Title

OpenPiton4HPC: Optimizing OpenPiton Toward High-Performance Manycores

Journal

IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 14(3)

ISSN

2156-3357

Authors

Leyva, Neiel

Monemi, Alireza

Oliete-Escuín, Noelia

et al.

Publication Date

2024-09-01

DOI

10.1109/jetcas.2024.3428929

Peer reviewed

OpenPiton4HPC: Optimizing OpenPiton Towards High Performance Manycores

Neiel Leyva, Alireza Monemi, Noelia Oliete-Escuín,
Guillem López-Paradís, Xabier Abancens, Jonathan Balkind, Enrique Vallejo, Miquel Moretó, and Lluc Alvarez

***Abstract*—In recent years, numerous multicore RISC-V platforms have emerged. Development frameworks such as OpenPiton are employed in designs that aim to scale to a large number of cores. While OpenPiton presents a large flexibility, supporting different requirements and processing cores, some of its design decisions result in designs that are not optimized for High-Performance Computing (HPC) requirements.**

This work presents OpenPiton4HPC, an extension and optimization of OpenPiton for high-performance manycores. The key contributions are enabling multiple memory controllers, supporting router bypassing and NoC concentration, adding support for configurable cache sizes and cache block sizes, and allowing configurable bus widths in the NoC and in the cache SRAMs. On a 64-core manycore architecture, these new features and optimizations provide a geometric mean speedup of 7.2x compared to the OpenPiton baseline.

***Index Terms*—Many-core, Network-On-Chip, optimization, OpenPiton, RISC-V**

I. Introduction

Multicore processors have dominated the landscape of high performance computer architecture for many years. Industry led the way during the early days of the multicore era, with numerous vendors designing and fabricating multicore processors, while academia heavily studied multicores using software simulators and modeling tools. This trend has changed in recent years thanks to the emergence of RISC-V, which has drastically facilitated designing cores both in industry and academia. In addition, multiple tools and frameworks have been developed within the RISC-V ecosystem, which allows designing multicore processors in an easy and practical manner.

NoCs are a key component of multicore processors. The purpose of the NoC is to interconnect multiple cores and the memory hierarchy, allowing efficient data transfer between them. Within the RISC-V ecosystem, NoCs are employed in large-scale multicores or manycores such as OpenPiton [1], BlackParrot [2], and the ESP open SoC platform [3], while simpler crossbar communication is utilized in platforms such as lowRISC [4] and PULP [5], which primarily target applications for the Internet-of-Things (IoT) and put special emphasis on low power consumption.

N. Leyva, A. Monemi, N. Oliete-Escuín, G. López-Paradís and M. Moretó are with the Barcelona Supercomputing Center and the Universitat Politecnica de Catalunya. E-mail: neiel.leyva@bsc.es, alireza.monemi@bsc.es, noelia.oliete@bsc.es, guillem.lopez@bsc.es, miquel.moreto@bsc.es

X. Abancens and L. Alvarez are with the Barcelona Supercomputing Center. E-mail: xabier.abancens@bsc.es, lluc.alvarez@bsc.es

J. Balkind is with UC Santa Barbara. E-mail: jbalkind@ucsb.edu

E. Vallejo is with the Universidad de Cantabria and currently a research visitor at LIRMM (CNRS). E-mail: enrique.vallejo@unican.es

Manuscript received March 4, 2024.

OpenPiton has received a lot of attention as a development platform for creating manycore processors due to the multiple benefits it provides. Among other benefits, OpenPiton is open source, easy to use, highly scalable and configurable, it provides a mature tool ecosystem with Linux support, it is easily synthesizable to FPGA and ASIC, and it provides a large test suite.

However, the NoC and the memory hierarchy of OpenPiton do not include certain features and characteristics typically found in high-performance manycores. This is because OpenPiton focuses on a general userbase, and the optimizations for high-performance cache hierarchies and NoCs can come at a non-negligible cost in terms of area or power consumption which may not be affordable in computing domains with more restrictive constraints. Yet, adding high-performance features and optimizations in a configurable manner can help expanding the horizon of OpenPiton without sacrificing its flexibility.

This paper extends a previously published paper [6], which presented an early version of OpenPiton4HPC. OpenPiton4HPC presents a set of extensions and optimizations to the NoC and the memory hierarchy of OpenPiton specifically aimed at improving the performance of large-scale multicore and manycore architectures. The key contributions of the previously published paper [6] were:

- The capability to including multiple memory controllers in the chip to increase the memory bandwidth of the system.
- Router bypassing and NoC concentration features to reduce the latency of core communications and data transfers inside the NoC.
- Support for configurable cache sizes and cache block sizes in the cache hierarchy to improve its efficiency.

Meanwhile, the key contributions of this paper extension are:

- We add support for configurable cache block sizes in the L1 instruction cache.
- We increase the RAM bus width of all the caches of the hierarchy to reduce the latency of cache accesses. This allows for a single-cycle cache access, but it increases the serialization latency in the NoC for data block transfer.
- We increase the width of the NoC buses to minimize the serialization latency and improve the communication bandwidth.
- We extend the evaluation with an area analysis of an FPGA prototype of the proposed optimized design, an analysis of the impact of memory controller placement and an analysis of the improvements in throughput.

Compared to the OpenPiton baseline, the combination of all the features and optimizations provides a geometric mean speedup of

7.2x (28.5% over the previous work [6]) on a 64-core manycore architecture, discussed in Section V-H1. In addition, this work preserves the flexibility of the OpenPiton framework by completely parameterizing all the proposed optimizations, in such a way that users can configure the different architectural parameters according to the performance goals or power and area constraints of their designs. Similarly, this work also preserves the open spirit of OpenPiton and its notorious involvement in the research community, so all the optimizations and new features presented in this work are open source [7].

The rest of the paper is organized as follows. Section II provides the required background and the motivation for the work. Section III presents the contributions in OpenPiton4HPC. Section IV presents the evaluation methodology and Section V presents and analyzes the results, followed by a discussion in Section VI. Finally, Section VII presents some related work and Section VIII concludes the paper.

II. Background and Motivation

Originally, OpenPiton was developed for SPARC v9 architectures (OpenSPARC T1); however, in recent projects, the platform has been adapted to work with RISC-V architectures [8]–[10].

The OpenPiton architecture consists of a single chipset and multiple tiles. The chipset handles tile-to-peripheral communication, featuring several modules like bootrom, memory controller, and UART. The tiles construct the multicore mesh, connecting tiles via three physical NoCs. Each tile contains the cache hierarchy (private and shared cache levels), the three NoC routers, and the core. The tiles can feature cores with different architectures, including RISC-V 32-bit, RISC-V 64-bit, x86, and SPARCv9 [11].

A. OpenPiton and RISC-V Architectures

OpenPiton is compatible with different architectures by means of the Transaction-Response Interface (TRI) [11]. Regardless of their architecture, cores with private caches must have an instruction cache, a write-through data cache, and (for application-class cores) an MMU. These modules should be connected directly to the TRI. This method is used in the integration of CVA6 into the OpenPiton framework [9], which currently is the most commonly used OpenPiton-based platform. As an alternative, cores without private cache levels nor MMU can also be directly connected to the TRI.

B. OpenPiton Cache Hierarchy

OpenPiton's cache hierarchy consists of a last-level cache, called L2 cache, a private cache level per tile, called L1.5 cache.

Each tile contains a slice of the shared L2 cache and a directory controller for the P-Mesh directory-based MESI coherence protocol. The default configuration for the L2 cache is 64KB per slice, 4-way set-associative, 8 MSHR and 64B cache blocks. The cache RAMs are arranged in 4 sublines of 16B each, and 64B cache requests require sequential accesses to the 4 RAMs.

Each tile also includes a write-back L1.5 private cache. The default L1.5 cache configuration is 8KB capacity, 2 MSHR and 4-way set-associative with 16B cache blocks. It is connected to the L2 cache through 3 physical NoC routers. The L1.5 cache implements TRI, managing core requests for data, instructions, and atomic operations. Notably, the L1.5 cache does not cache instruction memory blocks; these are forwarded to the L2 cache.

C. OpenPiton Memory Controller

OpenPiton implements a memory controller inside the chipset module. The platform can scale the number of tile modules, but not the number of chipsets nor memory controllers. Being limited to a single memory controller can cause bottlenecks, especially in medium to large systems. In simulation, OpenPiton employs an emulated simplistic memory controller in C code, where all memory requests (read and write) are served with a fixed latency of a single clock cycle. When the system is connected to a real memory controller (such as in FPGA implementation), the port used for the emulated memory controller is directly connected to one physical channel of the memory controller.

By default, the chipset links to the west port of tile 0, which is located in the northwest mesh corner. Consequently, memory requests must traverse the entire NoC to access the memory controller connected to tile 0. This setup can pose drawbacks in large systems, potentially leading to large memory access latency and NoC congestion.

D. OpenPiton NoC Routers

OpenPiton tiles are interconnected using three NoCs in a 2D mesh topology. Pairs of adjacent routers are interconnected using two 8-Byte uni-directional links. These links use credit-based flow control, and packets are routed using dimension-ordered wormhole routing. OpenPiton routers have a single-cycle latency when packets are moved in the same direction and a two-cycle pipeline latency when the packet involves any turn.

OpenPiton NoC packets implement a header flit and they require data serialization. Three flits are injected into the NoC to transfer a 16-Byte block of data from the L2 shared memory to the private caches. For 32-Byte requests from the instruction caches, five flits are required.

E. OpenPiton and HPC Requirements

Compared with high-performance manycores, OpenPiton has performance limitations that can impact its ability to execute computationally intensive tasks. Some of the limitations are:

Memory hierarchy: OpenPiton processors face memory constraints due to memory controller and cache settings. The single memory controller can bottleneck data flow from the main memory when all tiles handle large amounts of data. Enabling multiple memory controllers can enhance memory bandwidth per tile and address this limitation. Enlarging the cache sizes and the block sizes can also greatly improve performance.

NoC latency: This communication delay can severely impact performance, necessitating low-latency NoC design for large systems. Widening the NoC buses to reduce the number of flits per packet can drastically reduce the NoC latency. In addition, employing multi-hop bypass and increasing NoC concentration can help decreasing data transmission hops and lowering the average communication latency.

Data transactions latency: Efficient data movement is essential for shared memory systems. Updating data and bringing it closer to the cores with low latency is key to avoid bottlenecks in the cache hierarchy and to reduce the memory access time of the cores. Thus, reducing the latency of the memory block transactions within the cache itself can provide important performance benefits.

Improving these features is critical to increasing the performance of OpenPiton in large-scale multicore architectures.

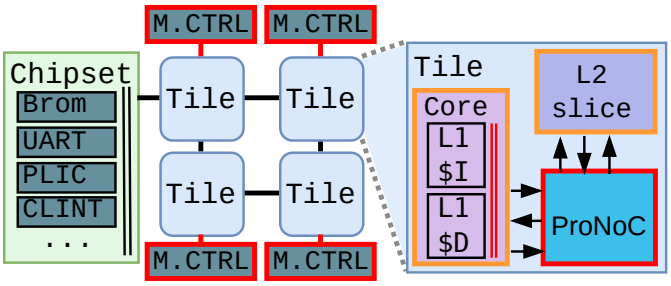


Fig. 1. OpenPiton architecture upgraded with new features. New modules are marked in red and optimized modules are marked in orange.

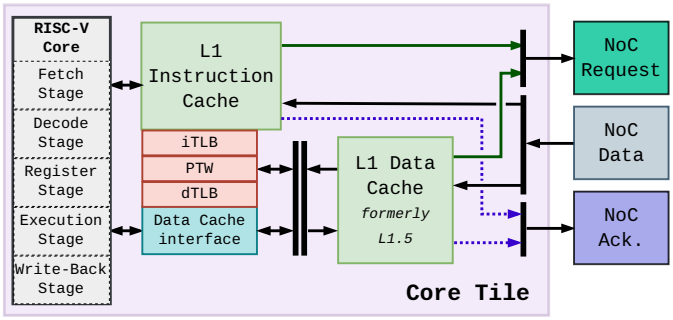


Fig. 2. Architecture of the Core Tile employing a modified version of TRI with one private data cache and direct connection of the instruction cache to the routers.

III. NoC and Cache Hierarchy Optimizations For High Performance

This section presents the modifications and optimizations made in OpenPiton. Our aim is to customize the design for high-performance multicores. Figure 1 visually summarizes the key improvements, showcasing the original OpenPiton modules alongside highlighted additions and modifications. New modules are marked in red and optimized ones are marked in orange. The next subsections explain the optimizations in detail.

A. Adapted Core Tile Using TRI

Core Tile is a new adaptability proposal for application-class cores within OpenPiton. Core Tile integrates into OpenPiton using the TRI interface. The main objective of the Core Tile is to reduce memory access time and area.

Figure 2 depicts the architecture of the Core Tile. Note that the Core Tile has a single L1 data cache, instead of the two private data cache levels (L1 and L1.5) present in the original OpenPiton+CVA6 platform [9]. We remove the L1 write-through data cache and we use the L1.5 cache from OpenPiton directly as the L1 data cache. This is done by directly connecting the core to the L1.5 via TRI.

1) Core: The core tile incorporates a core called DVINO [12]. This core is a 6-stage single-issue, in-order architecture, alongside a two-lane vector processor unit. The core implements the 64-bit RV64G scalar RISC-V ISA v2.2 and privileged ISA v1.11.

2) Instruction Cache: The core tile features a 16KB Virtually Indexed Physically Tagged (VIPT) instruction cache with a two-cycle hit latency. The address translation process is managed internally within the instruction cache. It has a direct link to the NoC via traffic arbitrators, avoiding passing through the L1 data cache. This reduces cycles and avoids extra traffic in the data cache. Additionally, it can accept block invalidations coming from the coherence protocol.

3) MMU: The core tile includes an SV39 MMU compatible with RISC-V architectures. It comprises two 8-entry Translation Lookaside Buffers (TLBs) and one Page Table Walker (PTW). The updates to the dirty and access bits are handled by hardware.

4) Data Cache Interface: This module is the primary glue logic responsible for directly connecting the core's Load-Store Unit (LSU) to the TRI. The data cache interface facilitates communication with the MMU; address translation is handled before sending a request to the L1 data cache. Additionally, this module handles exceptions generated by the MMU and exceptions related to misaligned addresses. The PTW is connected to the data cache to request Page Table Entries (PTE) and update dirty and access bits. An arbiter handles requests from the PTW and the core to the TRI.

5) NoC interface: The modified interface allows direct instruction cache access to the NoC. An arbitrator in the *NoC request* encoder prioritizes instruction cache requests over data cache requests. Another arbiter is added in the *NoC data* decoder to manage deliveries between instruction and data caches. These deliveries can consist of memory data blocks or requests, such as invalidations. A third arbiter is added in the *NoC Ack* encoder to control the access of requests from private caches, both data and instructions. This is because the instruction cache, after an invalidation, needs to notify that the invalidation has been received.

B. Enabling Multiple Memory Controllers

We propose adding the capability in OpenPiton to configure the number of memory controllers. In state-of-the-art large-scale multicores the memory wall can pose a challenge if memory bandwidth is not scaled proportionally to the number of tiles. Consequently, to prevent the saturation of memory bandwidth, high-performance multicores typically incorporate multiple memory controllers per chip.

We modify OpenPiton to allow a parameterizable number of emulated memory controllers that are directly connected to the edge tiles of the mesh, connected to the spare ports of the edge NoC routers. By default, the memory controllers are automatically distributed equidistantly across two opposite edges of the mesh (east, and west) by a generation script. Such design typically simplifies access to the physical DIMMS in both sides of the chip. However, any alternative placement can be manually specified by the designer. Furthermore, each L2 module is modified to route its main memory requests to a fixed memory controller. Each memory controller is assigned a balanced number of L2 modules to manage. At the network creation phase, a script selects the sets of L2 modules by a minimization process, which assigns them based on their Manhattan distance to each memory controller, minimizing the overall sum of distances. This process pretends to minimize both the memory controller congestion and NoC delay.

Moreover, an input buffer module with a configurable pipeline latency is integrated into the emulated memory model. The response time in cycles can be adjusted to evaluate designs with a realistic memory controller simulation. This enables studying the impact of different memory latencies on system performance. This parameter involves a delay in transactions between the last level cache and memory controller, which will increase simulation time. This increased time will be directly related to the memory controller's pipeline latency and the simulated application.

C. Optimization of the NoC

Interconnection latency is a significant factor for the overall system performance, so it is essential to keep it as low as possible. To this end, we propose a set of optimizations in the NoC routers to improve the latency of NoC transactions.

1) NoC Router Replacement: The first modification is the integration of ProNoC [13] routers into OpenPiton. ProNoC introduces advanced features aimed at optimizing interconnection latency. One such feature is multihop-bypass [14], which enables injected flits to bypass multiple routers in a single cycle, effectively reducing the overall latency. Additionally, we also leverage another ProNoC feature: increasing NoC concentration. This approach reduces the number of intermediate nodes or routers between cores, thereby diminishing communication latency, particularly in scenarios with low congested traffic.

2) Increased NoC Bus Width: Another modification performed in the routers is the increase of the data bus width. OpenPiton routers use a 16B data bus; therefore, using the default OpenPiton configuration, 1 flit is used to move data from the L2 cache to the L1.5 cache and 2 flits from the L2 cache to the instruction cache, in addition to the header flits. However, when increasing the cache block size (which is one of the optimizations we propose in this paper, as explained in the next section), this narrow 16B data bus introduces extra latency because more flits are needed to move larger cache blocks. To avoid this performance degradation when using larger cache blocks, we increase the data bus width of the routers to reduce the number of flits required in the NoC packets.

D. Improved Cache Configurability

We also propose enhancing the flexibility of the cache configurations in OpenPiton. Specifically, we introduce the ability to adjust cache sizes, cache block sizes, and to increase the data bus width in the cache SRAMs to enhance memory transactions. This flexibility allows users to explore these parameters, analyze their effects, and identify the most suitable configurations for different design objectives and constraints.

1) Cache sizes: The OpenPiton framework offers the option to configure many parameters when building a model, including the cache sizes. The SRAMs dimensions, the block indexing logic and the tag selection logic are adapted according to these parameters when the model is built. However, the cache replacement logic and the L2 address interleaving logic lacks parameterization and requires manual adjustment to different cache sizes. Otherwise, some cache configurations may result in irregular utilization of cache ways and L2 slices. To address this problem, we adjust the cache replacement logic and the L2 address interleaving logic so that they automatically adjust to the cache size.

2) Data Cache Block Sizes: OpenPiton uses different cache block sizes in its cache hierarchy. The L1 data cache and the L1.5 cache use 16B cache blocks (referred to as sublines in the original design), while the L2 cache employs 64B cache blocks. To increase the performance of the private data cache levels, we develop a configurable design that enables using cache block sizes of 16B or 64B in the L1 data cache and the L1.5 cache. Implementing this feature requires automatically adjusting the cache pipelines and the NoC channels according to the configured cache block size. 16B sublines are still preserved in the new design, because Remote Atomic instructions employ this granularity for the synchronization operations that are performed in the shared Last-Level Cache.

3) Instruction Cache Block Size: The OpenPiton cache hierarchy processes transactions from the data cache and the instruction cache separately. Similarly to data caches, the cache block size for instructions differs across cache levels: the L1 instruction cache uses 32B cache blocks, while the L2 operates with 64B blocks. Hence, we incorporate support for using 64B blocks in the L1 instruction cache and for handling 64B requests from the L1 instruction cache to the L2 cache. This support requires modifications to both the L2 cache pipeline and the L1 instruction cache. Furthermore, the invalidation mechanism for instruction cache blocks is also adjusted to 64B blocks.

E. Improved Memory Transactions

The default data size in memory transactions is 16B in the OpenPiton cache hierarchy. This size is constrained by the cache block size in the L1.5 cache, which is 16B, and by the SRAMs data bus size of the L1.5 and the L2 caches, both of which are at 16B. This configuration works well with the default requests and write-backs from the L1.5 cache, which are of 16B. With the new proposed feature of having up to 64B cache blocks, limiting SRAMs data bus size to 16B hinders achieving maximum performance.

When using a cache block size of 64B in the L1.5 cache, memory block requests from the L1.5 cache to the L2 cache require 1 transaction, and reading the L2 cache SRAMs takes 4 cycles in the best case (hit). However, for write-backs from the L1.5 cache to the L2 cache, the L1.5 cache sends 4 transactions of 16B each. When the L2 cache receives the first write-back transaction, it blocks until the other 3 transactions arrive, and then writes the data of each transaction to the appropriate SRAM separately. Thus, the whole write-back process (since the L1.5 cache sends the first transaction to the L2 cache until the L2 cache sends the final ACK to the L1.5 cache) takes at least 48 clock cycles.

In order to minimize the time needed for reads and writes to the L1.5 and the L2 caches, we modify the SRAMs to expand their data bus size to 64B in both cache levels. This enhancement allows for the reading and writing of a 64B cache block within a single cycle. Furthermore, we modify the write-back operation to take place in a single transaction, significantly cutting down on the necessary clock cycles. Modifications in the metadata logic related to the cache blocks (e.g., directory, status, etc.) are not required. This optimization, coupled with the previously proposed 64B cache block size optimization, significantly reduces the access latency to the caches.

IV. Experimental methodology

This section explains the tools, benchmarks and development platforms employed to evaluate our work, as well as the methodology used to accelerate RTL simulations.

A. Benchmarks

We use a set of bare-metal benchmarks to evaluate the proposed optimizations. These benchmarks are selected from the RISC-V tests [15] and the LMbench [16], RiVEC [17] and NAS Parallel Benchmarks [18] suites. Table I shows the benchmarks categorized into three main groups.

TABLE I
BENCHMARKS USED IN THE EVALUATION.

	Benchmark	Size (MB)	Description
a)	copy	128	Vector operation $\vec{C}=\vec{A}$
	scale	128	Vector operation $\vec{B}=k\times\vec{C}$
	add	128	Vector operation $\vec{C}=\vec{A}+\vec{B}$
	triad	128	Vector operation $\vec{A}=\vec{B}+k\times\vec{C}$
b)	matmul	9,5	Multiplication of two 2D matrices of the same dimension
	somier	22	Physics calculations using 3D matrices
c)	histogram	128	Histogram calculation of the distribution of numerical data
	int-sort	64	Sorting a large set of integer numbers

Group *a)* comprises the four kernels of the Stream benchmark (which is part of LMBench), which is aimed at evaluating memory bandwidth. These kernels perform different operations on large data vectors that exceed the cache sizes, and they feature a linear access pattern without data reuse, highlighting their absence of temporal locality.

Group *b)* encompasses matrix operation applications for evaluating arithmetic and memory subsystem efficiency. These compute-intensive apps involve significant data movement and reuse and are geared toward handling 2- and 3-dimensional matrices. MatMul is a matrix multiplication code obtained from the RISC-V tests. Somier (available at [19]) is an application from the physics Simulation domain, which calculates the trajectory of an object in a 3D-space.

Group *c)* consists of applications that count the frequency of distinct values or ranges within datasets. The distinguishing feature is their reliance on atomic operations, comprising a considerable number of such operations which are executed in the L2 shared memory.

These benchmarks have been adapted as required to run in our environment; for example, Int-Sort has been ported to run in baremetal, Histogram (from the Risc-V benchmarks repository) has been parallelized, and the algorithm implementation in MatMul has been fine-tuned. Overall, they represent several kernels typically found in HPC applications, with compute-intensive vector and matrix operations, and present diverse characteristics to explore different aspects of the system.

B. Simulation and Emulation Tools

We employ RTL simulation and design emulation in FPGA to validate and characterize our designs. RTL simulation employs the actual implementation of each module; memory controllers are not implemented, so in this case they are *emulated* using C code, as described in Section II-C. FPGA implementation synthesizes the design to emulate the system behavior, and employs the FPGA resources (memory controller) for elements that are not implemented.

We use Questasim-64 2020.4 and Verilator v4.104 for RTL simulation. While Questasim is primarily used for debugging, Verilator is used for the design space exploration since it allows the execution of multiple simulations in parallel without license restrictions. Being able to use the two simulators also shows the robustness of the implementation of the proposed optimizations.

To generate the results we use Verilator with Metro-MPI [20], which enables the parallelization of a single RTL simulation across different cores and nodes of a cluster and, thus, it greatly reduces the RTL simulation times. Our design space exploration requires approximately 44.25 hours to complete 776 RTL simulations, employing 50,440 cores within Marenostrum 4. Without the use of Metro-MPI this process would become excessively time-consuming.

The Xilinx Alveo U280 and U55C FPGA development boards have been used to emulate the system behavior with actual peripherals (such as a real memory controller) and validate Linux boot and the correct execution of bare-metal applications. Up to 4-core configurations in a 2x2 mesh have been synthesized and evaluated on these development boards.

V. EVALUATION RESULTS

This section delves into the results derived from a range of experiments focused on the design space exploration of Core Tile and the newly added features for high-performance manycores, ultimately resulting in a comparison of Core Tile and its best configurations against OpenPiton and its default configuration.

In this section, two designs are taken as references for the experiments. The Baseline is the default cache hierarchy of OpenPiton, with the most recent code from November 2023[1] employing the following default parameters: 8KB of L1 data cache, 8KB of L1.5 data cache, 16 KB of L1 instruction cache, 64KB of L2 cache per slice, 16B memory blocks, 1 memory controller, and OpenPiton routers. CoreTileBase is the Core Tile architecture with the exact same default parameters inherited from OpenPiton (i.e., it only removes the L1 data cache and uses the L1.5 cache as L1 data cache). Both designs use the DVINO core.

OpenPiton routers have single-cycle latency for packet forwarding in the same direction and two-cycle latency for changes in direction. Furthermore, the simulations are conducted using a memory controller pipeline latency of 150 cycles and a mesh with 64 cores in an 8x8 configuration.

A. Multiple Memory Controllers Exploration

Figure 3 shows the kernel execution time speed-up when varying the number of memory controllers using the automated script. The x-axis shows the number of memory controllers ranging from 1 to 16, and the y-axis shows the speed-up over the CoreTileBase configuration. CoreTileBase implements the memory controller within the chipset and is connected to NoC routers via a crossbar, alongside other modules. When multiple memory controllers are enabled, they are implemented outside the chipset to prevent resource sharing among other modules. This is why in the chart, there is also a case with only one memory controller, which shows a slight improvement in comparison to CoreTileBase.

Notably, group *b)* (blue lines) are not memory-bound applications and do not significantly benefit from adding memory controllers. In contrast, the other groups present significant speed-ups. As the number of memory controllers increases from 1 to 5, the speed-up in groups *a)* and *c)* (green and purple lines) increases drastically. However, as the number of memory controllers increases from 5 to 16, the speed-up reaches a plateau, only showing a slight performance increase in group *c)*. On average (*gmean* line), the speed-up achieved with 4 memory controllers is 2.2x, while with 16 memory controllers it is 2.4x.

[1]Note that the intermediate work in [6] employed an OpenPiton baseline version from April 2021, which may lead to minor discrepancies in the presented results.

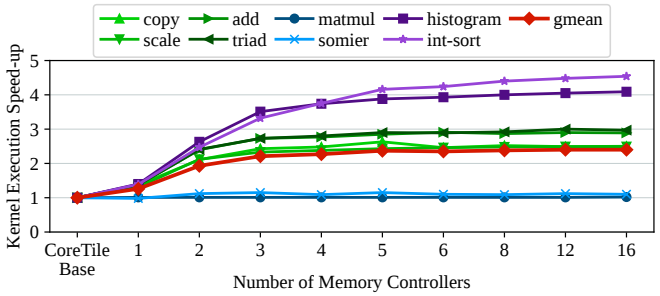


Fig. 3. Speed-up comparison employing Multiple Memory Controllers.

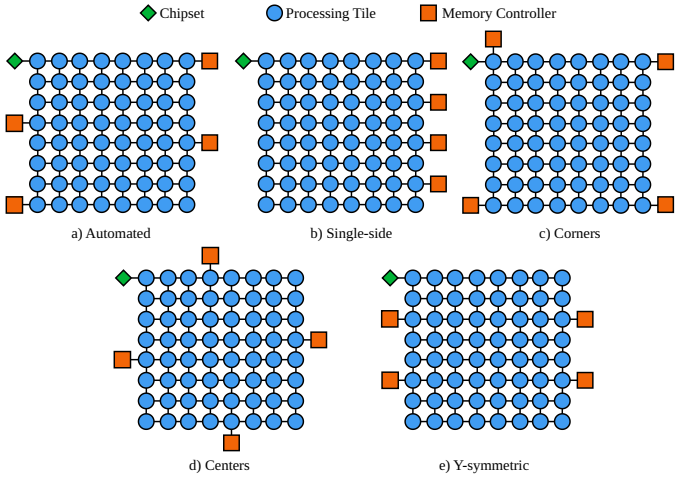


Fig. 4. Custom allocation of four memory controllers at different points along the edge routers.

The *int-sort* benchmark demonstrates a greater speed-up as the number of memory controllers increases. In particular, the speed-up achieved with 5 memory controllers is 4.1x over using only 1 memory controller. Moreover, when employing 16 memory controllers, the speed-up is enhanced even further, reaching 4.5x speed-up compared to a single memory controller scenario. This is attributed to the nature of being a memory-bound application. The application executes a substantial amount of atomic operations on shared memory together with numerous movements in memory.

Next, we evaluate the impact of the memory controllers placement, inspired by the analysis in [21]. We restrict our analysis to 4 memory controllers placed in the mesh edges, either using the automated script or manual placement. Figure 4 depicts the configurations studied. Configuration *a)* represents the base configuration, automatically generated by the script. Configuration *b)* allocates memory controllers on one side of the mesh, while configuration *c)* employs memory controllers at each corner of the mesh. Configurations *d)* and *e)* distribute the memory controllers near the centers of the edges: The first case follows a diamond configuration using the four edges, and the second employs Y-sides with memory controllers symmetrically distributed.

Figure 5 depicts the performance improvement achieved by each distribution of memory controllers in the mesh. On average, an improvement between 1.7% and 3.9% is achieved for all applications using these manual placement alternatives, with the highest improvement achieved by Y-symmetric. Group *a)* benefits the most from the organization of the memory controllers in the majority of cases, achieving up to 8.6% improvement in *triad*. This

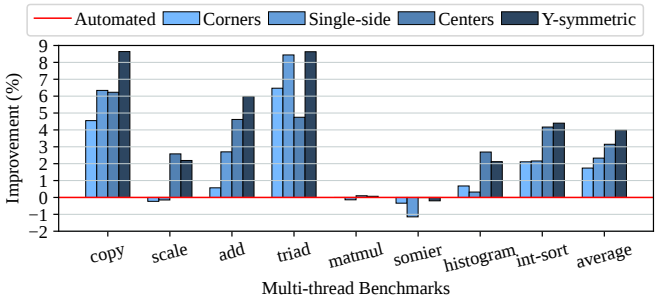
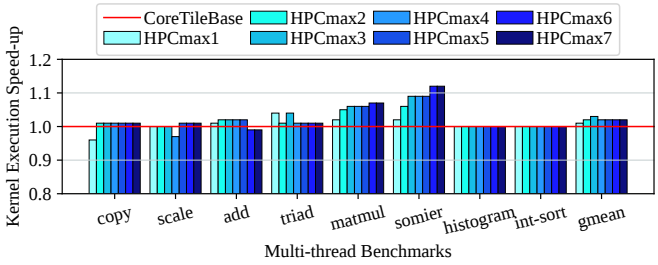
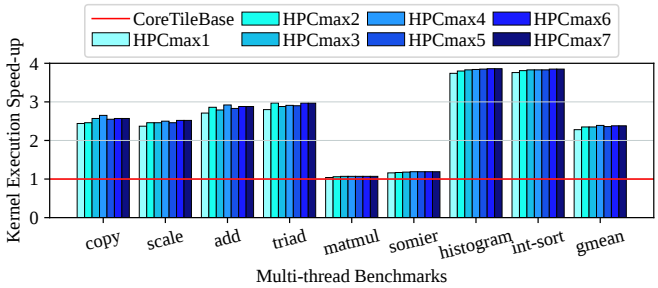


Fig. 5. Performance improvement achieved by allocating four memory controllers at different points along the edge routers.



(a) Different number of HPC_{max} and 1 memory controller.



(b) Different number of HPC_{max} and 4 memory controllers.

Fig. 6. Speed-up of different degrees of NoC multi-hop bypass (HPC_{max}) with 1 and 4 memory controllers.

occurs because this group is memory intensive, requiring fast access to main memory. In group *b)* the impact is despicable. Applications in this group exhibit temporal locality, with a large number of transactions hitting at the private and shared cache levels, reducing main memory accesses. The group *c)* has a mixture of transactions involving atomic operations and data loads. Since atomic operations are executed at the shared level (L2), this group yields intermediate results between groups *a)* and *b)*.

B. Multi-Hop Bypass Exploration

To analyze the impact of multi-hop bypass on application execution time, we perform simulations using 64 cores with 1 and 4 memory controllers. Figure 6 presents the results. The CoreTileBase configuration is compared against ProNoC with varying values of HPC_{max}, from 0 to 7, in which packets with a destination in the same direction can bypass $HPC_{max} - 1$ routers within a single cycle. For $HPC_{max} = 1$, ProNoC routers present the same base latency as the routers in the OpenPiton baseline, but with a different router architecture. On average, in a system with 1 memory controller, multi-hop bypass provides up to 3% speed-up

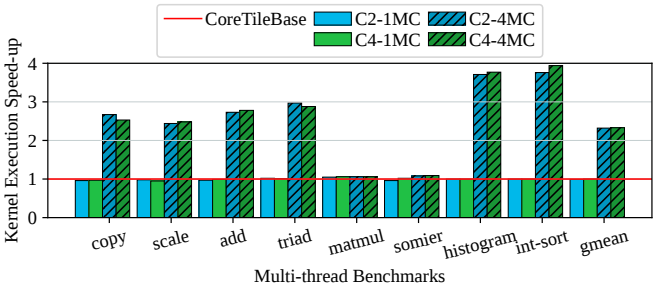


Fig. 7. Speed-up of NoC concentration 2 and 4 (C2/C4) with 1 and 4 memory controllers (1MC/4MC).

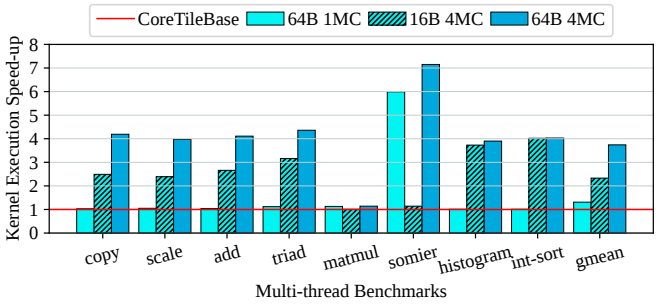


Fig. 8. Performance comparison between 16- and 64-Byte cache blocks for the L1 data cache across configurations with 1 and 4 memory controllers (1MC/4MC).

with HPC_{max} 3, while in a system with 4 memory controllers it provides up to 2.4x speed-up with HPC_{max} 4.

As shown in Figure 6a, the benefit achieved by multi-hop bypass when using 1 memory controller is negligible in groups *a)* and *c)*, while for group *b)* it is between 2% to 12%. The effectiveness of multi-hop bypass increases when the bandwidth bottleneck is alleviated by using 4 memory controllers, as shown in Firgure 6b. In this case, increasing HPC_{max} provides performance benefits in all the benchmarks, up to 8% in *add* with HPC_{max} 4 compared to HPC_{max} 1.

C. NoC Concentration Exploration

To observe the influence of Noc concentration on performance, we execute experiments with NoC concentrations of 2 and 4. The simulations are performed with two different configurations of memory controllers, 1 and 4. Figure 7 illustrates the results of this experiment.

When using 1 memory controller, the overall performance results indicate a 1% performance decrease with concentration 2 (bar C2-1MC), but using concentration 4 (bar C4-1MC) restores the performance of the baseline. When employing 4 memory controllers, a consistent 2.3x geometric mean speed-up is achieved in both configurations of concentration (bars C4-1MC and C4-4MC). The most noticeable speedups are achieved by group *c)* with 4 memory controllers, where increasing the concentration to 4 provides up to 5% speed-up in *int-sort* compared to using concentration 2. Groups *a)* and *b)* do not benefit from NoC concentration or even present a very slight performance degradation in group *a)*.

D. Cache Block Size Exploration

1) Data Cache: Figure 8 illustrates the speed-up achieved by increasing the L1 data cache block size from 16 to 64 bytes in two distinct scenarios, where the number of memory controllers is 1 and 4. The average speed-up obtained is 1.3x employing 64B cache blocks and 1 memory controller, while with 4 memory controllers 3.7x is achieved. It is challenging to perceive the impact of using 64B cache blocks with just 1 memory controller, given the bottlenecks generated in the main memory. However, when these bottlenecks are reduced by incorporating more memory controllers, the benchmarks take better advantage of this new feature, resulting in higher speed-ups.

In group *a)*, with only 1 memory controller, the increase in cache block size to 64B results in minimal improvement because the primary limitation remains in the memory bandwidth. In contrast, when 4 memory controllers are employed with 16B cache blocks, significant speed-ups ranging from 2.3x to 4x are achieved. Therefore, by increasing the cache block size to 64B, the speed-ups are further enhanced up to 4x.

Within group *b)*, we encounter two special cases. The data reuse within this group is notably high. However, *somier* exhibits a greater benefit than *matmul* when the cache block size is enlarged. *somier* processes 3D matrices, whereas *matmul* is limited to 2D matrices. Consequently, *somier* demonstrates more pronounced data reuse compared to *matmul*, resulting in increased traffic between shared and private caches, as opposed to traffic to/from the main memory. Conversely, *matmul* can efficiently handle the data within the private caches due to its smaller input dataset.

Group c) demonstrates minimal advantage when increasing the size of cache blocks, regardless of whether 1 or 4 memory controllers are employed. This is because these benchmarks exhibit notably low spatial locality. Atomic memory operations predominate in this kind of applications, which are served by the shared L2 cache. Therefore, these operations do not get any benefit when increasing the private L1 cache block size.

2) Instruction Cache: We do not observe any performance improvement by increasing the cache block size in the L1 instruction cache. This behavior arises from the small instruction footprint compared to the dataset size of our benchmarks. Since the entire program fits into the instruction cache, when the kernel is executed, the entire program is cached in the instruction cache.

The variations in the results are nearly insignificant, and they are primarily attributed to the invalidations in the instruction cache triggered by the coherency protocol. We observe that, when using 64-byte blocks, requesting a new instruction cache block takes slightly more time than when using 32-byte cache blocks. Yet, this request time is also affected by the distance from the requester core to the home node, for any cache block size.

E. NoC Bus Size Exploration

Figure 9 displays the geometric mean derived from a series of exploratory experiments in which various NoC bus sizes are evaluated across all applications. The figure shows the effect of increasing the NoC bus size in two scenarios: employing 1 and 4 memory controllers. The experiments labeled as 64b, 12b, 156b and 512b use the default cache size configurations but with 64-byte cache blocks in both L1 data and instruction caches. The results are normalized to the CoreTileBase results, which uses the default cache size and cache block size configurations.

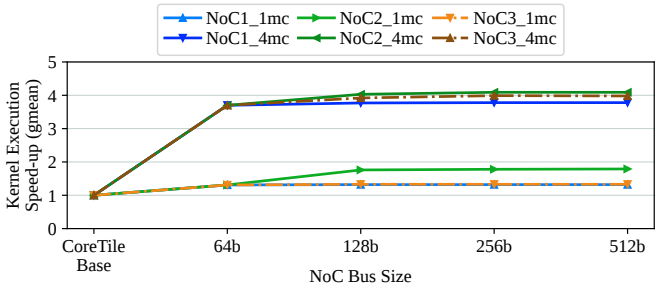


Fig. 9. Geometric mean speed-up achieved with different bus widths in the NoC routers using 1 and 4 memory controllers (1mc/4mc).

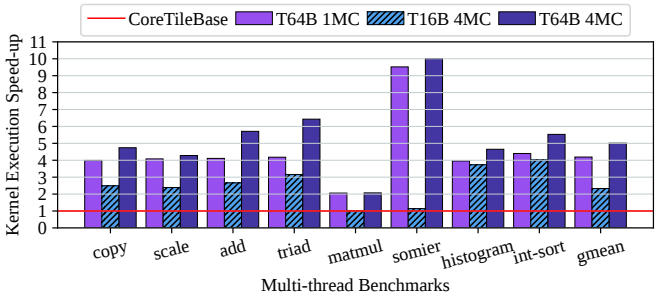


Fig. 10. Speed-up comparison between 16- and 64-byte transactions and data bus width in SRAMs and NoCs across configurations with 1 and 4 memory controllers (1MC/4MC).

In the scenario with 1 memory controller, increasing the width of the NoC2 to 512 bits shows up to a 37% benefit compared with 64 bits, and it also achieves a 1.8x speedup compared with the CoreTileBase. This is because NoC2 is used for data transfers and is responsible for moving entire memory blocks from the L2 cache to the L1 cache. For NoC1 and NoC3, increasing their widths from 64 bits to 512 bits provides performance benefits of less than 1%. This is attributable to the maximum payload size per type of request from the L1 cache to the L2 cache being 128 bits. NoC3 is also used to deliver memory blocks from main memory to the L2 cache but, with only one memory controller, the latency advantage of having a wider bus width is negligible compared to the latency of of accessing main memory.

When using 4 memory controllers the benefit of widening NoC3 can be observed, where a 7.5% speed-up is achieved with 512 bits compared to 64 bits. This configuration achieves a 4.0x speed-up compared to the CoreTileBase. In NoC 2 with 512 bits, we are achieving a 10.1% benefit compared to 64 bits and 4.1x against CoreTileBase. Finally, in NoC 1, the benefit continues to be less than 1%.

F. 64-byte Transactions Exploration

This section evaluates the combined impact of three design optimizations: wide buses in the NoCs, wide data buses in the SRAMs of the caches, and wide cache block sizes. We refer to the combination of these three parameters as the *transaction width*. Together, these three optimizations offer the necessary means to store, read, write, and transmit large memory blocks in the whole memory hierarchy. Importantly, these three parameters offer the best performance when they are configured in unison, as configuring them with different widths can lead to an underutilization of the wide resources due to the bottlenecks introduced in the narrower resources. For instance, 64-byte NoC buses are underutilized when transmitting 16-byte cache blocks, while employing 64-byte cache blocks and data buses in the SRAMs of the caches combined with 16-byte NoC buses can create bottlenecks when data is transferred through the NoC.

Figure 10 compares the performance of using transaction widths of 16B and 64B (T16B/T64B) with 1 and 4 memory controllers (1MC/4MC). All results are normalized to CoreTileBase, which employs 16-byte transactions and 1 memory controller. The bar *T16B 4MC* is CoreTileBase but employing 4 memory controllers. According to the geometric mean, a transaction width of 64B with 1 memory controller shows better results than the baseline and than 16-byte transactions with 4 memory controllers (4.1x and 1.8x respectively). The better results are achieved with 64-byte transactions and 4 memory controllers, which achieve up to 5.0x speed-up over the baseline. Comparing 64-byte with 16-byte transactions in the scenario with 4 memory controllers, results shows that groups *a)* and *b)* achieve huge speedups of around 2x in all cases except *somier*, which achieves 9.2x. Group *c)* achieves moderate speed-ups as well, but these are limited by the atomic operations, which are executed in L2 and are implemented with 16-byte transactions, so they do not get the most of the wider buses and cache blocks.

G. Cache Size Exploration

Figure 11 illustrates the results from exploring the use of larger caches in the system. In this experiment, we evaluate different cache setups, ranging from 8KB to 128KB for the L1 cache and from 64KB to 512KB for the L2 cache. The evaluations are conducted using 1 and 4 memory controllers. Overall, we observe that larger caches provide great performance improvements, specially when enlarging the L2 cache capacity. On average, the largest cache configuration achieves speed-ups of 2.5x and 4.8x with 1 and 4 memory controllers, respectively.

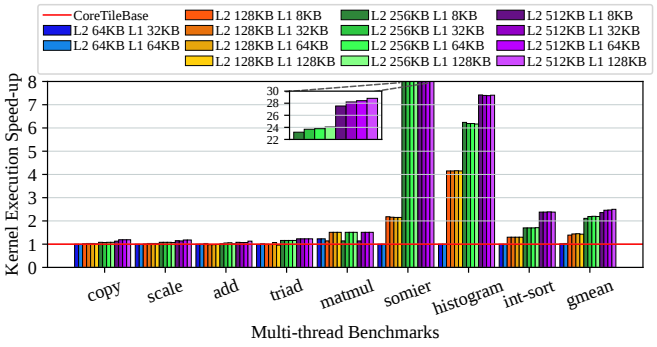
Group *a)* shows limited benefits after increasing the cache sizes. This behavior is attributed to the fact that this group of applications fails to exploit temporal locality. In the scenario with 4 memory controllers, a noticeable benefit of up to 3.2x is observed. This is a result of reduced bottlenecks when accessing main memory, rather than being due to an increase in cache sizes.

The significant data reuse observed in group *b)* contributes to achieving greater speed-ups when cache sizes are increased. Both with 1 and 4 memory controllers, in *somier* the benefits of larger caches are evident, with up to 38x speed-up when using a 128KB L1 cache and a 512KB L2 cache with 1 memory controller. The large speed-ups achieved in *somier* are the result of having more traffic between shared and private caches than between shared caches and main memory, due to data reuse. This avoids bottlenecks in the main memory. However, in the case of *matmul*, the benefit is not as pronounced due to the smaller dataset used.

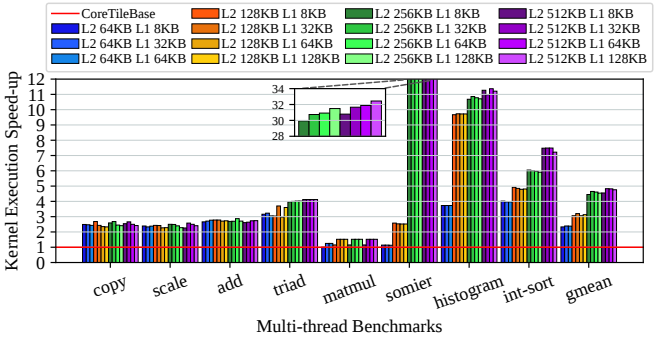
Group *c)* also experiences benefits from the large caches, particularly when the shared L2 cache is enlarged. A remarkable characteristic of this group is the utilization of atomic operations, which are executed within the shared L2 cache. With an increase in L2 cache capacity, the hit ratio for atomic operations improves, leading to a reduction of main memory accesses.

TABLE II
TESTED CONFIGURATIONS WITH NEW FEATURES.

Configuration name	L1 (KB)	L2 (KB)	Block Size (B)	Mem. Ctrl.	By-pass	Conc.	SRAM Bus (B)	NoC Bus (B)
Baseline	8	64	16	1	-	-	16	8
Large	32	512	16	1	-	-	16	8
Large+Ctrl	32	512	16	4	-	-	16	8
Large+Block+Ctrl	32	512	64	4	-	-	16	8
Large+Block+Ctrl+Byp	32	512	64	4	4	-	16	8
Large+Block+Ctrl+Con	32	512	64	4	-	4	16	8
Large+Block+Ctrl+T64	32	512	64	4	-	-	64	64



(a) Different cache sizes and 1 memory controller.



(b) Different cache sizes and 4 memory controllers.

Fig. 11. Speed-up of different combinations of L1 data cache and L2 cache sizes with 1 and 4 memory controllers.

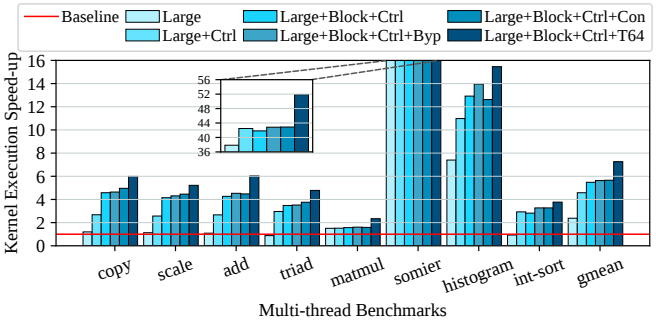


Fig. 12. Speed-up of different configurations with the newly added features against the OpenPiton baseline.

H. Comparison with OpenPiton Baseline

Finally, we conduct experiments that combine the proposed extensions and optimizations to OpenPiton, and we compare their performance with the OpenPiton baseline. The evaluated configurations are presented in Table II. The configurations are selected after analyzing the experiments previously conducted in this section, focusing on those that provide the largest performance benefits. The configurations are incrementally built according to the following selected parameters: cache sizes of 512KB for L2 and 32KB for L1 (*large*), 64B memory blocks (*Block*), 4 memory controllers (*Ctrl*), router bypassing (*Byp*), router concentration (*Con*), and a 64B buses in the NoC and in the SRAMs (*T64*).

1) Cache Hierarchy Optimizations: Figure 12 presents the results obtained from incorporating all the features and optimizations into OpenPiton. On average, the speed-up achieved by using large caches of 32KB L1 cache and 512KB L2 cache(*Large*) is around 2.3x. By adding more memory controllers (*Large+Ctrl*), the average speed-up increases to 4.5x. On top of these two features, increasing the cache block sizes to 64B (*Large+Block+Ctrl*) achieves an average speed-up of 5.4x. Incrementally introducing multi-hop bypass (*Large+Block+Ctrl+Byp*) or concentration (*Large+Block+Ctrl+Con*) in the routers results in a slight average performance increase, reaching 5.6x over the baseline. The maximum speed-up is achieved when the data bus size of the NoCs and the cache SRAMs is increased to 64B (*Large+Block+Ctrl+T64*), achieving a 7.2x speed-up over the baseline (28.5% over the intermediate work in [6]). With this configuration, the most important bottlenecks of the memory controller, caches, and NoC routers are effectively removed.

2) Average Memory Access Time: Figure 13 shows the Average Memory Access Time (AMAT) employing different features to improve the memory access latency in the Core Tile compared with the OpenPiton baseline. The results are obtained simulating a vector reduction benchmark with different input set sizes on a single core configuration with one memory controller. The benchmark iterates on a loop i doing the operation $accum = accum + A[i]$. The results are categorized into three ranges. The first range occurs when the whole data set fits in the L1 cache. In the second range, the size of the data set is larger than the L1 cache but smaller than the L2 cache. The third range covers scenarios where the data set is larger than the L2 cache, so the execution is dominated by accesses to main memory.

The first configuration, *CoreTileBase*, consists of removing the write-through cache level of the OpenPiton Baseline. The result of this modification is a 9.3% AMAT reduction in the third range (from 96KB to 4MB) and an AMAT reduction of 12.1% in the second range (from 8KB to 64KB). The second configuration, (*Large*), increases the size of the caches, which results in a range displacement because of the extra capacity. As expected with the large cache sizes

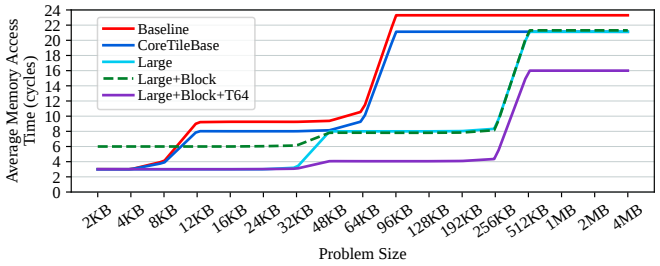


Fig. 13. Average memory access times achieved with various configurations with different input set sizes.

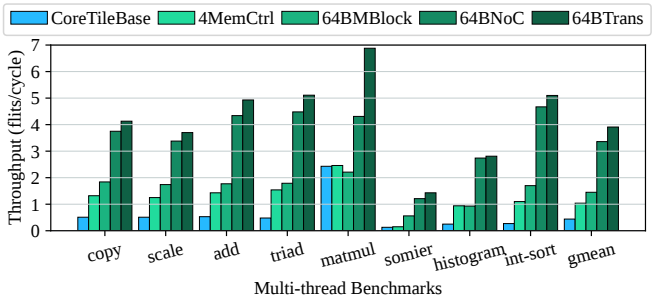


Fig. 14. Throughput achieved in NoC2 employing different configurations.

specified in Table II, the third range is shifted from 512KB to 4MB, and the second range from 32KB to 512KB. The AMAT in each of the ranges does not change compared to *CoreTileBase*.

When the cache block size increases to 64 bytes in the configuration *Large+Block*, the AMAT increases in the first range from 3 to 6 cycles. This happens because, using 64-byte cache blocks without the T64 optimization, the cache blocks are read in chunks of 16 bytes. Thus, 4 cycles are needed to read the complete 64-byte cache blocks, compared to a single cycle for 16-byte cache blocks. The increase for other ranges is approximately 0.8% in comparison to *CoreTileBase*.

Finally, a significant AMAT reduction is achieved when the widht of the data buses of the SRAMs and NoCs is increased in the configuration *Large+Block+T64*. We observe a reduction in the second and third ranges from 9.3 to 4.3 cycles and from 23 to 16 cycles, respectively. Additionally, the first range is reduced to 3 cycles again. All together, this final configuration shows much lower average memory acces times than the rest of configurations and than the OpenPiton Baseline.

3) Throughput: ProNoC latency and throughput parameters have been studied in isolation, including the impact of bypass and concentration [14]. This subsection explores the impact of the memory hierarchy changes on the memory throughput obtained by the application. We measure the throughput as the average number of 64B flits/cycle delivered by NOC2, which is the NOC that serves data requests, during the application Region Of Interest. This is an aggregate value for the 64 tiles in the system.

Figure 14 presents the measured throughput for different cumulative configurations: *CoreTileBase* employs the default configuration with a single Memory Controller; *4MemCtrl* increases this value to 4, using the automated placement; *64BMBlock* increases block size to 64 Bytes (but it requires several consecutive SRAM accesses); *64BNoC* increases the NoC bus width to 64B to reduce serialization delays; finally, *64BTrans* supports 64B SRAM transfers. Each of these improvements speeds-up the execution (reducing the

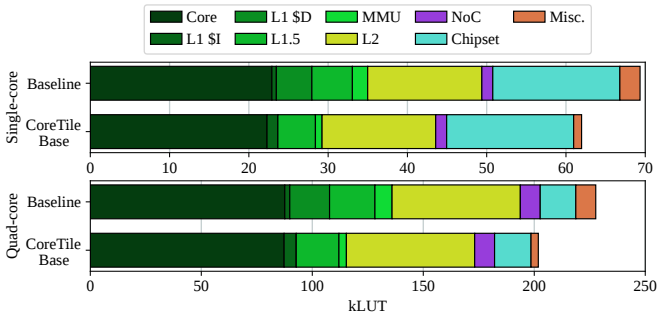


Fig. 15. LUTs utilization for a Single- and Quad-core implementations in the Xilinx Alveo development boards.

ROI duration), which increases average throughput values.

We observe that the throughput of the base configuration is highly dependent on the application; Matmul presents larger temporal locality, which increases measured throughput by reducing the percentage of main memory accesses. The different Stream benchmarks are memory intensive but they do not exploit any temporal locality, obtaining much lower values.

The configuration changes that yield the largest throughput improvements are the use of multiple Memory Controllers (particularly for Stream applications) and the wider NoC bus. On average, these changes improve mean throughput from 0.44 flits/cycle to 1.04 and 3.36 respectively. The 64B tranfer size is particularly effective in matmul, providing a 60% throughput improvement; overall, the complete configuration increases measured throughput by $8.95\times$ on average, larger than the application speedup improvements.

4) FPGA implementation: Figure 15 shows the resource utilization of the CoreTileBase configuration within an FPGA, comparing it with the resource utilization of the OpenPiton Baseline. Single- and Quad-core configurations are evaluated interconnecting via a 2D-mesh. Both cases exhibit an approximate 11% reduction in the number of LUTs required to implement the CoreTileBase.

The reduction is mainly attributed to the fact that the write-through L1 data cache is removed in CoreTileBase. This cache represents 6.5% of the total amount of LUTs implemented. The remaining percentage is attributed to differences produced by logic that is not implemented because of the removal of this cache (signals, buses, arbiters, etc.) or by other miscellaneous logic. For example, Baseline uses modules from CVA6 such as the MMU and the instruction cache, while CoreTileBase uses analogous modules developed in-house.

VI. Discussion

This section discusses certain aspects about the application and configuration of our design.

Application domain: The present work introduces a series of optimizations specifically designed and tested within the domain of High-Performance Computing (HPC). For this reason, they are tested using representative benchmarks for HPC workloads. However, these configurations can also be applied beyond the scope of HPC. The design improvements over OpenPiton will clearly benefit applications in other domains. For example, a larger cache capacity and block size will benefit any memory intensive application (such as AI, deep learning or accelerators) and streaming applications will similarly benefit from increased bandwidth.

Optimal Design tuning: We do not suggest a single optimal design point or configuration. Based on the specific application domain, each target chip requires an evaluation to determine the most suitable configuration, constrained by the requirements and limitations of the design, typically in terms of performance, area and power. The proposed tool provides flexibility to configure and test multiple setups, which allows the architect to test and find the optimal configuration for the target application domain.

Memory mapping policies: The assignment mechanisms that maps L2 modules to memory controllers, presented in Section III-B, statically assigns a set of L2 modules to each memory controller. Such simple design avoids any memory interleaving from an L2 module to main memory. The reason to select such design is that the access from L1 to L2 already implements an interleaved access policy, so the benefit obtained by interleaving accesses from L2 to the MC would be small while increasing the complexity of the design. Alternative policies could be implemented in the future to improve the flexibility.

VII. Related work

BlackParrot [2] is a 64-bit RISC-V multi-core processor featuring a two-level cache hierarchy with varied cache coherency protocols (VI, MSI, and MESI). Similar to OpenPiton, BlackParrot employs multiple memory controllers and a 2-D mesh NoC with three physical routers without virtual channeling. Unlike OpenPiton, BlackParrot has a specific network to connect the memory controllers with the L2 slices. BlackParrot has been taped-out with a 4-core configuration.

The PULP platform [5] comprises a RISC-V core and an octa-core accelerator within a low-power SoC targeting IoT applications. The cache hierarchy includes a 512KB shared L2 cache, registers banks, and scratch caches. The shared L2 cache is divided into four slices. The RISC-V core lacks a private cache level but has two 32KB register banks used to allocate the program stack and private data. The octa-core accelerator employs a scratch cache and directly accesses the L2 cache. PULP employs an AXI-based communication between cores and the cache hierarchy. Unlike OpenPiton, PULP is not focused on manycore systems.

Agiler [22] is a multi-core architecture based on RISC-V, targeting heterogeneous systems. This architecture is composed of three kinds of processing tiles. The main processing tile comprises a quad-core, a shared instruction cache, and a memory controller. The second type consists of compute tiles based on RISC-V. The compute tiles include a 64-bit dual-core or a 32-bit quad-core, both with shared data and instruction caches. The main processing tile and the compute tiles internally employ AXI-based communication. The third type of tiles consists of custom accelerators. The accelerators and the compute tiles are interconnected via mesh routers. Each compute tile has its own memory region to work in, and its respective data and instructions are loaded into the corresponding region. In contrast, in OpenPiton, each tile works within the same memory space.

Open ESP [3] is an open-source framework for developing embedded systems and prototyping SoCs. It features a modular FPGA SoC architecture using tiles interconnected via a 2D-Mesh NoC with look-ahead routing. In addition, it provides tools and libraries to create software applications. The SoC includes four tile types: processor, accelerator, memory, and auxiliary. The processor tiles house two different cores to choose, which are RISCV 64-bit Ariane or SPARC 32-bit with two MESI-coherent private cache levels. The accelerator tiles execute coarse-grained tasks during the exchanging of large datasets in the memory hierarchy, facilitating efficient data exchange. The memory tiles include a shared LLC slice and a memory controller port, while the auxiliary tiles employs controllers to manage peripherals. Similar to OpenPiton, ESP supports multiple memory controllers and coherence protocols, enhancing scalability in manycores.

VIII. Conclusions

In recent years, the emergence of RISC-V has led to an increase in both academic and industrial multicore processor prototypes. With the significance of NoCs in large multicore systems, platforms such as OpenPiton have garnered significant interest. Nevertheless, OpenPiton's NoC and memory hierarchy lack essential features commonly found in high-performance manycores.

This paper presents a set of extensions and optimizations to the NoC and the memory hierarchy of OpenPiton for improving the performance of large-scale multicores and manycores. In particular, we add the capability to increase the number of memory controllers in the system, reducing the bottleneck caused when a big number of tiles try to access a single memory controller at the same time. In order to optimize the latency transactions in the cache hierarchy, introduce bypassing and NoC concentration features in the routers and we increase the NoC bus width to 64 bytes. In the cache hierarchy, we enable increasing the cache sizes, the cache blocks sizes and the SRAM bus widths. All together, these new features allow reducing the latency and augmenting the bandwidth of the cache hierarchy and the NoC.

We evaluate our proposal using RTL simulations and we demonstrate that many applications with different characteristics can take advantage of the presented optimizations and new features, including memory-bound, cache-intensive and synchronization-intensive applications. Overall, the combination of the proposed new features and optimizations on a 64-core manycore architecture provides an speed-up of 7.2x compared to the OpenPiton baseline, or 28.5% over the intermediate work in [6].

Acknowledgments

This work has been partially supported by the European HiPEAC Network of Excellence, by the Spanish Ministry of Science and Innovation MCIN/AEI/10.13039/501100011033 (contracts PID2019-107255GB-C21, PID2022-136454NBC21 and TED2021-131176B-I00), by the Spanish Ministry of Economy and Digital transformation (contract TSI069200-2023-0011), by the Generalitat de Catalunya (contract 2021-SGR-00763), and by Lenovo-BSC Contract-Framework Contract (2022). The BZL project is funded by the Ministerio de Transformación Digital y de la Función Pública by the Plan de Recuperación, Transformación y Resiliencia - financed by the European Union - NextGenerationEU. G. López-Paradís has been supported by the Generalitat de Catalunya through a FI fellowship 2021FI-B00994.

REFERENCES

[1] J. Balkind, M. McKeown, Y. Fu, T. M. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang, M. Matl, and D. Wentzlaff, "OpenPiton: An open source manycore research framework," in *21st International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '16, 2016, pp. 217–232.

[2] D. Petrisko, F. Gilani, M. Wyse, D. C. Jung, S. Davidson, P. Gao, C. Zhao, Z. Azad, S. Canakci, B. Veluri, T. Guarino, A. Joshi, M. Oskin, and M. B. Taylor, "BlackParrot: An agile open-source RISC-V multicore for accelerator SoCs," *IEEE Micro*, vol. 40, no. 4, pp. 93–102, 2020.

[3] P. Mantovani, D. Giri, G. Di Guglielmo, L. Piccolboni, J. Zuckerman, E. G. Cota, M. Petracca, C. Pilato, and L. P. Carloni, "Agile soc development with Open ESP," in *39th International Conference on Computer-Aided Design*, ser. ICCAD '20, 2020.

[4] A. Bradbury, G. R. Ferris, and R. D. Mullins, "Tagged memory and minion cores in the lowRISC soc," 2014.

[5] A. Pullini, D. Rossi, I. Loi, G. Tagliavini, and L. Benini, "Mr.Wolf: An energy-precision scalable parallel ultra low power SoC for IoT edge processing," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 7, pp. 1970–1981, 2019.

[6] N. Leyva, A. Monemi, N. Oliete-Escuín, G. López-Paradís, X. Abancens, J. Balkind, E. Vallejo, M. Moretó, and L. Alvarez, "OpenPiton optimizations towards high performance manycores," in *16th International Workshop on Network on Chip Architectures*, ser. NoCArc '23, 2023, p. 27–33.

[7] OpenHW Group CV-MESH repository, https://github.com/openhwgroup/cv-mesh, 2024.

[8] K. Lim, J. Balkind, and D. Wentzlaff, "JuxtaPiton: Enabling heterogeneous-ISA research with RISC-V and SPARC FPGA soft-cores," 2018. [Online]. Available: https://arxiv.org/abs/1811.08091

[9] J. Balkind, M. Schaffner, K. Lim, F. Zaruba, F. Gao, J. Tu, D. Wentzlaff, and L. Benini, "OpenPiton+Ariane: The first SMP Linux-booting RISC-V system scaling from one to many cores," in *Workshop on Computer Architecture Research with RISC-V*, ser. CARRV '19, 2019.

[10] N. I. Leyva-Santes, I. Pérez, C. A. Hernández-Calderón, E. Vallejo, M. Moretó, R. Beivide, M. A. Ramírez-Salinas, and L. A. Villa-Vargas, "Lagarto I RISC-V multi-core: Research challenges to build and integrate a network-on-chip," in *Supercomputing*, 2019, pp. 237–248.

[11] J. Balkind, K. Lim, M. Schaffner, F. Gao, G. Chirkov, A. Li, A. Lavrov, T. M. Nguyen, Y. Fu, F. Zaruba, K. Gulati, L. Benini, and D. Wentzlaff, "BYOC: A "bring your own core" framework for heterogeneous-ISA research," in *25th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20, 2020, p. 699–714.

[12] G. Cabo, G. Candón, X. Carril, M. Doblas, M. Domínguez, A. González, C. Hernández, V. Jiménez, V. Kostalampros, R. Langarita, N. Leyva, G. López-Paradís, J. Mendoza, F. Minervini, J. Pavón, C. Ramírez, N. Rodas, E. Reggiani, M. Rodríguez, C. Rojas, A. Ruiz, V. Soria, A. Suanes, I. Vargas, R. Figueras, P. Fontova, J. Marimon, V. Montabes, A. Cristal, C. Hernández, R. Martínez, M. Moretó, F. Moll, O. Palomar, M. A. Ramírez, A. Rubio, J. Sacristán, F. Serra-Graells, N. Sonmez, L. Terés, O. Unsal, M. Valero, and L. Villa, "DVINO: A RISC-V vector processor implemented in 65nm technology," in *37th Conference on Design of Circuits and Integrated Circuits*, ser. DCIS'22, 2022, pp. 1–6.

[13] A. Monemi, J. W. Tang, M. Palesi, and M. N. Marsono, "ProNoC: A low latency network-on-chip based many-core system-on-chip prototyping platform," *Microprocessors and Microsystems*, vol. 54, pp. 60–74, 2017.

[14] A. Monemi, I. Pérez, N. Leyva, E. Vallejo, R. Beivide, and M. Moretó, "PlugSMART: A pluggable open-source module to implement multihop bypass in networks-on-chip," in *15th International Symposium on Networks-on-Chip*, ser. NOCS '21, 2021, p. 41–48.

[15] RISC-V International - riscv-tests, https://github.com/riscv-software-src/riscv-tests, 2012.

[16] L. McVoy and C. Staelin, "Lmbench: Portable tools for performance analysis," in *Annual Conference on USENIX Annual Technical Conference*, ser. ATEC '96, 1996, p. 23.

[17] C. Ramírez, C. A. Hernández, O. Palomar, O. Unsal, M. A. Ramírez, and A. Cristal, "A RISC-V simulator and benchmark suite for designing and evaluating vector architectures," *ACM Transactions on Architecture and Code Optimization*, vol. 17, no. 4, nov 2020.

[18] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga, "The NAS parallel benchmarks," *International Journal of High Performance Computing Applications*, vol. 5, no. 3, p. 63–73, 1991.

[19] RiVEC Bencmark Suite repository, https://github.com/RALC88/riscv-vectorized-benchmark-suite, 2022.

[20] G. López-Paradís, B. Li, A. Armejach, W. Stefan, M. Moretó, and J. Balkind, "Fast behavioural RTL simulation of 10B transistor SoC designs with Metro-MPI," in *Design, Automation and Test in Europe Conference*, ser. DATE'23, 2023, pp. 1–6.

[21] D. Abts, N. D. Enright Jerger, J. Kim, D. Gibson, and M. H. Lipasti, "Achieving predictable performance through better memory controller placement in many-core CMPs," in *36th International Symposium on Computer Architecture*, ser. ISCA '09, 2009, p. 451–461.

[22] A. Kamaleldin and D. Göhringer, "AGILER: An adaptive heterogeneous tile-based many-core architecture for RISC-V processors," *IEEE Access*, vol. 10, pp. 43 895–43 913, 2022.

AUTHOR BIOGRAPHIES

Neiel Leyva is a PhD. Student at Universitat Politècnica de Catalunya and Research engineer at Barcelona Supercomputing Center (BSC). His current research interest are Networks-on-Chip (NoC) design and memory hierarchies for many-core systems. He received the M.S degree in Computer Engineering from Computing Research Center of the National Polytechnic Institute of Mexico.

Alireza Monemi received the M.S. and Ph.D. degrees in electrical engineering from Universiti Teknologi Malaysia, Malaysia, in 2011 and 2017, respectively. He is currently a Postdoctoral research associate at Barcelona Supercomputing Center (BSC). His current research interest is cache-coherent Networks-on-Chip (NoC) design.

Noelia Oliete-Escuín is a Master's student at UPC. She received her BSc degree in Computer Engineering from the University of Zaragoza in 2022. She is currently working as a Research Engineer for the UNCORE: Cache Hierarchies and Interconnect group at the Barcelona Supercomputing Center (BSC). Her research interests are in computer architecture, focused on cache and memory hierarchies.

Guillem López-Paradís is a PhD student at BSC and UPC. He received his BSc degree in Computer Engineering from UPC in 2017 and MSc from UPC in 2021. He is currently at the High Performance Domain-Specific Architectures group at BSC. Additionally, he has been a visiting PhD student at the University of California, Santa Barbara (UCSB), USA; done an internship at Xlabs in Xilinx, Dublin (Ireland). His interests include interconnecting accelerators with the coherence systems, accelerating RTL Simulations and computer architecture.

Xabier Abancens graduated as telecommunications engineer (Dipl.-Ing. degree) in 2013 from the University of Navarra, obtaining the best academic record award. He worked last 10 years on research projects related to satellite communications on FPGA, as well as industry and medical start-up company. In 2022 he joined at Barcelona Supercomputing Center (BSC) as a leading research engineer. His main research interests include RISCV cores, FPGA and MPSOC based custom designs, low-latency and real-time systems, as well as secure communication protocols.

Jonathan Balkind is an Assistant Professor in the Department of Computer Science, UC Santa Barbara. His research interests include computer systems, programming languages, and computer architecture with the aim of improving the efficiency of modern multicore systems in mobile and datacenter environments. He received the M.Sci. degree in computing science from the University of Glasgow and the M.A. and Ph.D. degrees in computer science from Princeton University. Contact him at jbalkind@ucsb.edu.

Enrique Vallejo is an associate professor in the University of Cantabria. His research interests are in computer architecture, mainly interconnection networks and parallel architectures. He holds a PhD in computer architecture from the U. Cantabria.

Miquel Moretó received the B.Sc., M.Sc., and Ph.D. degrees from Universitat Politècnica de Catalunya (UPC), Spain. Currently, he is an Associate Professor at UPC Barcelona. Prior to joining UPC, he spent 5 years as a Senior Researcher with the Barcelona Supercomputing Center (BSC), Spain, and 15 months as a post-doctoral fellow with the International Computer Science Institute (ICSI), Berkeley. His research interests include high performance computer architectures, domain-specific accelerators and hardware–software co-design for future massively parallel systems.

Lluc Alvarez is a researcher at the Barcelona Supercomputing Center (BSC), where he is leading the group UNCORE: Cache Hierarchies and Interconnects. He received his B.Sc. degree from the Universitat de les Illes Balears (UIB) in 2006 and his M.Sc. and Ph.D. degrees from the UPC in 2009 and 2015. His main research interests are parallel architectures, cache and memory hierarchies, and accelerators for high-performance computing.