Enabling Design Automation for Data Analytics

Vito Giovanni Castellana,
Marco Minutoli, Antonino Tumeo
High Performance Computing - PNNL
{vitoGiovanni.castellana, marco.minutoli,
antonino.tumeo}@pnnl.gov

Marco Lattuada, Fabrizio Ferrandi
DEIB - Politecnico di Milano
{marco.lattuada,
fabrizio.ferrandi}@polimi.it

Data Analytics applications, such as graph databases, often employ pointer or linked list-based data structures that, although convenient to represents dynamically changing relationships among the data elements, induce an irregular behavior. These data structures, in fact, allows spawning many concurrent activities, but present many unpredictable, fine-grained, data accesses, and require many synchronization operations. Partitioning the datasets without generating load imbalance is also very difficult. Conventional general-purpose architectures are optimized for locality and reduced access latency, and do not cope well with these workloads, making application-specific accelerators (implemented, for example, on Field Programmable Gate Arrays - FPGAs) an appealing solution. However, conventional High-Level Synthesis (HLS) approaches, which provide a way to reduce the effort of designing custom accelerators by automatically translating program descriptions in high-level languages such as C to specifications in hardware description languages (Verilog or VHDL), are not yet well optimized for these types of workloads. In fact, conventional HLS flows traditionally target compute intensive workloads (i.e., digital signal processing) that mainly expose instruction level parallelism, and can be easily partitioned across replicated functional units. They also usually assume a simple memory system focusing more at reducing, rather than tolerating, data access latencies.

We have developed a set of novel architectural templates and HLS methodologies to automatically generate efficient accelerators for graph methods. To validate our approach, we have investigated the integration of the approach into GEMS (Graph Engine for Multithreaded Systems), a graph database for commodity homogenous clusters based on the Resource Description Framework (RDF) and the SPARQL query language. GEMS exploits and optimizes for graph methods at all layers of its stack, and is able to scale in size as more nodes are added to system while maintaining constant query throughput. Coupling GEMS with our HLS approach, we have been able to generate accelerators for SPARQL queries coming from the Lehigh University Benchmark (LUBM), a reference benchmarks for Semantic Web Repositories. We have integrated our HLS templates and methodologies in an openly available High Level Synthesis tool, Bambu (http://panda.dei.polimi.it). These include: a Parallel distributed Controller (PC) , a Hierarchical, multi-ported, Memory Interface (HMI), and a Dynamic Task Scheduler (DTS). The PC allows generating more efficient designs that exploit coarse grained (task level) parallelism than the typical centralized controllers of conventional HLS flows based around the Finite State Machine with Datapath (FSMD) model, in terms of performance and area utilization. This is a key element in accelerating graph algorithms that basically are composed of a varying number of nested loops, iterating on vertices or edges, where each iteration could identify a different task. By adopting the PC, it is easy to coordinate parallel execution of tasks on an array of replicated accelerators. The HMI provides a solution to dynamically disambiguate fine-grained memory accesses to locations of a large, multi-banked memory, while providing an abstract view of a shared memory to the HLS flow and the accelerator pool. In cooperation with the PC, the HMI also enables the support for atomic memory operations. Graph algorithms typically access unpredictable memory locations with fine-grained transactions (i.e., the follow pointers), and their implementation is much easier when considering a shared memory abstraction. Also, they often are synchronization intensive when parallelized, because the different tasks may access the same elements concurrently. Finally, the DTS provides a way to execute new tasks as soon as one of the multiple accelerators is free: instead of simply using a fork/join model, where all currently executing tasks on the set of accelerators must terminate before a new group could be executed, the DTS allow scheduling new tasks as soon as one of the accelerators is free. This adapt to a variety of graph algorithms where certain tasks (iterations) may execute for a long time, while other could terminate early, such as when a graph walk is pruned early because it reached an uninteresting part of the graph.

We have integrated GEMS with Bambu, synthesized 7 queries from LUBM, and tested the performance of the generated accelerators using a dataset of 5,309,056 RDF "triples". In the table, we compare, in terms of execution latency, a serial implementation of the architecture (Single Acc.), one that employs PC and the HMI, and one that also includes the DTS (Dynamic Scheduler). The parallel architectures include 4 accelerators and HMIs with 4 ports. With respect to the serial implementation, the architectures employing the DTS generally show a speed up close to the theoretical maximum. In many cases, the DTS also provides significant speed ups against the PC designs. The DTS also maximizes utilization of the available memory channels. With the DTS, 3 out of 4 of the memory ports are utilized for more than 75% of the time.

	Single Acc. # Cycles	Parallel Controller # Cycles	Dynamic Scheduler # Cycles	Speedup	
				Single Acc.	Parallel Controller
Q1	1,082,526,974	1,001,581,548	287,527,463	3.76	3.48
Q2	7,359,732	2,801,694	2,672,295	2.75	1.05
Q3	308,586,247	98,163,298	95,154,310	3.24	1.03
Q4	63,825	42,279	19,890	3.21	2.13
Q5	33,322	13,400	8,992	3.71	1.49
Q6	682,949	629,671	199,749	3.42	3.15
Q7	85,341,784	35,511,299	24,430,557	3.49	1.45

We believe that our results demonstrate the applicability of HLS to data analytics, providing a way for productively using custom accelerators in this area.