MASTER IN HIGH PERFORMANCE COMPUTING

Comparative Performance Analysis of RISC-V Architectures

Exploring Efficiency at the Core

Supervisors:

Xavier TERUEL (BSC)

Irina DAVYDENKOVA (ICTP)

Candidate:

Regina Mumbi GACHOMBA

10th edition

2023–2024

Acknowledgements

First and foremost, I would like to thank ICTP and the entire MHPC faculty for financing my education and supporting my journey towards building a career in the field of HPC. Their generous backing has been instrumental in enhancing my skill set.

My heartfelt gratitude also goes to the BePPP team at BSC for their steady support and constructive feedback. Your guidance has been essential in shaping this work, and I appreciate the many insights and helpful nudges that kept me moving in the right direction. Special thanks to the Mobile and embedded-based HPC team who were always there to assist whenever the HCA cluster presented challenges.

To my family and friends: thank you for the unwavering moral support and for graciously enduring my technical ramblings. I know most of it sounded like a foreign language, but your ability to listen attentively (or at least nod convincingly) has kept me grounded throughout this journey.

Finally, drawing on some wisdom from Snoop Dogg, I would like to thank myself. For staying committed to this project, for believing in my own capabilities, and for persevering through the challenges, with determination and curiosity to drive me forward.

This work is co-financed by the Barcelona Zettascale Laboratory with reference **REGAGE22e00058408992** which is financed by the Ministry for Digital Transformation and Public Services, within the framework of the Resilience and Recovery Facility - and the European Union - NextGenerationEU.

Abstract

The rise of RISC-V marks a significant evolution in HPC, driven by its open-source Instruction Set Architecture (ISA), which provides unmatched modularity and customisation compared to proprietary architectures such as x86 and ARM. This study, conducted as part of the Barcelona Zettascale Laboratory (BZL) initiative, presents a comparative performance evaluation of two notable RISC-V implementations: the SiFive HiFive Unmatched and Milk-V Pioneer. Using the High-Performance Computing Challenge (HPCC) benchmark suite, the analysis explores performance under MPI and vectorised configurations, examining both intranode and internode workloads. Key metrics such as floating-point operations per cycle and memory bandwidth are assessed alongside architecture-specific features, including vector processing units (VPUs) and memory hierarchies. The findings illuminate distinct scaling patterns and performance bottlenecks, underscoring RISC-V's potential for enabling energy-efficient and sustainable HPC solutions. These insights aim to guide future optimisations in computational throughput and support the integration of RISC-V into next-generation HPC systems prioritising energy efficiency and computational performance.

Contents

1 Introduction **3**

1.1 Project Background 3

1.2 Context and Scope 4

1.3 Objectives 7

1.4 Theoretical Background 8

1.4.1 Performance Metrics 8

1.4.2 Roofline Model 8

1.4.3 Per Second to Per Cycle 9

1.4.4 POP Methodology and Efficiency Metrics 11

2 Methodology **13**

2.1 Platform Exploration 13

2.2 Environment Description 14

2.2.1 RISC-V Board Features & Specifications 14

2.2.2 Single-core Performance Model 16

2.2.3 Benchmark Applications 18

2.2.4 Visualisation and Tracing Tools 21

2.2.5 Performance Analysis Tools 22

2.3 Benchmark Setup 23

2.3.1 Compilation 23

2.3.2 Execution 24

3 Performance Analysis **26**

3.1 Findings & Discussion 26

3.1.1 Intranode Performance 26

3.1.2 Internode Performance 32

3.1.3 Results analysis within the Roofline Model 36

3.1.4 Vectorisation in RISC-V Architectures 38

4 Conclusion **41**

5 Suggestions and Future Work **43**

A Annex **45**

A.1 HPCC Benchmark Makefile 45

A.2 HPL Input File 47

Introduction

Originally developed at UC Berkeley, RISC-V was created with the goal of enabling innovation and adaptability in processor design, making it well suited for applications across a wide range of computing domains, from low-power embedded systems to more complex HPC environments.

The open-source nature of RISC-V allows developers and organisations to tailor processor designs specifically to their needs, which is particularly valuable in fields like HPC, where efficiency and customisation are critical. This modular design of RISC-V also enables the inclusion of optional extensions, such as VPUs, which can provide additional computational power for data-parallel tasks in scientific computing and machine learning.

1.1 Project Background

The Barcelona Zettascale Project (BZL) exemplifies Europe's commitment to leveraging RISC-V to advance large-scale computing. Building on the achievements of earlier initiatives such as the European Processor Initiative (EPI) and the MareNostrum Experimental Exascale Platform (MEEP), the project is part of a cohesive strategy to enhance Europe's technological sovereignty and foster innovation in open hardware design.

On the research side, initiatives like EPI and MEEP have laid the groundwork for Europe's processor design capabilities. EPI introduced a hybrid architecture combining ARM-based cores with RISC-V accelerators. Meanwhile, MEEP played a pivotal role in the development of the Lagarto family of RISC-V processors [1]. These efforts have collectively contributed to Europe's ability to design processors independent of proprietary, non-European intellectual property (IP).

At the same time, RISC-V's commercial ecosystem has rapidly evolved, complementing these research efforts by demonstrating the architecture's viability across a wide range of applications. Currently, RISC-V is being integrated into various products supported by a growing ecosystem of compilers, software libraries, and development tools.

Both in research and commercial contexts, RISC-V's adoption reflects a shared ambition: to establish the architecture as a viable solution for the challenges of modern computing, from data-centric AI workloads to energy-efficient HPC. By

uniting research initiatives with commercial developments, this thesis seeks to bridge the gap between experimental advancements and market-ready applications. It builds on the strengths of earlier efforts, leveraging the growing commercial maturity of RISC-V to deliver impactful solutions.

1.2 Context and Scope

As part of this initiative, the BZL project is advancing the Lagarto processor family. The project targets Lagarto-Ox, a processor designed to support more complex applications, including booting a full Linux distribution. Available in multiple configurations—single-core scalar, single-core vector, and dual-core vector—Lagarto-Ox represents a key step toward meeting the diverse demands of modern workloads.

In parallel, the BZL project is building a comprehensive software ecosystem tailored for HPC. This ecosystem includes operating systems, compilers, parallel runtimes, and essential numerical libraries to support both traditional HPC applications and emerging workloads that integrate Artificial Intelligence with simulations rooted in the "first principles" of HPC. This thesis contributes to this broader effort by evaluating the viability of the proposed ecosystem.

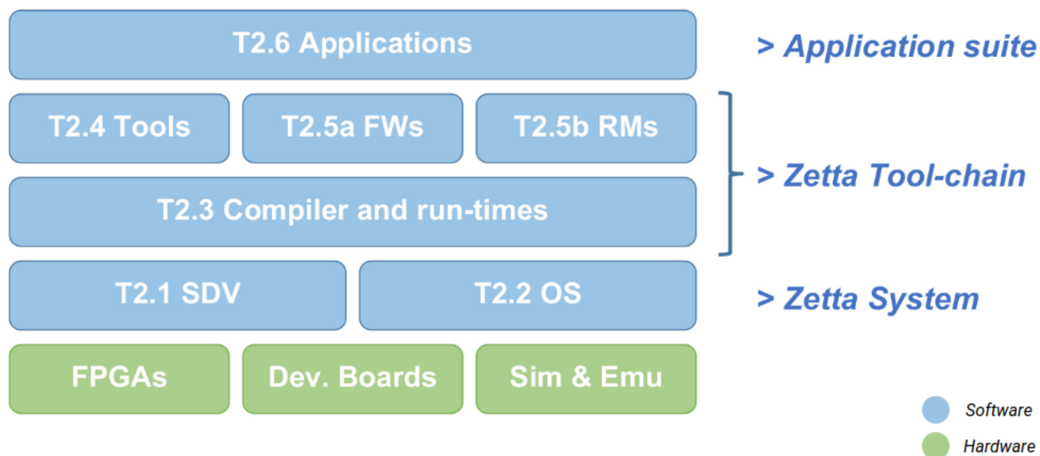


Figure 1.1: **Project Tasks (layers)**

Figure 1.1 shows how the main tasks of the project are compartmentalised. The software component of the BZL project is divided into several key areas:

Software Development Vehicle (SDV)

The SDV serves as a platform for the development and testing of software components. It includes multiple FPGA and development boards based on RISC-V ISA that provide a flexible and heterogeneous environment for software testing and optimisation.

Operating Systems (OS)

The BZL project integrates a variety of Linux distributions that are compatible with the kernel, such as Fedora and Ubuntu. This ensures flexibility, allowing for adaptation across different HPC environments. The support of these OS distributions fosters a highly scalable software ecosystem.

Compilers and Runtimes

A critical aspect to achieving a performant and efficient architecture lies in well-optimised compiler support. BZL prioritises the advancement of compiler technologies specifically tailored to the RISC-V architecture. Although LLVM is highly adaptable to various programming languages, it has not been explored as extensively for RISC-V compared to the more established GCC [7]. Therefore, this analysis will seek to identify the strengths and shortcomings of the current state of LLVM on RISC-V, with the findings feeding back into the compiler team for further optimisations.

Tools

The section encompasses tools used for performance analysis such as Paraver, Extrae and Dimemas, which are responsible for providing performance information on various kernels and applications.

Frameworks

The project explores the use of traditional libraries and frameworks such as OpenMP, MPI and BLAS in the execution of applications based on multiple processes and distributed systems.

Resource Managers

Resource management is critical for orchestrating workloads across the system. Slurm is the primary resource manager used, handling job scheduling and resource allocation across the architecture. The project also explores complementary resource manager tools and libraries built at BSC such as Energy Aware Runtime (EAR) and Dynamic Load Balancing (DLB).

Application suite

At the topmost layer of the proposed ecosystem is the application suite, where specific tasks and the main contributions of this thesis are conducted. This layer is essential as it allows for the testing and benchmarking of applications directly on the chosen RISC-V architectures, namely the SiFive HiFive Unmatched and the Milk-V Pioneer boards. By running a comprehensive suite of benchmark applications on these architectures, this study provides insights into how future software and applications will perform on the evolving ecosystem.

1.3 Objectives

The core purpose of this analysis is to investigate the architectural features of different computing architectures, with a particular focus on RISC-V and their potential applications in HPC. By assessing the performance of these architectures, the study aims to understand which aspects contribute the most significantly to the overall efficiency, scalability, and suitability for HPC tasks. This study aims to pinpoint key strengths and weaknesses in current RISC-V designs, providing critical insights for advancing future HPC hardware that aligns with emerging demands for energy efficiency and high computational throughput.

Understanding which features of the RISC-V architecture, such as core configurations, cache hierarchies, and vector processing capabilities contribute most to performance in HPC applications, is critical for guiding the design of future hardware platforms. These insights will help determine how the RISC-V ISA can be adapted and emulated to meet the specific needs of the BZL project, which requires a balanced approach to computational power and energy efficiency. The co-primary objective of this thesis is to advocate for a shift in the HPC community's priorities, moving from a focus on maximising raw computational speed to prioritising energy efficiency and sustainable performance. This study aims to demonstrate that architectural efficiency, measured in computational performance relative to energy consumption, should be given more importance than sheer processing speed.

The findings obtained from this analysis will not only inform the development of optimised hardware solutions but will also contribute to the broader goal of enhancing the ongoing evolution of the RISC-V ecosystem for HPC. By identifying which architectural features are most beneficial for HPC, this research will help illustrate how RISC-V's adaptability and open-source foundation make it an ideal candidate for energy-conscious innovation in HPC, potentially serving as a model for the development of future energy-efficient architectures , thus paving the way for the integration of RISC-V into the next generation of supercomputing systems.

Through these objectives, this thesis aims to contribute both technical data and a philosophical shift in HPC design priorities, advocating for a future where the Green500 list gains as much, if not more, prominence than the traditional TOP500 ranking.

1.4 Theoretical Background

1.4.1 Performance Metrics

In performance analysis, metrics typically fall into the following main categories: response time, throughput, utilisation, reliability, and availability [2]. This thesis focuses on throughput metrics, as we utilise the HPCC benchmark suite. Throughput refers to the rate at which requests are processed by a system, usually measured in requests per unit of time. The maximum throughput achievable under ideal workload conditions is known as the system's nominal capacity. In the context of computer networks and memories, this nominal capacity is referred to as the bandwidth.

For compute-bound benchmarks like HPL, DGEMM, and FFT, throughput is measured in FLOP/s. On the other hand, for benchmarks involving data movement either from memory to the CPU or across nodes i.e. STREAM, Latency-Bandwidth and PTRANS, the throughput is reported in Bytes per second (B/s).

1.4.2 Roofline Model

In HPC, understanding the limitations and potential of an architecture requires a detailed analysis of both computational capabilities and memory bandwidth. The Roofline model, a performance analysis tool introduced by Williams et al. [3], offers a graphical framework to assess these aspects by relating computational throughput to memory bandwidth limitations. This model enables a comprehensive view of how efficiently a program is using the system given the constraints of the architecture, which is essential for interpreting benchmark results on various HPC platforms.

The roofline model is particularly relevant in this thesis because it provides a systematic way to evaluate the performance boundaries of the RISC-V architectures studied. For both architectures, the roofline model captures the relationship between computational intensity (measured as FLOP per byte) and achievable throughput (in FLOP per cycle) and highlights whether the system is limited by memory bandwidth or processing power. The compute roof, which is the upper horizontal line in the plot, represents each core's theoretical maximum FLOP/cycle. Below this roof lies the bandwidth-limited region, where achievable performance is constrained by memory bandwidth rather than computational power. This information is critical when

assessing architectures which differ in cache hierarchies, memory configurations, and computational resources.

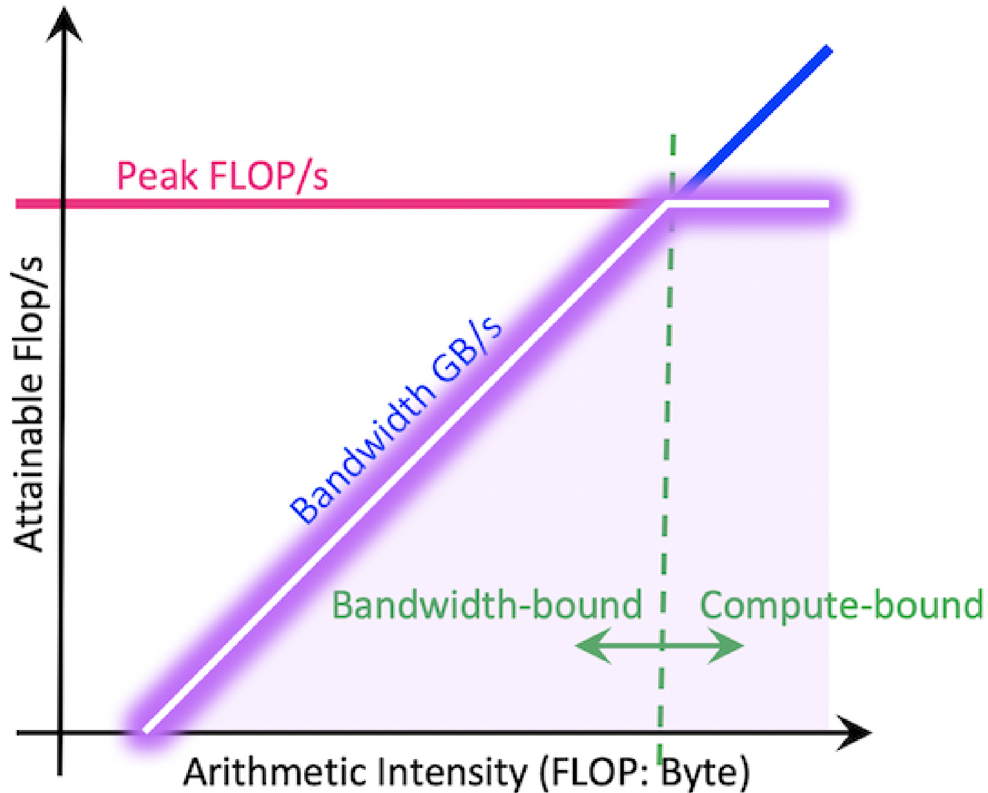


Figure 1.2: **Standard Roofline model**

Source: https://docs.nersc.gov/tools/performance/roofline/

1.4.3 Per Second to Per Cycle

The HPCC benchmark typically reports throughput in GFLOP/s and bandwidth in GB/s or MB/s. In this study, these metrics will be expressed per cycle. This approach isolates architectural efficiency from clock speed, allowing a fair comparison of different designs regardless of their operating frequency. Clock speeds can vary significantly across processors or systems, and focusing solely on raw performance numbers could make a processor with a higher clock speed appear superior, even if it is less efficient per cycle [6]. Per-cycle metrics allow for more accurate comparisons by revealing how much work a processor can accomplish in each cycle, offering insights into its intrinsic efficiency.

Expressing metrics per cycle provides a clearer understanding of a processor's computational efficiency per unit of time, independent of the additional power and thermal costs associated with higher clock speeds. It also allows us to assess a processor's performance based purely on its design, without the confounding influence of clock speed. For systems designed to scale, such as distributed computing environments, understanding per-cycle efficiency is essential for predicting how well the system will perform as additional resources (e.g., cores or nodes) are added.

In addition, per-cycle metrics offer predictive value when estimating the performance of new designs or configurations. By analysing efficiency per cycle, it becomes easier to anticipate the performance of future systems that may operate at different clock speeds or be configured with varying resources.

Given that :

$$Cycles = Duration \times Frequency$$

and :

$$FLOP/s = \frac{Num_FLOP}{Duration}$$

$$Bytes/s = \frac{Num_Bytes}{Duration}$$

then expressing the throughput and bandwidth per cycle requires that they are divided by the frequency :

$$FLOP/cycle = \frac{Num_FLOP}{Cycles} = \frac{Num_FLOP}{Duration \times Frequency} = \frac{FLOP/s}{Frequency}$$

$$Bytes/cycle = \frac{Num_Bytes}{Cycles} = \frac{Num_Bytes}{Duration \times Frequency} = \frac{Bytes/s}{Frequency}$$

1.4.4 POP Methodology and Efficiency Metrics

As efficiency is the prime focus of this study, it is crucial to establish an effective methodology to detail the performance analysis metrics. This project will use the methodology outlined by the Performance and Optimization Center of Excellence (POP CoE) [12].

POP has defined a methodology for the analysis of parallel codes to provide a quantitative way to measure the relative impact of the different factors inherent in parallelisation. The methodology uses a hierarchy of metrics where each one reflects a common cause of inefficiency in parallel programs. These metrics then allow comparison of parallel performance (e.g., over a range of thread/process counts, across different machines, or at different stages of optimisation and tuning) to identify which characteristics of the code contribute to inefficiency.

The metrics are then calculated as efficiencies between 0 and 1, with values closer to 1 being better. The hierarchy is multiplicative, meaning that each factor is computed as the product of its child metrics.

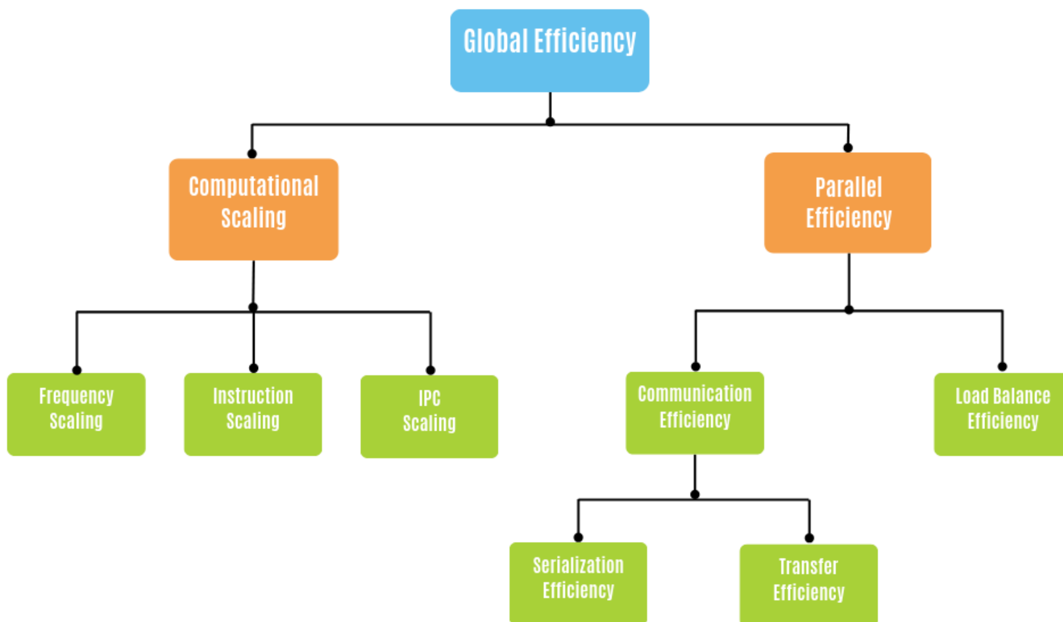
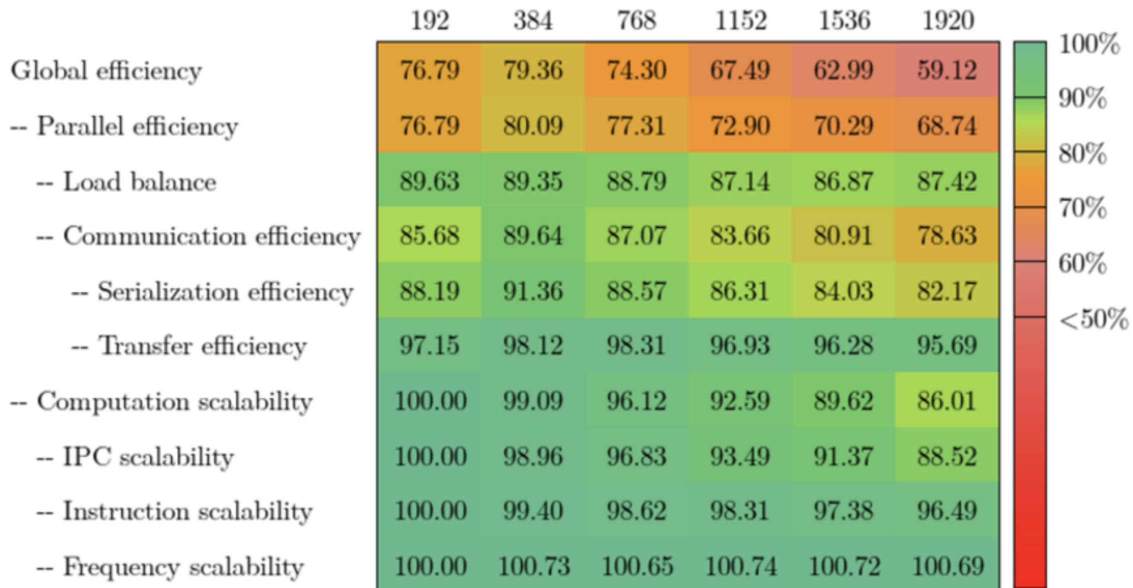


Figure 1.3: **Standard POP efficiency metrics.**

Figure 1.4 is an example of an output that we get using the POP methodology. This is from Alya , a simulation program used for computational fluid dynamics (CFD) problems, specifically simulating spray combustion [13]. Each row represents a metric that influences performance, and each column corresponds to the number of processing elements used (can be cores, CPUs or nodes). The color-coding indicates the efficiency range, with green indicating high efficiency and red indicating low efficiency.



	192	384	768	1152	1536	1920
Global efficiency	76.79	79.36	74.30	67.49	62.99	59.12
-- Parallel efficiency	76.79	80.09	77.31	72.90	70.29	68.74
-- Load balance	89.63	89.35	88.79	87.14	86.87	87.42
-- Communication efficiency	85.68	89.64	87.07	83.66	80.91	78.63
-- Serialization efficiency	88.19	91.36	88.57	86.31	84.03	82.17
-- Transfer efficiency	97.15	98.12	98.31	96.93	96.28	95.69
-- Computation scalability	100.00	99.09	96.12	92.59	89.62	86.01
-- IPC scalability	100.00	98.96	96.83	93.49	91.37	88.52
-- Instruction scalability	100.00	99.40	98.62	98.31	97.38	96.49
-- Frequency scalability	100.00	100.73	100.65	100.74	100.72	100.69

Figure 1.4: **PoP report for Alya (A simulation program for CFD problems)**

Methodology

2.1 Platform Exploration

The initial study focused on exploring available platforms and determining which hardware architectures were suitable for testing and benchmarking within the context of the project. At BSC, we have the Heterogeneous Computing Architecture (HCA) cluster, which hosts a variety of RISC-V boards. These boards include the Microchip PolarFire, SiFive Unmatched, Allwinner D1, and Milk V Pioneer [20].

The decision to select the SiFive Unmatched and Milk V Pioneer boards for further exploration was guided by several critical factors. Primarily, we prioritised the ability to scale, opting for the boards with the highest number of nodes, cores, and RAM available within the cluster. This scalability was essential for the kind of performance benchmarking and testing we intended to conduct, as it allowed for more comprehensive and representative results when evaluating the systems' capabilities. The HCA cluster has seven SiFive Unmatched boards and four Milk V Pioneer boards.

In addition, system and operating system compatibility were also critical considerations. Both the SiFive Unmatched and Milk-V Pioneer boards operate on Linux kernel versions that support hardware counter collection, enabling the use of performance analysis tools like PAPI. In contrast, some other boards use kernel versions without PAPI support, limiting detailed performance measurement. This compatibility ensured that the testing environment could effectively leverage the modules and tools required for in-depth performance analysis.

Thus, the combination of scalability and system compatibility made the SiFive Unmatched and Milk V Pioneer boards the most viable options for this study, aligning with the specific requirements of the project.

2.2 Environment Description

In this section, we discuss all the components that make up the environment for the analysis, including the boards, tools and software used for the purpose of benchmarking, visualisation, and obtaining efficiency metrics.

2.2.1 RISC-V Board Features & Specifications

We begin by highlighting the main features of the two boards used in the performance analysis.

SiFive HiFive Unmatched

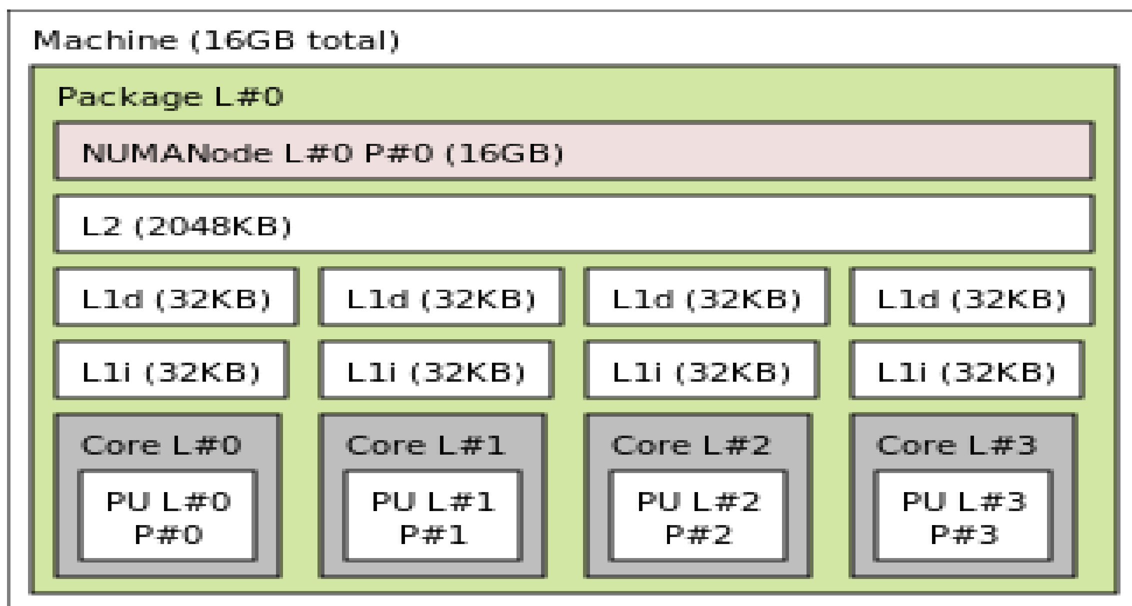


Figure 2.1: SiFive Unmatched System Architecture

Operating Frequency	1.2 GHz
L1 Cache	32KB I-Cache and 32KB D-Cache
L2 Cache	2 MB
VPU	No
Cores per node	4
DRAM	16 GB

Table 2.1: SiFive Unmatched System Specifications

Figure 2.1 is a hwloc output showing the Unmatched board which has a single CPU package with 16 GB of memory in one NUMA node. It features a shared 2 MB L2 cache and, for each core, dedicated 32 KB L1 instruction (L1i) and 32 KB L1 data

(L1d) caches. The package contains four cores, each with a single processing unit, totaling four logical processors. Unmatched houses the SiFive FU740 SoC, which includes a heterogeneous five-core architecture: four U74 application cores and one S7 real-time core.

Milk V Pioneer

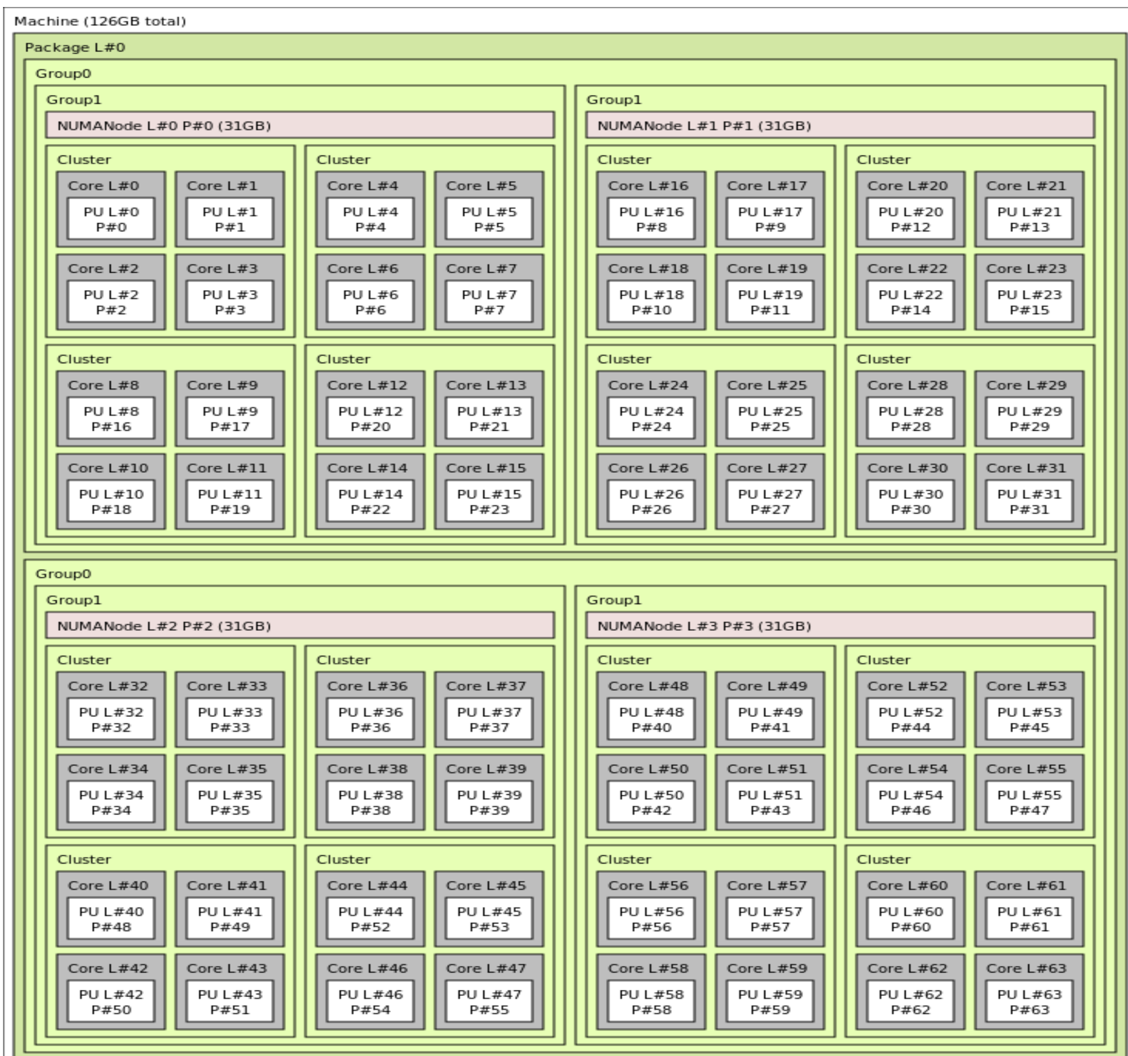


Figure 2.2: Milk V Pioneer System Architecture

Operating Frequency	2 GHz
L1 Cache	64KB I-Cache and 64KB D-Cache
L2 Cache	1 MB / cluster
L3 Cache	64 MB
VPU	Yes
Cores per node	64
DRAM	128 GB

Table 2.2: Milk V Pioneer System Specifications

The hardware configuration of the Milk-V Pioneer system, as depicted in Figure 2.2 shows that the board is configured with a single CPU package containing four NUMA nodes, each with 31 GB of memory for a total of 126 GB. These NUMA nodes are organised into four groups with 16 clusters in total. Each cluster houses two cores, resulting in 32 cores overall. Each core supports two processing units, providing 64 logical processors in total. The Milk-V is powered by the SOPHON SG2042 processor.

2.2.2 Single-core Performance Model

The Roofline models shown below in Figure 2.3 and Figure 2.4 are based on the theoretical single-core performance for each architecture that we are testing. This approach emphasises intrinsic computational and memory-bound characteristics without multicore interaction effects, which is essential for understanding baseline performance.

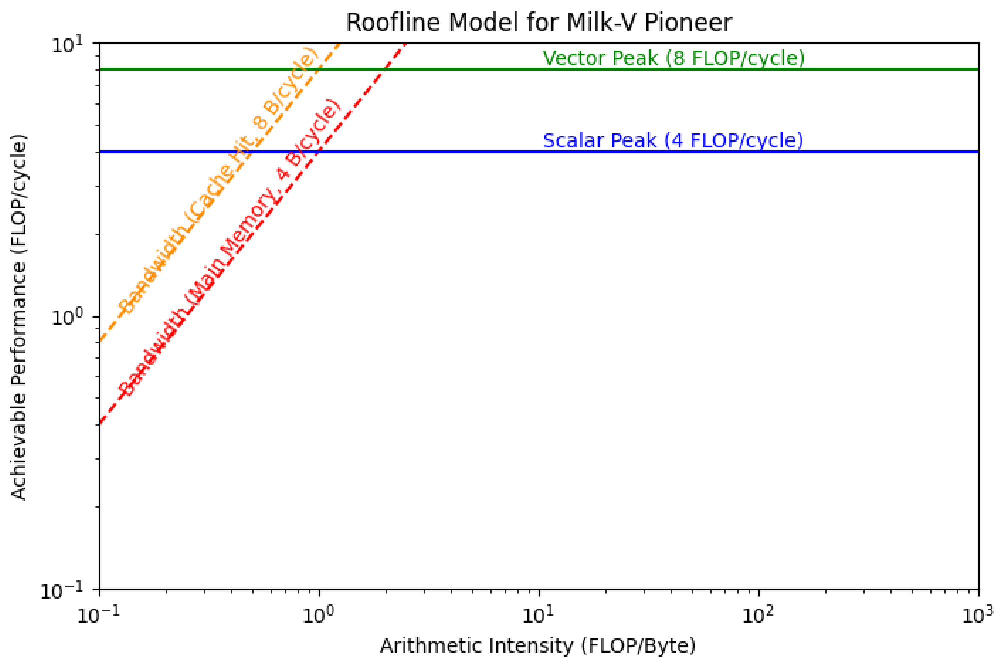


Figure 2.3: Milk-V Pioneer Roofline Model

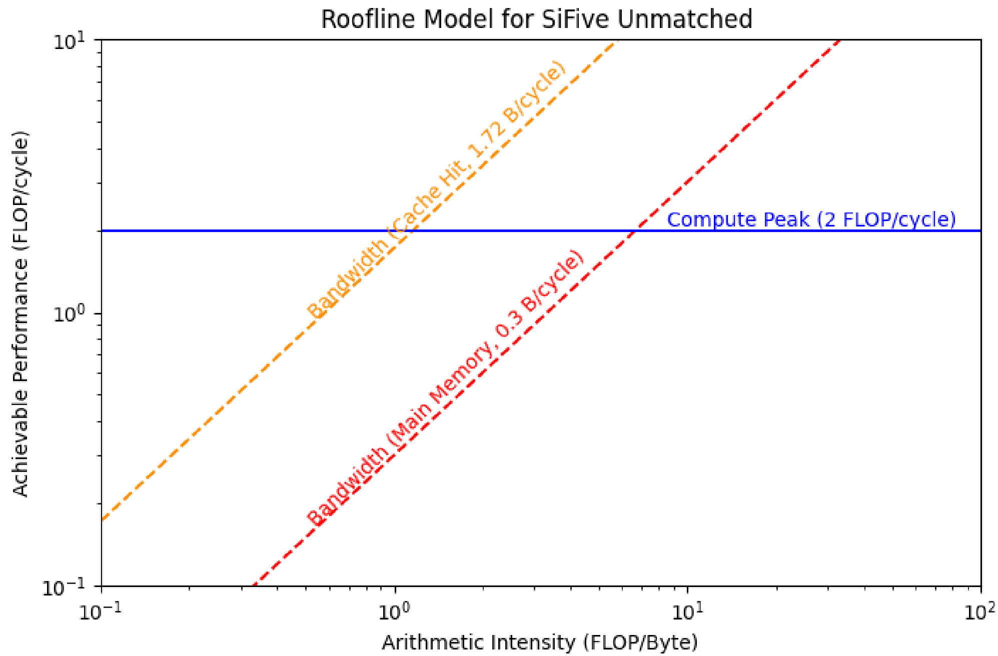


Figure 2.4: Sifive Unmatched Roofline Model

The Milk-V core demonstrates a higher compute roof than the SiFive Unmatched, due to its higher clock speed and a more advanced cache hierarchy. However, single-core bandwidth constraints continue to impact memory-intensive benchmarks, resulting in lower performance for memory-bound applications. Notably, the peak vector performance was obtained using a simple Fused Multiply-Add (FMA) program written in assembly, as challenges with autovectorisation limited our ability to fully leverage vector capabilities through compiler optimisations. These autovectorisation issues and their implications will be explored in detail in a separate chapter.

With a lower compute roof, the SiFive Unmatched core exhibits less peak computational throughput.

2.2.3 Benchmark Applications

When selecting a benchmark for evaluating computer architectures, it is important to consider the scope and focus of various benchmarks. For example, CoreMark [11] is widely used to measure CPU performance and efficiency in embedded systems, focusing on integer operations and basic algorithmic tasks. However, it may not capture the full range of performance characteristics in highly computationally intensive or high performance applications. MiBench [4], on the other hand, provides a more comprehensive evaluation by targeting embedded systems with a variety of real-world applications, including automotive and network processing. While useful for understanding performance in specific embedded contexts, MiBench may not fully address the requirements of HPC tasks. Benchmarks like these are important, but for a holistic evaluation of RISC-V architectures, especially in the context of HPC, benchmarks that cover a broader range of computational and memory-intensive tasks are often preferred. This is where the HPCC benchmark comes into play.

The HPCC (High Performance Computing Challenge) benchmark suite [15] is a set of tests designed to evaluate the performance of supercomputers and cluster-based computing systems. It provides a comprehensive assessment of various computational and data handling capabilities. The variety of tests aims to evaluate four key aspects of performance: double-precision floating-point arithmetic (covered by DGEMM and HPL [16]), local memory bandwidth (assessed by STREAM [17])), network bandwidth for large messages (evaluated by PTRANS, RandomAccess, FFT [18]), and LatencyBandwidth), and network bandwidth for small messages (also assessed by RandomAccess and LatencyBandwidth) [5]

1. **HPL (High Performance Linpack)**
   HPL is a widely used benchmark for measuring a system's floating-point performance by solving a dense system of linear equations. The metric used is the maximum number of floating-point operations per second (FLOP/s) that the system can achieve. HPL is critical because it provides a measure of the peak computational power of a system, which is particularly useful for applications that require intensive numerical calculations. This benchmark is the basis for the TOP500 list, which ranks the world's most powerful supercomputers.

2. **FFT (Fast Fourier Transform)**

   The FFT benchmark assesses the system's performance in computing a discrete Fourier transform of a very large one-dimensional complex data vector. FFTs are computationally heavy and require efficient memory and network performance. This benchmark reflects the system's ability to handle these intensive operations.

3. **STREAM**

   The STREAM benchmark measures sustainable memory bandwidth (in GB/s) and the corresponding computation rate for four simple vector kernel codes; Copy, Scale, Add, and Triad. This benchmark helps determine the bandwidth between the CPU and memory, which is critical for applications that require high data throughput, such as large-scale data analysis and simulations.

4. **PTRANS (Parallel Matrix Transpose)**

   PTRANS measures the rate at which large matrices can be transposed across different processors, involving a large amount of data movement. It evaluates the system's memory subsystem and network interconnect efficiency, which are crucial for parallel applications that require extensive data exchanges.

5. **DGEMM**

   DGEMM evaluates the performance of double-precision matrix-matrix multiplication operations. Efficient matrix multiplication is essential for tasks like simulations, optimisations, and many numerical algorithms that involve dense matrix operations.

6. **RandomAccess**

   RandomAccess measures the system's ability to perform random updates to a large global memory array, assessing how quickly and efficiently the system can handle irregular memory access patterns, calibrated in GUPS (Giga UPdates per Second). It tests the performance of the memory hierarchy and the network interconnect, especially in scenarios with non-sequential data access, common in database operations and other data-intensive tasks.

7. **LatencyBandwidth**

   The latency-bandwidth benchmark evaluates the effective communication bandwidth and latency of a system's interconnect by measuring the performance of different message-passing patterns, including small and medium-sized messages.

This is important for applications that rely heavily on communication between different parts of the system, especially those involving many small messages through MPI.

In addition to the HPCC benchmark, we also ran the AXPY benchmark. AXPY is a standard operation in linear algebra that computes y = alpha * x + y for vectors x and y and a scalar alpha. The AXPY operation is frequently used to measure the performance of vectorised computations on CPUs, GPUs, and other accelerators. It can provide insight to a system's memory bandwidth, compute efficiency and hardware optimisation capabilities.

In evaluating the performance of any architecture, it is crucial to consider various complexities. Levy et al. [6] advocate for a more nuanced approach to benchmarking that takes into account the broader context in which the system operates. Therefore, the aforementioned benchmarks were tested and run under three main criteria: intranode, internode and vectorised. The combinations of these criteria are shown in the table below:

Benchmark	***Intranode***	***Internode***	***Vectorization***
HPL	✓	✓	✓
FFT	✓	✓	✗
PTRANS	✓	✓	✗
STREAM	✓	✗	✗
RandomAccess	✓	✓	✗
Latency-Bandwidth	✓	✓	✗
DGEMM	✗	✗	✗
AXPY	✗	✗	✓

Table 2.3: Benchmark testing criteria

As outlined in the table, this study will exclude results from the DGEMM benchmark, considering it redundant given the inclusion of HPL. Furthermore, the STREAM benchmark will only be executed in the intranode scenario, as it specifically measures memory bandwidth. The Latency-Bandwidth benchmark will evaluate the shared memory fabric used by MPI in the intranode context and assess the network interconnect in the internode context. To examine vectorisation capabilities, the study will utilise the HPL and AXPY benchmarks.

2.2.4 Visualisation and Tracing Tools

The following tools, developed at BSC, are designed to visualise application behavior and identify performance-critical issues [24]. They are publicly available, accompanied by hands-on tutorials and comprehensive documentation to guide their use.

Extrae

Extrae is a dynamic instrumentation tool designed for collecting performance data from programs that use parallel programming models like MPI, OpenMP, CUDA, and combinations of these. It sets up the configuration based on an .xml file provided by the user. It monitors the application's environment to generate detailed trace files that can be analysed for performance bottlenecks. The trace files contain a breakdown of application activity, resource usage, and labels for numerical values. These files (.prv, .pcf, .row) are created during execution, offering insight into the behaviour of both hardware and software components.

Extrae also supports hardware counter instrumentation via PAPI [19], allowing users to select specific hardware events for performance monitoring. This data can be visualised and analysed with Paraver, which provides detailed performance views.

PAPI (Performance Application Programming Interface)

PAPI is used to access hardware counters, providing detailed metrics on low-level hardware events, essential for fine-tuning application performance.

Paraver

Paraver is a tool used for visualising and analysing performance data using both timelines and statistical views. It supports a wide variety of programming models and allows users to customise their analysis through configuration files.

Paraver allows users to conduct in-depth analysis of various performance metrics such as cache misses, Instructions Per Cycle (IPC), and load balance of parallel loops. In addition, it features options to compare multiple traces and provides advanced users with the ability to create custom metrics. One key strength of Paraver is its visual inspection capabilities, which help identify performance inefficiencies such as load imbalances or MPI communication issues. For advanced analysis, derived metrics and cooperative work are supported, enabling shared views and discussions among team members.

2.2.5 Performance Analysis Tools

Dimemas

Dimemas was originally conceived as a tool to predict how applications would behave on a standalone system by using a simple communication model. Subsequent development allowed Dimemas to predict the behaviour of multiple applications running simultaneously on shared processors.

Using Dimemas, developers can simulate message-passing applications, providing a detailed understanding of performance in different scenarios. The tool can model faster networks, better balanced applications, and other modifications without the need to rerun the actual application. Dimemas reconstructs application behaviour from trace files, focusing on the CPU time between communications and the communication primitives themselves. Notably, network contention is excluded from these traces(infinite bandwidth and zero latency), which allows Dimemas to simulate an ideal system free from network delays.

A significant feature of Dimemas is its ability to predict performance on a dedicated machine. It does this by accounting for the CPU time each process uses, disregarding any preempted time during instrumentation. This results in a more accurate representation of how an application would perform in a dedicated environment.

When used in conjunction with Basicanalysis, Dimemas processes Paraver traces and provides a breakdown of various inefficiencies according to the POP methodology.

Basicanalysis

Basicanalysis is a framework designed for the automatic extraction of key factors from Paraver traces. It acts a wrapper for setting up Dimemas configurations. By default, it generates a Python script called modelfactors.py to perform this extraction.

2.3 Benchmark Setup

This section discusses the parameters and configurations set for the benchmarking process and details how the benchmarks were executed.

2.3.1 Compilation

To evaluate the vectorisation capabilities of the Milk-V Pioneer's VPU, an AXPY benchmark was implemented using the Clang compiler with the following autovectorisation flags.

```
$ -O3 -ffast-math -mepi -mllvm -combiner-store-merging=0 \
  -Rpass=loop-vectorize -Rpass-analysis=loop-vectorize \
  -mllvm -vectorizer-use-vp-strided-load-store -mllvm \
  -enable-mem-access-versioning=0 -mllvm \
  -disable-loop-idiom-memcpy -fno-slp-vectorize
```

For the unvectorised version we used the following flags:

```
$  -O2 -fno-vectorize -fno-slp-vectorize
```

The compilation of the HPCC benchmark began with a makefile, typically named in the format Make.<configuration>. In our case, the file was named Make.Linux_BLIS. A copy of the file can be found in A.1. In the hpl/setup directory, various makefile templates are available to begin with. The makefile typically contains the paths to the libs and include directories for all the required libraries such as MPI and BLAS. Any necessary compilation flags are also specified.

The OpenMPI library was used to implement the Message Passing Interface (MPI), which facilitates communication between distributed processes in parallel computing environments. For linear algebra computations, the BLAS (Basic Linear Algebra Subprograms) library was used, integrated within the BLIS framework. BLIS was chosen over other popular linear algebra libraries such as OpenBLAS due to its highly modular design. This modularity is particularly advantageous when targeting new hardware architectures, like RISC-V, as it allows developers to easily adapt and optimise specific computational kernels (e.g., matrix multiplication, Axpy) without requiring a complete rewrite of the library. In addition, the benchmark was compiled with the LLVM compiler (clang) at an optimisation level

of -O3. For the vectorised version, the mentioned autovectorisation flags were added under the CCFLAGS variable.

Once the necessary modules were loaded, the executable is created by running the following command inside the main benchmark directory:

```
1 $ make arch=Linux_BLIS
```

2.3.2 Execution

For the AXPY benchmark, the vector size for this benchmark was set to 1,136,640 elements, with the operation repeated over 10,000 iterations to obtain consistent average performance metrics.

In the HPCC benchmark, if the makefile builds successfully, an executable `hpcc` is created. In addition, the parameters needed to run the benchmark such as matrix sizes, block sizes for HPL and error threshold were determined beforehand.

To further ensure a fair comparison between the two architectures, matrix sizes were selected based on their respective RAM capacity, while block sizes were determined by the size of their Last-Level Cache (LLC). For Unmatched, the LLC is the L2 cache of 2MB while for Pioneer it is the L3 cache of 64MB.

To calculate the matrix size, we used the following formula:

$$N_{max} = \sqrt{\frac{Total\ RAM \times Percentage\ of\ memory \times Number\ of\ nodes}{Element\ size}}$$

For both architectures, the occupation is set to 75% RAM.The number of node was set to be 1. The study purposed to do a strong scaling, whereby the problem size remained constant as the number of resources increased. The element size was taken to be 8 for the fact that the matrices consist of double-precision elements.

Choosing the optimal block size for matrix multiplication presents challenges due to multiple influencing factors, including cache size, cache levels, and cache associativity. [8]. Since no single method can determine the best block size for all scenarios [9], reducing cache misses becomes a primary objective. In this study, we propose a simplified approach to estimate the block size by focusing on the number of data elements that can be accommodated in the cache at a time. The formula requires all three blocks to fit into the cache at the same time, so the total memory requirement in terms of the number of elements is proportional to :

$$3 \times B^2 \times Element\ size$$

To create a conservative buffer that reduces the likelihood of cache thrashing , the number of elements is estimated to B^3. Thus, to calculate the block size, we used the following expression to solve for B:

$$B^3 \times Element\ size \leq LLC\ size$$

This formula assumes a simplistic model where the cache is fully dedicated to storing matrix elements and aims to comfortably fit data elements into the cache. By focusing solely on cache capacity, this approach simplifies the problem, providing a practical method for estimating the block size. Although it does not capture every factor impacting the blocking algorithm, this formula offers a straightforward approach to minimise cache misses and enhance matrix multiplication performance, especially in environments where cache size is the dominant concern.

To ensure better alignment and more efficient memory access patterns, the matrix size should be divisible by the block size.

Therefore, the matrix and block sizes for the respective architectures were as follows:

	Matrix size N	**Block size**
Unmatched	39936	64
Pioneer	109440	128

These values are set in the `hpccinf.txt` file. A.2 shows a copy of the file that was used for the Unmatched board. Other parameters such as table-size for Randomaccess and array size for STREAM is not explicitly set by the user through a direct parameter. Instead, it is determined automatically based on the system's available memory.

Performance Analysis

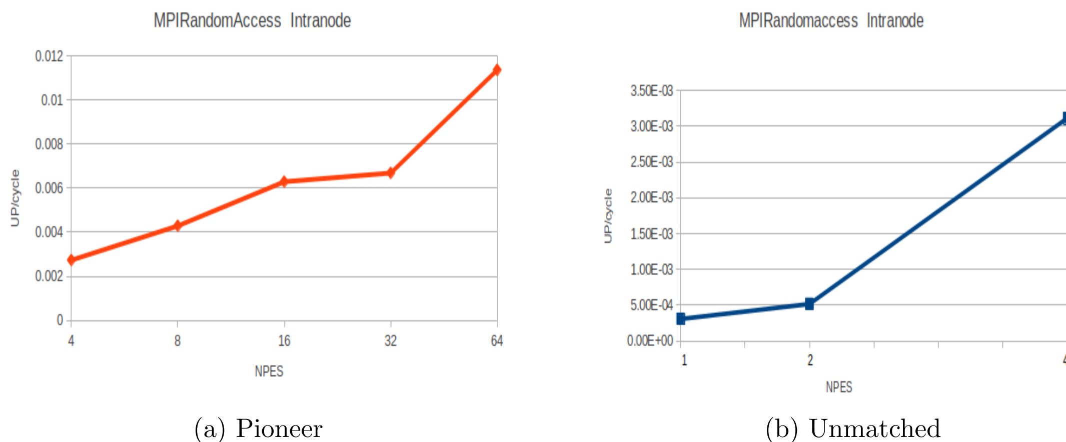
3.1 Findings & Discussion

In this section, we discuss the benchmark results obtained from the respective architectures. The scaling performance analysis is seperated into intranode, internode and vectorised scenarios. Finally, we interpret these results within the framework of the roofline model.

3.1.1 Intranode Performance

RandomAccess

In the intranode tests for RandomAccess as shown in Figure 3.1, for both architectures, the updates per cycle increased by one order at a full node with respect to the lowest tested configuration; four cores in Pioneer and one core in Unmatched. Interestingly, both the Milk-V Pioneer and SiFive Unmatched architectures achieved similar updates per cycle when configured with four processing elements. Despite Pioneer featuring a more complex memory hierarchy, its performance remains underwhelming. To achieve 10x the throughput, Pioneer needed to use 16 times more cores, whereas Unmatched only required a fourfold increase in cores.

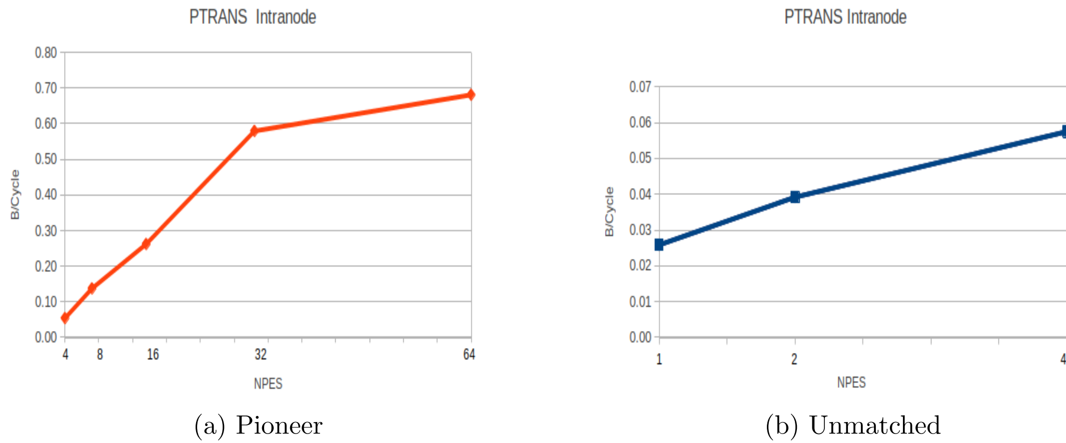


(a) Pioneer (b) Unmatched

Figure 3.1: MPI Randomaccess scaling performance on a single node

PTRANS

According to Figure 3.2 both the Pioneer and Unmatched architectures exhibited a consistent increase in bandwidth with additional processing elements, though Pioneer showed more steep incline until 32 cores then showing initial signs of saturation at 64 cores. Overall, Pioneer achieved higher peak bandwidth, likely due to its larger RAM capacity, which supports more extensive parallel data handling.

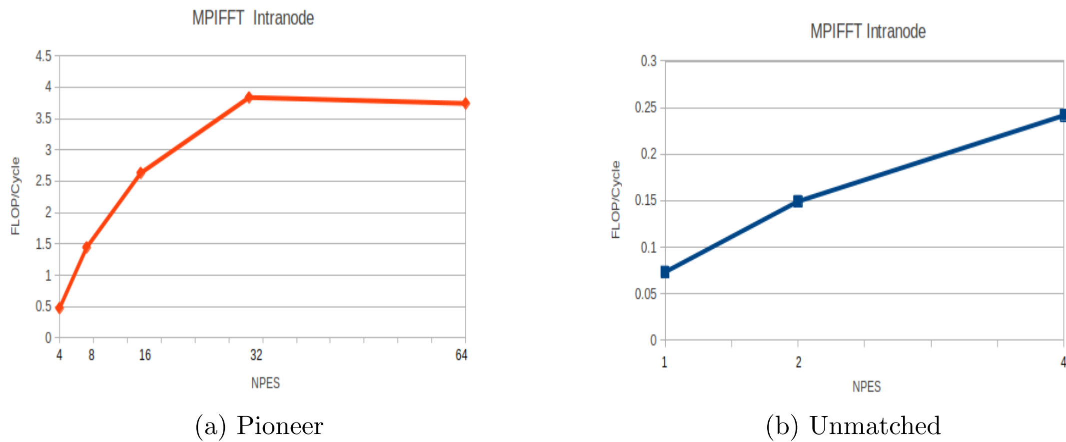


(a) Pioneer (b) Unmatched

Figure 3.2: PTRANS scaling performance on a single node

FFT

The results in Figure 3.3 indicate that the Milk-V Pioneer saturates at 64 processing elements, whereas the SiFive Unmatched exhibits nearly ideal scaling across the tested range. At the same number of cores (4), Milk-V Pioneer achieved twice the FLOP/cycle of SiFive Unmatched. However, as the number of processing elements increases, the diminishing returns on the Pioneer could indicate bottlenecks either within its interconnect or memory access paths, limiting scalability despite the hardware's theoretical potential.

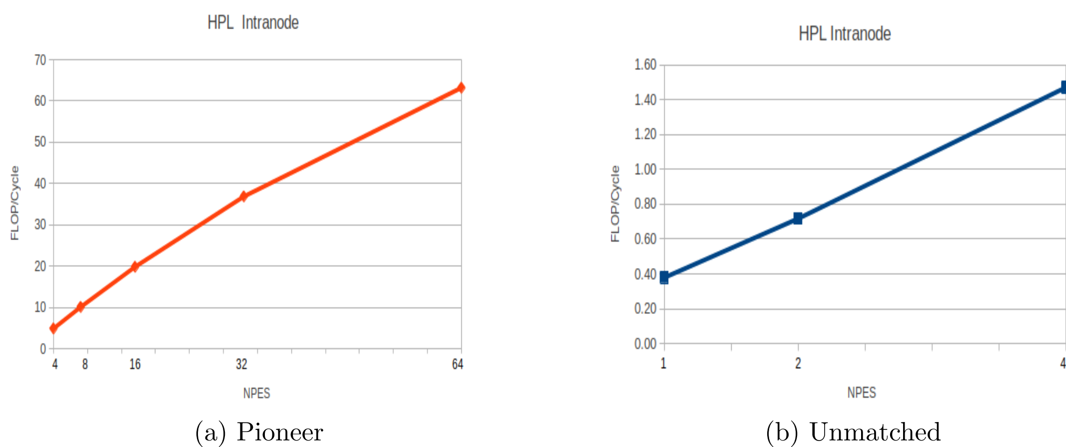


(a) Pioneer (b) Unmatched

Figure 3.3: MPI FFT scaling performance on a single node

HPL

Both architectures showed improved FLOP/cycle as the number of processing elements increased. SiFive Unmatched exhibited near-perfect scaling, while Milk-V Pioneer achieved about 80% of ideal scaling, as indicated by the global efficiency at 64 MPI processes in Figure 3.4. The efficiency data, shown in Figure 3.5 and obtained using Basicanalysis and Dimemas tools, highlights the benefits of Unmatched's simpler configuration for maintaining consistent scaling. The drop to 80% efficiency in Milk-V Pioneer is primarily due to a reduction in IPC scalability.



(a) Pioneer (b) Unmatched

Figure 3.4: HPL scaling performance on a single node

	Pioneer					Unmatched		
	4	8	16	32	64	1	2	4
Global efficiency	96.69	96.10	95.80	89.42	80.30	100.00	94.55	94.72
-- Parallel efficiency	96.69	95.76	94.23	90.13	88.79	100.00	97.34	97.44
-- Load balance	99.82	98.93	98.17	97.09	92.70	100.00	97.73	99.43
-- Communication efficiency	96.86	96.80	95.98	92.83	95.78	100.00	99.60	98.00
-- Serialization efficiency	96.97	97.09	96.29	93.43	96.17	100.00	99.68	98.20
-- Transfer efficiency	99.89	99.70	99.68	99.36	99.60	100.00	99.91	99.79
-- Computation scalability	100.00	100.35	101.67	99.21	90.44	100.00	97.13	97.21
-- IPC scalability	100.00	100.48	101.45	99.26	90.44	100.00	97.33	97.47
-- Instruction scalability	100.00	99.87	100.22	99.95	100.03	100.00	99.81	99.79
-- Frequency scalability	100.00	100.00	100.00	100.00	99.97	100.00	99.99	99.94

Figure 3.5: **Efficiency table for HPL intranode**

The relation between throughput and efficiency is explained in the following formula [25]:

$$\text{GFLOPs}_{\text{New}} = \frac{\text{Resources}_{\text{new}} \times \text{Efficiency}_{\text{new}}}{\text{Resources}_{\text{base}} \times \text{Efficiency}_{\text{base}}} \times \text{GFLOPs}_{\text{base}}$$

Where we consider the first part of the right-hand-side to be the speedup.

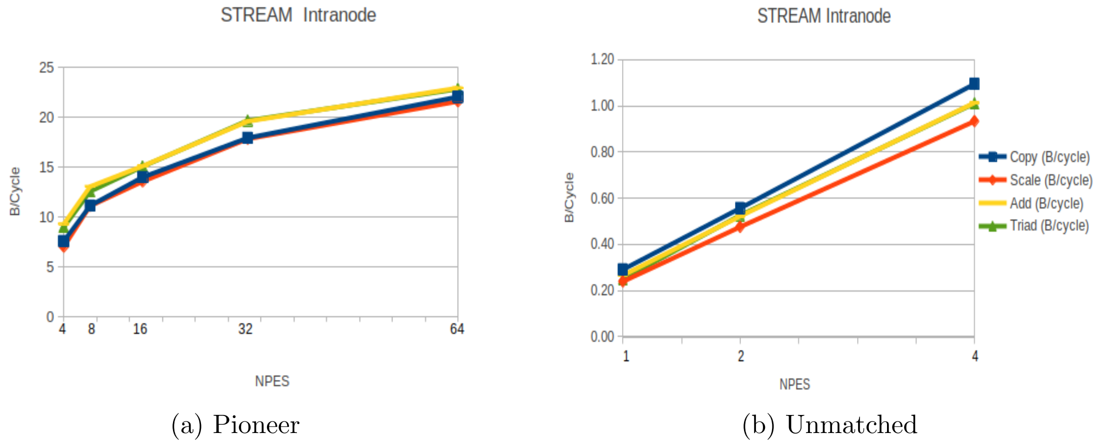
$$\text{Speed-up} = \frac{\text{Resources}_{\text{new}} \times \text{Efficiency}_{\text{new}}}{\text{Resources}_{\text{base}} \times \text{Efficiency}_{\text{base}}}$$

Therefore, we can simplify the formula this way:

$$\text{GFLOPs}_{\text{New}} = \text{Speed-up} \times \text{GFLOPs}_{\text{base}}$$

STREAM

The STREAM benchmark as shown in Figure 3.6 revealed a substantial performance edge for the Milk-V Pioneer, achieving approximately ten times the bytes per cycle compared to the SiFive Unmatched. This disparity underscores the advantage of advanced memory hierarchies in optimising data-intensive HPC applications, particularly where rapid data access is critical.



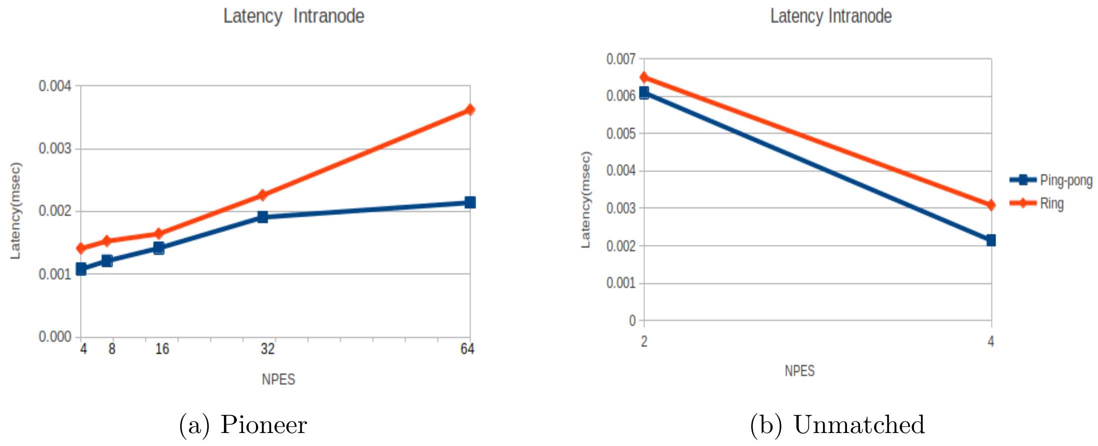
(a) Pioneer (b) Unmatched

Figure 3.6: STREAM scaling performance on a single node

Latency-Bandwidth

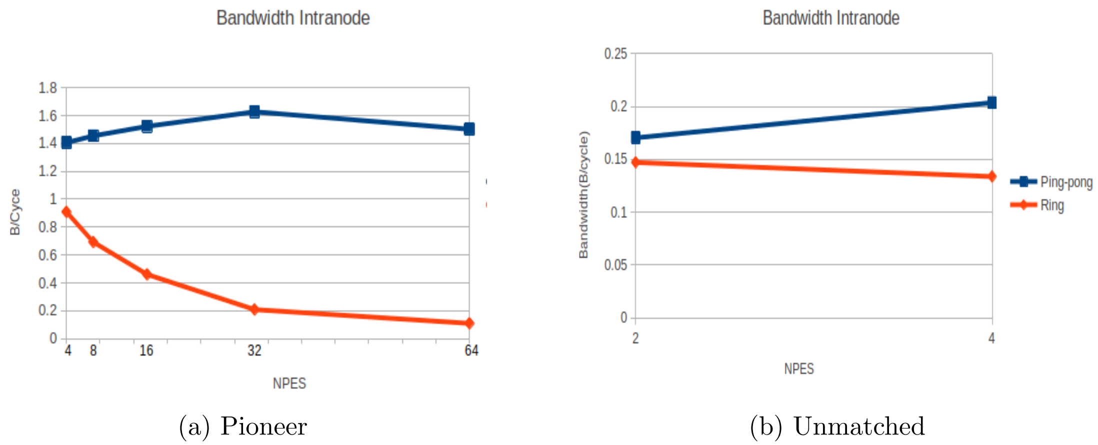
The latency in the results in Figure 3.7 highlights a distinct pattern between the two architectures. On the Pioneer, latency steadily rises as the number of cores increases, suggesting that the overhead within the shared memory fabric of MPI becomes more pronounced with additional cores. In contrast, Unmatched exhibits a decreasing latency trend under similar conditions, which could indicate more efficient handling of shared-memory communication or other architectural optimisations that help maintain lower latency as workload distribution grows.

In terms of bandwidth, the Pioneer board maintained nearly constant bandwidth for ping-pong (small message) communication, whereas bandwidth in the ring communication (medium-sized messages) gradually decreased. This indicates that Pioneer's architecture is well optimised for small and frequent data exchanges, but it can experience performance degradation with larger data patterns as the number of nodes increases. The Unmatched board displayed a more stable bandwidth for both ring and ping-pong communication patterns, indicating a consistent response to varying message sizes and a balanced memory interconnect that supports different communication patterns.



(a) Pioneer (b) Unmatched

Figure 3.7: Latency scaling performance on a single node



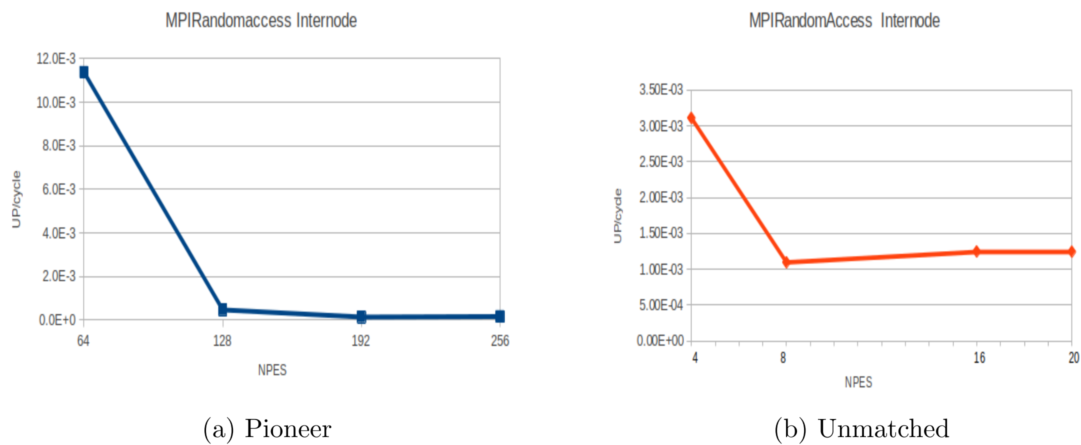
(a) Pioneer (b) Unmatched

Figure 3.8: Bandwidth scaling performance on single node

3.1.2 Internode Performance

RandomAccess

Figure 3.9 shows that in multinode settings, the updates per cycle for both architectures dropped significantly compared to single-node configurations. This decrease can largely be attributed to network latency, as inter-node communication introduces substantial overhead. On a single node, all memory accesses occur within the same memory space, whereas multinode systems rely on network-based communication between distinct memory spaces, resulting in distributed memory overhead.

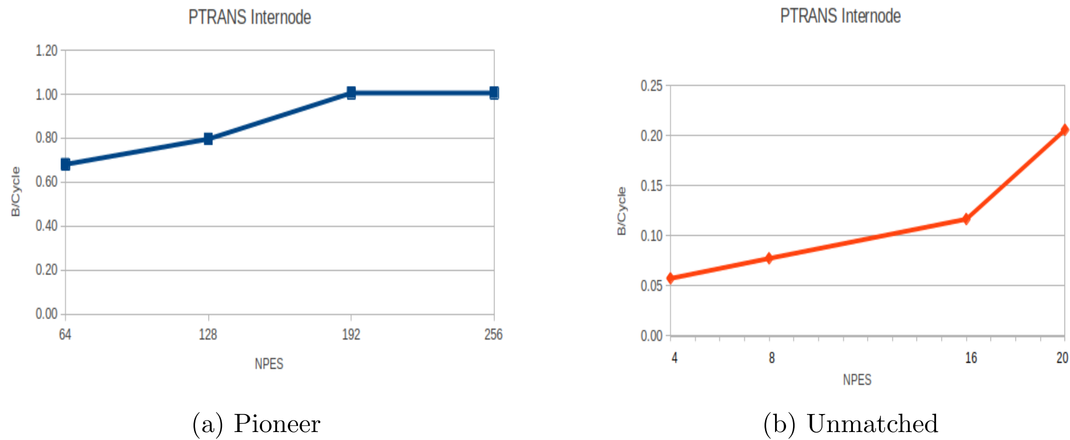


(a) Pioneer (b) Unmatched

Figure 3.9: MPI Randomaccess scaling performance on multiple nodes

PTRANS

In multinode configurations, bandwidth continued to increase for both architectures, but the Pioneer architecture reached a plateau at around 256 processing elements, as shown in Figure 3.10 (a). This saturation point indicates that the problem size per processing element has become less significant relative to the data movement overhead. On the other hand, the Unmatched board exhibited more linear scaling with fewer signs of saturation. Overall, Pioneer still delivers approximately one order of magnitude more bandwidth than the Unmatched board.

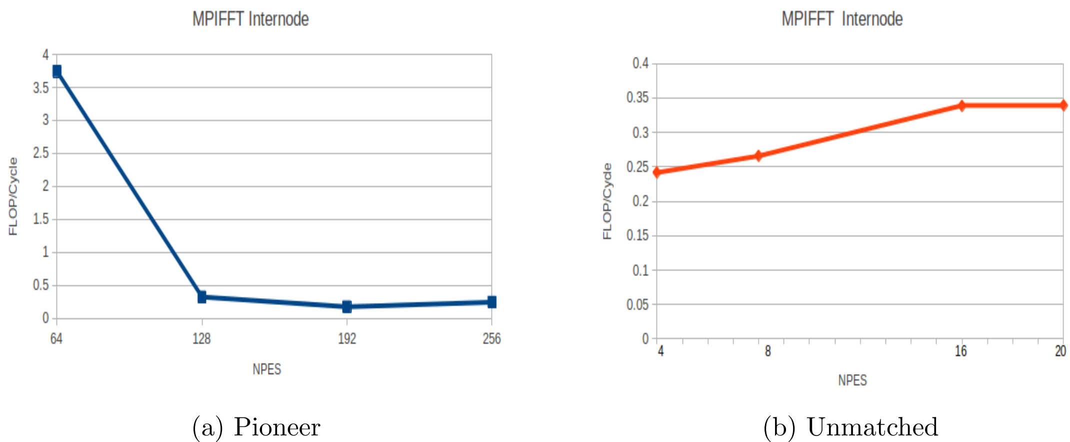


(a) Pioneer (b) Unmatched

Figure 3.10: PTRANS scaling performance on multiple nodes

FFT

FFT plots in Figure 3.11 shows that Milk-V Pioneer demonstrates a sharp drop in performance, with FLOP/cycle falling by a factor of ten in internode configurations, averaging around 0.25 FLOP/cycle. This contrasts sharply with the trend observed in the Unmatched architecture, where FLOP/cycle remain relatively stable even in an internode environment, saturating at 20 cores. This could imply that Pioneer's architecture is more sensitive to network latency and internode synchronisation issues than Unmatched, which maintains consistent throughput across nodes.



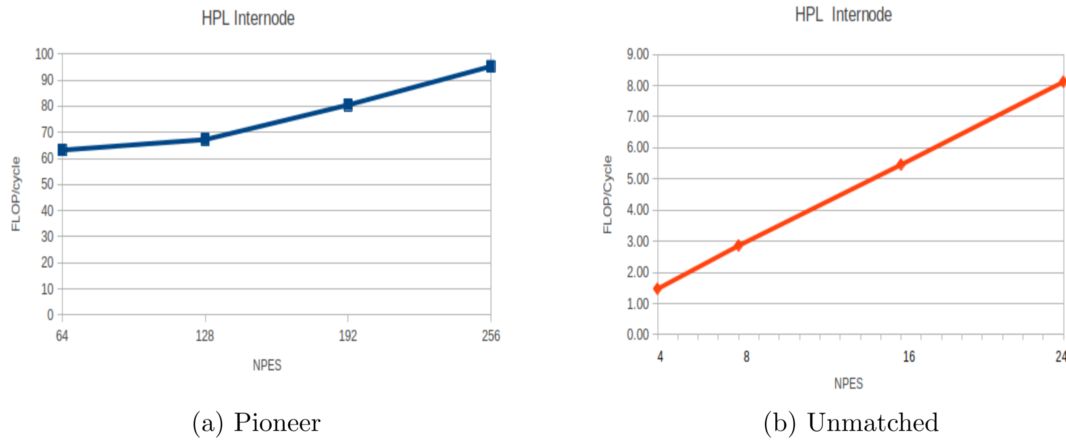
(a) Pioneer (b) Unmatched

Figure 3.11: MPI FFT scaling performance on multiple nodes

HPL

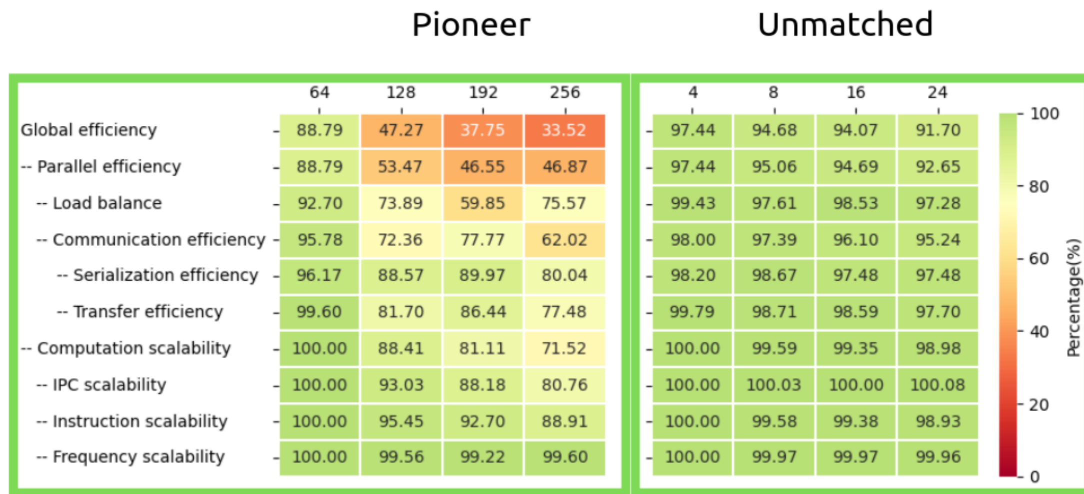
For HPL, we observe in Figure 3.12 that both architectures continue to increase in FLOP per cycle with additional resources, but Unmatched achieves scaling closer to ideal. Pioneer's efficiency is hampered by communication overhead and load

imbalance, dropping to about 30% at the highest tested configuration as seen in Figure 3.13.



(a) Pioneer (b) Unmatched

Figure 3.12: HPL scaling performance on multiple nodes



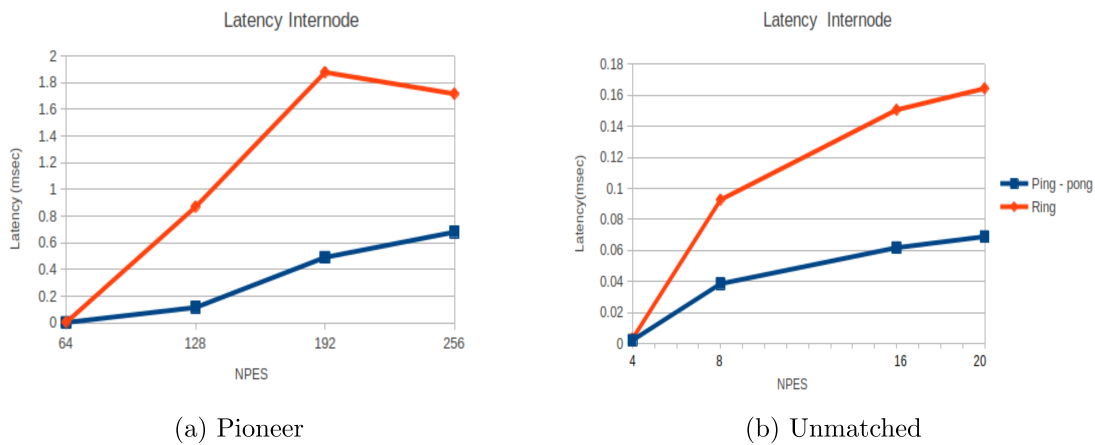
	Pioneer 64	Pioneer 128	Pioneer 192	Pioneer 256	Unmatched 4	Unmatched 8	Unmatched 16	Unmatched 24
Global efficiency	88.79	47.27	37.75	33.52	97.44	94.68	94.07	91.70
-- Parallel efficiency	88.79	53.47	46.55	46.87	97.44	95.06	94.69	92.65
-- Load balance	92.70	73.89	59.85	75.57	99.43	97.61	98.53	97.28
-- Communication efficiency	95.78	72.36	77.77	62.02	98.00	97.39	96.10	95.24
-- Serialization efficiency	96.17	88.57	89.97	80.04	98.20	98.67	97.48	97.48
-- Transfer efficiency	99.60	81.70	86.44	77.48	99.79	98.71	98.59	97.70
-- Computation scalability	100.00	88.41	81.11	71.52	100.00	99.59	99.35	98.98
-- IPC scalability	100.00	93.03	88.18	80.76	100.00	100.03	100.00	100.08
-- Instruction scalability	100.00	95.45	92.70	88.91	100.00	99.58	99.38	98.93
-- Frequency scalability	100.00	99.56	99.22	99.60	100.00	99.97	99.97	99.96

Figure 3.13: **Efficiency table for HPL internode**

Latency-Bandwidth

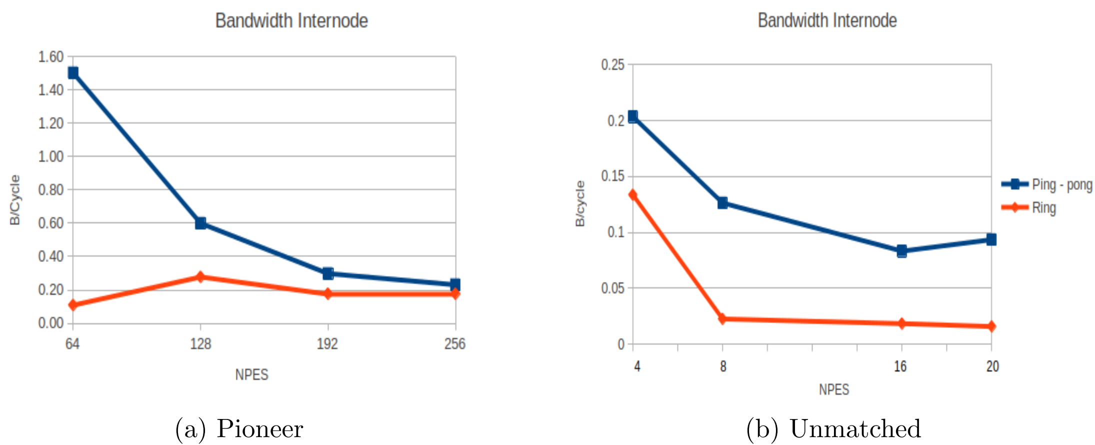
In the multinode configuration, both architectures exhibited increased latency as the number of nodes scaled up. This rise in latency as shown in Figure 3.14 suggests that network communication overhead becomes a limiting factor in distributed settings. For Pioneer, latency increased more steeply compared to Unmatched. This difference indicates that the Unmatched may handle inter-node communication more efficiently, with architectural elements that help to mitigate some of the delays introduced by multi-node scaling.

Bandwidth results in multinode scenarios as seen in Figure 3.15 further highlight the impact of increased node count on data exchange efficiency. Both architectures showed a gradual drop in effective bandwidth as nodes increased, with the Pioneer experiencing a more pronounced decrease. The consistent bandwidth reduction on the Pioneer further suggests that its network interconnect may be more sensitive to node scaling, impacting its ability to maintain high-speed data transfer in distributed systems. The Unmatched, while also experiencing a reduction, maintained more stable bandwidth across nodes, reflecting greater adaptability to multinode configurations.



(a) Pioneer (b) Unmatched

Figure 3.14: Latency scaling performance on multiple nodes



(a) Pioneer (b) Unmatched

Figure 3.15: Bandwidth scaling performance on multiple nodes

3.1.3 Results analysis within the Roofline Model

In this section, we will analyse the results from two benchmarks: HPL and STREAM, obtained from three runs each on a single core, within the previously discussed Roofline models of the respective architectures. These two benchmarks were strategically chosen as HPL is primarily compute-bound while STREAM is memory-bound.

To plot the results in the roofline model, we need to have the throughput (FLOP/cycle) and arithmetic intensity (FLOP/Byte). We use the following formulae to calculate the arithmetic intensity.

For HPL, three matrix sizes were selected corresponding to 35%, 50%, and 75% of the system's RAM capacity for the respective architectures. The matrix size N is used to estimate the amount of bytes moved during calculations by squaring the value then multiplying by 8 bytes which is the size of one double-precision element.

$$Data\ moved = N^2 \times 8$$

To get total number of floating-point operations, we multiplied the FLOP/s with total time.

$$Num_FLOP = FLOP/s \times Duration$$

Finally, the arithmetic intensity is calculated by dividing the number of floating-point operations with the amount of data moved.

$$Arithmetic\ Intensity = \frac{Num_FLOP}{Data\ Moved}$$

For the STREAM benchmark, we utilised three array sizes, each scaled to be 1x, 2x, and 4x larger than the last-level cache size of the respective architectures. The benchmark consists of four kernels of which we each assigned a constant arithmetic intensity based on the following criteria:

- **Copy** - 0 FLOP/Byte (16 bytes moved 0 calculations)
- **Scale** - 1/16 FLOP/Byte (16 bytes moved 1 calculation)
- **Add** - 1/24 FLOP/Byte (24 bytes moved 1 calculation)
- **Triad** - 1/12 FLOP/Byte (24 bytes moved 2 calculations)

We then proceed to calculate the theoretical bandwidth by dividing the total bytes moved by the average duration:

$$Data\ moved = N \times 8$$

$$Bandwidth = \frac{Data\ moved}{Avg_duration}$$

and finally we obtain the throughput with the formula below:

$$FLOP/Cycle = \frac{Arithmetic\ Intensity \times bandwidth}{Frequency}$$

From the above calculations, we plotted the HPL and STREAM results for the respective architectures.

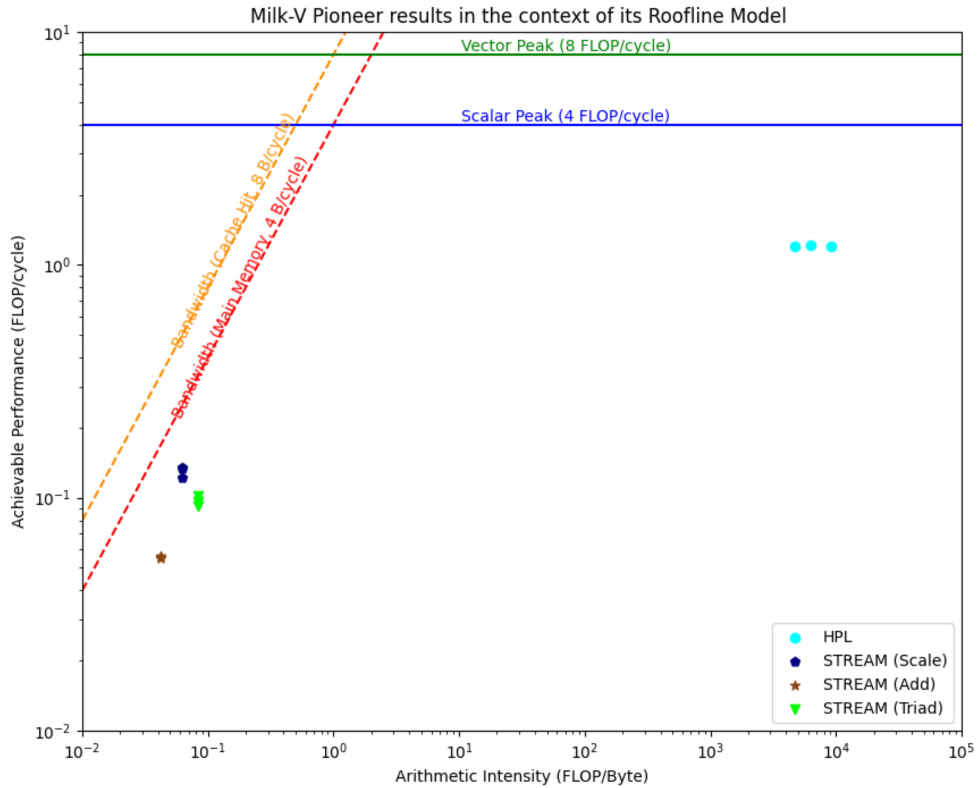


Figure 3.16: Milk-V Pioneer results in the context of its Roofline Model

As seen above in Figure 3.16 the HPL data points cluster relatively close to the scalar peak roof, indicating that Pioneer is able to sustain a high percentage of its theoretical peak performance for compute-heavy workloads.

As for the STREAM benchmark, the data points are closer to the main memory

bandwidth roof than the cache bandwidth's. These results coincide with what we expected as the STREAM benchmark requires that the array size chosen is bigger than the cache size so that the program is forced to traverse through the whole memory hierarchy to fetch the data.

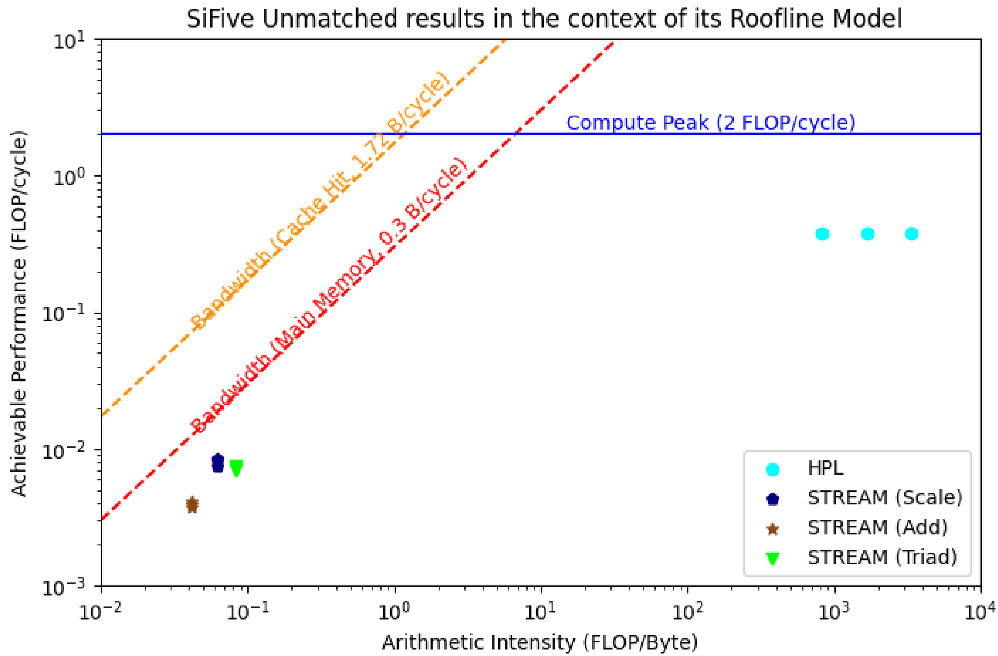


Figure 3.17: SiFive Unmatched results in the context of its Roofline Model

In Figure 3.17, Sifive Unmatched exhibits similar behaviour to Pioneer's, as the compute intensive HPL performance is closer to the compute peak roof and memory-bound STREAM data points lay to the lower left region of the roof.

3.1.4 Vectorisation in RISC-V Architectures

Vectorization is a key performance optimization technique that enables a processor to perform a single instruction on multiple data elements simultaneously. This is achieved through a specialized hardware component called the Vector Processing Unit (VPU), which is designed specifically for handling vector instructions. This parallel data processing can significantly improve computational throughput, especially in repetitive, data-heavy tasks such as scientific simulations or machine learning operations. In architectures that support vectorisation, such as RISC-V with optional vector extensions, VPUs enable more efficient handling of data-parallel workloads.

In this section, we explore the vectorisation potential of the Milk-V Pioneer board. Despite the theoretical advantages, benchmark results show that the Pioneer's VPU does not provide a measurable performance gain over non-vectorised code, likely due to hardware or memory bandwidth constraints. Ideally, since Pioneer's VPU has 2 lanes and maximum vector length of 2 double-precision elements, we would expect a speedup of about 2x for double precision calculation and 4x for single-precision from the scalar version.

The performance improvement was marginal, with only about an 8% increase in GFLOP/s and FLOP/cycle was observed. This result indicates that the VPU on the Milk-V Pioneer may not be fully utilised, as the expected performance gain from vectorisation was not achieved.

	Avg time per iteration(Seconds)	**GFLOP/s**	**FLOP/cycle**
Vectorised	0.001634	1.3909	0.6955
Unvectorised	0.001768	1.2858	0.6429

Table 3.1: Pioneer AXPY results

To further investigate the vectorisation capabilities, the HPL benchmark was executed with both vectorised and nonvectorised configurations. Similar FLOP/s were observed for both vectorised and nonvectorised runs, suggesting that the Milk-V Pioneer's architecture cannot leverage vector instructions effectively, even in computationally intensive tasks.

In contrast,when the same binary from the AXPY program was run on the FPGA SDV—originally not part of the initial study but included to explore the Pioneer's performance limitations—the vectorised version significantly outperformed the nonvectorised version, achieving 83.82% more thoughput. Furthermore, combining vectorisation with loop unrolling (4 times) resulted in a dramatic performance boost, delivering a 1242.65% increase in throughput compared to the nonvectorised version. The VPU in the SDV has a maximum vector length of 256 double-precision elements and 8 lanes, meaning that we expected a maximum speedup of 8x for double-precision calculation and 16x for single-precision from the scalar version.

These results indicate that the limitations are specific to the Pioneer's architecture rather than the vectorisation approach itself.

	Avg Time per Iteration (Seconds)	**GFLOP/s**
Vectorised	0.0181773413	0.0125
Vectorised with loop unrolling (4)	0.002688320	0.0913
Unvectorised	0.033213960	0.0068

Table 3.2: SDV AXPY results

At this juncture, it is important to reiterate that the Milk-V Pioneer houses the Sophon SG2042, the first RISC-V-based commodity chip developed for HPC [10]. Given this pioneering status, some limitations are expected as this represents an initial step in the evolution of RISCV in HPC. Consequently, performance improvements and architectural refinements can be anticipated in future iterations.

Since vectorisation did not yield any performance benefits on Pioneer, it was deemed unnecessary to conduct vector emulation on the SiFive Unmatched board.

Furthermore, comparing the autovectorisation capabilities between GCC and LLVM was not feasible because the current RISC-V GNU toolchain lacks support for RVV 0.7, the vector extension used in Pioneer's cores.

Conclusion

This thesis has presented an initial examination of the potential of RISC-V architecture for HPC, with a specific focus on two boards, the SiFive HiFive Unmatched and the Milk-V Pioneer. To assess these platforms, we began with a foundational introduction that outlined the relevance of RISC-V as an open-source ISA that encourages innovation and customisation in HPC. The project background section then explored the evolution of this project within the context of prior initiatives such as the MEEP project. Building on these advancements, this study aimed to contribute to the larger goals of the Barcelona Zettascale Laboratory.

In the methodology section, we detail the platform selection and benchmark configurations that allowed a comprehensive performance evaluation. Our methodology leveraged tools developed here at BSC, such as Paraver and Extrae, to gather performance metrics in areas such as floating-point operations, memory bandwidth, and computational efficiency.

The performance analysis chapter covered both intranode and internode evaluations across key benchmarks, including HPL, FFT, STREAM, PTRANS, AXPY and RandomAccess. The single-core performance of the HPL and STREAM benchmarks for each architecture was also analysed within the Roofline model framework. Finally, the chapter explored how current RISC-V implementations handle vector processing, highlighting the potential and limitations of the architecture in HPC-orientated tasks.

The benchmark results underscore the strengths of RISC-V in scalability and efficiency, demonstrating sufficiently good single-node performance across both architectures. Notably, the SiFive HiFive Unmatched demonstrated better scaling compared to the Milk-V Pioneer, maintaining consistent efficiency as processing elements increased. Although the Milk-V Pioneer exhibited higher peak performance, its efficiency decreased with increased internode communication, suggesting areas for further architectural improvements in network optimisation.

Autovectorisation proved limited in its implementation on the Milk-V Pioneer, where the VPU did not fully realise its theoretical potential due to hardware bottlenecks. Despite the limited gains observed in vectorised performance, both architectures demonstrated adequate efficiency in handling specific HPC workloads. For example, the Milk-V Pioneer's memory hierarchy and cache structure showed significant strengths in data-heavy benchmarks like STREAM, which are essential

for memory-bound HPC and AI tasks. These insights suggest that with targeted architectural refinements, RISC-V can become a versatile and efficient platform that is well suited to diverse computing demands in both traditional HPC and emerging AI applications.

As the global computing landscape increasingly prioritises energy efficiency and sustainability, the architectural efficiency of RISC-V positions it as a promising contender. This thesis thus supports the view that, with continued innovation, RISC-V can drive impactful change in HPC, delivering on goals for both high-performance and sustainable computing.

Suggestions and Future Work

This section presents the following key recommendations that aim to address current limitations in benchmarking flexibility, compatibility with evolving hardware, and support for advanced vectorisation. In addition, future work is proposed to better understand the potential of RISCV in HPC environments.

Enhancing the flexibility of performance testing could be achieved by improving the HPCC benchmark suite, which currently lacks support for running individual benchmarks independently. This limitation makes it difficult to isolate and retest specific architectural features. By modifying the HPCC executable to allow for selective execution of individual benchmarks, researchers could more accurately evaluate targeted aspects of performance, such as memory bandwidth or floating-point operations. This change would also allow for faster reruns of specific benchmarks, making the testing process more efficient.

Another important direction for future work is the regular updating of the HPCC repository. As the RISC-V ecosystem evolves, it is crucial that the benchmark repository is kept up-to-date to ensure it remains compatible with new compiler versions and emerging hardware features. Frequent updates to the HPCC repository would help address bugs, incorporate optimisations, and integrate advances not only in RISC-V but also in other architectures, making it a more reliable and effective tool for researchers. This would ensure that performance assessments are based on the most current capabilities of the architecture, leading to more accurate and relevant conclusions.

Further progress could also be made by integrating RISC-V with the latest and more stable version of its Vector Extension, RVV 1.0. The current study used architectures based on RVV version 0.7, which limited the effectiveness of vectorisation in certain tasks. This thesis is for the idea that transitioning to RVV 1.0 would provide more robust support for vector operations, likely resulting in significant performance improvements in data-parallel applications. Moreover, RVV 1.0 would allow for better compiler optimisations, maximising the performance benefits of vectorisation in both memory-bound and compute-intensive benchmarks.

Additionally, the lagarto-Ox - based SDV currently under development by the hardware team for the BZL project was originally intended to be included for benchmarking purposes. However, due to delays in its development, it could not be utilised in the present study. Future research should consider leveraging the

SDV once it becomes available to conduct more detailed and accurate performance evaluations of the targeted RISC-V architecture. Additionally, the next iteration of the Lagarto Ox processor will include an RVV 1.0-compliant VPU.

Finally, a more comprehensive approach to benchmarking RISC-V architectures for HPC applications would involve exploring additional benchmark configurations and scaling studies. This could include multi-threaded setups to investigate intra-node parallelism and weak scaling tests to assess the architecture's efficiency as both workload and processing resources are scaled proportionally. Such expanded tests would offer deeper insight into RISC-V's scalability and its potential for HPC, providing valuable information for future architectural optimisations and applications.

Annex

A.1 HPCC Benchmark Makefile

```
SHELL          = /bin/sh
#
CD             = cd
CP             = cp
LN_S           = ln -s
MKDIR          = mkdir
RM             = /bin/rm -f
TOUCH          = touch
#
# ----------------------------------------------------------------------
# - Platform identifier ------------------------------------------------
# ----------------------------------------------------------------------
#
ARCH           = Linux_BLIS
#
# ----------------------------------------------------------------------
# - HPL Directory Structure / HPL library ------------------------------
# ----------------------------------------------------------------------
TOPdir         = ../../..
INCdir         = $(TOPdir)/include
BINdir         = $(TOPdir)/bin/$(ARCH)
LIBdir         = $(TOPdir)/lib/$(ARCH)
#
HPLlib         = $(LIBdir)/libhpl.a
#
# ----------------------------------------------------------------------
# - Message Passing library (MPI) --------------------------------------
# ----------------------------------------------------------------------
#
MPdir        = /apps/riscv/ubuntu/openmpi/4.1.6_gcc11.4.0/
```

```
MPinc        = -I$(MPdir)/include
MPlib        = -lmpi -L$(MPdir)/lib


EXTraedir  =  /apps/riscv/ubuntu/extrae/4.2.0_papi-riscv
EXTraeinc    =  -I$(EXTraedir)/include
EXTraelib    =  -lseqtrace -L$(EXTraedir)/lib

# ----------------------------------------------------------------------
# - Linear Algebra library (BLAS or VSIPL) -----------------------------
# ----------------------------------------------------------------------

LAdir        = /apps/riscv/ubuntu/blis/0.8.1-llvmEPI0.7dev
LAinc         = -I$(LAdir)/include
LAlib         = -lblis -L$(LAdir)/lib
----------------------------------------------------------------------
# - HPL includes / libraries / specifics -------------------------------
# ----------------------------------------------------------------------
#
HPL_INCLUDES = -I$(INCdir) -I$(INCdir)/$(ARCH) $(LAinc) \
                $(MPinc)
HPL_LIBS     = $(HPLlib) $(LAlib) $(EXTraelib) $(MPlib)
# ---------------------------------------------------------------------
#
HPL_DEFS     = $(F2CDEFS) $(HPL_OPTS) $(HPL_INCLUDES)
#
# ---------------------------------------------------------------------
# - Compilers / linkers - Optimization flags ---------------------------
# ---------------------------------------------------------------------
#
CC           = mpicc
CCNOOPT      = $(HPL_DEFS) -fPIC
CCFLAGS      = $(HPL_DEFS) -g -O3 -fPIC
#
#
LINKER       = mpifort
```

```
LINKFLAGS      = $(CCFLAGS)
#
ARCHIVER       = ar
ARFLAGS        = r
RANLIB         = echo
```

A.2 HPL Input File

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out        output file name (if any)
8              device out (6=stdout,7=stderr,file)
1              # of problems sizes (N)
39936          Ns
1              # of NBs
64             NBs
0              PMAP process mapping (0=Row-,1=Column-major)
1              # of process grids (P x Q)
2              Ps
2              Qs
16.0           threshold
1              # of panel fact
2              PFACTs (0=left, 1=Crout, 2=Right)
1              # of recursive stopping criterium
4              NBMINs (>= 1)
1              # of panels in recursion
2              NDIVs
1              # of recursive panel fact.
1              RFACTs (0=left, 1=Crout, 2=Right)
1              # of broadcast
1              BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1              # of lookahead depth
1              DEPTHs (>=0)
2              SWAP (0=bin-exch,1=long,2=mix)
64             swapping threshold
```

```
28 0            L1 in (0=transposed,1=no-transposed) form
29 0            U  in (0=transposed,1=no-transposed) form
30 1            Equilibration (0=no,1=yes)
31 8            memory alignment in double (> 0)
```

Acronyms

BLIS BLAS-like Library Instantiation Software.

BSC Barcelona Supercomputing Center.

BZL Barcelona Zettascale Lab.

DGEMM Double Precision General Matrix Multiply.

EPI European Processor Initiative.

FFT Fast Fourier Transform.

FPGA Field-Programmable Gate Array.

GUPs Giga UPdates per second.

HCA Heterogenous Computing Architecture.

HPCC High-Performance Computing Challenge.

HPL High-Performance Linpack.

ISA Instruction Set Architecture.

MEEP Marenostrum Experimental Exascale Platform.

PAPI Performance Application Programming Interface.

PTRANS Parallel matrix transpose.

RISC V Reduced Instruction Set Computing(fifth version).

SDV Software Development Vehicle.

SpMV Sparse matrix–vector multiplication.

VPU Vector Processing Unit.

Bibliography

[1] Alexander Fell, Daniel J Mazure, Teresa C Garcia, Borja Perez, Xavier Teruel, Pete Wilson, and John D Davis. The marenostrum experimental exascale platform (meep). Supercomputing Frontiers and Innovations, 8(1):62–81, 2021.

[2] Raj Jain. The art of computer systems performance analysis, chapter 3.2,3.3. john wiley & sons, 1990.

[3] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. Communications of the ACM, 52(4):65–76, 2009.

[4] Matthew R Guthaus, Jeffrey S Ringenberg, Dan Ernst, Todd M Austin, Trevor Mudge, and Richard B Brown. Mibench: A free, commercially representative embedded benchmark suite. In Proceedings of the fourth annual IEEE international workshop on workload characterization. WWC-4 (Cat. No. 01EX538), pages 3–14. IEEE, 2001.

[5] Piotr R Luszczek, David H Bailey, Jack J Dongarra, Jeremy Kepner, Robert F Lucas, Rolf Rabenseifner, and Daisuke Takahashi. The hpc challenge (hpcc) benchmark suite. In Proceedings of the 2006 ACM/IEEE conference on Supercomputing, volume 213, pages 21–37, 2006.

[6] Henry M Levy and Douglas W Clark. On the use of benchmarks for measuring system performance. ACM SIGARCH Computer Architecture News, 10(6):5–8, 1982.

[7] J. Bjäreholt. Risc-v compiler performance: A comparison between gcc and llvm/clang. Bachelor's thesis, Lund University, 2017.

[8] Sasko Ristov, Marjan Gusev, and Goran Velkoski. Optimal block size for matrix multiplication using blocking. In 2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pages 295–300. IEEE, 2014.

[9] Yorick Bolster. Automatically finding the best blocking size for matrix multiplication. Bachelor's thesis, Leiden University, 2017.

[10] Nick Brown, Maurice Jamieson, Joseph Lee, and Paul Wang. Is risc-v ready for hpc prime-time: Evaluating the 64-core sophon sg2042 risc-v cpu. In Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis, pages 1566–1574, 2023.

[11] CoreMark® An EEMBC Benchmark. https://www.eembc.org/coremark/.

[12] CoE Performance Optimization and Productivity (POP). https://pop-coe.eu.

[13] Collaboration of POP and CoEC to audit reference combustion codes. https://coec-project.eu/news/collaboration-of-pop-and-coec-to-audit-flagship-combustion-codes/.

[14] Barcelona Zettascale Lab. https://bzl.es/en/.

[15] HPC Challenge. https://hpcchallenge.org/hpcc/.

[16] HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers. https://www.netlib.org/benchmark/hpl/.

[17] STREAM: Sustainable Memory Bandwidth in High Performance Computers. https://www.cs.virginia.edu/stream/.

[18] benchfft. https://github.com/FFTW/benchfft.

[19] PAPI. https://icl.utk.edu/papi/.

[20] HCA RISC-V clusters user guide. https://repo.hca.bsc.es/gitlab/epi-public/risc-v-vector-simulation-environment/-/wikis/HCA-RISC%E2%80%90V-clusters-user-guide#introduction.

[21] MEEP Release. https://release.meep-project.eu/index.html.

[22] Milk-V Pioneer. https://milkv.io/pioneer.

[23] SiFive. Hifive unmatched. https://www.sifive.com/boards/hifive-unmatched.

[24] BSC tools for performance analysis. https://tools.bsc.es/.

[25] POP Centre of Excellence. VAMPIRE Report (POP1-PP-013). https://gitlab.pop-coe.eu/documents/reports/-/raw/master/POP1-PP-013-VAMPIRE.pdf, 2017.