ECOSCALE: Reconfigurable Computing and Runtime System for Future Exascale Systems

Iakovos Mavroidis[1], Ioannis Papaefstathiou[2], Luciano Lavagno[3], Dimitrios S. Nikolopoulos[4], Dirk Koch[5], John Goodacre[5], Ioannis Sourdis[6], Vassilis Papaefstathiou[6], Marcello Coppola[7], Manuel Palomino[8]

[1]Telecommunication Systems Institute, Greece
[2]Synelixis, Greece
[3]Politecnico di Torino, Italy
[4]Queen's University of Belfast, United Kingdom
[5]University of Manchester, United Kingdom
[6]Chalmers University of Technology, Sweden
[7]STMicroelectronics, France
[8]Acciona Infraestructuras S.A., Spain

Abstract: - *In order to reach exascale performance, current HPC systems need to be improved. Simple hardware scaling is not a feasible solution due to the increasing utility costs and power consumption limitations. Apart from improvements in implementation technology, what is needed is to refine the HPC application development flow as well as the system architecture of future HPC systems. ECOSCALE tackles these challenges by proposing a scalable programming environment and architecture, aiming to substantially reduce energy consumption as well as data traffic and latency. ECOSCALE introduces a novel heterogeneous energy-efficient hierarchical architecture, as well as a hybrid many-core+OpenCL programming environment and runtime system. The ECOSCALE approach is hierarchical and is expected to scale well by partitioning the physical system into multiple independent Workers (i.e. compute nodes). Workers are interconnected in a tree-like fashion and define a contiguous global address space that can be viewed either as a set of partitions in a Partitioned Global Address Space (PGAS), or as a set of nodes hierarchically interconnected via an MPI protocol. To further increase energy efficiency, as well as to provide resilience, the Workers employ reconfigurable accelerators mapped into the virtual address space utilizing a dual stage System Memory Management Unit with coherent memory access. The architecture supports shared partitioned reconfigurable resources accessed by any Worker in a PGAS partition, as well as automated hardware synthesis of these resources from an OpenCL-based programming model.*

1. Introduction

In order to sustain the ever-increasing demand of storing, transferring and processing data, HPC servers need to improve their efficiency. Scaling the number of cores alone is not a feasible solution any more due to the increasing utility costs and power consumption limitations. While current HPC systems can offer petaflop performance, their architecture limits their capabilities in terms of scalability and energy consumption. Extrapolating from the top HPC systems, such as China's Tianhe-2 Supercomputer, we estimate that sustaining exaflop performance requires an enormous 1GW power. Similar, albeit smaller, figures are obtained by extrapolating even the best system of the Green 500 list as an initial reference.

Apart from improving transistor and integration technology, important refinements in HPC application development and HPC architecture design are also needed. The ECOSCALE project [26] meets these challenges with a novel and holistic approach for Exascale computing, combining a hybrid manycore+OpenCL programming environment; a hierarchical architecture; an intelligent runtime system and middleware; and hardware support for sharing distributed and reconfigurable accelerators.

2. HPC Application Characteristics

The ECOSCALE architecture is tailored to the characteristics and trends of future HPC applications to efficiently scale to exaflop performance. In our attempt to predict future HPC applications, we envision that they will have the following fundamental characteristics:

1) Massive parallelism: Applications can be partitioned in many parallel tasks or threads that can run in parallel. A 1000x increase in today's concurrency will be necessary to achieve exascale throughput [1].

2) Data locality: HPC applications should expose spatial and temporal locality in order to scale. The data accessed by an HPC application will be partitioned in memory sub-domains in such a way that memory transfers between these sub-domains are infrequent and efficient.

Partitioning into subdomains will dictate the mapping of tasks to cores and accelerators, so that each task executes on a device that is directly "attached" to the data that the task accesses. Instead of a flat partitioning of the application domain, we foresee that future large-scale HPC applications will perform hierarchical and topological partitioning (such as a high-radix Dragonfly or Slimfly topology) [2] of their data into domains, to reduce communication distance and latency. A leaf node in this partitioning would correspond to a data domain that fits in the local memory of a single processor-accelerator bundle, which we coin as "worker node" in the rest of this paper. Moving up one level in the hierarchy, domains would map to data that fits in multi-worker chips. Further up one level, domains would map to data that fit in multi-chip nodes, and further up in multi-node chassis and cabinets. Starting from the leaves, each level up the tree would add one hop in the maximum communication distance between any two processing units.

978-3-9815370-6-2/DATE16/

Existing Petascale systems have a maximum distance of five hops and Exascale systems will push this distance to six or seven (possibly longer) hops, with a corresponding number of levels in the hierarchical partitioning. This hierarchical partitioning can significantly reduce the communication overhead and the mapping algorithm complexity to achieve scalability [3][4].

The programming model used by the HPC applications should also be considered in the architectural decisions (as well as in the specifications of the runtime system) in order to improve HPC efficiency. Although MPI has been the most popular programming model for developing parallel scientific applications, the PGAS programming model is an attractive alternative for designing applications with irregular communication patterns. It is widely believed that a hybrid flexible MPI+PGAS programming model is an efficient choice for many scientific computing problems and for achieving exascale computing [5]. Figure 1 illustrates the proposed ECOSCALE partitioning of future HPC applications, which uses such a hybrid many-core MPI+PGAS programming model. PGAS is used for efficient intra-partition communication where the number of cores is limited by the hop-costs of a specific system instantiation. Since PGAS and related task scheduling algorithms have important scaling problems, MPI can also be used for efficient inter-PGAS communication.

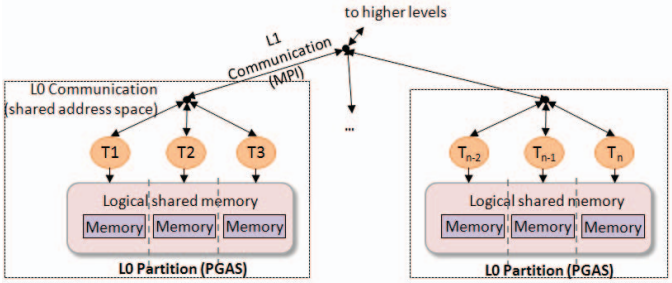

Figure 1.Example hierarchical partitioning (tasks, memory, communication) of an HPC application.

Exascale performance and energy-efficiency are also supported by the extensive use of reconfigurable accelerator technology and by the UNILOGIC (Unified Logic) architecture. The latter is introduced in this project for the first time as an extension of the UNIMEM architecture proposed in the EUROSERVER project [6]. UNIMEM provides shared partitioned global address space while UNILOGIC provides shared partitioned reconfigurable resources within the UNIMEM. The UNIMEM architecture gives the user the option to move tasks and processes close to data instead of moving data around [6] and thus it reduces significantly the data traffic and the associated energy consumption and communication latency. From the point of view of a processor in a multi-node machine, a memory page can be cacheable at the local coherent node or at a remote coherent node, but not at both. This is the basis of the UNIMEM consistency model, which eliminates global-scope cache coherence protocols providing a scalable solution. Progressive address translation [12] can be further applied on top of UNIMEM in order to provide interprocessor communication.

UNILOGIC adds to UNIMEM the capability to easily move the acceleration engine to local hardware, for instance through dynamic partial reconfiguration [7]. The proposed UNILOGIC+UNIMEM architecture partitions the design into several Worker nodes that communicate through a hierarchical communication infrastructure, similar to the one shown in Figure 1. These Worker nodes correspond to the partitions of the HPC application. Each Worker node is an entire sub-system including processing units, memory, and storage. Within a PGAS domain (consisting of several Workers), the proposed architecture offers (1) a shared global address space that can be partitioned for locality and (2) shared reconfigurable resources that can also access remotely cached data via regular load and store instructions, without using any global cache coherency mechanism to keep a local cache coherent. The ECOSCALE architecture and runtime system are further explained in Section 4.

3. Related Work

Recent advances allow the integration of reconfigurable hardware, alongside with general-purpose processors (CPUs) to form an efficient HPC system. Such systems combine the flexibility of the reconfigurable fabric with the general-purpose characteristic of CPUs. The advantage of such systems is that they can accelerate particular applications by mapping (parts of) them to reconfigurable hardware, substantially improving execution time and energy efficiency. Numerous such systems have been proposed [13][14][15]. The importance of the utilization of reconfigurable computing for future HPC systems is also demonstrated by the fact that there is a special National Science Foundation (NSF) research center in the USA consisting of more than 30 industrial and academic partners focusing only on this topic; the outcome is the most powerful HPC system in the world utilizing reconfigurable technology (Novo-G) [21].

Various HPC systems incorporate reconfigurable fabric into their computing nodes. Some of them integrate FPGAs directly on a system bus such as AMD's HyperTransport or Intel's Front Side Bus [16] and QuickPath Interconnect (QPI). Their main drawback is that the FPGA device replaces one CPU and thus it reduces the overall system performance for applications that cannot benefit from reconfigurable computing. Then, several large systems have also been built with distributed FPGAs, including the Cray XD-1, Novo-G, Maxwell [22] and QP [23] . These systems integrate the FPGA with the CPU, while the FPGA-to-FPGA communication must be routed through the CPU.

Moreover, some commercial systems provide FPGA-based supercomputing nodes, namely Maxeler's MPC series, Convey's HC-2, BeeCube's BEE4, SRC's MAPstation, and Timelogic's DeCypher [24]. These systems are using complex multi-FPGA infrastructures and were proven to be very efficient for certain applications e.g., the Convey HC-1 server has been used to accelerate data mining workloads using the CART algorithm for decision tree classification in big-data applications [17], while similar systems from Maxeler Technologies are used for financial applications [18]. The incorporation of

reconfigurable devices into HPC systems is expected to expand since FPGAs are getting significantly cheaper, simpler to program and more powerful over time.

Moreover, Microsoft has recently implemented their first CPU-FPGA server system called Catapult, which will be incorporated in its Bing data centers, and achieves a 40x speedup [19]. Similarly, IBM is rolling out FPGA-based data centers for data analytics onto the market.

Finally, reconfigurable resources have recently been integrated on the same chip as conventional multi-cores, creating a very powerful processing unit; both Xilinx and Altera have created FPGAs that contain dual core 32-bit ARM CPUs. More importantly Intel has recently announced the introduction of a Xeon Chip with an integrated reconfigurable fabric that is stated to achieve a 20x performance improvement for various applications when compared with a standard Xeon chip, while Altera has recently introduced a chip with a Quad Core 64-bit ARM coupled with a large amount of reconfigurable resources. However, all these approaches treat the FPGA as either a local accelerator for a single processing node, or as an independent node in an overall heterogeneous processing system.

4. ECOSCALE Approach

ECOSCALE aims to provide a novel methodology and architecture to automatically execute HPC applications onto an HPC platform that supports thousands or millions of reconfigurable hardware blocks, while taking into account the projected trends and characteristics of HPC applications. Within this context, ECOSCALE aims at placing FPGA-based acceleration as an integrated peer of the processing nodes within the UNIMEM system architecture and adapting them to work in an HPC environment. Thus its novel framework provides the locality and scalability model for FPGA-based acceleration from the ground up. In order to efficiently do so, we follow a holistic approach providing solutions for all aspects of an HPC environment, ranging from architecture and runtime management and optimization, to high level synthesis (HLS) and hardware virtualization.

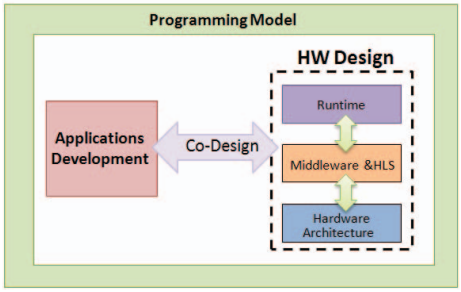

Figure 2. The ECOSCALE framework.

The proposed HW design consists of a stack of three interdependent HW abstraction layers, as shown in Figure 2. At the bottom layer, the proposed hardware architecture provides the basic hardware components and functionality in order to efficiently use the HW resources (CPU, memory, Reconfigurable Hardware, etc.) in an HPC system. In the middle layer, a middleware provides the primitives to reconfigure hardware blocks at runtime, while an HLS tool provides the synthesized application tasks to the middleware. In the top layer a runtime system schedules tasks inside a PGAS partition, provides the MPI primitives for communication between PGAS partitions and decides at run-time *which* functions of the accelerated application should be implemented and executed in reconfigurable hardware, and *where* data should be placed for locality.

The aforementioned layers and programming model are described in more detail below.

4.1. The ECOSCALE Architecture

This novel system architecture uses CPUs, memory and reconfigurable blocks in a highly parallel manner. Driven by the characteristics and trends of future HPC applications and following the high-radix partitioning of an HPC application (see Figure 1), the proposed UNILOGIC+UNIMEM architecture logically partitions hierarchically the hardware resources (CPUs, reconfigurable logic, memories, SSDs) into several interconnected Compute Nodes (corresponding to the PGAS partitions of the application) which are further partitioned into several Worker nodes, depending on the physical structure of the system. Thus, one or more Compute Nodes create an entire and independent PGAS sub-system including several Worker nodes and offer:

1) UNIMEM: a shared partitioned global address space that allows Worker nodes to communicate via regular loads and stores without global cache coherence and

2) UNILOGIC: shared partitioned reconfigurable resources that share the UNIMEM space with software tasks.

Other existing architectures either require a global cache coherent mechanism, which simply cannot scale, or support only DMA operations, which are not efficient for small data transfers such as messages to synchronize remote threads or to configure a remote peripheral [23]. The UNIMEM architecture allows moving tasks and processes close to data instead of moving data around [7].

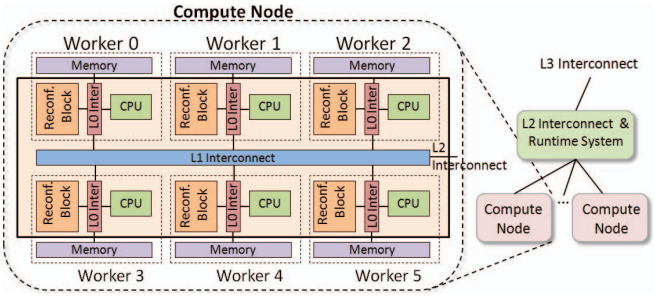

Figure 3. The ECOSCALE Hardware Architecture.

The proposed HPC architecture, where (1) each Compute Node is a PGAS sub-system providing a shared address space and reconfigurable acceleration logic, and (2) MPI is used for communication between Compute Nodes via CPU-based routers following the application topology, is shown

in Figure 3. It consists of several Worker nodes communicating through a multi-layer interconnection. The actual number of Workers inside a Compute Node depends on the integration capabilities of future technologies. Each Worker is an independent computing unit that can execute, fork, and join tasks or threads of an HPC application in parallel with the other Workers. It includes a CPU, a reconfigurable block and an off-chip DRAM memory. The communication and synchronization between the Workers is performed through a multi-layer interconnection, which allows load and store commands, DMA operations, interrupts, and synchronization between the Workers of a Compute Node (following the UNIMEM architecture). The Compute Nodes are PGAS sub-systems that correspond to the application's PGAS-based partitions shown in Figure 1. Matching the application logical topology of Figure 1, the Compute Nodes are interconnected through an MPI-based multi-layer interconnection.

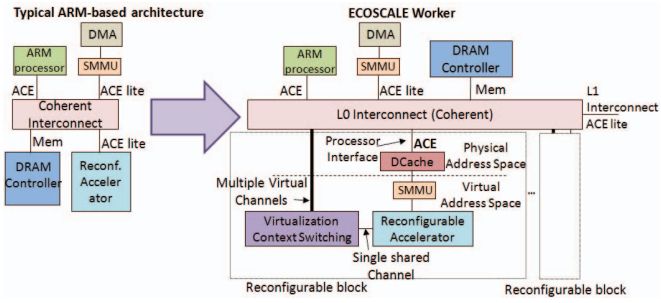

Figure 4. Block diagram of ECOSCALE Worker.

The communication overhead between a CPU and a hardware accelerator, i.e. the Reconfigurable Block inside a Worker node, is one of the most crucial challenges. A few years ago only explicit memory transfers between the host memory and the accelerator's memory were supported, like in a GP-GPU. Recent technological advances allow the integration of the host CPU and hardware accelerators on the same chip, and thus hardware accelerators can now access directly the host memory. Such a typical ARM-based system [25] is depicted on the left of Figure 4. However, there are still important limitations, as described below, that we will tackle in this project.

In the example state-of-the-art architecture (Figure 4), the ARM Cache Coherent Interconnect supports two types of coherent ports in order to provide hardware coherency in the system: (1) ACE ports, which can be used by masters containing caches, such as a processor, and (2) ACE-lite ports, which can be used by masters that do not have hardware coherent caches. ACE-lite ports are traditionally used for hardware accelerators such as GPUs and FPGAs, as shown in the figure.

Figure 4 (right side) shows the block diagram of our Worker node. The proposed architecture will extend such a typical architecture as follows. Accelerator blocks act as what is known in UNIMEM as a Unit of Compute, and hence they can interface directly with any other UNIMEM units of compute where each unit caches its local data coherently. Each accelerator can also cache its local data and likewise provide coherent access from remote UNIMEM units. If a single accelerator block needs to span across multiple FPGA local memories, then the FPGA units can provide their own coherence schemes independent of UNIMEM.

The reconfigurable resources are typically configured to use physical addresses in order to access shared variables. Since only the OS (or the hypervisor in virtualized systems) has access to the physical address space, the intervention of the OS (or the hypervisor) is unavoidable. A dual stage I/O MMU, such as the ARM SMMU shown in Figure 4, can resolve this problem by translating virtual addresses to physical addresses in hardware. Using an I/O MMU the proposed architecture will allow *"user-level access"* to the reconfigurable accelerators.

Virtualization and context switching enables multiple tasks or threads of an HPC application to share a single CPU in order to maximize the utilization of the CPU resources. Similarly, our architecture will support coarse-grain time-sharing of the reconfigurable resources through partial runtime reconfiguration. Moreover, it will support fine-grain sharing of those FPGA resources, where a function implemented in hardware can be "called" by different tasks or threads of an HPC application in parallel, through the Virtualization block shown in Figure 4. The Virtualization block and the HLS tool provide a mechanism to execute multiple function calls (from different virtual machines) in a fully pipelined fashion.

Sharing of the limited reconfigurable resources between Workers is very important. Thus, within a Compute Node, any Worker can access any Reconfigurable block (even remote blocks that belong to other Workers) through the multi-layer interconnect shown in Figure 3. Moreover, the L0 Interconnect in this example system provides an external ACE-lite port (connection to L1 interconnect in Figure 4) that can be used by remote Reconfigurable blocks to make coherent accesses. However, since this is not an ACE port (no snooping protocol is supported) the remote Reconfigurable block should disable its data cache (and would not be as efficient as a local one).

4.2. Runtime System

The ECOSCALE runtime environment will extend current OpenCL frameworks in three ways. First, by supporting a partitioned global address space within and between ECOSCALE workers and nodes, via the introduction of new data scoping and consistency abstractions in OpenCL. Second, by extending the semantics and providing a scalable and efficient implementation of OpenCL data transfers between partitions of the address space. This will be in addition to data transfers between devices (CPUs and reconfigurable subsystems) within the same address space, by using direct loads and stores from and to remote shared memories. Third, by allowing the programmer to specify functions that can be synthesized in hardware and can be accelerated, on-demand, at runtime, depending on the dynamic execution conditions of the system. ECOSCALE will explore and implement new algorithms to dynamically partition

computation between CPU cores and hardware accelerators on the ECOSCALE nodes.

ECOSCALE will also explore new methods and models for monitoring the execution time complexity and energy consumption of tasks on CPUs and reconfigurable systems, as well as new algorithms for choosing on the fly the most appropriate device to execute each function. We will specifically develop input-dependent models of execution time and energy to select the best device to execute a function. The models will attempt to capture the correlation between input/output size, input/output data shape (when available), and data access pattern in memory (model inputs) and execution time and power consumption (model outputs) using one or more CPU cores or accelerator(s). The models will be co-designed with the application use cases of ECOSCALE. This effort entails three parts. A first, training part will use the target applications with different realistic inputs, in order to capture static and dynamic properties of the input and record the corresponding execution time and power outputs. A second, model building part will entail an exploration of models for predicting outputs from inputs. We intend to use an array of regression, SVM and PCA techniques for this purpose, building on prior experience on models for predicting execution time and power for the purpose of multi-dimensional program adaptation [8]. A third, actuation part, will deploy the models with actual running applications, using hardware performance monitors and function instrumentation to capture the static and dynamic properties of the unseen input, and project execution time and power using the trained models with hardwired parameters. This will enable the runtime scheduler to judiciously and dynamically select and distribute functions for hardware acceleration.

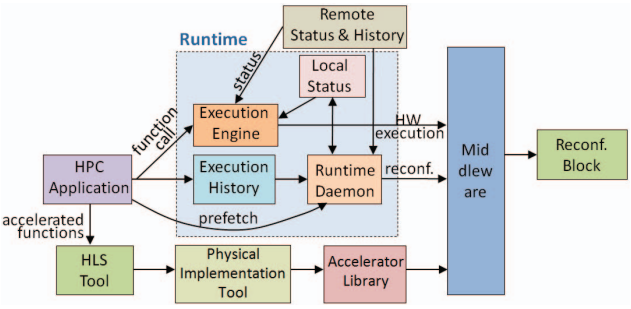

Figure 5. Interaction and control flow between the three abstraction layers.

We will also explore methods to minimize non-overlapped communication latency within each ECOSCALE node and across nodes in the same cluster. We will implement one scheduler per worker, which will manage the local reconfigurable blocks and the execution of the accelerated functions. Whenever a function is called, a work and data distribution algorithm in the runtime system (included in the Execution Engine in Figure 6) will decide whether the function will be executed in software or in hardware based on the local status and the status of other Workers in the vicinity. To curb the overhead of monitoring remote status, we will implement local work queues per worker and infer (approximately) the status of remote workers via the status of the local queue, using techniques inspired by Lazy Scheduling [9]. A history of the function calls as well as their execution time is stored in a History file (Execution History block). The runtime scheduler/daemon will read periodically the system status and the History file in order to decide at runtime what functions should be loaded on the reconfiguration block.

4.3. Middleware and High Level Synthesis

The middleware bridges the gap between the reconfigurable hardware and the software parts of the full application, providing the means to enable a fully software-driven development flow. The middleware will play two main roles, namely providing the partial-reconfiguration toolset and the SW-HW communication library. Thus, first, it performs partial reconfiguration at runtime. This includes the development of a low level driver backend that will add virtualization features, such as defragmenting the reconfigurable resources, accelerator migration, and pre-emptive hardware execution. Second, it provides a communication library and API in order to call any function that is implemented in hardware.

ECOSCALE will support hardware-assisted virtualization in order to increase the performance and to lower the power consumption. We will both extend the HLS tool developed in the FASTCUDA project [20], and exploit recently introduced industrial tools (e.g. the SDAccel tool flow from Xilinx). The main focus will be on effectively exploiting the huge cost/performance trade-off space provided by ECOSCALE, while requiring minimal intervention and no specific hardware design experience from the programmer.

The ECOSCALE HLS tool will tackle this problem by providing a way to specify performance and area constraints, and then automatically exploring high-performance hardware implementation techniques, such as pipelining, loop unrolling, as well as data storage and data-path partitioning and duplication, starting from a non-hardware specific OpenCL model. Current HLS tools require an experienced designer to take architectural decisions, such as the DRAM port parallelism, the local data memory partitioning, and so on. These will be automated as much as possible (while still retaining designer control, if and when needed).

The tool will generate at compile time a library with the hardware implementations of those functions that will be implemented on reconfigurable resources. These implementations will be transformed with the help of a Physical Implementation Tool, which will extend the existing GoAhead framework [10], automatically into an accelerator module library. This includes the steps of resource budgeting, floorplanning, communication infrastructure synthesis and physical constraint generation for the reconfigurable fabric's vendor place and route tools, as well as the final partial bitstream assembly. By minimizing module bounding boxes and by using configuration data compression [11], we will reduce

memory requirements, configuration latency and configuration power consumption at the same time.

At runtime, the system can use this library in a very flexible manner. For example, we consider chaining together different accelerator modules for building longer complex processing pipelines, when needed. This will substantially increase the amount of processing that is carried out per unit of transferred data and will consequently result in substantial energy savings.

4.4. Programming Model

ECOSCALE will develop co-designed HPC applications based on hierarchical data partitioning to achieve locality and reduced data traffic and associated power consumption and latency. The programming model for expressing hierarchical data partitioning will start from the widely used MPI-3.0 standard, leveraging the new topology abstractions. The focus of the programming model efforts in ECOSCALE is on two directions. The first is to extend OpenCL to support multiple workers ("devices" in OpenCL nomenclature), distributed command queues and transparent command queue management across workers in a node. The second direction is on providing a transparent substrate to achieve data access locality from OpenCL instances running in workers. We will treat the global memory in each compute node as a collection of NUMA domains accessible via the UNIMEM interface. We will explore topology-aware global memory allocators in these domains, to be used by the OpenCL runtime for implicit data allocation, migration and replication between workers.

5. Conclusion

Today's technologies and architectures cannot efficiently scale to exascale. A holistic approach tailored to the characteristics and trends of future HPC application is required. Towards this end, ECOSCALE employs a novel hardware architecture, runtime system and programming model in order to be able to directly map the HPC application hierarchical structures to hardware resources, while in parallel it takes full advantage of energy-efficient reconfigurable computing. ECOSCALE will provide solutions for all the aspects of an HPC environment, ranging from architecture and runtime optimizations, to partial reconfiguration, HLS and hardware virtualization.

6. Acknowledgement

This research project is supported by the European Commission under the H2020 Programme and the ECOSCALE project (grant agreement 671632).

References

[1] J. Dongarra, P. Beckman, T. Moore, and P. e. a., "*The International Exascale Software Project Roadmap"*, IJHPCA., 25(1):3–60, Feb. 2011.

[2] B. Prisacari, G. Rodriguez, P. Heidelberger, D. Chen, C. Minkenberg, and T. Hoefler, "*Efficient task placement and routing of nearest neighbor exchanges in dragonfly networks",* ACM, 23rd HPDC, 2014.

[3] I-Hsin Chung, C.-R. Lee, J. Zhou, and Y.-Ching Chung, "*Hierarchical Mapping for HPC Applications",* Parallel Processing Letters, 2011.

[4] A. Abdel-Gawad, M. Thottethodi, and A. Bhatele, "*RAHTM: Routing-Algorithm Aware Hierarchical Task Mapping",* SC, 2014.

[5] J. Jose, S. Potluri, H. Subramoni, X. Lu, e.a. "*Designing Scalable Out-of-core Sorting with Hybrid MPI+PGAS Programming Models",* 8th PGAS, 2014.

[6] Y. Durand et al., *"EUROSERVER: Energy Efficient Node for European Micro-servers",* Euromicro DSD, 2014.

[7] D. Koch, "*Partial Reconfiguration on FPGAs – Architectures, Tools and Applications",* Springer, 2012.

[8] M. Curtis-Maury, A. Shah, F. Blagojevic, D. Nikolopoulos, B. de Supinski, M. Schulz, "*Prediction models for multi-dimensional power-performance optimization on many cores",* PACT, pp. 250-259, 2008.

[9] A. Tzannes, G. C. Caragea, U. Vishkin, e.a., "*Lazy Scheduling: A Runtime Adaptive Scheduler for Declarative Parallelism"* ACM Trans. Program. Lang. Syst , 2014.

[10] C. Beckhoff, D. Koch and J. Torresen, "*GoAhead: A Partial Reconfiguration Framework"*, FCCM, 2012.

[11] Dirk Koch, Christian Beckhoff and Jurgen Teich, "*Hardware Decompression Techniques for FPGA-based Embedded Systems",* ACM TRETS, 2009.

[12] M. Katevenis, *"Interprocessor communication seen as load-store instruction generalization",* K. Bertels e.a. (Eds.), Delft, '07.

[13] D. Burke, et al., *"RAMP blue: Implementation of a manycore 1008 processor system"*, RSSI, 2008.

[14]A. P. Michael Showerman and J. Enos, *"QP: a heterogeneous multi- accelerator cluster",* SC, 2010.

[15] P. P. Kuen Hung Tsoi, A. Tse, and W. Luk, *"Programming framework for clusters with heterogeneous accelerators"*, Highly-Efficient Accelerators and Reconfigurable Technologies, 2010.

[16] L. Ling et al*, "High-performance, Energy-efficient Platforms Using In-socket FPGA Accelerators",* FPGA, 2009.

[17] G. Chrysos, P. Dagritzikos, I. Papaefstathiou, and A. Dollas. HC-CART, *"A parallel system implementation of data mining classification and regression tree (CART) algorithm on a multi-FPGA system"*, TACO, 2013.

[18] Q. Jin, D. Dong, A. H. T. Tse, G. C. T-Chow, D. B. Thomas, W. Luk, and S. Weston, *"Multi-level Customisation Framework for Curve Based Monte Carlo Financial Simulations"*, ARC, 2012

[19] http://www.wired.com/2014/06/microsoft-fpga/.

[20] I. Mavroidis., I. Papaefstathiou, L. Lavagno, e.a., *"FASTCUDA: Open Source FPGA Accelerator & Hardware-Software Codesign Toolset for CUDA Kernels",* DSD, 2012.

[21] http://www.chrec.org/ngforum/

[22] R. Baxter et al., *"Maxwell - a 64 FPGA Supercomputer",* Engineering Letters, 2008

[23] M. Showerman et al., *"QP: A Heterogeneous Multi-Accelerator Cluster",* High-Performance Clustered Computing,'09

[24] *http://www.timelogic.com/catalog/752/biocomputing-platforms.*

[25] *CoreLink CCI-400 Cache Coherent Interconnect: http://www.arm.com/products/system-ip/interconnect/corelink-cci-400.php.*

[26] Website of ECOSCALE: http://www.ecoscale.eu/