Advances in Adapting Memory-Bound CFD Computations to RISC-V Multicore Architecture

Tomasz Olas[1][0000−0002−7286−8023], Lukasz Szustak[1][0000−0001−7429−6981], Roman Wyrzykowski[1][0000−0003−1724−1786], Mateusz Olas[1], and Marco Lapegna[3][0000−0001−9953−1319]

[1] Department of Computer Science, Częstochowa University of Technology, Poland
{olas, szustak, roman}@icis.pcz.pl
[2] University of Naples Federico II, Italy, marco.lapegna@unina.it

Abstract. This paper tackles the challenge of adapting HPC codes to RISC-V architecture for real-world applications with memory-bound numerical codes. The Multidimensional Positive Definite Advection Transport Algorithm (MPDATA) application is the code we study as a use case. This work explores whether the methodology developed in our previous works for Intel and AMD x86 architectures can address performance trade-offs and bottlenecks of multicore RISC-V computing platforms while executing the memory-bound MPDATA code. The explored platforms include: (i) Banana Pi BPI-F3 low-power platform, and (ii) Milk-V system with the 64-core Sophon SG2042 processor. Special emphasis is given to efficient vectorization and using lower-precision computations. Besides performance, energy consumption is studied as well.

Keywords: RISC-V · SG2042 · SpacemiT K1 · CFD · MPDATA application · memory-bound codes · porting applications

1 Introduction

RISC-V is an open standard Instruction Set Architecture (ISA) that enables the royalty-free development of CPUs and a common software stack [6]. Following this community-driven ISA standard, a very diverse set of CPUs suited to a range of workloads have been, and continue to be, developed. While RISC-V has already become popular in some fields, it has yet to gain traction in general-purpose computing, including HPC and AI/ML. In particular, recent advances in RISC-V make it a more realistic proposition for HPC workloads than ever before. An example is the vectorization extension, which gives essential performance advantages for HPC codes but was only standardized in early 2022 as RVV 1.0, so we are only now seeing CPU designs fully implementing this extension [6], [3].

At the same time, the performance of publicly available RISC-V CPUs is still behind even mobile x86 and ARM CPUs, but developments in this area are progressing rapidly. Since the RISC-V software stack includes all necessary tools for application development, it is of considerable interest to study porting real-life codes to computing platforms based on RISC-V. Knowledge gained in this way will allow application programmers to identify bottlenecks in existing approaches to mapping and optimizing codes for HPC architectures, considering

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97632-2_11

the characteristics of available computing platforms and the software stack supporting them. Furthermore, the lessons learned in this way can provide helpful feedback for future hardware and software solutions developers.

This work tackles the challenge of adapting HPC applications to RISC-V platforms for real-life problems with memory-bound codes, for which memory performance is the main factor affecting computation time [17]. The code we study as a use case implements the Multidimensional Positive Definite Advection Transport Algorithm (MPDATA) - a CFD (computational fluid dynamics) algorithm that allows numerical modeling of advection transport phenomena [10].

The paper is organized as follows. Related works are discussed in Section 2. Sections 3 and 4 outline the studied RISC-V platforms and MPDATA application, respectively. Section 5 introduces the parallelization methodology for MPDATA on RISC-V platforms, while the vectorization of codes is described in Section 6. The performance evaluation of MPDATA on RISC-V is presented in Section 7. Section 8 deals with using single precision and evaluating energy efficiency, while Section 9 concludes the paper.

2 Related Works

In recent years, the research community has been actively investigating the capabilities of the RISC-V architecture. Most of the papers focus on the development of the ISA, and applications of RISC-V in some areas like embedded and edge computing, with less attention paid to optimizing the performance of parallel codes on RISC-V CPUs. However, with the development of high-performance RISC-V platforms, bridging the gap between the HPC community and RISC-V technology has become increasingly relevant.

At the moment, not many studies have been published regarding performance analysis and optimization on RISC-V CPUs. In particular, an overview of RISC-V vector extensions and the corresponding computing platforms (at the end of 2022) is given in [6]. In [11], the authors present benchmarking results of OpenFOAM, one of the most widely used frameworks for scientific simulations, comparing the performance and power consumption across devices with ARM and RISC-V architectures. However, only a single-core RISC-V processor is tested, and the analyzed CFD code is not subject to optimization. Paper [19] explores an important computing kernel, the Fast Fourier Transform (FFT), demonstrating that RISC-V-specific optimizations can significantly speed up calculations. In [3], the authors optimized a production CFD code kernel to run efficiently on the FPGA emulator of a RISC-V CPU with long-vector capabilities. Thus, the studies of various authors demonstrate the significant potential of RISC-V technology for HPC while emphasizing the need for new developments in hardware and software optimization methods, using real-world applications.

This work studies MPDATA, a CFD algorithm that represents a general approach to modeling complex geophysical flows from micro to planetary scales and one of the main parts of the EULAG multiscale fluid model [9]. In our previous papers [15, 12, 13], we proposed an adaptation methodology that allowed us

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97632-2_11

to develop the automatic transformation of the memory-bound MPDATA code. The resulting MPDATA code has been carefully optimized for achieving scalable, high performance on ccNUMA multicore platforms with Intel processors of various generations [13, 14] and AMD EPYC Rome architecture [16].

However, there is still a lack of RISC-V-specific optimizations of real-life stencil-based parallel codes, including MPDATA. This research has been conducted in this direction, exploring whether the proposed methodology can address performance trade-offs and bottlenecks of resource-constrained multicore RISC-V platforms while executing the memory-bound MPDATA code. The studied platforms are based on two state-of-the-art commodity CPUs with opposing characteristics in terms of performance and power requirements; they also differ in the vector extension version. While the first CPU has not been covered in the literature so far, the second one has only been studied in papers on optimizing FFT [19] and in works published by Nick Brown et al. on benchmarking the SG2042 CPU using RAJA and NPB suites [5].

3 RISC-V Computing Platforms

Banana Pi BPI-F3 Low-Power Platform

Banana Pi BPI-F3 is an industrial-grade RISC-V development board powered by the SpacemiT K1 RISC-V CPU [2], including eight 64-bit cores operating at a frequency of 1.05 GHz and providing an eight-stage in-order dual-issue pipeline execution. This CPU, launched in late 2023, adheres to the RISC-V 64GCVB architecture and RVA22 standard. Despite a relatively low performance, the attractiveness of this board for the HPC area is that SpacemiT K1 is the world's first commodity processor supporting the vector extension RVV 1.0. SpacemiT K1 provides a 256-bit vector length VLEN with a 128-bit x 2 execution width.

Apart from the in-order execution instead of the out-of-order one adopted in x86 CPUs, the main distinctive feature of the RISC-V platform is a much simpler on-chip memory hierarchy with only L1 and L2 caches, without an L3 cache. Every core has L1 instruction and data caches, each with 32KB. The shared 1MB L2 cache is divided into two 512KB banks. The board integrates 4GB of LPDDR4-2666 memory (up to 16GB) with a single memory controller, providing a modest bandwidth of up 10.6GB/s. At the same time, the important advantage of the platform is the use of the low-power CPU with a TDP of ∼3-5W [2].

Milk-V Pionier Platform with 64-core Sophon SG2042 Processor

Milk-V Pioneer is a developer motherboard based on the 64-core Sophon SG2042 RISC-V CPU in a standard microATX form factor [1]. SG2042 is the first mass-produced, commodity-available, high-core count RISC-V CPU designed for HPC workloads [5], [18]. It runs at 2GHz and is organized in 16 clusters of four XuanTie C920 cores. Clusters are connected through the network-on-chip (NoC) with a 2D mesh topology. Every 64-bit core, designed by T-Head, adopts a 12-stage out-of-order multiple-issue superscalar pipeline execution, supporting the RV64GCV instruction set.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97632-2_11

Each C920 core has L1 instruction and data caches, each with 64KB, while 1MB of L2 cache is shared across a cluster of four cores. All cores in the CPU share 64 MB of the system-level L3 cache, composed of 16 slices connected through the NoC. SG2042 integrates four DDR4-3200 memory controllers and 32 lanes of PCIe Gen4. The board has 128GB of DDR4-3200 memory, providing a maximum bandwidth of 102.4 GB/s - ten times higher than the previous one. At the same time, the board consumes much more energy since the typical power demand of SG2042 is 120W [1]. Finally, unlike the SpacemiT K1 CPU, the SG2042 C920 core provides only version 0.7.1 of the vectorization extension with a vector width of 128 bits. What is important is that, opposite to version 1.0, mainline compilers like gcc and Clang do not support RVV 0.7.1.

4 Overview of MPDATA

MPDATA corresponds to the second-order accurate nonoscillatory iterative algorithms and is defined using a finite-difference scheme over structured rectilinear grids. MPDATA solves the advection of a non-diffusive quantity Ψ in a flow field:

$$\partial\Psi/\partial t \; + \; \mathrm{div}(V\Psi) = 0, \tag{1}$$

where V is the velocity vector [8]. In this work, we focus on modeling 3D advection problems, when MPDATA is defined in a 3D domain of sizes $n \times m \times l$ according to $i-$, $j-$, and $k-$dimensions, respectively.

In general, MPDATA is intended to run long simulations that engage even many thousands of time steps. Each step takes five 3D arrays as input and returns a single 3D array reused in the next step. Each step performs a series of 17 kernels, depending on each other [13]. Every kernel is a 3D stencil code that updates all elements of its output array using a particular pattern.

5 Parallelization of MPDATA on RISC-V Platforms

Methodology for Adapting MPDATA to Multicore Architectures

MPDATA executes a set of stencil kernels with heterogeneous patterns. In the basic version of the parallel code, kernels are executed sequentially, and each kernel is processed in parallel using OpenMP. Data parallelism and vectorization are employed to distribute kernels across cores and vector units. Particularly, `#pragma omp for` directive across outer-most loop ($i-$dimension) is applied to split loop iterations among cores, and `#pragma omp for simd` directive allows us to incorporate vectorization along inner-most loop ($k-$dimension).

Since the basic version is not optimized for cache reusing, its performance is limited by the main memory bandwidth. Consequently, the low operational intensity of each kernel does not allow us to utilize modern CPUs well. In our works [15, 12], suitable optimizations were proposed to exploit multicore ccNUMA platforms more efficiently. The resulting methodology for adapting MPDATA to such platforms consists of the following optimizations [13]:

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97632-2_11

1. *(3+1)D decomposition* explores spatial blocking across kernels, using overlapped tiling with redundant computations. Moreover, loop fusion is used to group all kernels into five packages of kernels. Besides increasing the computational intensity, this approach reduces the main memory traffic and efficiently utilizes L3 and L2 levels of the cache hierarchy.
2. *Partitioning cores into work teams* relieves the overhead of data traffic within the cache hierarchy of the ccNUMA system by setting groups of cores – MPDATA work teams, also called *islands of cores*. The price for mitigating this overhead is the replication of some computations.
3. *Data-flow synchronization* – the aim is to reduce the cost of synchronization by synchronizing only interdependent threads following the data dependencies between the kernels instead of using the barrier approach.
4. *Vectorization of MPDATA kernels* – possible approaches include automatic vectorization, using intrinsics, or even assembly.

In order to parallelize the MPDATA workload across computing resources, the MPDATA domain is evenly split into S sub-domains of size $\frac{m}{S} \times n \times l$, processed in parallel by S hardware teams of cores (islands of cores) available in a given ccNUMA platform. Every team processes a given sub-domain following the (3+1)D decomposition. Each sub-domain is partitioned into blocks with a size that enables efficient utilization of L3 and L2 caches. The successive blocks are processed sequentially, one by one. Each block exploits data parallelism across $i-$ and $j-$dimensions to distribute workload among C_T cores of a given work team. As a result, each MPDATA block is partitioned into a set of C_T sub-blocks. Finally, the vectorization is performed along $k-$dimension for appropriate chunks of data arrays corresponding to the sub-blocks.

Transferring the Methodology to RISC-V Platforms

The expected use of the presented methodology depends on the platform features related primarily to the processor architecture. The lack of ccNUMA domains in the SpacemiT K1 CPU and its simple memory hierarchy with only two-level caches do not justify the usage of the islands-of-cores partitioning. At the same time, an important advantage of this CPU is the possibility of using a compiler-supported vectorization, either automatic or with intrinsics.

Unlike SpacemiT K1, the SG2042 CPU has the three-level cache hierarchy with a reasonably large L3 cache, and what is important is that this last-level cache is divided into slices distributed among four-core clusters connected through the network-on-chip with 4×4 mesh topology. The network is also used to connect four DD4 memory channels. These architecture features justify the need to consider all four optimizations the adaptation methodology provides, including the islands-of-cores partitioning. Moreover, it is advisable to study the scalability of MPDATA execution depending on how the MPDATA workload is distributed across 16 clusters of SG 2042. Finally, the lack of compiler support for version 0.7.1 of the vector extension forces us to incorporate manual vectorization in assembly language to utilize the resources of vector hardware.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97632-2_11

6 Vectorization using Various RVV Extensions

Using vector (or SIMD) unit has yielded notable performance improvements for Intel and AMD processors [13]. In RISC-V processors, the vector (or "V") extension of ISA is dedicated to supporting vectorization. The studied platforms implement different versions of this extension - either the obsolete 0.7.1 version for the SG2042 CPU or the up-to-date RVV 1.0 extension for SpacemiT K1, which forces us to use different approaches. In both cases, the vectorization is carried out for all 17 MPDATA kernels grouped into five packages [12, 13].

SG2042 with RVV 0.7.1 Extension

Due to difficulties in finding a compiler that supports at least vector intrinsics for this CPU, we use a manual vectorization of code fragments written in assembly. For this aim, the gcc compiler in version 9.2.0 is used, allowing us to compile the assembly code corresponding to the `xtheadvector` extension adopted by this processor. Here, this gcc compiler is used exclusively for compiling assembly code fragments, while the remaining C++ code is compiled by the gcc 13.2.0 compiler, which also handles the linking stage for the entire code.

Fig. 1 presents an example of vectorization in assembly language for a function corresponding to a fragment of kernel 4. The code includes three stages: (i) initialization, (ii) processing loop, and (iii) updating pointers and loop control. First, the processing range for a given thread is initialized with `lCoreStart` passed to `t0` register. The loop begins with calculating the remaining range to be processed (`t1 = lCoreEnd - t0`) and setting the vector length `VL` in `t2` register. Next, the kernel operations are performed: loading data from memory (`tmp_A[k]` and `tmp_A[k+1]`) into vector registers `v1` and `v2`, performing subtraction `v3 = tmp_A[k+1] - tmp_A[k]`, and then writing the result to `tmp_f3` array in memory. In the third stage, the data pointers (`tmp_A` and `tmp_f3`) are advanced by the number of processed elements, while index `t0` is incremented by `VL`. The loop is repeated until the entire range has been processed. The other functions being vectorized have a similar structure, but typically, they handle more arguments (up to 24) and perform more complex operations.

A major challenge for assembly vectorization is optimally using registers to avoid accessing memory. To this end, unused registers such as `s0` and `gp` are leveraged. Their states are saved on the stack before the function begins and restored upon completion.

SpacemiT K1 with RVV 1.0 Extension

Since the state-of-the-art versions of Clang and gcc compilers incorporate vectorization support for the RVV 1.0 extension, it becomes possible to burden a compiler with automatic code vectorization. The codes of MPDATA kernels must be suitably modified for this aim - in a way similar to x86 [13]. Clang directives, including `#pragma clang loop vectorize(enable)`, are used to enforce vectorization and pass additional configuration parameters. We also leverage compiler-supported vectorization with intrinsics to improve the performance obtained by auto-vectorization. This solution makes programming vector operations much more productive than using assembly.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97632-2_11

```
    # Arguments:
    # a0: pointer to tmp_A (input)
    # a1: pointer to tmp_f3 (output)
    # a2: lCoreStart (starting index)
    # a3: lCoreEnd (end index)
    # Initialization
    mv t0, a2           # t0 <- lCoreStart (current index)
    # Processing loop
loop:
    sub t1, a3, t0      # t1 <- lCoreEnd - t0
    vsetvli t2, t1, e64 # Set VL (Vector Length) for 64-bit elements
    # Load input data
    vle.v v1, (a0)      # v1 <- tmp_A[k]
    addi t3, a0, 8
    vle.v v2, (t3)      # v2 <- tmp_A[k+1]
    vfsub.vv v3, v2, v1 # v3 <- tmp_A[k+1] - tmp_A[k] = v2 - v1
    vse.v v3, (a1)      # Store the result into tmp_f3
    # Updating pointers
    slli t4, t2, 3      # t4 <- VL * 8 (data size in bytes)
    add a0, a0, t4      # Advance tmp_A pointer by VL * 8 bytes
    add a1, a1, t4      # Advance tmp_f3 pointer by VL * 8 bytes
    add t0, t0, t2      # Increment t0 index by VL
    # Loop control
    blt t0, a3, loop    # If t0 < lCoreEnd, repeat the loop
    ret                 # Return from the function
```

Fig. 1: Vectorization of a function implementing vector subtraction.

7 Performance Evaluation of MPDATA on RISC-V

7.1 Evaluation Metodology

For Banana Pi BPI-F3, the experiments presented in this section focus on evaluating the influence of optimization steps on the execution time of MPDATA. Besides the basic, non-optimized code, the studied versions of MPDATA embrace the (3+1)D decomposition, data-flow synchronization, and three variants of vectorization: (i) automatic, (ii) using intrinsic, and (iii) in assembly. The measured execution times are obtained for the Clang compiler in version 20.0.0git. They correspond to 100 time steps and 3D grid of size $512 \times 480 \times 64$.

For the second platform, the range of experiments is much broader. First, the memory throughput is tested, followed by benchmarking the platform performance and scalability with the NPB (NAS Parallel Benchmark) test suite. Then, the impact of optimizations on performance is tested together with evaluating the scalability of codes, assuming allocation of 1, 2, 3, or 4 threads per each four-core cluster of SG2042. While gcc 9.2.0 is used only for compiling assembly parts, the gcc 13.2.0 compiler handles compiling and linking for the entire code. For both platforms, computations are performed in double precision.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97632-2_11

Table 1: Execution time for running different versions of MPDATA code on Banana Pi BPI-F3, where Std denotes standard deviation.

Code version	T_{mean} [s]	T_{med} [s]	Std	T_{Min}[s]	T_{Max}[s]
Basic	361.9	383.7	37.3	316.5	397.6
(3+1)D	235.2	236.1	4.15	229.7	241.3
(3+1)D + auto-vec	203.5	205.1	4.2	197.4	210.1
(3+1)D + intr vec	174.1	172.9	4.5	168.9	182.2
(3+1)D + asm vec	165.5	165.0	4.3	158.8	171.9
(3+1)D + df synchr	216.7	217.5	3.4	211.4	220.0
(3+1)D +df synchr +auto-vec	184.4	183.7	3.9	179.4	189.9
(3+1)D +df synchr +intr vec	160.4	161.4	3.5	154.7	166.4
(3+1)D +df synchr +asm vec	157.1	157.6	6.1	146.3	167.2

7.2 Evaluation of Banana Pi BPI-F3 Platform

Table 1 presents execution times for nine versions of MPDATA, starting with the non-optimized code and ending with three codes implementing (3+1)D decomposition together with the data-flow synchronization (denoted as df synchr) and vectorization using either auto-vectorization (auto-vec), intrinsic (intr vec) or assembly (asm vec). For each version, we provide the mean T_{mean} and median T_{med} values of the execution time obtained for 10 repeated measurements, giving results in the range from T_{min} to T_{max}.

The results in Table 1 prove the effectiveness of the proposed optimizations. The (3+1)D decomposition achieves the most significant effect, which decreases the median execution time by 1.63 times. Subsequently, even auto-vectorization is more productive than data-flow synchronization (DFS). At the same time, leveraging assembly-based vectorization yields considerably better results than auto-vectorization, almost avoiding the usage of DFS. In fact, vectorization in assembly without DFS permits decreasing the median execution time 2.33 times against the basic version, while mixing DFS with assembly-based vectorization speeds up MPDATA 2.43 times.

7.3 Evaluation of Milk-V Pionier Platform

Benchmarking the Platform

Measurements of memory throughput (in MB/s) for various thread (core) numbers (Table 2) show that the total throughput rises only slightly with increasing the thread number, which radically reduces the throughput per thread - e.g., from 2591.9 MB/s (16 threads) to 738.8 MB/s (64 threads) for `daxpy`, or 3.5 times. This decrease in throughput per thread with scaling thread number correlates with the scalability of codes from the NPB suite (Fig.2). Only embarrassingly parallel computations are scalable up to 64 cores. Some codes feature good scalability up to 32 cores, with a slight growth for 48 cores (LU) or a fall after 32 cores (CG). Finally, benchmarks such as MG provide only a slight performance growth for 32 cores compared to 16, with a fall afterward.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97632-2_11

Table 2: Total throughput B(P) and per thread B1(P) in MB/s for P=16 and P=64 threads and various functions measured on SG2042.

Function	B(16)	B1(16)	B(64)	B1(64)
Copy	36215.8	2263.5	44940.7	702.2
Scale	36438.0	2277.4	44874.6	701.2
Add	40847.4	2553.0	47626.5	743.2
Triad	38731.5	2420.7	45744.6	714.8
Daxpy	41470.4	2591.9	47285.8	738.8

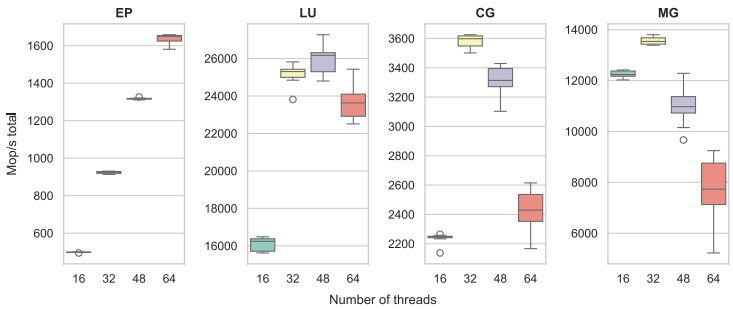

Fig. 2: Scalability of selected NPB codes (class C) on SG2042, where EP, LU, CG and MG are for embarrassingly parallel, LU decomposition, conjugate gradient and multigrid codes, respectively.

Evaluating Scalability and Efficiency of Optimization Steps

Table 3 shows execution times for various core numbers and four versions of MPDATA: the basic, non-optimized code, and three codes with growing optimization levels - from using only the (3+1)D decomposition, through leveraging also the DFS step, to the code including vectorization as well. We have not yet been able to leverage the islands-of-cores optimization step to speed up the computation. Consequently, each work team includes only a single core. Taking advantage of this optimization will be the subject of our future work.

The median values T_{med} from Table 3 are the basis of our further analysis. They are also used to calculate the performance of MPDATA codes (in Gflop/s) presented in Fig. 3. The obtained results once again confirm the effectiveness of the proposed optimizations. While the basic code is not scalable, yielding only a slight performance gain when going from 16 to 32 cores, with a radical slowdown afterward, the resulting optimized code provides quite good scalability, reducing the median execution time by 1.5 and 2.34 times for 32 and 64 cores, respectively, when compared to 16 cores. Even more appealing is the final speedup S_F yielded by the fully optimized code against the basic one: $S_F = 47.12/6.60 = 7.14$ times.

The analysis of the impact of various optimizations on performance is of considerable interest. This impact, measured by the speedup achieved by a given code against the code with a lower level of optimization, is changing with in-

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97632-2_11

Table 3: Execution times for running various MPDATA versions on SG2042.

Code version	P	T_{mean} [s]	T_{med} [s]	Std	T_{min}[s]	T_{max}[s]
Basic	16	51.85	51.38	1.31	49.86	54.39
Basic	32	47.33	47.12	1.26	45.75	49.19
Basic	48	94.85	90.97	14.45	79.42	116.76
Basic	64	128.59	123.56	20.15	104.89	168.96
(3+1)D	16	28.20	28.34	0.33	27.52	28.50
(3+1)D	32	20.04	20.33	1.53	16.94	21.94
(3+1)D	48	20.80	21.53	1.56	17.76	22.17
(3+1)D	64	23.70	16.24	16.36	9.18	61.30
(3+1)D + DFS	16	25.49	25.55	0.18	25.23	25.71
(3+1)D + DFS	32	15.22	15.22	0.23	14.85	15.62
(3+1)D + DFS	48	13.54	13.50	0.55	12.88	14.47
(3+1)D + DFS	64	9.18	7.92	2.78	7.87	14.82
(3+1)D + DFS + vec	16	15.7	15.7	0.2	15.5	16.0
(3+1)D + DFS + vec	32	10.5	10.5	0.2	10.3	10.9
(3+1)D + DFS + vec	48	10.4	9.9	1.4	8.9	13.2
(3+1)D + DFS + vec	64	10.55	6.60	6.63	6.50	22.90

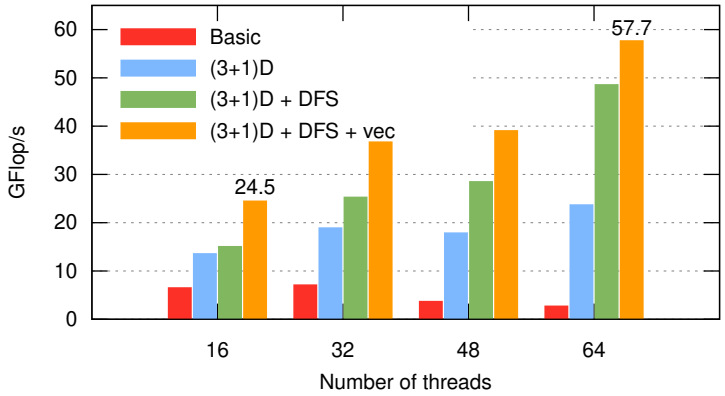

Fig. 3: Performance (in Gflop/s) of different versions of MPDATA on SG2042.

creasing the core number. Thus, the impact of the (3+1)D decomposition is increasing from 1.9 times on 16 cores, through 2.32 times on 32 cores, to 7.6 times on 64 cores. The impacts of two subsequent optimizations are interrelated. While on 16 cores, switching on DFS speeds up MPDATA only $S_{DFS} = 1.11$ times, and adding vectorization allows us to shorten the execution time by S_{vec} =1.63 times, for high numbers of cores, these speedups are as follows: $S_{DFS} = 1.34$, $S_{vec} = 1.45$ on 32 cores, and $S_{DFS} = 2.05$, $S_{vec} = 1.2$ for 64 cores. The increased impact of DFS can be explained by the performance bottlenecks of NoC which heavily favor increasing computing locality achieved by DFS. At the same time, the reason for the decreased impact of vectorization lies in the limited memory bandwidth.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97632-2_11

Performance analysis based on the Roofline model
Fig. 4 presents the preliminary Roofline model [16] built to analyze MPDATA performance on the Milk-V platform. This model expresses the attainable performance AP (in Gflop/s) as a function of the operational intensity O (in flop/B). Fig. 4 shows that the proposed optimizations allow us to increase the intensity significantly. While for 17 kernels of the basic code, $0.14 \leq O \leq 0.54$, we have $0.72 \leq O \leq 1.04$ for five packages of the optimized code. Multiplying the intensity O by the measured memory bandwidth $B_{DRAM} = 47.6$ GB/s permits us to estimate the range of AP from 34.6 to 58.1 Gflop/s. Here AP = 39.5 Gflop/s for P_1 package, which takes ∼40% of the total execution time, and AP = 58.1 Gflop/s for P_3 package, which takes only ∼15% of the total time.

Comparing these estimations of AP with the performance MP = 57.7 Gflop/s measured on 64 cores (Fig. 3), we conclude that, like x86 architectures [13, 16], the packages $P_2 - P_5$ leverage the L3 cache. Its bandwidth is significantly high, leading to an adequate increase in the estimation of the attainable performance AP. Our future work will focus on reliably incorporating the impact of L3/L2 cache performance characteristics into the model.

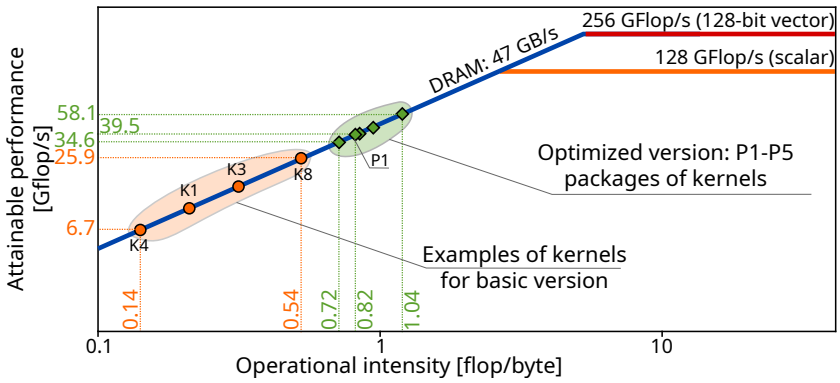

Fig. 4: The Roofline model for the double-precision MPDATA on Milk-V.

8 Using Single Precision for Improving Performance and Evaluation of Energy Efficiency

Using Single Precision
Paper [7] showed that in some application domains, using the single precision format of data instead of double precision is enough to provide the required accuracy. This transition to lower precision increases the performance of computing units and allows more efficient utilization of the available memory bandwidth.

Fig. 5 compares the median execution time of the basic and fully optimized single-precision codes on SG2042. One can conclude that while using single precision for the basic code allows us only to slightly shorten the execution time - from

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97632-2_11

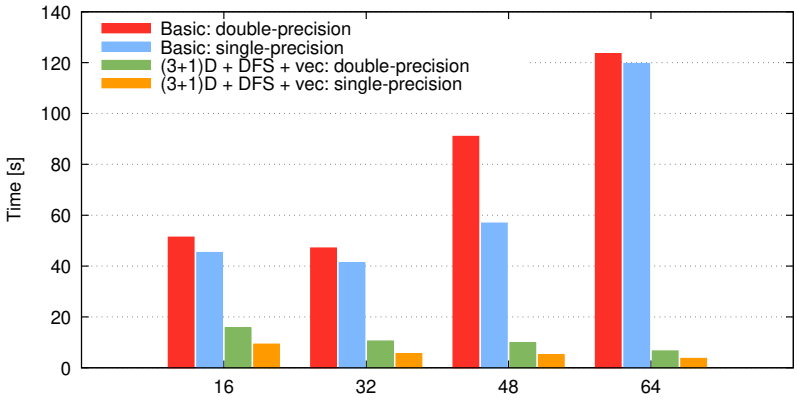

Fig. 5: Performance of single-precision codes on SG2042 versus double-precision.

$T_b^d(32) = 47.12$ s for double precision to $T_b^s(32) = 40.0$ s (both values achieved on 32 cores), the optimized code is quite scalable, accelerating the execution by 1.67 and 2.54 times for respectively 32 and 64 cores when compared to 16 cores. Interestingly, unlike the double-precision code, the contribution of DFS to this acceleration decreases with scaling the core number while the contribution of vectorization grows. Consequently, the final speedup S_F achieved by the fully optimized code versus the basic one is given by $S_F = T_b^s(32)/T_o^s(64) = 40.0/3.65 = 10.96$. An even more practically interesting result is that the transition from double to single precision in the optimized code permits improving the performance by the ratio of $T_o^d(64)/T_o^s(64) = 6.60/3.65 = 1.80$, i.e., slightly less than twice.

Evaluation of Energy Efficiency

By reducing the execution time, the proposed optimizations also allow us to decrease the energy consumed by MPDATA [14, 16]. We employ the Yokogawa WT310 digital power meter to obtain accurate and reliable measurements of the energy and power consumed by the tested platforms. The power meter passes the power to the platform under the load and implements measurements in real time. The USB interface and YokoTool software allow us to collect data without a noticeable influence on energy/power measurements.

Fig. 6 summarizes the measurement results for double precision. For both platforms, the proposed optimization improves computations' energy efficiency significantly. These improvements are in line with the reduction in execution time. At the same time, energy gains can be noticeably higher than time reductions. For example, for 64 cores running on Milk-V Pionier, the energy consumption is decreased by more than 11.5 times compared to the basic code, while the code is executed only 7.14 times shorter. What is also interesting is that the energy consumed by Milk-V is decreasing with the increasing number of cores.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97632-2_11

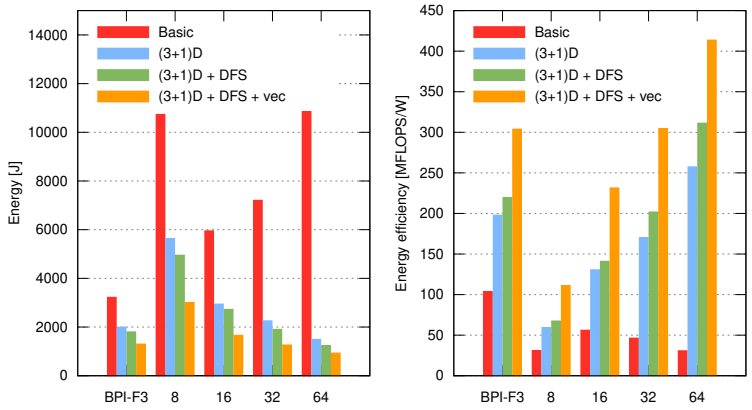

Fig. 6: Energy consumption in joules (left) and energy efficiency in MFlop/s/W (right) for double-precision MPDATA on two tested platforms.

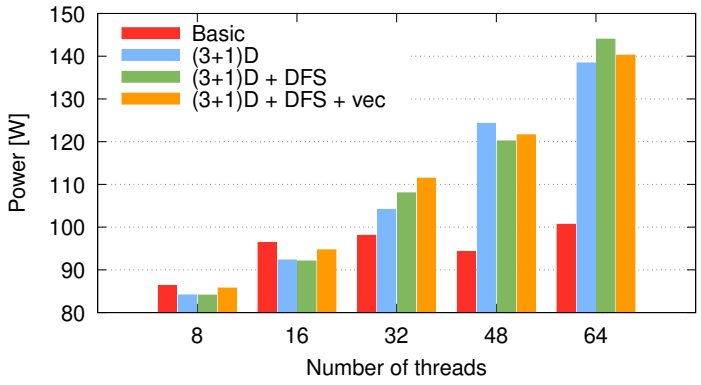

Fig. 7: Average power consumption in watts for double-precision versions of MPDATA on the Milk-V platform.

In particular, the energy consumed for 64 cores is 1.79 and 1.36 times lower than the energy required for 16 and 32 cores, respectively.

We also compare the energy efficiency (in MFlop/s/W) of MPDATA codes for both platforms. While Banana BPI-F3 beats the second platform for eight cores, already by using 32 cores Milk-V Pionier catches up with BPI-F3, and by exploiting 64 cores Milk-V beats the opponent by 36%. The energy efficiency evaluation is finished by analyzing the average power consumption of MPDATA versions on both platforms. This analysis shows that BPI-F3 consumes practically the same power for all versions - about 8.3 W. The power consumed by Milk-V is much higher (Fig. 7). It starts with about 85 W for all versions running on eight cores, while already for 32 cores, we can observe some increase in

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97632-2_11

average power consumed by more optimized versions of MPDATA compared to the basic one. This increase is particularly visible for 64 cores when the average power consumption increases from about 100 W for the basic version to about 140 W for the most optimized code.

9 Conclusions and Future Work

The RISC-V architecture is rapidly expanding. The current level of infrastructure development allows porting state-of-the-art software onto existing RISC-V platforms and identifying the most promising RISC-V-specific approaches to improving performance. In this paper, we tackle the challenge of adapting HPC codes to RISC-V architecture for real-world applications with memory-bound numerical codes such as MPDATA CFD application. Our findings can be summarized as follows:

1. We demonstrate that our optimization methodology developed previously for Intel and AMD x86 architectures can efficiently address performance trade-offs and bottlenecks of two resource-constrained, multicore RISC-V platforms while executing the memory-bound MPDATA code.
2. To efficiently utilize the vector hardware of the considered CPUs, besides using the auto-vectorization for the SpacemiT K1 CPU, we develop a manual vectorization of MPDATA codes for both versions of the RVV extension and different numerical precisions.
3. The experimental evaluation of MPDATA codes on the Sophgon SG2042 CPU shows that, unlike most tests from the NPB test suite and the basic MPDATA code, our optimized code is scalable up to all 64 cores of this CPU.
4. In double precision, the code optimizations allow us to speed up computation more than 7 times compared to the original code. For single precision, this speedup is even higher, exceeding 10 times.
5. The experimental evaluation of energy consumption demonstrates convincingly the energy savings achieved by the code optimizations performed for both platforms. In particular, on the Milk-V Pionier platform, energy consumption is reduced radically - by more than 11 times.
6. The evaluation of energy efficiency shows that while for eight cores, the Banana BPI-F3 low-power platform beats the Milk-V Pionier platform by more than two times, already by using 32 cores, Milk-V catches up with BPI-F3, and by exploiting 64 cores, Milk-V beats the opponent by 36%.

Below, we outline three possible directions for future work. The first one concerns using the islands-of-cores step in the optimization methodology, as well as including the L3/L2 cache performance characteristics in the performance analysis of Section 7. The second direction involves porting other real-life applications, including ML/AI workloads such as Bayesian network learning considered in paper [4]. The third direction concerns exploiting long vectors [3] offered by upcoming RISC-V platforms.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97632-2_11

References

1. Milk-V Pioneer. https://milkv.io/docs/pioneer/ (2024)
2. Banana Pi BPI-F3. https://wiki.banana-pi.org/Banana_Pi_BPI-F3 (2024)
3. Blancafort, M., et al.: Exploiting long vectors with a CFD code: a co-design show case. In: 2024 IEEE Int. Conf. Parallel and Distributed Processing Symposium (IPDPS). pp. 453–464 (2024)
4. Bratek, P., Szustak, L., Zola, J.: Parallel Auto-Scheduling of Counting Queries in Machine Learning Applications on HPC Systems. In: Euro-Par 2023: Parallel Processing Workshops. vol. 14325, pp. 327–333. Lect. Notes Comp. Sci. (2024)
5. Brown, N., Jamieson, M.: Performance characterisation of the 64-core SG2042 RISC-V CPU for HPC. In: ISC High Performance 2024 Int. Workshops. vol. 15058, pp. 354–377. Lect. Notes Comp. Sci. (2024)
6. Lee, J., Jamieson, M., Brown, N., Jesus, R.: Test-Driving RISC-V Vector Hardware for HPC. In: ISC High Performance 2023 Int. Workshops. vol. 13999, pp. 419–432. Lect. Notes Comp. Sci. (2023)
7. Rojek, K., Halbiniak, K., Kuczynski, L.: CFD code adaptation to the FPGA architecture. Int. J. High Performance Computing Applications **35**(1), 33–46 (2015)
8. Rosa, B., et al.: Adaptation of multidimensional positive definite advection transport algorithm to modern high-performance computing platforms. Int. Journal of Modeling and Optimization **5**(3), 171–176 (2015)
9. Smolarkiewicz, P., Charbonneau, P.: EULAG, a computational model for multiscale flows: An MHD extension. J. Computational Physics **236**, 608–623 (2013)
10. Smolarkiewicz, P., Margolin, L.: MPDATA: A Finite-Difference Solver for Geophysical Flows. J. Computational Physics **140**(2), 459–480 (1998)
11. Suarez, D., Almeida, F., Blanco, V.: Comprehensive analysis of energy efficiency and performance of ARM and RISC-V SoCs. The Journal of Supercomputing **80(9)** (2024)
12. Szustak, L.: Strategy for Data-Flow Synchronizations in Stencil Parallel Computations on Multi-/Manycore Systems. The Journal of Supercomputing **74**(4) (2018)
13. Szustak, L., Bratek, P.: Performance portable parallel programming of heterogeneous stencils across shared-memory platforms with modern Intel processors. Int. J. High Performance Computing Applications **33**(3), 507–526 (2019)
14. Szustak, L., Wyrzykowski, R., T., O., Mele, V.: Correlation of Performance Optimizations and Energy Consumption for Stencil-Based Application on Intel Xeon Scalable Processors. IEEE Trans. Parallel Distrib. Syst. **31**(11) (2020)
15. Szustak, L., et al.: Adaptation of MPDATA heterogeneous stencil computation to Intel Xeon Phi coprocessor. Scientific Programming **2015** (2015)
16. Szustak, L., et al.: Architectural Adaptation and Performance-Energy Optimization for CFD Application on AMD EPYC Rome. IEEE Trans. Parallel Distrib. Syst. **32**(12), 2852–2866 (2021)
17. Volokitin, V., et al.: Case Study for Running Memory-Bound Kernels on RISC-V CPUs. In: Paralel Computing Technologies (PaCT 2023). vol. 14098, pp. 51–65. Lect. Notes Comp. Sci. (2023)
18. Wei, C.: SG2042 Technical Reference Manual. https://github.com/milkv-pioneer/pioneer-files/blob/main/hardware/SG2042-TRM.pdf (2023)
19. Zhao, X., Zhang, X., Zhang, Y.: Optimization of the FFT Algorithm on RISC-V CPUs. In: ISC High Performance 2023 Int. Workshops. vol. 13999, pp. 515–525. Lect. Notes Comp. Sci. (2023)

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97632-2_11