NanoStreams: A Microserver Architecture for Real-time Analytics on Fast Data Streams

Minhas, U. I., Russell, M., Kaloutsakis, S., Barber, P., Woods, R., Georgakoudis, G., ... Bilos, A. (2017). NanoStreams: A Microserver Architecture for Real-time Analytics on Fast Data Streams. IEEE Transactions on Multi-Scale Computing Systems. DOI: 10.1109/TMSCS.2017.2764087

Published in:
IEEE Transactions on Multi-Scale Computing Systems

Document Version:
Peer reviewed version

NanoStreams: A Microserver Architecture for Real-time Analytics on Fast Data Streams

U. I. Minhas, M. Russell, S. Kaloutsakis, P. Barber, R. Woods, *Senior Member, IEEE* G. Georgakoudis, C. Gillan, D. S. Nikolopoulos *Senior Member, IEEE* and A. Bilas

Abstract—Ever increasing power consumption has created great interest in energy-efficient microserver architectures but they lack the computational, networking and storage power necessary to cope with real-time data analytics. We propose NanoStreams, an integrated architecture comprising an ARM-based microserver, coupled via a novel, low latency network interface, Nanowire, to a Analytics-on-Chip architecture implemented on Field Programmable Gate Array (FPGA) technology; the architecture comprises ARM cores for performing low latency transactional processing, integrated with programmable, energy efficient Nanocore processors for high-throughput streaming analytics. The paper outlines the complete system architecture, hardware level detail, compiler, network protocol, and programming environment. We present experiments with an industrial workload from the financial services sector, comparing a state-of-the-art server based on Intel Sandy Bridge processors, an ARM based Calxeda ECS-1000 microserver and ODROID XU3 node, with the NanoStreams microserver architecture. For end-to-end workload, the NanoStreams microserver achieves energy savings up to 10.7×, 5.87× and 5× compared to the Intel server, Calxeda microserver and ODROID node respectively.

Index Terms—Reconfigurable computing, transactional Analytics, microserver, FPGAs

1 Introduction

With ever increasing data volume and computation complexity, future energy consumption of data warehouses will be a key challenge. Van Heddeghem et al [1] predicts that data centres will consume about 2% of global electricity consumption. Although the key motivating factors of server design had been speed and availability, energy consumption can no longer be ignored as it represents a $B economy burden and increased threat to global climate [2].

With interest in real-time and large-scale data analytics growing, servers are under increased pressure to deliver higher performance and data processing throughput under a shrinking power budget. Whilst microservers based on the ARM architecture have been proposed as an alternative to Intel servers to tackle this problem, they have shown high energy-efficiency only for light weight transactional tasks but limited computational power for analytical tasks [3] [4].

Earlier work has explored the integration of ARM cores with wide SIMD units for higher energy efficiency [5]. However, this work focused on high performance computing (HPC) workloads which like modern real-time analytics, have high throughput needs but do not have transactional components with latency constraints. In real-time analytics, servers must simultaneously execute both low-latency transactional components that decode and store data from streaming messages arriving at a high speed and high-throughput analytical components that query the logged data to extract knowledge.

Field Programmable Gate Arrays (FPGAs) are seen as an attractive proposition to address energy issues and are used in the IvyTown Xeon + Stratix V FPGA system [6] and Amazon's Elastic Compute Cloud (EC2) F1 instances. Issues include the need for specialist programming languages and the lengthy synthesis times. Vendor optimized soft cores, e.g. Microblaze, tackle this by synthesizing statically programmable processors for which new code can be compiled and linked. However, this reprogramming needs to be carried out using the vendor tools which limits its use in *just-in-time* reprogramming of a many core co-processing accelerator which is accessible remotely to microservers.

Another challenge is the need to overcome the traditional network stack overhead resulting in increased latency and CPU processing workload incurred when connecting the accelerator to the microserver. We require the system to share accelerators to allow them to be decoupled from the host/server technology cycle, as accelerators evolve at a different pace. Ideally, this should use Ethernet to allow use in existing Ethernet infrastructure in data centres.

In this paper, we present a power-efficient, microserver architecture, called Nanostreams that addresses the challenges of building computing systems that can efficiently support heavyweight computation on hybrid transactional-analytical workloads. It effectively exploits FPGA technology through the creation of a highly programmable and dynamically reconfigurable Analytics-on-Chip (AoC) architecture based on the *Nanocore* processor. It is highly programmable and easily repurposed to execute different analytical tasks in a just-in-time manner with low energy consumption. It is programmed using a streaming data flow

- *U. I. Minhas, R. Woods, G. Georgakoudis, C. Gillan and D. S. Nikolopoulos are with the School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, Belfast, UK*
E-mail: u.minhas, r.woods, g.georgakoudis, c.gillan, d.nikolopoulos@qub.ac.uk
M. Russell, P. Barber and R. Woods are with Analytics Engines Ltd., 1 Chlorine Gardens, Belfast, UK
E-mail: m.russell, p.barber, r.woods@analyticsengines.com
S. Kaloutsakis and A. Bilas are based in FORTH, Institute of Computer Science, Heraklion, Crete, Greece
E-mail: kaloutsa, bilas@ics.forth.gr

Manuscript received December 1, 2016; revised ...

model, and allows the cores to act as parallel computing black boxes where the functionality for each core is compiled using a lightweight instruction set.

We also present the *Nanowire*, a novel high-speed network interface which enables low latency and efficient sharing of multiple AoCs between tasks for easy scalability. It attaches the AOC to microservers via raw Ethernet, thus enabling sharing of accelerators across microservers which can lead to improved overall cost with mid- and high-end accelerators. Use of Ethernet decouples the accelerator from server technology cycle, allowing each the server and accelerator to evolve at a different pace.

We have prototyped our AoC architecture on the Xilinx Zynq-7000 SoC which offers an ARM processor together with hardware logic. On-chip integration of the ARM and Nanocores reduces the communication latency and enables efficient on-chip parallelization of analytics kernels.

The key contributions can be summarized as:

- A novel microserver architecture for real-time analytics, based on a new highly programmable and dynamically reconfigurable many-core accelerator that takes advantage of the lower power of FPGAs but which overcomes long programming times.
- A streaming data flow model which provides a complete, domain-specific programming environment for data analytics using streaming input/output functions at hardware level and a supportive, lightweight instruction set for FPGAs comprising only 26 instructions.
- A new network interface for low-latency multi-tasking and sharing of accelerators, which provides a simple and convenient software Application Programming Interface (API) to virtualise and manage accelerators, while minimizing host CPU overheads.
- A prototype of the microserver giving up to 10.7×, 5.87× and 5× better energy efficiency compared to an Intel server, Calxeda microserver and ODROID node respectively for a financial services workload.

The paper is organized as follows. We survey related work on low power servers in Section 2. We specify the Nanocore architecture in Section 3 and provide more detailed information on the Nanocore memory system, dataflow, frequency and power consumption in Section 4. Section 5 gives details of the AoC server and focuses on communication and the system interconnect. We present our experimental setup using a financial services workload in Section 6 and our experimental analysis in Section 7. We conclude the paper in Section 8.

2 Background Work

The performance and power consumption of alternative approaches to traditional servers in data centres and HPC such as those based on ARM processors compare well with Intel x86 servers in terms of energy and cost [7] [8], but it is acknowledged that they need to evolve for HPC [5]. EuroServer [9] [10] is a scale-out architecture with support for resource mutualization and sharing that uses integrated interconnects and 3D silicon integration technology to efficiently share resources. We take a different approach to improve the energy-efficiency of servers using FPGA-based reconfigurable accelerators [11] and new methods to program and share these accelerators.

Examples of FPGA-servers include IBM's POWER8 Coherent Accelerator Processor Interface port which allows coherent memory access for FPGA based accelerators via the Peripheral Component Interconnect Express (PCIe). A 5× gain in energy efficiency has been demonstrated for FPGA-based, FFT acceleration when compared to an optimized parallel software FFT running on a 12-core CPU [12]. Microsoft's FPGA-based Catapult data center infrastructure demonstrated a 95% improvement in ranking throughput in a production search infrastructure at comparable latency to a software only, with only a 10% increase in power consumption [13]. Their focus, however, has been on reducing system design constraints with logic customised for each application, needing hardware design expertise.

Reconfigurable many core architectures for data analytics have been proposed, including the use of FPGAs for data-intensive applications via the strong coupling between storage and FPGA [14] and switching of hardware and software threads for a heterogeneous system involving a CPU and reconfigurable computing units [15]. It focused on the run-time management of tasks but did not provide insight on the architectural challenges of processing elements. Work in [16] uses operation units placed in a $n \times n$ layout and configured for different queries using a compiler; it can reduce the effort to deploy analytical workloads on FPGAs but the functionality achievable by operation units is limited, since there is no memory hierarchy and limited options for implementing a data flow.

Various approaches are being applied to enhance FPGA programmability, including vendor high level synthesis tools [17] and soft/hard core vector/scalar processors [18] [19]. We use soft core scalar processors for their ease of scalability and configurability for a range of applications, although commercial offering exist including Xilinx's MicroBlaze [20], Altera's Nios II [21] and ARM Cortex-M1 [22]. However, these are heavyweight general purpose processors which support an extensive instruction set, which is unnecessary for domain specific purposes. Indeed, we provide more detailed insights with Microblaze and ARM Cortex-M1 against the proposed core here in the later sections.

A Dynamic Streaming Engine uses programmable input/output buffers and computational lanes which can be configured at run-time using micro-programming and configuring registers [23]. Streaming elements is an FPGA accelerator for signal and image processing [24] that focuses on pre-synthesis configurability and a tight application specific design, thus ensuring minimum resource usage at the cost of reduced flexibility post-synthesis. Authors in [25] present a programmable core for relational operation of text analytics leveraging stream processing model that uses shared memory and virtual streams for array processing.

We aim to go from high level stream programming languages to FPGA hardware while maintaining programmability and flexibility. Using intelligent workload distribution and a simple, low-power and scalable core coupled to an ARM processor, we achieve lower latency and higher transactional throughput for streaming applications. Developers need limited effort to migrate existing applications to our platform because of the abstraction of parallelism, easy-to-use core interfaces and run-time compilation support.

To integrate AOC with microserver, we propose Nanowire, an Ethernet-based protocol which reduces overheads when communicating within the rack. Similar protocols have emerged for communication between servers and storage systems [26] as well as for general purpose communication within the datacenter [27]. Tyche [26] reduces protocol processing overheads over raw Ethernet, but unlike Nanowire, targets in-kernel communication protocols that do not require user/kernel communication. IX [27] is an Ethernet-based protocol that supports TCPcommunication and looks to improve VM to VM comms. within the datacenter, whereas Nanowire is aimed at host accelerator comms.

Our experimental analysis uses a real-time option pricing workload with a realistic, industrial strength setup. We use an option pricing approach [28] that is more accurate than other models such as Monte Carlo and Finite Difference, whilst remaining fast. Prior energy efficient, FPGA-based implementations include: a customized hardware for binomial option pricing which was able to achieve 250× and 2× improvement over a Core2 Duo processor and an nVidia Geforce 7900GTX processor, respectively [29]; an OpenCL-based software based solution which was able to process 2000 options/s at an average of 20W [30]; a hybrid CPU/FPGA based implementations of the Longstaff-Schwartz algorithm [31] used dynamic reconfiguration to achieve up to 16× and 268× improvement in run-time and energy respectively compared to CPU only implementation. However, these approaches employ classical FPGA design approaches and are very time consuming for multiple use cases.

The work makes use of commercially available general purpose hardware rather than specialised system architectures developed by vendors for their data centres [12] [13] and can be incorporated easily into existing small to medium scale data centres and near the edge computing nodes without any specific interface requirements. To verify validity, we have provided integrated system results on the lowest speed grade hardware using real-time data as compared to work providing simulation or theoretically estimated results or only logic power consumption numbers [31] [23] [24]. Furthermore the results are compared to state-of-the art servers and microservers and showing energy efficiency for an industrial real-time workload. Moreover, fine-grained and coarse-grained power profiling results are applied to give real-time energy consumption insight at core level (for scale-out projections) and actual power cost (for economic impact), respectively. This has been missing in existing work on soft core processors designs [19] [32] [18] [25].

3 AoC Architecture Specification

The AoC lies at the heart of this microserver architecture and is an amalgam of embedded general purpose RISC processor and many core processing unit based on application-domain specific Nanocores. Nanocores are a new class of programmable, reconfigurable data-driven accelerators for stream processing. The AoC architecture system specification is given in Fig. 1. The approach overcomes the issue of FPGA design time and look to increase analytical processing throughput by exposing the parallel computational capabilities of the underlying hardware to the software domain, allowing rapid, run-time reconfiguration and programmability.

The targeted methodology is stream processing i.e. processing of an ordered set of data organised at run-time without storing [33]. The input main stream is divided into multiple streams operated by multiple cores in parallel as the operations being performed on multiple streams is the same i.e. Single Instruction, Multiple Data (SIMD) type of processing. Otherwise, it may require different operations being executed on different data constituting Multiple Instruction, Multiple Data (MIMD) type processing.

The initial goal of the architecture design was to describe a single core which combined with a number of other Nanocores that may not necessarily have the same feature set, is sufficiently flexible to allow the acceleration of a wide range of streaming applications. The key aspect of the approach is that we have optimized the use of underlying FPGA hardware to allow the creation of a core which operates substantially faster than existing FPGA-based cores and comparable with High Level Synthesis (HLS) tools.

With the availability of a low latency ARM core on the same chip, portions of software can be distributed such that the complex and serial tasks are executed on RISC core while Nanocores are used for simple yet large computations by exploiting parallelism. The control flow lies with the ARM core which acts as a master utilizing Nanocores as required. However, due to heterogeneous nature of processing, both the ARM and Nanocores have independent access to resources such as memory etc., while still maintaining required synchronisation and program flow.

To achieve maximum parallelism, cores should be lightweight with a simple instruction set, thus allowing large number of cores to be replicated in the reconfigurable logic. Furthermore the streaming data flow needs cores to act as the parallel computing black boxes oblivious to the rest of flow. This requires description of stream input/output functions at hardware level and a supportive instruction set.

Due to the streaming nature of data, operations normally have a relatively higher data read/write requirements to/from stream buffers, registers, local memory and off-chip memory; hence the architecture should allow parallel reads from various sources to reduce time for memory operations. The requirements, such as those in financial applications, demand a high precision of computation as well. Although 64-bit precision capabilities are increasingly being provided in GPU and CPU architectures, it is still considered an expensive design choice for FPGAs.

Furthermore, the Nanocore architecture should support easy integration for multi-core design and so this work targets SIMD and MIMD type approaches. This simplifies abstraction of parallelism at a higher level. This is also in line with most parallel programming paradigms and suits the traditional decomposition of most streaming compute intensive applications. This also allows for efficient use of the hardware resources and optimum scaling for larger devices or different configurations by avoiding the complex networks for communication between cores.

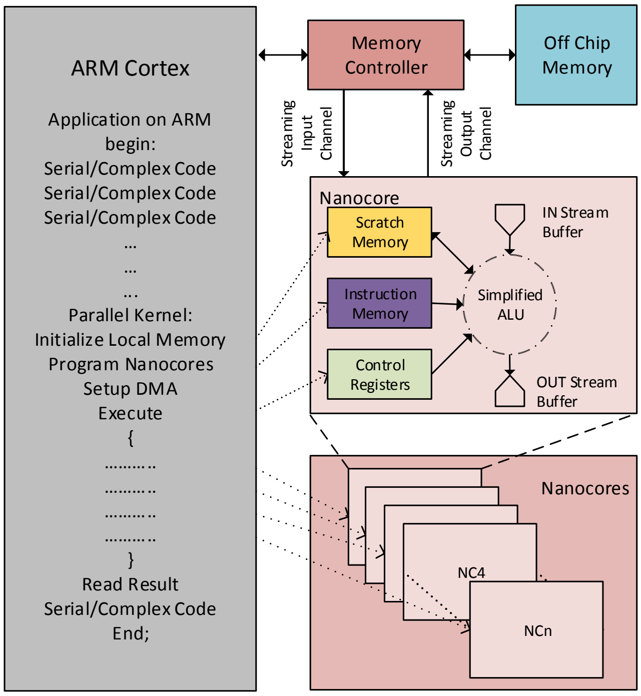

Fig. 1. AoC Architecture Design

Finally, the design of core accelerator hardware should be such that the AoC architecture fits in nicely in a high-level streaming data flow model, leading to a complete, domain-specific programming environment for data analytics. Although reconfigurable computing has always been appreciated for energy efficiency, the programming and subsequent integration in the programming flow, is not considered convenient compared to other platforms (CPUs, GPUs, etc). The usual approach requires designing in specialist hardware descriptive language (HDL or Verilog) which needs time and hardware design expertise and even after that, synthesis, place and route and verification is still required.

For our application, we target the use of the AoC as shared accelerator among multiple microservers in a cluster. This means the Nanocores should not only be easily programmable, but any microserver should be able to program as per its unique requirements, at run-time over Ethernet. This feature not only provides flexibility of use at run-time, it also abstracts the cores for higher level programming and helps with exploitation of parallelism. At the core level, the hardware should support for reprogramming. At system architecture level, supportive libraries should provide an application interface from the microserver to the ARM core on the AoC architecture and from the ARM to Nanocores.

4 Nanocore Architecture

The Nanocore architecture given in Fig. 2, supports 32-bit and 64-bit fixed point arithmetic; this configuration was selected to demonstrate a key precision benefit over existing 32-bit only FPGA cores, e.g. Microblaze, Nios II and utilises the fixed point DSP capability of the current FPGAs. To this end, the work has focused on architecture optimization irrespective of underlying arithmetic precision support. This is because, whilst the arithmetic operations possible within the core will differ depending upon its build-time configuration, the underlying design philosophy of the core is expected to remain consistent. Program control flow and use of registers are to be consistent across all configurations.

4.1 Memory Architecture

There are 16 registers within each processor (R0-R15) and as the only path to processor, all operands need to be stored in them first. Furthermore, each core has a read-write, addressable memory for storing intermediate calculation results and use as a stack i.e.'scratch' memory. This ensures that each core is capable of relatively complex behaviour and enables the core to support operations where data has to be spilled out of the internal registers. Scratch memory also helps to hide memory latency with ARM controlled data transfers [32]. The data can also be fed via input and output stream memories as well as constant load into registers.

4.2 Data Flow

In order to minimise latency, the input and output memories are configured as first-in, first-out buffers (FIFOs). Nanocores are designed as data-driven processors making use of special blocking read/write instructions which suspend operation of the core when reading from an empty FIFO or writing to a full FIFO. This is a key feature which greatly simplifies control of data flow for streaming data.

In order to maximise the run-time configurability of Nanocore, some additional elements are necessary. As the cores have no visibility of system data flow, additional elements are needed to handle the following functions:

- input of data stream from external memory or interfaces;
- output of data stream to external memory or interfaces;
- demultiplexing of single stream into multiple streams;
- multiplexing of data from multiple sources into a single stream;
- routing of data between Nanocores and flow units, including stream replication.

These units are controlled by the master and configured at run-time, the same time as the Nanocores are programmed. Reconfiguration of the data flow during processing session is not provided.

4.3 Instruction Set

The instruction set has been kept very small, 26 instructions, in order to minimise the resource utilisation of the processor when configuring cores for different applications, but still provides enough functionality to be able to support standard software program flow constructs. It is summarized in TABLE 1 with the number of execution cycles required for each instruction. The default instruction set is Turing complete and additional application specific instructions can be added to improve performance. Special instructions are included for input read and output write.

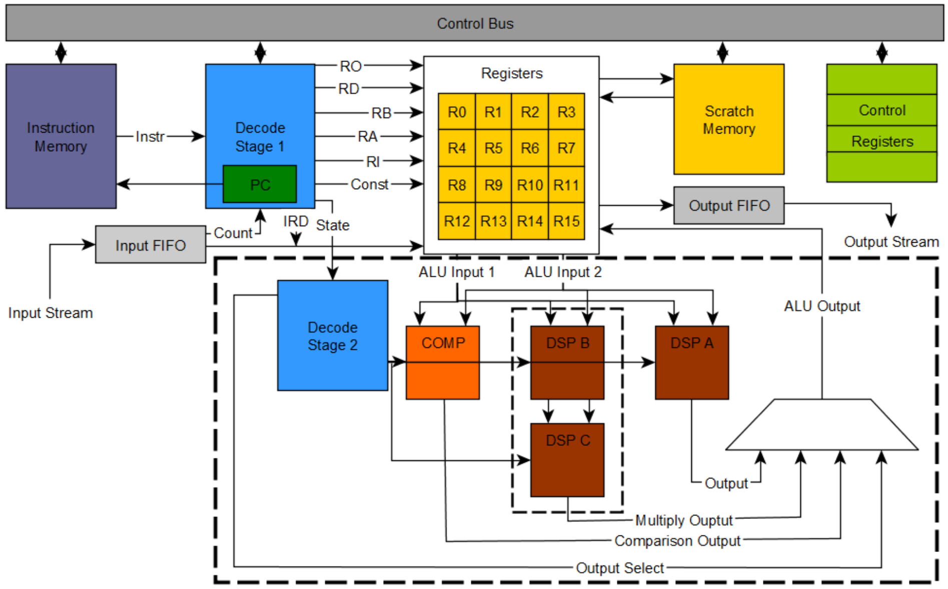

Fig. 2. Nanocore Block Diagram.

TABLE 1
Instruction Set with delay cycles for 32 bit configuration

Instruction	Delay
No Operation, Unconditional Jump, Load Constant	1
Input Write, Output Write	4, 2
Memory Write, Memory Read	6, 3
Jump (If Equal, If Not Equal)	3
Shift (Left, Right, Arithmetic Right)	5
Compare/Signed Compare (Greater Than, Less Than)	5
Bitwise Operator (Invert, OR, AND, XOR)	5
Arithmetic (ADD, SUB)	5
Multiply	8
Multiply High (64-bit for 32-bit architecture)	9

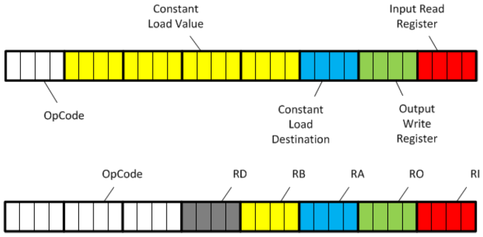

Fig. 3. Instruction word structure with support for 4 operations in one cycle

We have defined a 32-bit instruction word as shown in Fig. 3; whilst the Zynq block RAM allows us to have 36-bit instruction words, four bits are kept free for future or processor internal use. The instruction word has been designed to allow the core to execute four operations within a single clock cycle: an input read to a register, and output write from a register, an always jump and either a constant load or another instruction. The ability to perform multiple operations within a single instruction should minimize latency for high density memory operations. This should help to reduce number of instructions/delays required for a tight loop and helps with high throughput for stream processing.

4.4 Resource Utilization and Scalability

The Nanocore code base has been built up as a result of considerable design engineering work. Explorations primarily surrounded the application and design requirements (control and flexibility), and focused on memory utilization. The amount of logic that can be fit into a device is limited by the resources available on the FPGA such as DSP48s, BRAMs, LUTs, registers and routing. The explorations were also based around balancing the resource usage to allow the maximum number of cores to be accommodated on any device. Starting with DSP48s, our design requires 3 and 8 DSP48s to implement a 32-bit and 64-bit ALUs respectively.

As for the memory, each BRAM in Zynq devices can store up to 36 kbits. Each Nanocore essentially has three memory areas: code, input data and scratch memory. For the purpose of reprogramming and reading/writing data from Nanocore, access of code and scratch memory from outside nancores respectively, is needed. The Nanocores operate independently on a single clock, with other components (memory controllers etc.) being on different clock domains. This imposes a constraint on the code and scratch memory, in that they must be true dual-port memories, with read and write clocks. The input data memory only requires a single clock. Both configurations are possible using the Zynq BRAM. We have implemented the code and scratch memories as dual-port 1024 and 512 elements 36-bit memory, utilizing $1\frac{1}{2}$ BRAMs respectively. Similarly a $\frac{1}{2}$ BRAM has been used for FIFO memory. Scratch and FIFO memory sizes double for 64 bit version.

LUTs and registers are required for other functionalities such as providing data paths within the core, etc. TABLE 2 summarizes the resource usage for one Nanocore and its percentage utilization against the total number of resources available on ZC7020 chip of Zynq SoC family (the same chip has been used for testing the system)

With the utilization information available for resources for a single core, we are now able to place an upper

TABLE 2
Resource Utilization by Single Core

	Nanocore 32-bit		Nanocore 64-bit		Microblaze 32-bit	
Resource	Usage	%	Usage	%	Usage	%
LUTs	1773	3.3%	3144	5.9%	1105	2.1%
Flip-flops	1426	1.3%	2725	2.6%	786	0.7%
DSPs	3	1.4%	8	3.6%	3	1.4%
BRAMs	2.5	1.8%	4	2.8%	2.5	1.8%

TABLE 3
Contribution of each resource towards limiting cores per device

Zync	**FF**		**LUT**		**BRAM**		**DSP**	
Device	32-b	64-b	32-b	64-b	32-b	64-b	32-b	64-b
Z010	24	12	**9**	**5**	24	15	26	10
Z015	64	33	**26**	**14**	38	23	53	20
Z020	74	39	**30**	**16**	56	35	73	27
Z030	110	57	**44**	**25**	106	66	133	50
Z045	306	160	**123**	**69**	218	136	300	112
Z100	389	203	**156**	**88**	302	188	673	252

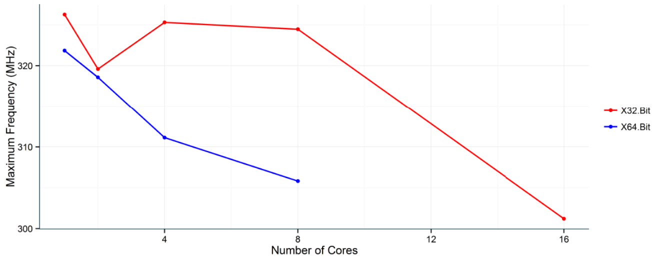

Fig. 4. Maximum frequency with number of cores

bound on the number of Nanocores per device. By regularly performing scalability experiments with chains of cores, we closely monitored and enabled the minimization of the effect of routing on the scalability of the design. Extrapolating the resources usage of a single Nanocore and considering the maximum resources available on each device, TABLE 3 shows the maximum number of cores that can be fit into different Zynq devices. This directly translates to scalability of the design which we discuss more in Section 6.3. The minimum bound for each device configuration is highlighted in bold. This analysis shows that the cost of adding 64-bit support is much smaller in terms of BRAM than DSP blocks.

4.5 Frequency

The current maximum frequencies of the core against number of cores for 32- and 64-bit versions are shown in Fig.4, obtained using Vivado 2014.3.1's implementation defaults with a constraint of 312.5 MHz and a FPGA device with lowest speed grade. Since most of these results exceed this frequency, it is likely that optimisation stops once this frequency is met. This would explain the drop in frequency for the 32 bit Nanocore with 2 cores. Our synthesis resulted in a single Microblaze core with a frequency of 160 MHz. The reported figures for ARM Cortex-M1 is 200 MHz on Virtex-5 for the maximum speed grade [22].

4.6 Power Consumption

Energy efficiency of a single core serves as a main metric when investigating scalability trade-off but has not been investigated previously [19] [25]. The real-time dynamic power consumption for a single 32-bit and 64-bit Nanocore is given as 41 and 126mW respectively. This was measured using the three TI power controllers (UCD9248PFC) on the ZC702 development board and allowed us to distinguish between the programmable logic power from the ARM cores and other peripherals power. The power was profiled in terms of Watts/instruction by measuring average dynamic power consumption of 8 Nanocores running the same instructions in a loop for a significant period of time.

4.7 Initialization

The core is controlled by a master, which has the ability to start, stop and reset the core, as well as to write to the program memory. In the context of the Zynq board and the development of the main framework, the ARM core acts as master for all of the Nanocores within the FPGA fabric. A core is initialized with a fixed program, allowing it to start on boot, but the real feature of the core lies in its reprogrammability at run-time.

4.8 Multi-core Infrastructure

An important step in enabling the easy integration of the Nanocore with other IP is the adoption of the AXI protocol for communication. For the FIFO interfaces, this only requires replacing a full signal with a ready signal, which is an inverter. The control interface is more complex because AXI Lite supports simultaneous reads and writes to different addresses but the Nanocore only has one control address port. This issue has been resolved by adding control logic which blocks traffic on one port while a transfer is occurring on the other; write transactions are given priority in the event of a collision.

One major challenge in design of multicore infrastructure is the efficient scatter and gather operations at the input and output of the Nanocore array. Generally, applications will seek to run SIMD or MIMD type applications, with each Nanocore operating on a different burst of inputs and returning a burst of result data. This is facilitated by the scatter and gather modules which can be configured to split the incoming data stream for a variable number of Nanocores with different word sizes and then join the outputs. They can also be configured to pad input and output values if the data sizes are not a multiple of number of cores.

The cores also facilitate clock crossing, so the Nanocores can operate at a higher frequency than the input and output data streams. This means the processing of data happens at a faster frequency than data transfer and thus helps to reduce computation latency against streaming data.

5 System Architecture

The typical application development on FPGA includes not only the algorithm coding and data management, but also building and integrating system architecture responsible for host-to-FPGA communication. System integration needs additional time and expertise and often exceeds the original effort required for application development itself. This is achieved using a step wise bottom up approach from reconfigurable logic all the way up to C programming on host with an aim to abstract everything from the programmer.

5.1 Intra-Analytic on Chip Communication

The ARM controller on the Zynq device has the ability to directly address memory location for Nanocore's instruction RAM and scratch memory. This allows for the creation of libraries to assist the programmer to debug hardware-based implementation of an algorithm in a standalone mode, allowing validation of design in hardware which is much superior to purely software-based simulation.

The libraries run on the ARM core and stimulate a real Nanocore on the FPGA, allowing the programmer to access the Nanocore's data and control interfaces through wrapper functions. This library provides functions such as initialization, programming/reprogramming and control of transfer of data to, and from, the Nanocore/multicore architecture. Developer-level functions to aid debugging, such as scratch memory access, have also been provided. A series of library functions have been created, giving a high level of access by abstracting hardware interface around Nanocore. The Nanocore needs to be reset when it is being reprogrammed or reconfigured and can also be paused to allow access to other control operations.

5.2 Nanowire and Host OS

The interconnect between the AoC architecture and host microserver is provided through Nanowire, a lightweight, low-latency communication protocol. Besides efficiency, Nanowire aims to enable shared accelerators, e.g., at the node level or rack level for cost reasons, but also decouples accelerators from the host/server technology cycle, as accelerators evolve at a different pace compared to servers.

Nanowire is based on raw Ethernet to utilize existing Ethernet infrastructure in data centres and transparently coexist with other Ethernet-based protocols. The main goals of Nanowire are to:

- provide a simple and convenient API to virtualise and manage accelerators;
- sustain reliable, high-throughput and low-latency transfers;
- minimize host side CPU overheads while supporting high concurrency and;
- achieve μ s response and tight tail latency QoS.

In order to address and fulfil the aforementioned goals, Nanowire adheres to the following design principles:

- Application-kernel communication based on shared memory queues and helper cores; this eliminates data copies and system calls at kernel boundary.
- Dedicated kernel threads and accelerator queues (channels) which eliminates coherence traffic and synchronization
- Adaptive policies in issue and completion path that eliminates expensive sleep/wakeup operations.
- Separation of concerns, composed of two abstractions: The Host-Accelerator Transport (HAT) layer that handles networking aspects and the Task Issue Protocol (TIP), a task queue layer that issues task requests from the hosts and receive task results from the accelerators.

HAT provides the network tier of NanoStreams and offers a common abstraction of the network level services and I/O primitives to both the host and accelerator nodes. HAT code runs on both sides of the interconnect and although the host side makes use of the host OS services, on the accelerator side, HAT runs on more diverse platforms. In our prototype, the accelerator side of HAT is implemented as custom firmware directly on top of the Cortex-A9 core in the Processing System (PS) of the FPGA card.

HAT provides lightweight connection-less channels as the lowest-level communication. A channel is a point-to-point unidirectional queue of packet slots used for communication between a source host and destination accelerator node. Resources per channel, e.g., slot size, are chosen at creation time. Channels aim at providing a low-overhead and low-latency communication path, while allowing the system to tune the allocation of resources for each channel.

TIP provides the run-time system with the ability to transparently issue tasks to the accelerators without any knowledge of the underlying network infrastructure or the accelerators themselves. TIP implements a simple client-server protocol to decouple analytics kernel invocation from execution: it utilises HAT channels to enqueue kernel service requests to remote accelerator nodes and retrieve completions/replies. It supports both blocking and non-blocking interfaces. Further details about the network protocol and preliminary latency results are available in [34].

5.3 Compiler and Higher Level Language Support

The Nanostreams tool chain support is highlighted in Fig. 5. The Nanocore compiler and the higher level language run at the host. At the core level, a beta version of a C99-compliant compiler for Nanocore has been developed by ACE using the CoSy compiler development system. Along with compiling C using the default 26 instruction set supported by Nanocore, the compiler provides support for the Nanocore special instructions for high speed input read and output write. An assembler has also been developed for generation of machine code from manually optimized assembly code.

Furthermore, the aforementioned libraries for intra-AOC communication, have been abstracted via the Nanowire at the microserver to provide direct access to the high level programming tool chain. This abstracts the underlying FPGA hardware from the programmer and allows the AoC to be accessed in a software domain. For an efficient application development, the user can then tune the computation as per the underlying number of cores per AoC, the number of AoCs and the infrastructure in place. At the high level programming, data flow graphs within an application are defined in terms of annotated C functions, called kernels. The defined specification, NANOCOREDF, allows the focus to be laid on known characteristics of the inter Nanocore communication, so the communication can be derived with minimal additional specifications. A data flow graph is identified in NANOCOREDF by the following properties:

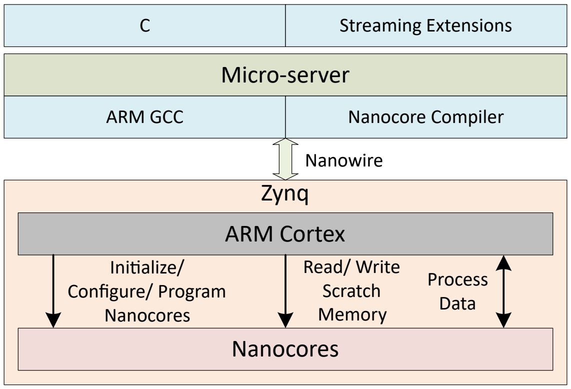

Fig. 5. Nanostreams tool chain support

- One or more calls annotated as NANOCOREDF_ PRODUCE(N). They represent the communication into the Nanocore data flow graph from outside. The initial N arguments are data flow communication arguments: the compiler requires that the values in these are fully written by this kernel, and will not be modified by any other kernel in the graph. These arguments are considered to be outputs of this kernel.
- One or more calls annotated as NANOCOREDF_ CONSUME(N). They represent the communication from the Nanocore data flow graph to outside the graph. The initial N arguments are data flow communication arguments that are used by the kernel.
- Other calls that accept some arguments that were outputs of earlier kernels in the control flow (e.g. a NANOCOREDF_ PRODUCE kernel) are considered part of the graph, and such arguments are considered input arguments. Any other arguments are considered to be output by this kernel.

With this definition, the data flow graph and the sizes of the inter-kernel communication within the graph can be determined by the compiler, the basic information needed to explore how the graph should be mapped to the allowed configurations of Nanocores and inter-core communication.

6 Experimental Setup

6.1 Application Use Case : Option Pricing

In finance, the option to buy or sell a stock at a given price can be traded. Valuation of these options can be complicated and numerous models exist for the accurate pricing of options with reduced complexity to allow trading or occur in real-time. To evaluate the AoC processor, we implemented an option pricing algorithm for European style options using the Binomial Tree model. This offers a generalized numerical method for valuations of options and can be applied for more exotic options which require handling of complex features not easily applied using other models. The algorithm holds a lot of significance in financial computing and was chosen for unbiased evaluation of the platforms.

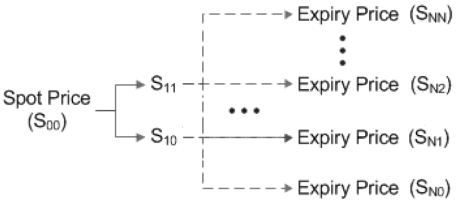

Fig. 6. Binomial tree algorithm

The Binomial Options Pricer (BOP) models the price of an asset as a discrete lattice of prices where the stock can take either an upwards jump u or a downwards jump d at each instant in time (Fig. 6). There are two main stages to the BOP model, the first of which requires calculation of the value of the option at each one of the final nodes. This can be calculated using equation (1), where n_u is the number of upwards jumps and n_d is the number of downwards jumps taken to get to the node.

$$S_N = S_0 * u^{n_u} * d^{n_d} = S_0 * u^{n_u - nd} \tag{1}$$

The second stage involves walking backwards up the tree calculating the price at each node until the first node is reached as mathematical given by Equation (2) being run $(n+1)^2/2$ times, where p_u and p_d are the probabilities of an upwards and downwards movement respectively.

$$S_{i-1,j} = e^{-R\Delta T}(p_d S_{i,j} + p_u S_{i,j+1}) \tag{2}$$

This second stage is more computationally expensive, suits a SIMD style architecture and does not involve complex function such as power functions. It makes for an ideal case for acceleration using FPGA logic.

6.2 Definition of Metrics

We use the following application-specific and platform-independent metrics for a fair performance analysis:

- **Time/option**: From users point of view, the atime per option is the most critical entity that defines the end-to-end latency to price all contracts for a given stock.
- **Joules/option**: The energy consumed per execution of a pricing kernel is a fundamental metric that translates to most significant portion of total energy consumed at datacenters when considering high frequency trading. Any change to the $T/option$ metric, by introducing high performance options, has to be balanced against a change in the $J/option$ metric.

6.3 Evaluation Platforms

Our experimental setup consists of three classes of platforms (Intel, ARM, Nanostreams) on which we executed the OptionPricer under various workloads, measuring energy consumption and performance in each case. The technology characteristics of devices in all platforms have been summarized in TABLE 4.

We used the $4.7.3$ version of the GCC compiler for ARM based systems and the Intel Compiler ICC version $14.0.020130728$ for code generation on Intel platform. Both

TABLE 4
Device characteristics summary

Platform	Device	Technology (nm)	Transistors (Billions)	Clock (GHz)	Parallelism (Threads)
Server	2 x Intel Xeon CPU E5-2650	32	4.40	2.80	16
Microserver	Viridis	40	1.6	1.40	64
ODROID	Samsung Exynos 5422	40	0.15	2.0	8
AoC-8	XC7Z020	28	0.31	0.25	8
AoC-32	XC7Z100	28	1.5	0.25	32

platforms offer the possibility of scaling their frequency and voltage through a DVFS interface. We conducted experiments only with the highest voltage-frequency settings on each platform, to which we refer as performance mode which previous work has shown to be more energy efficient [35]. Other details of the platforms are given below.

6.3.1 Intel

The first platform is an Intel Sandy Bridge processors based HPC server architecture (referred simply as Intel in rest of paper). This is an x86-64 server with Sandy Bridge architecture, with 2 Intel Xeon CPU E5-2650 processors operating at a maximum frequency of 2.8 GHz and equipped with 8 cores each. The machine has 32 GB of DRAM (4 × 8 GB DDR3 @ 1600 MHz). The evaluation of the server for the use case has been accomplished using both the processors and all 8 cores of each processor with 32 threads parallel threads running on the system.

6.3.2 ARM based Microserver

The second class is of ARM based computing nodes. Among that we analyse a Calxeda ECS-1000 microserver using ARM Cortex-A9 processors, packaged in a Boston Viridis rack-mounted unit (referred to as Viridis) and ODROID-XU3 based on Samsung Exynos-5422 (referred to as ODROID). Viridis server is a 2U rack mounted server containing sixteen microserver nodes connected internally by a high-speed 10 Gb Ethernet network. This means that the platform appears logically as sixteen servers within one box. Each node is a Calxeda EnergyCore ECX-1000 comprising 4 ARM Cortex-A9 cores and 4 GB of DRAM running Ubuntu 12.04 LTS. The performance mode frequency is 1.4 GHz. To evaluate the performance, all 16 nodes are run in parallel with 4 threads per node and 64 threads in total.

ODROID node is based on Samsung Exynos-5422 having quad core Cortex-A7 and quad core Cortex-A15 running at 2.0 GHz. The node has 2 GB of DDR3 RAM at 933 MHz. We only use 4 Cortex-A15 for our evaluation purposes to avoid load imbalance between big and little cores.

6.3.3 Nanostreams

The final class is our proposed microserver architecture, Nanostreams, which uses Viridis supported by AoC. We analyse two AoC accelerators; the first one, AoC-8, comprises of 8 64-bit Nanocores implemented on Avnet ZedBoard and the second one, AoC-32, has 32 64-bit Nanocores implemented on Avnet Mini-ITX. The second platform has been included to analyse the scaled-up performance compared to the base architecture. Along with Nanocores, ARM Cortex-A9 serves as the master for both platforms.

ZedBoard supports Zynq-7000 All Programmable SoC XC7Z020 along with 512 MB of DDR3 RAM. There are two single/double floating point supported ARM Cortex-A9 CPUs on SoC acting as PS. They support a frequency up to 866 MHz with 32 KB level 1 cache, 512 KB level 2 cache and 256 KB on-chip memory. The PL is an equivalent of Artix-7 FPGA with about 85K logic cells, 220 programmable DSP slices and 4.9 Mb of block RAM. Mini-ITX is based on XC7Z100 SoC of Zynq-7000 family with 1 GB SDRAM each for PS and PL. The PS on SoC is the same as XC7Z020 while PL is an equivalent of Kintex-7 FPGA with 444K logic cells, 2020 programmable DSP slices and 26.5 Mb of block RAM. Both the designs have cores running at 250 MHz. The frequency has been kept limited as the performance was limited by the lower frequency of memory access and a higher frequency for cores only resulted in higher power consumption.

For the best possible implementation of BOP algorithm on the AoC SoC, we distribute the processing among ARM and Nanocores. To test our AoC processors, we integrated it with a Viridis microserver through a fast and lightweight network stack. The ARM calculates and send the initial values $(p_u, p_d, e^{-R\Delta T})$ to the Nanocores since they are calculated only once. The control logic is a state machine with four states; initialization of instruction memory, idle and waiting for go signal from ARM, sending of data to Nanocores and them performing the binomial walk. We also compared it to available solutions for the non-programmable solutions on FPGA both from performance and programming effort point of view.

6.4 Real-time fast stream processing

A significant effort has been put in to mimic a real-time environment for stream processing of option contracts. Stock price data at the real update rate was recorded for a full session of normal trading in July 2014 i.e. no significant initial public offerings (IPOs) or other skewed trading patterns took place on the market that day, and subsequently used to perform all of the experiments. The same session was replayed from a central server via UDP multicast and streamed to all the nodes in the system including Intel, ARM and Nanostreams based nodes. Each node is running an instance of OptionPricer which is triggered when it receives a new price update. The pricing of contracts needs to be accomplished in time before the new price update and

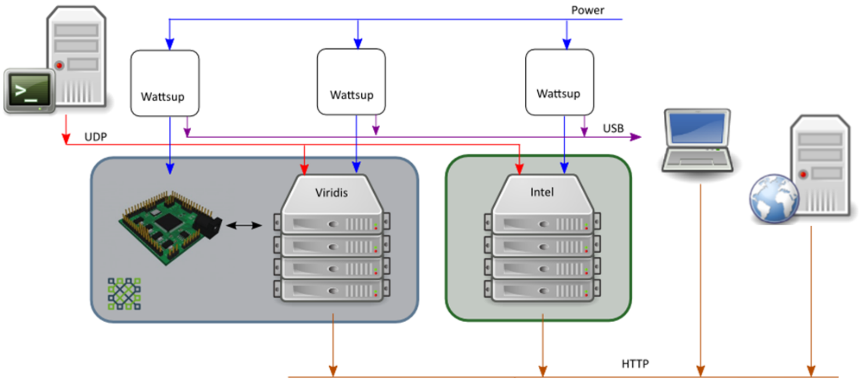

Fig. 7. Binomial tree option pricing experimental setup

projects a deadline based computation. For Intel and ARM, all the option contracts were evaluated on the host while for Nanostreams they were streamed into the AoC which evaluates the contracts and relays them back to the host. We evaluated performance on Facebook stock at the time which had 617 option contracts. The total time taken to evaluate 617 contracts as well as average power during that evaluation was measured and processed to evaluate seconds/option and joules/option metrics.

7 Results

The different blocks, AoC Architecture and Nanowire are evaluated followed by system performance results and comparison with other platforms.

7.1 Nanocore and AoC Architecture Evaluation

The performance of programmable Nanocores with other high level synthesis options available from FPGA vendors has been compared using three approaches. Firstly, we evaluated a single Nanocore against a single Microblaze configured as close as possible to the Nanocore, involving a shifter, 32-bit integer multiplier and extended stream instructions. In our Zedboard-based study, we found the performance mode as compared to the area or frequency modes, gave the best possible trade-off between resource utilization and frequency. In this setting, a single soft-core resulted in a frequency of 160 MHz against a Nanocore frequency of 325 MHz (Fig. 4) for default implementation settings on Vivado 2016.4 for lowest speed grade ZedBoard chip, the same used for Nanocores frequency analysis. Details are given in TABLE 2 and shows that Microblaze takes about 2/3 of the resources while running at half the frequency, meaning that Nanocores is about 33% better than Microblaze considering the for area-delay product in terms of used LUT resource.

The frequency and resource usage alone cannot provide an accurate estimate of performance. Hence we compiled the base kernel of BOP for Microblaze. The assembly code generated by the compiler of Microblaze takes 23 cycles compared to 22 cycles of Nanocore. This defines the best case performance of Microblaze as about half of Nanocore. Furthermore, the better scalability that can be expected from less resource usage of Microblaze may not directly translate to better performance due to I/O and memory bottlenecks on a scale-up design.

The second evaluation is for 8 of 64-bit Nanocores against hardware generated by Xilinx HLS tools. To quantify performance, we implemented the same second stage of BOP kernel via HLS using 64-bit fixed precision. Basic loop unrolling and pipelining directives from HLS were applied to enhance performance, although the limiting factor was the nature of the algorithm, that requires data dependency and unsymmetrical compute architecture in each iteration of loop.

The third is a system level comparison for a complete end-to-end solution, accomplished using OpenCL support on Altera FPGAs provided through SDK for OpenCL by Altera and implemented on Altera Stratix V GX A7 SoC built on Nallatech P385-A7. The kernel had 8 threads with 3 parallel instances of option pricers running in parallel. This bigger FPGA device is compared with AoC-32 system implementation for complete kernel execution.

7.1.1 *Performance*

The DSPs were the limiting factor in both designs, 87% in the Xilinx HLS design and 77% in the Altera FPGA, showing the compute intensiveness of the algorithm. The performance comparison is given in Figure 8. For Nanocores, both hardware simulation (for Nanocores only) and complete system (for AoC) results are shown for comparison with HLS based hardware design and AOCL based system design respectively. Xilinx HLS based design was found to be about 25% slower than the programmable Nanocores. AOCL based design is about 4x slower while it consumed 1.5x more power than AoC-32.

7.1.2 *Productivity and Programmability*

In terms of productivity and programmability, typically synthesis times using Xilinx High Level Synthesis tools and Altera Stratix V synthesis using OpenCL will range from 15 minutes to 24 hours. The Microblaze avoids the lengthy synthesis process by providing a programmable single general purpose processor. The program code is compiled and linked and Microblaze application file is generated via Xilinx tools. This, along with hardware description is used to generate a bitstream that needs to be loaded onto FPGA. Once a hardware description file is available for

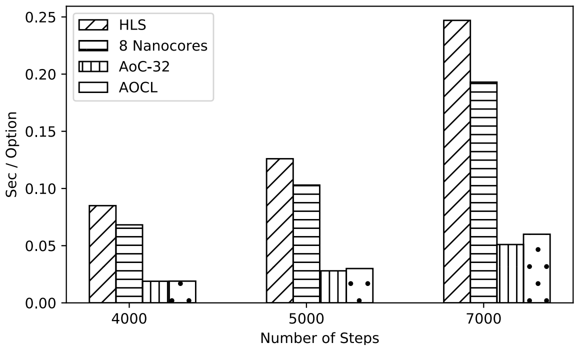

Fig. 8. Binomial tree kernel execution time (seconds/option) comparison for varying number of steps for various reconfigurable solutions

programmable core, a new code can be compiled and linked to soft core using associated tool chain. This makes the cores more suitable for general purpose processing rather than as a co-processing accelerator that can be abstracted to high level parallel programming model and accessible remotely to microservers.

Our solution is able to retest the underlying hardware for design iterations in less than a second, thus allowing for rapid prototyping of multiple applications. If a statically optimized and compiled code is already available, the runtime can reprogram the underlying FPGA based accelerator in a few milliseconds. This is faster than any of the vendor provided solutions including partial reconfiguration and together with remote reprogramming via microserver allows for easy sharing and scalability of accelerators.

7.2 Nanowire Evaluation

We evaluated the performance of Nanowire with three scheduling policies for the Nanowire protocol threads in the kernel:

- *s*pin: Each thread spins on a queue waiting for requests and completions. The spin policy results in the best latency, whilst consuming more CPU.
- *s*leep: Threads sleep and are woken up by interrupts (completion path) or conditional variables (request path) when there is work to do. The sleep policy, results in the lowest CPU utilization, however, at the cost of increased latency.
- *a*daptive:Threads adaptively spin for a while and if there is no work, they go to sleep. The adaptive policy balances latency and CPU utilization.

Figure 9 shows a comparison between the roundtrip overhead of our baseline user-space only implementation of Nanowire for a 512 bytes task, against our kernel-based one for each scheduler policy. The user-space implementation utilized the existing user-space packet socket API and provided a blocking (synchronous) interface. The improvement achieved over this baseline approaches 50% on the host side clearly demonstrating the higher efficiency of our latest design. For the evaluated policies, the spin policy achieves a host latency of 15.2 μs/task, whereas the sleep policy is 20% worse at 19.1 μs/task. The adaptive policy is within 2.5% of the spin policy.

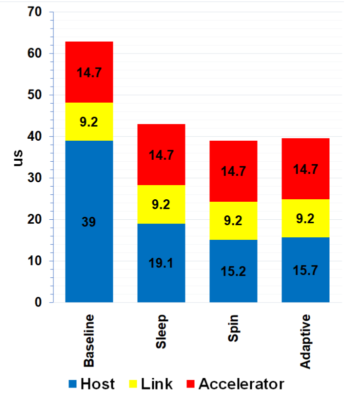

Fig. 9. Nanowire task latency for the three scheduling policies.

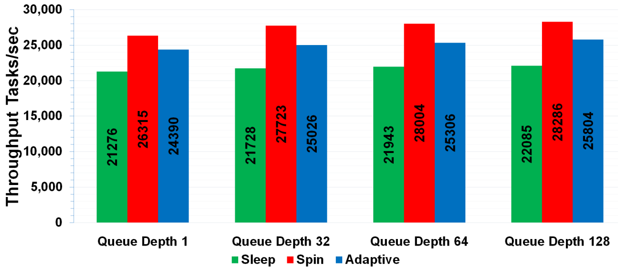

Fig. 10. Nanowire task rate for the three scheduling policies.

Figure 10 shows task rate achieved by each policy. We examine the case with 1 outstanding task, so each task is issued after the previous one has finished. All tasks have a size of 512 bytes (task descriptor and required input data). We see that the spin policy achieves a throughput of 26.3K tasks/s, about 19% higher than the sleep policy (21.2K tasks/s). The adaptive policies manages to achieve about 24.4K tasks/s and is within 8% of the spin policy.

7.3 System Evaluation Across Different Technologies

Before evaluating the platforms, we performed an analysis to determine the problem size. Ideally, the BOP is an iterative model that converges as the number of steps tend to infinity and is limited by time and resource constraints. We performed some experiments to identify the right number of steps that provides required convergence. Simulation results are provided in Fig.11 for one set of parameters while varying the number of steps. Each set of bar graphs represent the price calculated for 3 consecutive number of steps (+1, main value, -1). For 1000 steps, the variations occur up to 2 decimal places while for 4000 and 7000 steps the solution converges to 2 and 3 decimal places, respectively.

TABLE 5 give performance number for Intel, ARM and Nanostreams in terms of seconds/option for various number of steps which can directly be related to the problem/computation size. The best case for both Intel and Viridis is for the lower number of steps, 4000. In that case, Viridis is about 2.22x faster than Intel and about 21x faster than Nanostreams (AoC-32). Nanostreams (AoC-32) was found to be about 9.45x slower than Intel while 1.4 times faster than ODROID. The overall performance gap can be related to difference in scale of devices. The performance

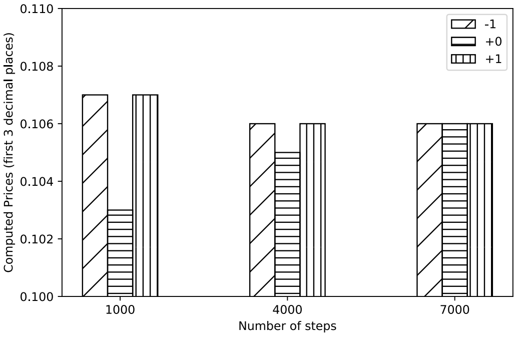

Fig. 11. BOP convergence with number of steps

for a higher step size is interesting. Although performances for Intel and Viridis are better than Nanostreams, they scale worse in terms of time taken per node in the binomial tree. The number of nodes in tree scale as $n(n+1)/2$ where n is the number of steps. For example, for 7000 steps, the time per node for Intel is 3.4ns compared to 2.4ns for 4000 steps, respectively whereas for Nanostreams (AoC-32) it is 20.9ns and 23.6ns, respectively.

TABLE 5
BOP execution time (s/option) for various number of steps

No. of Steps	Viridis (S_{opt})	Intel (S_{opt})	ODROID (S_{opt})	AoC-8 (S_{opt})	AoC-32 (S_{opt})
4000	0.0009	0.002	.005	0.073	0.0189
5000	0.00136	0.0037	.008	0.11	0.028
7000	0.0026	0.0084	0.018	0.201	0.0514

For a better understanding of architecture and more accurate power consumption estimate for Nanostreams, a breakdown of time spent by each of Viridis server and Zedboard (ARM+Nanocores) during execution of a single option pricing for their respective functionalities was undertaken. This supports the concept of Nanocores as a shared accelerator. Viridis server is only responsible for communication and the light communication protocol makes sure that Viridis is kept busy for a minimum time. Thus, in a scaled out system with large number of accelerators, Viridis node will only contribute a small portion of time and power while being engaged with single accelerator. TABLE 6 shows the distribution of time while executing a single option.

TABLE 6
BOP processing time (ms) distribution to price one option

No. of Steps	System Time (ms)	BOPM Kernel (ms)	ARM (ms) % of Kernel time	Nanowire / Network Stack (ms)
4000	72.9	72.8	4.6 (6.3%)	0.10
5000	108.7	108.6	5.7 (5.2%)	0.12
7000	201.2	201.1	0.1 (4.0%)	0.11

Two analyses are important. Firstly, the Network stack which includes the processing time on Viridis, is a very small portion of the overall execution time, in the range of 100 μs. This has significant implications when measuring the system power consumption. For subsequent results, we will only add power for Viridis for when it is active. The second analysis is that ARM on the Zedboard is not fully utilized at the moment and it is only used for about 5% of total kernel execution time. Mini-ITX implementation essentially follows the same computation model and has similar limitations/implications. This offers an opportunity for improvement in future versions or other applications.

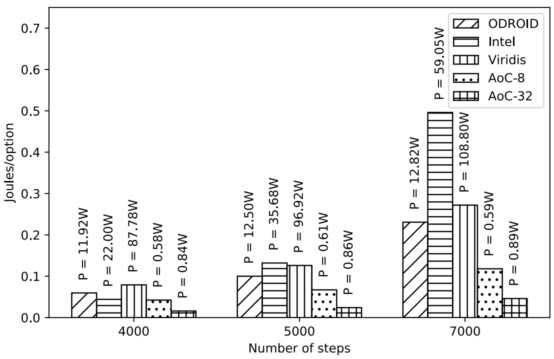

Fig. 12. BOP kernel energy efficiency for various number of steps

Comparison of power consumption across technologies with multiple scale factors, memory hierarchy and peripherals is challenging. Measuring the total power from the wall socket fails to distinguish between actual computing cost and peripherals power, etc. Reporting socket / node / chip power using on-board power sensors such as Running Average Power Limit (RAPL) counters, Intelligent Platform Management Interface (IPMI), etc, can give fine-grained measurements of computing nodes but due to diverse system hierarchy and run-time resource utilization of various technologies, it can be still be inaccurate to measure the exact cost of computing.

For these reasons, we focus on the run-time dynamic power consumption, P_{dyn}. It is calculated at system level that is the total power from the wall but is a difference of compute power and idle power. Compute power in this case is the average system power when the processing is taking place and idle power is the system power when no computation is taking place. This takes into account all the system components that are active during processing. This is more suited to our profiling metric of Joules/option as it gives a micro-benchmark evaluation of particular task on the whole power consumption. For a heterogeneous data center environment with multiple systems working on a range of tasks, this provides a better base for power and energy based run-time decision making. Watts-UP Pro meter [36] was used for all power measurements and has a sampling rate of 1 sec and accuracy within 1.5%.

The results are given in Fig. 12. Generally the energy efficiency gap between Nanostreams and other platforms increase with problem size. Intel and Viridis consume up to about 10.7x and 5.87x more energy to execute the same task than AoC-32 system (mini-ITX). ODROID performs slightly better than Viridis in terms of energy efficiency by avoiding the overhead of distributed computing.

Furthermore for Nanocores, the scaled up system shows

3x better energy efficiency compared to the basic version. The improvement can be related to tight integration and better utilization of memory, input/output and control circuits while the computation resources are increased. The FPGA systems used for experiments are at a much smaller scale factor compared to the other platforms.

Another important analysis comes if relative energy consumption is considered on the same platform for different workload sizes. Joules/option for Intel, for example, for 7000 steps is 11.3x that of 4000 steps while for AoC-32 it is 2.9x. The scaling is significantly higher than execution time scaling presented in TABLE 5. This is because the power increased significantly for long operations owing to cooling requirements of much larger chips on Intel platform while remained almost constant for FPGA based system. This is in line with major concerns laid by scientific community about the end of Dennard scaling; transistors count keep increasing while keeping the voltage constant resulting in higher power density in newer technology nodes [37].

In addition to the energy efficiency improvements, the Zynq-based demonstrator system has a much smaller transistor footprint than the Xeon E5 based on Sandy Bridge-E. The Dual Xeon E5 system has about 4.4 Billion transistors, compared to 1.5 Billion transistors on the Zynq FPGA on Mini-ITX and runs at 2.8 GHz, compared to 250 MHz. Whilst the Viridis platform has comparable number of transistors, the clock speed is still lower. With doubts being cast about future technology scaling, the ability to make these savings at a much lower clock frequency provides a vital protection in the implementations of increasingly complex algorithms.

7.4 Discussion on wider System Applicability

We discuss the overall system model with an aim to relate architecture to various application computing models. The discussion may help designers chose the approach depending on the application.

The proposed approach makes use of heterogeneous computing utilizing tightly coupled computing units. As with the BOP, the exponential functions benefited from ARM cores, which also acts as mater controller, and backward walk benefited from SIMD style computation enabled by parallel Nanocores. Furthermore, the parallel Nanocores can offer better energy efficiency for in-order streaming kernels such as moving average filter, convolution, etc. The Nanocore ISA allowing 4 memory operations per cycle is also suited well for applications requiring higher data operations per computation.

Such types of low latency, high throughput systems are also being pushed into edge analytics, where FPGA is located at the edge of the network for localised processing [38]. Programmable FPGAs here can be used for pre-processing, filtering or transformation of information in a streaming fashion before transferring to subsequent units. Other real-time processing domains using low power programmalbe soft cores include image processing, network control, text analytics, etc. For example, a soft-cores based system with lower precision cores has been applied to traffic sign recognition in [39] and was able to achieve up to 33× speed up over software based implementation. Also, authors in [24] have proposed software programmable soft-cores for communication and video standards (motion estimation, fast Fourier transform) and sobel edge detection in real-time.

7.5 System Limitations

Here we discuss some of the limitations of the system and planned future work. Although, the system proposes to utilize the heterogeneous capabilities of compute units, currently the decomposition and scaling of algorithms on heterogeneous architecture consisting of ARM cores and independently programmable multi-cores is performed by the user. Future work will involve its integration into the tool set via use of automated intelligent code compilation and mapping techniques. Also, currently work has utilized in-order data streaming for a fixed SIMD architecture which limits suitability for algorithms with complex memory architectures. Further work will look to explore more flexibility for inter-core and memory to cores communication.

For example, sparse matrix vector using compressed data storage mechanism such as compressed sparse row, requires access of vector elements based on indexes provided as input. In addition, the order of input data is dependent on the sequence of non-zero elements of the sparse matrix. This requires a complex memory controller which is not supported by Nanocores and if implemented and routed through ARM core, will make the computation extremely memory bound.

8 Conclusion

To cater for the challenging computing and energy efficiency demands of transactional anlytical workload, an integrated microserver architecture, Nanostreams is proposed; it utilizes the AoC architecture which comprises embedded RISC cores for transactional processing and programmable Nanocores for maximum overall energy efficiency. The Nanocores architecture is custom designed for stream processing and at a higher level, provides a complete tool chain and programming directives for the same allowing abstraction of parallelisation through standard C language. The AoC is tightly integrated with microserver using low latency and lightweight network protocol, Nanowire. Nanostreams has been shown to be more energy efficient than a traditional Intel server and ARM based Viridis microserver and ODROID node for application specific and platform independent metrics for BOP, an industry driven use case and has shown performance improvement with scalability.

Acknowledgment

The work was supported by the European Commission under its Seventh Framework Programme, grant number 610509 (NanoStreams).

References

[1] W. Van Heddeghem, S. Lambert, B. Lannoo, D. Colle, M. Pickavet, and P. Demeester, "Trends in worldwide ict electricity consumption from 2007 to 2012," *Computer Communications*, vol. 50, pp. 64–76, 2014.

[2] I. Mitrani, "Managing performance and power consumption in a server farm," *Annals of Operations Research*, vol. 202, no. 1, pp. 121–134, 2013.

[3] P. Stanley-Marbell and V. C. Cabezas, “Performance, power, and thermal analysis of low-power processors for scale-out systems,” in *IEEE International Symposium on Parallel and Distributed Processing*, 2011, pp. 863–870.

[4] E. L. Padoin, D. A. de Oliveira, P. Velho, and P. O. Navaux, “Time-to-solution and energy-to-solution: a comparison between arm and xeon,” in *Workshop on Applications for Multi-Core Architectures*, 2012, pp. 48–53.

[5] N. Rajovic, A. Rico, N. Puzovic, C. Adeniyi-Jones, and A. Ramirez, “Tibidabo: Making the case for an arm-based hpc system,” *Future Generation Computer Systems*, vol. 36, pp. 322–334, 2014.

[6] Y. k. Choi, J. Cong, Z. Fang, Y. Hao, G. Reinman, and P. Wei, “A quantitative analysis on microarchitectures of modern cpu-fpga platforms,” in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016, pp. 1–6.

[7] B. M. Tudor and Y. M. Teo, “On understanding the energy consumption of arm-based multicore servers,” in *Proceedings of International Conference on Measurement and Modeling of Computer Systems*, 2013, pp. 267–278.

[8] Z. Ou, B. Pang, Y. Deng, J. K. Nurminen, A. Yla-Jaaski, and P. Hui, “Energy- and cost-efficiency analysis of arm-based clusters,” in *Proceedings of the IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2012, pp. 115–123.

[9] Y. Durand, P. M. Carpenter, S. Adami, A. Bilas, D. Dutoit, A. Farcy, G. Gaydadjiev, J. Goodacre, M. Katevenis, M. Marazakis *et al.*, “Euroserver: Energy efficient node for european micro-servers,” in *Euromicro Conference on Digital System Design*, 2014, pp. 206–213.

[10] M. Marazakis, J. Goodacre, D. Fuin, P. Carpenter, J. Thomson, E. Matus, A. Bruno, P. Stenstrom, J. Martin, Y. Durand *et al.*, “Euroserver: Share-anything scale-out micro-server design,” in *Design, Automation & Test in Europe Conference*, 2016, pp. 678–683.

[11] M. Lavasani, H. Angepat, and D. Chiou, “An fpga-based in-line accelerator for memcached,” *Computer Architecture Letters*, vol. 13, no. 2, pp. 57–60, 2014.

[12] H. Giefers, R. Polig, and C. Hagleitner, “Accelerating arithmetic kernels with coherent attached fpga coprocessors,” in *Design, Automation & Test in Europe Conference*, 2015, pp. 1072–1077.

[13] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray *et al.*, “A reconfigurable fabric for accelerating large-scale datacenter services,” in *ACM/IEEE 41st International Symposium on Computer Architecture*, 2014, pp. 13–24.

[14] M. Yoshimi, R. Kudo, Y. Oge, Y. Terada, H. Irie, and T. Yoshinaga, “An fpga-based tightly coupled accelerator for data-intensive applications,” in *IEEE International Symposium on Embedded Multicore/Manycore SoCs*, 2014, pp. 289–296.

[15] Y. Wang, X. Zhou, L. Wang, J. Yan, W. Luk, C. Peng, and J. Tong, “Spread: A streaming-based partially reconfigurable architecture and programming model,” *IEEE Transactions on VLSI Systems*, vol. 21, no. 12, pp. 2179–2192, 2013.

[16] T. Miyoshi, H. Kawashima, Y. Terada, and T. Yoshinaga, “A coarse grain reconfigurable processor architecture for stream processing engine,” in *IEEE International Conference on Field Programmable Logic and Applications*, 2011, pp. 490–495.

[17] T. Feist, “Vivado design suite,” *White Paper*, vol. 5, 2012. [Online]. Available: http://www.xilinx.com/support/documentation/white_papers/wp416-Vivado-Design-Suite.pdf

[18] J. G. Tong, I. D. Anderson, and M. A. Khalid, “Soft-core processors for embedded systems,” in *International Conference on Microelectronics*, 2006, pp. 170–173.

[19] P. Yiannacouras, J. G. Steffan, and J. Rose, “Vespa: portable, scalable, and flexible fpga-based vector processors,” in *Proceedings of the ACM International Conference on Compilers, architectures and synthesis for embedded systems*, 2008, pp. 61–70.

[20] Xilinx, “Microblaze processor reference guide,” Xilinx, Tech. Rep., 2006, uG081 (v9.0).

[21] Altera, “Nios ii processor reference handbook,” Altera, Tech. Rep., 2009, nII5V1.

[22] ARM, “Cortex-m1 processor.” [Online]. Available: http://www.arm.com/products/processors/cortex-m/cortex-m1.php

[23] C. Brugger, D. Hillenbrand, and M. Balzer, “River: Reconfigurable flow and fabric for real-time signal processing on fpgas,” *ACM Transactions on Reconfigurable Technology and Systems*, vol. 7, no. 3, p. 24, 2014.

[24] P. Wang and J. McAllister, “Streaming elements for fpga signal and image processing accelerators,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 6, pp. 2262–2274, 2016.

[25] R. Polig, H. Giefers, and W. Stechele, “A soft-core processor array for relational operators,” in *IEEE International Conference on Application-specific Systems, Architectures and Processors*, 2015, pp. 17–24.

[26] P. González-Férez and A. Bilas, “Reducing CPU and network overhead for small I/O requests in network storage protocols over raw ethernet,” in *IEEE 31st Symposium on Mass Storage Systems and Technologies*, 2015, pp. 1–12.

[27] A. Belay, G. Prekas, A. Klimovic, S. Grossman, C. Kozyrakis, and E. Bugnion, “Ix: A protected dataplane operating system for high throughput and low latency,” in *Proceedings of the 11^th Conference on Operating Systems Design and Implementation*, 2014, pp. 49–65.

[28] Q. Jin, W. Luk, and D. B. Thomas, “On comparing financial option price solvers on fpga,” in *IEEE International Symposium on Field-Programmable Custom Computing Machines*, 2011, pp. 89–92.

[29] Q. Jin, D. B. Thomas, W. Luk, and B. Cope, “Exploring reconfigurable architectures for binomial-tree pricing models,” in *International Workshop on Applied Reconfigurable Computing*, 2008, pp. 245–255.

[30] V. M. Morales, P.-H. Horrein, A. Baghdadi, E. Hochapfel, and S. Vaton, “Energy-efficient fpga implementation for binomial option pricing using opencl,” in *Proceedings of Conference on Design, Automation & Test in Europe*, 2014, p. 208.

[31] C. Brugger, J. A. Varela, N. Wehn, S. Tang, and R. Korn, “Reverse longstaff-schwartz american option pricing on hybrid cpu/fpga systems,” in *Proceedings of Design, Automation & Test in Europe Conference*, 2015, pp. 1599–1602.

[32] C. H. Chou, A. Severance, A. D. Brant, Z. Liu, S. Sant, and G. G. Lemieux, “Vegas: Soft vector processor with scratchpad memory,” in *Proceedings of International Symposium on Field Programmable Gate Arrays*, 2011, pp. 15–24.

[33] M. Stonebraker, U. Çetintemel, and S. Zdonik, “The 8 requirements of real-time stream processing,” *ACM SIGMOD Record*, vol. 34, no. 4, pp. 42–47, 2005.

[34] S. Kaloutsakis, “D3.1 base os and user-space network access (version 1),” D3.1_Base_OS_and_User_Space_Network_Access.pdf, Aug. 2014. [Online]. Available: http://www.nanostreams.eu/wp-content/uploads/2014/09/D3.1_Base_OS_and_User_Space_Network_Access.pdf

[35] C. Gillan, D. Nikolopoulos, G. Georgakoudis, R. Faloon, G. Tzenakis, and I. Spence, “On the viability of microservers for financial analytics,” in *Proceedings of WHPCF14: 7th Workshop on High Performance Computational Finance*, 11 2014, pp. 29–36.

[36] E. E. Devices, “Watts up pro,” 2009.

[37] C. Kachris and D. Soudris, “A survey on reconfigurable accelerators for cloud computing,” in *26th International Conference onField Programmable Logic and Applications*. IEEE, 2016, pp. 1–10.

[38] R. Hill, J. Devitt, A. Anjum, and M. Ali, “Towards in-transit analytics for industry 4.0,” 2017.

[39] F. M. Siddiqui, M. Russell, B. Bardak, R. Woods, and K. Rafferty, “Ippro: Fpga based image processing processor,” in *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*. IEEE, 2014, pp. 1–6.

PLACE PHOTO HERE

Umar Ibrahim Minhas received the B.S. degree in Communication Systems Engineering from the Institute of Space Technology, Islamabad, Pakistan, in 2010 and the M.Sc. in Analog and Digital IC design from Imperial College London, U.K. in 2013. He is currently a Research Assistant at Queens University Belfast. His research interests include use of heterogeneous systems, particularly FPGA based SoCs, for energy efficient computing.

PLACE PHOTO HERE

Matthew Russell (M13) received the M.Eng.(Hons.) degree in Electrical and Electronic Engineering from Queens University Belfast, Belfast, U.K., in 2014. Since graduating he has worked as an Engineer with Analytics Engines Ltd. in Belfast, where he has lead development of the FPGA based processor for the Nanostreams project. His research interests include design of heterogeneous systems and big data analytics.

PLACE
PHOTO
HERE

S. Kaloutsakis received the B.Sc from Brunel University in 1998. From 2006 until 2014, he worked at the Software Group of ISD S.A. as a principal Embedded Systems HW/SW Engineer. Since 2014, he as been a Software Research Engineer at the Institute of Computer Science of the Foundation of Research and Technology Hellas.

PLACE
PHOTO
HERE

Paul Barber Paul Barber is an Engineering Manager at Analytics Engines, Belfast, U.K. He has over 15 years experience in software and programmable logic design and implementation in the telecommunications, broadcast and data analytics industries.

PLACE
PHOTO
HERE

Roger Woods (M95SM01) received the B.Sc and Ph.D. degree from Queens University Belfast, U.K., in 1985 and 1990, respectively and is a Full Professor at the university, where he has created and leads the Programmable Systems Laboratory. He has authored over 200 papers and is the holder of four patents. His research interests are in heterogeneous programmable systems and design tools for data, signal and image processing, and telecommunications.

PLACE
PHOTO
HERE

G. Georgakoudis received his Ph.D. from the University of Thessaly, Greece in 2016 and is a Research Fellow in Queen's University Belfast, UK. He has extensive experience of EU and nationally funded projects, in affiliation or collaboration with leading research institutions, including CERTH and FORTH, Greece and LLNL, USA. His research interests are in hardware/software co-design, system software for parallelization and acceleration, and HPC architectures.

PLACE
PHOTO
HERE

Charles Gillan obtained his PhD at QUB in 1988 working on HPC for electron molecule collisions and then worked in the Theoretical Chemistry group at the IBM Research Division in San Jose, USA. He worked for Nortel Networks (U.K.) Ltd until 2004 and since then, Dr Gillan has been a Principal Engineer for Software in ECIT at Queens University Belfast, U.K.

PLACE
PHOTO
HERE

Dimitrios S. Nikolopoulos FBCS FIET (M'97-SM'11) earned the B.Sc.(Hons.), MSc and Ph.D. degrees from University of Patras in 1996, 1997 and 2000. He is currently a Professor with Queens University Belfast, Head of School, Acting Director of the Centre for Data Science and Scalable Computing, and a Royal Society Wolfson Research Fellow. His research explores the boundaries of performance, power and reliability of large-scale computing systems.

PLACE
PHOTO
HERE

Angelos Bilos earned a Diploma from University of Patras in 1993, and MSc (1995) and Ph.D. (1998) from Princeton University. Between 1998-2002, he held an Assistant Professor post in ECE at the Univ. of Toronto. He is now a Professor of Computer Science at FORTH-ICS and the Univ. of Crete, Greece. Current interests include architectures and systems software support for efficient storage systems, low-latency high-bandwidth comms. protocols, and runtime-system support for multicore processors.