2024 IEEE International Conference on High Performance Computing and Communications (HPCC) | 979-8-3315-4046-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/HPCC64274.2024.00072

swYAKL: A Data Parallel Runtime on Manycore Architecture

Yanwei Ye[1], Junshi Chen[1,2], Hong Qian[1], Kunxian Lin[1], Yuanhang Li[1], Hong An[1,2]
[1]University of Science and Technology of China, Hefei, Anhui, China
[2]Laoshan Laboratory, Qingdao, Shandong, China
yanweiye@mail.ustc.edu.cn, cjuns@ustc.edu.cn, {qh1986, linkx, voyage}@mail.ustc.edu.cn, han@ustc.edu.cn
Corresponding author: Junshi Chen, Hong An, cjuns, han@ustc.edu.cn

***Abstract*—The traditional architecture of supercomputers comprises a control unit and heterogeneous accelerators with distinct memory spaces, necessitating developers to code separately for each accelerator. The notion of performance portability has been introduced to facilitate the swift adaptation of a singular codebase across multiple heterogeneous accelerators, enabling the same code to efficiently harness the performance of various platforms with minimal alterations. Frameworks such as Kokkos, RAJA, and YAKL accomplish this through a data parallel model, primarily targeting SIMT devices but also applicable to SIMD devices. With the advent of manycore architectures that provide high levels of parallelism and performance, it becomes imperative to extend performance portability to these architectures, which currently require users to manually partition and map tasks to fully exploit their capabilities. This paper introduces a data parallel programming runtime for the Sunway manycore architecture and adapts YAKL to this platform, termed swYAKL. This runtime capitalizes on the computational characteristics of Sunway, including task division and mapping, and supports Sunway's CPE LDM and native vectorization. It employs a specialized task division method for multi-dimensional stencil kernels to leverage the DMA capabilities of Sunway's CPE. Performance evaluations using several common computational kernels and a proxy application named miniWeather indicate that swYAKL achieves speedups exceeding 100x compared to execution on Sunway's MPE for various workloads, and under specific conditions, it surpasses the performance of the NVIDIA A100 GPU.**

***Index Terms*—Parallel Computing, HPC, Performance Portabiltiy, Sunway, Stencil**

I. Introduction

Supercomputer today adhere to a conventional model that includes nodes consisting of a control unit, at least one compute unit equipped with its own storage, and various architectures that necessitate coding specific to each architecture. To tackle this issue, the concept of performance portability [1] was introduced, enabling code to execute across different platforms with minimal to no modifications required. Frameworks like Kokkos [2], RAJA [3], and YAKL [4] offer performance portability through a unified data parallel model. In this parallel model, task is a functor running on a set of data, as the functor is unchanged in a task, it is easy to parallelize these type of task by partition data into several compute units.

Most parallel programming runtimes focus on traditional Single Instruction, Multiple Threads (SIMT) architectures, as these architectures provide significant parallelism through hardware-implemented low-overhead context switching and automatic vectorization, thereby simplifying programming while delivering high performance. Conversely, contemporary manycore processors, which contain a substantial number of SIMD cores, require developers to handle task scheduling and partitioning via software. For example, with the Chinese SW26010pro manycore processors, developers must create their own task partitioning strategies, manually map these strategies to core groups on the processor, perform operations such as Direct Memory Access (DMA) to main memory, and manually vectorize tasks, making application development on this processor more complex. To design an efficient parallel runtime for these processors, it is crucial to implement an effective task partitioning and mapping strategy, as well as incorporate support features like vectorization, to fully harness the performance potential of these platforms.

This study introduces swYAKL, a parallel programming library meticulously designed for data parallel programming on the Sunway manycore architecture. swYAKL extends the YAKL programming interface, integrating dynamic task scheduling and mixed compilation methods to facilitate parallel programming on the Sunway architecture. Additionally, it offers platform-specific features such as Local Data Memory (LDM) management and native Single Instruction, Multiple Data (SIMD) support. This library significantly simplifies the development process on the Sunway platform while maintaining performance portability across it. Our contributions are as follows:

- Proposing task scheduling mechanisms tailored for the Sunway manycore system, enabling task dispatch to CPEs through a unified interface.
- Proposing an optimized runtime task partitioning strategy for stencil computations with over 2D index space to improve DMA operations utilization.
- Evaluating the runtime using various kernels and a proxy application, observing speedups ranging from approximately 5x to over 50x across different workloads comparing to MPE only version. In particular workloads, a single core group outperforms the NVIDIA A100 GPU.

979-8-3315-4046-3/24/$31.00 ©2024 IEEE
DOI 10.1109/HPCC64274.2024.00072

II. Related Work

A. Data Parallel Framework

Heterogeneous computing has increasingly become the dominant approach for achieving high-performance computing. Supercomputers may incorporate components from various manufacturers due to numerous reasons. For instance, in the TOP500 list as of November 2023 [5], among the top 10 supercomputers, components designed and manufactured by NVIDIA, AMD and Intel are evident. These components feature distinct hardware architectures, and each manufacturer has developed specialized acceleration APIs for their use.

The concept of performance portability aims to maximize the utilization of a platform's performance with minimal or no modifications to the source code. Parallel frameworks such as Kokkos [2], RAJA [3], and YAKL [4] facilitate this by managing macro switches at compile time to support various platforms. These frameworks are all based on a data-parallel programming model, which partitions tasks according to the index space of the task data, and thus they are also referred to as data-parallel frameworks.

However, most of these frameworks do not provide support to most manycore architecture currently, necessitating significant refactoring for applications moving to these architectures. Developers must invest time in adapting to the target platform's APIs rather than optimizing algorithms.

B. Manycore Runtime Design

Manycore processors are categorized as processors that are composed using a large number of identical cores with near-complete normal CPU functionality, such as the Hammerblade [6], Cerabras CS-3 [7], SW26010 Pro [8]; or using more processors composed of stream processors with only a single function, such as NVIDIA GPUs.

Runtimes designed for GPU-like heterogeneous acceleration devices are relatively more mature, and these runtimes are usually designed and implemented by the manufacturers themselves, such as CUDA [9], HIP [10], etc., or a unified specification is jointly specified and implemented by several manufacturers, such as OpenCL [11], OpenACC [12]. These runtimes include more than just basic parallel programming interfaces; they also provide a range of libraries that can simplify programming, such as cublas for the CUDA platform.

Advancements in runtime environments for manycore processors are on going. OpenMP [13] is a popular framework for parallel computing on high-performance CPUs, including Japan's Fugaku supercomputer [14]. Intel's OneAPI [15] boosts parallel performance on Intel systems, used in the AURORA supercomputer. Wang et al. [16] developed a dynamic scheduling system with a software-centric cache for RISC-V processors with BIG.little architecture. Cheng et al. [17] enhanced task scheduling with a work-stealing mechanism in Scratchpad Memory (SPM) for better distribution and balance in the HammerBlade architecture. Cerebras introduced CSL for its CS processors, supporting advanced inter-core communication and SRAM management. For Sunway architecture, the athread runtime [18] provides low-level task-based parallel programming model with API to support some platform specific features such as DMA of CPE.

III. Background

A. Sunway Manycore Architecture

Manycore architectures, which feature dozens to hundreds of cores per chip, can be categorized into two types: those composed of a large number of stream processors capable of performing specific operations, known as SIMT architectures, such as GPUs; and those with an architecture similar to traditional CPUs but with a significantly higher core count, exemplified by the SW26010pro [8], Fujitsu A64FX [14], Cerebras WSE [7] and HammerBlade.

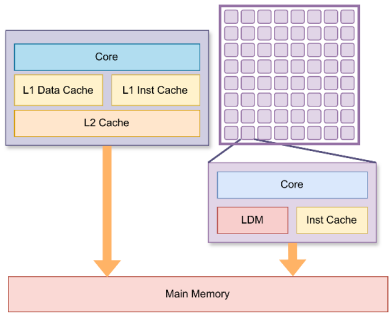


Fig. 1. Architecture of Sunway's single core group, there are 6 core groups in one SW26010pro.

The SW26010pro [8] processor, which is the focus of this study, drives the latest Sunway supercomputer. Illustrated in Figure 1, this processor boasts a shared-memory heterogeneous architecture that divides the computing resources into six core groups. Each group comprises an MPE (Management Processing Element) and 64 CPEs (Computing Processing Elements), with a collective access to 16GB of DDR4 memory. Within each core group, a uniform address space facilitates memory access—MPEs utilize ld/st instructions for accessing the master memory, while CPEs make use of gld/gst instructions or DMA for efficient read and write.

The CPE features a software-configurable 256KB Scratchpad Memory, known as LDM, along with a 32KB instruction cache. The LDM of the CPE can allocate up to 128KB for autocache, with the remaining space operable similarly to main memory. Data within the LDM can be directly transferred between CPEs via RMA operations, substantially lowering access overhead and enhancing performance in scenarios necessitating data exchange between CPEs, including operations like reduction, matrix multiplication, etc.

B. YAKL's Programming Model

In this paper, we utilize the YAKL front-end interface [4], a programming framework that employs the `parallel_for` template to express parallelism. The control device, upon initiating a task through this interface, constructs and packages the task into an object, which is then sent to the computing

device for execution. Initially, YAKL's role was limited to task building, with task scheduling managed by the computing device vendor's platform runtime.

```
Array<double, 2, memDevice, styleC>
   a("a", N, M), b("b", N, M), c("c", N, M);

parallel_for("Kernel_Name", Bounds<2>(N, M),
   YAKL_LAMBDA (int i, int j) {
      c(i, j) = a(i, j) + b(i, j);
   }
}
```

Listing 1. An example code snippet of YAKL program

YAKL employs an `Array` data structure to manage data storage and access on both control and computing devices, requiring developers to ensure data is correctly positioned. To facilitate data transfer between devices, YAKL offers interfaces like deep copy and mirror creation. An example YAKL program in Listing:1 demonstrates declaring data with the `Array` interface and capturing it in a lambda within `parallel_for`. The `YAKL_LAMBDA` macro, specific to YAKL, adapts to different platforms by expanding into respective lambda tags.

IV. Method

A. Dynamic Task Scheduling on Heterogeneous Cores

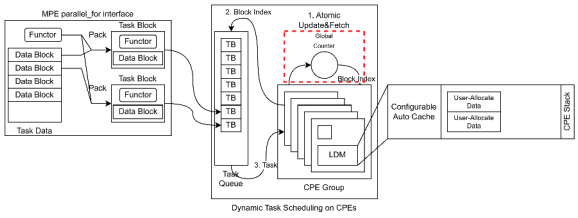


Fig. 2. The swYAKL runtime workflow includes several key steps. First, the MPE divides tasks by packaging functors and data into task blocks, which are then sent to the CPEs. The CPEs dynamically obtain data from the task queue by updating a shared counter. Each CPE has a local data memory space that can be configured as a cache or allocated according to user specifications.

In the original version of YAKL, task scheduling and task partitioning were dependent on the runtime of the target platform. For instance, with OpenMP Target, this process utilized the static scheduler within OpenMP. Similarly, for CUDA and HIP Targets, it leveraged the scheduling or mapping policies inherent in the runtime supplied by the manufacturer. However, the Sunway manycore architecture previously lacked a runtime capable of offering analogous functionalities for our utilization.

1) Task Partition Strategy: The initial issue to address is the formulation of the task partitioning strategy. Most Single Instruction, Multiple Threads (SIMT) devices, such as GPUs, possess intrinsic capabilities for task scheduling and partitioning by their hardware, along with inherent support for auto-vectorization. In contrast, on Single Instruction, Multiple Data (SIMD) devices, developers are required to manually partition tasks and map them to the hardware.

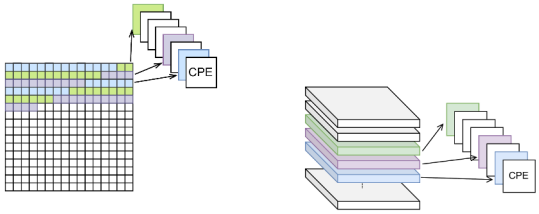


Fig. 3. Partition strategy for 2D and 3D tasks: (a) Sequential partition strategy for task with 2D index space, (b) Layer-based partition strategy for task with over 3D index space.

In the swYAKL framework, we utilize two distinct methods for task partitioning. For tasks with one-dimensional (1D) and two-dimensional (2D) indices, we linearize and partition them sequentially, as shown in Figure 3(a). This approach is cache-friendly, as the data within each task pack remains contiguous. However, applying this strategy to tasks with three-dimensional (3D) indices or higher results in a large number of task packs, leading to considerable scheduling overhead. To mitigate this issue, while maintaining parallelism, we introduced a layer-based partitioning strategy, as shown in Figure 3(b). This strategy partitions the original data space along its second-last dimension, thereby ensuring each task pack contains a layer of data. The layer-based partitioning significantly enhances the computational density of each task pack by increasing the workspace of each pack, thereby reducing the scheduling overhead.

2) Dynamic Scheduler for Sunway Manycore: To enhance the adaptability of our runtime across a wider array of workloads, especially those with load imbalance issues, we have developed a dynamic scheduler, as shown in Figure 2. This scheduler specifically addresses the load imbalance problems frequently encountered in real-world applications, particularly those involving control flow within kernels. Comparing with exist static task schedule strategy, dynamic scheduling on Sunway shows similar performance on load-balance workload but out-performed significantly on load-unbalanced workload.

In swYAKL, a task counter is used to simulate a task queue, as shown in the referenced figure. This method requires all CPEs involved in the scheduling to share a task counter. The CPEs update this counter using atomic operations to get a task block for each execution. The task blocks are created using the task partition strategy discussed earlier. Since this scheduling model can't predict the next task a CPE will execute, swYAKL doesn't support cache prefetching at the task block level.

In other parallel computing models, like the `std::thread` target in Kokkos, dynamic task scheduling is implemented using a work-stealing approach. This strategy involves frequently checking the state of other task queues by the current execution unit. However, Sunway architecture lacks the atomic operations like test-and-set, which are

essential for implementing such a runtime. Using alternatives like mutex or semaphore (CPE lock) on Sunway is slow because they are simulated by software. As a result, swYAKL does not implement a work-stealing runtime due to these limitations.

When `parallel_for` constructs a task, the MPE first reset the shared task counter to 0 and initializes necessary data structure for CPE on the main memory. Typically these data structures contain the reference to the task functor, the execution range of the task, and the range of the task block, etc. The CPE then use DMA to move the these data structures from the main memory into its LDM. After the initialization of the CPE, it enters the execution loop, which repeats the process of updating the shared task counter and executing the task block corresponding to the current task index until all tasks are executed.

Since Sunway's current programming interface does not support queues or execution streams, the MPE must wait for all CPEs to finish after submitting a task and starting CPEs, thus preventing errors if the CPEs start the next task before they are idle, swYAKL's `parallel_for` interface is set to be a blocking interface by default, non-blocking mode is not supported at this time.

B. On-Chip Local Memory Allocation

In the Sunway architecture, each CPE has a special memory area called LDM that can be configured by software. Accessing this LDM is much faster for the CPEs compared to accessing the main memory. To make computations more efficient, developers should use the LDM for quicker memory access. Sunway architecture allows part of the LDM to function as a cache, which can automatically manage the CPE's access to the main memory. This cache can be adjusted by setting specific parameters when submitting tasks. However, accessing memory in a non-sequential (stride) pattern, which happens often in real applications, can be challenging for the cache to handle efficiently.

To solve this issue, swYAKL provides a simple interface for managing the LDM. One key feature in swYAKL is the `SArray`, a static array allocated on the stack. This array is more efficient for accessing and creating multidimensional data compared to the `yakl::Array`, which is allocated on the heap. The `SArray` is typically used for constants or frequently accessed data, usually containing small amounts of data.

In swYAKL, when an `SArray` is created within a compute kernel, it is directly allocated in the LDM stack of the CPE. Data can be moved into the `SArray` using specific instructions or DMA before starting the task. If an `SArray` is created on the MPE, the first time it is accessed within the kernel, space is allocated on the LDM heap, and the data is transferred there using DMA. Compared to automatic caching, using `SArray` in the user-allocatable region of LDM gives developers more control. This means they can manage the data needed for the kernel without worrying about it being replaced by the automatic cache mechanism.

C. Mixed Compilation For MPE & CPE

swYAKL uses many advanced C++ features like templates and namespaces. It needs a compiler that can compile C++ code for both the MPE and CPEs. To pass objects between different architectures and use lambda, swYAKL needs a compiler that supports mixed programming—embedding CPE code into MPE code. Previously, Sunway compilers only supported C++ for the MPE and required C programming for CPEs, necessitating separate coding for MPE and CPEs.

`swuc` is a special compiler made for the Sunway architecture that fully supports C++ for both MPE and CPEs. This makes it easier to transfer objects between different parts of the system and is essential for running swYAKL. `swuc` uses specific markers to indicate which parts of the code should run on the MPE or CPEs, allowing for a seamless mixed programming environment.

When working with C++ objects, the tool `swuc` creates two versions of each unmarked object: one for the MPE and one for the CPE. These two versions will have the same member variables and functions, but their names will be changed by the compiler. Then, these versions are compiled separately and linked together to create an object that works with both MPE and CPE.

For example, if you have lambda objects, `swuc` will create them for both MPE and CPE and then link them to make a lambda object that can run on both architectures. If a method only works on MPE or CPE, developers should use special macros (`__sw_host__` for MPE and `__sw_slave__` for CPE) to ensure the method is compiled correctly for the right architecture. Alternatively, they can specify which architecture to compile for using an attribute.

D. Data Movement in Parallel

The Sunway architecture is designed so that both the main processing unit (MPE) and the computing processing units (CPEs) share the same memory. This setup avoids the slow process of copying data between different devices, making it more efficient.

The CPEs can access memory much faster than the MPE. Normally, the MPE handles memory copying, but the swYAKL library uses the faster CPEs to do this in parallel, speeding up the process. swYAKL uses CPE's DMA (Direct Memory Access) to transfer data between the main memory and local memory of the CPEs. All the CPEs work together to read and write different parts of the data at the same time, making the process faster.

However, starting and stopping these parallel operations on Sunway takes about 300 nanoseconds. For small amounts of data, this overhead makes the parallel method slower than a simple serial copy. So, swYAKL sets a threshold: it only uses the parallel method for data larger than this threshold, which adjusts based on different execution modes to ensure the best performance.

TABLE I
THEORETICAL PERFORMANCE OF HARDWARE IN EXPERIMENT

Hardware	**#cores**(per socket)	**frequency**	**FLOPS**(theoretical)
NVIDIA A100-40GB	3456(FP64) 6912(FP32)	1410 MHz	9.7 TFLOPS
Intel Xeon Platinum 8358	32	2.60GHz	1.33 TFLOPS
AMD EPYC 7543	32	2.80GHz	2.87 TFLOPS
SW26010pro	6 MPE & 384 CPE	2.25GHz (CPE)	13 TFLOPS
SW26010pro (Core Group)	1 MPE & 64 CPE	2.25GHz (CPE)	2.17 TFLOPS

V. RESULT

In this section, this paper uses several computing kernels and 1 proxy application to test the acceleration capability and performance portability of swYAKL for Sunway architecture. All codes are written using the programming method recommended by YAKL and can run on all platforms supported by YAKL.

A. Experimental Setup

The Sunway experiments used a single node with an SW26010 Pro, compiling with swuc and O3 optimization with CPE auto vectorization enabled, in kernel benchmark we only use single core group and in proxy application benchmark we will use all core groups in single node. For mainstream platforms, tests involved two nodes: one with 2-way AMD EPYC 7543s and the other with 2-way Intel Xeon Platinum 8358s and a NVIDIA A100 GPU. For CPU target we use gcc 11.4 compiler with O3 compile flag and for GPU we use nvcc 11.4 with arch_70 and O3 flags. The theoretical performance of the platforms utilized in the experiments is shown in Table I.

This paper evaluates swYAKL's performance on the Sunway architecture using two different workloads. First workload are a series of kernels, includes 5 typical stencil templates and 3 stencil kernels from the PLUTO [19] project (head-2d, jacobi-2d, fdtd-2d) relevant to simulations in atmosphere and ocean. The second workload features practical applications, including a proxy application called miniWeather developed by original YAKL developers.

B. Experimental Results

1) Multiply-Add Kernel: The multiply-add kernel operates element-wise, multiplying corresponding values from two arrays and adding them to a third array. As depicted in Figure 4(e), a single core group on Sunway outperforms two-way AMD and Intel CPUs at large scales due to its high parallel efficiency. However, at smaller scales, the high overhead of kernel launches may result in slower performance compared to serial execution on Sunway. In this kernel, our framework is approximately 41 times faster than MPE in single core group mode.

The maximum performance speedup observed in this test was 41.65x compared to MPE's serial execution in single CG mode.

2) Stencil Kernels: The experiment selected five typical stencil template and three stencils from PLUTO [19], which were used to test the automatic parallelization capability of PLUTO in the original paper. This paper rewrites these kernels using `paralle_for` to verify the acceleration effect and performance portability of swYAKL on the Sunway platform. The experiment ran three kernels in N * N matrices, where N ranged from 64 to 4096.

a) Typical stencil templates: Typical stencil templates contains 5-point and 9-point 2D stencil as well as 7-point and 27-point 3D stencils. These stencil templates are common on many numerical simulation to represent different compute mode.

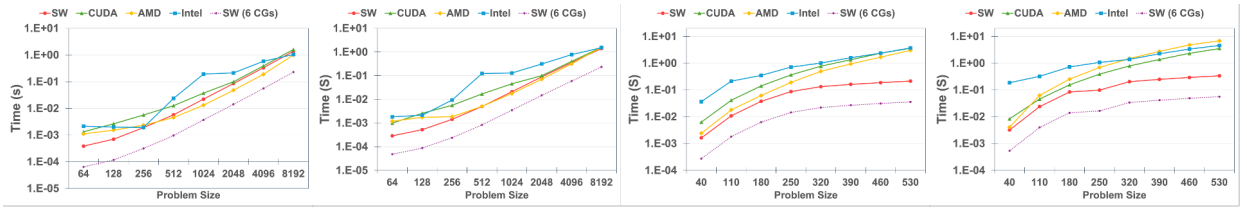
b) heat-2d: heat-2d has a linear continuous memory access with excellent memory access locality. It is the most cache-friendly stencil among the three stencils.

c) jacobi-2d: Which has forward and backward stride memory access for each position. SArray is used to cache the data through the LDM of the CPE in Sunway. When the stencil kernel starts, all the data involved in the task block are moved to the LDM for caching using DMA.

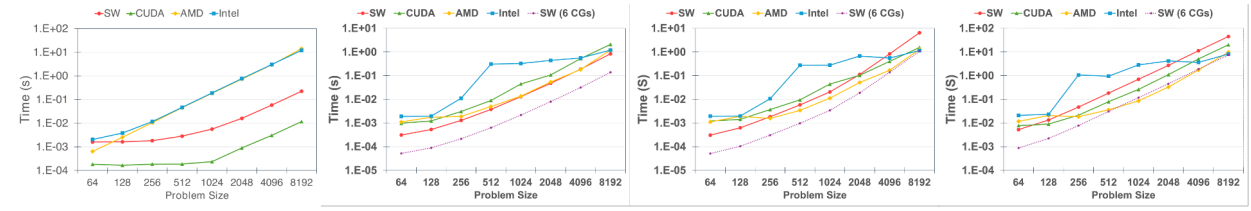
d) fdtd-2d: Which is a multi-stage computing kernel. This kernel have one initialize stage and two update stages. The original kernel uses four loops to express the three-stage computation. One loop is used for initialization, two loops are used for the first updating of the gradients in two directions, and then the last loop uses the gradients in two directions to update the value of the current position. The two loops in the first update stage do not have dependencies. The parallel implementation of this article uses three computing kernels to express this calculation process, fuses two loops in first update stage.

The performance of stencil computations varies significantly depending on the specific stencil configuration. For 2D kernels, the introduction of non-linear memory access patterns and an increase in computational density contribute to superior performance on the Sunway architecture compared to other CPU platforms, particularly at large scales. In the case of 3D stencils, the performance of kernels on Sunway surpasses that on GPUs due to a novel partitioning strategy that enhances computational efficiency across the 3D data space.

The performance evaluation depicted in Figures 4(a) and 4(b) indicates that the Sunway architecture achieves slightly lower computational efficiency than AMD and GPU platforms but surpasses Intel systems. This discrepancy largely stems from the kernel's limited computational intensity, which in-



(a) 5-point 2D Stencil (b) 9-point 2D Stencil (c) 7-point 3D Stencil (d) 27-point 3D Stencil



(e) Element-wise Multiply-Add Kernel (f) Heat-2d Kernel (g) Jacobi-2d Kernel (h) Fdtd-2d Kernel

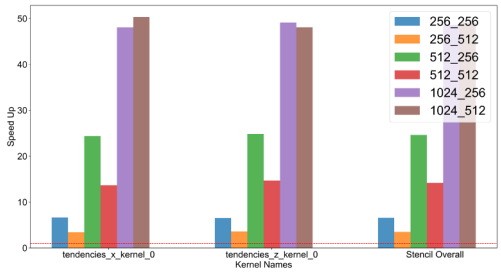
Fig. 4. Result of each stencil kernels on different platforms. (SW stand for Sunway single core group, CUDA stand for single NVIDIA A100, Intel stand for 2-way Xeon Platinum 8358 and AMD stand for 2-way EPYC 7543)

volves about five fused multiply-add operations per memory access for 5-point stencils and eight for 9-point stencils. Sunway's architectural limitations, such as reduced memory bandwidth, inefficient cache usage, and difficulties in utilizing non-blocking DMA for streaming operations, significantly impact these performance outcomes.

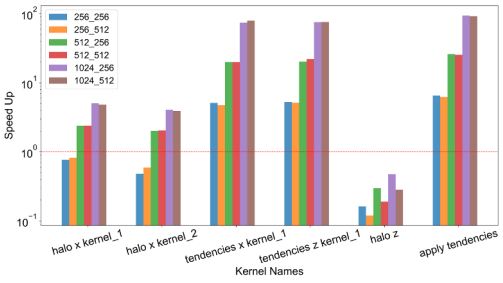
In contrast, three-dimensional kernel implementations on Sunway exhibit substantial performance advantages, as illustrated in Figures 4(c) and 4(d). Despite similar computational density to their two-dimensional counterparts, the memory access patterns of the 7-point 3D stencil enhance effective utilization of non-blocking DMA, resulting in significant performance improvements. Specifically, Sunway achieves approximately a 12-fold speedup for 7-point stencils and up to a 14-fold increase for 27-point stencils compared to other platforms.

For the kernels from the PLUTO benchmark suite shown in Figures 4(f), 4(g), and 4(h), Sunway exhibits competitive performance relative to other platforms. The Heat 2D and Jacobi 2D kernels, sharing computational patterns with the 5-point 2D stencil, display differences such as constant reuse within the kernel, potentially enhancing compiler optimization compared to previously discussed kernels. The FDTD 2D implementation, consisting of three stages with inter-stage data transfer and multiple `parallel_for` regions, presents distinct challenges. While mainstream architectures can use advanced data-flow optimizations for cache reuse between kernels, the Sunway architecture requires clearing the CPE cache and LDM space between kernel executions, impeding effective compiler optimizations and hardware-level prefetching.

These benchmarks demonstrate the swYAKL framework's efficacy in optimizing stencil computations on the Sunway architecture, despite certain conventional optimization techniques being constrained by the limitations of the CPE runtime design.



(a)



(b)

Fig. 5. Speedup rate comparing with MPE of different types of kernels: (a) Stencil kernels in miniWeather (b) Other kernels in miniWeather, including regular computation kernels and data movement kernel. (512_256 means problem size is set to 512, while scheduler block size is set to 256.)

C. Proxy Application: miniWeather

This paper uses a proxy application miniWeather that mirrors a real-world atmospheric simulation application as a case. The task scale depends on the number of grids in the x and z directions and the number of time steps. The scale mentioned later is number of grids in the x direction, denoted

as nx, and the number of grids in the z direction is 1/2 of the x direction. Measurement of per-kernel performance uses the YAKL built-in profile instrumentation to time each kernel separately. When testing the overall performance, the YAKL built-in timer is turned off and the C++ built-in chrono library is used for timing with lower overhead. Each test runs 1000s time step.

There are three types of kernel in miniWeather, they are data movement kernel, normal computation kernel and stencil kernel.

1) Data Movement Kernel: Data movement kernels are `halo_x_kernel_1`, `halo_x_kernel_2`, and `halo_z`. The first two kernels are used to pack or unpack data before and after the MPI operation is started, and the latter kernel is a hybrid kernel that performs polynomial calculations or data movement for different physical quantities.

As they mainly copy data from one array to another array, contain no computation operation, Sunway's CPE cannot accelerate it because it mainly use load and store instruction of CPE, which have high latency and low bandwidth.

2) Computation Kernel: The computation kernels include three kernels. The `apply_tendencies_kernel` involves a single multiply-add operation, which the compiler can optimize into a single multiply-add instruction for the CPE (Computing Processing Element). Consequently, the computational density of this kernel is relatively low, with each multiply-add instruction necessitating two read operations and one write operation. The other two kernels, `tendencies_x_kernel_1` and `tendencies_z_kernel_1`, incorporate division calculations. However, due to the absence of a hardware division component in the Sunway architecture, these divisions must be simulated through software, resulting in lower efficiency compared to the `apply_tendencies_kernel`.

Despite these kernels having a lower computational density than the stencil kernels, they possess a 3D index space. This characteristic allows them to benefit significantly from the new partition strategy introduced in the previous section, enabling notable acceleration.

3) Stencil Kernel: The stencil kernel encompasses other kernels and represents the segment with the highest computational density within the entire program. These specific stencils exhibit a memory access pattern analogous to that of the Jacobi-2D kernel.

These kernels are implemented using SIMD style, and they store certain high-frequency data in LDM to enhance memory access efficiency. The actual computational density of these kernels is significantly greater compared to the previously mentioned kernels, involving numerous computational operations, alongside five read operations and one write operation, thus achieving substantial speedup. The swYAKL framework can accelerate the kernel by over 50 times compared to the version utilizing only the MPE (Main Processing Element), as illustrated in Figure 5(a).

4) Overall Performance: In terms of overall performance, compared with the same number of MPEs, using swYAKL with six processes can bring about 70.4 times performance improvement to miniWeather. Different block sizes show different performance, as shown in Figure V-C4, with growth of problem size, the influence of different block size become smaller.

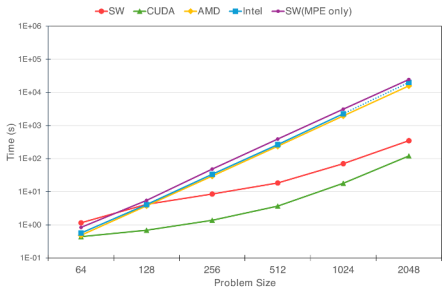


Fig. 6. Overall time consumption of miniWeather on different platforms. (SW stand for Sunway **single node**, other are same as Figure V-A

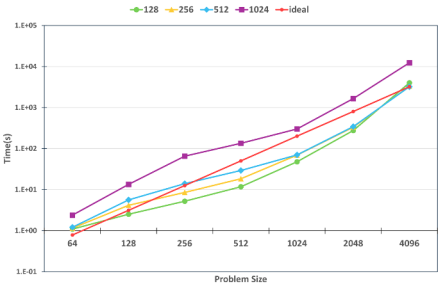


Fig. 7. Overall time consumption of miniWeather with different granularities. Ideal means ideal scalability of this application with the same overhead on the problem size of 4096.

According to the per-kernel performance information in Figure 5(a) and Figure 5(b), it can be seen that the overall performance improvement mainly comes from the performance improvement of stencil kernels. As the application scale expands, other kernels with less computing density than stencil kernels can also get some improvement on performance, and the addition of parallel memcpy will also greatly improve the performance of deep copy operations in large-scale applications, making the scalability of applications gradually improve. Compared with 2-way Xeon 8358 and 2-way EPYC 7543 of the same scale, the execution speed can be improved up to 32.2x and 27.8x, and about only 3.8x slower than our A100 GPU platform.

This proxy application shows that swYAKL makes the performance portable code can be reused directly on Sunway platform. Therefore, it can be concluded that swYAKL can efficiently accelerate applications under the Sunway architecture and bring the performance portability to this architecture.

VI. Conclusion & Discussion

This research demonstrates that swYAKL significantly improves performance portability on the Sunway manycore architecture by incorporating a dynamic task scheduler, an

innovative task partitioner, and several platform-specific optimizations. Performance evaluations reveal that swYAKL can enhance computing kernel performance by factors ranging from 5x to over 100x compared to the MPE-only version, potentially even surpassing GPU performance in some workloads. In proxy application tests, swYAKL achieves an overall performance increase of approximately 70.4x, with scalability improving with larger test sizes.

However, swYAKL faces limitations such as considerable kernel startup overhead and NUMA-related challenges during memory access, potentially resulting in reduced performance compared to serial execution for kernels with low compute density or small scale. Specifically, Sunway's CPE lacks an efficient automatically managed multi-level memory mechanism to mitigate NUMA issues common on modern platforms. Manual management is restricted by platform design constraints, such as the need for LDM access alignment to 4 bytes, and the limitations of the programming library's ability to gather workload information at compile-time without specific compilation techniques. Moreover, the shared memory architecture can lead to unnecessary deep copies, causing memory inefficiency and diminished execution performance. While parallel memory copying might offer some relief, it does not fully resolve the issue. As a programming library, swYAKL cannot inherently detect application workloads, requiring the use of compilation techniques to enable workload-aware optimization and generate efficient target code.

In conclusion, this study enhances the computational infrastructure of China's Sunway supercomputing platform by optimizing cross-platform performance and streamlining application development for the Sunway architecture. Although primarily focused on the Sunway ecosystem, the architectural principles and methodological frameworks developed here have potential applicability to similar manycore processing systems. These contributions may foster performance consistency and code portability across emerging heterogeneous computing environments characterized by manycore configurations.

VII. Knowledge

This work is partly supported by the Strategic Priority Research Program of Chinese Academy of Sciences (XDB0500102) and the National Natural Science Foundation of China (Grant No. 62102389). The computing resources are financially supported by Laoshan Laboratory (LSKJ202300305). We appreciate the reviewers for dedicating their time and providing constructive comments on this paper.

References

[1] T. Deakin, S. McIntosh-Smith, J. Price, A. Poenaru, P. Atkinson, C. Popa, and J. Salmon, "Performance Portability across Diverse Computer Architectures," in *2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, 2019, pp. 1–13. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8945642

[2] C. R. Trott, D. Lebrun-Grandie, Daniel Arndt, D. Arndt, Jan Ciesko, J. Ciesko, Vinh Q. Dang, V. Dang, N. D. Ellingwood, Rahulkumar Gayatri, R. Gayatri, Rahulkumar Gayatri, E. Harvey, D. S. Hollman, D. Ibanez, D. Ibanez, N. Liber, J. R. Madsen, Jonathan Madsen, J. S. Miles, D. Poliakoff, A. J. Powell, Amy Jo Powell, S. Rajamanickam, M. Simberg, D. Sunderland, D. Sunderland, B. Turcksin, and J. J. Wilke, "Kokkos 3: Programming model extensions for the exascale era," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 4, pp. 805–817, 2021.

[3] D. A. Beckingsale, T. R. Scogland, J. Burmark, R. Hornung, H. Jones, W. Killian, A. J. Kunen, O. Pearce, P. Robinson, and B. S. Ryujin, "RAJA: Portable Performance for Large-Scale Scientific Applications," in *2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*. IEEE, 2019, pp. 71–81. [Online]. Available: https://ieeexplore.ieee.org/document/8945721/

[4] M. R. Norman, Isaac Lyngaas, Isaac Lyngaas, Abhishek Bagusetty, and Mark Berrill, "Portable C++ Code that can Look and Feel Like Fortran Code with Yet Another Kernel Launcher (YAKL)," *International Journal of Parallel Programming*, 2022.

[5] S. Erich, D. Jack, S. Horst, and M. Martin. (2023) TOP500 List - November 2023 — TOP500. [Online]. Available: https://www.top500.org/lists/top500/list/2023/11/

[6] A. Rovinski, C. Zhao, K. Al-Hawaj, P. Gao, S. Xie, C. Torng, S. Davidson, A. Amarnath, L. Vega, B. Veluri, A. Rao, T. Ajayi, J. Puscar, S. Dai, R. Zhao, D. Richmond, Z. Zhang, I. Galton, C. Batten, M. B. Taylor, and R. G. Dreslinski, "A 1.4 ghz 695 giga risc-v inst/s 496-core manycore processor with mesh on-chip network and an all-digital synthesized PLL in 16nm CMOS," in *2019 Symposium on VLSI Circuits*, 2019, pp. C30–C31.

[7] (2024) Cerebras system. [Online]. Available: https://www.cerebras.net/product-system/

[8] Y. Ye, Z. Song, S. Zhou, Y. Liu, Q. Shu, B. Wang, W. Liu, F. Qiao, and L. Wang, "swNEMO_v4.0: An ocean model based on NEMO4 for the new-generation Sunway supercomputer," *Geoscientific Model Development*, vol. 15, no. 14, pp. 5739–5756, 2022. [Online]. Available: https://gmd.copernicus.org/articles/15/5739/2022/

[9] (2024) CUDA Toolkit Documentation 12.4. [Online]. Available: https://docs.nvidia.com/cuda/

[10] (2024) ROCm/HIP: HIP: C++ Heterogeneous-Compute Interface for Portability. AMD. [Online]. Available: https://github.com/ROCm/HIP

[11] (2013) OpenCL - The Open Standard for Parallel Programming of Heterogeneous Systems. The Khronos Group. [Online]. Available: https://www.khronos.org//

[12] (2024) Openacc homepage. [Online]. Available: https://www.openacc.org/

[13] O. A. R. Board. (2021) Openmp application programming interface 5.2. OpenMP. [Online]. Available: https://www.openmp.org/

[14] S. Sreepathi and M. Taylor, "Early evaluation of fugaku A64FX architecture using climate workloads," in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, 2019, pp. 719–727.

[15] (2024) oneapi threading building blocks (onetbb). Intel. [Online]. Available: https://github.com/oneapi-src/oneTBB

[16] M. Wang, T. Ta, L. Cheng, and C. Batten, "Efficiently Supporting Dynamic Task Parallelism on Heterogeneous Cache-Coherent Systems," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 173–186. [Online]. Available: https://ieeexplore.ieee.org/document/9138949/

[17] L. Cheng, M. Ruttenberg, D. C. Jung, D. Richmond, M. Taylor, M. Oskin, and C. Batten, "Beyond Static Parallel Loops: Supporting Dynamic Task Parallelism on Manycore Architectures with Software-Managed Scratchpad Memories," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. ACM, 2023, pp. 46–58. [Online]. Available: https://dl.acm.org/doi/10.1145/3582016.3582020

[18] W. Wu, H. Qian, Q. Zhu, J. Wang, and X. Fan, "Research on Full-Chip Programming for Sunway Heterogeneous Many-core Processor," in *2021 The 3rd World Symposium on Software Engineering*. ACM, 2021, pp. 174–179. [Online]. Available: https://dl.acm.org/doi/10.1145/3488838.3488868

[19] U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan, "A practical automatic polyhedral parallelizer and locality optimizer," in *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 2008, pp. 101–113. [Online]. Available: https://dl.acm.org/doi/10.1145/1375581.1375595