Collaborative Computing for Heterogeneous Integrated Systems

Li-Wen Chang[†], Juan Gómez-Luna[*], Izzat El Hajj[†], Sitao Huang[†], Deming Chen[†], Wen-mei Hwu[†]

[†]University of Illinois at Urbana-Champaign, [*]Universidad de Córdoba
lchang20@illinois.edu, el1goluj@uco.es, {elhajj2, shuang91, dchen, w-hwu}@illinois.edu

ABSTRACT

Computing systems today typically employ, in addition to powerful CPUs, various types of specialized devices such as Graphics Processing Units (GPUs) and Field-Programmable Gate Arrays (FPGAs). Such heterogeneous systems are evolving towards tighter integration of CPUs and devices for improved performance and reduced energy consumption. Compared to traditional use of GPUs and FPGAs as offload accelerators, this tight integration enables close collaboration between processors and devices, which is important for better utilization of system resources and higher performance. Programming interfaces are also adapting rapidly to these tightly integrated heterogeneous platforms by introducing features such as shared virtual memory, memory coherence, and system-wide atomics, making collaborative computing even more practical.

In this paper, we survey current integrated heterogeneous systems and corresponding collaboration techniques. We evaluate the impact of collaborative computing on two heterogeneous integrated systems, CPU-GPU and CPU-FPGA, using OpenCL. Finally, we discuss the limitation of OpenCL and envision what suitable programming languages for collaborative computing will look like.

1. INTRODUCTION

While GPUs have been a central part of computing systems due to their ability to provide high performance at low energy costs while being programmable, FPGAs have also been integrated into computing systems by industry vendors such as Microsoft [19, 5], Intel/Altera [18, 2], Xilinx [22], and IBM [4] due to their extremely high power efficiency. The continuous demand for higher performance under constrained power and energy budgets is driving tighter system-level integration of CPUs with other coprocessors or accelerators, such as GPUs or FPGAs.

With such tight integration, fine-grain collaboration between processors becomes a practical approach to improving resource utilization and system performance [12]. For this reason, programming interfaces such as OpenCL 2.0 and CUDA 8.0 have introduced features such as shared virtual memory and system-wide atomics to productively express fine-grain collaboration. Among existing programming interfaces, OpenCL is perhaps the most suitable interface for programming these heterogeneous architectures because it is supported by a wide range of processors, including CPUs and GPUs, as well as FPGAs via high-level synthesis (HLS). Although OpenCL FPGA stacks [13] currently support only OpenCL 1.2 which does not support the new features, it is expected to support OpenCL 2.0 in the near future.

Multiple studies [8, 14] have investigated characteristics of and optimization for heterogeneous systems with CPUs, GPUs, and FPGAs. Recent literature [21, 15, 16, 11, 10] has focused on fine-grain collaboration in the context of system-level CPU-GPU integration. A variety of general collaboration patterns have been investigated [21, 11], which can be mainly classified into data or task partitioning. However, similar collaborative computing techniques have not been studied for CPU-FPGA systems.

We envision future integrated heterogeneous system including CPUs, GPUs, and FPGAs. Features such as shared virtual memory and system-wide atomics will be widely adopted in most programming models. Coherence will be supported, which can either be achieved via unified physical memory or via coherence protocols over various interconnect technologies. These features are a step forward in programmability, simplifying fine-grain collaboration which will become ubiquitous, instead of traditional use of GPUs and FPGAs as offload accelerators.

Although OpenCL and other existing programming models support expressing fine-grain collaboration in applications for specific integrated systems, these programming models show limited support for automatically converting a program from one collaboration pattern to another pattern across systems. We envision new high-level programming languages that are capable of synthesizing kernels with different collaboration patterns from generic representations of the program. These languages will replace OpenCL as a programming interface for end users.

The rest of this paper is organized as follows. Section 2 describes our envisioned heterogeneous integrated system, and surveys existing techniques to realize these systems. Section 3 describes collaborative computing, its patterns, and provides a preliminary evaluation using current integrated systems as proxies. Section 4 describes the limitation of current programming models and our envisioned model. Section 5 concludes this paper and outlines future work.

ICPE'17, April 22-26, 2017, L'Aquila, Italy

© 2017 ACM. ISBN 978-1-4503-4404-3/17/04...$15.00

DOI: http://dx.doi.org/10.1145/3030207.3030244

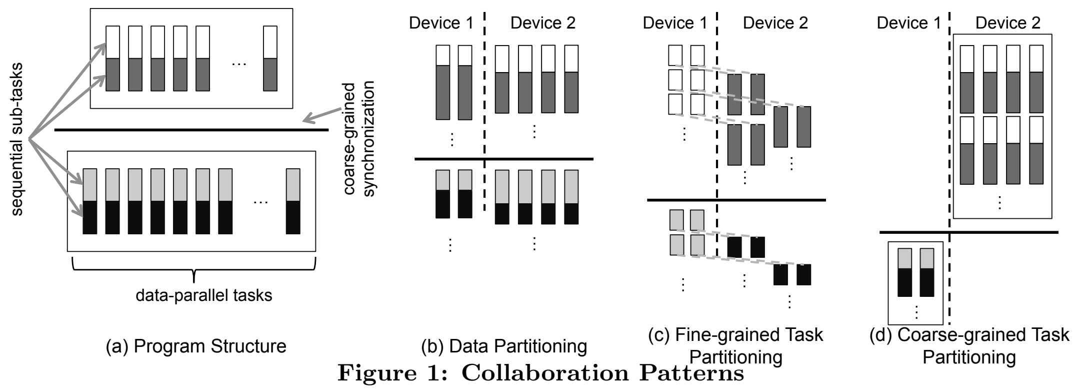
sequential sub-tasks
coarse-grained synchronization
data-parallel tasks
(a) Program Structure
Device 1
Device 2
(b) Data Partitioning
(c) Fine-grained Task Partitioning
(d) Coarse-grained Task Partitioning

Figure 1: Collaboration Patterns

2. INTEGRATED HETEROGENEOUS SYSTEMS

Traditionally, accelerators such as GPUs and FPGAs have been connected to CPUs through PCIe or similar interfaces, which provide efficient large-block data transfer through Direct Memory Access (DMA) engines between devices. However, DMA overhead is a burden on performance for smaller data transfers due to non-uniform data accesses in many contemporary workloads (e.g., graph-based algorithms, machine learning inference, etc.). For this reason, heterogeneous systems need shared memory between CPUs and accelerators. There have been several recent efforts led by industry vendors in this direction for both CPU-GPU and CPU-FPGA systems. They all provide shared coherent memory in a similar way, but differ in CPU architecture, processor implementation, and silicon fabrication.

For CPU-GPU systems, NVIDIA has launched the Pascal architecture [17] which implements coherence over PCIe and NVlink. AMD APUs provide even closer integration by coupling devices on the same die, and using specialized memory buses. The CPU-FPGA counterparts are Intel QPI [18], Hyper Transport, Front Side Bus (FSB), AXI Coherency Extension (ACE) [22], Acceleration Coherency Port (ACP), ARM Core Link Interconnect, IBM CAPI [4], and CCIX [1].

We envision that future heterogeneous systems will contain CPU cores with one or multiple GPUs and/or FPGAs. The three devices have access to coherent memory through specific interfaces. Additional non-coherent interfaces might also be included for data not shared across devices, or not simultaneously accessed. The system could include a shared last-level cache (LLC) that can help accelerate collaborative workloads [10]. The three devices might be integrated at either chip-level or realized as separate chips. As mentioned above, current trends integrate CPU and GPU or CPU and FPGA [2] in the same die. However, systems with the three devices in the same die are already available, though the GPU is not OpenCL-programmable yet [22].

3. COLLABORATIVE COMPUTING

3.1 Collaboration Patterns

Multiple collaboration patterns have been studied for integrated heterogeneous architectures [11, 21]. These patterns can be classified into two major types based on work partitioning strategies: data partitioning and task partitioning. Task partitioning can be further refined into fine-grain and coarse-grain, based on partitioning points in the code.

Figures 1(b)-(d) illustrate different patterns for an example program in Figure 1(a) containing two sets of data-parallel tasks, each with two sequential sub-tasks (white/dark gray, and light gray/black). We provide a few key insights behind the collaboration patterns, while referring to [11] for more detailed explanations. First, given an application, different devices might prefer different sub-tasks due to different device characteristics (such as parallelism and resources), making collaborative computing non-trivial. Second, while fine-grain task partitioning (Figure 1(c)) improves utilization of parallelism across devices via pipeline execution, it may require coherent memory and system-wide atomic support which may introduce some communication overhead. Third, traditional accelerator models with CPUs and accelerators are special cases of coarse-grain task partitioning (Figure 1(d)). Last, coarse-grain task partitioning may utilize all devices in a pipeline fashion if multiple independent data sets are processed independently.

Multiple factors may impact the choice of collaboration pattern. These factors include: the diversity of sub-tasks, the cost of communication between devices (latency and bandwidth), the relative metrics (such as performance, power, energy, etc.) among devices for a specific workload, and hardware resource constraints. Moreover, collaborative computing can be considered with multiple objectives such as maximizing performance, minimizing latency, minimizing power, or minimizing energy. In this paper, we mainly focus on performance.

3.2 Preliminary Evaluation

We evaluate the performance benefits of collaborative execution in two current heterogeneous integrated systems: CPU-GPU and CPU-FPGA. Our CPU-GPU system is an AMD Kaveri A10-7850K APU, while our CPU-FPGA system is an Intel Xeon E3-1240 v3 connected through PCIe 3.0 x8 with an Altera Stratix V GX FPGA on a Terasic DE5-Net board. The AMD APP SDK 3.0 with OpenCL 2.0 is used for Kaveri, and Intel OpenCL FPGA SDK 16.0 with OpenCL 1.2 is used for Stratix V, since OpenCL 2.0 is not supported. Two OpenCL benchmarks, Canny Edge Detection (CED) and Random Sample Consensus (RSC) from the Chai benchmark suite [11] are used for evaluation. CED is evaluated for both data partitioning and coarse-grain task partitioning, while RSC is evaluated for both data partitioning and fine-grain task partitioning.

CPU-GPU. Figure 2 presents the speedups of CED and RSC over CPU-only on the CPU-GPU system. For CED, while GPU-only delivers 9.30× more performance than CPU-only, collaboration with data partitioning does

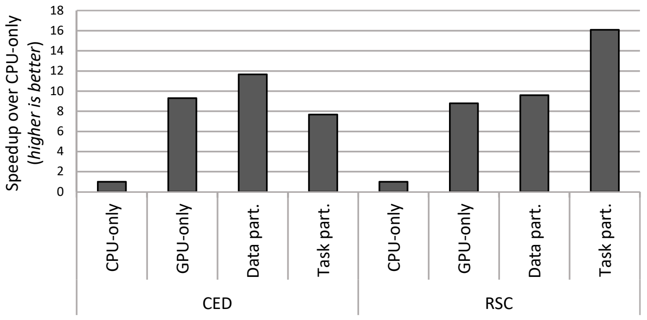

Figure 2: Speedup on CPU-GPU

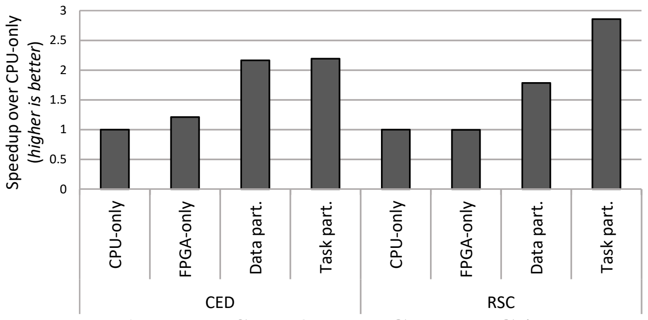

Figure 3: Speedup on CPU-FPGA

even better with 11.67× and 1.26× over CPU-only and GPU-only respectively. However, unlike data partitioning, coarse-grain task partitioning is only 0.83× of GPU-only performance. One reason is that by assigning tasks to the CPU, the performance of the entire flow is hampered, since in CED the GPU performance dominates the CPU. Another reason is that task partitioning introduces cache coherence cost for data sharing between the CPU and the GPU.

For RSC, while GPU-only achieves an 8.80× speedup over CPU-only, collaboration with fine-grain task partitioning outperforms CPU-only and GPU-only by 16.14× and 1.83× respectively. The main reason is that one sub-task in RSC is inherently sequential so its efficiency is improved by moving it to the CPU. However, data partitioning is only slightly better than GPU-only by 1.09×, and significantly worse than fine-grain task partitioning by 0.59×, because the GPU efficiency in data partitioning still suffers from that inherently sequential sub-task.

These two examples show that the best collaborative execution pattern highly depends on application characteristics.

CPU-FPGA. Figure 3 shows speedups for CED and RSC over CPU-only on the CPU-FPGA system. The best CED collaboration pattern is coarse-grain task partitioning which is 2.25× and 1.85× faster than CPU-only and FPGA-only respectively. Data partitioning is just slightly slower than coarse-grain task partitioning by a factor of 0.96×. The best RSC collaboration pattern is fine-grain task partitioning, with 2.57×, 2.36×, and 1.38× speedup over CPU-only, FPGA-only, and data partitioning respectively. The reason is similar to CPU-GPU collaboration.

An important consideration for improving performance on FPGAs is the OpenCL HLS *kernel duplication* factor. Performance typically improves with more duplication, but then saturates when a limit on one of the resources is reached. Different applications, or the same application with a different collaboration pattern, may saturate for different resources. In our evaluation, the performance of CED with data partitioning and task partitioning both saturate the memory bandwidth. However, for RSC, while task partitioning saturates the memory bandwidth, data partitioning saturates the DSP resource on FPGAs before the memory bandwidth limit is reached. This explains why the disparity between collaboration patterns differs across applications.

Comparison. In our evaluation for both systems, collaborative execution seems beneficial for performance if applied correctly. We observe that different applications favor different collaboration patterns. For example, CED performs better with data partitioning, while RSC favors task partitioning. We also observe that different devices may favor different collaboration patterns. For example, for CED, data partitioning is better on CPU-GPU, but task partitioning is slightly better on CPU-FPGA.

Note that the collaboration had more performance benefit in the CPU-GPU platform than in the CPU-FPGA platform. One reason is that in the CPU in our CPU-FPGA platform is more powerful than the CPU in our CPU-GPU platform, diminishing the speedups gained from using FPGA. Another reason is the difference in OpenCL support. OpenCL 2.0, which is used for CPU-GPU execution, comes with heterogeneous features such as shared virtual memory and system-wide atomics that enable more practical collaboration than OpenCL 1.2 which is used for CPU-FPGA execution. Moreover, OpenCL stacks for FPGAs based on HLS might be less mature than OpenCL stacks for GPUs.

4. PROGRAMMING INTERFACE

4.1 Limitation of Current Practice

Different collaboration strategies can be seen as different choices of program optimizations. In this sense, conversion between collaboration strategies is a code transformation between specific optimizations. Finding the best collaboration strategy for a given program is an optimization-space exploration problem, while adapting across different integrated systems is a performance portability problem. Current practices of programming languages, such as OpenCL, require programmers to explicitly express collaboration strategies. Doing so limits the potential for automatic code transformation for collaboration in multiple ways.

First, it is challenging to automatically apply fine-grain task partitioning to a generic OpenCL kernel. Sub-tasks can only be partitioned at specific points of the code. More precisely, it requires kernel fission at specific points of the code and introduces extra queue objects for communication among those new kernels. Therefore, while coarse-grain task partitioning is commonly adopted in adaptive runtimes [9, 3] and data partitioning can be achieved through parameter tuning or dynamic fetching [11], fine-grain task partitioning, to the best of our knowledge, has not yet been automated by existing OpenCL frameworks for general purpose applications.

Second, it is even more challenging to automate conversion from one collaboration strategy to another in OpenCL. Conversion from data partitioning or coarse-grain task partitioning to fine-grain task partitioning is at least as difficult as automation of fine-grain task partitioning, which requires kernel fission. On the other hand, conversion from fine-grain task partitioning into data partitioning or coarse-grain task partitioning requires kernel fusion. Considering two sub-task kernels might not have identical work-item/work-group mapping, kernel fusion requires sophisticated analyses and transformations to harmonize the mapping. Even though it is possible, the introduced branch divergence might tax the benefits of kernel fusion.

Lastly, to the best of our knowledge, OpenCL delivers limited performance portability when used as one source for different devices [7, 6].

4.2 High-Level Programming Language

High-level languages such as TANGRAM [6] and Halide [20] provide generic unified programming interfaces for applications, and deliver promising performance portability across devices. By introducing collaboration strategies as optimizations, kernels with collaborative computing can be directly synthesized in these high-level languages. Therefore, compared to OpenCL, we envision these high-level languages will be more suitable programming interfaces for collaborative execution.

For example, TANGRAM is able to synthesize kernels with various granularities. For fine-grain task partitioning, atomic codelets in TANGRAM can be considered the most fine-grain sub-tasks, thus defining sub-task boundaries. Doing so avoids having to identify fission points in an OpenCL kernel, and significantly simplifies the automation of fine-grain task partitioning. Coarse-grain task partitioning can be achieved by simply synthesizing each kernel to the target device it will be mapped to. Data partitioning can be achieved by extending TANGRAM's containers such as `map` and `partition` for adjusting work-item/work-group mapping. While TANGRAM currently supports only CPUs and GPUs, introducing HLS as a backend can extend TANGRAM to support FPGAs.

While these high-level languages have promising potential to automatically synthesize kernels with various collaboration patterns, there remains the step of searching for the optimal collaboration strategy for a given integrated system. This emphasizes the need for performance modeling, effective heuristics, offline tuning, or adaptive runtime for collaborative computing on heterogeneous integrated systems.

5. CONCLUSION AND FUTURE WORK

In this paper, we have explored collaborative computing for heterogeneous integrated systems. We surveyed techniques in integrated architectures for enabling effective collaboration. We demonstrated the benefits of collaborative computing by evaluating both CPU-GPU and CPU-FPGA systems. We also discussed the limitation of current programming interfaces, such as OpenCL. Finally, we envision that multiple high-level languages, such as TANGRAM and Halide, will be more suitable programming interfaces for collaborative computing than OpenCL.

For future work, we will study more heterogeneous implementations of collaborative workloads in order to find generic optimization techniques which can be automatically applied by compilers of high-level languages.

Acknowledgment

This material is based upon work supported by the DoE National Nuclear Security Administration (DE-NA0002374), Hewlett-Packard Enterprise Labs, the Starnet Center for Future Architecture Research (C-FAR), the Huawei Project (YB2015120003), and the IBM Center for Cognitive Computing Systems Research Center at UIUC. We also thank the Ministry of Education of Spain (TIN2013-42253P) and the Junta de Andalucía of Spain (TIC-1692).

6. REFERENCES

[1] Cache Coherent Interconnect for Accelerators (CCIX). http://www.ccixconsortium.com, 2016.

[2] Altera. Altera's User-Customizable ARM-Based SoC, 2015.

[3] C. Augonnet et al. StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. *CCPE*, 23(2):187–198, 2011.

[4] Bruce Wile. IBM Systems and Technology Group. Coherent Accelerator Processor Interface (CAPI) for POWER8 systems. White paper, September 2014.

[5] A. M. Caulfield et al. A cloud-scale acceleration architecture. In *ISCA*, 2016.

[6] L.-W. Chang et al. Efficient kernel synthesis for performance portable programming. In *MICRO*, 2016.

[7] L.-W. Chang et al. A programming system for future proofing performance critical libraries. In *PPoPP*, 2016.

[8] E. S. Chung et al. Single-chip heterogeneous computing: Does the future include custom logic, FPGAs, and GPGPUs? In *MICRO*, 2010.

[9] A. Duran et al. OmpSs: a proposal for programming heterogeneous multi-core architectures. *PPL*, 21(02):173–193, 2011.

[10] V. Garcia-Flores et al. Evaluating the effect of last-level cache sharing on integrated GPU-CPU systems with heterogeneous applications. In *IISWC*, 2016.

[11] J. Gómez-Luna et al. Chai: Collaborative heterogeneous applications for integrated-architectures. In *ISPASS*, 2017 (in press).

[12] W.-m. W. Hwu. *Heterogeneous System Architecture: A New Compute Platform Infrastructure*. Morgan Kaufman, 2015.

[13] Intel. Intel FPGA SDK for OpenCL. Programming Guide, October 2016.

[14] A. Morad et al. Generalized multiAmdahl: Optimization of heterogeneous multi-accelerator SoC. *IEEE CAL*, 13(1):37–40, Jan. 2014.

[15] S. Mukherjee et al. Exploring the features of OpenCL 2.0. In *IWOCL*, 2015.

[16] S. Mukherjee et al. A comprehensive performance analysis of HSA and OpenCL 2.0. In *ISPASS*, 2016.

[17] NVIDIA. NVIDIA Tesla P100. White paper, 2016.

[18] PK Gupta. Intel. Xeon+FPGA Platform for the Data Center, June 2015.

[19] A. Putnam et al. A reconfigurable fabric for accelerating large-scale datacenter services. In *ISCA*, 2014.

[20] J. Ragan-Kelley et al. Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In *PLDI*, 2013.

[21] Y. Sun et al. Hetero-Mark, a benchmark suite for CPU-GPU collaborative computing. In *IISWC*, 2016.

[22] Xilinx. Zynq UltraScale+ MPSoCs. White Paper, June 2016.