Telescoping Architectures: A Methodology for Evaluating Next-Generation Heterogeneous Computing

Konstantinos Krommydas
Dept. of Computer Science
Virginia Tech
kokrommy@vt.edu

Wu-chun Feng
Dept. of Computer Science
Virginia Tech
wfeng@vt.edu

***Abstract*—Architectural innovation has telescoped the HPC community from the commodity (Beowulf) cluster in a machine room, i.e., a multi-node system with Ethernet interconnect, to a commodity cluster on a chip, i.e., multicore CPU with an on-die interconnect. We project that this "telescoping architecture" will apply more broadly to heterogeneous computing, namely from heterogeneous clusters like Tianhe-2 in a machine room to on a chip. To that end, we present an experimental study that extends the notion of telescoping architectures to identify the ideal mixture of compute engines (CEs) and the number of such CEs on a chip to create a heterogeneous "cluster on a chip" (CoC). Specifically, we experiment with heterogeneous architectures that contain single or multiple instances of CPUs, GPUs, Intel MICs, and FPGAs to demonstrate their performance efficacy given continuing advances in hardware technology, software, tools, and run-time support.**

***Index Terms*—architecture; microprocessor design; heterogeneous computing; dwarfs; motifs; system on a chip;**

I. Introduction

Moore's Law has fueled processor advances for half a century now. Transistor scaling, as outlined in Dennard's law [1], calls for a reduction in transistor dimensions by 30% with every generation and area shrinkage by 50%. This doubling of transistor density has allowed microarchitectural innovations that provide further performance increases (on the order of 40%, as per Pollack's Rule [2]) and the design of multi-level cache hierarchies to address the increasing gap between processor and memory speeds. While these trends have held true of single-core designs for decades; limitations in transistor scaling, prohibitive heat dissipation levels, and power/energy-related constraints have mandated a switch to multicore in the mid-2000s. Looking ahead, large-scale parallelism with *heterogeneous* cores now appears to be on the path to exascale computing.

The trend of heterogeneity is not new; it is a recurring pattern observed in chip design. Heterogeneity was first introduced in the form of discrete devices that were later *unified* on-chip in a subsequent technology iteration. For instance, specialized floating-point or other co-processors (e.g., encryption, signal processing) that complemented early single-core CPUs eventually relocated onto the CPU die itself. By the mid-2000s, the multi-core CPU era effectively telescoped the homogeneous compute cluster from a machine room to an *homogeneous cluster on a chip*. By the late 2000s, this same telescoping trend saw discrete GPUs that were out on the PCIe interface move onto the CPU die to create a fused CPU+GPU die called an *accelerated processing unit* (APU). Discrete FPGAs have also been used as co-processors to CPUs. With Intel's recent acquisition of Altera, a key FPGA vendor, Intel will telescope previous "discrete CPU+FPGA" offerings into a fused on-package integration with the Purley platform and then a fused on-die integration with the Tinsley platform.

Currently, a typical supercomputer node may exploit the synergistic (parallel) performance gains of architectures of heterogeneous nature, such as multi-core CPUs, GPUs, Intel MICs (Xeon Phi), or even FPGAs. These constituent elements are interconnected via PCIe at the intra-node level and over high-speed interconnects (e.g., Infiniband) at the inter-node level. Following the above examples on the recurring, a la *déjà vu*, trend of *architectural unification* that has defined the chip evolution, we project that it may only be a matter of years before the notion of telescoping architectures that led from a "homogeneous cluster in a machine room" to a "homogeneous, multicore-CPU cluster on a chip" is extended to create a *heterogeneous cluster on a chip* (CoC).

In light of this trend and given the variety of homogeneous architectures (i.e., CPU, GPU, Intel MIC, FPGA), the research questions that arise are as follows:

1) What is the ideal mixture of compute engines (CEs) and number of such CEs that will fuel telescoping architectures and enable the transformation of a heterogeneous cluster in a machine room to a heterogeneous CoC?
2) What methodology should we use to answer the above question in a systematic and generalizable (i.e., "state of the practice") way?

To address these questions, we study heterogeneous CoC and discuss the roadblocks that need to be addressed before such a concept is materialized at the intra-node level. Such a preliminary study is imperative before it can be expanded to the inter-node level, where new sets of challenges arise.

We summarize our contributions below:

- A systematic and generalizable way to identify directions in heterogeneous computing via the use of *dwarf-based (or motif-based) benchmarking*.
- Application of the above "state-of-the-practice" methodology to quantify the performance benefits of different instantiations of a *cluster on a chip* (CoC), each employing single or multiple instances of CPUs, GPUs, Intel MICs and FPGAs.

The rest of the paper is organized as follows: Section II provides an overview of the trend in architectural unification and background on the dwarfs concept and the rationale in using them as the driving force of architectural innovation. Section III provides details on our approach and experimental methodology, and Section IV presents our results and insights. In Section V, we highlight challenges that affect the longer-term feasibility of the proposed architectures. Finally, Section VI discusses related work, and Section VII concludes the paper and provides directions for future research.

II. Background

A. Architectural Unification in the History of Computing

In this work we examine the trends towards architectural unification as we move from a heterogeneous cluster in a machine room to a heterogeneous cluster on a chip. In order to put this trend in perspective it is useful to examine how the unification trend has manifested throughout computing history. Moore's law has allowed doubling the transistor density in a chip and this has facilitated heterogeneity at various conceptual levels and at the same time on-chip replication of homogeneous resources.

As far as *heterogeneous coupling* is concerned, there are various notable examples. At the early days of computing, floating-point arithmetic was implemented in software (e.g., on Intel 8086). The Intel 8087 was the first math co-processor for the 8086 line that allowed fast, hardware implementation of floating-point instructions. Nowadays, all modern processors include an (integrated) floating-point unit (FPU) and dedicated floating-point registers on-chip.

Graphics processing units (GPUs), in a similar fashion, originated as separate devices (graphics co-processors) with dedicated graphics memory. While still widely available in a discrete form factor, *integrated graphics processors* (IGPs) are the norm in many cases, especially in the laptop market. For example, Intel HD Graphics [3] in Ivy Bridge and Intel Iris Graphics [4] in Haswell CPUs are IGPs on the same package or die as the CPU and utilize a portion of the computer system RAM. Similarly, in 2011, AMD introduced Llano [5], the first generation of *accelerated processing units* (APUs), which combined a CPU and GPU on the same die.

FPGAs have also been used to accelerate computations in many domains, such as bioinformatics [6], [7] and finance [8], [9]). The Microsoft Catapult project [10] is exploring the use of FPGAs in the datacenter, with its first instantiation in the context of web search (i.e., Bing). FPGAs connect to a host platform typically via the PCIe interface or Ethernet. However, there are system-on-chip (SoC) implementations, where a CPU is embedded in an FPGA board (e.g., big.LITTLE by ARM [11], Cyclone V SoC with ARM Cortex-A9 [12]). To this end, Intel is also introducing their Xeon and FPGA accelerator platform that incorporates an FPGA module attached to the processor via a Quick Path Interconnect (QPI), After Altera's acquisition by ntel, it is expected that in the next year, we will see a FPGA fabric as part of the package or die.

As far as *replication of homogeneous resources* is concerned, the CPU and GPU cases are indicative examples. In the former case, single-core CPUs initially got connected over a network to form compute clusters. When manufacturing technology allowed, multiple cores (dual-, quad-, octa-cores, etc.) fitted on the same chip. Later, with Intel Many Integrated Core (MIC) [13] architecture, tens of (simpler) cores (60 or 61) became a reality. The next generation of Intel MIC (code-named *Knight's Hill*) will be bootable, obviating the need for a host processor needed in current generation's MICs. Similarly, GPUs include an ever-increasing number of compute units (or *CUDA cores* in NVIDIA terminology) and more advanced architectural features.

Both above trends, in isolation and, more so, combined, heed towards the concept of *supercomputing on a chip*.

B. Dwarf-Based Benchmarking and the OpenDwarfs

High-performance computing and benchmarking have traditionally gone hand in hand. Benchmarks are being routinely used in assembling lists that rank supercomputers according to various metrics, like performance or power (e.g., Top500 [14], Green500 [15], HPCG [16]). Additionally, computer architects have been utilizing benchmarks as part of characterizing their platforms and in the subsequent decision-making that is related to the microarchitectural advances of future renditions thereof. Related research [17] from a software engineering standpoint highlights the need for benchmarks that address real-world problems that are tied to *scientific paradigms*. This notion of paradigm encapsulates a universally agreed-on set of solutions to such real-world problems. These scientific paradigms mirror what is otherwise as a *computational dwarf* (or *motif*).

A *dwarf* is an algorithmic method that embodies a certain computation and communication pattern. Such patterns may be more or less suitable for one architecture over another. This motivates the need for heterogeneous architectures, like CPUs plus GPUs within a node, or ideally, a combination of more heterogeneous platforms in the context of a single chip, the type and characteristics of which will make them ideal for addressing the needs of applications. Original unpublished work by P. Colella introduced the concept of the dwarfs and a set of seven original dwarfs. The concept and work was later formalized and extended by Asanovic et al. [18] and became what is widely known as the *Berkeley Dwarfs*.

In this work, we use OpenDwarfs [19], a benchmark suite based on the concept of the Berkeley Dwarfs. OpenDwarfs includes instantiations for dwarfs in [18] and is developed in OpenCL, which allows a seamless benchmarking experience across different architectures, like multi-core CPUs,

TABLE I: Configuration of the Target Fixed Architectures

Model	AMD Opteron 6272 (CPU)	AMD Radeon HD 7970 (GPU)	Intel Xeon Phi P1750 (MIC)
Type	CPU	Discrete GPU	Co-processor
Frequency	2.1 GHz	925 MHz	1.09 GHz
Cores	16	32†	61
Threads/core	1	4	4
L1/L2/L3 Cache (KB)	16/2048/ 8192‡	16/768/- (L1 per CU)	32/512/- (per core)
SIMD (SP)	4-way	16-way	16-way
Process	32nm	32nm	22nm
TDP	115W	210W	300W
GFLOPS (SP)	134.4	3790	2092.8

† Compute Units (CU) ‡ L1: 16KBx16 data shared, L2: 2MBx8 shared, L3: 8MBx2 shared

TABLE II: Defining Base Unit (BU) for Chip Area Size

Type	**Model**	**transis-tors (bil.)**	**Process (nm)**	**Base Units**
CPU	AMD Opteron 6272	2.4	32	1
GPU	AMD Radeon HD 7970	4.3	28	1.372
MIC	Intel Xeon Phi P1750	5	22	0.985
FPGA	Xilinx Virtex-6 LX760	5.8	40	3.778

GPUs, Intel MIC (Xeon Phi), and even FPGAs by use of Altera OpenCL or the SOpenCL tool [20]. More importantly, OpenDwarfs is characterized by uniformity of optimization levels across the supported architectures, creating a level field for studying architectures with respect to their suitability for the computation and communication patterns themselves. The above set of features render OpenDwarfs an ideal benchmark suite for the purposes we seek to address in this work. In Section III-B, we discuss how we exploit dwarf instantiations in OpenDwarfs in a systematic and generalizable way that allows us to draw broader conclusions about next-generation heterogeneous architectures.

III. Methodology

In this section, we present our methodology in addressing the research questions set forth in Section I. We start with identifying the search space for candidate *Cluster on a Chip* (CoC) platforms. Then (Sections III-B and III-C) we describe in detail how we evaluate the performance of each CoC candidate. Section III-D discusses assumptions we make in the process.

A. Cluster on a Chip (CoC)

By *Cluster on a Chip* (CoC) we refer to the combination of discrete types and numbers of compute engines (CEs). This conglomerate of CEs constitutes the expected result of the unification trend that we describe in Section II. Such a hypothetical platform may not be feasible under current technology and manufacturing constraints. We discuss our assumptions in Section III-D. CEs under consideration include a general-purpose CPU, a high-performance discrete GPU, an Intel Xeon Phi (MIC) co-processor, and an FPGA. Table I shows the detailed characteristics of each platform.

Given these CEs we construct hypothetical CoCs by creating combinations thereof (also allowing multiple CE instances). In order to restrict the CoC search space, we enforce an (arbitrary) *chip area constraint*. Table II shows how we define a *chip area base unit* (BU), based on the number of transistors of each device and the process technology used in each to approximate the chip area size of each CE, assuming that transistors for all four types of CEs are laid out in the two-dimensional space. A BU measures the relative chip area using the calculated CPU chip area as a baseline. For example, given the values for number of transistors and process technology in Table II for CPU and GPU, the BUs of a GPU are $(\frac{28}{32})^2 * \frac{4.3}{2.4} = 1.372$. Following the above, the restriction we enforce is that all CoCs we consider have a chip size area that is equal or less than the aggregate area of one CPU, plus one GPU, plus one Intel Xeon Phi, plus one FPGA. Based on Table II (*Base Units* column) this adds up to 7.135 BUs. According to this constraint there are 100 possible CoCs.

TABLE III: OpenDwarfs Benchmark Test Parameters/Inputs

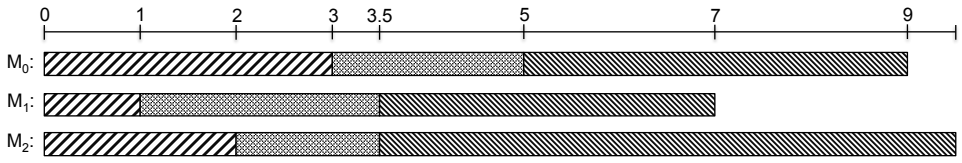
Dwarf	**Algor.**	**Problem Size**
N-body methods	GEM	Input file: nucleosome
Dynamic programming	NW	Two 4096-letter protein sequences
Structured grids	SRAD	2048x2048 FP, 128 iterations
Graph traversal	BFS	248,730 nodes, 893,003 edges
Combinational logic	CRC	Input data-stream: 100MB
Sparse linear algebra	CSR	2048^2 x 2048^2 sparse matrix

TABLE IV: Execution Time (in msec) of Dwarf Benchmarks

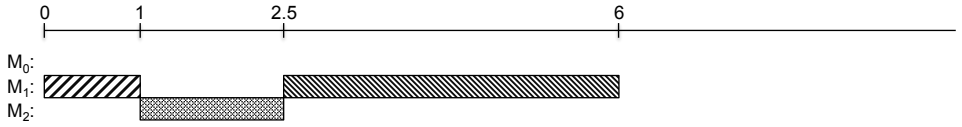
	GEM	**NW**	**SRAD**	**BFS**	**CRC**	**CSR**
CPU	21592	112	5093	331	672	22
GPU	401	672	232	96	19	4
MIC	11871	222	2298	278	881	5
FPGA	25345	35	17651	105	24	83

B. Performance Evaluation

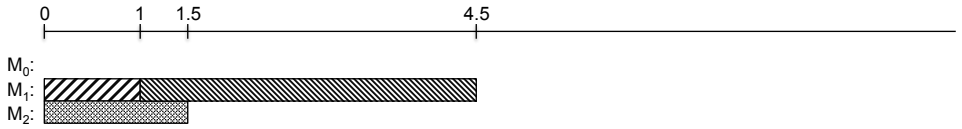
For evaluating the CoC candidates (Section III-A) we use the OpenDwarfs benchmark suite. Specifically, we employ a subset of the dwarf instantiations: GEM, NW (Needleman-Wunsch), SRAD (Speckle-Reducing Anisotropic Diffusion), BFS (Breadth-first Search), CRC (Cyclic Redundancy Check), and CSR (Compressed Sparse-Row Matrix-Vector Multiplication). The dwarf categories are shown in Table III together with the dwarf instantiations listed above and their input parameters/datasets. Using these dwarfs we create a large number of *synthetic benchmarks*. These synthetic benchmarks contain all possible combinations of four, five, and six dwarfs. This allows for the creation of benchmarks that cover a sufficiently large number of potential real-world applications. This stems from the fact that dwarfs, by definition, represent computation and communication patterns and real-world applications are largely composed of such patterns (dwarfs) that can be *temporally or spatially* distributed across a set of CEs (examples of dwarf composition of five ParLab applications and seven general application areas are given in [21]). Along these lines, in each



(a) Baseline schedules (running all jobs on the same, single CE, i.e., M0, *or* M1, *or* M2)



(b) Schedule 1 (running on fastest executing CE, one active at a time)

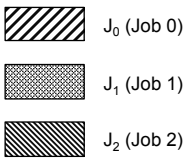


(c) Schedule 2 (running on fastest executing CE, multiple active at a time)

(d) Schedule 3 (running on fastest available CE, multiple active at a time)

	M_0	M_1	M_2
J_0	3	1	2
J_1	2	2.5	1.5
J_2	4	3.5	6

(e) Execution times



(f) Legend

Fig. 1: Scheduling examples (single instance, Mi, of each CE type available)

synthetic benchmark we assume the latter (spatial) distribution of constituent parts, i.e., dwarfs within an application are independent and can run in parallel in a form of *request-level parallelism* (similar assumptions are used in similar works, e.g., [22]). Each dwarf runs in parallel by itself on the specific CE it is scheduled on. The way each dwarf of a benchmark is scheduled to each part of a CoC is described in Section III-C. The above methodology allows us to draw broader conclusions that are better representative of real-world workloads.

For our results we use the kernel execution times presented in [23] and shown in Table IV. Characterizing the dwarfs on each of the different architectures (CPU, GPU, Intel MIC, FPGA) is also done in [23] and is beyond the scope of this paper. In this work we focus on the *overall* performance benefits of using CoCs. For this reason, the results of synthetic benchmarks that contain all possible combinations of four, five, and six dwarfs (among the total six used) are averaged (*4-mers, 5-mers, 6-mers*). As we would see in Section IV the trends observed in the results are similar, irrespective of the number of constituent dwarfs, which is indicative of the generality of our methodology.

C. Scheduling

In Section III-B we describe the performance evaluation methodology with respect to the workloads. In this section, we discuss the details of scheduling each constituent part of our synthetic benchmarks on the disparate parts of CoC architectures. Our scheduling methods assume *oracle* prediction (as, e.g., in [24] or [22]), i.e., we assume a priori perfect knowledge of execution times of each dwarf on each type of CE within a CoC. Note that this work is not focused on scheduling, but in providing an early evaluation of CoC architectures, hence the oracle assumption.

Given the above, scheduling can be conducted as described below:

1) Baseline schedule: In this schedule we execute each synthetic benchmark on a single-type CE, e.g., all dwarfs in a synthetic benchmark run on the CPU, or all run on the GPU. We assume a single instance of the given CE, so dwarfs within a synthetic benchmark run sequentially one after another. Each dwarf by itself, though, can run in parallel on the given CE.

2) Schedule 1: In this schedule we execute each dwarf on the most suitable CE. We assume one CE active at any given time and that the rest are turned off or at a low-power state. This schedule is closer to reality, since it takes into account power and energy constraints. We can have a single instance of each CE or multiple instances, given space constraints as described in Section III-A.

3) Schedule 2: Like in Schedule 1, we execute each dwarf on the most suitable CE, but this time we allow multiple CEs to be active at any given time.

4) Schedule 3: In this schedule, we execute each dwarf on the *best available* CE (i.e., not necessarily on the fastest one, which would correspond to the local minimum). This schedule is feasible given use of an off-line scheduling algorithm, where we know the execution time of each dwarf, or a performance

prediction model. Such schedules may allow late start for a given dwarf, if such a schedule leads to a *globally optimal* solution. Like in *Schedule 1*, we explore cases with multiple instances of CEs, and like *Schedule 2*, we allow multiple CEs to be active simultaneously.

We give examples for each of the above schedules in Figure 1. In our evaluation (Section IV) we employ *Schedule 3* to identify the ideal performance we can obtain from a CoC architecture.

D. Assumptions

We start our study with a simplified model that incorporates various assumptions. Our goal is to provide an initial systematic study of the CoC concept based on existing hardware and to rationalize the benefits of a conceptual architecture like it. While a CoC may be unfeasible given the current state of the practice, we envision it can materialize in the (near) future, once further advances in chip manufacturing and other areas are made. Similar approaches with respect to assumptions are made in related work (Section VI), like [25], [22], [24]. We discuss such relevant required technology and system-level advances in Section V.

1) Data transfers: The architectures, as discrete devices, that comprise a CoC would require data transfers between a CPU host and the corresponding accelerator in a real OpenCL execution scenario. Such data transfers could in fact be potential bottlenecks also limiting maximum parallelism [26]. In the context of this work we deem them negligible, expecting a broader adoption of the *unified memory* trend (*a la* CPU and GPU in the context of an APU system).

2) Prediction/scheduling: In Section III-C we discuss scheduling. When multiple CEs, and especially CEs of different types, are available, it is essential that scheduling of workloads be done in an efficient manner, taking advantage of the specific characteristics of each CE type. This would require a form of pattern/application signature recognition. Such scheduling is non-trivial and adds an extra overhead on top of the actual execution time of a given workload. Within the confines of our study, we assume perfect prediction, given off-line scheduling and zero scheduling overhead.

3) Power: One of the main premises of CoCs, besides performance scaling, is power efficiency and energy savings by assigning workloads to the most appropriate CE type (as opposed to a generic *fat core*). The actual power budget of an architecture that consists of one (or more) of each of a CPU, MIC, GPU and FPGA could be prohibitive without employing advanced power management techniques. Our study focuses on the performance aspect of the CoC concept. Ensuring that CoCs are power-efficient is an important future research avenue beyond the scope of this work.

4) Parallelism within applications: Our synthetic benchmarks consist of applications that fall under the dwarfs classification, for reasons we discuss in Section III-B. For the purposes of our work, we assume that workloads are composed of independently parallelizable parts (i.e., the constituent dwarfs of each synthetic benchmark) following a request-level parallelism paradigm.

IV. Results

In this section we present the findings of our experiments that are based on the methodology outlined in Section III. The main variables in our experiments include:

1) **Synthetic benchmarks** (i.e., benchmarks comprising different combinations of dwarf N-mers).
2) **Cluster on a Chip (CoC) instantiations** (i.e., different combinations of types and numbers of CEs under consideration).

Despite the magnitude of the potential options search space, we attempt to provide relevant experiments and group the results we obtain in such ways so as to allow us to identify certain trends in a clear way and draw useful insights with respect to the following research questions:

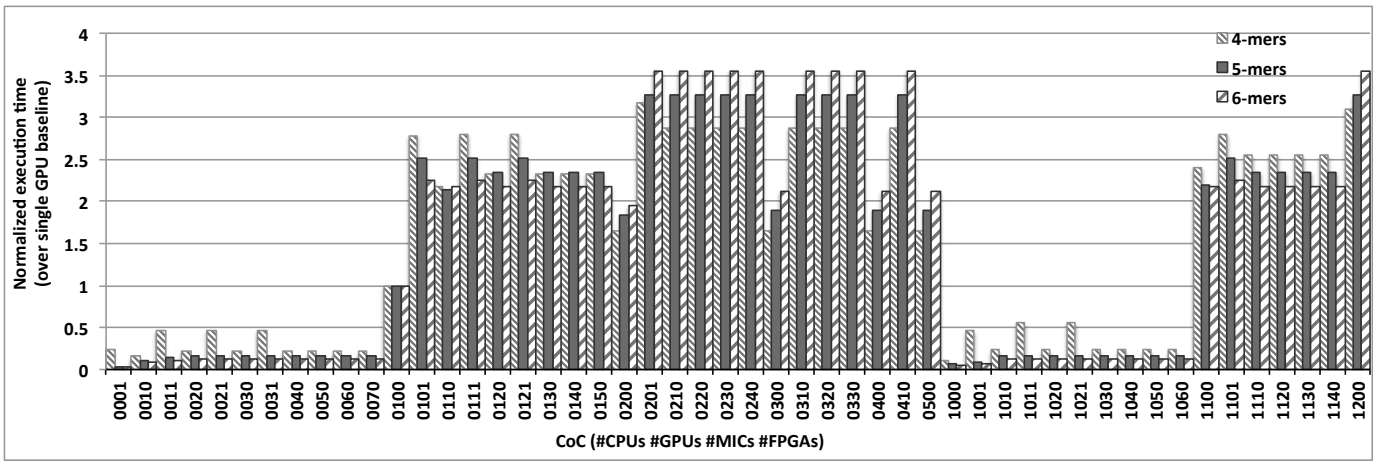
1) For a widely-varying set of dwarf combinations forming synthetic benchmarks what are the best combinations of CEs to form a CoC across the potential CoC spectrum? (Section IV-A)
2) What are the trade-offs between the CoCs' chip area and performance? What are the most efficient CoCs based on a performance per chip area metric? (Section IV-B)
3) What is the expected benefit of fusing an FPGA in three specific CoC instantiations and what are the CE usage trends with different combinations of dwarfs in smaller synthetic benchmarks? (Section IV-C)

Answering the above questions enables us to draw useful conclusions related to the main research question of *what is the best combination and number of CEs within a CoC* for a varied set of algorithmic patterns (as classified by the *dwarfs* concept).

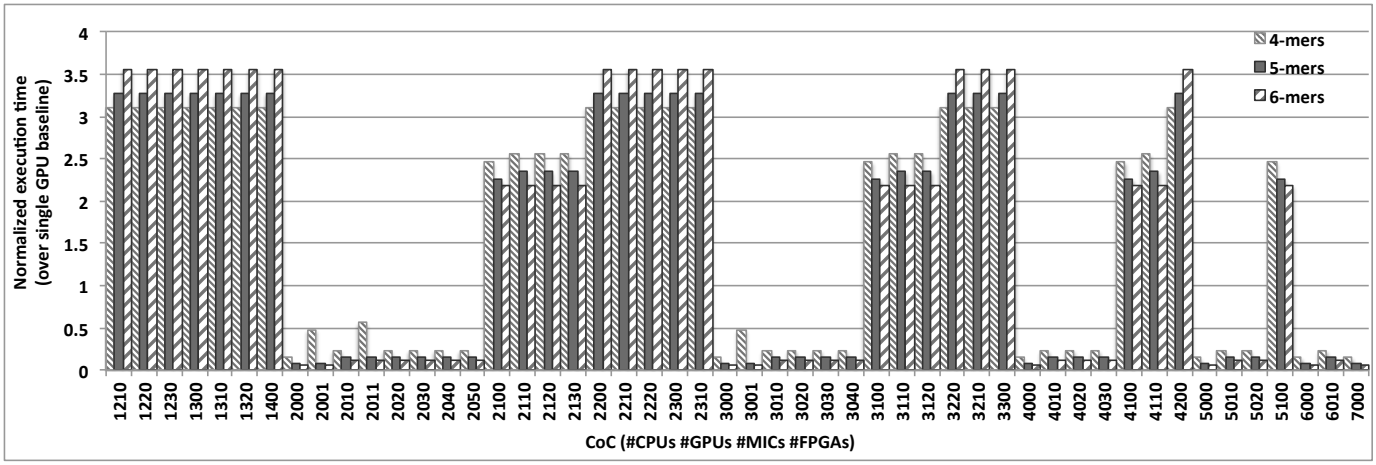
A. Performance of CoC Instantiations

Figure 2 presents the results for all possible CoC instantiations and for the average performance across all three categories of synthetic benchmarks built and scheduled as described in detail in Section III. Performance (execution time) is normalized over the performance of a CoC that contains a single GPU. In this case, separate dwarfs within a synthetic benchmark are executed one after the other on the single GPU, but each dwarf itself is executed in parallel within the GPU. Each CoC is identified by the number of its constituent CEs (number of CPUs, GPUs, MICs, FPGAs). For example, *1210* corresponds to a CoC that includes one CPU, two GPUs, one Intel MIC and zero FPGAs.

First, on a high-level we observe that for the majority of CoCs the achieved performance is irrespective of the number of dwarfs contained in a synthetic benchmark (*4-mers, 5-mers, 6-mers*). The cases where the observed performance is higher for the 4-mers case than in 5- or 6-mers are generally these where the number of CEs in a CoC is less than four. In these cases, one or two dwarfs in the 5- or 6-mer synthetic benchmarks need to run sequentially (with respect to the other



(a) Combinations 1-50



(b) Combinations 51-100

Fig. 2: Performance of Clusters on a Chip (CoC): Normalized execution time over single-GPU baseline for all possible combinations (100) restricted by the maximum number of Base Units (BUs), for three classes of synthetic benchmarks

dwarfs) on one of the available CEs, thereby increasing the overall execution time. The fact that the general performance trends remain the same, irrespective of the number of dwarfs in our synthetic benchmarks, indicates the suitability of our benchmarking methodology (Section III) for our purpose of evaluating next-generation heterogeneous architectures, in the form of CoCs.

Without loss of generality, we focus on the 6-mer case and provide a more detailed analysis of the observed results. Without a GPU, (normalized) performance is limited in the 0.033 to 0.12 range. A GPU is indeed an indispensable CE in any CoC. Note that a CoC with two GPUs (0200 in graph) provides a 1.952-fold speed-up over a CoC with one GPU. This may seem counter-intuitive at first (expecting a 2-fold speed-up), but one needs to remember that the execution time of different dwarfs within a synthetic benchmark vary and the way they can be scheduled onto the two GPUs (i.e., as a whole) may lead to such schedules. For similar reasons, CoCs with three, four or five GPUs only (0300, 0400, 0500) exhibit the same performance (2.123). Maximum performance is capped by the longest-running dwarf within the synthetic benchmarks. So if chip area (and consequently power) is a concern, a CoC with three GPUs is better than one with four or five. Or one may deem the 8.7% performance increase between using two and three (or four or five) GPUs negligible and elect a CoC with two GPUs only. Similar observations can be gleaned from Figure 2 for CoCs entailing other CEs, like MIC (e.g., cases 1210, 1220, 1230). The best performance (3.555) can be obtained using different combinations of CEs within a CoC. Again, these combinations exhibit varying chip area requirements. The above observations provide useful insights with respect to the performance per area ratio that we discuss further in Section IV-B.

B. Performance vs. Area Trade-offs

In this Section we attempt to provide another view of our experimental results focusing on the performance versus area trade-offs. Figure 3 presents the normalized execution time with respect to the chip area required for 300 points (100 CoCs

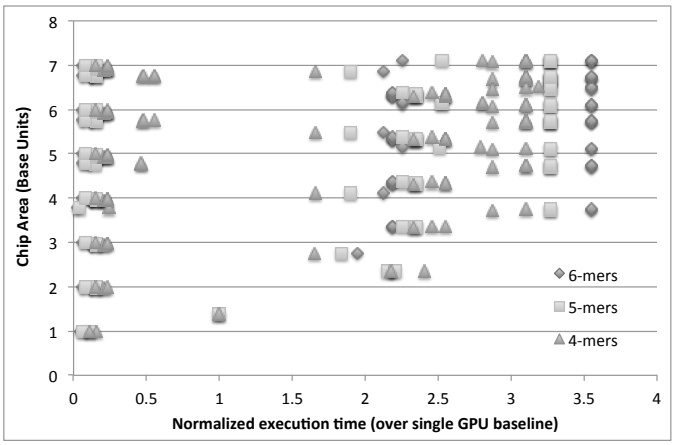


Fig. 3: Performance vs. Area Results

times three different set of experiments to include all possible 4-, 5-, and 6-mers). While the 300 points are not annotated, due to their large number, our purpose is to provide a high-level view of the trade-offs entailed, and which we only briefly discuss in Section IV-A. Specifically, one can observe three large clusters for each of the three experiments. The higher the number of dwarfs in each of the three experiments, the more concentrated the clusters are along the horizontal axis. For example, in the 6-mer synthetic benchmark experiments, most points fall on or near the 0.12, 2.18, 3.56 execution time (x-axis). Focusing on the 3.56 execution time case, there are 26 CoCs with varying chip area requirements (a lot of them overlapping in Figure 3). The best performance (3.56) with the least area (3.73 BUs) within this performance point is achieved on a CoC with two GPUs and one MIC. The same performance with the second best area is a CoC with two GPUs and one CPU (chip area of 3.74 BUs). The following best one (chip area of 4.7 BUs) comprises two GPUs and two MICs. The above observations are indicative of two cases: a) *different combinations of CEs in CoCs can achieve the same performance*, and b) *more CEs are not always beneficial*. In the former case, the choice of one CoC over the other may make more sense for non-technical reasons, too. For example, a 1-CPU + 2-GPUs CoC is a cheaper choice over a 1-MIC + 2-GPUs CoC (assuming the cost of a combination of CEs within a CoC is the same as the CE separately). Also, there are power implications depending on the CE selection (e.g., MIC TDP is generally higher than CPU TDP). In the latter case the higher number of CEs (especially of the same type, as discussed above) does not provide any benefit.

Minimizing chip area while keeping performance steady is important, due to manufacturing costs, cooling requirements, power, etc. As such, it is important to evaluate CoCs' performance with respect to occupied chip area. To do so we introduce a *performance per area* metric. We present the results in Figure 4 in descending order (the higher the better). This provides insight with respect to what combinations of CEs within a CoC provide the best performance per chip real estate. As in Section IV-A, we focus on the 6-mer synthetic benchmarks experiments. We show the CoCs that achieve a value of performance per area over 0.3 (all the rest CoCs score less than 0.06 and are not shown). As can be seen in Figure 4 the two best CoCs with respect to the performance per area metric are the ones we discussed in the previous paragraph (0210, 1200). The next best ones with a performance per area value over 0.9 (and still close to the maximum) incorporate one GPU and one MIC (0110), and one CPU and one GPU (1100), respectively. Notice that while these two CoCs are efficient with respect to our target metric, they are not optimal in terms of performance alone. This accentuates the trade-offs entailed and the fact that the choice of a CoC depends on the design goals. The most efficient CoCs, both from a performance per area perspective (Figure 4) and performance alone (Figure 2), contain at least one GPU. Intel MIC is also sufficiently present across the top performance per area CoCs, but always in combination with GPU(s).

C. Case Study: FPGAs in CoCs

In Section II we discuss practical implementations of telescoping architectures, such as fusing CPUs and GPUs in the form of an Accelerated Processor Unit (APU). Also, we heed towards the unification of reconfigurable fabric (i.e., *FPGA*) with CPU on-chip, in future implementations. This Section discusses the performance benefits of CoCs that include an FPGA CE. In contrast to Section IV-A, here we focus on simpler CoCs that may be more viable in the shorter term. As such, we focus on three sample cases, that is CoCs that include: a) one CPU and one FPGA, b) one CPU, one GPU and one FPGA, c) one CPU, one GPU, one MIC and one FPGA. As far as workloads are concerned, we use synthetic benchmarks that comprise dwarf kernels from the OpenDwarfs subset discussed in Section III minus GEM (to keep number of combinations tractable). However, in these experiments we focus on synthetic benchmarks that are composed of three dwarfs (i.e., 3-mers); specifically all combinations of three dwarfs possible out of the five dwarfs: NW, SRAD, BFS, CRC, CSR. Table V shows the results of the above experiments. For each of the three sample cases, we list the number of times dwarfs from each synthetic benchmark are run on the CPU, GPU, MIC or FPGA. Also, we indicate the speed-up obtained with the CoC at hand compared to the same configuration of CEs without the FPGA. For instance, in the 1-CPU + 1-FPGA case, and for the NW/SRAD/BFS synthetic benchmark, the CPU is used for one of the three constituent dwarfs of that benchmark and the FPGA for two. Also, the indicated speed-up is the performance obtained using this CoC compared to a CPU-only equipped CoC. Although we do not show the details of where dwarfs of each synthetic benchmark are scheduled, we provide details where necessary below.

In the first case, we examine the case where we incorporate an FPGA with a CPU on a chip. This would be conceptually similar to Intel's announced Xeon CPU + FPGA chip. Apparently, the FPGA is utilized on average twice as much as the CPU in the CoC (1.1 vs. 1.9). The performance

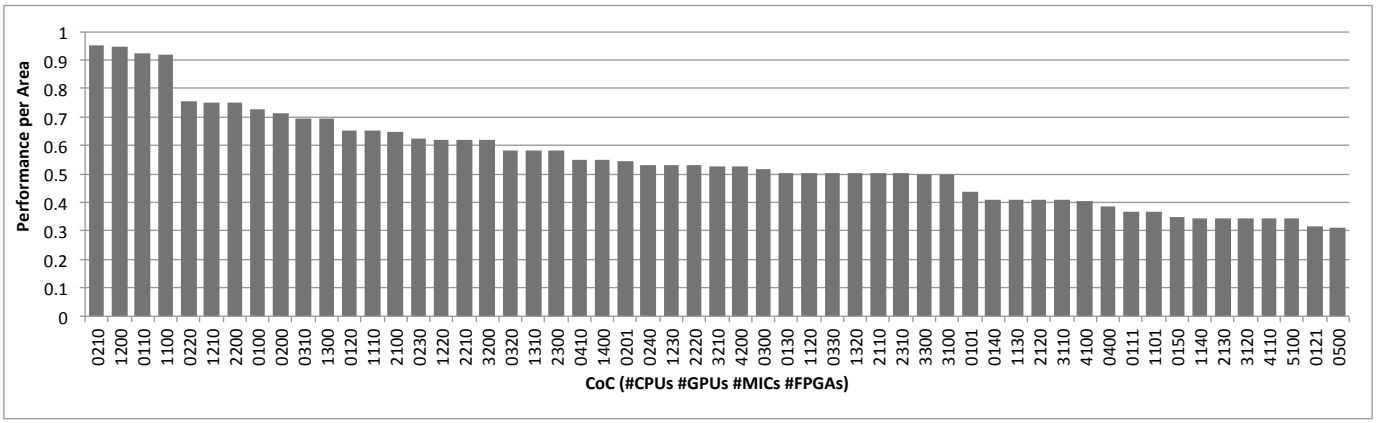


Fig. 4: Performance per Area Results

TABLE V: CoCs that include an FPGA: CE utilization and speed-up over corresponding CoC without FPGA

	1 CPU + 1 FPGA			1 CPU + 1 GPU + 1 FPGA				1 CPU + 1 GPU + 1 MIC + 1 FPGA				
Synthetic Dwarf	**#CPU**	**#FPGA**	**Speed-up w/o FPGA**	**#CPU**	**#GPU**	**#FPGA**	**Speed-up w/o FPGA**	**#CPU**	**#GPU**	**#MIC**	**#FPGA**	**Speed-up w/o FPGA**
NW / SRAD / BFS	1	2	1.09	1	1	1	1.41	1	1	0	1	1.2
NW / SRAD / CRC	1	2	1.15	1	1	1	1.08	1	1	0	1	1.08
NW / SRAD / CSR	1	2	1.03	2	1	0	1	2	1	0	0	1
NW / BFS / CRC	1	2	8.59	0	1	2	1.2	0	1	0	2	1.2
NW / BFS / CSR	2	1	3.46	1	1	1	1.16	1	1	0	1	1.16
NW / CRC / CSR	1	2	13.51	1	1	1	3.2	1	1	0	1	3.2
SRAD / BFS / CRC	1	2	1.2	0	1	2	1.43	0	1	0	2	1.2
SRAD / BFS / CSR	1	2	1.07	1	1	1	1.42	1	1	0	1	1.2
SRAD / CRC / CSR	1	2	1.14	1	1	1	1.08	1	1	0	1	1.08
BFS / CRC / CSR	1	2	7.9	1	1	1	1.2	1	1	0	1	1.2
Average:	**1.1**	**1.9**	**4.01**	**0.9**	**1**	**1.1**	**1.42**	**0.9**	**1**	**0**	**1.1**	**1.35**

obtained compared to a CPU-only CoC (practically a typical CPU) ranges from a meager 1.09- to a considerable 13.51-fold increase. In the average case, we observe a 4.01-fold speed-up. Specifically, as expected, all ten cases exhibit performance improvement. However, speed-up in six out of ten lies below 1.20x. These are the cases where the SRAD dwarf is present in the synthetic benchmark. Even for the best schedule, where the CPU is chosen for SRAD (3.47x than the FPGA), and the FPGA is used for the (faster) execution of the remaining two dwarfs in each case, the actual execution time of SRAD compared to the other two dwarfs dominates total execution time. While not shown in Table V, a CPU + GPU CoC exceeds a CPU + FPGA CoC performance for the workloads under consideration, for nine out of ten cases. Specifically, all six synthetic benchmarks that contain SRAD are considerably slower on the CPU + FPGA CoC. Of the four remaining ones the CPU + GPU CoC is faster in three out of four cases (by 1.12-1.20x), whereas in one case (NW/CRC/CSR) the CPU + FPGA CoC is faster by 1.88x. In that case, NW runs 3.2x faster on the FPGA, CRC runs 27.35x faster on the FPGA, and CSR runs 3.77x faster on the CPU. Since all three dwarfs can run in parallel and we have one CPU and one FPGA in the CoC, NW and CRC run one after the other on the FPGA while CSR runs on the CPU. The above shows how important heterogeneity (or hardware customization) is to cover special cases that may appear and completely nullify any expected performance gains.

The second case studies a CoC that contains an FPGA on a chip together with CPU and GPU CEs. That would correspond to an APU chip, as available today, with the addition of an FPGA on that same chip package. As we observe in Table V, despite the presence of a GPU, the FPGA is still utilized at least once in nine out of ten cases. That makes it obvious that the addition of an FPGA on chip would be beneficial for performance, if added alongside a CPU or a CPU and GPU. Even though the benefits of adding an FPGA to an APU chip are not as high as doing so on a CPU-only, they still provide an average of 1.42-fold performance increase. In our example synthetic benchmarks, the major benefit comes from scheduling SRAD on the GPU. As we mentioned the actual execution time for SRAD (even on the GPU) is comparably bigger than the rest of the dwarfs. This may mean that the scheduler picks the second fastest CE for a dwarf, as long as SRAD is assigned to the GPU. For example, in the NW/SRAD/BFS case, NW is assigned to the CPU and BFS to the FPGA, although these CEs are the second best for these dwarfs. As one can see in the CPU + FPGA and CPU + GPU + FPGA cases, adding an extra CE is always better. Even in the case a CE may not be the most suitable (i.e., fastest) for a given dwarf, it provides more scheduling possibilities that can lead to a better overall performance. While this may be worse with respect to power, clever power-saving techniques

could counter such effects (e.g., switching off CEs when not used, etc.)

In the third case, we consider the case of placing an FPGA together with a CPU, GPU, and MIC CEs. This experiment provides useful insights, that also tie back to Section IV-B. Specifically, it is an example of a case where an extra CE, which also occupies useful chip real estate, does not contribute any performance benefit. As we see, the MIC is not used at all across the synthetic benchmarks search space. in fact, the statistics reduce to the previous case (CPU + GPU + FPGA CoC). While the number of constituent workloads within our synthetic benchmarks (three) is less than the available CEs (four), it is still interesting to see that MIC is not used over one of the other CEs in *any* of the synthetic benchmarks. Similarly, comparing the CPU + GPU + MIC + FPGA CoC to the corresponding one without the FPGA, we identify only three cases where adding the FPGA benefits performance (the rest remains the same). These are the cases where the BFS dwarf is present and now scheduled on the FPGA, instead of MIC.

V. Discussion

Telescoping architectures in the form of CoCs bears many similarities with related work (Section VI) and can provide performance and power benefits. In this work, we focus on a first-order approach on the performance aspect of CoCs, identifying an upper bound on attainable performance, under certain assumptions. Telescoping architectures in the context of heterogeneous computing (i.e., CoCs) is a non-trivial effort. In order for CoCs to find practical implementation a number of issues needs to be addressed, some of which warrant a whole research area by themselves. Borkar and Chien discuss challenges regarding the future of microprocessors, many of which are relevant to a CoC, in more detail in [27]. Here, we provide a high-level overview of three issues related to our assumptions in Section III-D.

A. Programming CoCs:

One of the obstacles in democratizing heterogeneous computing has been learning a disparate set of programming languages and optimization techniques. Eventually, key hardware vendors introduced the OpenCL standard that allows programming CPUs, GPUs, Intel MICs and even FPGAs. The "write once, run anywhere" concept of OpenCL renders it ideal for a heterogeneous cluster on a chip. While OpenCL, as is, covers the *functional portability* aspect (i.e., correct results across architectures), there yet remains the issue of *performance portability* (i.e., equally fast performance across architectures, given the same code). It is expected that compiler technology advances, in tandem with auto-tuners, and optimized/customized libraries will assist towards bridging the performance portability gap across heterogeneous architectures.

B. Identifying Patterns/Signatures and Scheduling:

Scheduling dwarfs within an application to the CEs of a CoC in an efficient way requires identifying the dwarfs within that application. Research efforts include using performance counters to dynamically identify the best program phase to core matching ([28], [25]) or signature-driven approaches ([29], [30], [31]) and other scheduling techniques/run-time systems for workloads with various levels of parallelism (e.g., task-based, loop-based) on heterogeneous resources ([32], [33], [34]). Efficient recognition of dwarfs as constituent parts of real applications is challenging because applications may be composed by multiple dwarfs in a non-trivial way. Related work could be expanded to leverage CoC architectures and dwarf-based workloads.

C. Hardware-Related Issues: Chip Integration, Data Transfers and Memory Unification, Power:

For CoCs to materialize there is a need for major advances on the hardware level, specifically in the area of System on Chip (SoC) or System in Package (SiP) [35]. The latter is on the focus of ITRS [36], where complex, 3D SiP architectures are proposed on the road to *heterogeneous integration*. In both cases assembly, packaging, and most importantly interconnect of separate parts of a die (or chips in a SiP) will play a very important role in the fruition of CoC architectures. Issues like energy efficiency and energy proportionality are also of paramount importance. An interesting discussion on interconnects of future multi-processors, as well as power-related concerns, is given in [27]. We expect that the trend towards unifying the memory space and collocating heterogeneous cores on the same die (e.g., in the APU case) will continue allowing *more* and *different* instances of compute engines to perform synergistically within the same chip or package.

VI. Related Work

The concept of architectural unification in the context of homogeneous and heterogeneous computing is not new. In Section II we discuss examples from the commercial domain. Here, we focus on prior research that attempts to address the issue from a theoretical and practical standpoint.

In [37], Borkar discusses the prospect of many-core architectures that comprise hundrends or thousands of cores, as an answer to the unreasonable power envelop of integrating multiple complex cores on a die. In the proposed solution the cores correspond to simpler cores, as opposed to the "fat" cores (e.g., typical Xeon cores). This concept materialized with Intel Many Integrated Core (MIC) architecture (up to 61 cores in Knight's Corner and 72 cores in Knight's Landing co-processors). Borkar's work refers to homogeneous cores, as opposed to the following works, however it provides useful background related to fine-grain power management, memory bandwidth, on-die networks and system resiliency for many-core systems that are also relevant in the heterogeneous context.

In [25] Kumar et al. study single-ISA heterogeneous multi-core architectures. Specifically, they present a chip-level multi-processor with four Alpha cores of varying complexity and power consumption, with the same ISA. Their evaluation is

simulation-based, and includes certain assumptions (e.g., oracle scheduling, assumed architectural and power-related characteristics). One of the assumptions is that different phases of workloads can be assigned to the most appropriate core type, while the rest cores are switched off. The authors conclude that the proposed multi-core architecture can demonstrate up to three times higher energy efficiency with small sacrifices in performance.

Chung et al. [22] evaluate the prospects of single-chip heterogeneous computing from a stand-point similar to ours. Specifically, they try to answer the question: *"Does the future include custom logic, FPGAs, and GPGPUs?"* To address the question in the context of performance scaling and energy efficiency they investigate designs that place *unconventional cores* alongside traditional CPU cores. They focus on an analytical model that extends Hill and Marty's work [38] to include unconventional cores, i.e., custom logic, FPGAs, and GPUs. As in [25] and our work, their methodology includes assumptions with respect to the parallel workloads, like perfect scheduling and infinite divisibility.

In [24] Lukefahr et al. propose *Composite Cores*, an architecture that brings the notion of heterogeneity within a *single* core, with the main target being to reduce *switching overheads* – typically observed in other heterogeneous computing approaches. Composite Cores is based around the concept of big and little compute micro-engines that are characterized by high performance and high energy efficiency, respectively. Cycle accurate simulations show that their design achieves considerable energy savings (18%) at a minimal performance loss (5%).

Chien et al. [39] propose an alternative approach to heterogeneity and energy efficiency through *hardware customization*. Specifically, they argue about steering away from the traditional 90/10 optimization paradigm that guides architectural designs and addresses the "common case". Instead, they propose the *10x10* architecture that includes cores (or *micro-engines*) optimized for ten different 10% cases (where the number ten is arbitrarily chosen). Guha et al. [40] examine a broad selection of benchmarks from major benchmark suites in order to cluster applications by computation and memory behavior. Conceptually, each of these clusters would correspond to the "10%" cases optimally executed by each of the "10" corresponding micro-engines. [41] presents a case study, on a higher-level, of a "7x7" architecture, while [42],[43],[44] focus on the details of specific micro-engines within "10x10" architectures.

Our work, complements prior work in that it tries to address heterogeneous computing in the context of what we term *telescoping architectures*. In contrast to the above works, we omit a theoretical analysis or simulator-based approaches, in favor of a high-level practical approach, where we conceptually combine existing heterogeneous architectures and evaluate their performance, under certain assumptions. As shown in previous works above, assumptions in initial approaches for novel architectures are acceptable (but need to be refined and addressed in subsequent work). Clusters on a Chip (CoCs), as defined in Section III-A, in contrast to [25] have disparate ISAs. However, as opposed to some of the other works, they can be universally programmed using OpenCL. In our work, we employ dwarf-based benchmarks for evaluation purposes that allow a certain level of generalization of conclusions. The majority of the work, except "10x10"-related papers that use the notion of clustering of computation and communication patterns, uses standard benchmarks that may not allow such broader insights.

VII. Conclusions and Future Work

In this work we present an experimental study that extends the notion of telescoping architectures, that is the unification trend that has manifested throughout the history of computer architecture.This notion of *telescoping* allows us to envision, in a form of déjà vu, that similar advances that led from a commodity (heterogeneous) cluster in a machine room to a commodity cluster on a chip, will be repeated, now in the context of heterogeneity. To this end, we experiment with heterogeneous architectures that comprise multiple types and instances of CPUs, GPUs, Intel MICs, and FPGAs and attempt to provide an early study on the performance benefits of such heterogeneous clusters on a chip (CoC). We find that CoCs exhibit performance benefits overall, as expected, but also interesting characteristics with respect to their performance and constituent compute engines (CEs), as well as with respect to specific workloads.

The work presented on this paper constitutes an initial approach to the matter at hand and as such, relaxing some of the assumptions is needed in future studies. Beyond that, there are multiple exciting research avenues. First, we plan to extend our study on CoCs to consider a broader set of workloads, scheduling techniques, and examine the benefits of CoCs in harnessing parallelism on the dwarf-level *across* CEs within a CoC (in this work we focused on scheduling *whole* dwarfs in separate CEs). Second, to harness the full potential of heterogeneous architecture, we plan to study the performance benefits of CoCs when optimized implementations for dwarfs within a synthetic benchmark are available for each CE. Last, but not least, in future work we plan to factor in power- and energy-related considerations on top of performance.

References

[1] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, October 1974.

[2] F. J. Pollack, "New Microarchitecture Challenges in the Coming Generations of CMOS Process Technologies (Keynote Address)," in *Proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture (MICRO)*, 1999, p. 2. [Online]. Available: http://dl.acm.org/citation.cfm?id=320080.320082

[3] V. George, T. Piazza, and H. Jiang, "Technology Insight: Intel® Next Generation Microarchitecture Codename Ivy Bridge," in *Intel Developer Forum*, 2011.

[4] P. Hammarlund, A. J. Martinez, A. A. Bajwa, D. L. Hill, E. Hallnor, H. Jiang, M. Dixon, M. Derr, M. Hunsaker, R. Kumar, R. B. Osborne, R. Rajwar, R. Singhal, R. D'Sa, R. Chappell, S. Kaushik, S. Chennupaty, S. Jourdan, S. Gunther, T. Piazza, and T. Burton, "Haswell: The Fourth-Generation Intel Core Processor," *IEEE Micro*, vol. 34, no. 2, pp. 6–20, March 2014.

[5] A. Branover, D. Foley, and M. Steinman, "AMD Fusion APU: Llano," *IEEE Micro*, vol. 32, no. 2, pp. 28–37, March 2012.

[6] E. Sotiriades and A. Dollas, "A General Reconfigurable Architecture for the BLAST Algorithm," *The Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, vol. 48, no. 3, pp. 189–208, September 2007. [Online]. Available: http://dx.doi.org/10.1007/s11265-007-0069-2

[7] O. Knodel, T. B. Preuer, and R. G. Spallek, "Next-Generation Massively Parallel Short-Read Mapping on FPGAs," in *2011 IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, September 2011, pp. 195–201.

[8] V. M. Morales, P.-H. Horrein, A. Baghdadi, E. Hochapfel, and S. Vaton, "Energy-Efficient FPGA Implementation for Binomial Option Pricing Using OpenCL," in *Proceedings of the Conference on Design, Automation & Test in Europe (DATE)*, 2014, pp. 208:1–208:6. [Online]. Available: http://dl.acm.org/citation.cfm?id=2616606.2616862

[9] G. L. Zhang, P. H. W. Leong, C. H. Ho, K. H. Tsoi, C. C. C. Cheung, D. U. Lee, R. C. C. Cheung, and W. Luk, "Reconfigurable Acceleration for Monte Carlo Based Financial Simulation," in *Proceedings of the 2005 IEEE International Conference on Field-Programmable Technology (FPT)*, December 2005, pp. 215–222.

[10] A. Putnam, A. Caulfield, E. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Xiao, and D. Burger, "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services," in *ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, June 2014, pp. 13–24.

[11] P. Greenhalgh, "Big.little Processing with ARM Cortex-A15 & Cortex-A7," *ARM White Paper*, pp. 1–8, 2011.

[12] "Cyclone-V SoC," https://www.altera.com/products/soc/portfolio/cyclone-v-soc/overview.html.

[13] G. Chrysos, "Intel Xeon Phi Coprocessor (Codename Knights Corner)," in *Proceedings of the 24th Hot Chips Symposium*, 2012.

[14] "Top500," http://www.top500.org.

[15] "Green500," http://www.green500.org.

[16] "HPCG Benchmark," http://www.hpcg-benchmark.org.

[17] S. E. Sim, S. Easterbrook, and R. C. Holt, "Using Benchmarking to Advance Research: a Challenge to Software Engineering," in *Proceedings of the 25th International Conference on Software Engineering (ICSE)*, May 2003.

[18] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick, "The Landscape of Parallel Computing Research: A View from Berkeley," Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Tech. Rep. UCB/EECS-2006-183, 2006.

[19] "OpenDwarfs Benchmark Suite," https://github.com/vtsynergy/OpenDwarfs.

[20] M. Owaida, N. Bellas, K. Daloukas, and C. D. Antonopoulos, "Synthesis of Platform Architectures from OpenCL Programs," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2011.

[21] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick, "A View of the Parallel Computing Landscape," *Commun. ACM*, vol. 52, no. 10, pp. 56–67, October 2009. [Online]. Available: http://doi.acm.org/10.1145/1562764.1562783

[22] E. S. Chung, P. A. Milder, J. C. Hoe, and K. Mai, "Single-Chip Heterogeneous Computing: Does the Future Include Custom Logic, FPGAs, and GPGPUs?" in *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, December 2010, pp. 225–236.

[23] K. Krommydas, W.-c. Feng, C. D. Antonopoulos, and N. Bellas, "OpenDwarfs: Characterization of Dwarf-Based Benchmarks on Fixed and Reconfigurable Architectures," *Journal of Signal Processing Systems (JSPS)*, pp. 1–20, October 2015. [Online]. Available: http://dx.doi.org/10.1007/s11265-015-1051-z

[24] A. Lukefahr, S. Padmanabha, R. Das, F. M. Sleiman, R. Dreslinski, T. F. Wenisch, and S. Mahlke, "Composite Cores: Pushing Heterogeneity Into a Core," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, December 2012, pp. 317–328. [Online]. Available: http://dx.doi.org/10.1109/MICRO.2012.37

[25] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas, "Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance," in *Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA)*, June 2004, pp. 64–75. [Online]. Available: http://dl.acm.org/citation.cfm?id=998680.1006707

[26] M. Daga, A. Aji, and W.-c. Feng, "On the Efficacy of a Fused CPU+GPU Processor (or APU) for Parallel Computing," in *2011 Symposium on Application Accelerators in High-Performance Computing (SAAHPC)*, July 2011, pp. 141–149.

[27] S. Borkar and A. A. Chien, "The Future of Microprocessors," *Commun. ACM*, vol. 54, no. 5, pp. 67–77, May 2011. [Online]. Available: http://doi.acm.org/10.1145/1941487.1941507

[28] M. Becchi and P. Crowley, "Dynamic Thread Assignment on Heterogeneous Multiprocessor Architectures," in *Proceedings of the 3rd Conference on Computing Frontiers (CF)*, May 2006, pp. 29–40. [Online]. Available: http://doi.acm.org/10.1145/1128022.1128029

[29] D. Shelepov, J. C. Saez Alcaide, S. Jeffery, A. Fedorova, N. Perez, Z. F. Huang, S. Blagodurov, and V. Kumar, "HASS: A Scheduler for Heterogeneous Multicore Systems," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 2, pp. 66–75, Apr. 2009. [Online]. Available: http://doi.acm.org/10.1145/1531793.1531804

[30] A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia, and A. Purkayastha, "A framework for performance modeling and prediction," in *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, ser. SC '02. Los Alamitos, CA, USA: IEEE Computer Society Press, 2002, pp. 1–17. [Online]. Available: http://dl.acm.org/citation.cfm?id=762761.762785

[31] L. Carrington, M. M. Tikir, C. Olschanowsky, M. Laurenzano, J. Peraza, A. Snavely, and S. Poole, "An Idiom-finding Tool for Increasing Productivity of Accelerators," in *Proceedings of the International Conference on Supercomputing (ICS)*, 2011, pp. 202–212. [Online]. Available: http://doi.acm.org/10.1145/1995896.1995928

[32] D. Koufaty, D. Reddy, and S. Hahn, "Bias Scheduling in Heterogeneous Multi-core Architectures," in *Proceedings of the 5th European Conference on Computer Systems (EuroSys)*, 2010, pp. 125–138. [Online]. Available: http://doi.acm.org/10.1145/1755913.1755928

[33] J. Planas, R. M. Badia, E. Ayguadé, and J. Labarta, "SSMART: Smart Scheduling of Multi-architecture Tasks on Heterogeneous Systems," in *Proceedings of the Second Workshop on Accelerator Programming Using Directives (WACCPD)*, 2015, pp. 1:1–1:11. [Online]. Available: http://doi.acm.org/10.1145/2832105.2832109

[34] T. Scogland, W. Feng, B. Rountree, and B. de Supinski, "Coretsar: Core task-size adapting runtime," *Parallel and Distributed Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2014.

[35] S. K. Lim, "Physical design for 3D system on package," *IEEE Design Test of Computers*, vol. 22, no. 6, pp. 532–539, November 2005.

[36] "ITRS Public Website," http://www.itrs2.net/.

[37] S. Borkar, "Thousand Core Chips: A Technology Perspective," in *Proceedings of the 44th Annual Design Automation Conference (DAC)*, June 2007, pp. 746–749. [Online]. Available: http://doi.acm.org/10.1145/1278480.1278667

[38] M. D. Hill and M. R. Marty, "Amdahl's Law in the Multicore Era," *Computer*, vol. 41, no. 7, pp. 33–38, July 2008.

[39] A. A. Chien, A. Snavely, and M. Gahagan, "10x10: A General-purpose Architectural Approach to Heterogeneity and Energy Efficiency," *Procedia Computer Science*, vol. 4, pp. 1987 – 1996, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050911002754

[40] A. Guha, P. Cicotti, A. Snavely, and A. A. Chien, "The 10x10 Foundation for Heterogeneity: Clustering Applications by Computation and Memory Behavior," *University of Chicago, Tech. Rep. TR-2012-01*, 2012.

[41] A. A. Chien, T. Thanh-Hoang, D. Vasudevan, Y. Fang, and A. Shambayati, "10x10: A Case Study in Highly-Programmable and Energy-Efficient Heterogeneous Federated Architecture," *SIGARCH Comput. Archit. News*, vol. 43, no. 3, pp. 2–9, December 2015. [Online]. Available: http://doi.acm.org/10.1145/2856113.2856115

[42] Y. Fang, T. T. Hoang, M. Becchi, and A. A. Chien, "Fast Support for Unstructured Data Processing: The Unified Automata Processor," in *Proceedings of the 48th International Symposium on Microarchitecture (MICRO)*, December 2015, pp. 533–545. [Online]. Available: http://doi.acm.org/10.1145/2830772.2830809

[43] D. Vasudevan and A. A. Chien, "The Bit-Nibble-Byte MicroEngine (BnB) for Efficient Computing on Short Data," in *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI (GLVLSI)*. New York, NY, USA: ACM, May 2015, pp. 103–106. [Online]. Available: http://doi.acm.org/10.1145/2742060.2742106

[44] T. Thanh-Hoang, A. Shambayati, and A. A. Chien, "A Data Layout Transformation (DLT) Accelerator: Architectural Support for Data Movement Optimization in Accelerated-centric Heterogeneous Systems," in *Proceedings of the 2016 Design, Automation and Test in Europe (DATE)*, March 2016.