A combined fast/cycle accurate simulation tool for reconfigurable accelerator evaluation: application to distributed data management

Erwan Lenormand, Thierry Goubier, Loïc Cudennec, Henri-Pierre Charles

A combined fast/cycle accurate simulation tool for reconfigurable accelerator evaluation: application to distributed data management

Erwan Lenormand, Thierry Goubier
Institut LIST
Université Paris-Saclay, CEA
F-91191, PC 172, Gif-sur-Yvette, France
firstname.lastname@cea.fr

Loïc Cudennec
Department of Artificial Intelligence
DGA MI
BP 7, 35998 Rennes Armées, France
loic.cudennec@intradef.gouv.fr

Henri-Pierre Charles
Institut LIST
Université Grenoble-Alpes, CEA
F-38000 Grenoble, France
henri-pierre.charles@cea.fr

***Abstract*—Parallel computing systems based on reconfigurable accelerators are becoming (1) increasingly heterogeneous, (2) difficult to design and (3) complex to model. Such modeling of a parallel computing system helps to evaluate its performance and to improve its architecture before prototyping. This paper presents a simulation tool aiming to study the integration of reconfigurable accelerators in scalable distributed systems and runtimes, such as S-DSM systems, where S-DSM (software-distributed shared memory) is a paradigm to ease data management among distributed nodes. This tool allows us to simulate the execution of irregular compute kernels accessing distributed data. To deal with the complexity of modeling (3) the complete system we used a hybrid methodology. We integrated the simulation engine into the S-DSM. The distributed data management part is executed on the physical architecture allowing to generate precise and faithful latencies, and the accelerator simulation is cycle accurate. We used general sparse matrix-matrix multiplication (SpGEMM) as a case study.**

We show that the use of this tool makes it possible to analyze the behavior of an heterogeneous system (1) with rapid prototyping and simulation. The analysis of the results allowed to determine the correct sizing of the architecture (2) to obtain the best performance. The tool allowed to identify the bottleneck of our architecture and confirmed the possibility of hiding data access latencies.

Our simulation platform allows to emulate a heterogeneous distributed system by introducing a slowdown between 1.2 and 3.7 times compared to the compute kernel simulation alone.

***Index Terms*—Hybrid simulation; Distributed systems; Reconfigurable Computing**

I. Introduction

Parallel computer systems are becoming increasingly heterogeneous. This trend can be observed at all scales from System-on-Chip (SoC) to supercomputers. Heterogeneity is a response to the power wall problem [1] by scheduling a task on the appropriate processing unit depending on its power and performance characteristics. However, as illustrated by the efficiency on the high performance conjugate gradient (HPCG) benchmark [2], current systems are inefficient for applications with irregular data access and low arithmetic intensity. A good data management is critical to improve the performance of these systems. A runtime system can transparently provide developers with intelligence in data management. The increase in heterogeneity requires further research to improve the integration of new types of computing resources. In this context we want to study the integration of reconfigurable accelerators (Field-programmable gate array (FPGA)) in a software system unifying the distributed memories. Our goal is to allow accelerators to request access to distributed data. Designing this system is a complex task requiring to prototype a hardware IP and then to integrate this IP into a software system that is also complex. Finally we want to study the acceleration of irregular compute kernels with unpredictable (or hard to predict) memory accesses. To understand how to develop this system and to predict its performance, we need to model it. Thus, we need to model a compute kernel whose evolution depends on its memory accesses and induced latencies. The specificity of the system to model and the need to limit the prototyping time led us to develop our own simulation tool presented in this paper.

This tool should answer the following questions:

- What performance could the system achieve?
- Which configuration provides the best performance?
- What are the bottlenecks?

Finally, this tool should enable rapid modeling and rapid simulation to facilitate the exploration work.

The paper is organized as follows: Section II describes the system to model, Section III gives some references on related work, Section IV presents our simulation tool, Section V describes the experiments conducted with the tool, finally, Section VI concludes this article.

II. Integration of Reconfigurable Accelerator in Software-Distributed Shared Memory

A parallel programming model with shared memory is convenient to develop multi-threaded applications, which concurrently access data in a global memory space. In a single memory system, data sharing can be based on reliable hardware mechanisms. For distributed memory systems, the implementation of a shared memory is more complex and requires to minimize the processing and communication costs. The study of distributed shared memory (DSM) have started

in the late eighties with systems such as Ivy [3] and later adapted to clusters [4], grids [5], many-core processors [6] and recent heterogeneous architectures [7], [8]. In this work we use a Software-DSM (S-DSM) [9] [10] allowing to federate the physical memories of heterogeneous architectures. We study the integration of reconfigurable accelerators in this S-DSM to accelerate compute kernels.

A. Software Distributed Shared Memory

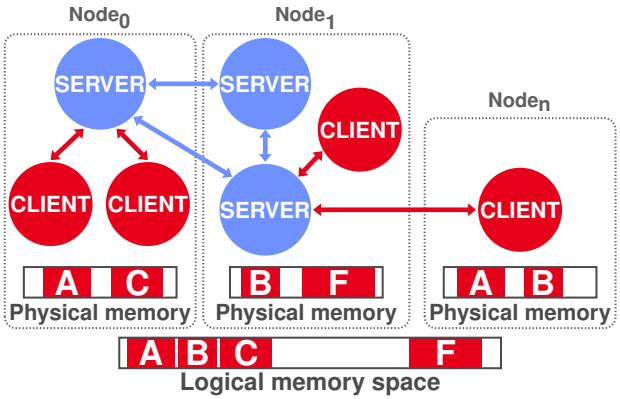


Fig. 1: S-DSM semi-structured super-peer topology

The S-DSM allows tasks to allocate and access memory in a shared logical space. The S-DSM is organized as a semi-structured super-peer network, as represented in Figure 1. A set of clients are connected to a peer-to-peer network of servers. Clients execute the user code and servers manage the shared data and metadata. Allocated data is split into *chunks* whose maximum size is defined by the user. Each *chunk* is under the control of a data coherence protocol. A *chunk* has a unique identifier (ID) and metadata indicating its size, state and location. The coherence protocol is in charge of the localization and the transfer of the *chunk*.

Using accelerators within S-DSM-based applications follows the classical hybrid programming paradigm, widely adopted in HPC with MPI/CUDA or MPI/OpenMP applications for instance. In these software architectures, the user code is split into two parts: 1) a distributed overlay that manages communications between remote processes, schedules jobs and orchestrates data transfer between nodes and 2) a local proxy code on each node that locally exploits the computing resources using the provided application programming interface (API). This model results in a two-step procedure to retrieve data from the distributed overlay,. Firstly, manually converts this data into the accelerator format (potentially involving multiple copies in memory), then transfers data to the accelerator memory and offload the processing. The same applies for fetching the results from the accelerator back to the distributed overlay. Such hybrid-programming applications require from the user to manage data twice, both at the distributed overlay level and the accelerator level. One must note that one of the evolutions of systems with accelerators is shared memory at the node level, via solutions based on NVLink, OpenCAPI [11], CCIX [12] and others, clearly pointing out that this manual data management is a problem.

In this work, we propose a simulation tool that helps in modeling and exploring different configurations of a system in which data is transparently managed between computing kernels, either it is an FPGA IP or a software process.

B. Reconfigurable Accelerator Integration

All actors of the S-DSM (clients and servers) are basically software processes. To integrate a compute kernel implemented on an FPGA in the S-DSM, we need to create an interface between a software process and the programmable logic (PL) of the accelerator. As shown in Figure 2, this interface is based on the cooperation of a software process (*FPGA-client*) executed on a processing system (PS) and a hardware component (*FPGA-server*) implemented in the PL. The PS and the PL communicate through an on-chip (AXI) or off-chip (PCI Express) interconnect.

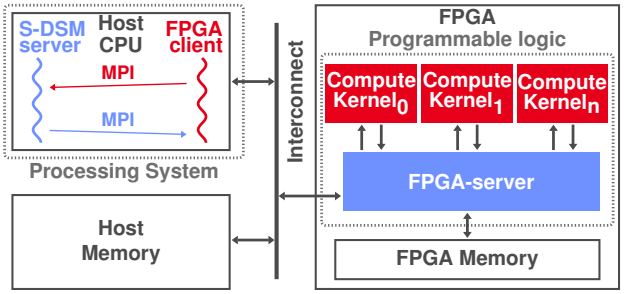


Fig. 2: Overview of the reconfigurable accelerator integration architecture

The FPGA memory is segmented into *chunk-sized* locations. The *FPGA-server* operates as a cache directory. Each entry associates a location with a *chunk* and its metadata (ID, size, state). When a compute kernel requests access to a *chunk*, if it is present in memory, the transfer is directly initiated. Otherwise, the *FPGA-server* forwards the access request to the *FPGA-client* and the transfer will be initiated after the allocation of the *chunk*. Thus, the time required to access the data can be very variable and depends on the state of the memory. The *FPGA-client* is a S-DSM client process. It interprets the requests coming from the FPGA and translates them for the S-DSM server. In this way, the FPGA appears as a regular client for all S-DSM processes.

C. Compute Kernel

The compute kernel is based on the dataflow model. It receives data within streams, performs the processing function and generates the results within streams. This is the typical model of compute kernels generated by high level synthesis tools [13]. In our system, data streams are associated with requests generated by the kernels. Thus, read requests result in an incoming data stream and write requests in a outgoing data stream. As shown in Figure 3, each stream is implemented by a FIFO and several signals. For read requests, the kernel communicates the *chunk ID* to the *FPGA-server*. Then, when

the data is available, the *FPGA-server* transmits the *chunk* size and fills the FIFO with the data. The *chunk* size is required when coping with irregular compute kernels. For write requests, the kernel communicates the size and the *chunk* ID. Then the *FPGA-server* extracts data from the FIFO to write it back in memory.

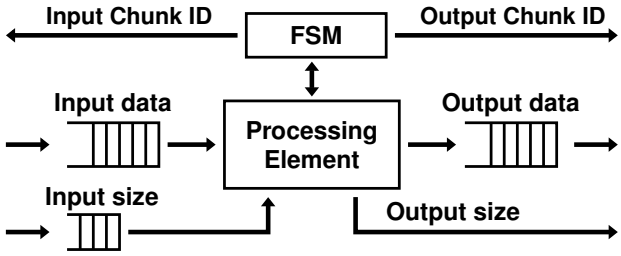


Fig. 3: Compute kernel interface

III. Related Work

Powerful simulation tools, such as [14] [15] [16] [17], allow the simulation of a full heterogeneous CPU-FPGA platform to provide highly accurate performance estimates. These tools are based on the integration of a simulator, such as gem5 [18], Multi2Sim [19] or Verilator [20]. Each have various constraints and limitations. For example, HeteroSim [16] can not support the simulation of a runtime that controls the interactions between a CPU and an FPGA. PARADE [15] only simulates programming models where all data must be copied to local scratchpad memory (SPM) before launching a compute kernel. gem5-Aladdin [14] represents the accelerator as a set of dynamic data-dependent graphs generated from the high-level language descriptions of algorithms. This approach, which is efficient for simulation, limits the reconfiguration of the accelerators. PAAS [17] and PARADE are focused on simulating heterogeneous SoC.

Full system simulation can be time consuming. One method to speed up simulation consists in associating a temporal dimension with an event. An automatic delay-annotated mechanism to replace the use of Instruction-Set-Simulation (ISS) in SystemC simulations is presented in [21]. The contributions presented in [22], are based on packet latency estimations to replace detailed Network-on-Chip models in full-system performance simulators. Latencies are estimated from an analytical model. With this method, the accuracy of the simulation depends on the reliability of the analytical model. Making a reliable model is not always simple.

FPGA Computer Aided Design (CAD) flow tools, such as Verilog-To-Routing (VTR) [23], integrate simulation tools. These tools allow to ensure the correctness of the design and estimate the circuit-activity [24]. However, these tools cannot simulate an entire distributed system.

Finally some methods are hybrid, such as [25]. To speed up the simulation of an HW/SW system, this work combines a virtual platform simulation for the software part with an FPGA-based physical prototype for the hardware part.

IV. Hybrid Simulation Tool

The purpose that has led to the development of this simulation tool is the evaluation of the performance of compute kernels integrated into the system described in Section II. The kernels we want to study have two forms of irregularities: they perform unpredictable (or hard to predict) memory accesses and their arithmetic intensities are data dependent. The distributed nature of the system we are studying implies high and variable data access latencies. Thus, performance evaluation is based on the analysis of the compute kernel activity and the generation of data access latencies relating to this activity. A high-level view of kernel activity, based on a dataflow model, comes down to consume and produce data at different speeds. At a lower scale, the activity of the compute kernel is constrained by the resources allocated to it, access to shared resources and the data flow provided to it over time. Thus, our tool is based on (IV-B) the activity simulation of a compute kernel and (IV-A) the generation of latencies. It was developed in C++ as a library of modular components. This approach makes it possible to generate various compute kernels quickly. The simulation is executed by three processes (one S-DSM server and two S-DSM client), which communicate and synchronize through S-DSM requests. The first client reads the input matrices and writes the output matrix. The second client runs the simulation engine.

A. S-DSM Interface Simulation

The role of the S-DSM interface is to generate data access latencies. It reproduces the implementation of the directory by associating to each entry with a *chunk*. The number of locations is configurable. Thus, the size of the FPGA memory can be defined when launching the simulation. When a request is issued by the kernel four scenarios are possible, depending on whether the chunk is allocated or not in the FPGA and if there is a directory entry for this chunk in the FPGA or not:

1) the *chunk* is allocated, then the transfer is possible immediately;
2) the *chunk* is not allocated and a directory entry is available, then a latency is generated indicating the cycle number when the *chunk* will be allocated;
3) the *chunk* is not allocated, but a request is pending, then the chunk will be available at the cycle defined in scenario 2;
4) the *chunk* is not allocated and any directory entry is available.

In scenario 1, 2 and 3 the request sent by the kernel is acknowledged. In scenario 4, the request remains blocked until an entry becomes available.

To generate the latencies we use a S-DSM client, which performs the requests in the real environment and we measure the elapsed time. This elapsed time is converted into a number of FPGA cycles and is added to the current cycle counter. Finally cycles are added corresponding to the transfer time between the *FPGA-server* and the *FPGA-client*.

B. Compute Kernel Simulation

A compute kernel is developed as a pipeline, where each stage is separated by FIFOs. Each stage performs actions that can be 1) to interact with an external component or 2) to perform processing on data. A global clock is used to synchronize each stage. A stage can perform an action only if its input FIFOs are not empty and its output FIFOs are not full.

Figure 4 is an example of a compute kernel architecture for a sparse matrix scalar multiplication. Data access patterns are generated by a finite state machine (FSM). These patterns are supplied to stages (Chunk read prefetch and chunk write prefetch), which perform *chunk* prefetch. It consists in sending a request to the S-DSM interface. When the request is accepted, the *chunk* ID is written to a virtual FIFO. We call it virtual FIFO, because, it does not correspond to a component implemented in a hardware design. However, in our study this FIFO allows to limit the number of prefetched elements by stream. Thus, this limit corresponds to the depth of the FIFO (prefetch depth). The memory access stages correspond to the transfer of data between the memory controller and each FIFO associated with a stream. To reproduce the real behavior of a memory controller, only one request can be accepted per cycle when the controller is idle. Each memory access is associated with a latency in order to reproduce behavior of DDR memories. The size of a memory transfer is limited according to a maximum burst length. At each cycle the number of data that can be read or written depends on the width of the bus. Finally, a read access can only be initiated if the FIFO *Data* has enough space to memorize all the data of a chunk. The size of a FIFO is configurable.

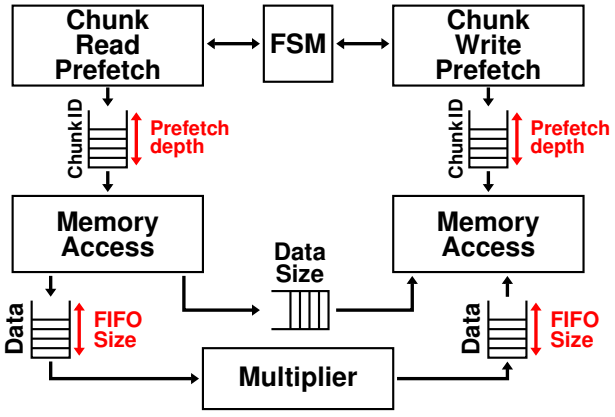


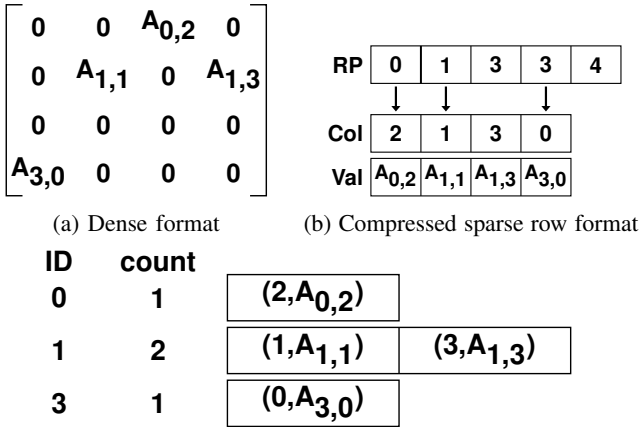
Fig. 4: Sparse matrix scalar multiplication kernel architecture overview

V. Experiments

The case study of our experiments is the general sparse matrix-matrix multiplication (SpGEMM) with matrices, which cannot be fully stored in the memory of the accelerator or the host node. This case requires the transfer of data between the distributed shared memory and the accelerator during execution. The study aims to determine the best configuration of the architecture to optimize the performance of the application.

A. Case Study: Sparse General Matrix-Matrix Multiplication

SpGEMM is widely used to study acceleration methods for sparse linear algebra. This application has a well known behavior and generates irregular memory access patterns that makes it complex to optimize, with usually a low efficiency in terms of floating point operations per unit of time.



(a) Dense format

(b) Compressed sparse row format

(c) Compressed sparse row format with chunk

Fig. 5: Matrix representation

Sparse matrices are compressed to reduce their memory footprint and to accelerate access to their nonzero elements (NNZ). The compressed sparse row format (CSR), shown in Figure 5b, is one of the most used sparse matrix representations. Non-zero elements of the matrix are stored in the array *Val* in row-major order. The column indices of each element are stored in the array *Col* in the same order. Finally, $RP[i]$ indicates the position of the first element of row i in the arrays *Val* and *Col* and the operation $RP[i+1] - RP[i]$ is equal to the number of elements in the row. As shown in Figure 5c, We have adapted the CSR format to the use of *chunks*. We colocalize the value and the column index of an element to form a pair. The set of pairs representing a row is stored in a *chunk*, which makes it possible to use the chunks ID to browse the matrix row by row. Then we use chunks metadata to indicate the number of elements contained in the row. This structure reduces the number of memory accesses required to read or write a matrix row. It can be easily adapted to another compressed format (e.g. compressed sparse column format).
To develop the compute kernel, we used the row-wise sparse matrix-matrix multiplication algorithm formulated by Gustavson [26]. It limits random data access and is quite straightforward to parallelize. The kernel reads and distributes the non-zero elements of the first input matrix to several processing elements (PEs). Each PE multiplies the elements received by a row of the second input matrix. Finally the partial products are reduced and written. So, for a kernel with NPE, there are $N+2$ streams of data between the memory and the

kernel (one to read first input matrix, one to write the output matrix and one by PE to read the second input matrix). The kernel processes single precision floating point numbers.

B. Simulation Environment

TABLE I: Simulation platform setup

Local node		Remote node (Hikey 970)
Intel Core i7-6800k 3.6 GHz 6 cores / 12 threads LLC: 15 MB RAM: 64 GB	Xilinx VC707 200 MHz 2.8 k DSP SPM: 5 MB RAM: 1 GB	ARM Cortex-A73 / Cortex-A53 2.36 GHz/ 1.8 GHz 2 × 4 cores LLC: 8 MB / 2 MB RAM: 6 GB
PCIe Gen2x8		
Gigabit Ethernet		

Table I describes some characteristics of the platform used for the simulation. This platform is made up of two nodes. For both configurations the S-DSM server is executed on the Intel Core i7 local node. The Core i7 local node integrates an FPGA interconnected by a PCI Express bus. The Hikey 970 remote node integrates two Arm processors and is interconnected to the local node by a gigabit Ethernet network (Ethernet is implemented over USB 3.0 on the Hikey 970 development board). We use the remote node to model Xilinx FPGAs of the Zynq family, exploiting the fact that their host systems, made of big.LITTLE Cortex processors are similar.

We used a set of matrices (Table II) from the SuiteSparse Matrix Collection [27]. All matrices are derived from real applications. We chose matrices of variable sizes and densities because the performance of SpGEMM are data-dependent.

TABLE II: Square matrices used for simulations

Name	Row	NNZ	Density (%)
consph	83334	6010480	0.087
cop20k_A	121192	2624331	0.018
F2	71505	5294285	0.10
m_t1	97578	9753570	0.10
s3dkt3m2	90449	3753461	0.046

C. Results

We designed this simulation tool to explore different architecture configurations. We based our experiments on the ability of the tool to show the impact of different parameters on the performance of the architecture. For each experiment we limit the number of memory locations so that a matrix cannot be fully stored on the FPGA. In the first part of this subsection we present and analyze the results obtained. In the second part, we discuss the execution speed and the ins and outs of using this simulation tool. For the first test, we use a configuration with 8 processing elements. We vary the number of prefetched elements per stream of data. Intuitively, the higher the number, the less latency has effect. Figure 6 shows the results obtained. They partially confirm our intuition. We can observe that increasing the number of prefetched elements improves performance up to 1024 elements, after which we observe a decrease in performance. This phenomenon is explained by the fact that too early prefetching increases conflicts in the FPGA memory. This first experiment allows us to determine that the best parameter is 1024 prefetched elements.

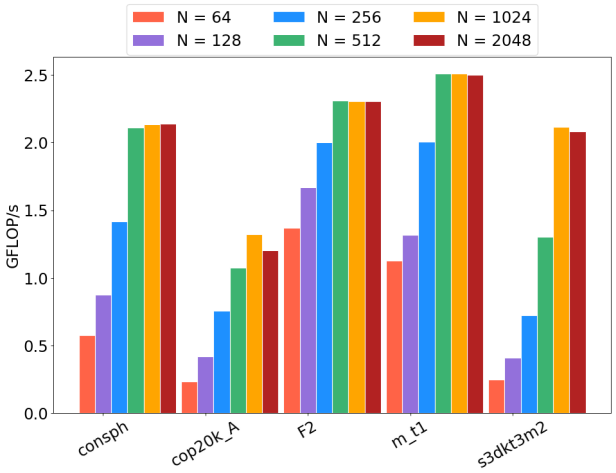


Fig. 6: Computation speed in GFLOP/s according to the number of prefetched elements by stream (higher is better)

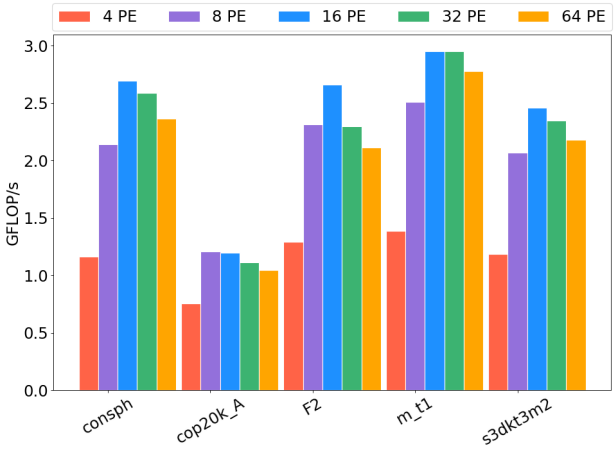


Fig. 7: Computation speed in GFLOP/s according to the number of processing elements (PEs) (higher is better)

For the second experiment, we vary the level of parallelism and we prefetch 1024 elements per stream. The objective is to observe the limits of the acceleration capacity of the compute kernel. Figure 7 illustrates the results. We can observe that the increase of processing elements allows a speedup, up to 16 PEs. Beyond, the increase in parallelism no longer speed up the execution. These results show that the processing elements are under-exploited due to an insufficient supply of data (data starvation). This phenomenon can be explained either by excessive data access latencies, or by a local bottleneck in the design. The integration of counters in the simulation tool allows us to monitor the activity of each element of the design. In Figure 8, we can see the memory controller occupancy rate. We can see that the bus is saturated for the 16 PEs configuration. This information highlights that the memory bandwidth is the bottleneck of our design. Also, it allows us

to estimate the computation speed limit around 3 GFLOPS

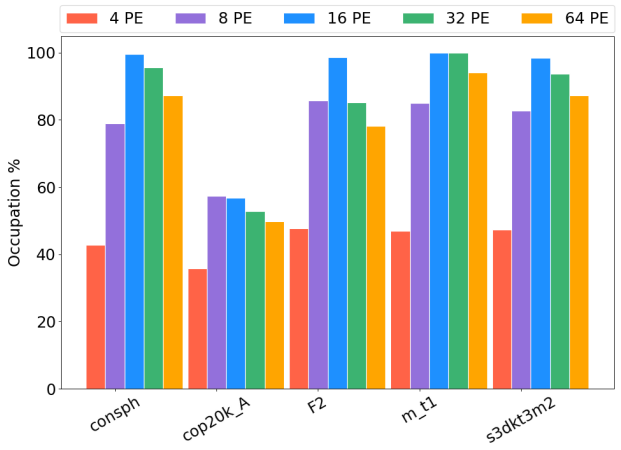


Fig. 8: Memory controller activity (occupancy percentage) according to the number of processing elements. Close to 100% means saturation

For the last experiment, we studied three topologies, which generate different memory access latencies. The first corresponds to the ideal case where all the data is contained in memory at startup (i.e no latency for data access). For the second configuration, we consider an FPGA linked by a PCI Express bus to the node where the S-DSM server is located. Finally, the third configuration models a system where the server and the FPGA are not running on the same node and where the two nodes are interconnected by an Ethernet network. In this configuration the S-DSM client process is executed on the processor of the remote node. For this experiment we use a configuration with 16 processing elements and 1024 prefetched elements per stream. Figure 9 represents the results obtained. It shows

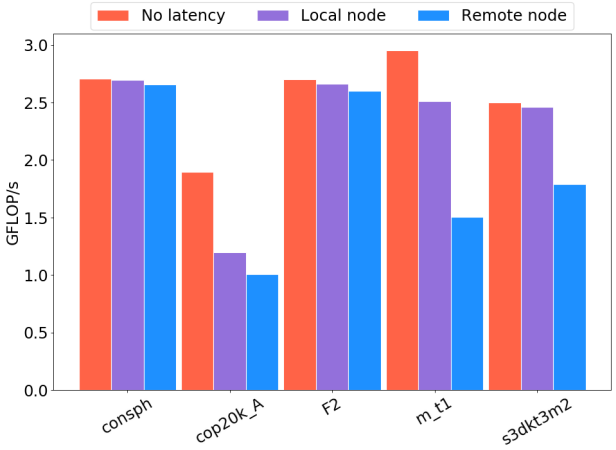


Fig. 9: Computation speed in GFLOP/s according the system topology (higher is better)

that it is possible to hide the latency of data access with our approach. For the *consph* and *F2* matrices, the difference in performance is small between the configuration without latency and the configuration with the biggest latency. However for the matrices *cop20k_A*, *m_t1* and *s3dkt3m2* we can observe a degradation of performances. We can conclude that performances depend on the data set and that, to optimize the performances it is necessary to reorder the matrices to favor the temporal locality of data access.

The results previously presented show the ability of the tool to explore different architecture configurations. Our hybrid method makes this exploration work easier by using a real software-distributed shared memory system to run simulation on several nodes. We think that this tool has several advantages compared to a homogeneous method, where the whole architecture (processors, interconnects, accelerators) are simulated. The first is the use of physical architecture ensures the generation of accurate and faithful data. The second is to speed up modeling by concentrating all of the efforts on the compute kernel development. Finally, this method makes it possible to quickly simulate a complex system.

Fig. 10: Simulation speed in thousands of cycles per second (higher is better)

Figure 10 represents the execution times for the experiment shown in Figure 9. For red bars, as we do not simulate latency we do not use the S-DSM environment (i.e. only the simulation engine process is run). In this way, they represent the speed to simulate the compute kernel and the hardware S-DSM server only. The blue and purple bars represent respectively a local execution and a remote execution of the simulation engine. Thus, Figure allows to visualize the cost of modeling the environment of the S-DSM (which includes, a multi-core processor and its memory system, a gigabit Ethernet network and at least 2 MPI processes). Obviously, we observe a decrease in the speed of simulation (between 1.2 and 2.3 times slower for local execution and 1.7 and 3.7 times slower for remote execution). However, considering the complexity of the simulated model we can say that our hybrid method still allows rapid simulation.

VI. Conclusion and Future Work

In this article, we have presented a simulation tool, which aims to study the integration of reconfigurable accelerators in a software-distributed shared memory (S-DSM). Our objective was to simulate the execution of irregular compute kernels, which access to distributed data with variable latencies. To deal with the complexity of modeling this complete system we used a hybrid methodology. This approach consists in simulating only the compute kernel and integrating the simulation engine in the S-DSM. This method generates real latencies from physical architecture. We used sparse general matrix-matrix multiplication as a case study. We have shown, through the results of several experiments, that our simulation tool allows a correct sizing of the architecture to improve the performance of a compute kernel. The tool also allowed to determine that the memory bandwidth is the architecture bottleneck. Finally, the tool allowed to conclude that our approach could make it possible to hide the data access latencies, which is usually a limitation when coping with distributed systems.

To continue the exploration work, we plan to model new accelerator architectures with higher memory bandwidth. In particular we want to model cards that embed High Bandwidth Memory (HBM) technologies. Also, we want to develop other compute kernels with higher arithmetic intensities. Finally, we would like to estimate the energy consumption of the kernels. To do this we plan to integrate the generation of traces to replay the execution with consumption estimation tools.

Acknowledgements

This work was supported by the LEXIS project, funded by the EU's Horizon 2020 research and innovation programme (2014-2020) under grant agreement no. 825532.

References

[1] L. Eeckhout, "Heterogeneity in response to the power wall," *IEEE Micro*, vol. 35, no. 4, pp. 2–3, 2015.

[2] "High-performance conjugate gradient (hpcg) benchmark results," june 2020. [Online]. Available: https://www.top500.org/lists/hpcg/06/

[3] K. Li, "IVY: a shared virtual memory system for parallel computing," in *Proc. 1988 Intl. Conf. on Parallel Processing*, 1988, pp. 94–101.

[4] C. Amza, A. L. Cox, S. Dwarkadas, P. Keleher, Honghui Lu, R. Rajamony, Weimin Yu, and W. Zwaenepoel, "Treadmarks: shared memory computing on networks of workstations," *Computer*, vol. 29, no. 2, pp. 18–28, 1996.

[5] G. Antoniu, L. Bougé, and M. Jan, "JuxMem: An Adaptive Supportive Platform for Data Sharing on the Grid," *Scalable Computing : Practice and Experience*, vol. 6, no. 33, pp. 45–55, Nov. 2005, also available as an INRIA Research Report 4917: http://www.inria.fr/rrrt/rr-4917.html.

[6] J. A. Ross and D. A. Richie, "Implementing openshmem for the adapteva epiphany risc array processor," *Procedia Computer Science*, vol. 80, pp. 2353 – 2356, 2016, international Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA.

[7] J. Nelson, B. Holt, B. Myers, P. Briggs, L. Ceze, S. Kahan, and M. Oskin, "Latency-tolerant software distributed shared memory," in *2015 USENIX Annual Technical Conference (USENIX ATC 15)*. Santa Clara, CA: USENIX Association, 2015, pp. 291–305.

[8] S. Kaxiras, D. Klaftenegger, M. Norgren, A. Ros, and K. Sagonas, "Turning centralized coherence and distributed critical-section execution on their head: A new approach for scalable distributed shared memory," in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, 2015, pp. 3–14.

[9] L. Cudennec, "Software-Distributed Shared Memory over heterogeneous micro-server architecture," in *Euro-Par 2017: Parallel Processing Workshops*, 2017.

[10] ——, "Merging the Publish-Subscribe Pattern with the Shared Memory Paradigm," in *Euro-Par 2018: Parallel Processing Workshops*, 2018.

[11] J. Stuecheli, W. J. Starke, J. D. Irish, L. B. Arimilli, D. Dreps, B. Blaner, C. Wollbrink, and B. Allison, "IBM POWER9 opens up a new era of acceleration enablement: OpenCAPI," *IBM Journal of Research and Development*, vol. 62, no. 4/5, pp. 8:1–8:8, 2018.

[12] CCIX Consortium, "An introduction to CCIX: white paper," November 2019. [Online]. Available: https://www.ccixconsortium.com/wp-content/uploads/2019/11/CCIX-White-Paper-Rev111219.pdf

[13] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for fpgas: From prototyping to deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473–491, 2011.

[14] Y. S. Shao, S. L. Xi, V. Srinivasan, G. Wei, and D. Brooks, "Co-designing accelerators and soc interfaces using gem5-aladdin," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–12.

[15] J. Cong, Z. Fang, M. Gill, and G. Reinman, "Parade: A cycle-accurate full-system simulation platform for accelerator-rich architectural design and exploration," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2015, pp. 380–387.

[16] L. Feng, H. Liang, S. Sinha, and W. Zhang, "Heterosim: A heterogeneous cpu-fpga simulator," *IEEE Computer Architecture Letters*, vol. 16, no. 1, pp. 38–41, 2017.

[17] T. Liang, L. Feng, S. Sinha, and W. Zhang, "Paas: A system level simulator for heterogeneous computing architectures," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–8.

[18] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, p. 1–7, Aug. 2011.

[19] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, "Multi2sim: A simulation framework for cpu-gpu computing," in *2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2012, pp. 335–344.

[20] W. Snyder, "Verilator: the fast free verilog simulator," 2012. [Online]. Available: http://www.veripool.org

[21] C. M. Kirchsteiger, H. Schweitzer, C. Trummer, C. Steger, R. Weiss, and M. Pistauer, "A software performance simulation methodology for rapid system architecture exploration," in *2008 15th IEEE International Conference on Electronics, Circuits and Systems*, 2008, pp. 494–497.

[22] M. K. Papamichael, J. C. Hoe, and O. Mutlu, "Fist: A fast, lightweight, fpga-friendly packet latency estimator for noc modeling in full-system simulations," in *Proceedings of the Fifth ACM/IEEE International Symposium on Networks-on-Chip*, 2011, pp. 137–144.

[23] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, M. Eldafrawy, J.-P. Legault, E. Sha, A. G. Graham, J. Wu, M. J. P. Walker, H. Zeng, P. Patros, J. Luu, K. B. Kent, and V. Betz, "Vtr 8: High-performance cad and customizable fpga architecture modelling," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 13, no. 2, May 2020.

[24] S. Seeley, V. Sankaranaryanan, Z. Deveau, P. Patros, and K. B. Kent, "Simulation-based circuit-activity estimation for fpgas containing hard blocks," in *2017 International Symposium on Rapid System Prototyping (RSP)*, 2017, pp. 36–42.

[25] A. Wicaksana, A. Charif, C. Andriamisaina, and N. Ventroux, "Hybrid prototyping methodology for rapid system validation in hw/sw co-design," in *2019 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, 2019, pp. 35–40.

[26] F. G. Gustavson, "Two fast algorithms for sparse matrices: Multiplication and permuted transposition," *ACM Trans. Math. Softw.*, vol. 4, no. 3, p. 250–269, Sep. 1978.

[27] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, Dec. 2011.