to augment the Codelet Model with metadata fields, attached to both its threaded procedures and their contained codelets.

1) Running Programs with Specific User Goals: A given codelet program may provide high-level user goals. Such goals could be for example "Target a 200W power consumption per chip, but be as parallel as possible." In the context of our Target Exascale Architecture, this would mean that each 2000-core chip should try to power up as many cores as possible while trying to reduce as much as possible power consumption. The user-input goals will determine how chips are being powered up from the get go. For instance, assuming only one block is active when launching the program with these specific user goals (and all other blocks are clock-gated), it could power up half of the remaining blocks that were inactive at near-threshold voltage (NTV) levels, and start running the program. That way, the *initial* configuration of the system will yield quite a lot of parallelism: Roughly 1000 cores will be up, thus satisfying the "high-parallelism" constraint; and all cores will be set at very low voltages, ensuring a low power consumption. As the execution progresses, if, despite NTV levels used to power the cores, power consumption is still too high, then the underlying system will need to adapt, possibly by clock-gating part or totality of some of the blocks that are currently in use. Such self-aware behavior needs to happen both at the local (block) level, where an individual control engine (CE) monitors itself and proceeds to make decisions and act on them, but also at the hierarchical level, where "super-CEs" will apply the self-aware concept using the ODA loop described in Section II to larger portions of the system: "normal" CEs will regularly inform their hierarchy of their health (in a manner not unlike Weis et al.'s proposed use of a

heartbeat for fault-tolerance [16]), while (elected) super-CEs will evaluate the global health of a cluster of blocks, and send back punctual orders to individual blocks to ensure that user goals are enforced.

However, using control theory and ODA loops as a basis to achieve self-awareness means that a "steady state" of a sort, satisfying all the high-level goals, is also achieved, preferably from the get-go. Simply running some kind of system software on top of the hardware may converge too slowly toward a solution, thus wasting precious cycles, energy, or both. It is imperative that, in such a parallel environment, each fine-grain task, and their containers embed additional information which will help the system converge faster toward the goals specified by the user.

2) Driving the Execution of Threaded Procedures: Threaded procedures are containers for a statically defined codelet graph. As such, they can retain information such as "contended" codelets—*e.g.*, codelets that are akin to barriers in more traditional execution models, and which collect a significant number of signals from previously ran codelets, only to trigger other "successor" codelets once fired; or they can store the "width" of their contained codelet graph, *i.e.* the maximal parallelism with which a given TP will require; and other resource-related information, *e.g.*, if some network connection is required by a TP, it may be beneficial to "push" it toward a cluster of cores that is located near the actual network port.

3) Coupling Fine-Grain Resource Management and Fine-Grain Dataflow Multi-Threading: Codelets themselves can significantly enhance program execution by providing information about the type of resources they will be requiring. Probably the most common types of resources that will need to be signaled will be the nature of the compute unit (*i.e.*, a simple compute core or an accelerator, whether it requires floating-point units, etc.); the data intensity required (*i.e.*, only local load/store operations, or cluster-to-cluster data movements, or "remote" data movements such as DRAM-to-local and vice-versa); and I/O operations (*i.e.*, fast interconnection network, disk access, *etc.*).

4) Mechanics of the Threaded Procedure–Codelet Meta-Data Interactions: Threaded procedures provide a rather "coarse-grain" view of the computation it contains. Its meta-data should be viewed as "general rules" to be taken into account by the manager which provides the required resources to the codelets contained in their TP. The metadata attached to codelets are supposed to add to the "general goals and rules" described in their TP's resource information. If for some reason a codelet's metadata contradicts its containing TP's, the codelet's resource requirements always win over its container.

B. Combining Codelet Resource Usage Description with Hierarchical and Distributed Adaptation to Satisfy User Goals

Once all threaded procedures and their containing codelets are tagged (by a hero programmer or a codelet-aware compiler) with relevant meta-data, the underlying codelet abstract machine model (or *Codelet AMM*, combining system sofware and hardware) can start executing the program they make up. To ensure a fast convergence toward the user goals however, additional mechanisms must be added to the Codelet AMM. Just as with "physical" metrics (*e.g.*, temperature, power consumption, *etc.*) that are used to ensure the integrity of the system w.r.t. user goals, per-codelet or per-threaded procedure metrics must be stored both locally at the block level, and at the various levels in the control hierarchy. By storing the local history of each codelet and TP's "type[1]," the codelet self-aware system can regularly update the behavior of codelets that it ran in the past. As a result, the resource manager can quickly decide what to turn on and off both locally and possibly on a larger scale, *e.g.*, how many local block memory banks require to be turned on, how many cores need to clock- or power-gated, what voltage level should be used, etc.

The system can also embed a series of "codelet patterns" as a knowledge base, both for local and global self-awareness. Predefined patterns could be as simple as recognizing "load data from a single codelet–compute in parallel with multiple codelets–store result using a single codelet" schemes, which are frequent in (say) dense linear algebra kernels. By recognizing such codelet graph patterns, a smart self-aware resource manager could decide to clock-gate all but the one core required to run the "load codelet," then exploiting the running time of a previously ran instance of the same codelet, trigger the order to reactivate the other cores just in time for them to be assigned the "compute codelets", right before the "load codelet" is done moving data. It is also expected that local histories for the same codelet types (or templates) will yield slightly or even widely differing resource usage: a given codelet may have only local data movement interactions in a given block, or it may have intense remote data movement interactions if it draws from the same block memory from another block.

IV. Conclusion and Future Work

We have presented a path toward a self-aware codelet-based program execution model. Such model relies on attaching the right meta-data to fine-grain event-driven tasks so as to inform the underlying system of the required resource usage of a given task. In addition, user goals must be taken into account, and will heavily influence the decision process underlying the self-aware system software.

We are currently working on a prototype which features a self-aware system targeting the X-Stack/Traleika Glacier architecture. It relies on an API that will be compatible with the OCR event-driven fine-grain execution model.

V. Acknowledgements

This material is based upon work supported by the Department of Energy [Office of Science] under Award Number DE-SC0008717. This work was partly supported by European FP7 project TERAFLUX, id. 249013.

[1]Here, we use the term loosely, and while typing fine-grain dataflow tasks (from a compilation perspective) is indeed an interesting area of research, it is not our intent to discuss it here.

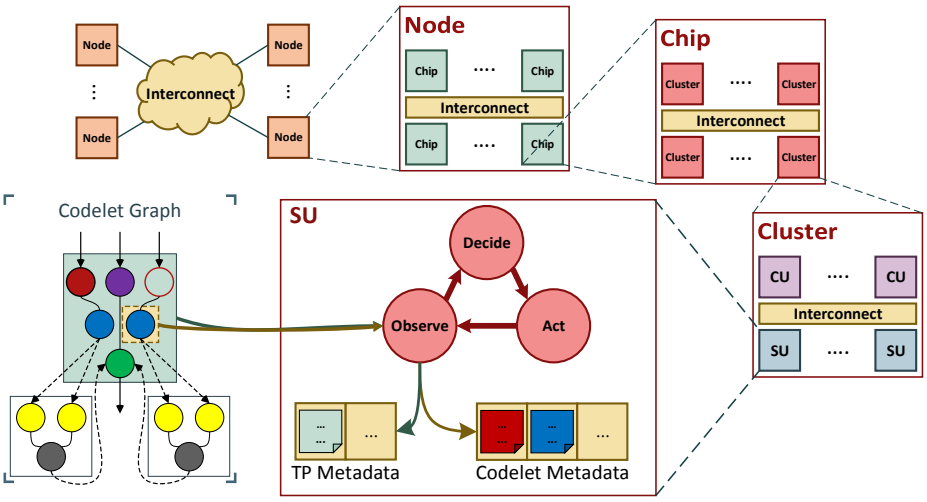

Fig. 1. An overview of our proposed self-aware codelet execution model. The scheduling units at both the TP and Codelet levels observe execution and record metadata and use it in the decision making process. The process feeds back into itself to constantly keep relevant and updated information about previously ran codelets.

REFERENCES

[1] L. V. Kale and S. Krishnan, "CHARM++: A Portable Concurrent Object Oriented System Based on C++," in *Proceedings of the Eighth Annual Conference on Object-oriented Programming Systems, Languages, and Applications*, ser. OOPSLA '93. New York, NY, USA: ACM, 1993, pp. 91–108. [Online]. Available: http://doi.acm.org/10.1145/165854.165874

[2] M. Pérache, H. Jourdren, and R. Namyst, "MPC: A Unified Parallel Runtime for Clusters of NUMA Machines," in *Euro-Par 2008 – Parallel Processing*, ser. Lecture Notes in Computer Science, E. Luque, T. Margalef, and D. Benítez, Eds. Springer Berlin Heidelberg, 2008, vol. 5168, pp. 78–88. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-85451-7_9

[3] P. Carribault, M. Pérache, and H. Jourdren, "Enabling Low-Overhead Hybrid MPI/OpenMP Parallelism with MPC," in *Beyond Loop Level Parallelism in OpenMP: Accelerators, Tasking and More*, ser. Lecture Notes in Computer Science, M. Sato, T. Hanawa, M. Müller, B. Chapman, and B. Supinski, Eds. Springer Berlin Heidelberg, 2010, vol. 6132, pp. 1–14. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-13217-9_1

[4] 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

[5] DARPA-BAA-10-37, *UHPC: Ubiquitous High Performance Computing*. Arlington VA, USA: DARPA, 2010-2012.

[6] N. P. Carter, A. Agrawal, S. Borkar, R. Cledat, H. David, D. Dunning, J. Fryman, I. Ganev, R. A. Golliver, R. Knauerhase, R. Lethin, B. Meister, A. K. Mishra, W. R. Pinfold, J. Teller, J. Torrellas, N. Vasilache, G. Venkatesh, and J. Xu, "Runnemede: An architecture for ubiquitous high-performance computing," *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, vol. 0, pp. 198–209, 2013.

[7] Department of Energy, "X-Stack — Extreme Scale Software Stack," 2012–2014. [Online]. Available: http://www.xstack.org

[8] R. Giorgi, R. M. Badia, F. Bodin, A. Cohen, P. Evripidou, P. Faraboschi, B. Fechner, G. R. Gao, A. Garbade, R. Gayatri, S. Girbal, D. Goodman, B. Khan, S. Koliaï, J. Landwehr, N. M. Lê, F. Li, M. Lujàn, A. Mendelson, L. Morin, N. Navarro, T. Patejko, A. Pop, P. Trancoso, T. Ungerer, I. Watson, S. Weis, S. Zuckerman, and M. Valero, "Teraflux: Harnessing dataflow in next generation teradevices," *Microprocessors and Microsystems*, no. 0, pp. –, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0141933114000490

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

[10] C. Lauderdale and R. Khan, "Towards a codelet-based runtime for exascale computing: Position paper," in *Proceedings of the 2Nd International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era*, ser. EXADAPT '12. New York, NY, USA: ACM, 2012, pp. 21–26. [Online]. Available: http://doi.acm.org/10.1145/2185475.2185478

[11] J. Suetterlein, S. Zuckerman, and G. Gao, "An implementation of the codelet model," in *Euro-Par 2013 Parallel Processing*, ser. Lecture Notes in Computer Science, F. Wolf, B. Mohr, and D. an Mey, Eds. Springer Berlin Heidelberg, 2013, vol. 8097, pp. 633–644.

[12] A. Landwehr, S. Zuckerman, and G. R. Gao, "Toward a self-aware system for exascale architectures," in *Euro-Par 2013: Parallel Processing Workshops*, ser. Lecture Notes in Computer Science, D. an Mey, M. Alexander, P. Bientinesi, M. Cannataro, C. Clauss, A. Costan, G. Kecskemeti, C. Morin, L. Ricci, J. Sahuquillo, M. Schulz, V. Scarano, S. L. Scott, and J. Weidendorfer, Eds., vol. 8374. Springer, 2014, pp. 812–822.

[13] V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F. Lo Presti, and R. Mirandola, "Moses: A framework for qos driven runtime adaptation of service-oriented systems," *Software Engineering, IEEE Transactions on*, vol. 38, no. 5, pp. 1138–1159, 2012.

[14] M. Giampapa, T. Gooding, T. Inglett, and R. Wisniewski, "Experiences with a lightweight supercomputer kernel: Lessons learned from blue gene's cnk," in *High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for*, 2010, pp. 1–10.

[15] H. Hoffmann, "Seec: A framework for self-aware management of goals and constraints in computing systems," Ph.D. dissertation, Massachusetts Institute of Technology, February 2013.

[16] S. Weis, A. Garbade, B. Fechner, A. Mendelson, R. Giorgi, and T. Ungerer, "Architectural support for fault tolerance in a teradevice dataflow system," *International Journal of Parallel Programming*, pp. 1–25, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10766-014-0312-y