OpenPiton Optimizations Towards High Performance Manycores

Neiel Leyva*†, Alireza Monemi*, Noelia Oliete-Escuín*†, Guillem López-Paradís*†,
Xabier Abancens*, Jonathan Balkind§, Enrique Vallejo‡, Miquel Moretó*†, Lluc Alvarez*
*Barcelona Supercomputing Center, Barcelona, Spain. †Universitat Politècnica de Catalunya, Barcelona, Spain
‡Universidad de Cantabria, Santander, Spain. §UC Santa Barbara, California, United States
{neiel.leyva,alireza.monemi,noelia.oliete,guillem.lopez,xabier.abancens,miquel.moreto,lluc.alvarez}@bsc.es
jbalkind@ucsb.edu,enrique.vallejo@unican.es

ABSTRACT

In recent years, numerous multicore RISC-V platforms have emerged. Within the RISC-V ecosystem, Networks-on-Chip (NoCs) such as OpenPiton are employed in designs that aim to scale to a large number of cores. This paper presents a set of extensions and optimizations to OpenPiton for high-performance manycores. The key contributions are enabling multiple memory controllers, supporting router bypassing and NoC concentration, and adding support for configurable cache sizes and cache block sizes. On a 64-core manycore architecture, these new features and optimizations provide a geometric mean speedup of 6.5x compared to the OpenPiton baseline.

CCS CONCEPTS

• **Computer systems organization → Interconnection architectures**; **Multicore architectures**; • **Hardware → Networking hardware**; **Buses and high-speed links**.

KEYWORDS

Network-on-chip, OpenPiton, high-performance computing

ACM Reference Format:
Neiel Leyva, Alireza Monemi, Noelia Oliete-Escuín, Guillem López-Paradís, Xabier Abancens, Jonathan Balkind, Enrique Vallejo, Miquel Moretó and Lluc Alvarez. 2023. OpenPiton Optimizations Towards High Performance Manycores. In *16th edition of International Workshop on Network on Chip Architectures (NoCArc '23), October 28, 2023, Toronto, ON, Canada.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3610396.3623265

1 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 modelling 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 [3], BlackParrot [16], and the ESP open SoC platform [12], while simpler crossbar communication is utilized in platforms such as lowRISC [5] and PULP [17], which primarily target applications for the Internet-of-Things (IoT) and put special emphasis on low power consumption.

OpenPiton has received a lot of attention as a development platform for creating manycore processors due to the benefits it provides. 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, as it focuses on a general userbase, OpenPiton's NoC and memory hierarchy do not include certain features and characteristics typically found in high-performance manycores.

This paper presents a set of extensions and optimizations to the NoC and the memory hierarchy of OpenPiton. These extensions and optimizations are specifically aimed at improving the performance of large-scale multicore and manycore architectures. The key contributions are (i) adding the capability to including multiple memory controllers in the chip to increase the memory bandwidth of the system, (ii) adding router bypassing and NoC concentration features to reduce the latency of core communications and data transfers inside the NoC, and (iii) adding support for configurable cache sizes and cache block sizes in the cache hierarchy to improve its efficiency. Compared to the OpenPiton baseline, the combination of these new features and optimizations provides a geometric mean speedup of 6.5x on a 64-core manycore architecture.

2 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 [4, 9, 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 [2].

NoCArc '23, October 28, 2023, Toronto, ON, Canada

https://doi.org/10.1145/3610396.3623265

2.1 OpenPiton and RISC-V Architectures

BYOC [2] enables implementation with different architectures using the Transaction-Response Interface (TRI). This interface enables connecting different types of cores to OpenPiton. TRI supports cores with or without private cache levels, a configuration that has been followed in the integration of RISC-V cores.

Under TRI, cores with private caches must have an instruction cache, a write-through data cache, and (for application-class cores) an MMU. The MMU must be compliant with the specific core architecture to perform address translation. Cores without private cache levels nor MMU can be directly connected to the TRI, though such cores are not typically composed into cache-coherent manycores.

2.2 OpenPiton Cache Hierarchy

The OpenPiton framework provides a slice of the shared L2 cache and a private cache level named L1.5 per tile. The shared L2 cache serves as the point of coherence and it implements a directory-based MESI coherence protocol. The default L2 cache configuration is 64KB per slice, 4-way set-associative with 64B cache blocks.

The L1.5 cache is a Physically Indexed Physically Tagged (PIPT) write-back cache connected to the L2 cache through 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. The default L1.5 cache configuration is 8KB capacity, 4-way set-associative with 16B cache blocks.

2.3 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 addition, OpenPiton uses a simplistic simulation model for the memory controller in which all the memory requests (read and write) are served with a fixed latency of a single clock cycle.

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.

2.4 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.

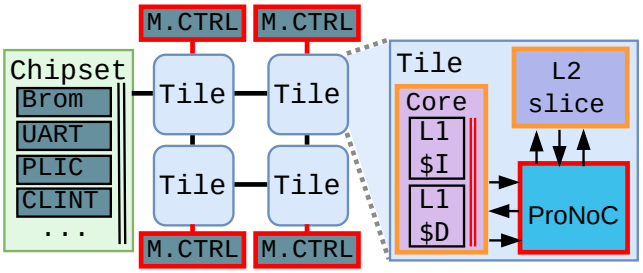


Figure 1: OpenPiton architecture upgraded with new features. New modules are marked in red and optimized modules are marked in orange.

2.5 OpenPiton and HPC Requirements

Compared with commercially available manycore architectures, 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. Improving cache sizes and block sizes also improves performance.

NoC latency: This communication delay can severely impact performance, necessitating low-latency NoC design for large systems. Employing multi-hop bypass and increasing NoC concentration are techniques to decrease data transmission hops and lower average communication latency.

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

3 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. They are described next.

3.1 Adapted Core Tile Using BYOC

Following the BYOC [2] methodology, we first change the core tile to explore OpenPiton without performance decline or compatibility issues arising from the use of foreign caches or cache modifications. 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) in the original OpenPiton. In our core tile we remove the the L1 data cache of OpenPiton and we use its L1.5 cache directly as the L1 data cache.

3.1.1 Core. The core tile incorporates a core called DVINO [6]. 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.

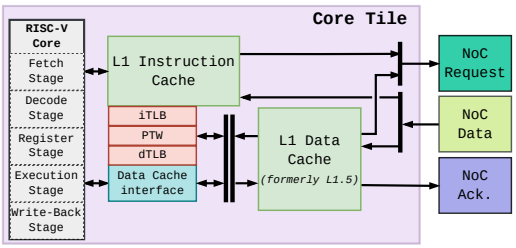


Figure 2: Architecture of the core tile using BYOC.

3.1.2 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. The instruction cache has a direct link to the L2 cache via traffic arbitrators. This is different than the default OpenPiton, where instructions are not cached in the L1.5 cache but the instruction requests are still sent through the L1.5 cache.

3.1.3 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.

3.1.4 Data Cache Interface. This module is the primary glue logic responsible for directly connecting the RISC-V 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 needs to be 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.

3.1.5 NoC interface. The modified interface allows direct instruction cache access to routers. An arbiter in the *NoC request* encoder prioritizes instruction cache requests, while another arbiter in the *NoC data* decoder manages deliveries between instruction and data caches. Instruction memory blocks solely reach the instruction cache, while other responses target the data cache.

3.2 Enabling Multiple Memory Controllers

The first optimization we propose is the capability 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 memory controllers that are directly connected to the tiles at the edges of the mesh. The memory controllers are equally distributed across the edges of the mesh (north, south, east, and west). Furthermore, the L2 module is modified to route main memory requests to the memory controller, taking into account the minimum Manhattan distance. Each memory controller is assigned an equal number of L2 modules to manage. Moreover, an input buffer module with a configurable pipeline latency is integrated into the memory model for a more realistic memory controller simulation. This enables studying the impact of different memory latencies on system performance.

3.3 Optimization of NoC Routers

The second optimization we perform is the integration of ProNoC [15] routers into OpenPiton. Interconnection latency is a significant factor that can impact overall performance, and it is essential to keep it as low as possible. 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.

3.4 Improved Cache Configurability

The third optimization consists of improving the cache configurability of OpenPiton. In particular, we add the capability to automatically configure the cache sizes and the cache block sizes, so we can explore the impact of these two parameters and find optimal configurations for different design targets and constraints.

3.4.1 Cache sizes. OpenPiton offers cache size configuration flags that can be specified when building the model. The sizes of the SRAMs, the block indexing logic, and the tag selection logic of the caches adapt to the specified cache sizes. However, the cache replacement logic and the L2 address interleaving logic require manual adjustment in OpenPiton when changing the cache sizes, or else there will be unbalanced utilization of cache ways and L2 slices. To address this issue, we modify the cache replacement logic and the L2 address interleaving logic of the OpenPiton caches so that they automatically adapt to the cache size.

3.4.2 Cache Block Sizes. OpenPiton uses different cache block sizes in its cache hierarchy. The L1 instruction cache uses 32B cache blocks, the L1 data cache and the L1.5 cache use 16B cache blocks, 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 use of cache block sizes of 16B or 64B in the L1 data cache and the L1.5 cache. Implementing this feature requires adjusting the cache pipelines and the NoC channels.

4 EXPERIMENTAL METHODOLOGY

In this section, we provide a comprehensive overview of the tools, benchmarks and development platforms employed to evaluate our work, as well as the methodology used to accelerate RTL simulations.

4.1 Benchmarks

We use a set of bare-metal benchmarks to evaluate the proposed optimizations. These benchmarks are selected from the RISC-V tests [7], the LMbench [13] and the NAS Parallel Benchmarks [1] suites. Table 1 shows the benchmarks categorized into three main groups.

Group *a)* comprises the four kernels of the Stream benchmark, 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.

Table 1: 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 *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.

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.

4.2 Simulation Tools

We use Questasim-64 2020.4 for simulation. Our design is compatible with Verilator 4.03, which allows the execution of multiple simulations in parallel without license restrictions. To generate the results we use Verilator with Metro-MPI [11], 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.

5 EVALUATION RESULTS

This section delves into the results derived from a range of experiments focused on the design space exploration of OpenPiton and the newly added features for high-performance manycores.

All experiments take as reference the following *default* configuration of OpenPiton: 8KB of L1 data cache, 16 KB of L1 instruction cache, 64KB of L2 cache per slice, 16B memory blocks, 1 memory controller, and OpenPiton routers. These 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.

5.1 Multiple Memory Controllers Exploration

Figure 3 shows the kernel execution time speed-up when varying the number of memory controllers. The x-axis shows the number of memory controllers ranging from 1 to 16, and the y-axis shows the speed-up over the baseline configuration. 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.

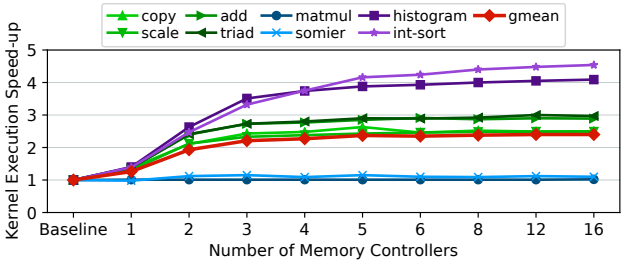
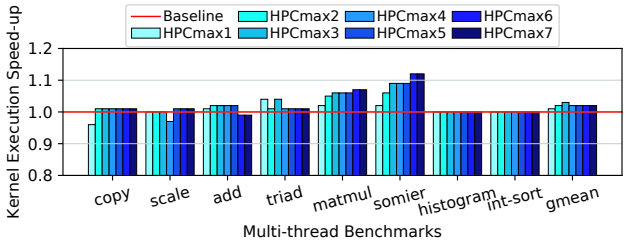
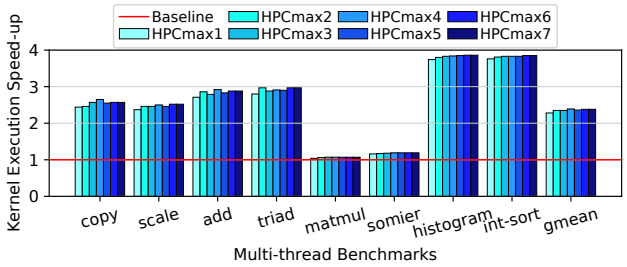


Figure 3: Speed-up with multiple memory controllers.



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



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

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

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.

5.2 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 4 presents the results. The baseline 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

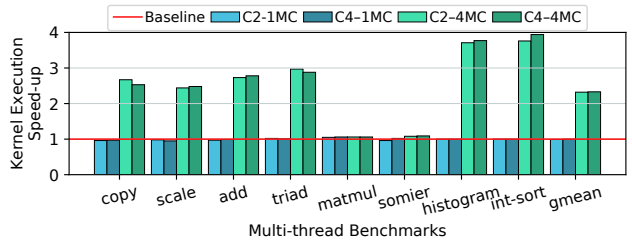


Figure 5: Speed-up of NoC concentration 2 and 4 (C2/C4) with 1 and 4 memory controllers (1MC/4MC).

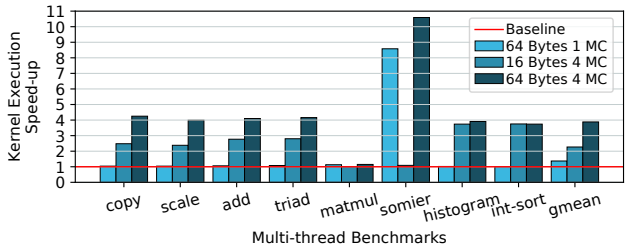


Figure 6: Speed-up of 16- and 64-Bytes cache blocks with 1 and 4 memory controllers (1MC/4MC).

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 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 Firgure 4a, 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 4b. In this case, increasing HPC_{max} provides performance benefits in all the benchmarks, up to 17% in *triad* with HPC_{max} 7 compared to HPC_{max} 1.

5.3 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 5 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)*.

5.4 Cache Block Size Exploration

Figure 6 illustrates the speed-up achieved by increasing the L1 cache block size from 16 to 64 Bytes. Furthermore, it highlights the impact of enlarging the memory block size in two distinct scenarios, wherein the number of memory controllers is varied. The average speed-up obtained with 64B cache blocks is 1.3x with 1 memory controller and 3.8x with 4 memory controllers.

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 1 memory controller the speed-up achieved by enlarging the cache blocks to 64B is negligible because the main bottleneck remains in the memory bandwidth. By contrast, using 4 memory controllers with 16B cache blocks provides good speed-ups of 2.3x to 2.8x, and enlarging the cache block size to 64B with 4 memory controllers further increases the speed-ups to more than 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.

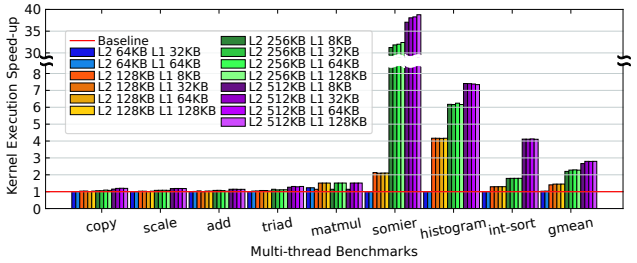
Group *c)* shows the least benefit when enlarging the cache blocks, both with 1 and 4 memory controllers. These benchmarks have very poor spatial locality, as they are dominated by sparse atomic memory operations to large data structures, which are served by the shared L2 cache. Thus, these atomic operations do not show any benefit when increasing the private L1 cache block size.

5.5 Cache Size Exploration

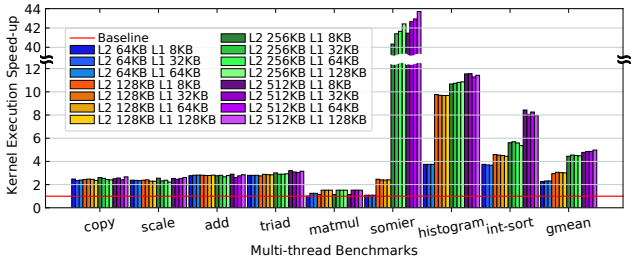
Figure 7 depicts the exploration results of using larger caches within the system. In this experiment we explore different cache configurations with L1 cache capacities ranging from 8KB to 128KB and L2 cache capacities ranging from 64KB to 512KB. To observe the impact of large caches we perform simulations 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 2.8x and 4.9x 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



(a) Different cache sizes and 1 memory controller.



(b) Different cache sizes and 4 memory controllers.

Figure 7: Speed-up of different combinations of L1 data cache and L2 cache sizes with 1 and 4 memory controllers.

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.

5.6 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 2. The configurations are selected after analyzing the experiments previously conducted in this section, focusing on those that provide the largest performance benefits.

Figure 8 presents the results obtained from incorporating all the features and optimizations into OpenPiton. On average, the speed-up achieved by implementing large caches (*Large*) and enlarging the cache block size (*Large+Block*) from 16 to 64 Bytes is quite similar, around 2.8x. By adding more memory controllers (*Large+Ctrl*), the average speed-up increases to 4.8x. By combining these three features we can build a system with 32KB of L1 cache, 512KB of L2 cache, 64B cache blocks and 4 memory controllers (*Large+Block+Ctrl*), which achieves an average speed-up of 6.3x. On top of this configuration, 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 6.5x over the baseline.

Table 2: Tested configurations with new features.

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

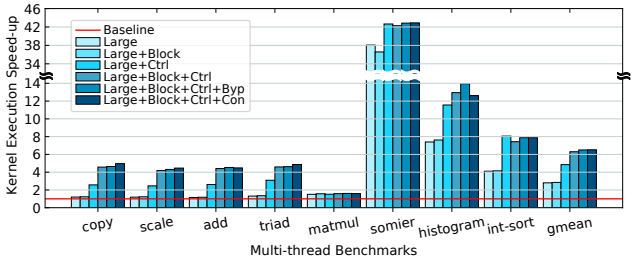


Figure 8: Speed-up of different configurations with the newly added features against the OpenPiton baseline.

6 RELATED WORK

BlackParrot [16] is a 64-bit RISC-V multi-core processor featuring a cache hierarchy with private L1 and shared L2 cache levels. It employs VI, MSI, and MESI cache coherency protocols and introduces specific-purpose tiles to enhance the L2 cache. Each L2 tile provides an additional slice of the L2 cache. Similar to OpenPiton, both share multiple memory controllers, a 2-D mesh using three physical channels NoC routers and lack virtual channeling. Unlike OpenPiton, BlackParrot integrates a network to connect memory controllers, and each L2 slice accesses this network. While system scalability isn't detailed, a taped-out quad-core design exists.

The PULP platform [17] is a low-power SoC targeting IoT applications. It features a RISC-V core and an octa-core accelerator. Its cache hierarchy includes a 512KB L2 shared cache split into four banks. The RISC-V core lacks a private cache level but has two 32 KB banks for program stack and private data. The octa-core accelerator directly accesses the L2 cache and a scratch cache. PULP employs AXI-based interconnections for core and memory communication. Unlike OpenPiton, PULP is not focused on manycore systems.

Agiler [8] is a RISC-V multi-core architecture designed for heterogeneous systems, featuring two types of processing elements. The main type comprises a quad-core using AXI-based interconnection for communication between cores, shared instruction cache, and memory controller. The second type consists of accelerators with distinct tile architectures interconnected via mesh routers. This includes a 64-bit dual-core and a 32-bit quad-core RISC-V architecture, both internally linked with AXI. Tiles are mapped to memory regions, and tasks are allocated by loading data and instructions into corresponding memory spaces during compilation. In contrast, in OpenPiton, each tile works within the same memory space.

Open ESP (Open Embedded Systems Platform) [12] is an open-source framework for accelerator-rich SoC prototyping. In addition

to providing tools and libraries to create software applications, it features a modular FPGA SoC architecture using tiles interconnected via a 2D-Mesh NoC with look-ahead routing. The four tile types (processor, accelerator, memory, and auxiliary) offer diverse functionalities. The processor tiles house a dual-cache core with MESI-coherent L2 cache, the accelerator tiles facilitate efficient data exchange with memory, the memory tiles include a shared LLC slice and a memory controller port, and the auxiliary tiles manage peripherals. Similar to OpenPiton, ESP supports multiple memory controllers and coherence protocols, enhancing scalability in manycores.

7 CONCLUSIONS

In the last years, the advent of RISC-V has caused the proliferation of academic and industrial muticore processor prototypes. Given the importance of NoCs in large-scale multicores and manycores, development platforms like OpenPiton have received a lot of attention. However, the NoC and the memory hierarchy of OpenPiton do not include key features usually present 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, we add router bypassing and NoC concentration features, and we add support for configurable cache sizes and cache block sizes in the cache hierarchy. 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 speedup of 6.5x compared to the OpenPiton baseline.

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, PID2019-105660RB-C22 and TED2021-131176B-I00), by the Generalitat de Catalunya (contract 2021-SGR-00763), by the European Union within the framework of the ERDF of Catalonia 2014-2020 under the DRAC project [001-P-001723] and by the European NextGenerationEU/PRTR, and by Lenovo-BSC Contract-Framework Contract (2022). The MEEP Project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 946002. The JU receives support from the European Union's Horizon 2020 research and innovation program and Spain, Croatia and Turkey. G. López-Paradís has been supported by the Generalitat de Catalunya through a FI fellowship 2021FI-B00994.

REFERENCES

[1] D.H. Bailey, E. Barszcz, J.T. Barton, D.S. Browning, R.L. Carter, L. Dagum, R.A. Fatoohi, P.O. Frederickson, T.A. Lasinski, R.S. Schreiber, H.D. Simon, V. Venkatakrishnan, and S.K. Weeratunga. 1991. The Nas Parallel Benchmarks. *Int. J. High Perform. Comput. Appl.* 5, 3 (sep 1991), 63–73. https://doi.org/10.1177/10943420910050306

[2] Jonathan Balkind, Katie Lim, Michael Schaffner, Fei Gao, Grigory Chirkov, Ang Li, Alexey Lavrov, Tri M. Nguyen, Yaosheng Fu, Florian Zaruba, Kunal Gulati, Luca Benini, and David Wentzlaff. 2020. BYOC: A "Bring Your Own Core" Framework for Heterogeneous-ISA Research. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 699–714. https://doi.org/10.1145/3373376.3378479

[3] Jonathan Balkind, Michael McKeown, Yaosheng Fu, Tri Minh Nguyen, Yanqi Zhou, Alexey Lavrov, Mohammad Shahrad, Adi Fuchs, Samuel Payne, Xiaohua Liang, Matthew Matl, and David Wentzlaff. 2016. OpenPiton: An Open Source Manycore Research Framework. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2016, Atlanta, GA, USA, April 2-6, 2016*, Tom Conte and Yuanyuan Zhou (Eds.). ACM, 217–232. https://doi.org/10.1145/2872362.2872414

[4] Jonathan Balkind, Michael Schaffner, Katie Lim, Florian Zaruba, Fei Gao, Jinzheng Tu, David Wentzlaff, and Luca Benini. 2019. 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, CARRV 2019, Phoenix, AZ, USA, June 22, 2019*. https://carrv.github.io/2019/papers/carrv2019_paper_12.pdf

[5] Alex Bradbury, Gavin Robert Ferris, and Robert D. Mullins. 2014. Tagged memory and minion cores in the lowRISC SoC.

[6] Guillem Cabo, Gerard Candón, Xavier Carril, Max Doblas, Marc Domínguez, Alberto González, César Hernández, Víctor Jiménez, Vatistas Kostalampros, Rubén Langarita, Neiél Leyva, Guillem López-Paradís, Jonnatan Mendoza, Francesco Minervini, Julián Pavón, Cristóbal Ramírez, Narcís Rodas, Enrico Reggiani, Mario Rodríguez, Carlos Rojas, Abraham Ruiz, Víctor Soria, Alejandro Suanes, Iván Vargas, Roger Figueras, Pau Fontova, Joan Marimon, Víctor Montabes, Adrián Cristal, Carles Hernández, Ricardo Martínez, Miquel Moretó, Francesc Moll, Oscar Palomar, Marco A. Ramírez, Antonio Rubio, Jordi Sacristán, Francesc Serra-Graells, Nehir Sonmez, Lluís Terés, Osman Unsal, Mateo Valero, and Luís Villa. 2022. DVINO: A RISC-V Vector Processor Implemented in 65nm Technology. In *2022 37th Conference on Design of Circuits and Integrated Circuits (DCIS)*. 1–6. https://doi.org/10.1109/DCIS55711.2022.9970128

[7] RISC-V International. 2012. riscv-tests. https://github.com/riscv-software-src/riscv-tests. Accessed Aug. 2023.

[8] Ahmed Kamaleldin and Diana Göhringer. 2022. AGILER: An Adaptive Heterogeneous Tile-Based Many-Core Architecture for RISC-V Processors. *IEEE Access* 10 (2022), 43895–43913. https://doi.org/10.1109/ACCESS.2022.3168686

[9] Neiel I. Leyva-Santes, Ivan Pérez, César A. Hernández-Calderón, Enrique Vallejo, Miquel Moretó, Ramón Beivide, Marco A. Ramírez-Salinas, and Luis A. Villa-Vargas. 2019. Lagarto I RISC-V Multi-core: Research Challenges to Build and Integrate a Network-on-Chip. In *Supercomputing*, Moisés Torres and Jaime Klapp (Eds.). Springer International Publishing, Cham, 237–248.

[10] Katie Lim, Jonathan Balkind, and David Wentzlaff. 2018. JuxtaPiton: Enabling Heterogeneous-ISA Research with RISC-V and SPARC FPGA Soft-cores. https://doi.org/10.48550/ARXIV.1811.08091

[11] Guillem López-Paradís, Brian Li, Adrià Armejach, Wallentowitzm Stefan, Miquel Moretó, and Jonathan Balkind. 2023. Fast Behavioural RTL Simulation of 10B Transistor SoC Designs with Metro-MPI. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE'23)*.

[12] Paolo Mantovani, Davide Giri, Giuseppe Di Guglielmo, Luca Piccolboni, Joseph Zuckerman, Emilio G. Cota, Michele Petracca, Christian Pilato, and Luca P. Carloni. 2020. Agile SoC Development with Open ESP. In *Proceedings of the 39th International Conference on Computer-Aided Design* (Virtual Event, USA) *(ICCAD '20)*. Association for Computing Machinery, New York, NY, USA, Article 96, 9 pages. https://doi.org/10.1145/3400302.3415753

[13] Larry McVoy and Carl Staelin. 1996. Lmbench: Portable Tools for Performance Analysis. In *Proceedings of the 1996 Annual Conference on USENIX Annual Technical Conference* (San Diego, CA) *(ATEC '96)*. USENIX Association, USA, 23.

[14] Alireza Monemi, Iván Pérez, Neiel Leyva, Enrique Vallejo, Ramón Beivide, and Miquel Moretó. 2021. PlugSMART: A Pluggable Open-Source Module to Implement Multihop Bypass in Networks-on-Chip. In *Proceedings of the 15th IEEE/ACM International Symposium on Networks-on-Chip* (Virtual Event) *(NOCS '21)*. Association for Computing Machinery, New York, NY, USA, 41–48. https://doi.org/10.1145/3479876.3481601

[15] Alireza Monemi, Jia Wei Tang, Maurizio Palesi, and Muhammad N. Marsono. 2017. ProNoC: A low latency network-on-chip based many-core system-on-chip prototyping platform. *Microprocessors and Microsystems* 54 (2017), 60–74. https://doi.org/10.1016/j.micpro.2017.08.007

[16] Daniel Petrisko, Farzam Gilani, Mark Wyse, Dai Cheol Jung, Scott Davidson, Paul Gao, Chun Zhao, Zahra Azad, Sadullah Canakci, Bandhav Veluri, Tavio Guarino, Ajay Joshi, Mark Oskin, and Michael Bedford Taylor. 2020. BlackParrot: An Agile Open-Source RISC-V Multicore for Accelerator SoCs. *IEEE Micro* 40, 4 (2020), 93–102. https://doi.org/10.1109/MM.2020.2996145

[17] Antonio Pullini, Davide Rossi, Igor Loi, Giuseppe Tagliavini, and Luca Benini. 2019. Mr.Wolf: An Energy-Precision Scalable Parallel Ultra Low Power SoC for IoT Edge Processing. *IEEE Journal of Solid-State Circuits* 54, 7 (2019), 1970–1981. https://doi.org/10.1109/JSSC.2019.2912307