# NanoStreams: Codesigned Microservers for Edge Analytics in Real Time

# NanoStreams: Codesigned Microservers for Edge Analytics in Real Time

Giorgis Georgakoudis, Charles Gillan, Ahmad Hassan,
Umar Minhas, Ivor Spence, George Tzenakis,
Hans Vandierendonck, Roger Woods, Dimitrios S. Nikolopoulos
School of EEECS, Queen's University Belfast Belfast, UK BT9 6AY

Murali Shyamsundar
The Centre for Infection and Immunity
School of Medicine, Dentistry and Biomedical Sciences
Queen's University Belfast, Belfast, UK BT9 7BL

Paul Barber,
Matthew Russell
Analytics Engines
Belfast, UK, BT9 5DJ

Angelos Bilas,
Stelios Kaloutsakis
Institute of Computer Science, FORTH
Heraklion, Greece, GR–70013

Heiner Giefers, Peter Staar,
Costas Bekas
IBM Research Zurich
Rüschlikon, Switzerland CH-8803

Neil Horlock
Credit Suisse
One Cabot Square
London, UK, E14 4QJ

Richard Faloon,
Colin Pattison
Neueda
Belfast, UK, BT12 5GH

***Abstract*—NanoStreams explores the design, implementation, and system software stack of micro-servers aimed at processing data in-situ and in real time. These micro-servers can serve the emerging Edge computing ecosystem, namely the provisioning of advanced computational, storage, and networking capability near data sources to achieve both low latency event processing and high throughput analytical processing, before considering off-loading some of this processing to high-capacity datacentres. NanoStreams explores a scale-out micro-server architecture that can achieve equivalent QoS to that of conventional rack-mounted servers for high-capacity datacentres, but with dramatically reduced form factors and power consumption. To this end, NanoStreams introduces novel solutions in programmable & configurable hardware accelerators, as well as the system software stack used to access, share, and program those accelerators. Our NanoStreams micro-server prototype has demonstrated 5.5× higher energy-efficiency than a standard Xeon Server. Simulations of the microserver's memory system extended to leverage hybrid DDR/NVM main memory indicated 5× higher energy-efficiency than a conventional DDR-based system.**

## I. Introduction

Instant access to data and real-time data analytics have the potential catalyse knowledge acquisition, discovery, and responsiveness. Unfortunately, accessing and processing data with low latency and high bandwidth pushes the computational, storage, and networking resources available in high-capacity datacentres to their extremes, due to massive demand [1]. An important change in the computing ecosystem that can help alleviate the pressure of data volume and velocity from high-capacity datacentres is to process data at or near their sources, the so called 'Edges' of the Internet.

NanoStreams is a European Seventh Framework Programme research project that explores the design, implementation and software stack of lean micro-servers that can ingest and process data in-situ at the edges of the network, with real-time guarantees. The project's vision is to achieve energy-efficient processing of concurrent data streams at or near the data sources, thus reducing the latency and energy footprint of real-time data analytics. The project explores a scale-out micro-server architecture to achieve QoS equivalent to that of more conventional rack-mounted servers for high-capacity datacentres, but with dramatically reduced form factors and power consumption. A distinguishing aspect of NanoStreams is that it builds real-silicon prototypes for field deployment. For example, NanoStreams has already demonstrated a successful deployment of a micro-server at the Belfast Royal Victoria Hospital, in collaboration with the NHS and the local Health Trust, to monitor and analyse ICU respiratory data in real-time.

By placing more computational, storage and networking power near data sources, NanoStreams aims to achieve the low latency targets of modern analytics, off-load the brunt of data processing from warehouse scale datacentres, and enable emerging applications such as real-time video analytics, smart cities, grids and buildings, and 5G mobile data analytics. To achieve the aspirations of the project, NanoStreams adopts a hardware-software co-design approach that is common in the embedded systems domain, combined with an HPC system software stack [2]. NanoStreams innovates in multiple areas of the emerging micro-server ecosystem, by proposing:

- A new Analytics-on-Chip (AoC) architecture that enhances the computational capacity of micro-servers with configurable accelerators; the accelerators are based on the *nanocore* (Section II), a configurable tiny core for FPGAs that is directly programmable in C [3].
- A novel bare-metal Ethernet networking infrastructure, the *nanowire* (Section III), which achieves low latency access and sharing of accelerators without affecting the host processor architecture [4].
- A scalable streaming programming model (Section IV-A) embedded within domain-specific sequential programming environments, such as database environments and graph processing tools [2].
- Non-volatile memory technology for the micro-servers that is accessible directly as 'main memory' (Section IV-B) [5].
- Workload-specific optimisation using the concept of iso-quality of service (iso-QoS), applied to three use cases from the healthcare, capital markets, and business analyt-

ics sectors (Section V) [6], [7].
- A range of new methods to fairly compare the efficiency of server architectures (Section VI) and scale these architectures on demand to meet workload QoS requirements [6], [7].

NanoStreams advances the state of the art in micro-servers in several ways by: (a) adding application-specific but programmable hardware accelerators to micro-servers, as opposed to existing solutions that use elaborate hardware design flows and target a single algorithm [8]; (b) providing general purpose low latency networking to access accelerators in the datacentre, as opposed to custom fabrics [9]; (c) effectively integrating streaming and accelerator-aware programming models into domain specific software stacks, moving one step ahead of ongoing efforts to unify heterogeneous programming models [10]; (d) significantly improving server energy-efficiency of micro-servers via on demand and QoS-aware scale-out and acceleration [6], [7].

The NanoStreams micro-server prototype has demonstrated 5.5× higher energy-efficiency than a standard Xeon Server (Section VI). Simulations of the NanoStreams hybrid DDR/NVM memory system indicate 5× higher energy-efficiency than a conventional DDR-based system.

## II. Nanocores: Programmable Energy-efficient Acceleration

The NanoStreams Analytics-on-Chip (AoC) architecture targets low latency stream processing of compute intensive tasks. It is an amalgam of low-power RISC processors for the embedded systems domains and *nanocores*, a new class of programmable compute units. The AoC processor is a heterogeneous SoC that reduces latency in processing of streaming operators issued to the micro-server with the latency-optimised RISC cores, while improving analytical processing throughput on compute and data intensive tasks with the nanocores. The AoC architecture is currently implemented on Xilinx Zynq-7000 family which offers SoC integration of a dual-core general purpose ARM Cortex A9 based processing system (PS) and 28 nm Field Programmable Gate Array (FPGA) programmable logic (PL). This level of integration reduces the communication latency between RISC cores and nanocores and enables efficient on-chip parallelisation of analytical tasks.

### A. Nanocores

Nanocores are a new class of programmable and configurable processors. The single core is designed to allow easy integration with a number of other nanocores, which may not necessarily have the same feature set, to form a multi-core platform. The multi-core platform improves analytical processing throughput by exposing the parallel computational capabilities of the underlying hardware to the software domain and is sufficiently flexible to allow the acceleration of a wide range of applications.

Our nanocore prototype supports 32-bit and 64-bit fixed point arithmetic. This configuration has been selected to demonstrate a key benefit over existing 32-bit only FPGA soft-core microprocessors (e.g. Xilinx Microblaze and Altera Nios II) and to utilise the fixed point DSP capability of the current generation of FPGAs. Besides word size, the nanocore instruction set is also customisable at build time. The default instruction set is Turing complete and additional application-specific instructions can be added as required to provide improved performance. The key aspect of the approach is to optimise the underlying FPGA hardware to allow the creation of a light core which operates substantially faster than existing FPGA-based cores.

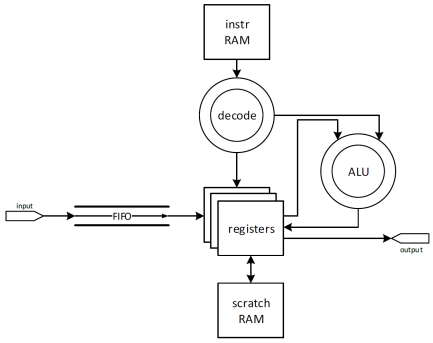


Fig. 1. Nanocore Block Diagram

*1) Memory Hierarchy:* The block diagram of a single nanocore is given in Fig. 1. In order to minimise latency, the input and output memories are configured as first-in-first-out buffers (FIFOs). There are 16 registers (R0-R15) within each processor. Each core has an area of read/write addressable memory for storing intermediate calculation results and use as a stack (the 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.

*2) Instruction Word:* Nanocore operates on a 32-bit instruction word; whilst the Zynq block RAM allows to have 36 bit instruction words, four bits are kept free for future or processor internal use. The instruction set has been designed to allow the core to execute four operations within a single clock cycle: an input read, and output write, an always jump and either a constant load or another instruction. The ability to perform multiple operations within a single instruction should help to reduce the number of instructions required for a tight loop and increase throughput for stream processing.

*3) Hardware Resources and Power Consumption:* Table I summarises the resources usage for one nanocore. The goal of the project is to make the core as lightweight as possible with the minimum instruction set required for the application domain. The per-core power estimates quoted were observed over a long period of time for various programs. These numbers are important for estimating the scale out performance of nanocores on a larger device.

The current maximum frequencies of the cores is graphed against number of cores for the 32 bit and 64 bit version of the core in Figure 2. Note that these results have been obtained

TABLE I
RESOURCE UTILISATION BY SINGLE CORE

| Resource | 32-bit | 64-bit |
|---|---|---|
| LUTs | 1773 | 3144 |
| Flip-flops | 1426 | 1773 |
| MemLUT | 82 | 208 |
| DSPs | 3 | 8 |
| BRAMs | 2.5 | 4 |
| Power/Core | 184mW | 377mW |

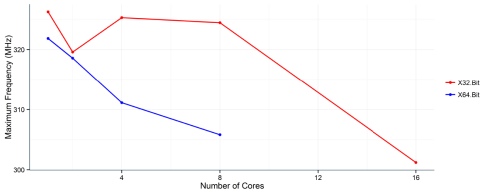


Fig. 2. Maximum frequency with number of cores

using Vivado 2014.3.1s implementation defaults with a constraint of 312.5 MHz. Since most of these results exceed this frequency it is likely that implementation stops optimisation once this frequency is met. This would explain the drop in frequency for the 32 bit nanocore with 2 cores.

*4) Multicore Architecture:* Our prototype supports only Single Instruction Multiple Data (SIMD) type processing, with each nanocore operating on a different burst of data and returning a burst of results. Individual nanocores are operated such that they themselves have no visibility of system data flow. Hence, to maximise the run-time configurability of the nanocore system, some additional elements are necessary in the system architecture to control the input and output streams of data. They also control multiplexing and demultiplexing of data from or to different sources. We call them scatter and gather modules (Fig. 3). All this is to support the NanoStreams goal of operating seamlessly on streaming data and achieve high throughput.

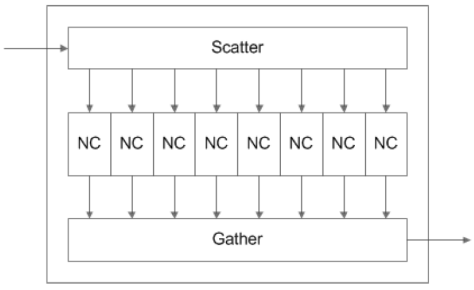


Fig. 3. Nanocore array architecture

The existing architecture can be configured at build time to support a variable number of nanocores with different word sizes and different instruction sets. The nanocore design will also allow run-time configuration of inter-core data routing, providing the capability for the AoC architecture to dynamically adjust to workloads.

*5) Programmability:* The general purpose ARM core works on low latency transactional processing tasks and offloads analytical tasks to the accelerators. The exact distribution of functionality can vary for different applications and can be programmed accordingly using the NanoStreams high level programming model. The ARM core also acts as master for controlling the data-flow through the nanocore fabric, as well as having the ability to program and reprogram the nanocores.

A library has been written to support communication and control of nanocores via the ARM processor. This library provides functions such as initialisation, programming and control of transfer of data to and from the multicore architecture.

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.

Finally, to support faster time-to-market, a development and testing platform, hardware-in-loop (HIL), has been developed using the above mentioned ARM-nanocore control and communication library. HIL runs on the ARM core to stimulate a real nanocore running in the FPGA portion of the chip. The application interface for the tester allows the programmer to access the nanocore's data and control interfaces through wrapper functions. These functions give a high level of access by abstracting the hardware interface around the nanocore and allow for significantly faster hardware verification in real-time.

## III. NANOWIRE: BARE METAL ETHERNET

Nanowire is our communication protocol among hosts and accelerators. Besides efficiency, nanowire primarily aims to:

- Enable shared accelerators, e.g. at the node level or rack level for cost reasons.
- Decouple accelerator from host/server technology cycle, required for accelerators that evolve at a different pace compared to servers.

Ethernet as an interconnect has dominated the datacentre, its performance has been constantly scaling with technology, and is used by all servers. For these reasons, we base nanowire on raw Ethernet. Figure 4 provides an overview of the overall architecture and design but also identifies the data paths established between the host and accelerator nodes during an end-to-end communication transaction. Next, we discuss the main aspects of nanowire in more detail.

### A. Nanowire Protocol

The main functions of the nanowire communication substrate are: (1) A simple and convenient API to virtualise and manage accelerators; (2) Reliable, high-throughput and low-latency transfers; and (3) Minimal host-cpu overheads while supporting high concurrency. Nanowire (Figure 4) is composed of two abstractions: the Host-Accelerator Transport (HAT) layer that handles networking aspects and the Task Issue

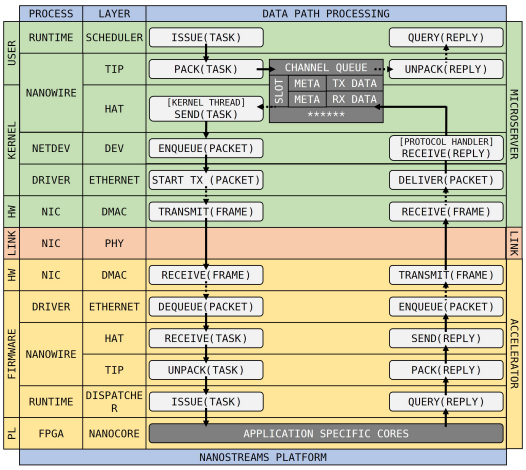


Fig. 4. Host-accelerator communication path in nanowire.

| | PROCESS | LAYER | LATENCY | DIFF | LATENCY | DIFF | |
|---|---|---|---|---|---|---|---|
| USER | RUNTIME | SCHEDULER | 0.0us | +0.0us | +33.1us | +0.0us | MICROSERVER |
| KERNEL | NANOWIRE | TIP | 2.3us | +2.3us | +33.1us | +2.0us | |
| | | HAT | 4.9us | +2.6us | +31.1us | +2.1us | |
| | NETDEV | DEV | 9.9us | +5.0us | 29.0us | +3.6us | |
| | DRIVER | ETHERNET | | | | | |
| HW | NIC | DMAC | | | | | |
| LINK | NIC | PHY | 10.9us | +1.0us | 25.4us | +1.0us | LINK |
| HW | NIC | DMAC | 17.4us | +6.5us | 24.4us | +2.5us | ACCELERATOR |
| FIRMWARE | DRIVER | ETHERNET | | | | | |
| | NANOWIRE | HAT | 18.6us | +1.2us | 21.9us | +1.0us | |
| | | TIP | 19.7us | +1.1us | 20.9us | +1.0us | |
| | RUNTIME | DISPATCHER | 19.8us | +0.1us | 19.9us | +0.1us | |
| | PLATFORM STACK | | ISSUE PATH | | RETURN PATH | | |

Fig. 5. Performance of kernel-based protocol implementation.

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 will run on more diverse platforms. In our prototype the accelerator side of HAT is implemented as custom firmware directly on top of the A9 core in the Processing System (PS) of the FPGA card.

HAT allows multiple hosts to share the same accelerator, it supports variable size packets (up to the Ethernet MTU size), and supports reliable transmission. Packets can be switched via Ethernet switches, however, HAT provides the ability to further customise the Ethernet header for efficiency, when switching is not required.

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

On the host side, HAT implements channels based on user-kernel shared memory to eliminate expensive system calls and achieve low-latency, similar to the approach taken in [11]. This approach has been examined in the past when designing low-latency networks [12], [13], [14], [15] but eventually has not been attractive due to the cost of selective spinning for fat cores. The current trend towards thin and tightly packed micro-servers makes this approach attractive.

TIP provides the runtime 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.

TIP uses a task descriptor to identify the service/kernel to be executed on the remote accelerator node and the corresponding completion notification and result. It supports both blocking and non-blocking interfaces. The protocol is designed to perform end-to-end flow-control and reliable transmission via message retransmission whenever an error is detected. For error detection, we use the checksum mechanism available in Ethernet NICs.

### B. Preliminary performance analysis

Figure 5 illustrates preliminary results for the current prototype of Nanowire. Round trip latency is about 33$\mu$s for the full path, including the network links and NICs. About 11.5$\mu$s is the cost of the protocol on the accelerator (ARM core), 15.6$\mu$s is the host overhead (8.9$\mu$s in the issue path and 6.7$\mu$s in the receive path), and about 6$\mu$s is spent on the network interfaces and the wire.

We are currently working on adaptive interrupt and context switch reduction techniques [16] via interrupt coalescing or polling (NAPI) in the Ethernet driver and the NIC itself. Such techniques are particularly effective when there are concurrent tasks in the system. Additionally, we would like to evaluate in more detail the impact of the shared structure between user and kernel space on system performance and to examine alternatives, as well as to use a custom path path in the Linux kernel and avoid the use of the *netdev* interface, which imposes overheads.

### C. Summary

Nanowire provides an efficient, transparent, and flexible transport between hosts and accelerators. It implements reliable, low-latency, high-throughput, and low-overhead communication channels between the host runtime and shared, application specific cores. We envision that nanowire will be

used to connect accelerators to hosts, both at the board and the rack levels.

## IV. NanoStreams Streaming Programming Model

We focus our discussion on two key aspects of the NanoStreams programming model and runtime environment: Faithful C langauge extensions for supporting hybrid analytical-transactional applications on streaming data; and memory management in heterogeneous, non-volatile memories, which we consider as a viable and sustainable pathway to extend the memory capacity of future micro-servers.

### A. Programming Model

NanoStreams is proposing simple dataflow extensions to C to support a streaming parallel programming model, where tasks and the dataflow between them are explicitly identified via code annotations. We apply minimalistic and faithful extensions of the C language for explicit parallelisation and seamless scaling, considering that the programming model is deployed to support domain specific programming environments such as databases and graph processing tools, as opposed to general purpose parallel programming. This approach also fills a vital gap in landscape of existing parallel high-level languages, namely the lack of a very basic C language extension giving the programmer full control over parallelism.

Stream parallel programming deconstructs a program into multiple kernels linked together via their input and output streams into a graph representing an algorithm. Kernels at each level of the graph may execute independently. Data driven applications with fairly regular computation tend to be a good fit for this programming model. Underpinning its programming model, the project provides a common runtime environment, the *nanoruntime*, suitable for elastic scaling of core provisioning and control of load balancing and data access locality between threads.

The key to exploiting the existing C language lies in having fully referentially transparent code omitting use of pointers, global variables and variable aliasing through function calls. Full use of the type system addresses issues of data locality, bandwidth, and memory hierarchy. Furthermore, by abandoning the monolithic single memory space, we can employ more appropriate and efficient memory management schemes.

We have applied our prototype implementation to our financial and healthcare use cases (Section V). We have implemented a library that enables the programmer to produce a directed acyclic graph expressed in C which can be compiled with gcc using pthreads to produce a running multi-threaded binary.

### B. Memory Management

Extending the memory capacity of micro-servers is challenging both because of grim DDR scaling projections and because micro-servers are fundamentally power and area limited. We are exploring the use of non-volatile memory technologies as a pathway to extend the memory capacity of micro-servers with virtually no additional static power budget and controlled performance cost.

The NanoStreams programming model exposes hybrid main memory composed of conventional and Non-Volatile RAM (NVM) directly to the programmer for data placement and memory management. We are designing a system interface where the DRAM and NVM chips are assigned distinct physical address regions. This complies with how BIOS reports DIMMs and their physical address ranges to the operating system. The operating system can then select to allocate virtual memory pages on either type of memory, using either hints from the user level or an automated kernel-level policy.

To enable user-level management of hybrid main memory we extend the C memory allocation functions (`mma[`, `malloc`) to direct the allocation of memory to NVM or DRAM. We intend to explore more structured and extended interfaces such as numactl. Our modified memory allocation functions implement a default allocation on NVM. We furthermore extend the linker file format to provide two versions of each data segment. For instance, for the ELF file format, the segment .bss hotmem holds zero-initialised data that is frequently accessed in main memory and are destined for DRAM, while .bss holds cold data that can be stored in NVM without performance penalty. The GCC annotation "attribute ((section("bss hotmem")))" specifies global variable placement. We also provide an object migration function between DRAM and NVM, where the programmer allocates a new copy of an object on the opposite memory type and copy the data.

From the OS perspective, allocating memory on a hybrid memory system is similar to allocating memory in a non-uniform memory architecture (NUMA). Every NUMA region is split into a DRAM region and an NVM region. As such, the OS and system libraries utilise the same memory allocation algorithm for either type of memory. Moreover, virtual memory management and virtual address translation are the same for DRAM and NVM and are unmodified in comparison to a NUMA system.

Using the system support described here we have developed an LLVM compiler framework to instrument programs in order to profile the access patterns to all allocated objects, and selectively place objects in NVM or DDR. A limitation is that operating system implementations of NUMA-aware page allocation and migration may contradict programmers choice. For example, Linux will not keep track of the NUMA preference of swapped out pages and may swap them in in a NUMA partition that is not in the memory type and region requested by the programmer. Moreover, there may be conflicting constraints when mapping pages into multiple virtual address spaces. These issues are subject of ongoing study.

Our hybrid memory allocation and policies have resulted in up to 5$\times$ reduction of energy consumption in the memory systems compared to DDR-only or NVM-only designs, for workloads emerging from column-oriented, key-value data stores [5], [17].

## V. Use Cases

This section outlines three commercial use cases of the NanoStreams co-designed micro-server and software stack. s

### A. Reconfigurable compute in a volatile market facing infrastructure

FPGAs have become a byword for compute acceleration in financial services. However, their adoption is far from straightforward and many FPGA-based products have failed to make a lasting impact. One of the most significant constraints to GPU and FPGA adoption in Financial Services, and in particular in the performance critical area of front office trading systems, is the extended software development lifecycle [18].

The trading domain is a fast moving area where market behaviour can change overnight, where new regulations can be imposed with short notice and thus, where agility and adaptability are important contributors to both the financial and reputational risk exposure of the firm.

Bacon *et al.* [19] set out the problems faced with exposing FPGA to conventional development techniques. In particular, they highlight the observable trend towards tighter integration of heterogeneous compute capability, firstly with GPU on die and proposed FPGA on die, packages from Intel and the proliferation of ARM cores integrated alongside gates in commodity FPGA boards. Close integration in hardware makes it clear that a universal way to use these that does not require a different developer, with a different skill set is essential.

NanoStreams embodies a number of key technological innovations that seek to address these challenges. We use the model of options pricing, a problem commonly solved by highly parallel models, to demonstrate that a the combination of micro-server, networking and parallelisation tools of NanoStreams can be harnessed seamlessly. Using the nanocore abstraction we write code for common, high-level languages that is executable both on CPUs and the the nanocore FPGA, allowing reduced hardware and energy footprints without increasing maintenance cost. NanoStreams further demonstrates the allocation of tasks to compute resources by the runtime controller, allowing a choice of execution platform to be made, with the nanowire interconnect connecting the distributed resources seamlessly.

### B. Real-time Graph Analytics at Scale

Graph analytics is fast becoming a key Big Data workload. Graph algorithms have revolutionised the way we interact with the internet, powering for instance search services (e.g. PageRank). Recently, more complicated algorithms that aim to give a far deeper analysis of the underlying graph structures and properties have been emerging. The main driving reason is the use of complex Knowledge Graphs as objects to represent knowledge and allow deep queries.

In all these cases the basic kernel relies on sparse matrix-vector or sparse matrix-matrix multiplication. Although these kernels are not new in traditional scientific computing, it is the very characteristics of the new workloads that make the sparse matrices exhibit very different sparsity characteristics than those in engineering and scientific applications. These lead to extremely irregular memory access patterns that do not typically follow a clear application driven structure. This drives the computational intensity to even lower levels than traditional applications. Therefore, acceleration of particular kernels on customised, low energy platforms is a very appealing path.

A general roadmap that we follow in NanoStreams targets to compute a few characteristic features of the sparse matrices and calibrate prediction models for energy and time to solution [20]. The lessons learned and models calibrated serve as input for the co-design approach that NanoStreams follows to customise the architecture and instruction set of the nanocores, as well as scale resources on demand.

### C. Real-time Computation in Diagnostic Medicine

Today a patient in ICU is surrounded by a large number of monitoring devices recording the time variance of physiological parameters, such as blood pressure or oxygen saturation. The concept of correlating such readings and even feeding these to a predictive mathematical model, is an active research topic in clinical medicine. Earlier research [21] showed the viability of physiological monitoring to detect sleep apnea in neo-natal ICU. NanoStreams extends the concept of utilising real time data to address the challenges of improving the standard of care by using a rapidly responsive automated surveillance system in an adult ICU.

Mechanical ventilation is a common and essential therapeutic intervention performed in critical care units throughout the world, for respiratory and neuromuscular diseases, sepsis, shock, for airway protection, or for temporary support after surgery. Epidemiological studies have shown that up to 2.8% of patients admitted to hospitals undergo mechanical ventilation. Mechanical ventilation can only be performed in critical care units which are a limited and expensive resource. However, mechanical ventilation can worsen the injury in previously damaged lungs [22] and can initiate injury in normal lungs.

The NanoStreams medical use case encapsulates several computational challenges because it seeks to provide a responsive system that is scalable to incorporate multiple physiological parameters and multiple patients. An in-memory database provides a key mediation stage between the patient sensor readings and the subsequent multistage analytical processing. Performance criteria suggest that an in-memory database implementation is a key component of the architecture. Furthermore, in NanoStreams we seek to test the hypothesis of whether microservers provide a suitable platform with which to scale out the processing of this kind of medical data with minimal computational infrastructure in the ICU.

## VI. Putting it All Together

Our fully working NanoStreams prototype integrates an ARM-based microserver, the FPGA nanocores accelerator and the nanowire communication layer. In its current instance, the

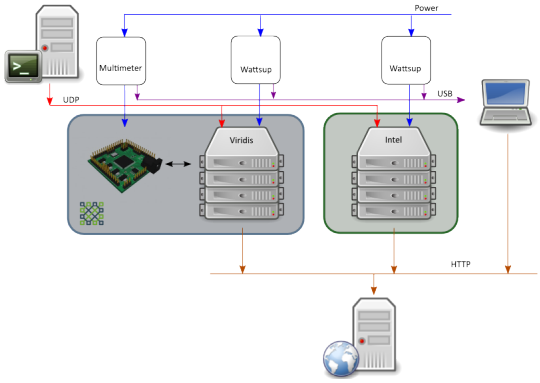


Fig. 6. Setup of the demo system

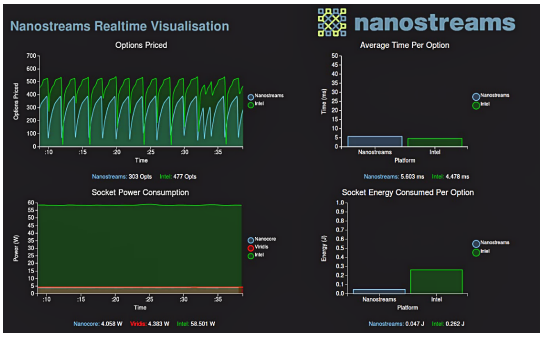


Fig. 7. Output of the monitoring visualisation tool

TABLE II
SUMMARY OF DEMO RESULTS

| | Power (W) | QoS | $T_{opt}$ (ms) | $J_{opt}$ (J) |
|---|---|---|---|---|
| NanoStreams | 8.44 | 49% | 5.60 | 0.047 |
| Intel | 58.5 | 77% | 4.48 | 0.262 |

microserver is a Boston Viridis 2U rack box [23], hosting a cluster of ARM nodes. Each ARM node consists of a quad-core ARM A9 System-on-Chip (SoC) with a shared 4MB L2 cache, 4GB of DDR3 RAM, a 250GB SATA3 disk and a Gigabit Ethernet interface. We are in the process of migrating the microserver host to the Applied Micro XGene platform.

The nanocores prototype is implemented on a Zedboard development kit [24] featuring the Xilinx Zynq ZC020 device and an integrated Gigabit Ethernet interface. The Zynq device includes a dual-core ARM processor for managing the FPGA fabric and facilitating software development. The Viridis microserver and the Zedboard accelerator connect directly via their Gigabit Ethernet interfaces. For measuring energy consumption, each machine equips a Wattsup Pro meter [25] taking power measurements at the PSU level, with a sampling frequency of 1 Hz. Additionally, we make use of on-board IPMI power monitoring sensors with a sampling frequency of 4 Hz. A digital multimeter attached to an electrical interface on the Zedboard is able to measure the whole board's power consumption with a sampling frequency of up to 25 Hz.

Each ARM node in the microserver runs Ubuntu Linux 12.04 LTS which provides a stable environment to develop the application use cases. The server of the nanowire communication layer runs as a bare metal daemon on the ARM management cores of the Zedboard. The nanowire client is implemented as an offloading library that transparently handles communication with the server. An application running on the Viridis ARM microserver compiles against the nanowire client library to access the offloading API.

### A. Demo and Initial Results

We demonstrate our integrated, fully working prototype using the financial use case of option pricing. We emulate a realistic application scenario by replaying a real-time trading feed from the New York stock exchange over a multicast UDP channel. The Viridis microserver, extended with the nanocore accelerator, as well as an competing Intel server tap on this channel to retrieve stock updates and price the *same* set of options. Figure 6 shows an overview of the configuration used. Moreover, each machine exports real-time power and performance measurements to an external visualisation tool (Figure 7) on the web accessible through HTTP.

In the context of option pricing, QoS is defined as the ratio of options that have been priced before the next stock update over the total number of options to be priced. We use a single socket of an Intel Sandybridge as a baseline for comparative analysis. On the Intel server, we measure power at the processor socket level, including the attached DRAM, by using IPMI and RAPL sensors. For the Viridis microserver we measure power at the node level, including the ARM SoC and DRAM. For the accelerator we measure the whole board's power consumption. The accumulated power of the Viridis node and the accelerator board constitutes the total power of the NanoStreams solution.

Table II summarises the results, which suggest that the NanoStreams approach of co-designing microservers can greatly improve energy efficiency. For a fraction of the power consumption of Intel (7× less), NanoStreams achieves an average QoS of 49% timely priced options to a QoS of 77% of the Intel server. Including metrics of average Time per option ($T_{opt}$) and Joules per option ($J_{opt}$) pricing, NanoStreams is roughly 20% slower than Intel when pricing an option but it reduces energy by a factor of 5.5×. Note that the energy reduction from NanoStreams is less than the power reduction because of the slightly increased $T_{opt}$ prolonging the computation. Nevertheless, in the financial use case, this performance difference is readily recoverable by scaling out NanoStreams with more nodes. However, we are investigating more possibilities for better energy efficiency, including using multiple accelerators, improving the architecture of nanocores, using 64-bit ARM hosts with higher-energy efficiency than the current A9 hosts, and further reducing the offloading communication cost.

## VII. Conclusion

NanoStreams has been successful in bringing together best practices from embedded systems design and high-performance computing. We have achieved higher energy-efficiency for analytical tasks on data streams than state of the art servers on real silicon prototypes, while making tangible progress in the development and adoption of new hardware technologies for the European microserver roadmap. As an industry-focused project (five out of seven project partners are companies, including three SMEs), NanoStreams has successfully engaged stakeholders who wish to explore new technologies for in-situ analytics without investing on massive warehouse scale datacentres to meet their needs.

Reflecting on the project, we have also identified areas where taking different directions could have achieved better adoption potential: We have used two compiler infrastructures, one based on ACE's CoSy framework for easy compiler generation and C programming of nanocores, and another based on LLVM for memory access profiling. We believe that unifying everything under a single compiler infrastructures, preferably LLVM due to its prevalence, will significantly broaden the user base. We also believe that there is significant commercial opportunity for isolated components of NanoStreams, including the nanocores and nanowire as core datacentre components, and parts of the language technology as a concurrency extension to the C standard.

Taking a workload-specific approach to sizing and optimising system scale might limit the scope of our results and not cover more general solutions for Edge or high-capacity datacentres. These limitations combined with an intention to broaden the micro-server market potential in Europe have led us to form a new Horizon2020 project, UniServer, which since February 2016 explores general-purpose micro-servers and system software that jointly cope with and exploit intrinsic architectural variation to improve efficiency across a range of Edge computing and IoT workloads.

## Acknowledgement

This research is partially supported by the European Commission under Grant Agreement 610509.

## References

[1] A. Pentland, "The data-driven society," *Scientific American*, vol. 309, Oct. 2013.

[2] C. Gillan, D. S. Nikolopoulos, I. Spence, A. Bilas, and C. Bekas, "Advancing the hardware and software stack for real-time analytics on fast data streams," in *Proceedings of the IEEE 2014* ***eChallenges e-2014*** *Conference*, Belfast, UK, Oct. 2014.

[3] F. Engel, R. Leupers, G. Ascheid, M. Ferger, and M. Beemster, "Enhanced structural analysis for c code reconstruction from ir code," ser. SCOPES '11, 2011.

[4] P. Gonzalez-Ferez and A. Bilas, "Tyche: An efficient ethernet-based protocol for converged networked storage," ser. MSST, June 2014.

[5] A. Hassan, H. Vandierendonck, and D. S. Nikolopoulos, "Software-managed energy-efficient hybrid dram/nvm main memory," ser. Computing Frontiers, May 2015.

[6] G. Georgakoudis, C. J. Gillan, A. Sayed, I. Spence, R. Faloon, and D. S. Nikolopoulos, "Iso-quality of service: Fairly ranking servers for real-time data analytics," ***Parallel Processing Letters***, 2015.

[7] ——, "Methods and metrics for fair server assessment under real-time financial workloads," ***Concurrency and Computation: Practice and Experience***, 2015.

[8] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. G. M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, "A reconfigurable fabric for accelerating large-scale datacenter services," ser. ISCA, 2014.

[9] P. Lotfi-Kamran, B. Grot, M. Ferdman, S. Volos, O. Kocberber, J. Picorel, A. Adileh, D. Jevdjic, S. Idgunji, E. Ozer, and B. Falsafi, "Scale-out processors," ser. ISCA, 2012.

[10] Y. Yan, P.-H. Lin, C. Liao, B. R. de Supinski, and D. J. Quinlan, "Supporting multiple accelerators in high-level programming models," ser. PMAM '15, 2015.

[11] L. Soares and M. Stumm, "Flexsc: Flexible system call scheduling with exception-less system calls," ser. OSDI, Berkeley, CA, USA, 2010.

[12] L. Rizzo, "netmap: A novel framework for fast packet i/o," ser. USENIX Security, Bellevue, WA, Aug. 2012.

[13] L. Deri, "ncap: wire-speed packet capture and transmission." in *E2EMON*, 2005.

[14] 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," ser. OSDI, Broomfield, CO, Oct. 2014.

[15] S. Han, S. Marshall, B.-G. Chun, and S. Ratnasamy, "Megapipe: A new programming interface for scalable network i/o," ser. OSDI, Hollywood, CA, 2012.

[16] P. Gonzalez-Ferez and A. Bilas, "Reducing cpu and network overhead for small i/o requests in network storage protocols over raw ethernet," ser. MSST, Santa Clara, CA, May 2015, pp. 1–12.

[17] A. Hassan, H. Vandierendonck, and D. S. Nikolopoulos, "Energy-efficient in-memory data stores on hybrid memory hierarchies," in *Proceedings of the 11th International Workshop on Data Management on New Hardware* ***(DAMON)****, in conjunction with ACM SIGMOD/PODS 2015*, Melbourne, Australia, Jun. 2015.

[18] M. O'Hara. Fpga and hardware accelerated trading, part four challenges and constraints. [online].

[19] D. Bacon, R. Rabbah, and S. Shukla. Fpga programming for the masses. [online].

[20] A. Malossi, Y. Ineichen, C. Bekas, A. Curioni, and E. Quintana-Orti, "Performance and energy-aware characterization of the sparse matrix-vector multiplication on multithreaded architectures," ser. ICCPW Workshop, Sept 2014.

[21] M. Blount, M. Ebling, J. Eklund, A. James, C. McGregor, N. Percival, K. Smith, and D. Sow, "Real-time analysis for intensive care: Development and deployment of the artemis analytic system," *Engineering in Medicine and Biology*, vol. 29, 2010.

[22] V. Herasevich, M. Tsapenko, M. Kojicic, A. Ahmed, R. Kashyap, C. Venkata, K. Shahjehan, S. Thakur, B. Pickering, J. Zhang, R. Hubmayr, and O. Gajic, "Limiting ventilator-induced lung injury through individual electronic medical record surveillance," *Crit Care Med*, vol. 39, 2011.

[23] Boston Limited. Boston Viridis datasheet. [online].

[24] Avnet. Zedboard documentation. [online].

[25] Watts up? Product website. [online].