Low Power and Scalable Many-Core Architecture for Big-Data Stream Computing

Karim Kanoun†, Martino Ruggiero†, David Atienza†, and Mihaela van der Schaar⋆
† Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland. ⋆ University of California, Los Angeles (UCLA), U.S.A. email: {karim.kanoun, martino.ruggiero, david.atienza}@epfl.ch, mihaela@ee.ucla.edu.

***Abstract*—In the last years the process of examining large amounts of different types of data, or Big-Data, in an effort to uncover hidden patterns or unknown correlations has become a major need in our society. In this context, stream mining applications are now widely used in several domains such as financial analysis, video annotation, surveillance, medical services, traffic prediction, etc. In order to cope with the Big-Data stream input and its high variability, modern stream mining applications implement systems with heterogeneous classifiers and adapt online to its input data stream characteristics variation. Moreover, unlike existing architectures for video processing and compression applications, where the processing units are reconfigurable in terms of parameters and possibly even functions as the input data is changing, in Big-Data stream mining applications the complete computing pipeline is changing, as entirely new classifiers and processing functions are invoked depending on the input stream. As a result, new approaches of reconfigurable hardware platform architectures are needed to handle Big-Data streams. However, hardware solutions that have been proposed so far for stream mining applications either target high performance computing without any power consideration (i.e., limiting their applicability in small-scale computing infrastructures or current embedded systems), or they are simply dedicated to a specific learning algorithm (i.e., limited to run with a single type of classifiers). Therefore, in this paper we propose a novel low-power many-core architecture for stream mining applications that is able to cope with the dynamic data-driven nature of stream mining applications while consuming limited power. Our exploration indicates that this new proposed architecture is able to adapt to different classifiers complexities thanks to its multiple scalable vector processing units and their re-configurability feature at run-time. Moreover, our platform architecture includes a memory hierarchy optimized for Big-Data streaming and implements modern fine-grained power management techniques over all the different types of cores allowing then minimum energy consumption for each type of executed classifier.**

I. Introduction

Emerging Big-Data stream computing applications, such as, stream mining applications (e.g., social media analysis, financial analysis, video annotation, surveillance, medical services and traffic prediction domains) have stringent delay constraints, dynamic data-driven topology graphs of classifiers, stochastic input data stream characteristics and are highly demanding in terms of parallel data computation.

Stream mining applications [20] are used to classify a high input of data stream and are in general modeled with a Directed-Acyclic-Graph (DAG) where each node denotes a task (e.g., classifier, features-extraction task), each edge from node j to node k indicates that task k execution depends on task j execution output. Fig. 1 illustrates an example of Big-Data processing in the context of large-scale healthcare data analytics [22]. It shows how the data collected from medical devices are categorized by means of features identification and selection, and multiple types of classifiers are applied to make the final predictions and define the actions to be performed. Unlike traditional tasks, a stream processing task is open-ended as it can continue to execute as long as input data stream are available. Each task can be seen as a pipeline stage that extracts valuable information from the stream input in real time and propagate to the following one.

In order to cope with the Big-Data stream application model, several hardware and software approaches [1][2][4][6][7][9][10][11][12] have been proposed to increase the performance of Big-Data streaming applications execution. All these solutions have focused on either optimizing a specific classifier (e.g., Support Vector Machines or SVM) to an existing hardware solution (e.g., Graphics Processing Unit or GPU [10] and Cell Broadband Engine [9]) where several optimizations are applied on the application layer in order to take advantage of the available vector units, or the other way around where hardware-accelerators are designed to accelerate the execution of specific classifiers [11][12]. While these approaches showed promising results, solutions based on GPU and high performance processor target high performance computing without any power consideration.

Moreover, these solutions work very well if the topology of the classifiers used in the stream mining application is already known in advance and the type of input data stream does not change over the time. However, modern stream mining applications implement systems with heterogeneous classifiers and adapt online to its input data stream characteristics variation. For instance, in [8], an online learning algorithm is proposed to optimize the classifiers chain for Big-Data stream mining problems. The optimal classifiers topology is then selected online on the fly with respect to the non-stationary input stream which is unknown at run-time. Thus, the complete computing pipeline changes dynamically to adapt to the input stream [8]. New classifiers and processing functions are then invoked. None of the proposed platform solutions so far have considered this additional layer that can control the classifiers graph on the fly and adapt the classifiers pipeline to the type of data.

As a result, in this paper, we propose a novel low-power many-core architecture for modern Big-Data stream mining applications that is able to cope with the dynamic nature of

This work was supported in part by a Joint Research Grant for ESL-EPFL by CSEM, and the BodyPoweredSenSE (no. 20NA21 143069) RTD projects evaluated by the Swiss NSF and funded by Nano-Tera.ch with Swiss Confederation financing and in part by US Air Force Office of Scientific Research under the DDDAS Program.

978-1-4799-3765-3/14 $31.00 © 2014 IEEE
DOI 10.1109/ISVLSI.2014.77

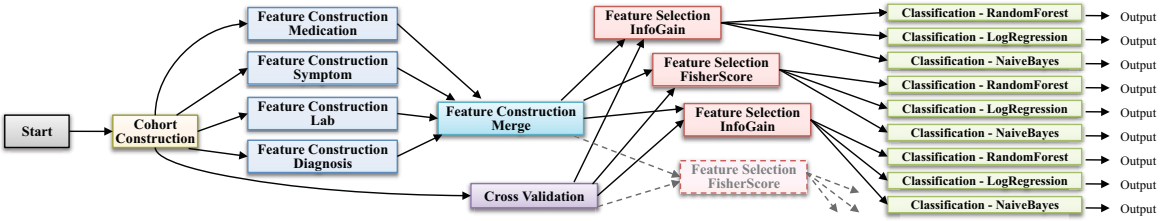

Fig. 1. Big-Data stream computing applications: example of Big-Data processing in the context of large-scale healthcare data analytics [22]

the input data stream while consuming limited power. The key contributions of this work are as follows:

• Our solution integrates low-power, scalable and reconfigurable cores able to adapt to the characteristic of each classifier.

• Memory hierarchy optimized to manage multiple concurrent Big-Data stream and able to adapt to the dynamic data-driven nature of modern stream mining applications.

Our results for machine learning algorithms running in parallels demonstrate that our memory hierarchy solution allows to significantly reduce the data memory access conflicts even for a high number of active cores in the same cluster.

The remainder of this paper is organized as follows. In Section II, we describe the limitations of current platforms dedicated to Big-Data streaming applications. In Section III, we describe our proposed low-power many-core architecture. In Section IV, we explain how modern stream mining applications with highly variable stream input can exploit the different features provided by our proposed platform. In Section V, we present our experimental results. Finally, we summarize the main conclusions in Section VI.

II. Related work

A. General stream processors

Several stream processor architectures [3], such as Imagine [1], Merrimac [2] and Storm [4] have been proposed for streaming applications with large amount of parallel computations. These cores share the same generic stream processor architecture where the functional units are distributed in ALU clusters and the memory hierarchy is partitioned into three levels. The Local Register File (LRF) is the first level used for local data communication and fast access of temporary data by the functional units. Then, the second level Stream Register File (SRF) is used to store streams and transfer data between the LRFs of major components. Finally the off-chip memory is used for global data. Stream processors are used as co-processors, where the scalar code is executed on a host processor and only the kernel is mapped to the ALU of a streaming processor. A more advanced processors have been also used for stream processing, the Cell Broadband Engine (Cell BE) [13], a multicore platform developed by IBM, Sony and Toshiba, was also used for stream processing. Cell BE integrates 8 SPEs (Synergistic Processing Elements) connected by a bidirectional ring bus and controlled by one PPE (Power Processing Element). Unlike the clusters of stream processor, each SPE may run a different kernel. Finally, a recent work in [7], integrating an Atom processor coupled to an ION GPU and a FPGA accelerator, studied an energy-efficient system for embedded learning and classification application.

B. Acceleration of machine learning applications

We describe existing work realized to accelerate the execution of machine learning applications in two categories. In the first set of existing work, classifiers were optimized in order to take advantage of either the stream processors, the Cell Broadband Engine or the GPU unit. In [10], the authors presented a fast, parallel, large-scale, high-level semantic concept detector that leverages the GPU for image/video retrieval and content analysis. In [9], an optimized version of the SVM classifier has been proposed for the Cell processor architecture. In contrast to these software-based optimization approaches, in the second set of existing work, it is the other way around where a VLSI implementation of new designed hardware on FPGA are proposed in order to accelerate the execution of the targeted classifier. In [12], a VLSI implementation of the real-time learning and recognition system based on adaptive K-Means learning algorithm has been implemented on FPGA. In [11], the authors proposed an optimized hardware architecture that performs object detection using Support Vector Machines. Finally, a novel many-core solution, called SmyleVideo [6], has been also proposed for video mining applications. SmyleVideo was architected as clustered heterogeneous cores optimized for video applications while the software layer was architected as a distributed processing model based on Kahn Process Network.

We contend that most of the proposed solutions addressed the problem of optimizing the execution of specific classifiers by either optimizing the application layer or adding new hardware accelerators. However, regarding the targeted type of data, in existing solutions the data was considered Big only in terms of its volume. However, in modern stream mining applications, data is Big because it also changes continuously over time. Big-Data are then accumulated in time and changes in a non-stationary way. Thus, existing solutions are not able to cope with modern stream mining applications where the optimal classifiers topology configuration is selected online on the fly and the complete computing pipeline is changing, as entirely new type of classifiers can be invoked to adapt to the variation of the input data which is unknown at tun-time [8].

III. Proposed low-power many-core architecture

A. Overview

Our proposed low-power many-core architecture is composed of clusters connected with a Network on Chip (NoC) [21]. Each cluster is composed of M cores and an optimized memory hierarchy for Big-Data streaming applications. In our platform, we propose to use the *icyflex4* core [5], a low-power core that can adapt to different algorithms thanks to its reconfigurable capabilities. Moreover, the icyflex4 provides 2 dedicated 512-bit ports for data. Thus, we propose to use one port for the stream reference and the application local data with a multi-level memory architecture with both private and shared cache, while the second port is used to pass the data between the cores through multiple First In First Out (FIFO) buffers using also a multi-level memory architecture with both private and shared memory and Direct Memory

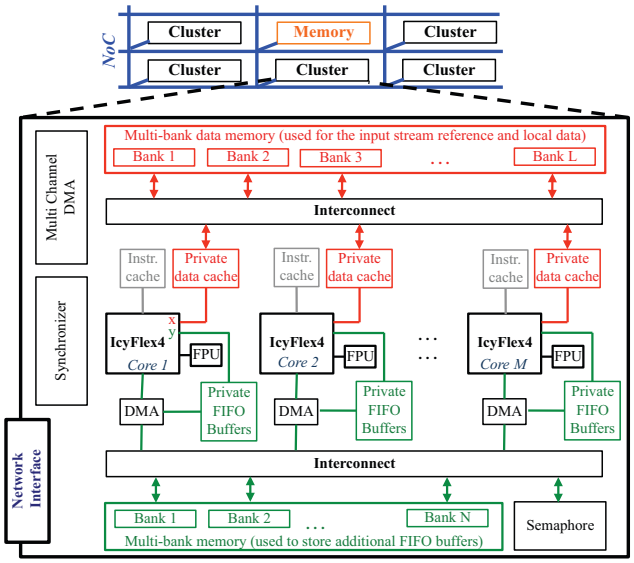

Fig. 2. The proposed low-power many-core architecture

Access (DMA) units. Each core has its own private instruction memory. Fig. 2 illustrates the proposed many-core solution for Big-Data stream computing. In this architecture, the cores operate as pipeline stages at the classifiers or feature-extraction level for stream mining applications and at the tasks level for general streaming applications. Thus, depending on the complexity of the application and the available resources, a core can execute one or more classifiers (or feature extraction algorithm). All these decisions are made by a scheduler at a higher layer. The producer consumer mechanism is handled through multiple FIFO buffers stored in the private and shared multi-bank memory accessed via the second data bus.

We discuss the architecture features of icyflex4 core and the memory hierarchy of the proposed platform in detail in the next two Sections. Then, we explain the reconfigurability feature of icyflex4 core and the relation between the FIFO buffers, the proposed memory hierarchy and the dependencies between the different tasks in Section IV.

B. Reconfigurable low-power icyflex4 core

The icyflex4 core [5] is a low-power DSP core with a Vector Processing Unit (VPU) composed of 8 slices. The datapath of a single slice contains sixteen 64-bit registers and eight 64-bit accumulators. A quad 16-bit multiplier and a recombination unit can perform a complex 16x16 bit multiplication or a scalar 32x32 bit multiplication at each cycle. The VPU slice additionally contains a 64-bit arithmetic and logic unit (ALU), a 64-bit barrel shifter and a 64-bit move unit, each able to compute up to four 16-bit operations per cycle. The data move unit (DMU) contains two addressing units which drive two dedicated data busses called X and Y with up to 512-bit data access each, allowing then higher throughput. The DPU and the VPU are reconfigurable at run time by means of the pre-configured addressing modes and the micro-operation (MOP) mechanism. In fact, the user is thus able to generate completely new complex instructions exploiting very efficiently all the computational units present in the VPU and in the DMU. This feature can be used to adapt each core to its mapped classifier/feature-extraction algorithm in order to minimize the clock cycles and power consumption. Moreover, the icyflex4 integrates several dynamic power management features such as DVFS or switching off unused computational units. Finally, while the icyflex4 core supports fixed-point operations, it also allows to connect a Floating Point Unit (FPU) as a co-processor. Thus a connected FPU unit can be switched on and off on demand depending on the mapped application.

C. Memory hierarchy dedicated for streaming applications on many-core platforms

In a streaming application, each task works on 3 types of data, namely, its private set of data, the stream input reference and the data provided through the other connected tasks. Thus, in our solution, we propose to separate the local task data and the stream reference from the data coming from connected tasks by using the two data memory ports of the icyflex4 core connected to two different memory hierarchies. Each of these two memory hierarchies is optimized for its mapped data type.

We connect the first data-memory port to a multi-level memory architecture with both private and multi-bank shared cache. These memory banks are dedicated to the input stream reference and the local data of each task. By adding an L1 private data cache for each core, this can significantly minimize the access conflicts to shared memory as each core will work most of the time with its private cache. Moreover, there is no need for a cache coherency protocol. This is guaranteed by the application model where the edges connecting the node of the graph model the only shared data in the application. In the next paragraph, we describe how we handle separately these shared data between the cores.

In our proposed architecture, we propose to pass these shared data through dedicated FIFO buffers managed with a separated memory hierarchy through the second data memory port available in the icyflex4 core. Our second multi-level memory architecture is composed of private memories, shared multi-bank memory, DMA and a semaphore unit. The private memories are used as a private FIFO buffers and the communication between the cores is realized through a producer consumer mechanism (e.g., push or pull request) [14]. In the following, we explain the main reasons and benefits behind our choice of having private and shared data memory to manage the FIFO buffers. First, using only private data memory adds restriction on the size and the number of FIFO buffers per core while the FIFO buffers requirement for each core is different and vary at run-time in modern stream mining applications [8]. Therefore, using a second memory shared between the cores gives then more flexibility to partition at run-time the memory among the cores. Second, using only shared memory creates significant memory access conflicts as several cores may access the same memory bank at the same time, thus using a private memory to pull with DMA the required prepared data from either the other private memory or the shared one for the next execution is more efficient. Third, by using both private and shared memory, this will not only provide higher flexibility and performance but it will be also efficient in terms of energy consumption. In fact, by optimizing the distribution of the FIFO buffers among the available memories, the unused shared memory banks can be switched off to save energy.

Finally, by separating the memory access between the input stream reference, the local task data and the FIFO buffers, this removes the need of cache coherency protocol and minimizes

the access delay and latency to data memory allowing then higher throughput. Moreover, by accessing both data memory hierarchies in parallel, kernels loops using both type of data as input can run more efficiently.

IV. Exploiting the proposed architecture for Big-Data stream mining

In this section, we explain the different mechanisms and techniques that our proposed low-power many-core architecture can offer to Big-Data streaming applications. We first describe how the icyflex4 core can be reconfigured to the characteristics of its mapped tasks. Then, we explain how the FIFO buffers are mapped to the proposed memory hierarchy of many-core platform. Finally, we formulate the problem of energy-efficient scheduling algorithm of streaming applications for the proposed platform. The overall mapping flow of an application is illustrated in Fig. 3.

A. Optimizing the IcycFlex4 for each mapped task

As already described in Section III-B, the slices of the Vector Processing Unit (VPU) integrate several computational units. The datapath configuration defining the instruction behaviour in these slices is specified in one of the eight available micro-operation registers. Configuring a new micro-operation can be done in only two cycles. The micro-operation instruction can be invoked by specifying one of the configured micro-operation registers and additional parameter (register indexes, immediate values, etc.). Therefore, a programmer can create new instructions targeting specific datapaths to accelerate the execution of its application kernel.

We propose to exploit the computational units available in each slice in two different ways: the standard method implements micro-operations that target multiple operations at the same time and which use data related to the same iteration of the kernel loop. However, in some cases where the operations (e.g, subtraction and multiplication) inputs depends on each other, the exploitation of the VPU unit using this first method can be limited. For instance if operation B (e.g., multiplication) input uses the results of a subtraction realized in operation A, it will not be possible to execute both operations in parallel. Thus a more advanced technique to exploit the VPU slices is to use the computational units of a same slice as a pipeline stages where the same micro-operation instruction processes different operations from multiple iterations at the same time. For instance, if an iteration inside a kernel includes two operations where operation B inputs uses operation A output, the two operations can be pipelined thanks to the computational units that can run in parallel inside the same slice. In fact, the micro-operation register can be configured in a way that it executes in parallel operation A with iteration i inputs and operation B with iteration $i-1$ inputs.

B. Data mapping to many-core memory hierarchy

In this section, we first describe how a streaming application can be efficiently mapped in the memory hierarchy that we are proposing. Then, we explain how modern streaming mining applications that adapt on the fly to the type of the input stream can take advantage of the proposed memory hierarchy.

We model each edge in a task-graph of an application with two buffers. The first buffer is stored in the private memory of

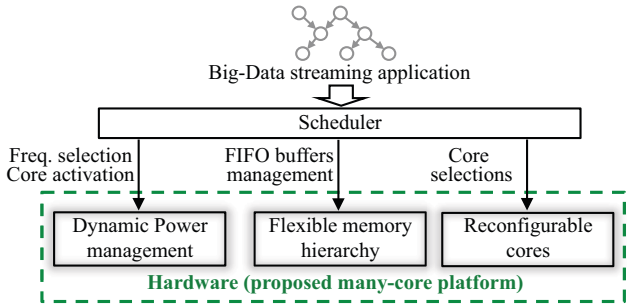

Fig. 3. Exploiting the proposed architecture for Big-Data stream application through a many-core energy-efficient scheduler

the core that is generating the data while the second buffer is stored in the private memory of the core that reads the data. The data are transferred from first buffer to the second one using DMA. Semaphores are used for synchronization. In streaming applications, we can identify two typical cases where a task may have either multiple outgoing edges (e.g., Fig. 4 (a)) or multiple incoming edges (e.g., Fig. 4 (b)). In the first case, the producer writes in the buffer stored in its private memory and the connected tasks pull the data to their private FIFO buffers using DMA. This is illustrated in Fig. 4 (c), where one buffer is used to feed multiple buffers. Unlike the Kahn Process Network (KPN) model [14] which allows only one consumer and one producer per FIFO, our model allows multiple consumers for the same buffer. Then, in the second case, the multiple ingoing edges are translated to multiple buffers stored in the same private memory unit of the concerned core. We illustrate this case with an example in Fig. 4 (d).

If all the required FIFO buffers are fixed and can fit in the available private memories, the shared memory is then not needed and can be switched off to save energy. However, depending on the input stream, some buffers are more used then the others. The buffers differ then by how frequent they are accessed by producers/consumers and the minimum required size. Moreover, due to the dynamic data-driven nature of stream mining applications and the highly variable stream input, these two buffer characteristics vary continuously at run time. Thus, using only private memories may not give enough flexibility to the programmer to handle all these dynamics. The

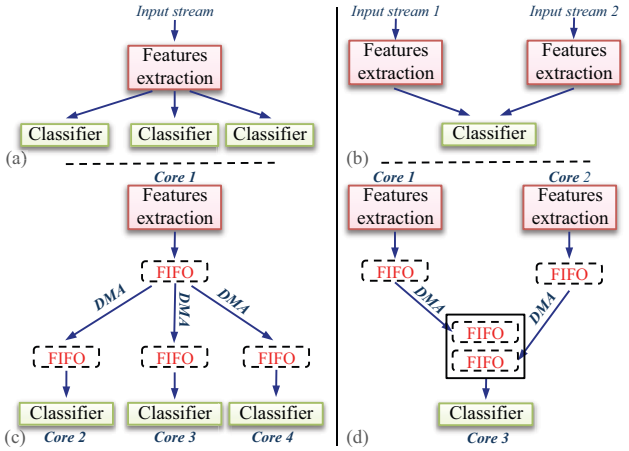

Fig. 4. Example of FIFO buffers mapping: (a, c) tasks with multiple outgoing edges. (b, d) Tasks with multiple ingoing edges

shared memory can be used as a FIFO buffers repository to all the cores and extends the space available in the private memory. The shared memory can be also used to minimize the number of DMA access to the producer buffer where only one DMA transfer is needed to copy the generated data from the private to the shared memory and the consumers will then fill their FIFO buffers by accessing the shared memory.

An efficient online mapping of the FIFO buffers to the different memory banks is then very important. For instance, a highly used buffer can be either kept in the private memory or to a dedicated memory bank in the shared memory, while buffers that are used less frequently can be populated in the same memory bank.

C. Power management scheme

As explained in previous sections, the proposed architecture offers several features that allow the platform to scale to the requirement of the mapped streaming application with the minimum energy consumption. An energy-efficient scheduler can efficiently exploit these control knobs when mapping an application to the proposed platform. Therefore, in the following we model our platform and the targeted application. Then, we formulate the problem of energy-efficient scheduling of streaming applications for the proposed many-core platform. Fig. 3 illustrates the different features that can be efficiently exploited by a scheduler to map the application tasks. Numerous related works on scheduling streaming applications on many-core platform have been proposed in the literature. However, the development of energy-efficient scheduler optimized for our platform is out of the scope of this paper and a solution for this problem is left for a future work.

Our proposed many-core platform is composed of C clusters. Each cluster includes M cores, M private memories with size s_{pm}, a shared multi-bank memory with size $s_{sh} = N * s_b$ with N the number of banks and s_b the size of the bank. For the memory related to stream reference and local data, a delay of d_c is taken into account to fill the cache when a new task is mapped to the core. The major sources of power dissipation from each core can be broken down into dynamic power P_{dyn} and leakage power P_{leak}. The dynamic power consumption can be controlled by the selected frequency and the supply voltage (e.g., using DVFS) while the leakage power can be minimized with power gating techniques (e.g., using DPM). Each cluster can operate at a different frequency $f_i \in F$, where F denotes the set of available operating frequencies and $f_i < f_{i+1}$. Finally, each core has 2 different modes namely, *active* and *sleep* modes. In the active mode, the core runs normally (i.e., full leakage power consumption) while in the sleep mode the core is power gated (i.e., inactive with reduced leakage power consumption). Each time a core is switched to sleep mode, it requires X_{switch} clock cycles to wake up and switch to active mode. We illustrate the main features provided by our platform model in the lower part of Fig. 3.

We model Big-Data streaming application as a DAG $G =< \mathcal{N}, \mathcal{E} >$ of dependent tasks t_j with non-deterministic workload w_j and soft deadlines. $\mathcal{N}$ is the node set containing all the tasks. $\mathcal{E}$ is the edge set, which models the dependencies among the tasks. Each node in the DAG denotes a task t_j. e_j^k denotes that there is a directed edge from t_j to t_k indicating that task k depends on task j. Each task can be seen as a pipeline stage that extracts valuable information from the stream input in real time. A stream processing task is open-ended as it can continue to execute as long as input stream data are available. However, In modern stream mining applications, the topology of the graph is determined online based on the variation of the type of the input stream. Moreover, these applications implement different operating points on its classifiers to control which part of data stream to process from the full available stream. This will have impact on the size of the FIFO buffers which need then to be adapted on the fly with respect to the online selected operating points.

Given the proposed platform and application model, the goal of an energy-efficient scheduler is to efficiently map the tasks to the available M cores and N clusters with their corresponding DVFS values and to switch unused cores to sleep mode to save energy consumption. Moreover, the scheduler is responsible for determining the right FIFO buffer size of each edge in the task graph with respect to the application requirement and the available private memory size s_{pm} and shared memory size s_{sm}. In case the private memory is not sufficient, a scheduler should then efficiently partition the shared memory space among the required FIFO buffers and minimize the number of active memory banks to reduce the energy consumption. The scheduler is also responsible for continuously adapting the FIFO buffers repartition in the memory with respect to the stream input variation and the decision made at the application level (e.g., load shedding and classifiers operating points). Finally, a modification of the task graph at run time requires a new mapping of the tasks to the cores. Available scheduler actions are illustrated in Fig. 3 with the arrows from the scheduler to the hardware layer

V. Experimental results

We demonstrate the advantages of our proposed low power many-core platform for stream computing applications on a set of experimental benchmarks related to stream mining and machine learning applications. In our experiments, we mainly focus on the memory hierarchy responsible for passing the data. The exploitation of the reconfigurability feature of the icyflex4 processor is out of the scope of this paper.

To simulate the proposed memory hierarchy, we use the VirtualSOC simulator [15], which is a complete SystemC simulation environment for MPSoC architectural design and exploration. VirtualSOC provides cycle-accurate and bus signal-accurate simulation of many-core platform. The default configuration of VirtualSOC simulates the P2012 platform architecture [16] where a multi-bank data memory is shared between all the cores in a cluster. We have first modified VirtualSOC to include a memory hierarchy with two data ports as the icyflex4 architecture requires. Then, we modified the simulated memory hierarchy in order to simulate four types of configurations: (a) All data and buffers are stored in the same shared memory, (b) the FIFO buffers are stored in a shared memory separately from the input stream and local data, (c) interleaved memory with 16 banks and (d) our proposed memory hierarchy where the FIFO buffer content is pulled to each of the private data memory and the stream input is stored in a second private data memory with the local data.

For the application layer, we simulate the case of Fig. 4(a)(c). This type of task-graph topology is typical in many

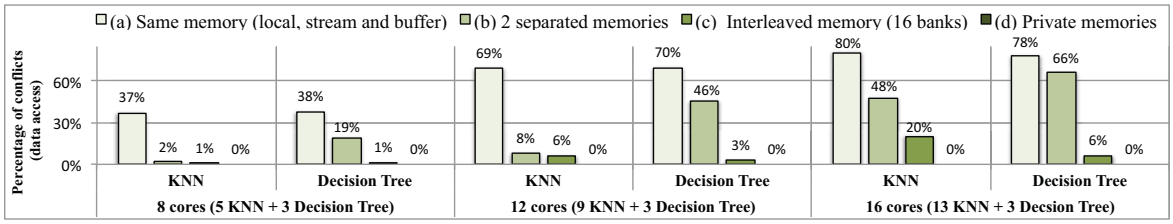

Fig. 5. Percentage of data memory access conflicts per core with respect to the number of cores, the deployed memory hierarchy and the mapped application

machine learning algorithms. For instance, in ensemble learning [18], a strong classifier is build based on a combination of several classifiers through a weighted voting procedure. In Adaboost [17], multiple weak learners are applied on the same set of features. They are weak learners in the sense that a single one will not do the job. In our benchmark, we use bagged K-Nearest Neighbour and bagged decision tree classifiers for hand written digit recognition. Bagged classifiers technique is a bootstrap ensemble method where each classifier is trained on a random subset of the training data and the final decision is made with majority voting. We use the MNIST database [19] of images of handwritten digits. The memory access pattern of KNN and decision tree are completely different. In fact, KNN accesses more frequently both data memory compared to the decision tree algorithm which accesses more its local data where the tree is stored. In KNN, all the features (i.e., pixels) of an image are used, while in the decision tree only a small subset is selected by its nodes to classify the image.

In all our experiments, we allocate 3 cores for 3 different decision tree classifiers and we allocate the remaining cores to KNN classifiers in order to simulate Big-Data application. Fig. 5 illustrates the results obtained with respect to the number of cores, the type of the executed classifier and the deployed memory hierarchy. The results show that for a low number of cores (i.e., standard multi-core platform with up to 8 cores), separating the FIFO buffers from other data input is already enough to drop the data memory access conflicts from 37% to 2%. However, for the case of many-core platforms (e.g., 12 or 16 cores), separating both memories is not enough to avoid access conflicts. For instance, in the case of 16 cores, the access conflicts can go up to 48% for KNN and 66% for the decision tree. By using interleaved memory with 16 banks, the memory access conflicts is significantly reduced. However it still has a high percentage for platform with 16 cores (up to 20% for cores running a KNN classifier). Thus, the use of private memories as proposed by our memory hierarchy is then mandatory for many-core platform running Big-Data streaming applications. Moreover, by separating the FIFO buffer from the local data memory, the DMA transfers can be scheduled while the application is processing its local data. For instance in KNN, the DMA can be activated to transfer data buffer while the application is sorting the K nearest neighbours which are stored in its local data. Thus, the DMA latency can be hidden.

VI. Conclusion

In this paper, we have proposed a novel low-power many-core architecture for Big-Data stream computing. The key contributions of this work were as follows: (i) Our solution integrates low-power, scalable and reconfigurable cores that can adapt to the characteristic of each classifier. (ii) Memory hierarchy optimized for Big-Data stream and able to adapt to the dynamic data-driven nature of modern stream mining applications. Our results for machine learning algorithms have demonstrated that our solution allows to significantly reduce the data memory access conflicts even for a high number of cores in the cluster.

References

[1] U. J. Kapasi, *et al.*, "The imagine stream processor," in *ICCD*, 2002.

[2] W. J. Dally, *et al.*, "Merrimac: Supercomputing with Streams, in *SC*, 2003.

[3] C. Kyrkou, "Stream Processors and GPUs: Architectures for High Performance Computing," Survey on Stream Processor and Graphics Processing Units.

[4] SPI, "Stream Processing: Enabling the new generation of easy to use, high-performance DSPs, White Paper, June 2008.

[5] J.-L. Nagel, *et al.*,"The icyflex4 Processor, a Scalable DSP Architecture, CSEM Scientific and Technical Report 2009

[6] Y. Matsumoto, *et al.*, "Manycore processor for video mining applications," in Proc. *ASP-DAC*, 2013.

[7] A. Majumdar, *et al.*, "An Energy-Efficient Heterogeneous System for Embedded Learning and Classification," in *IEEE ESL*, vol. 3, no. 1, pp. 42–45, March 2011.

[8] J. Xu, *et al.*, "Learning optimal classifier chains for real-time Big-Data mining," in Proc. *Annual Allerton Conference*, 2013.

[9] M. Marzolla, "Letters: Fast training of support vector machines on the Cell processor," Neurocomputing vol.74, no. 17, pp. 3700–3707, October 2011.

[10] M. Diao, *et al.*, "Large-Scale Semantic Concept Detection on Manycore Platforms for Multimedia Mining," in Proc. *IPDPS*, 2011.

[11] C. Kyrkou, *et al.*, "A Parallel Hardware Architecture for Real-Time Object Detection with Support Vector Machines," in *IEEE TC*, vol. 61, no. 6, pp. 831–842, June 2012.

[12] Z. Hou, *et al.*, "Real-Time Very Large-Scale Integration Recognition System with an On-Chip Adaptive K-Means Learning Algorithm, Jpn. J. Appl. Phys., vol. 52, 04CE11, Apr. 2013.

[13] J. A. Kahle, *et al.*, "Introduction to the cell multiprocessor," in *IBM J. Res. Dev.*, vol. 49, no. 4/5, pp. 589–604, July 2005.

[14] D. Nadezhkin, *et al.*, "Realizing FIFO Communication When Mapping Kahn Process Networks onto the Cell," in Proc. *SAMOS* 2009.

[15] D. Bortolotti, *et al.*, "VirtualSoC: A Full-System Simulation Environment for Massively Parallel Heterogeneous System-on-Chip," in Proc. *IPDPS* 2013.

[16] D. Melpignano, *et al.*, "Platform 2012, a many-core computing accelerator for embedded SoCs: Performance evaluation of visual analytics applications," in Proc. *DAC*, 2012.

[17] Y. Freund, *et al.*, "A decision-theoretic generalization of on-line learning and an application to boosting," J. Comput. Syst. Sci., vol. 55, no. 1, pp. 119–139, August 1997.

[18] D. Opitz, *et al.*, "Popular Ensemble Methods: An Empirical Study," in *JAIR*, vol. 11, pp. 169–198, 1999.

[19] Y. LeCun, *et al.*, "The MNIST database of handwritten digits," http://yann.lecun.com/exdb/mnist/.

[20] R. Ducasse, *et al.*, "Adaptive Topologic Optimization for Large-Scale Stream Mining" in *IEEE JSTSP*, vol. 4, no. 3, pp. 620–636, June 2010.

[21] S. Kumar, *et al.*, "A network on chip architecture and design methodology," in Proc. *ISVLSI*, 2002.

[22] J. Sun, *et al.*, " Big data analytics for healthcare," in Proc. *SIGKDD*, 2013. Slides: http://dmkd.cs.wayne.edu/TUTORIAL/Healthcare/