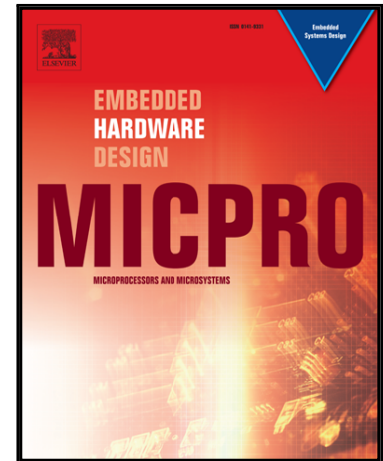


Accepted Manuscript

Exploring Manycore Architectures for Next-Generation HPC Systems through the MANGO Approach

José Flich, Giovanni Agosta, Philipp Ampletzer, David Atienza Alonso, Carlo Brandolese, Etienne Cappe, Alessandro Cilardo, Leon Dragić, Alexandre Dray, Alen Duspara, William Fornaciari, Edoardo Fusella, Mirko Gagliardi, Gerald Guillaume, Daniel Hofman, Ynse Hoornenborg, Arman Iranfar, Mario Kovač, Simone Libutti, Bruno Maitre, José Maria Martínez, Giuseppe Massari, Koen Meinds, Hrvoje Mlinarić, Ermis Papastefanakis, Tomás Picornell, Igor Piljić, Anna Pupykina, Federico Reghenzani, Isabelle Staub, Rafael Tornero, Michele Zanella, Marina Zapater, Davide Zoni



PII: S0141-9331(18)30024-3
DOI: [10.1016/j.micpro.2018.05.011](https://doi.org/10.1016/j.micpro.2018.05.011)
Reference: MICPRO 2692

To appear in: *Microprocessors and Microsystems*

Received date: 17 January 2018
Revised date: 2 May 2018
Accepted date: 15 May 2018

Please cite this article as: José Flich, Giovanni Agosta, Philipp Ampletzer, David Atienza Alonso, Carlo Brandolese, Etienne Cappe, Alessandro Cilardo, Leon Dragić, Alexandre Dray, Alen Duspara, William Fornaciari, Edoardo Fusella, Mirko Gagliardi, Gerald Guillaume, Daniel Hofman, Ynse Hoornenborg, Arman Iranfar, Mario Kovač, Simone Libutti, Bruno Maitre, José Maria Martínez, Giuseppe Massari, Koen Meinds, Hrvoje Mlinarić, Ermis Papastefanakis, Tomás Picornell, Igor Piljić, Anna Pupykina, Federico Reghenzani, Isabelle Staub, Rafael Tornero, Michele Zanella, Marina Zapater, Davide Zoni, Exploring Manycore Architectures for Next-Generation HPC Systems through the MANGO Approach, *Microprocessors and Microsystems* (2018), doi: [10.1016/j.micpro.2018.05.011](https://doi.org/10.1016/j.micpro.2018.05.011)

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

Exploring Manycore Architectures for Next-Generation HPC Systems through the MANGO Approach

José Flich^{a,*}, Giovanni Agosta^{b,*}, Philipp Ampletzer^c, David Atienza Alonso^d, Carlo Brandolese^b, Etienne Cappe^e, Alessandro Cilardo^e, Leon Dragić^f, Alexandre Dray^g, Alen Duspara^f, William Fornaciari^b, Edoardo Fusella^h, Mirko Gagliardiⁱ, Gerald Guillaume^g, Daniel Hofman^f, Ynse Hoornenborg^h, Arman Iranfar^d, Mario Kovač^f, Simone Libutti^b, Bruno Maitreⁱ, José María Martínez^a, Giuseppe Massari^b, Koen Meinds^h, Hrvoje Mlinarić^f, Ermis Papastefanakisⁱ, Tomás Picornell^a, Igor Piljić^f, Anna Pupykina^b, Federico Reghenzani^b, Isabelle Staub^g, Rafael Tornero^a, Michele Zanella^b, Marina Zapater^a, Davide Zoni^b

^aUniversitat Politècnica de València, Spain

^bDEIB – Politecnico di Milano, Italy

^cPRO DESIGN Electronic GmbH, Germany (provider of FPGA-based Hardware - www.profpga-hpc.com)

^dESL – École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

^eCentro Regionale Information Communication Technology SCRL, Italy

^fUniversity of Zagreb, Croatia

^gEaton Industries SAS, France

^hPhilips Medical Systems, The Netherlands

ⁱThales Communications & Security, France

^jUniversità degli Studi di Napoli Federico II, Italy

1. Introduction

The push towards Exascale is going to radically change High-Performance Computing (HPC). First, the sheer amount of computational resources available are pushing the energy envelope available through the power grid to the point where the size of an HPC centre may be constrained by the availability of power supply. Second, the increase in scale of HPC resources across the world is enabling new use case scenarios, where players previously unable to access HPC resources may now do so through

*Corresponding author

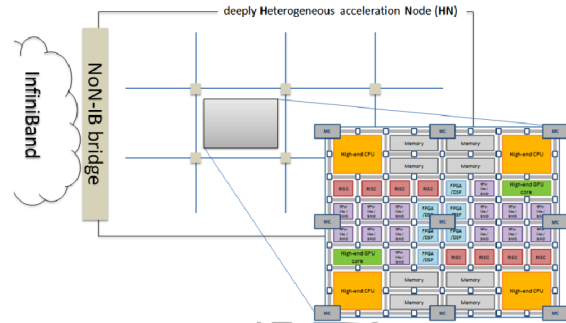


Figure 1: MANGO Hardware Architecture

innovation in delivery modes, e.g. through cloud HPC [1]. Thus, the evolution of HPC hardware and software architectures needs to embrace technologies with high performances and low power consumptions. The current trend is to leverage application-based customization to this end. Deeply heterogeneous architectures can provide such performance/watt improvements, but are clearly much more difficult to program and manage. Furthermore, new application classes, that are QoS sensitive, are entering the HPC domain. In particular, applications such as video transcoding or medical imaging need time-predictability. Since time-predictability and QoS are often not taken into account in HPC, it is mandatory to extend the traditional optimization space from power/performance to *power, performance, and predictability* – the PPP space. In fact, predictability, power, and performance appear to be three inherently diverging perspectives on HPC.

MANGO's [2, 3] key goal consists in addressing the PPP space by achieving extreme resource efficiency in future QoS-sensitive HPC. The present research investigates the architectural implications of HPC applications' requirements to define a new generation of high-performance, power-efficient, deeply heterogeneous architectures with native mechanisms for isolation and QoS.

1.1. The MANGO Approach

Currently, the major challenge faced by HPC is the performance/power efficiency. Looking straight at the heart of the problem, the hurdle to the full exploitation of today's computing technologies ultimately lies in the gap between the applications' demand and the underlying computing architecture: the better the match between

the computing system and the structure of the application, the most efficient the exploitation of the available computing power is exploited. Consequently, enabling a deeper customization of architectures to applications will eventually lead to computation power efficiency. Theoretically, customization can enable improvements in power efficiency as high as two orders of magnitude, since it allows the computing platform to approximate the ideal intrinsic computational efficiency (ICE). ICE is defined as the energy consumption per operation achieved by purely computation circuits, e.g. FP adders.

The current uncertainty regarding on-chip HPC solutions and the essentially open nature of current architecture-level research will be regarded by MANGO as an opportunity rather than as a limitation. The fundamental intuition behind the project is that effective techniques for both performance/power efficiency and predictability ultimately share a common underlying mechanism, i.e., some form of fine-grained adaptation, or customization, used to tailor and/or reserve computing resources only driven by the application requirements. Along this path, the project will involve many different, and deeply interrelated, mechanisms at various architectural levels, from the heterogeneous computing cores, up to the memory architecture, the interconnect, the runtime resource management, power monitoring and cooling, also evaluating the implications on programming models and compilation techniques. In particular, to explore a new positioning across the PPP space, MANGO will investigate system-wide, holistic proactive thermal and power management aimed at extreme-scale energy efficiency by creating a hitherto inexistent link between hardware and software effects, which will involve all layers of an HPC system, from server to rack, to datacenter. The combined interplay of the multi-level innovative solutions brought by MANGO will result in a new positioning in the PPP space. This, in turn, will ensure a sustainable performance as high as 100 PFLOPS for the realistic levels of power consumption (< 15MWatt) delivered to QoS-sensitive applications in large-scale capacity computing scenarios. MANGO will provide essential building blocks at the architectural level to enable the full realization of the long-term objectives foreseen by the ETP4HPC strategic research agenda [4].

The MANGO Consortium is composed of high profile academic institution and industrial partners, well suited

to the execution of the project. In particular, Universitat Politècnica de Valencia plays the role of project coordinator, and provides, in collaboration with Centro Regionale per l'ICT (CeRICT) the set of processor and NoC architectures composing the target heterogeneous nodes. Such nodes will be deployed on FPGA boards produced by ProDesign Electronic GmbH, integrated by Eaton Industries. Politecnico di Milano coordinates the software stack, whereas École Polytechnique Federale de Lausanne (EPFL) focuses on thermal management on both the hardware and software side. Finally, Thales Communications & Security, Philips Medical Systems, and University of Zagreb contribute each a use case scenario.

1.2. Organization of the paper

The rest of the paper is organized as follows. In Section 2, we introduce the three application scenarios that drive the project. In Section 3 we describe the targeted MANGO architecture. Then, in Section 4 we describe the programming models and runtime management resources to be used in MANGO. In Section 5 we discuss the power monitoring framework employed in heterogeneous nodes and in Section 6 the thermal and cooling innovations proposed in the project. Section 7 shows the prototype roadmap. Finally, we draw our conclusions in Section 8.

2. MANGO Application Space

The MANGO project aims at showcasing the need for a dynamic nature of the hardware/software architecture and its capabilities to dynamically use heterogeneous processing elements in a QoS sensitive computing scenario. Therefore, the project draws its requirements and performs its validation on a set of three applications that involve significant QoS aspects.

2.1. MANGO architecture online video transcoding application platform

Multimedia playback has experienced significant growth on different presentation devices. In 2016, the annual run rate for global IP traffic was 1.2 zettabytes (1000 exabytes) and some market forecasts [5] show that it will reach 2 zettabytes per year by 2019. Simultaneously, at a global level, IP video traffic will be in the range of 80 to

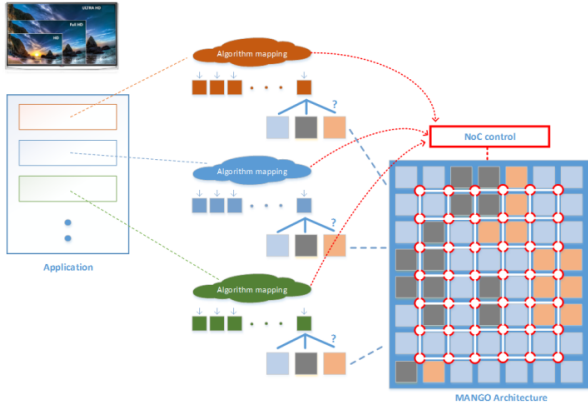


Figure 2: Mapping Applications on the MANGO Platform

90 percent of all IP traffic (both business and consumer) by 2019.

Because of the great variety of devices which are accessing the multimedia content under adverse network conditions, often current video streaming systems do not provide optimal video quality, thus wasting valuable resources or lowering the Quality of Experience (QoE).

Efficient processing of video transcoding, which is extremely compute-intensive and has stringent timing requirements, provides an ideal case-study for the QoS-aware HPC solutions explored by MANGO. The heterogeneous core MANGO HPC transcoding application platform can therefore truly demonstrate how the novel MANGO HPC architecture may become the dominant architecture for applications that generate more than 80% of global internet traffic. The application of the same algorithms to medical domains where interoperability requirements are defined (see [6]) is also under consideration.

The real time video transcoding will use novel video coding algorithms such as High Efficiency Video Coding HEVC/H.265. To enable efficient transcoding, significant work will be required in modeling, mapping and optimizing parts of the algorithms to different underlying MANGO architecture elements (tiles), as shown in Figure 2. Research will not only be focused on high optimization of SW implementation but also on the design of application specific tiles, implemented in HW (such as [7]), that will allow more efficient processing from a performance, power and QoS points of view.

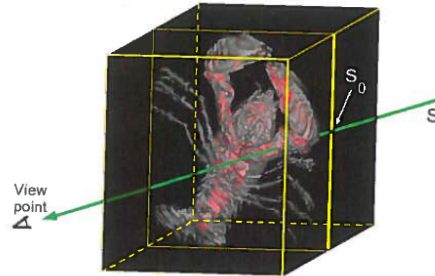


Figure 3: Ray casting through a volume [10]

2.2. Volume rendering for medical imaging

Philips ships a variety of imaging equipment, that can acquire images according to a different process; i.e., MRI (Magnetic Resonance Imaging), CT (Computed Tomography), X-Ray and PET (Positron Emission Tomography). Next to the imaging equipment itself, Philips provides products and services to aid clinicians in their diagnosis on these images. Various image processing algorithms are used for this. The choice of algorithm and the configuration depend on the device used to acquire these images but also on the anatomy and other clinical factors. The acquired images are stored as sets of parallel planar images. These sets can be stacked together, forming a 3D grid of equidistant samples. We refer to such a structure as a volume. The Philips HealthSuite aims, amongst other things, to provide access to such algorithms to both patients and care providers, as health care continues to move outside of the hospital walls and into the homes and everyday lives [8]. This means that these algorithms are performed by more potential users, increasing the performance scalability requirements on the system. Within the MANGO project, Philips studies the scalability behavior of such an algorithm. One of the most commonly used algorithms, is the Direct Volume Rendering algorithm. The prototype used in the MANGO project uses this algorithm. Direct Volume Rendering is a visualization technique that uses the ray casting technique to visualize a volume on a 2D screen. Ray casting visualizes this volume by calculating the amount of reflectance of virtual rays of light and the attenuation along the ray [9].

In Ray Casting, the value of each resulting screen pixel is determined by following a single ray of light through

the volume, as shown in Figure 3. Rays may be processed in parallel since the algorithm is identical for every ray and rays do not influence each other. Along the ray, density values in the volume are sampled. These samples return a grey value, interpolated from the parallel input images. To transform the grey values into colors, a lookup table is used. This lookup table maps a single grey value to a color and an opacity value, based on the reflectance of light modeled by that particular grey value. The accumulation of these colors with their opacities along a single ray will determine the color of a resulting screen pixel. In a volume visualization use case, the calculations are highly memory intensive and input volumes may range from 250 MB to 1 GB. For real-time rendering rates fast algorithm implementation and a low latency are crucial. This is to ensure a better hospital workflow and better support for diagnosis. The solution must scale among many health-care users, all operating on different patient data, while these users interact with the system in real time, with a low latency requirement, and a rendering frame rate of around 25 frames per second. In another medical use cases, the input volume is updated continuously: imaging equipment is increasingly used during minimal invasive intervention. The surgeon relies on the rendered volume for real time feedback to see what he/she is doing. To preserve a good eye-hand coordination a screen refresh time with low latency and low jitter is vital here. The MANGO solution will allow Philips to improve the offering for diagnosis equipment to the hospitals and scale our solution to more users and better serve the patient home market.

2.3. Error correcting codes in communications

Thales will explore the offloading and parallelization capabilities of Low Density Parity Check (LDPC) using the MANGO architecture. LDPC is a linear error correcting code, a method of transmitting a message over a noisy transmission channel [11]. It is among the most efficient error-correction techniques, and although it has been invented in the 1950s, it was practically used from the 1990s [12] because of the high-end floating point computations required. Nowadays, LDPC is used in a growing number of standards in wireless and satellite communications such as WiFi, Wimax, DVB-S2 [13]. In communications, data integrity can be severely impacted from transmission channel's interference, noise and fading. Error correction algorithms are used to maintain transmission capacity by

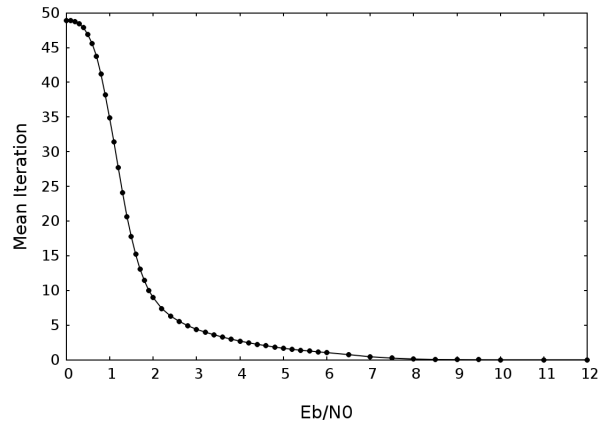


Figure 4: LDPC convergence (mean iterations) in relation to SNR (energy per bit to noise power spectral density ratio, E_b/N_0)

tolerating a loss in transmission rates and latency. The compromise in transmission rates is due to the additional information related to data redundancy tables and information reconstruction which increases the payload size and reduces the useful bandwidth. Similarly, end-to-end latency is added for the processing time needed to encode the useful payload at the sender and for decoding at the receiving end. Since communication capacities and storage capacities are growing, it is necessary to provide architectures in which the scalability of LDPC can be maintained.

Figure 4 shows the calculations, in form of number of iterations (limit fixed to 50), needed to reconstruct the information depending on the Signal-to-Noise Ratio (SNR). As the noise in the channel decreases (increasing SNR), less iterations are needed to converge. In Figure 5 we can observe the quality of reconstruction of an image in respect to the different levels of SNR in a simulated noisy channel.

Through the advancements in the MANGO project, Thales will benefit from energy efficiency and from performance increase, providing QoS guarantees to time sensitive transmission (ex. voice, video). In more detail, parallelism will allow to obtain a significant performance gain in terms of the number of transmission flows that can be active simultaneously. In addition, dynamic resource allocation and heterogeneity will enhance time predictability and energy efficiency. This will be achieved by adapting the aggressiveness of parallelism and the na-

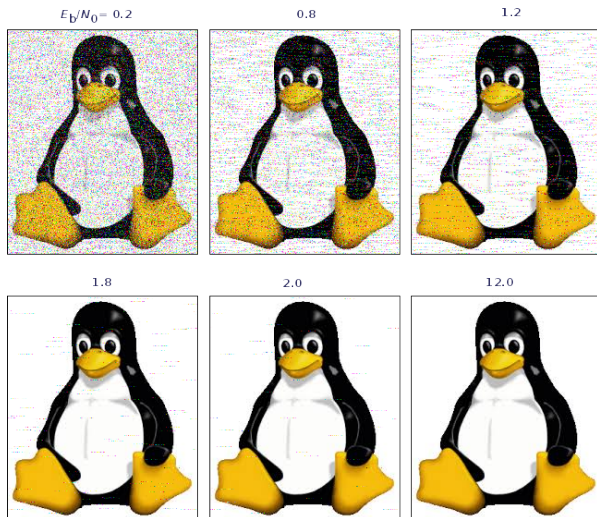


Figure 5: LDPC convergence in relation to SNR

ture of the processing nodes to the channel's SNR in order for all communications to maintain their latency bounds. Finally, optimally placing tasks will improve resource usage and provide energy efficiency.

3. Hardware Architecture Concept

At the architectural level, the MANGO project foresees a scenario where General-purpose compute Nodes (GNs), hosting commercial-off-the-shelf solutions (e.g. Intel Xeon Phi processors or high-end NVIDIA GPU accelerators), coexist with *Heterogeneous Nodes* (HNs), forming a common HPC infrastructure. HNs, as depicted in Figure 1, will essentially be on-node clusters of next-generation manycore chips coupled with deeply customized heterogeneous computing resources. The manycore architecture will be open, it will not rely on COTS solutions available today, and it will enable broad-spectrum, ground-breaking research in the area of on-/off-chip architecture. Building on recent trends in HPC research, in fact, HNs will allow engineers to borrow solutions from the embedded/System-on-Chip domain, which is now recognized as a promising pathway to extreme-scale low-power HPC. HNs will contain a multi-chip mesh of power-efficient RISC cores augmented with custom vector resources (SIMD and lightweight GPU-like cores) as

well as a dedicated memory architecture and a custom Network-on-Chip. All together these will provide advanced support for partitionability and time-predictability. The cores in the multi-chip manycore architecture will be connected through a *Network-on-Node* (NoN), forming a continuum at the off-chip (on-node) level from the on-chip interconnect.

Since the first stages of the project, the architecture exploration will be extensively supported by a purposely developed emulation platform. HNs will not be prototyped in a final ASIC form; rather a mixed approach will be adopted. In fact, RISC processors will be instantiated as ASIC cores tightly coupled with a large-scale reconfigurable hardware fabric used to emulate in near real-time the customized acceleration units, the advanced memory management architecture and the NoN, as well as the NoN bridge to the external interconnect. The platform will support fast design space exploration and validation of the solutions at both the software- and the thermal/power-level. These techniques will inherently involve multiple aspects within the system, from programming down to the architecture definition, deeply intertwined with chip- and system-wide control mechanisms of physical parameters, primarily power consumption and temperature. To gain a holistic understanding of their impact on performance/power/predictability (PPP) and quantitative information about their effectiveness, MANGO will also develop a comprehensive toolset for PPP and thermal models which will operate in close relation with the PPP run-time information collected from the platform.

The MANGO experimental platform will include 16 GN nodes with standard high-end processors, i.e. Intel Xeon E5, as well as NVIDIA Kepler GPUs, along with 64 HN nodes. GNs and HNs will be connected through InfiniBand. HNs will contain ASIC ARM cores and a high-capacity cluster of FPGAs used to emulate the rest of the HN system. The final HN infrastructure will contain dozens of manycore chips, and thus thousands of cores. The prototypical board will enable components to be easily plugged and removed, and it will allow different resource mixes, e.g. nodes highly populated with ARM cores and few high-end FPGAs (e.g. 192 + 64) or vice versa (e.g. 64 + 192), plus memory modules.

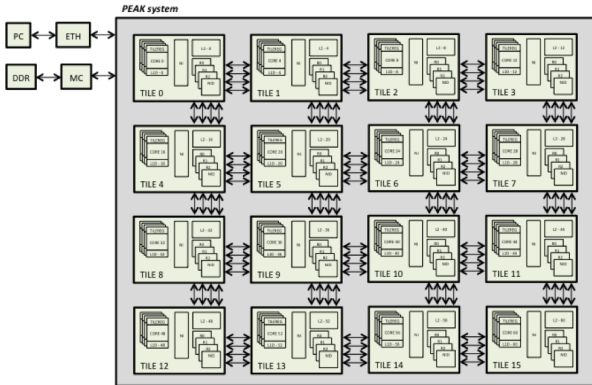


Figure 6: PEAK tile-based architecture.

3.1. PEAK Unit

To explore heterogeneity, The MANGO project is developing different compute units (referred to as UNIT). One of these UNITS is PEAK. PEAK stands for Partitioned Enabled Architecture for Kilocores and it is a research manycore prototype for generic computing.

The main goal of PEAK within MANGO is to offer a configurable processor able to be adapted to different configurations and capabilities, thus enabling exploration of adaptations to the different target applications in the project. The processor has the following key characteristics:

- Runs a large (and sufficient) set of MIPS R32 ISA instructions
- Can be instantiated to any given number of cores, restricted only to the resources available in the target FPGA
- Implements private L1 caches to each core and a shared bank set of L2 caches
- Supports shared memory by implementing an invalidation-based coherence protocol (MESI)
- Implements a sophisticated Network-on-Chip enabling communication between cores, L1 caches, L2 banks, configuration registers, and memory
- Supports exceptions and interrupts.

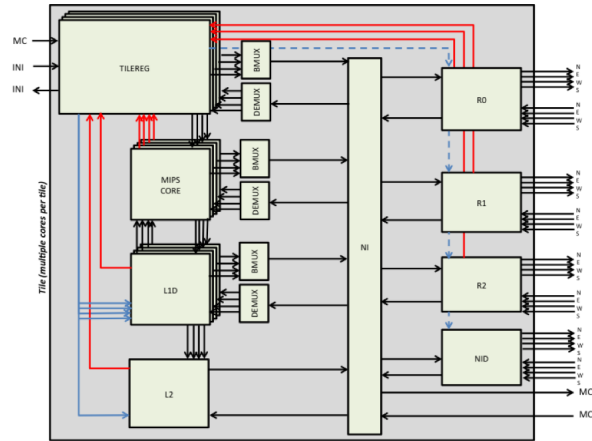


Figure 7: PEAK tile organization

With the functionalities provided above, the PEAK processor is able to run a lightweight OS, simplifying the communication with the Resource Manager used in the project. It is also C compatible enabling simple compilation process with a MIPS cross compiler.

3.1.1. PEAK Architecture

Figure 6 shows the overall architecture design of the PEAK processor. PEAK can be configured as a set of tiles interconnected using a 2D grid array. The number of rows and columns (thus, the number of tiles) can be defined at design time. Each tile is identical in PEAK, including a set of cores (minimum one), one private L1 cache attached to each core, one TILEREG structure also attached to each core, an L2 cache bank, a Network Interface (NI), and four routers.

In standalone mode (without being integrated in MANGO infrastructure), the PEAK system is connected to a memory controller attached to a DDR-3 memory, and to an Ethernet device that enables communication to the server for configuration, and for bidirectional data communication.

Connections at tile level can be seen in Figure 7. The TILEREG module is highly coupled to the core and L1 cache, enabling its configuration and collection of statistics. For a given set of cores in the tile (n), there are n TILEREG modules, each associated to a core. TILEREG associated to core zero is also connected to the rest of tile

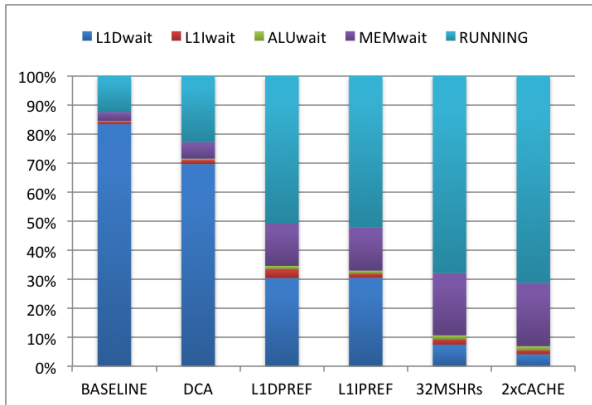


Figure 8: Relative execution time and delays of the PEAK core when booting the system (PEAKos code)

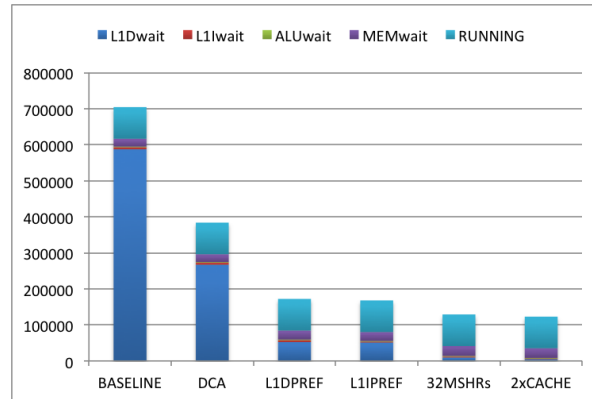


Figure 9: Absolute execution time (and delays) of the PEAK core when booting the system (PEAKos code)

components (NI, L2 bank, routers), collecting also statistics and enabling their configuration. The TILEREGs are connected to a special NI module (not shown) which enables to input and output both configuration and statistics from and outside PEAK.

PEAK core. The PEAK core follows a pipelined in-order architecture with five stages, that is a very simplified processor model. It is argued that simpler cores deliver acceptable performance but with much reduced power consumption. Also, the complexities of out-of-order processors require large FPGA resources, thus impeding the use of large multicore configurations.

The core is a five stage pipelined processor, whose stages are: IF, DI, EX, MEM, WB. The IF stage fetches instructions from a non-blocking L1I cache. The cache can be configured in a number of entries and it uses a direct map strategy. The Program Counter is also associated with the L1I cache. The DI stage decodes incoming instructions from the cache and generates all the control signals to be applied on the next stages. Also, in DI stage, the operands of the instructions are read from a register bank of 32 entries, each of 32 bits. The EX stage performs all logic and arithmetic operations. The next stage, MEM, is in charge of interfacing with the L1 data cache for all load and store instructions. Finally, the WB stage performs writes to the Register Bank.

PEAK memory subsystem. The PEAK processor implements an efficient shared memory programming approach by combining private L1 caches, shared L2 cache banks, and a coherence protocol. L1 caches are private to the core, even if there are several cores per tile. Then, at each tile, L2 banks build up the L2 cache system which is shared by all the tiles. Each L2 cache bank implements a DIRECTORY structure to keep pointers to the L1 caches in the system that shares memory blocks. The last level in the memory subsystem is the main memory implemented with at least one DDR memory. PEAK supports 32-bit addresses, thus having a 4 GB memory address space. When accessing main memory, those addresses, can be translated through specific MANGO structures. However, this is not part of the PEAK system.

PEAK network. The PEAK system implements one network. However, the network supports a configured number of virtual networks (VNs). The fixed value in MANGO for VNs is three. The first two networks (NET0 and NET1) are used for memory-based traffic between L1 and L2 caches. The third network is used to globally access the main memory from L2 caches and for accessing TILEREG structures (in order to control and configure all PEAK components). Traffic injected in a VN will not use resources from other VNs. Therefore, a logical separation exists and protocol-level deadlocks are prevented from occurring.

The router is designed in a pipelined fashion, following

the approach used for the PEAK core. The router stages are IB, RT, VA, SA, XOP. IB (Input Buffer) stores incoming flits (a flit is the minimum amount of info that can be flow controlled, in PEAK is set to 64 bits). Those flits are forwarded to the RT (Routing Unit) where the output port is computed for the incoming message. Only header flits are used in RT. The flit then accesses both the VA (Virtual Channel Allocator) and SA (Switch Channel Allocator) in order to win access to a free VC and to the output port. Each VN has its own VA module. However, only one SA exists per output port. Both VA and SA modules implement a round-robin arbiter. Finally, if all needed resources (VC and output port) are granted, the flit crosses the router in the XOP (Crossbar Output Port) stage.

Basic Core Performance. We analyze how the baseline core behaves. We are interested in analyzing the execution time of a basic application code. In our case, we measure the time that the core takes to boot the low level runtime (PEAKos). This runtime, when executed, detects the configuration, and initializes basic structures such as thread lists, semaphores, and barriers. This process takes around 88,000 processor cycles.

We analyze six different PEAK configurations, each with an additional feature added to the previous one. Table 1 shows the six configurations and the label used for each one.

In the experiment we obtain the number of cycles the core is not stalled (RUNNING) but also we obtain the stall time in cycles of the core due to L1I access (L1Iwait), L1D access (L1Dwait), ALU access due to multicycle instructions (ALUwait), and memory stall cycles due to in-core memory-ALU dependencies (MEMwait). These values will steer our discussion and highlight where core deficiencies are located. Each configuration will address a particular deficiency.

Figure 8 shows the relative percentage of each measure for the six different configurations. As we can observe, in the BASELINE configuration the core is stalled during the boot process mostly because of the L1Dwait component (around 85% of the time). The core is running the code only for 13% of the time. Other stall components (L1Iwait, ALUwait, and MEMwait) are negligible for that configuration due to the excessive L1Dwait component.

One of the reasons for this blocking time accessing the L1D cache is related to the internals of the PEAK design

in the access logic to the cache. The L1D cache has a complex design as it needs to serve requests from multiple sources, mainly the core, the network interface (external requests) and replacement requests. In the BASELINE design the requests are serialized by a multiplexer and, therefore, the core access requests go through that logic, enlarging the core access latency to 6 cycles on a cache hit. This hit penalty severely affects performance as we can deduce from the previous figure. Therefore, with the second configuration (DCA) we implement a direct core access logic to the L1D cache, enabling two ports for concurrent access (when not accessing the same cache blocks). As we see from the Figure 8, the stall time due to L1D access (L1Dwait) has been reduced to 70% and the core is now running efficiently during around 25% of the time. Figure 9 shows the absolute values obtained for a better assessment of the improvement achieved. As we can see, total execution time has been reduced from 700,000 cycles down to 390,000 cycles.

Although this improvement is significant and even if it has a direct access logic, the core is still blocked most of the time due to the L1D access. The reason for this high blocking time is due to the miss penalty component. The boot process is in charge of initializing different structures in memory and this means a series of loops initializing table entries. Tables are consecutive in memory. Whenever the core reads a cache block for the first time, thus resulting in an initial miss, the core has to wait. Once the block arrives to the L1D cache the execution is resumed and subsequent accesses to the block result on hits. However, because the next block is probably fetched soon, due to spatial locality, another miss will occur.

To solve this issue we have added an aggressive 4-way prefetcher to the L1D cache. Whenever the prefetcher detects a pattern access it will launch prefetch operations of future blocks that potentially can be accessed by the core. This logic is highly complex and it may congest memory access. Results, however, show its effectiveness as it is able to reduce blocking time in L1D access from 70% (DCA) down to 30% (L1DPREF). Now, the core is effectively running the code during 50% of the total time. The core starts to be an efficient one.

We can now imagine that the performance results, obtained using the prefetchers in L1D cache, can be expected applying the same approach to the instruction cache (L1I). Indeed, we added to the following configura-

Table 1: PEAK configurations for performance analysis

Configuration	Description
BASELINE	2 cores, with 4KB L1I cache, 64KB L1D cache 64 KB L2 cache bank (128KB in total) 8 MSHR entries in both L1D and L2 caches
DCA	BASELINE + Direct Cache Access from the core to L1D
L1DPREF	DCA + L1D prefetcher units
L1IPREF	L1DPREF + L1I prefetcher units
32MSHRs	L1IPREF + 32 MSHR entries for both L1D and L2 caches
2xCACHE	32MSHRs + Double L1I, L1D, and L2 caches

tion (L1IPREF) a 4-way prefetcher unit in the L1I cache which enables to get instruction cache lines before the core needs them. As a result, we can see that the blocking time due to L1I cache access has been reduced of 50%. However, the impact of this optimization is not as significant as in the case for the L1D cache. The high hit rate already experienced by the L1I cache means that, most of the time, the core is in small loops where all instructions are stored in cache. Anyway, with this optimization we have doubled the effectiveness of the L1I cache.

The next configuration (32MSHRs) deals with the MSHR resources. Indeed, in the baseline configuration the number of MSHR entries is set to 8. This number is sufficient when the core is not prefetching data. Notice that MSHR registers are used to store in-transit memory transactions and they can occur because of core requests, incoming requests from the network, prefetchers issued, and replacement transactions of memory blocks. Therefore, if no proper MSHR entries are set, the core could block because of the reduced effectiveness of the prefetches (as there are not enough MSHR entries to let launch parallel prefetches). For this reason, we increased the number of prefetchers to 32. The net result is that performance is boosted again and the blocking time of the core for L1D accesses is further reduced from 30% to 9%. This means that now the core effectively runs the code during 69% of the total execution time.

Finally, in the last configuration (2xCACHE) we experiment by increasing all the cache modules and doubling their capacity. Indeed, the caches may fill with data and subsequent fetches will infer a replacement action that may affect performance. Results show that this is the case but for the boot process the impact is small. The L1D ac-

cess blocking time (L1Dwait) is further reduced from 9% to 4%, and the core is now 72% efficient.

Notice that in this final configuration, the MEMwait component becomes the main source of blocking for the core. This blocking time, however, is due to the high pressure set to the core as the L1D and L1I caches are now highly efficient and make the core run most of the time. MEMwait cycles are set because of data dependencies between load operations that need to be fed to ALU units. This can not occur in the same cycle as it would significantly impact the clock frequency. Therefore, this blocking is intrinsic to the pipeline design of the core.

To summarize, we can say that, by adding architectural features (booting PEAKos in 3.1 ms instead of 17.64 ms for a 40 MHz clock frequency), the performance of the core has been optimized by a factor of six. However, those optimizations come with a cost, mainly resources taken. Indeed, the overheads in implementation are a main driving factor that will judge whether those performance gains are effective or not. In this sense, in Figure 10 we show the resources needed by each configuration in terms of LUTs for logic, LUTs for RAMs, FFs, and BRAMs for our Virtex 2000T FPGA. Notice that no special effort has been devoted to optimize FPGA resource assignment to the design components, as this may benefit the results for a particular configuration.

As we can see, in relative terms, the overheads needed by the different configurations is moderated. As we add functionalities to the core infrastructure, the overheads increase but with moderated increments of LUTs (logic) and LUT RAMs and FFs (memory). BRAMs are rarely used in the design (they are in fact not optimized for mapping FPGA resources). Therefore, we can conclude that

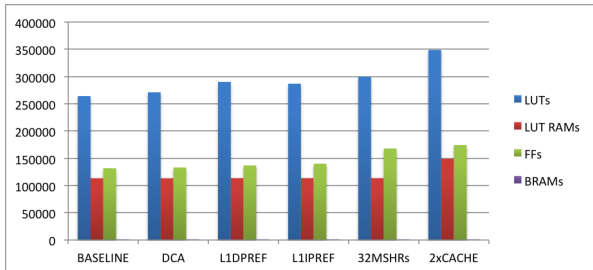


Figure 10: Resources taken by the different PEAK configurations

all the optimizations added to the core are worth being incorporated as they provide much gains (6x improvement) for the small overheads they incur (averaged in 30% increment from BASELINE to 2xCACHE configuration for LUTs, LUT RAMs and FFs).

3.2. Custom tile for video transcoding

Just-in-time video transcoding based on HEVC standard is extremely compute intensive process bound by strict timing requirements which makes it an excellent candidate for exploiting the computation resources of heterogeneous high performance computers.

Efficient video transcoding requires significant work on modelling, mapping and optimizing parts of the algorithms to different underlying architectural elements. Due to its high configurability, considerable number of trade-offs need to be considered to achieve the desired QoS or to reach expected performance in limited amount of time. Software optimizations are required but not sufficient, and the use of the hardware accelerator kernels for critical parts of the algorithm is mandatory to enable efficient processing from the performance, power and QoS perspective. Balancing between these 3 characteristics in real-time presents a great challenge and is often considered as a critical point shown as Advanced transcoding module in Figure 11.

Within the HNs a custom hardware tile was designed specifically for the purposes of video transcoding application. Video transcoding application is based on a novel standard High Efficiency Video Coding (HEVC). Main characteristics of HEVC is 50% better compression efficiency than its predecessor but also high processing and memory requirements bound by stringent timing.

Within the architecture of the HEVC codec there is one kernel that is called frequently and takes significant amount of execution time, regardless of the transcoder configuration. This kernel is used for DCT transformation. Due to its potential for hardware acceleration, Custom accelerator hardware tile (highlighted in Figure 11) was designed and integrated with the MANGO architecture.

The functionality of the hardware tile is shown in Figure 12. The input to the designed accelerator is an $N \times N$ matrix of 9-bit residuals with N ranging from 4 to 32. The input matrix is transformed using two 1D transform operations followed by scaling. The output of the 2D transform is an $N \times N$ matrix of 16-bit coefficients which are then forwarded to quantization and scaling operations. Final output of the accelerator is an $N \times N$ matrix of 16-bit levels.

3.2.1. Video transcoding tile architecture

Discrete cosine transformations are taking a significant part of the video coding time and can be considered as a bottleneck of the system. Therefore, a usage of hardware accelerated calculations can be greatly exploited. In HEVC, a two-dimensional forward discrete cosine transformation (2D-DCT) is used to put the residual matrices into the frequency domain. Implementation of such transformations is usually done as two separated one-dimensional transformations (1D-DCT), firstly applied on the columns and then on the rows. After each of 1D-DCT, the output values are scaled with factors which values are calculated from bit depth and input matrix size. The equation of 2D forward transform can be adjusted to fit the hardware architecture for reusability and area efficiency.

$$[DA \times S_{T1}]D^T \times S_{T2} = [D[DA \times S_{T1}]^T \times S_{T2}]^T \quad (1)$$

Equation 1 represents the HEVC integer 2D DCT of size $N \times N$ where D is the HEVC transform matrix of size $N \times N$ with constant values, A is the residual matrix of size $N \times N$, S_{T1} and S_{T2} are the scaling factors. The right side of the equation uses the property of the transpose operator where it can be seen that transform matrix is multiplier twice. First, it multiplies the residual matrix and then it multiplies the transposed scaled coefficient matrix. This property can be used to design reusable structure

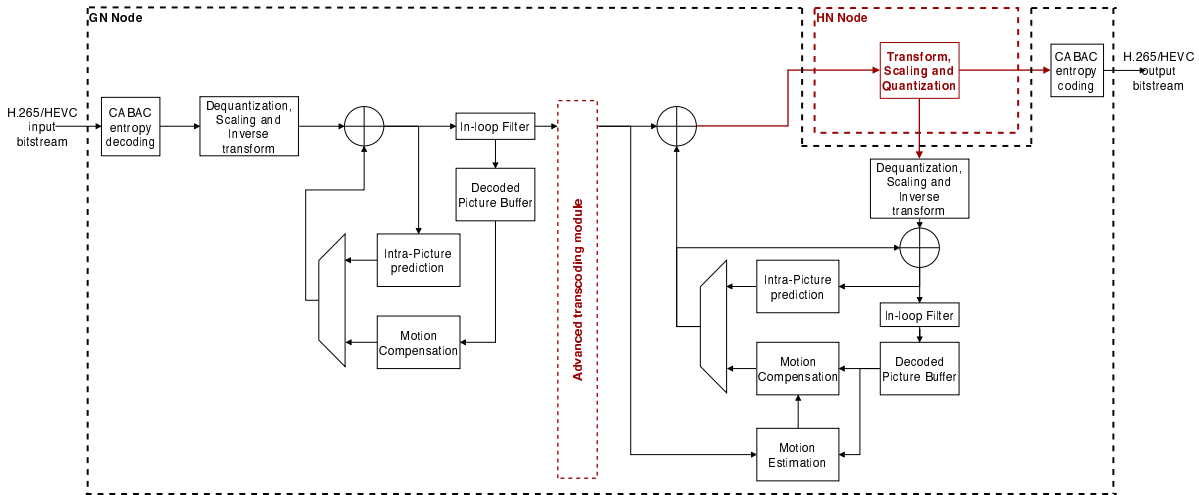


Figure 11: HEVC transcoder scheme

in hardware where the result of first matrix multiplication is brought as feedback to the input and transformed again. Designed hardware unit solution is executing matrix multiplication, scaling and transposition of result matrix. When cascading two of these units, a full 2D-DCT is performed on an input matrix (Figure 13). The communication between the hardware accelerator and memory is the biggest bottleneck of the system. To confront that problem, the design decision was to synthesize both units of the cascade to avoid additional memory communication.

There are four different sizes of the transformation blocks defined in HEVC standard. The core transformation matrices are designed to allow all transform sizes above the 4x4 to reuse the arithmetic operations. Also, these matrices have the symmetry properties that allow implementations to reduce the number of the used multiplications. To reduce the usage of the hardware, the Multiple Constant Multiplication (MCM) units are used where the multiplication process is replaced with a cascade of add-shift operations. Also, to exploit symmetry, the partial butterfly operation is performed on the input vectors. This can be seen on the design of the DCT core in Figure 14.

3.2.2. Reusing DCT cores in hardware architecture

At the higher level, DCT cores are wired in a way that allows hardware reuse (Figure 15). Larger transformation sizes use all smaller cores in the process of calculation. Because the design contains the cores for all transformation sizes, it has the ability to dynamically adopt the size of input matrices. The configuration is set through mode interface.

Input values are in interval from -256 to 255 which gives bit depth of 9 and the result values are 16-bit wide. Thus, interface can manage input and output vectors of 32 half-word values. These values can represent, depending on input matrix size, one vector with size of 32 or two vectors with size of 16, etc.

Every time the 512-bit input is set and valid, the chip enable signal must be raised. This signal is propagated through all the stages of the transformation process. At the end, it can be used as completion indicator or the enable signal for further stages in pipeline.

DCT HW accelerator is designed and integrated in MANGO infrastructure as a single tile in the NoC architecture.

Figure 16 shows an example of tile topology in a multi-FPGA system in which two HW DCT accelerators are instantiated. A wrapper containing TILEREG, TLB, M2U, and U2M modules was designed and provided by UPV

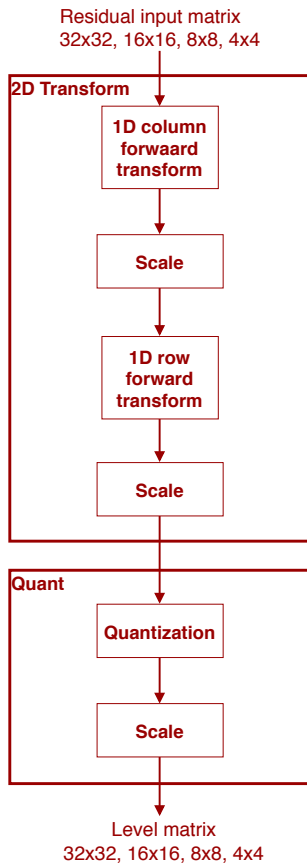


Figure 12: Functional scheme of transform, scaling and quantization module

for easier integration of the accelerator in the MANGO architecture. Two main communication channels shown in Figure 17 are established, Accelerator \leftrightarrow TILEREG and Accelerator \leftrightarrow Mango infrastructure. HW DCT accelerator uses U2M and M2U modules to access memory for reading the input data and writing the results in memory which can be located on any tile or FPGA. For mapping of logical and virtual memory addresses, TLB module is used.

3.2.3. Reusing DCT cores in hardware architecture

Since there are four different sizes of the transformation blocks defined in HEVC standard, a challenge was to design a single hardware accelerator that is able to pro-

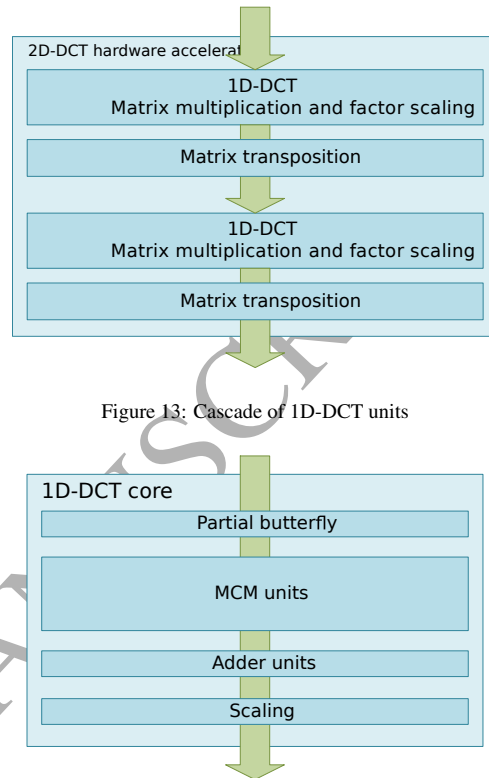


Figure 13: Cascade of 1D-DCT units

Figure 14: The design of the DCT cores

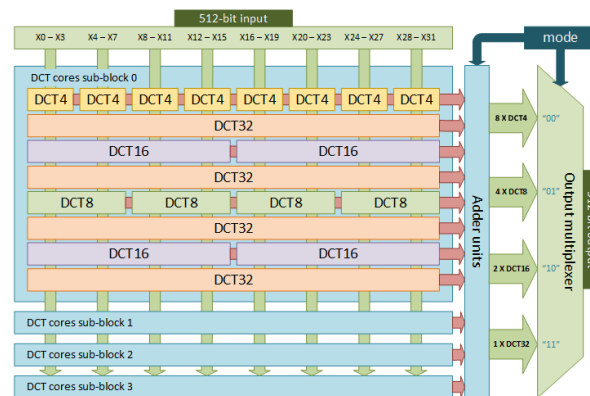


Figure 15: Reusing DCT cores in hardware architecture

cess matrices of all sizes. Designing four different accelerators, one for each block would increase the usage

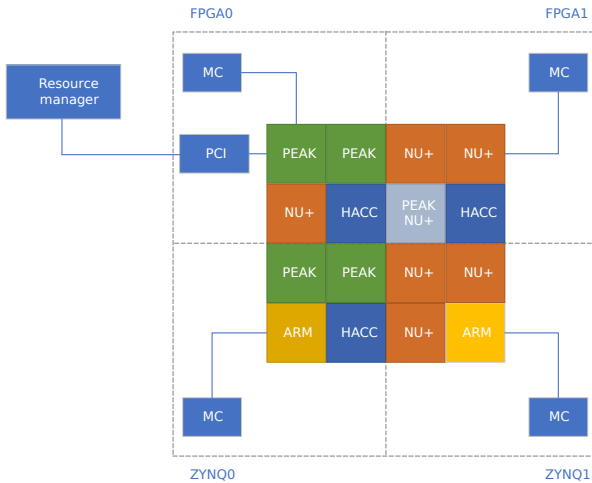


Figure 16: Example of tile topology on multi-FPGA system

of the hardware resources and decrease efficiency, since all initialized accelerators would not be fully exploited. Therefore, a reconfigurable DCT accelerator that is able to transform matrices of all sizes (4×4 to 32×32) is designed. This approach allows better hardware utilization because one instance of DCT accelerator can be used regardless of the type of transformation used in video coding.

Another aspect that was widely considered during design of DCT hardware accelerator is the approach to receiving input data. The bus of the accelerator is 512-bit wide which gives maximal throughput of 32×16 -bit values per clock cycle. The accelerator architecture was designed as a pipelined structure that can consume and output the whole input throughput, regardless of the selected transformation size. For the sizes under 32×32 , the transformation of the input vectors is parallelized.

3.3. *nu+* Unit

nu+ is a complex and configurable GPU-like accelerator core, allowing flexible customization driven by application requirements. It is designed to support the exploration of advanced architecture features deviating from current general-purpose heterogeneous architectures. In particular, it offers the following key characteristics:

- Support for hardware multithreading

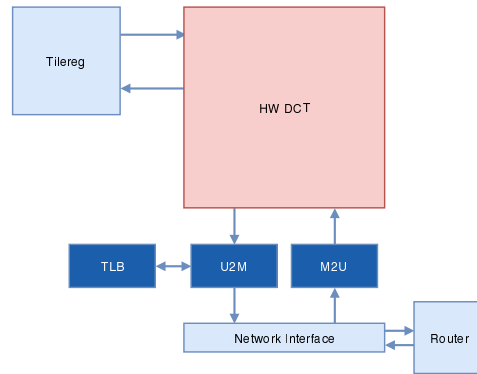


Figure 17: Communication with TILEREG and MANGO infrastructure

- Data-level parallelism through large-size vector/SIMD support
- Multi-/many-core organization allowing non-SIMT execution
- Advanced mesh-based Network-on-Chip
- Lightweight control flow constructs exposed to the programmer
- Hybrid memory hierarchy providing both coherent caches and non-coherent scratchpad memory
- Non-standard floating-point precision values as well as dedicated functions like fused operators.

The system provides a number of knobs for customizing the *nu+* accelerator core to match the characteristics of the applications being accelerated. Some examples include: NoC size, number of threads per core, number of hardware lanes per thread, register file size, L1 and L2 cache size and number of cache ways, mapping between addresses and scratchpad memory banks etc.

On top of the hardware core, we developed an LLVM backend targeting *nu+*. The LLVM infrastructure can potentially be used with any language, but we currently rely on the clang C/C++ frontend.

3.3.1. *nu+* Architecture

Figure 18 shows the overall architecture design of the *nu+* system. It is organized as a manycore system with

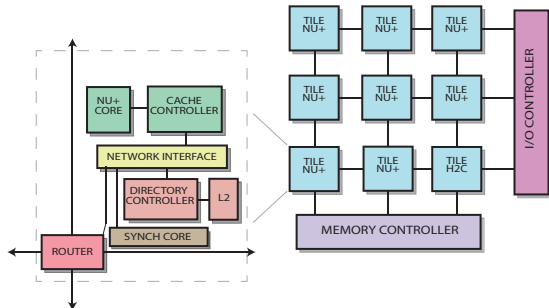


Figure 18: nu+ tile-based architecture

a mesh-based NoC architecture. That means that the nu+ system, forming a single tile in the whole MANGO infrastructure, is in turn a manycore architecture, providing coherent memory access and an intra-tile interconnect that is not made directly visible to the MANGO infrastructure. Each nu+ tile includes the configurable GPU-like core as well as a Cache Controller and a Directory Controller, handling data coherence between different cores in different tiles within the nu+ system.

Each GPU-like core, in turn, offers two further degrees of parallelism in addition to the manycore organization: 1) a hardware-multithreaded organization, where independent execution flows, each controlled by its own Program Counter, are multiplexed to the same functional units; 2) a single-instruction multiple-data (SIMD), or vector datapath, exposed to each thread.

nu+ core. The core (highlighted in Figure 19) is based on a RISC pipeline supporting out-of-order execution and simultaneous multithreading. Memory and long operation latencies are masked by relying on hardware multithreading. Each hardware thread has its own PC, register files, and control registers. All threads share the same compute units. Execution pipelines are organized in hardware vector lanes (like vector processors, each operator is replicated N times). Each thread can perform a SIMD operation on independent data, either floating point operations (IEEE-754 compliant) or integer operations, while data are organized in a vector register file. While the size and organization of the register files can be changed, by default the register file contains 64 scalar and 64 vector registers. The first 58 scalar registers are general purpose, while the remaining 6 are special purpose registers. Each

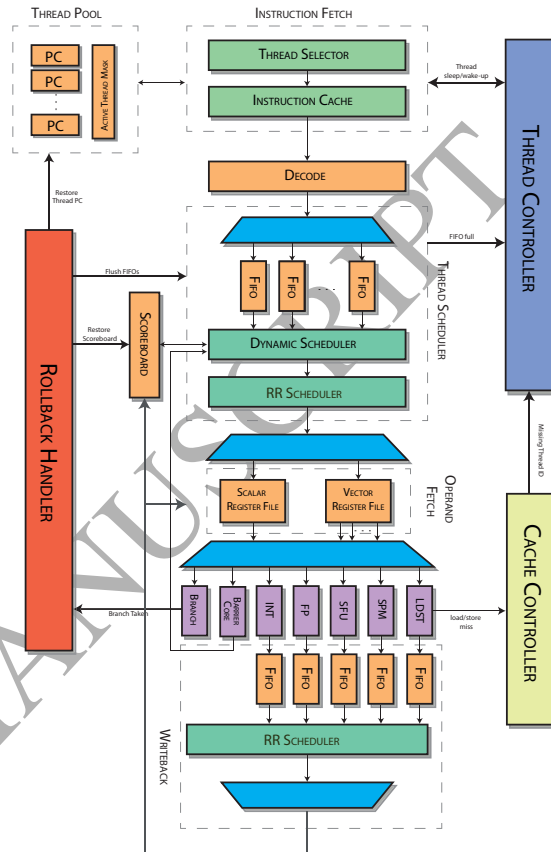


Figure 19: nu+ tile organization

scalar register can store up to 32 bits of data. However, the nu+ architecture can support also 64-bit data, storing them in a pair of contiguous registers. Differently, each vector register can store up to 512-bit data, i.e. 16×32 -bit or 8×64 -bit data. However, it is also possible to store 16×16 -bit, 16×8 -bit or 8×32 -bit, 8×16 -bit, 8×8 -bit data.

nu+ memory subsystem. nu+ implements a hybrid memory hierarchy providing both non-coherent scratchpad memory and coherent memory system with a private L1 cache for each core and a distributed L2 cache that is shared among all cores.

Coherence is handled through a distributed directory-based mechanism for improved scalability. Each L1 cache

is managed by its controller which manipulates data, handles status information, and sends messages to other controllers via the NoC. Data is decoupled from protocol-specific information within the datapath: the Load/Store unit in the core is unaware of the coherence protocol implemented.

The coherence protocol, in the default configuration, is a modified version of the MSI scheme with three stable states for L1 cache (Modified, Shared, and Invalid) and four stable states for L2 cache (Modified, Shared, Non-Cached, and Invalid). The state semantics are the same as the literature, except for the Non-Cached state, which indicates that a memory line is not in the L2 level, hence the main memory has the ownership.

nu+ system comes with an invalidate coherence protocol: when a core modifies a memory block, the coherence system has to invalidate other copies in order to ensure that no core is reading a non-coherent value. This forces each core to request the new value. L2 Caches implement a write-back policy: modified data are forwarded back to the main memory when the data is evicted from the cache itself.

In addition, nu+ supports a high-throughput on-chip non-coherent scratchpad memory. The SPM is divided in a parameterized number of banks. Therefore, if all memory accesses request data mapped to different banks, they can be handled in parallel. The memory controller resolves bank collisions at run-time trying to minimize bank conflicts and ensuring a correct execution of SPM accesses from concurrent threads [14].

nu+ network. nu+ implements a Network-on-Chip based on a 2D mesh topology, wormhole routing, flit-based on/off flow control, and DOR routing. Each tile is equipped with a router and a network interface (see Figure 18). In the default configuration the network provides four virtual channels, three of which are required by the coherence protocol, while the remaining one is shared between the synchronization and boot mechanisms. The router provides five ports, i.e. the four ports corresponding to the cardinal directions and the local port connected to the nu+ core. The router is designed in a pipelined fashion with a look-ahead mechanism allowing routing computation one hop in advance. This allows an improved pipeline design with a lower number of stages compared with a baseline version.

nu+ synchronization architecture. Being targeted at parallel applications, our GPU-like accelerator is expected to provide some form of support for thread synchronization, e.g. barriers. We thus provide a distributed approach supporting multiple barriers for intra- and inter-core synchronization [15]. A dedicated virtual channel, exclusively used for synchronization, ensures synchronization messages not to affect memory/coherence messages. The synchronization core is the key component of our solution. This module acts as the synchronization master, but unlike previously proposed approaches, it is distributed among all tiles in the manycore. Basically, each barrier instance is identified by a different ID and synchronization cores are chosen according to the barrier ID in a modular fashion.

3.3.2. nu+ Toolchain

MANGO relies on LLVM [16] as the reference compiler for the whole project due to its clean, flexible, and modular design and easy-to-use programming interfaces supporting many language extensions, such as vector extensions. HNs require custom compilation support allowing applications written in high-level languages to be converted into target-specific code. In that respect, the compiler for the GPU-like core includes a custom version of the Clang frontend and a native backend implemented from scratch. In addition, the toolchain includes an LLD-based linker that is customized to provide the required support to handle several GPU-like resources such as the scratchpad memories.

The Clang front-end supports application-specific builtin functions that are required to fully customize the GPU-like architecture. In addition, load gather and store scatter, masked instructions as well as other kind of vector instructions, such as shuffle or sign extension operations, are provided to the C/C++ programmer through intrinsic functions. On the other hand, the compiler backend supports 32-bit and 64-bit scalar and vector operations, either floating point operations (IEEE-754 compliant) or integer operations.

Vector types are managed via the *ext_vector_type* attribute in compliance with the OpenCL style. In that respect custom types are defined for all the required vector types. For instance, 512-bit vector registers can be configured to store a vector of 16 32-bit elements or 8 64-bit elements. C/C++ programmers can exploit these vector

types to use parallel instructions that concurrently operate on all the elements of the vector. Similarly to modern GPUs, data that should be placed in the scratchpad memory is managed via the *address_space* and *section* attributes. In this way, a proper set of load/store instructions to the scratchpad memory will be automatically generated.

3.3.3. Results

As an example of the customization capabilities enabled by nu+, we demonstrate how to reduce bank conflicts by changing the parameters of the configurable SPM. As described in Section 3.3.1, the SPM can be deeply customized by fine-tuning its size, number of banks and the data partitioning strategy. In particular, the SPM is embedded with an Address Mapping Unit that implements a generalization of cyclic mapping. It can be profitably used for kernels that generate conflicts with other mapping strategies to reduce the number of conflicts.

We first identified a few kernels that have potentially highly parallel memory accesses and that can benefit from the scratchpad memory support. Many such kernels exist in benchmark suites like PolyBench [17]. We then extracted the access patterns for each kernel and we collected the resulting performance in terms of total bank conflicts for different remapping functions identified for the specific kernel as well as for a variable number of banks.

To show the impact of customization, we provide here some results related to a 5×5 Image Mean Filter that is a typical bank conflict sensitive kernel. The filter replaces each pixel value in an image with the mean value of its neighbors, including itself. We rewrote the code so as to maximize the exploitation of the available parallelism of the GPU-like core. We considered a fixed square matrix size $DIM = 128$ and a fixed number of lanes $numLane = 30$. The total scratchpad memory size is kept constant and equal to $BANKnumber * ENTRYperBank = DIM^2$. We evaluated the bank conflicts for a variable number of banks and for two bank remapping functions:

1. no remapping
2. $(Entry \times 5 + Bank) \bmod NUMBANK$

Table 2: Impact of the SPM configurations on the access conflict count (Image Mean Filter case study)

Banks	No remapping	Remapping
16	7565	1722
32	7565	0
64	7565	0
128	7565	0
256	0	0
512	0	0
1024	0	0

The results are shown in Table 2. It is easy to see that the remapping function has the largest impact on the bank conflict count.

4. Programming Model and Runtime Management

To reach exascale parallelism, the programming model needs to be hierarchical, much like the runtime management system. Traditionally, the programming model for homogeneous HPC systems is based on a combination of MPI and OpenMP. When heterogeneity comes into the game picture, the programming model needs to be extended to allow the exploitation of hardware resources. OpenCL is an open standard for the development of parallel applications on a variety of heterogeneous multi-core architectures [18]. It provides explicit management of heterogeneity, but at a significant cost in terms of tuning performance, which must be performed by the programmer, and building boilerplate code [19, 20]. In MANGO, we aim at integrating the expression of new architectural features as well as QoS concerns and parameters within the existing stack of languages and libraries for extreme-scale HPC systems. We do this by augmenting the runtime library APIs with new functions, as well as by introducing new pragmas or keywords to the language. In a previous work, we performed a performance evaluation and comparison of this extended programming model with respect to the actual standard OpenCL, showing benefits in term of both average execution time and standard deviation [21].

4.1. Resource Management

The main challenge for heterogeneous resource management is the optimization of resource allocation while

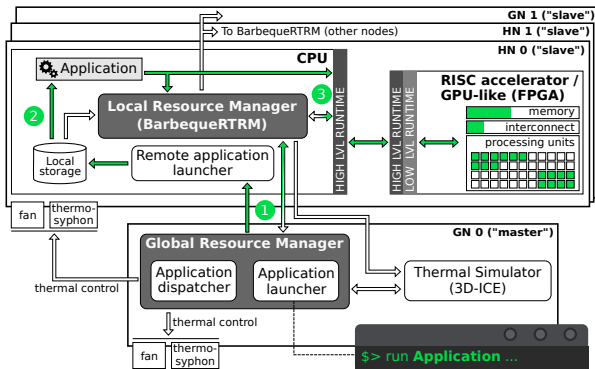


Figure 20: The MANGO Runtime Resource Management Approach

accounting for: 1. each application may be composed of multiple tasks, each of them possibly having data and timing dependencies with the other ones; 2. executing a task on different computing units of an heterogeneous architecture would lead to different throughput, QoS, and power/energy consumption; 3. especially in case of data dependencies, the performance of an application depends not only on where its tasks are executed, but also on where the data of its task are located in the system; 4. requirements coming from each application (usually throughput and QoS) must be complied with, while also addressing the system-wide (power/thermal/energy) requirements. To address this problem, we decouple the description of a task graph, which encodes the work to be done, and its QoS requirements from the decisions that must be taken to optimally allocate tasks and data. The former is addressed by the application developer through an appropriate programming model, while the latter is handled by the Barbeque Run-Time Resource Manager (BarbequeRTRM) [22], which has a system-wide view of the available resources and workload, as shown in Figure 20.

4.1.1. Memory Management

The MANGO architecture is based on a shared memory among all the heterogeneous units in a node. To efficiently manage the available memory resources, we design a memory manager that serves memory requests in a resource allocation-aware fashion, employing knowledge about the evolution of the workload to maximize the utilization of resources while optimizing the ability of the

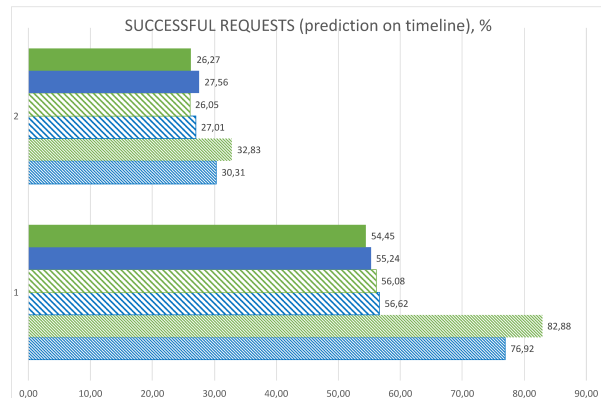


Figure 21: Initial experiments on predictive algorithm for global memory management.

node to serve high priority applications [23]. We focus on the necessity to balance the needs of the application currently requesting resources with those of future requests, in sight of the presence of high priority applications. To this end, we build the predictive models of the requests, and adapt the memory allocation in order to leave enough memory resources for the execution of new tasks on units that are currently free. Since MANGO hardware is typically employed to consolidate applications from a small set of application domains, that are currently deployed on local servers to a remote HPC cluster, this prediction approach can prove quite effective.

In Figure 21, we show the outcome of an initial experiment, run in simulation, where we perform the allocation of 60,000 kernels including a mix of periodic and aperiodic requests. Experiment 1 and 2 differ in memory size – in experiment 1, memory is twice as much as in experiment 2. Green bars represent the percentage of successful requests on high-priority ones, whereas blue bars consider the overall set of requests. Three algorithms are considered: a baseline allocator with no prediction (solid bar), two predictors employing a moving average method (thin stripes) and an exponential weighted average method (thick stripes).

It can be observed that the moving average method provides a good advantage, especially in experiment 1, where the workload is lighter with respect to the available resources.

4.2. Programming Model Support

MANGO aims at supporting parallel programming models across a wide range of different accelerators. We adopt an intermediate runtime layer that exposes basic features which easily map on the hardware features common to all the accelerators (i.e., those provided by the communication architecture). The intermediate runtime support exposes basic tools for communication, synchronization and task spawning. For brevity, only the Context, Event, Kernel and Buffer classes are shown. Context provides facilities to access the resource manager, registering instances of the other three classes. Event, Kernel and Buffer are the objects to be provided with resources (either memory or execution units). The final main class, the TaskGraph, represents a subset of the objects registered with the Context, which is used in a specific run.

Higher level models then build over the intermediate model. The programming model exposes the managed device in a transparent way, allowing the application programmer to define one or more versions of each kernel, and delegating to the runtime manager the selection of the actual execution unit employed to run it.

The following listing exemplifies the use of the MANGO API to run a simple kernel on an accelerator.

```
using namespace mango;

class KernelRunner {
private:
    BBQContext *mango_rt;
    KernelArguments *argsKSCALE;
    KernelArguments *argsKSMOOTH;
    TaskGraph *tg;
    enum { HOST=0, KSCALE=1, KSMOOTH };
    enum { B1=1, B2, B3 };

public:
    KernelRunner(int SX, int SY){
        // Initialization
        mango_rt = new BBQContext();
        auto kf_scale = new KernelFunction();
        kf_scale->load("./scale_kernel",
            UnitType::GN,
            FileType::BINARY);
        auto kf_smooth = new KernelFunction();
        kf_smooth->load("./smooth_kernel",
            UnitType::GN,
            FileType::BINARY);

        // Registration of task graph
```

```
        auto kscale = mango_rt->register_kernel(
            KSCALE, *kf_scale, {B1}, {B2});
        auto ksmooth = mango_rt->register_kernel(
            KSMOOTH,*kf_smooth, {B2}, {B3});

        auto b1 = mango_rt->register_buffer<Buffer>(
            B1, SX*SY*3*sizeof(Byte),
            {HOST}, {KSCALE});
        auto b3 = mango_rt->register_buffer<Buffer>(
            B3, SX*2*SY*2*3*sizeof(Byte),
            {KSCALE}, {HOST});

        tg = new TaskGraph({ kscale, ksmooth },
            { b1, b2, b3 });

        // Resource Allocation
        mango_rt->resource_allocation(*tg);

        // Execution setup
        auto argB1 = BufferArg( b1 );
        auto argB2 = BufferArg( b2 );
        auto argB3 = BufferArg( b3 );
        auto argSX = ScalarArg<int>( SX );
        auto argSY = ScalarArg<int>( SY );
        auto argSX2 = ScalarArg<int>( SX*2 );
        auto argSY2 = ScalarArg<int>( SY*2 );
        auto argE = EventArg( b3->event );

        argsKSCALE = new KernelArguments(
            { &argB2, &argB1, &argSX, &argSY },
            kscale );
        argsKSMOOTH = new KernelArguments(
            { &argB3, &argB2, &argSX2, &argSY2,
              &argE },
            ksmooth );
    }

    ~KernelRunner() {
        // Deallocation and teardown
        mango_rt->resource_deallocation(*tg);
    }

    void run_kernel(Byte *out, Byte *in) {
        auto b1 = mango_rt->buffers[B1];
        auto b3 = mango_rt->buffers[B3];
        auto kscale = mango_rt->kernels[KSCALE];
        auto ksmooth = mango_rt->kernels[KSMOOTH];

        // Data transfer and kernel execution
        b1->write(in);
        auto e1=mango_rt->start_kernel(kscale,
            *argsKSCALE);
        e1->wait();
        auto e3=mango_rt->start_kernel(ksmooth,
            *argsKSMOOTH);
        e3->wait();
        b3->event->wait();
    }
};
```

```

    b3->read(out);
  }
};

```

4.3. Low Level Runtime Access Support

The set of accelerators in MANGO will be interconnected through a QoS-aware interconnect and spread over an infrastructure of FPGAs physically (pin-to-pin) interconnected. The system will be connected to high-end servers through PCIe and Gigabit connections. All the communication variety must be uniformly accessed by the resource manager. Indeed, the different communication interfaces must be transparent. A low level runtime library has been developed to provide transparent access to the heterogeneous components. The runtime library provides efficient means for the key functional processes required by the resource manager, such as: 1. booting the system and the accelerators; 2. querying about current utilization of resources and other structural information required, such as power consumption or temperature; 3. enabling the system's configuration, mostly for the proper configuration of QoS parameters of the interconnect; 4. reading and writing memory distributed over the heterogeneous system; 5. spawning tasks into the accelerators; 6. providing means of synchronization between tasks and main applications running on the high-end servers.

5. RTL Power Monitoring Framework in MANGO

Power represents a key metric to be optimized at both design- and run-time metric to deliver a successful computing architecture. Traditionally, design-time power methodologies focus on power optimization during the design stages. However, modern applications usually traverse several execution phases having different computing requirements, thus possibly allowing for non-negligible power optimizations at run-time. The online power monitoring constitutes a de-facto solution to deliver run-time power estimates of the platform that can be used to implement different power-based run-time optimizations, e.g., power-performance resource allocation and thermal-control. Despite their key role, the state of the art in online power monitoring methodologies leverages the so called performance counter based power models to extract

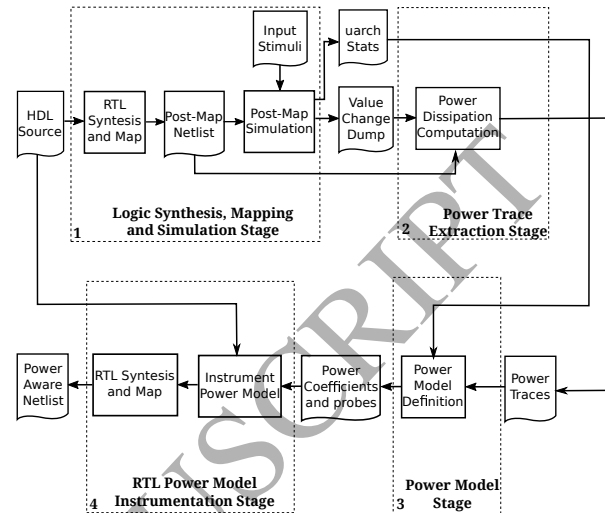


Figure 22: Overview of the MANGO RtPwrMon toolchain. The power traces from the post-map logic simulation and the micro-architectural statistics are used to identify the power model of the target. Last, the obtained power model is integrated in the target's RTL.

an online power prediction. It does so using the performance counters available in the target architecture with a net performance overhead due to the CPU time devoted to the model update. The MANGO Online Power Monitoring System delivers a fresh solution to the online power monitoring problem by extracting the power model of the target architecture from the RTL description without resorting to any, already available, performance counter. The MANGO Runtime Power Monitoring System (RtPwrMon) is made of two parts: the simulation framework and the power modeling module. The simulation framework takes the netlist of the target architecture as input and outputs a time-based power trace of its post-synthesis simulation coupled with a time-based set of sampled architectural statistics. The power modeling module takes the time-based power trace and the sampled architectural statistics (toggle counts) to identify the coefficients of the linear model that minimizes the prediction error on the power-trace. An equivalent RTL description of the identified linear model is instrumented in the RTL of the target architecture to enable the online power monitoring capability without having CPU performance overhead.

5.1. Power Monitoring Framework

The final goal of the MANGO RtPwrMon is a novel solution to automatically instrument the RTL of a generic design to enable software-level power-aware optimization methodologies. The proposed solution overcomes the two limitations of the current performance counter based power models: i) hindrance in the execution of the platform tasks by subtracting precious CPU cycles to update the power estimate and ii) leveraging on the available performance counters that are not primarily intended for power modeling, thus possibly inducing inaccurate results. Figure 22 highlights the four-stage MANGO RtPwrMon simulation flow [24]. It is general enough to allow the power model creation and RTL instrumentation of any architecture for which the HDL description is available, thus making the methodology suitable for both peripherals or hardware accelerators and for general purpose CPUs. Starting from the HDL description of the target architecture, the Logic Synthesis, Mapping and Simulation Stage employ Vivado 2017.1 to synthesize, map and simulate the design and, ultimately, to extract the Value Change Dump (VCD) information and the architectural statistics that are later used for the power model estimation. The toggle counts of the primary input and output signals for each module in the target architecture are collected, in addition to the micro-architectural statistics (see uarch Stats in Figure 1). The extensive literature on performance counter power models demonstrate the effectiveness of relating the toggle count of a signal to the power consumption. In particular, we followed such principle to analyze the correlation between the power trace and any possible primary input and output signal for each module in the target architecture. The VCD file is then pre-processed and fed into the Vivado Report Power tool to obtain the time-based power trace (see Power Trace Extraction Stage in Figure 1). Indeed, starting from the Switching Activity Interchange Format (SAIF) file and the target netlist, Vivado Report Power provides the average power consumption for the design at hand. To this extent, a specific VCD to SAIF converter has been developed to interface Vivado Report Power by iterating on the single VCD file to produce multiple SAIF files. To balance the computational effort and the accuracy of the power trace each SAIF contains data to compute power on a time window set to 100 clock cycles. The final power trace for each module in the architectural hierarchy is ob-

tained as a series of power samples, each one integrating the power consumption of the target architecture within the time window. The Power Model Stage takes the power traces and the micro-architectural statistics to deliver the power model for the target architecture. The power model is made of a set of coefficients and the related RTL signals of the target architecture for which we collect the toggle count to feed the model. The selected RTL signals are a set of primary inputs and/or outputs of one or more modules within the design hierarchy of the target architecture, for which the Power Model Stage estimates a good fit with the real power consumption of the target.

6. Thermal and Cooling Innovations in MANGO

MANGO will extend the experience acquired in the latest research on advanced compact modelling for liquid-cooling monitoring [25] to explore the time constants of thermal and energy control knobs to develop next-generation cooling technologies for HPC systems. In particular, we will explore the use of a novel passive thermosyphon (gravity-driven) cooling technology that will attempt to include multiple *parallel heat sources at multiple elevations* to eliminate energy consumption [26]. Thus, in MANGO we will carry out for the first time in an heterogeneous HPC system a preliminary evaluation of the benefits and drawbacks of a gravity-driven two-phase liquid cooling prototype, developed and measured in the facilities of EPFL. The objective will be to proof the possibility of achieving radically low Power Usage Effectiveness (PUE) values for heterogeneous HPC systems, contributing at improving the efficiency of next-generation HPC workloads by working on the PPP axis of MANGO: power, performance and predictability.

Moreover, apart from those three metrics, thermal management is a major challenge that needs to be tackled jointly with cooling control to ensure reliability and maximize energy efficiency. Therefore, as a complimentary measure to the design of efficient cooling systems, one of the goals of the project is the development of thermal-, power- and performance-aware allocation strategies, both at the global and the local level, able to exploit the new architectures and the heterogeneity of the MANGO platform for the particular target applications. In this sense, the MANGO project will characterize the applications, to understand their constraints, and propose novel thermal-,

power-, and performance-aware run-time resource management strategies. As a first case-study, within MANGO the research undertaken has focused first on the HEVC *video transcoding application*.

6.1. Development of a framework for thermal, power and performance characterisation

We envision two options for the thermal, power and performance characterisation that needs to be undertaken within the MANGO project. Even though the final demonstrator infrastructure will allow automated and real-time monitoring, as will be explained in the following subsections, there is a need for off-line profiling and characterisation of the applications within the heterogeneous MANGO resources. In this sense, we have followed two approaches:

1. **The direct measurement of applications running on the hardware.** This implies running the applications on the target platform while collecting performance counters, power, and temperature values. For x86 architectures, such as the ones of GNs, we can use Intel Running Average Power Limit (RAPL) [27] to estimate the power consumption of the CPU (cores and package). Temperature sensors are usually available as an average for the whole CPU. Thus, to be able to obtain a finer granularity (i.e., to obtain the temperature gradients of the chip), we leverage the usage of the 3D-ICE simulator [28].
2. **Using a full simulation framework.** In this sense, we propose running the MANGO applications first on the Gem5 architectural simulator [29], to obtain performance metrics. Those metrics can then be plugged into McPAT [30] to obtain power traces, and finally into 3D-ICE to compute temperature floorplans.

It must be taken into account that, to profile separately the various kernels of the applications, a small effort on code instrumentation needs to be performed. This effort is needed to separate the kernels. However, this does not add an overhead to the application programmer, as the separation and characterisation is performed per-kernel.

The proposed setup is currently being used for the profiling and characterisation of the MANGO applications.

In particular, characterisation of the x86 GNs is being performed via direct measurements (performance counters, RAPL and 3D-ICE), whereas currently the ARM cores are being profiled via simulation (Gem5, McPAT and 3D-ICE). We envision also the incorporation of Gem5 models of other accelerators of the MANGO platform.

6.2. Challenges and constraints of the video transcoding application

The undeniable complexity of the HEVC encoders, together with the increase of video streaming users, poses an important challenge for power- and thermal-aware resource allocation and management of these applications when running on MPSoCs. Because of the lack of a HEVC encoder that is able to transcode on real-time (i.e., to achieve an encoding frame rate of 30 fps), current solutions in the area are mostly focused on the optimization of one or several blocks of the HEVC encoding algorithm to reduce processing time per frame [31, 32]. However, to address the challenge of power and thermal management in HEVC transcoding applications, application-level configuration and system-level knobs need to be jointly integrated on top of algorithmic optimizations. Few works jointly consider temperature constraints as well as encoding efficiency of next generation video encoders [33]. Nonetheless, none of these works consider power consumption as a different parameter from temperature. Moreover, power and thermal management of HEVC has not been addressed when multiple streams are running at the same time on a multicore platform.

Each block in HEVC encoder contains several parameters to configure the encoder (i.e., configuration knobs). A few of these configuration knobs have large impacts on the encoding efficiency, power consumption, temperature and processing time, including search area, prediction mode, size of Group of Picture (GOP), Quantization Parameter (QP), and Coding Unit (CU) size. All these knobs can be dynamically tuned frame-by-frame, except for the GOP size that can only be changed every several frames.

Finally, apart from inherent and exclusive features of each video type, such as frame rate, frame resolution, bit depth, etc., the contents of a video also play a major role in the obtained performance (encoding time per frame), quality (peak signal-to-noise ratio, PSNR, measured in dB), compression (bitrate, measured in bits per second,

bps), power consumption, and peak temperature, resulting from a specific encoding configuration.

The frame-by-frame power and thermal management is well motivated by such variations. As a consequence, the encoding configuration and the CPU frequency must be dynamically adjusted to provide the best possible outcomes. The great number of different combinations of configuration knobs, in addition to sudden content variations within a video and substantial differences between different videos require a more generic solution than that proposed by previous works.

Despite the sophistication of managing power and temperature in MPSoCs for HEVC, ML-based methods, and among them, reinforcement learning algorithms, are promising solutions, as they cope with environment-dependant problems using dynamic optimization programming.

6.3. Leveraging the MANGO HNs for the transcoding application

The first step towards exploiting the HNs to increase the efficiency of the MANGO applications is understanding which kernels would benefit the most from a hardware implementation of from acceleration. For this purpose, we have performed a profiling of the video transcoding application, obtaining its task call graph to understand which function and kernels require higher computational effort. This exercise has been performed for both the x86 and the ARM cores of MANGO, obtaining similar relative results.

Figure 23 summarises the percentage of time spent in each functions for a particular encoding configuration and video input of the transcoding application. As can be observed, the interpolation and DCT phases are the ones that would benefit the most from a hardware implementations and thus, represent good candidates for parallelization on the MANGO HNs.

6.4. Extending the approach to other MANGO applications

As opposed to transcoding, medical imaging has strict deadlines for the processing of each frame. When a deadline cannot be met, it is better to drop the frame. Because of the nature of this problem, we envision tackling the thermal and power-aware resource allocation problem by

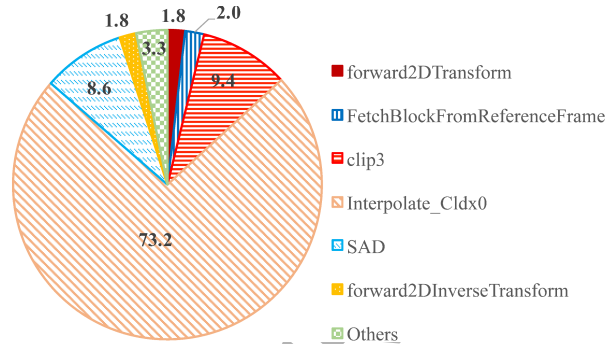


Figure 23: Percentage of time used by the different phases of the transcoding application when encoding a frame in a GN

using Staged Multi-Armed Bandits (MABs) [34]. MABs are generally used in solving decentralized sequential decision making problems involving multiple learners. We believe that the problem of scheduling multiple streams for the MANGO bio-medical application can be seen as a staged decision problem in which the performance obtained for various resource allocations is unknown a priori but learned over time. Unlike other online learning methods such as standard multi-armed bandits and reinforcement learning, in our tentative formulation the outcome of each scheduling action depends on a sequence of previous scheduling decisions and feedbacks that are taken at a certain stage of time.

7. The MANGO Platform Roadmap

The MANGO strategy for building an effective largescale emulation platform will be articulated in three phases.

Phase 1 – Stand-alone single-board emulator. The research activities involving architecture exploration initially relies on current available hardware made of a stand-alone emulation platform based on FPGA devices and a general purpose node. The standalone emulator is based on a modular and scalable approach, with several FPGAs being assembled on dedicated daughter modules plugged on a common motherboard. The motherboard gives complete access to all available I/Os of the FPGA, leaving maximum freedom regarding the FPGA interconnection structure, which will allow to define the HN interconnect.

The proFPGA quad V7 system [35], provided by PRO DESIGN as a standalone emulation platform, is used. The board is equipped with three Xilinx Virtex 7 XCV2000T FPGA modules and one Zynq module, containing a dual core ARM processor as well as a reconfigurable hardware fabric to prototype external subsystems, handling up to 48 M ASIC gates alone in one board. Several proFPGA systems are interconnected enabling the full HN infrastructure to be implemented. Due to the fact that multiple proFPGA quad or duo systems can be stacked or connected together, scalability is ensured. The highspeed boards together with the specific high speed connectors allow a maximum point to point speed of up to 1.8 Gbps over the standard FPGA I/O and up to 12.5 Gbps over the MGT of the FPGA.

Phase 2 – From FPGA stand-alone board to a dedicated chassis. A new board for HPC will be implemented complying with the physical constraints of HPC and datacenter racks, considering as well requirements for cooling and power supply researched within the project. The board will be extended to deliver further number of daughter boards. Pin-to-pin connectivity between FPGAs will allow expandability and scalability. This enables MANGO to explore future chip configurations in a predictable and accurate manner. Daughter boards will be extensible and open to new developments, particularly to new 64-bit ARM cores or even more advanced solutions like the hybrid Xeon E5+FPGA chip recently announced by Intel. In this phase, the HN interconnect will be applied to the set of HN nodes (the board) developed. It will embrace connectivity at the board level, between ARM and FPGA modules, inside the FPGA modules (within the accelerators and RISC processors implemented), and between the boards. This means a single and unified interconnect will be designed for the overall HN infrastructure (made of 64 nodes).

Phase 3 – Rack assembly. As a final phase, the complete rack will be implemented and populated of GNs and HN nodes. The system will enable a large-scale platform used to reproduce in near real-time the behavior of the MANGO manycore architecture. The full platform will consist of a rack collecting up to 16 blades equipped with high-end CPUs, e.g. Intel Xeon chips, and GPUs, mounted on the motherboard, as well as 64 HN nodes. A custom back-

plane will provide connectivity across the blades, both through standard bridges and using pin-to-pin connections across the FPGA chips, effectively providing a single large-scale reconfigurable hardware fabric used to emulate the fine-grained accelerator tiles envisioned in the MANGO architecture. The inter-FPGA pin-to-pin backplane interconnection will be reconfigurable on-field, providing a large degree of flexibility for the emulation of the on-chip network interconnect.

8. Conclusions

The three-years MANGO project, which started in October 2015, aims at addressing power, performance and predictability in HPC systems. To this end, it leverages customization and deep heterogeneity to adapt the available computing resource. In this paper, we presented the main approach and architectural solution, the application scenarios considered, and a more in-depth view of the software stack. We also discussed initial results on processing element characterization, memory management, RTL power monitoring and a characterization of the HEVC transcoding application from the point of view of thermal management.

Acknowledgements

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 671668.

References

- [1] W. Ziegler, R. D'ippolito, M. D'Auria, J. Berends, M. Nelissen, and R. Diaz, "Implementing a "one-stop-shop" providing smes with integrated hpc simulation resources using fortissimo resources," in *eChallenges e-2014, 2014 Conference*. IEEE, 2014, pp. 1–11.
- [2] J. Flich, G. Agosta, P. Ampletzer, D. A. Alonso, C. Brandolese, A. Cilardo, W. Fornaciari, Y. Hoorneborg, M. Kovač, B. Maitre, G. Massari, H. Mlinarić, E. Papastefanakis, F. Roudet, R. Tornero, and D. Zoni, "Enabling hpc for qos-sensitive applications: The mango approach," in *2016 Design*,

- Automation Test in Europe Conference Exhibition (DATE)*, March 2016, pp. 702–707.
- [3] J. Flich, G. Agosta, P. Ampletzer, D. A. Alonso, C. Brandolese, E. Cappe, A. Cilaro, L. Dragic, A. Dray, A. Duspara, W. Fornaciari, G. Guillaume, Y. Hoornenborg, A. Iranfar, M. Kovac, S. Libutti, B. Maitre, J. M. Martnez, G. Massari, H. Mlinaric, E. Papastefanakis, T. Picornell, I. Piljic, A. Pupykina, F. Reghenzani, I. Staub, R. Tornero, M. Zapater, and D. Zoni, “Mango: Exploring many-core architectures for next-generation hpc systems,” in *2017 Euromicro Conference on Digital System Design (DSD)*, Aug 2017, pp. 478–485. [Online]. Available: <https://doi.org/10.1109/DSD.2017.51>
- [4] European Technology Platform For HPC, “ETP4HPC Strategic Research Agenda: Achieving HPC leadership in Europe,” <http://www.etp4hpc.eu/strategy/strategic-research-agenda/>, 2013.
- [5] CISCO, “The Zettabyte Era – Trends and Analysis,” <http://www.cisco.com>, Nov 2017.
- [6] M. Kovač, “E-Health Demystified: An E-Government Showcase,” *Computer*, vol. 47, no. 10, pp. 34–42, Oct 2014.
- [7] M. Kovač and N. Ranganathan, “Vlsi circuit structure for implementing jpeg image compression standard,” Aug. 1997, US Patent 5,659,362.
- [8] PHILIPS, “About Health Suite.” [Online]. Available: <https://www.philips.co.uk/healthcare/innovation/about-health-suite>
- [9] J. Pawasauskas, “Volume Visualization With Ray Casting,” <http://web.cs.wpi.edu/~matt/courses/cs563/talks/powwie/p1/ray-cast.html>, Feb 1997.
- [10] B. Preim and C. P. Botha, *Visual Computing for Medicine: Theory, Algorithms, and Applications*. Newnes, 2013.
- [11] R. Gallager, “Low-density parity-check codes,” *IRE Transactions on information theory*, vol. 8, no. 1, pp. 21–28, 1962.
- [12] D. J. MacKay and R. M. Neal, “Near shannon limit performance of low density parity check codes,” *Electronics letters*, vol. 32, no. 18, p. 1645, 1996.
- [13] S. Muller, M. Schreger, M. Kabutz, M. Alles, F. Kienle, and N. Wehn, “A novel ldpc decoder for dvb-s2 ip,” in *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE’09*. IEEE, 2009, pp. 1308–1313.
- [14] A. Cilaro, M. Gagliardi, and C. Donnarumma, *A Configurable Shared Scratchpad Memory for GPU-like Processors*. Springer International Publishing, 2017, pp. 3–14.
- [15] A. Cilaro, M. Gagliardi, and D. Passaretti, *NoC-Based Thread Synchronization in a Custom Many-core System*. Springer International Publishing, 2018, pp. 673–682.
- [16] C. Lattner and V. Adve, “Llvm: A compilation framework for lifelong program analysis & transformation,” in *Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization*. IEEE Computer Society, 2004, p. 75.
- [17] L.-N. Pouchet, “Polybench: The polyhedral benchmark suite,” URL: <http://www.cs.ucla.edu/pouchet/software/polybench>, 2012.
- [18] Khronos Group, “The Open Standard for Parallel Programming of Heterogeneous Systems,” <https://www.khronos.org/opencl/>, (retr. Jul 2015).
- [19] G. Agosta, A. Barengi, A. Di Federico, and G. Pelosi, “Opencl performance portability for general-purpose computation on graphics processor units: An exploration on cryptographic primitives,” *Concurr. Comput. : Pract. Exper.*, vol. 27, no. 14, pp. 3633–3660, Sep. 2015. [Online]. Available: <http://dx.doi.org/10.1002/cpe.3358>
- [20] G. Agosta, A. Barengi, G. Pelosi, and M. Scandale, “Towards Transparently Tackling Functionality and Performance Issues across Different OpenCL Platforms,” in *2nd Int’l*

- Symp. on Computing and Networking (CANDAR)*, Dec 2014, pp. 130–136. [Online]. Available: <https://doi.org/10.1109/CANDAR.2014.53>
- [21] G. Agosta, W. Fornaciari, G. Massari, A. Pupykina, F. Reghenzani, and M. Zanella, “Managing heterogeneous resources in hpc systems,” in *Proceedings of the 9th Workshop and 7th Workshop on Parallel Programming and RunTime Management Techniques for Manycore Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms*, ser. PARMA-DITAM '18. New York, NY, USA: ACM, 2018, pp. 7–12. [Online]. Available: <http://doi.acm.org/10.1145/3183767.3183769>
- [22] P. Bellasi, G. Massari, and W. Fornaciari, “Effective runtime resource management using linux control groups with the barbequertrm framework,” *ACM Trans. Embed. Comput. Syst.*, vol. 14, no. 2, pp. 39:1–39:17, Mar. 2015. [Online]. Available: <http://doi.acm.org/10.1145/2658990>
- [23] A. Pupykina and G. Agosta, “Optimizing memory management in deeply heterogeneous hpc accelerators,” in *2017 46th International Conference on Parallel Processing Workshops (ICPPW)*, Aug 2017, pp. 291–300. [Online]. Available: <https://doi.org/10.1109/ICPPW.2017.49>
- [24] D. Zoni, L. Cremona, and W. Fornaciari, “Powerprobe: Run-time power modeling through automatic rtl instrumentation,” in *Design, Automation Test in Europe Conference Exhibition (DATE), 2018*, March 2018, pp. 749–754. [Online]. Available: <https://doi.org/10.23919/DATE.2018.8342106>
- [25] A. Sridhar, A. Vincenzi, M. Ruggiero, and D. Atienza, “Neural network-based thermal simulation of integrated circuits on gpus,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 1, pp. 23–36, 2012.
- [26] N. Lamaison, C. L. Ong, J. B. Marcinichen, and J. R. Thome, “Two-phase mini-thermosyphon electronics cooling: Dynamic modeling, experimental validation and application to 2u servers,” *Applied Thermal Engineering*, vol. 110, pp. 481 – 494, 2017.
- [27] Intel Corp., “Intel 64 and IA-32 Architectures Software Developer Manual,” 2012.
- [28] A. Sridhar, A. Vincenzi, M. Ruggiero, T. Brunschwiler, and D. Atienza, “3d-ice: Fast compact transient thermal modeling for 3d ics with inter-tier liquid cooling,” in *Proceedings of the International Conference on Computer-Aided Design*. IEEE Press, 2010, pp. 463–470.
- [29] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sadashti *et al.*, “The gem5 simulator,” *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.
- [30] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, “Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures,” in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM Int. Symp. on*. IEEE, 2009, pp. 469–480.
- [31] G. Correa, P. Assuncao, L. Agostini, and L. A. S. Cruz, “Complexity scalability for real-time hevc encoders,” *Journal of Real-Time Image Processing*, vol. 12, no. 1, pp. 107–122, 2016.
- [32] M. Shafique, M. U. K. Khan, and J. Henkel, “Power efficient and workload balanced tiling for parallelized high efficiency video coding,” in *Image Processing (ICIP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 1253–1257.
- [33] D. Palomino, M. Shafique, A. Susin, and J. Henkel, “Tone: Adaptive temperature optimization for the next generation video encoders,” in *Proceedings of the 2014 international symposium on Low power electronics and design*. ACM, 2014, pp. 33–38.
- [34] K. Kanoun, C. Tekin, D. Atienza, and M. Van Der Schaar, “Big-data streaming applications scheduling based on staged multi-armed bandits,” *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3591–3605, 2016.

- [35] PRO DESIGN, “proFPGA Products quadV7system overview.” [Online]. Available: http://www.prodesign-europe.com/proFPGA_Products_quadV7system_overview.html

ACCEPTED MANUSCRIPT

Biography of Michele Zanella

Michele Zanella received the M.Sc Degree with Laude in Computer Science and Engineering in April 2017 at Politecnico di Milano with the thesis “Energy-aware Run-Time Management of Distributed Mobile Devices”. He joined the Heap Lab at DEIB in may 2017 and he is continuing to work with as a PhD student.

His work mainly focuses on the run-time resource management in distributed mobile computing and his research interests include run-time resource management, service-based smart environment architectures and high performance mobile applications. He is also passionate about psychology, music, as a organ player and choir conductor, and aviation, as a private pilot.