POLITECNICO
MILANO 1863

Software Defined Architectures for Data Analytics

Vito Giovanni Castellana, Marco Minutoli, Antonino Tumeo, Marco Lattuada, Pietro Fezzardi, Fabrizio Ferrandi

Vito Giovanni Castellana, Marco Minutoli, Antonino Tumeo, Marco Lattuada, Pietro Fezzardi, and Fabrizio Ferrandi. Software defined architectures for data analytics. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, ASPDAC '19, pages 711–718, New York, NY, USA, 2019. ACM

The final publication is available via http://dx.doi.org/10.1145/3287624.3288754

©ACM, 2019. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in Proceedings of the 24th Asia and South Pacific Design Automation Conference http://doi.acm.org/10.1145/3287624.3288754

Software Defined Architectures for Data Analytics

Vito Giovanni Castellana
Pacific Northwest National Laboratory
vitoGiovanni.castellana@pnnl.gov

Marco Minutoli
Pacific Northwest National Laboratory
marco.minutoli@pnnl.gov

Antonino Tumeo
Pacific Northwest National Laboratory
antonino.tumeo@pnnl.gov

Marco Lattuada
Politecnico di Milano — DEIB
marco.lattuada@polimi.it

Pietro Fezzardi
Politecnico di Milano — DEIB
pietro.fezzardi@polimi.it

Fabrizio Ferrandi
Politecnico di Milano — DEIB
fabrizio.ferrandi@polimi.it

ABSTRACT

Data analytics applications increasingly are complex workflows composed of phases with very different program behaviors (e.g., graph algorithms and machine learning, algorithms operating on sparse and dense data structures, etc). To reach the levels of efficiency required to process these workflows in real time, upcoming architectures will need to leverage even more workload specialization. If, at one end, we may find even more heterogenous processors composed by a myriad of specialized processing elements, at the other end we may see novel reconfigurable architectures, composed of sets of functional units and memories interconnected with (re)configurable on-chip networks, able to adapt dynamically to adapt the workload characteristics. Field Programmable Gate Arrays are more and more used for accelerating various workloads and, in particular, inferencing in machine learning, providing higher efficiency than other solutions. However, their fine-grained nature still leads to issues for the design software and still makes dynamic reconfiguration impractical. Future, more coarse-grained architectures could offer the features to execute diverse workloads at high efficiency while providing better reconfiguration mechanisms for dynamic adaptability. Nevertheless, we argue that the challenges for reconfigurable computing remain in the software. In this position paper, we describe a possible toolchain for reconfigurable architectures targeted at data analytics.

CCS CONCEPTS

• **Computer systems organization → Reconfigurable computing**;

KEYWORDS

Reconfigurable Computing, CGRAs, FPGAs, HLS

ACM Reference Format:
Vito Giovanni Castellana, Marco Minutoli, Antonino Tumeo, Marco Lattuada, Pietro Fezzardi, and Fabrizio Ferrandi. 2019. Software Defined Architectures for Data Analytics. In *ASPDAC '19: 24th Asia and South Pacific Design Automation Conference (ASPDAC '19), January 21–24, 2019, Tokyo, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3287624.3288754

ACM acknowledges that this contribution was authored or co-authored by an employee, contractor, or affiliate of the United States government. As such, the United States government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for government purposes only.

ASPDAC '19, January 21–24, 2019, Tokyo, Japan
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6007-4/19/01…$15.00
https://doi.org/10.1145/3287624.3288754

1 INTRODUCTION

Data analytics applications increasingly employ complex workflows that couple graph and machine learning methods. For example, predictive logistics pipelines employ associative learning algorithms to learn spatio-temporal graphs of transport movements then use the graphs to predict activity in busy zones. A radar imagery classification pipeline combines common machine learning stages (detection, feature extraction and tracking, classification) with Deep Belief Networks to learn features in high-dimensional data, and Hierarchical Bayesian Nonparametric machine learning methods to learn and recognize known and unknown targets in real time. While both are data intensive, the graph and machine learning methods involve algorithms with different behaviors that require distinct processor architecture trade-offs to achieve high performance and low power consumption. Reconfigurable architectures are attractive platforms for supporting the heterogeneity of modern analytic workflows and data-dependent optimizations, particularly for those DoD-related environments that cannot afford the level of investments and hardware development times for fully custom Application Specific Integrated Circuits (ASICs). Various solutions exploiting fine-grained (re)configurable devices, such as Field Programmable Gate Arrays (FPGAs), have been developed. Tools and approaches to generate hardware description languages synthesizable on FPGAs from specifications in high-level languages such as C have been explored in research for many years and recently have experienced increased commercial success (e.g., Xilinx Vivado High-Level Synthesis (HLS), Altera OpenCL HLS, Micron Convey OpenHT) as application programming interfaces, such as OpenCL or OpenMP, allow annotating code to simplify the architecture generation process. In fact, the hints provided by such frameworks afford additional information, allowing the synthesis tool to make assumptions on code fragments that would be not easily analyzable by employing modern compiler analysis and optimization passes. While C-to-H synthesizers represent an increasingly impactful category of tools to accelerate development of customized designs, the majority of their optimization focuses on extracting instruction-level parallelism and generating kernels starting from codes characterized by high-arithmetic intensity, vectorizable loops, easily partitionable datasets, and high locality. Instead, data analytics pipelines integrating property graph methods and machine learning typically present unpredictable, data-dependent, memory accesses, operations on pointer or linked-list-based data structures, sparse matrices, high synchronization intensity, and frequent interplay among sparse and dense data types. However, they

typically have large amounts of task-level parallelism, providing interesting opportunities to scale performance. Conversely, from the architectural point of view, fine-grained reconfigurable devices, such as FPGAs, need a non-trivial amount of architectural design to build register transfer level templates for functional units, interfaces, and control logic (centralized finite state machines or dataflow-like distributed controllers) that support the tools in the synthesis and architecture generation process. These templates must support the software toolchain adequately, as well as be specialized for the target workloads. The specialization of the templates, in turn, may then influence the programming model and require additional types of code conventions to enable the toolchain to effectively perform analysis and optimizations. Additionally, while partial dynamic reconfiguration at runtime is possible on these devices, reconfiguration times are typically high due to limited bandwidth, the time to modify fine grained bit streams, and placement of the components. Coarse-grained reconfigurable arrays (CGRAs), providing sets of functional units with different levels of specialization, memory components of various types, modifiable memory planes, and configurable interconnects, offer the hardware features to execute diverse workloads at high efficiency and, at the same time, the opportunity to provide quick and effective reconfiguration mechanisms for dynamic adaptation. While current research and commercial approaches seem to distinctly identify reconfigurable (software-defined) architectures as a viable opportunity to increase efficiency when processing data-dependent workloads, there still are key gaps in the software stack to enable exploration of the required hardware/software optimizations and data-aware adaptability without crippling programmer productivity. In this position paper, we discuss the organization of a toolchain for Software Defined Architectures for Data Analytics ($SO(DA)^2$). We discuss the various layers of such a toolchain and provide references to some of the seminal work we have done in the C-to-H synthesis area to better support Data Analytics applications.

2 RELATED WORK

Reconfigurable computing has long been explored. Since the mid-1980s, FPGAs have been an appealing platform for low-volume mission-critical systems that could not afford development and production of Application Specific Integrated Circuits (ASICs). Favorable performance-per-watt trade-offs with respect to other specialized accelerators targeting flop-intensive applications (general-purpose GPUs; manycores with wide vector units such as the Xeon Phi) have recently reignited significant interest in them [41]. For example, accelerators for molecular dynamics [47], genomics [2], and machine learning [13, 37], which do not require full double floating-point precision for their computation (and, in many cases, not even single), can greatly benefit from finely customized designs. However, programming abstractions, languages, and compute models have been a key limitation for the broad adoption of reconfigurable hardware. Hardware design languages (HDLs), such as VHDL or Verilog, make them only accessible to hardware designers. As such, research has looked for solutions to raise the abstraction level. Configurability requires abstractions that, even if succinct, should allow verification prior to the hardware implementation, as well as be amenable to efficient compilation to take advantage of specialization. This is even more critical for FPGAs, which also require the design tools to perform mapping, placing, and routing of the design onto the configurable blocks.

High-Level Synthesis (HLS) approaches, able to generate (semi)-automatically descriptions in HDLs starting from high-level languages, have always been an important part of the research for reconfigurable designs [15, 33]. The appearance of new synthesis tools, based on parallel programming interfaces, also has reinforced interest regarding FPGAs in HPC environments. Among them, there are solutions that exploit OpenCL, such as Xilinx Vivado HLS and Altera/Intel SDK for OpenCL, or OpenACC. such as Oak Ridge's OpenARC [4]. However, OpenCL, because of its data-parallel nature, and OpenACC, based on its offload style, expose too many architectural details, limiting productivity and performance portability across different types of reconfigurable devices. Research has identified OpenMP as a suitable programming interface for HLS. Various works have identified applicable pragmas and preliminary guidelines [17], and proposed extensions [7], even translating to C-hardware-oriented languages [26]. Some solutions target hybrid architectures (general-purpose processors with FPGAs) for embedded system design [14], but they only implement offload models without considering hierarchical (nested) parallelism when tasks are actually synthesized on the FPGA. Moreover, the approach has limited support for complex external memory models. In general, while these solutions represent significant progress, they remain tied to conventional HLS methodologies. They perform well with digital signal processing workloads, which mainly expose instruction-level parallelism, and present highly vectorizable loops that operate on small, easily partitionable, datasets with high data locality. They do not at all consider requirements to scale to multiple devices or multiple nodes. We have significantly extended for OpenMP-like annotation and tasking, introducing templates based around a distributed controller to efficiently exploit nested thread level parallelism [30], a memory interface that supports parallel memory subsystems and enables implementing atomic memory operation [11], and dynamic task scheduling approaches to efficiently execute heavily unbalanced workloads [29].

FPGAs still expose many limitations in the hardware and, crucially, in the software to be effectively employed for productive acceleration of data analytics applications. They require describing the finest details of each functional unit (at the bit-level) and have a number of fixed functional units that while increasing from one device generation to the other, do not reach the same peak floating-point performances of other specialized accelerators. FPGAs also still have reconfiguration times that are sufficiently long to adversely impact execution latencies when performing reconfiguration at runtime. While they are improving, their programming abstractions still require developers to finely tune their code to generate efficient hardware descriptions. However, as FPGAs progress towards integrating more specialized units (digital signal processing blocks, advanced memory interfaces, or complete processors), general-purpose processors move towards a more fine-grained level of configurability, exploiting aspects such as subword parallelism, narrower or specialized floating-point units, configurable memory planes and interfaces, and even reconfigurable logic. This suggests a convergence toward CGRAs. A variety of CGRAs

has been proposed. Generically defined as reconfigurable architectures that dynamically adapt the datapath at runtime to the application, CGRAs provide an architecture with computational blocks at a granularity of the functional unit or higher (but as large or smaller than a core) that needs to be programmed spatially, i.e., where a function is assigned to a specific set of resources for some amount of time, for example, until the end of the program or when such a function terminates. Designs range from architectures with simple general-purpose cores with (software) configurable interconnect, such as MIT's RAW [40], to more proper CGRAs that tightly integrate a general-purpose core with an array of functional units (typically identical arithmetic-logic units), such as GARP [8], Piperench [22], ADRES [27], Tartan [31], and DySER [23]. The Plasticine [39] spatially reconfigurable design combines pattern compute units (PCUs), hierarchically composed of a reconfigurable pipeline with multiple stages of SIMD functional units, and pattern memory units (PMUs), simplifying mapping of inner loops and feedback edges to the hardware and enabling execution of applications expressed as parallel patterns. Ongoing efforts in Path Forward projects and the DARPA Electronic Resurgence Initiative (ERI) demonstrate the intense interest around CGRAs.

Along with coarse-grained reconfigurability, architectures also are progressively integrating more fine-grained types of configurability. For example, a number of accelerators allow for deciding at runtime if on-chip memories should act as software-managed scratchpads or caches. When activated through specific hardware knobs, even dynamic voltage and frequency scaling (DVFS), which has been broadly explored at all levels of the software stack with models and approaches working at runtime [25], also can be considered a fine-grained dynamically reconfigurable element.

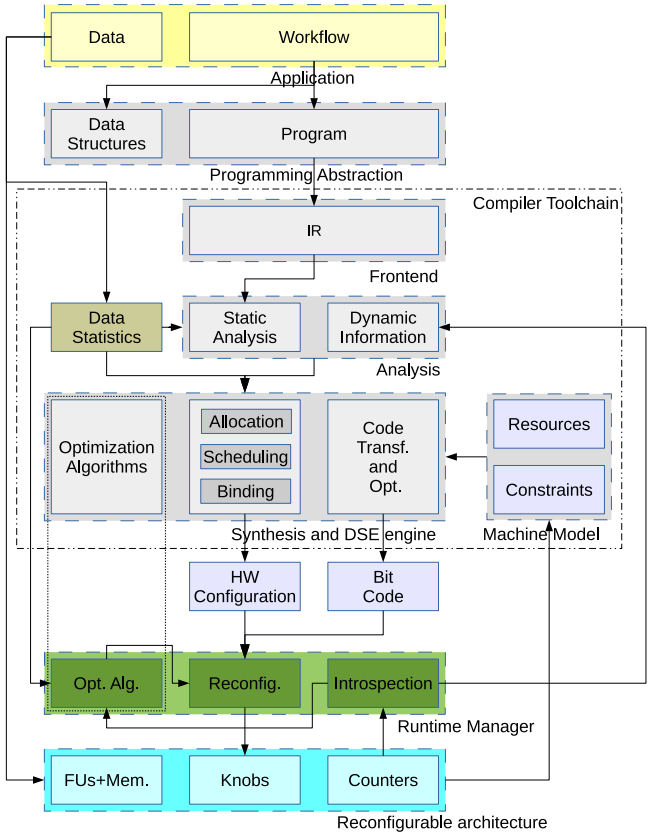

Figure 1: Overview of our proposed $SO(DA)^2$

3 PROPOSED FLOW

Figure 1 shows a high-level overview of the $SO(DA)^2$ toolchain. Reconfigurable designs provide the opportunity to be *spatially* programmed. Thus, any integrated toolchain should be able to efficiently identify task-level parallelism (TLP) beside instruction-level parallelism (ILP), and, given a limited amount of resources, reason about the potential trade-offs among the two to maximize objective metrics for the implementation, such as performance and power consumption. An important area of exploration is if extending existing parallel programming models or domain specific languages could be sufficient, or if new solutions are needed, with obvious tradeoffs in enabling quick portability of existing code-bases. The high-level abstraction, in any case, should enable co-optimization of hardware and software. In our current toolchain, we are supporting GCC [3] as a frontend, thus naturally supporting existing analyses, intermediate representation views (e.g., control and dataflow graphs, or program dependency graph), and existing compiler optimizations. Additionally, as a way to indicate parallelism, we are exploiting OpenMP annotations.

A core novelty of the proposed approach resides in devising new synthesis and design exploration methods to explore software and hardware configurations simultaneously, both statically and with dynamic feedbacks. These needs to be integrated in a Design Space Exploration and Synthesis (DSES) engine that leverages the information provided by the high-level abstractions and supports appropriate representations. The objective is identifying specific parallel patterns and explore trade-offs among multiple optimization objectives (e.g., number of unrolled iterations and number of concurrent execution flows in loops). Another key aspect is the support of a hierarchical and parametric machine model, to enable recognition of opportunities for mapping operations to units supporting different precision/accuracy (for performance and energy reasons) or specialized units (e.g., tensor cores able to perform matrix multiply-accumulate operations). Partial dynamic reconfiguration also needs to be considered a key dimension for the design space exploration process. As an application is partitioned in different work units for different phases, these needs be progressively mapped on the hardware using a subset of the available resources and replaced as soon as they terminate. A work unit that replaces another may be mapped on fewer resources than typically possible given its parallelism, but, because those are actually available, the application will be able to proceed and execute faster. A comprehensive framework will require a runtime manager to schedule the work units on the hardware substrate and to orchestrate the datapath reconfiguration across multiple devices and nodes for different application phases. Because the upcoming reconfigurable architectures will present more complex memory hierarchies, it will be critical to place and move data efficiently. As such, it would be natural,

in toolchain focused at supporting data analytic workflows, to support data-aware analysis to inform the optimization. Such an an analysis would probably require intrinsic knowledge of the data and application domain (thus, benefiting from the high-level abstractions), or runtime information. The architectures also are expected to provide fine-grained adaptability (bus widths, data fetch size, cache line size, configurable on-chip memories, precision of functional units) that the runtime manager, exploiting appropriate instrumentation mechanisms (e.g., introspection and performance counters), should be able to change dynamically through provided hooks (knobs), employing stochastic optimization algorithms and targeting power and performance metrics. Finally, the same mechanisms should provide a way to feedback dynamic information to the higher layers of the stack for hardware-in-the-loop optimization.

3.1 Programming Abstractions

Domain-specific languages provide data structures and language constructs custom-designed for a domain's computational methods. They support the parallel programming model implied by the method's parallel operations, control, and data movement. Consequently, domain experts enjoy in a highly productive programming environment that still enables compilers and schedules to generate efficient code. Unfortunately, modern data analytic workflows are a complex composition of methods, data types, and parallel models. Pertinent domains are linear and relational algebra, graphs, machine learning, and statistics. Data types include dense and sparse matrixes, tensors, tables, (property) graphs, sets, and trees. Parallel methods span embarrassingly parallel tasks, loosely coupled tasks, K-step methods, data parallelism, recursive methods, and speculative computations with and without rollback. A significant question is thus if a new DSL is required, or perhaps it would be better to look at opportunities to extend current DSLs. There is a variety of DSLs that either target graph methods and machine learning, and no one appears to target them at the same time.

Our preliminary work has focused on DSLs for graph methods. We implemented a SPARQL compiler (a query language for datasets in the Resource Description Framework) [10] that generates graph pattern matching routines in C, and defined GraQL [12] an extension of SQL to support graph walks when tables are combined to provide a graph view of the data. Reconfigurable hardware must be programmed spatially. Thus, the toolchain should be able to efficiently identify task-level parallelism (TLP) beside instruction-level parallelism (ILP), and, given a limited amount of resources, reason about the potential trade-offs among the two to maximize the objective metrics. Annotations that explicitly express concurrency can provide compilers and synthesizers with ways to better identify parallel patterns. The area of C-to-Hardware compilers long focused on the extraction of ILP (a classical example is loop unrolling), which is abundant in the typical digital signal processing applications (DSP) that have been originally targeted. With the advent of larger devices and new classes of data-analytics applications, methods exploiting CUDA [34], OpenCL [42], and OpenACC [35] to express higher levels of parallelism than ILP have been explored for hardware synthesis. However, they are narrow in scope: OpenACC is focused only to offload models, while CUDA or OpenCL massively threaded architectures, embed in their syntax low level assumptions on the target architecture and adapts well mainly to fine-grained data-parallel algorithms which mostly operate in a SIMD fashion. Crucially, they do not work well with nested and irregular parallelism, which are instead significant in data analytics applications. Our current C-to-RTL synthesis tool for FPGA, optimized for Data Analytics, supports, in fact, OpenMP annotated code. Nevertheless, with novel CGRAs, it will be necessary to explore even more the trade-offs between TLP and ILP, depending on number and type of computation resources available. Thus, annotations need sufficient expressivity to let the tool reason about the best trade-off. The opportunity to reconfigure at runtime also makes it possible to further trade off TLP and ILP to cope with actual data behaviors.

3.2 Frontend and Intermediate Representations

Our current synthesis infrastructure employs GCC as its frontend and, through the OpenMP annotations, provides a *hierarchical* task graph (HTG) view of the program. Our envisioned $SO(DA)^2$ approach for novel reconfigurable designs will also take advantage of a conventional compiler frontend, exploiting typical compiler analysis (dataflow graph, control flow graph, program dependency graph) and optimizations (inlining, unrolling, vectorization, loop transformations, polyhedral transformation). $SO(DA)^2$ will also support a refined version of HTGs [20, 38]. HTGs can be directly assumed from the actual application code. HTGs are an especially versatile and efficient way to capture program characteristics, including dataflow and parallelism information. Intuitively, nodes in an HTG represent tasks, and edges represent data/control dependencies among tasks. However, HTG nodes can be expanded as tasks may comprise lower-level HTGs or, at the finest granularity, individual operations. Using HTGs makes it is possible to analyze a program at different granularities. $SO(DA)^2$ optimizations also will be data aware, driven by dynamic information fed back from the architecture (profile-driven optimization). Dynamic information can be related to performance (e.g., operations and performance counters), as well as to energy, depending on the introspection mechanisms provided by the architecture. However, given the application's data-dependent behavior, the optimization process may require targeting for average cases with predefined datasets and applications and could possibly use machine learning approaches, where the parameters are learned from multiple profiling runs with training datasets.

3.3 Design Space Exploration and Synthesis (DSES) Engine

Machine model. $SO(DA)^2$ will employ machine models, comprising a resource library and the set of constraints imposed by the target architecture. The resource library will provide methods to describe the components of the target architecture at different levels of detail. Components that need to be described range from general-purpose cores with different features (e.g., different instruction set architectures, vector units, floating-point units) to specialized cores

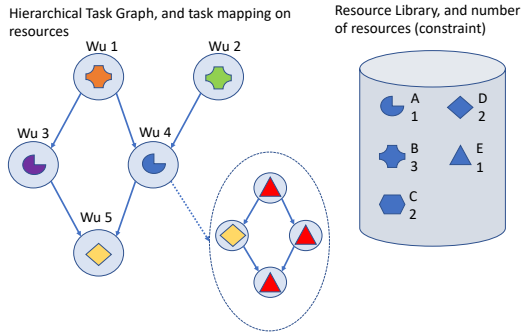

Figure 2: Example of HTG with Resource Library. Nodes and resources are colored as Work Units (WUs) are assigned to resources

(digital signal processing and vector processors) to memory elements (caches, on-chip memories, different types of volatile and non-volatile memories) to functional units (for CGRAs that may provide arithmetic logic units with different functionalities and specialized operators, such as tensor units) and to interconnect mechanisms (point-to-point connections, buses, switched matrices). The machine model format must describe component parameters that may be set during offline synthesis and changed as different phases of the program are executed and the configuration vectors are loaded onto the target device (e.g., precision for floating-point units that allow variable precision to reduce power consumption), as well as fine-grained reconfigurable parameters that a runtime layer should be able to tune during device operations. The machine model must include device constraints, such as the number of available functional units and the size of the memory components. It also will include overall target metrics, for example, maximum power or aggregate external memory bandwidth. The machine model must also include *cost* metrics for each component, depending on their type.

Synthesis. Given the HTG representation and machine model, the $SO(DA)^2$ synthesis engine will be able to associate work units (task and HTG nodes) to architectural resources (resource allocation), determine the scheduling of the work units, and bind the work units to the functional units. Depending on the specific architecture characteristics, the scheduling may only represent ordering of operations with their dependences. Partial or pure dataflow hardware substrates could provide mechanisms to actually start the work unit as soon as dependences are satisfied. The synthesis engine also will be able to generate the glue code for the runtime layer if the scheduling operations need to be managed through a general-purpose core. The HTG representation provides a simple abstraction to match a high-hierarchy node with a specialized functional unit (e.g., a vector or tensor unit) or, if a matching is not possible, to recursively expand nodes until a match with simpler functional units is found. The approach adapts well to proper CGRAs and mixed (heterogeneous) designs. If the target architecture also provides general-purpose cores, the mapping may result in optimized compilation of the operations described in the HTG node with the opportunity to optimize the data movement operations identified by data and control dependences.

Figure 3: Chromosome encoding for combined exploration of resource allocation, register memory allocation, scheduling, and interconnect allocation

The scheduling and resource binding processes will play a vital role in identifying opportunities for reconfiguration. In fact, if the hardware substrate requires a complete reconfiguration, the DSES engine could cut the HTG appropriately in the most convenient sections, exploiting the annotation information on program phases and work units but also analyzing similarities and reuse in the execution path. If the hardware substrate supports partial dynamic reconfiguration and/or dataflow-like processing, the DSES will be able to explore other aspects, such as granularity of the tasks, and, in collaboration with the runtime manager, opportunities to fit work units on resources that dynamically become available. The DSES could also identify opportunities for reusing the same sets of resources previously allocated for work units of similar types in different phases of the program, pipeline the execution of work units on these resources (if allowed by the target architecture), or allocate a new set of resources altogether to exploit higher parallelism [28]. For example, Figure 2 describes resource mapping as a coloring problem over an HTG. A feasible mapping is found when all of the HTG nodes have been colored. Given an application and system configuration, several mappings are feasible. Each of them has an overall associated weight, obtained through the cost metrics related to each individual operation/resource binding. Thus, the synthesis engine can build additional intermediate representations, such as a weighted compatibility graph (WCG). In the WCG, edges indicate tasks/operations that can be mapped on the same resource, and the associated weights represent profitability as a result of sharing. Weights account for performance, as well as costs, associated with reconfiguration. Once a proper WCG is built, the mapping can be solved as weighted clique covering.

Multi-objective Design Space Exploration. Finding the best mappings of the application to the target architecture is a complex problem because of the large amount of variables involved. These include the mappings' granularity, available resources, the units' customization, costs, and performance benefits of reconfiguration. In addition, a multitude of factors, such as code transformations (e.g. coalescing and unrolling), scores of assignment policies, and even the algorithm choice for individual synthesis processes, may affect complexity and the final quality of results.

As a result, the design space is extremely wide. Our envisioned $SO(DA)^2$ approach should allow the exploration of a variety of mapping and scheduling techniques, including heuristic approaches and evolutionary and bio-inspired algorithms. The latter, in particular, provide an opportunity to deal simultaneously with multiple phases of the synthesis process (allocation, scheduling, binding), each one potentially an NP-complete problem, using a single encoding and algorithm formulation. Hence, they allow for considering the full design space and identifying solutions that may not be achievable if greedy heuristics are used to solve each

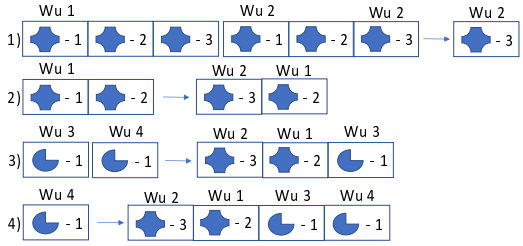

Figure 4: First four ACO steps for combined mapping and scheduling of the HTG in Figure 2 on the provided resource library

phase separately. GAs [21] allow for exploring non-convex design spaces through mutation (introducing local variations), crossover (to jump in across different areas of the space and exit from local minima or maxima), and selection (to lead the search along the most promising area of the space). GAs enable searches across a wide, complex design space in reduced time. Figure 3 depicts an example of a chromosome encoding for the combined problem of allocating and binding resources, allocating memories/buffers, scheduling, and allocating interconnect components [1]. Considering only the first hierarchy of the HTG in Figure 2, the genes in the first part of the chromosome identify work units and their potential assignments (e.g., there are three instances of resource B. Wu 1 is assigned to B1, and Wu 2 is assigned to B3). The remaining genes represent the algorithms chosen for memory allocation, scheduling, and interconnect allocation as the engine could provide different solutions for each one. For genes that represent work units, mutation will change assignments to compatible units. For genes representing algorithms, mutation will select another algorithm. Crossover simply recombines parts of different chromosomes, thus always admitting a feasible solution (if there is only one resource for two different work units, the scheduling algorithm may have to define the priority of execution). The objective (fitness function) may be performance, as well as power, or a combination of the two metrics. Performance and energy results of the solutions are obtained by either running the resulting selected chromosomes on the target architecture and feeding back the results through introspection to the DSES or by integrating estimation mechanisms in the DSES itself.

ACO [16] is a swarm intelligence optimization technique that combines probabilistic local decision with global information provided by cooperating agents. Figure 4 shows the first steps of an ACO formulation that combines scheduling and resource selection for the HTG in Figure 2 [19, 46]. To simplify the example, communication mapping is not considered, which can be realized by extending the approach to also assign arcs to communication components with different costs and constraints [18]. A set of ants separately performs the exploration. At the first step, an ant can select the assignment of Wu 1 and 2 to all three resources of type B. Randomly, Wu 2 on B3 is selected. The next allowable choices, given the HTG dependences, and current use of B3, are only for Wu 1. Wu 1 on B2 is selected, so now Wu 3 and Wu 4 are schedulable on the only resource A available. This will allow for scheduling Wu 4 only after Wu 3. The ants proceed until all nodes have been visited and resources assigned. Local choices for each step can employ a problem-dependent weight (e.g., favor mapping on resources with high availability). At the end of a traversal, the ant will "reinforce" the path it has followed ("pheromone trail") proportionally to the quality of the result (total execution time, total energy, etc.). This reinforcement will influence the probability at each decision step for ants in subsequent runs.

3.4 Runtime Manager

Objective of the DSES is to produce configuration vectors, bitcode for each work unit, and related control glue code, identifying assignments of functional units to work units, actual placements of the functional units for each work unit, data dependencies, synchronization points, and timings. We envision that a full toolchain for reconfigurable architectures needs a runtime manager to effectively provide the interface between the target architecture itself and the DSES, orchestrating the execution of work units and setting configurations. On proper CGRAs composed of functional units and memories and reconfigurable interconnects, the runtime will dynamically load configuration vectors and set functional units to trigger execution of the work units. For solutions composed of cores (at higher granularity of functional units) and configurable interconnects, the runtime will effectively trigger work unit execution following scheduling, assignment, and placing provided by the DSES but also enable dynamic adaptation and rescheduling. The runtime will dynamically determine the available functional units and map the ready work units to the functional units given current available concurrency, the amount of concurrency the work unit will uncover (based on the dependence graph), available memory bandwidth, available resources, and locality of the data bound to the work unit.

Introspection and hardware knobs. Provided the availability of counters and actionable knobs, the runtime manager should be able to control any fine-grained tunable parameters that architectures expose for memory components, cores, interconnect, and/or functional units. Ideally, the implementation of our proposed flow will require the definition of specific (perhaps novel) counters and knobs with the objective of optimizing performance and energy through system introspection (dynamic monitoring and modifications). For proper CGRAs, appropriate counters may for example monitor interconnect traffic; utilization of the functional units; memory accesses; and knobs may enable tuning interconnect bit widths, data fetch size, memory components behavior (scratchpads or caches and size of the cache lines), frequencies, and voltages of the various components. For general-purpose cores with configurable interconnects and malleable data planes, instead, counters may monitor aspects such as stall reasons, caching inefficiencies, and power consumption.

Optimizer and DSES feedback. In the envisioned approach, the runtime layer should be able to share optimization algorithms with the DSES . However, because the runtime manager needs to perform fine tuning of parameters as dynamic feedback is provided, gradient descent and stochastic (regressive) optimization approaches likely will be more effective [24]. The runtime should be able to monitor memory components and interconnect behaviors and change

their modes (i.e., load size, coherent/non-coherent caches, scratchpad, prefetch modes). For power optimizations, the envisioned runtime could use the roofline models and its extensions for memory hierarchies [6, 48] to determine bottlenecks and reduce power through DVFA [32, 36], clock-gating, power-gating, and modifying the number of data lanes used. In other instances, for arithmetic operations, the runtime could monitor integer/floating-point error exceptions and modify code to trade off precision with arithmetic logic unit operations/cycle (i.e., four 32-bit adds instead of eight 16-bit adds per cycle). Integrating the runtime layer and DSES optimization algorithms, however, will enable the exploration of other heuristic approaches. For example, an online ACO formulation with identical parallel work units running in iterations may consider each work unit as an independent agent and employ profiled results and previous reinforcements to select parameters for the following iteration. A strict integration between runtime layer and DSES engine is also required to enable sharing of collected introspection information to enable collection of dynamic statistics and dynamic profiling, and enable profile-driven optimizations by feeding back the collected data to the DSES. Finally, the runtime layer could enable, for certain target platforms, the use of bitcode (or code in a virtual instruction set) at a higher abstraction level than the final target architecture binary code that will could be just-in-time compiled before actual execution. This could allow the runtime manager itself to dynamically introduce modifications to the bitcode and providing extended functionalities with respect to hardware knobs (for example, modifying size of loads or precision of floating-point operations at runtime).

4 PROOFS OF CONCEPT

While we envision $SO(DA)^2$ to target CGRAs to take full advantage of quick reconfiguration, and many research prototypes have appeared, actual commercial devices still are upcoming (E.g., Xilinx's Adaptable Computing Acceleration Platform - ACAP). Thus envision that the various toolchain components could be tested and validated by targeting modern FPGAs, proving that abstractions, design space exploration, synthesis techniques, and reconfiguration mechanisms will work [43–45]. In seminal work, we have extended HLS methodologies to extract TLP and ILP from OpenMP-like annotated code [9, 29, 30] and implemented them in an open-source toolchain [5]. These include a refined memory model that supports multiple memory channels, multiple memory banks, and atomic memory operations [11]. These methodologies can be adapted and integrated in the $SO(DA)^2$ toolchain. Thus, the $SO(DA)^2$ hierarchical machine model could include FPGA-synthesizable components that closely mimics the resources that will be available on CGRAs. $SO(DA)^2$ will be able to employ general-purpose cores embedded in modern FPGAs or as part of hybrid architectures both as executors of the runtime layer and targets for applications' work units.

5 CONCLUSIONS

Novel reconfigurable architectures may provide the adaptability required to address the complex data analytics workflows that integrate graph algorithms, machine learning, and operate on a variety of data structures (dense and sparse matrices, tables, graphs, trees, grids). However, we argue that there remain challenges for the software stack, that needs to jointly explore the space of software optimizations and hardware parameters/configurations without hindering programmer productivity. Current fine-grained FPGAs have already demonstrated success in accelerating machine learning, and memory-intensive workloads, including graph algorithms. In this position paper we describe a full toolchain to enable design space and exploration to map complex data analytics workflows on reconfigurable architectures. We describe the components of the stack, discuss challenges and opportunities, and highlight research that needs to be performed. We also propose references to previous works, focused on high-level synthesis from C for parallel specifications of graph algorithms, that provide proofs of concept for some layers of the stack of the described toolchain.

REFERENCES

[1] 2008. Improving evolutionary exploration to area-time optimization of FPGA designs. *Journal of Systems Architecture* 54, 11 (2008), 1046 – 1057. Embedded Systems: Architectures, Modeling and Simulation.

[2] 2018. BlueBee High Performance Genomics. Available at http://www.bluebee.com/news. [Online; accessed 19-March-2018].

[3] 2018. GCC, the GNU Compiler Collection. https://gcc.gnu.org. Online; accessed 22-March-2018].

[4] 2018. OpenARC: Open Accelerator Research Compiler. Available at: https://ft.ornl.gov/research/openarc. Online; accessed 21-March-2018].

[5] 2018. PandA: on Open Source Framework for Hardware-Software Codesign. Available at https://panda.dei.polimi.it. Online; accessed 22-March-2018].

[6] V. C. Cabezas and M. Püschel. 2014. Extending the roofline model: Bottleneck analysis with microarchitectural constraints. In *2014 IEEE International Symposium on Workload Characterization (IISWC)*. 222–231.

[7] D. Cabrera, X. Martorell, G. Gaydadjiev, E. Ayguade, and D. Jimenez-Gonzalez. 2009. OpenMP extensions for FPGA accelerators. In *SAMOS IX: International Symposium on Systems, Architectures, Modeling, and Simulation*. 17–24.

[8] T. J. Callahan, J. R. Hauser, and J. Wawrzynek. 2000. The Garp architecture and C compiler. *Computer* 33, 4 (Apr 2000), 62–69.

[9] Vito Giovanni Castellana, Marco Minutoli, Alessandro Morari, Antonino Tumeo, Marco Lattuada, and Fabrizio Ferrandi. 2015. High Level Synthesis of RDF Queries for Graph Analytics. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2015, Austin, TX, USA, November 2-6, 2015*. 323–330.

[10] V. G. Castellana, A. Morari, J. Weaver, A. Tumeo, D. Haglin, O. Villa, and J. Feo. 2015. In-Memory Graph Databases for Web-Scale Data. *Computer* 48, 3 (Mar 2015), 24–35.

[11] Vito Giovanni Castellana, Antonino Tumeo, and Fabrizio Ferrandi. 2014. An adaptive Memory Interface Controller for improving bandwidth utilization of hybrid and reconfigurable systems. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2014, Dresden, Germany, March 24-28, 2014*. 1–4.

[12] D. Chavarr/'a-Miranda, V. G. Castellana, A. Morari, D. Haglin, and J. Feo. 2016. GraQL: A Query Language for High-Performance Attributed Graph Databases. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 1453–1462.

[13] Eric S. Chung, Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Adrian M. Caulfield, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Maleen Abeydeera, Logan Adams, Hari Angepat, Christian Boehn, Derek Chiou, Oren Firestein, Alessandro Forin, Kang Su Gatlin, Mahdi Ghandi, Stephen Heil, Kyle Holohan, Ahmad El Husseini, Tamás Juhász, Kara Kagi, Ratna Kovvuri, Sitaram Lanka, Friedel van Megen, Dima Mukhortov, Prerak Patel, Brandon Perez, Amanda Rapsang, Steven K. Reinhardt, Bita Rouhani, Adam Sapek, Raja Seera, Sangeetha Shekar, Balaji Sridharan, Gabriel Weisz, Lisa Woods, Phillip Yi Xiao, Dan Zhang, Ritchie Zhao, and Doug Burger. 2018. Serving DNNs in Real Time at Datacenter Scale with Project Brainwave. *IEEE Micro* 38, 2 (2018), 8–20.

[14] Alessandro Cilardo, Luca Gallo, Antonino Mazzeo, and Nicola Mazzocca. 2013. Efficient and scalable OpenMP-based system-level design. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*. 988–991.

[15] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang. 2011. High-Level Synthesis for FPGAs: From Prototyping to Deployment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30, 4 (April 2011), 473–491.

[16] M. Dorigo, M. Birattari, and T. Stutzle. 2006. Ant colony optimization. *IEEE Computational Intelligence Magazine* 1, 4 (Nov 2006), 28–39.

[17] P. Dziurzanski and V. Beletskyy. 2004. Defining Synthesizable OpenMP Directives and Clauses. In *ICCS 2004: 4th International Conference on Computational Science*. 398–407.

[18] Fabrizio Ferrandi, Pier Luca Lanzi, Christian Pilato, Donatella Sciuto, and Antonino Tumeo. 2010. Ant Colony Heuristic for Mapping and Scheduling Tasks and Communications on Heterogeneous Embedded Systems. *IEEE Trans. on CAD of Integrated Circuits and Systems* 29, 6 (2010), 911–924.

[19] Fabrizio Ferrandi, Pier Luca Lanzi, Christian Pilato, Donatella Sciuto, and Antonino Tumeo. 2013. Ant Colony Optimization for mapping, scheduling and placing in reconfigurable systems. In *2013 NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2013, Torino, Italy, June 24-27, 2013*. 47–54.

[20] Milind Girkar and Constantine D. Polychronopoulos. 1994. The hierarchical task graph as a universal intermediate representation. *International Journal of Parallel Programming* 22, 5 (01 Oct 1994), 519–551.

[21] David E. Goldberg. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning* (1st ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[22] S. C. Goldstein, H. Schmit, M. Moe, M. Budiu, S. Cadambi, R. R. Taylor, and R. Laufer. 1999. PipeRench: a coprocessor for streaming multimedia acceleration. In *Proceedings of the 26th International Symposium on Computer Architecture (Cat. No.99CB36367)*. 28–39.

[23] V. Govindaraju, C. H. Ho, T. Nowatzki, J. Chhugani, N. Satish, K. Sankaralingam, and C. Kim. 2012. DySER: Unifying Functionality and Parallelism Specialization for Energy-Efficient Computing. *IEEE Micro* 32, 5 (Sept 2012), 38–51.

[24] Henry Hoffmann, Jonathan Eastep, Marco D. Santambrogio, Jason E. Miller, and Anant Agarwal. 2010. Application Heartbeats: A Generic Interface for Specifying Program Performance and Goals in Autonomous Computing Environments. In *Proceedings of the 7th International Conference on Autonomic Computing (ICAC '10)*. 79–88.

[25] Melanie Kambadur and Martha A. Kim. 2014. An Experimental Survey of Energy Management Across the Stack. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications (OOPSLA '14)*. 329–344.

[26] Y. Y. Leow, C. Y. Ng, and W.F. Wong. 2006. Generating Hardware from OpenMP Programs. In *FPT 2006: IEEE International Conference on Field Programmable Technology*. 73–80.

[27] Bingfeng Mei, Serge Vernalde, Diederik Verkest, Hugo De Man, and Rudy Lauwereins. 2003. ADRES: An Architecture with Tightly Coupled VLIW Processor and Coarse-Grained Reconfigurable Matrix. In *Field Programmable Logic and Application*, Peter Y. K. Cheung and George A. Constantinides (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 61–70.

[28] M. Minutoli, V. G. Castellana, A. Tumeo, and F. Ferrandi. 2015. Inter-procedural resource sharing in High Level Synthesis through function proxies. In *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*. 1–8.

[29] Marco Minutoli, Vito Giovanni Castellana, Antonino Tumeo, Marco Lattuada, and Fabrizio Ferrandi. 2016. Efficient synthesis of graph methods: a dynamically scheduled architecture. In *Proceedings of the 35th International Conference on Computer-Aided Design, ICCAD 2016, Austin, TX, USA, November 7-10, 2016*. 128.

[30] Marco Minutoli, Vito Giovanni Castellana, Antonino Tumeo, Marco Lattuada, and Fabrizio Ferrandi. 2016. Enabling the high level synthesis of data analytics accelerators. In *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, CODES 2016, Pittsburgh, Pennsylvania, USA, October 1-7, 2016*. 15:1–15:3.

[31] Mahim Mishra, Timothy J. Callahan, Tiberiu Chelcea, Girish Venkataramani, Seth C. Goldstein, and Mihai Budiu. 2006. Tartan: Evaluating Spatial Computation for Whole Program Execution. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XII)*. 163–174.

[32] Sparsh Mittal. 2014. A Survey of Techniques For Improving Energy Efficiency in Embedded Computing Systems. *CoRR* abs/1401.0765 (2014).

[33] R. Nane, V. M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, and K. Bertels. 2016. A Survey and Evaluation of FPGA High-Level Synthesis Tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 10 (Oct 2016), 1591–1604.

[34] NVIDIA. 2018. About CUDA. Available at https://developer.nvidia.com/about-cuda. Online; accessed 22-March-2018].

[35] OpenACC-standard.org. 2018. OpenACC More Science Less Programming. Available at https://www.openacc.org/. Online; accessed 22-March-2018].

[36] Anne-Cecile Orgerie, Marcos Dias de Assuncao, and Laurent Lefevre. 2014. A Survey on Techniques for Improving the Energy Efficiency of Large-scale Distributed Systems. *ACM Comput. Surv.* 46, 4, Article 47 (March 2014), 31 pages.

[37] Jian Ouyang, Shiding Lin, Wei Qi, Yong Wang, Bo Yu, and Song Jiang. 2014. SDA: Software-defined accelerator for large-scale DNN systems. In *2014 IEEE Hot Chips 26 Symposium (HCS)*. 1–23.

[38] Constantine D. Polychronopoulos. 1991. The Hierarchical Task Graph and Its Use in Auto-scheduling. In *Proceedings of the 5th International Conference on Supercomputing (ICS '91)*. 252–263.

[39] R. Prabhakar, Y. Zhang, D. Koeplinger, M. Feldman, T. Zhao, S. Hadjis, A. Pedram, C. Kozyrakis, and K. Olukotun. 2017. Plasticine: A reconfigurable architecture for parallel patterns. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. 389–402.

[40] Michael Bedford Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fae Ghodrat, Ben Greenwald, Henry Hoffman, Paul Johnson, Jae-Wook Lee, Walter Lee, Albert Ma, Arvind Saraf, Mark Seneski, Nathan Shnidman, Volker Strumpen, Matt Frank, Saman Amarasinghe, and Anant Agarwal. 2002. The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs. *IEEE Micro* 22, 2 (March 2002), 25–35.

[41] R. Tessier, K. Pocek, and A. DeHon. 2015. Reconfigurable Computing Architectures. *Proc. IEEE* 103, 3 (March 2015), 332–354.

[42] The Khronos Group. 2018. OpenCL Overview. Available at: https://www.khronos.org/opencl/. Online; accessed 22-March-2018].

[43] Antonino Tumeo, Simone Borgio, Davide Bosisio, Matteo Monchiero, Gianluca Palermo, Fabrizio Ferrandi, and Donatella Sciuto. [n. d.]. A multiprocessor self-reconfigurable JPEG2000 encoder. In *RAW'09: Reconfigurable Architectures Workshop, in 23rd IEEE International Symposium on Parallel and Distributed Processing, IPDPS*.

[44] Antonino Tumeo, Marco Branca, Lorenzo Camerini, Marco Ceriani, Matteo Monchiero, Gianluca Palermo, Fabrizio Ferrandi, and Donatella Sciuto. 2008. A Dual-Priority Real-Time Multiprocessor System on FPGA for Automotive Applications. In *Design, Automation and Test in Europe, DATE 2008, Munich, Germany, March 10-14, 2008*. 1039–1044.

[45] Antonino Tumeo, Marco Branca, Lorenzo Camerini, Marco Ceriani, Matteo Monchiero, Gianluca Palermo, Fabrizio Ferrandi, and Donatella Sciuto. 2009. Prototyping pipelined applications on a heterogeneous FPGA multiprocessor virtual platform. In *Proceedings of the 14th Asia South Pacific Design Automation Conference, ASP-DAC 2009, Yokohama, Japan, January 19-22, 2009*. 317–322.

[46] Antonino Tumeo, Christian Pilato, Fabrizio Ferrandi, Donatella Sciuto, and Pier Luca Lanzi. 2008. Ant colony optimization for mapping and scheduling in heterogeneous multiprocessor systems. In *Proceedings of the 2008 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (IC-SAMOS 2008), Samos, Greece, July 21-24, 2008*. 142–149.

[47] H. M. Waidyasooriya, M. Hariyama, and K. Kasahara. 2016. Architecture of an FPGA accelerator for molecular dynamics simulation using OpenCL. In *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*. 1–5.

[48] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM* 52, 4 (April 2009), 65–76.