SN Computer Science (2026) 7:24
https://doi.org/10.1007/s42979-025-04575-0

ORIGINAL RESEARCH

High-Performance Mixed-Precision Matrix Multiplication via Tile-Centric Design on Modern Architectures

Qiao Zhang[1] · Rabab Alomairy[2,3] · Dali Wang[4] · Zhuowei Gu[1] · Qinglei Cao[1]

Received: 18 June 2025 / Accepted: 25 November 2025

Abstract
General Matrix Multiplication (GEMM) is a critical operation underpinning a wide range of applications in high-performance computing (HPC) and artificial intelligence (AI). The emergence of hardware optimized for low-precision arithmetic necessitates a reevaluation of numerical algorithms to leverage mixed-precision computations, achieving improved performance and energy efficiency. This research presents an adaptive mixed-precision GEMM framework that enables support for various precision formats at fine-grained tile and block levels, offering a reliable foundation for trustworthy mixed-precision computations. Furthermore, we leverage the PaRSEC runtime system to effectively balance workloads across diverse architectures. The performance exhibits strong scalability across both homogeneous platforms (Intel CPU-based systems and the ARM CPU-based `Fugaku` supercomputer) and heterogeneous systems (Nvidia V100, A100, and H100 GPU-based platforms, as well as the AMD GPU-based `Frontier` supercomputer). This work aims to improve computational efficiency and accuracy by bridging algorithmic innovations with hardware capabilities, fostering transformative advancements across a wide range of applications.

Keywords High-performance computing · Task-based runtime · General matrix multiply · Mixed precision

Introduction

Over the last ten years, hardware designers have prioritized crafting processors that excel in high-speed, energy-efficient, and reduced-precision computations [1]. A notable example is the widespread use of Nvidia GPUs in both High-Performance Computing (HPC) and Artificial Intelligence (AI) tasks, driven by their exceptional processing capabilities and efficiency. The latest Nvidia GPUs, including the H100 and the B100, incorporate a range of precision formats, each influencing peak theoretical throughput differently. The lowest precision levels (FP8/INT8 on H100 and FP4 on B100) achieve speedups of 58× and 233×, respectively, relative to FP64 computations, underscoring the dramatic acceleration possible when shifting from conventional double-precision arithmetic to Tensor Core-accelerated lower-precision execution. A similar evolution is observed in CPUs, FP32 peak throughput is typically about twice that of FP64, and FP16 about four times that of FP64. This transition has motivated researchers to reassess classical numerical techniques, identifying areas where reduced precision is sufficient without compromising the overall quality of solutions. Additionally, leveraging lower precision not only diminishes memory demands–allowing more data to fit within high-speed memory and thereby improving execution rates [2, 3]–but also cuts down on energy consumption, making it a compelling strategy for boosting energy efficiency in computational workflows [4].

✉ Qiao Zhang
qiao.zhang@slu.edu

✉ Qinglei Cao
qinglei.cao@slu.edu

Rabab Alomairy
rabab.alomairy@mit.edu

Dali Wang
dali.wang@ornl.gov

Zhuowei Gu
zhuowei.gu@slu.edu

1 Saint Louis University, Saint Louis, USA

2 Massachusetts Institute of Technology, Cambridge, USA

3 King Abdullah University of Science and Technology, Thuwal, Kingdom of Saudi Arabia

4 Oak Ridge National Laboratory, Oak Ridge, USA

Published online: 20 December 2025

The synergy between hardware advancements and algorithmic refinements plays a pivotal role in tackling complex computational challenges. Mixed-precision computations, which involve utilizing multiple numerical precisions within a single task, have emerged as a powerful strategy [5, 6]. This methodology has been successfully integrated into numerous scientific and engineering disciplines, such as deep learning [7], computational fluid dynamics [8, 9], astronomy [10], climate modeling [11], molecular dynamics [12], and quantum mechanics [13]. The primary advantage of these techniques is their ability to harness lower-precision hardware capabilities to enhance computational speed while preserving necessary accuracy constraints.

One common approach, mixed-precision iterative refinement [14–16], has been proposed to speed up linear solvers. The initial solution is computed in lower precision, while the residual is refined iteratively in higher precision to ensure convergence. This approach has demonstrated a substantial reduction in both time-to-solution and energy consumption, making it an appealing strategy for improving energy efficiency in computing operations [4, 17–19]. Nevertheless, besides the additional memory requirements, a key drawback of iterative refinement is its sensitivity to the conditioning of the system and the accuracy of the low-precision solve, which can lead to slow convergence or even failure if the underlying numerical properties are unfavorable [18–21].

Another popular strategy employs direct mixed-precision computation with adaptive control [3, 22], where not every part of the computation demands the same precision level. Therefore, precision is selectively reduced at the tile or block level based on runtime behavior or data characteristics: by carefully applying lower precision where it has minimal impact on the result, and reserving higher precision for critical calculations, it is possible to accelerate computations with reduced memory usage while maintaining accuracy and reliability. This solution has demonstrated its effectiveness in many applications, such as geostatistical modeling and prediction [2, 3, 22], climate emulation [23], and genomics [24]. In scenarios where user productivity is potentially at stake, one might opt for coding simplicity; however, this adaptive solution at the tile/block level could be a potential game-changer, providing the opportunity for a trustworthy mixed-precision solution for scientific and AI workloads.

Current research on this adaptive approach mainly targets Cholesky factorization in different applications [3, 22–24]. This paper extends it to General Matrix Multiplication (GEMM) and introduces GEMM-MP, an adaptive tile-centric mixed-precision GEMM framework. GEMM lies at the core of linear algebra and acts as a critical kernel in numerous HPC applications—such as earthquake simulation [25] and weather and climate prediction [23]—and in AI applications, including fully connected layers in traditional neural networks and natural language processing [26]. GEMM is a computationally intensive operation with high demands on both memory (quadratic) and computation (cubic), which can also benefit from this adaptive solution. For instance, in long-context Transformers for tasks such as document summarization, multi-turn dialogue, and retrieval-augmented generation, where the complexity of attention (mainly from GEMM) imposes prohibitive memory and compute costs. Existing implementations rely on coarse-grained precision adjustments, where the entire matrix is uniformly assigned to a reduced precision (e.g., FP16 [27], INT8 [27, 28], or even INT4 [29] in Quantization), leading to unnecessary accuracy loss. By adaptively assigning lower precisions to numerically stable regions of the attention matrix while preserving higher precisions in sensitive areas, this adaptive approach at the fine-grained level has the potential to overcome (1) the loss of long-range dependencies and retraining overhead inherent in block sparse attention, and (2) the instability and unnecessary accuracy degradation characteristic of traditional uniform mixed-precision schemes.

On the other hand, with the increasing complexity and heterogeneity of modern computing architectures, ranging from multi-core CPUs to GPUs from diverse vendors such as NVIDIA, AMD, and Intel, traditional parallel programming models often struggle to efficiently utilize available resources. This challenge has catalyzed the development of task-based runtime systems, which are rapidly gaining traction among researchers and developers [30–35]. These systems decompose applications into fine-grained, independent tasks that can execute concurrently, providing a flexible and adaptive mechanism for managing parallelism across heterogeneous platforms. By dynamically scheduling tasks to the most suitable processing units, task-based runtimes enhance resource utilization and exploit the full potential of hardware concurrency. This approach leads to significant improvements in performance, energy efficiency, and scalability across a broad range of scientific and engineering applications [22, 36–41]. Therefore, we adopt the task-based runtime system PaRSEC to support the execution of GEMM-MP on both homogeneous and heterogeneous architectures, orchestrating tasks with diverse computational characteristics and precision levels to achieve high performance and portability on modern GPU platforms at scale.

The contributions of this paper are as follows:

- Introducing a tile-centric mixed-precision GEMM algorithm that carefully balances accuracy and performance, enabling a robust and trustworthy mixed-precision solution for HPC and AI applications.

- Leveraging a dynamic task-based runtime system to efficiently orchestrate heterogeneous tasks, dynamically adapting to workload variations while minimizing scheduling overhead and maximizing resource utilization.
- Demonstrating strong scalability and portability across a diverse range of homogeneous and heterogeneous computing platforms, ensuring broad applicability and efficiency.

To the best of our knowledge, this is the first work to introduce a tile-centric mixed-precision GEMM algorithm.[1]

The structure of this paper is as follows. Section "Related Work" discusses prior research relevant to our work. In Section "The PaRSEC Runtime System", we provide an overview of PaRSEC. The tile-centric mixed-precision GEMM framework is described in Section "Algorithm Descriptions". After that, Section "Implementation Details" depicts in detail how we implement GEMM-MP in PaRSEC. A performance evaluation is then conducted in Section "Performance Results and Analysis", and finally, Section "Conclusion and Future Work" outlines our conclusions and future directions.

Related Work

Mixed-Precision Matrix Computations

Mixed-precision approaches have garnered extensive interest [5, 6] and are widely applied across multiple computational fields [7–13]. A well-known example is iterative refinement techniques for linear solvers, which begins by solving a linear system in low precision to quickly obtain an approximate solution. The algorithm then computes the residual in high precision to measure the error, and uses the low-precision solver to correct the approximation based on this high-precision residual. By alternating between low-precision solves and high-precision corrections, the method leverages the speed and energy advantages of low-precision arithmetic while relying on high-precision steps to maintain accuracy. Mixed precision reduces overall computational cost and memory usage without sacrificing solution quality, making it particularly effective for large-scale, performance-critical applications. This strategy has demonstrated significant reductions in computational time and energy consumption, making it an effective approach for enhancing efficiency in large-scale computations [4, 19]. However, its practical effectiveness is sensitive to problem conditioning and the availability of effective preconditioning. As conditioning worsens, the accuracy required from the low-precision stage may exceed its capability, and the requirement to store multiple precision copies of matrices can introduce substantial memory overhead.

Ongoing research explores performance optimization while ensuring numerical reliability by selectively employing reduced precision in non-critical computations while preserving higher precision where necessary [22–24]. These methodologies primarily target linear system solutions [43]. In modern neural network training, it has become common to perform the bulk of matrix multiplications and convolution operations in half or bfloat16 precision, since these formats run faster on specialized hardware and cut memory bandwidth. However, the tiny gradient updates and weight accumulations can quickly underflow or round to zero if kept in the same low-precision format, so optimizers typically switch to single or double precision to accumulate parameter updates [44]. This mixed-precision regime strikes a balance: most of the heavy lifting happens in low precision for throughput and cache efficiency, while the critical accumulation of small changes uses higher precision to preserve small-magnitude information. Despite this, the coarse granularity of reduced-precision arithmetic in the forward and backward passes can still introduce quantization noise that subtly distorts training dynamics, slows convergence, or degrades final model accuracy–especially on very deep or highly sensitive architectures. Techniques such as dynamic loss scaling, stochastic rounding, and block-wise precision tuning are therefore essential to mitigate these effects and ensure stable, high-quality training at scale [45]. In this work, we introduce an adaptive tile-centric mixed-precision GEMM framework that dynamically adjusts precision levels within a single computation. This fine-grained approach can provide another solution to ensure that accuracy remains uncompromised by recognizing that not all computational components require the same precision level in real applications.

Runtime Systems

Recent advancements in task-based programming models have driven the scalability of scientific applications on modern supercomputers. A variety of runtime systems have been developed to facilitate efficient task execution, optimizing resource utilization across diverse architectures.

OpenMP [32] offers a directive-based interface for exploiting shared-memory parallelism in C, C++ and Fortran codes on multicore and many-core systems. Programmers annotate loops or code regions to spawn threads, and the OpenMP runtime orchestrates scheduling, load balancing, and synchronization. Since the introduction of its task construct in OpenMP 3.0 and the addition of explicit task

[1] This article is an extended version of [42].

dependencies via the depend clause in OpenMP 4.0, developers can mark arbitrary work units for dynamic execution, allowing the runtime to resolve dependencies and distribute tasks across cores without manual thread management.

Building upon OpenMP, OmpSs [31], from the Barcelona Supercomputing Center, enriches OpenMP's model by letting users declare task data inputs and outputs with simple annotations. The runtime builds a dependency graph at compile or launch time and dispatches tasks to CPUs, GPUs or other accelerators as soon as their data are available. By automatically handling data movement and synchronization, OmpSs transparently exploits fine-grained parallelism on heterogeneous HPC nodes.

COMPSs [46] (COMP Superscalar) transforms ordinary sequential programs into distributed workflows by wrapping methods in Java, Python or C/C++ with task annotations. During execution, the runtime inspects these annotations to generate a global task graph, then automatically distributes computation and data across cluster or cloud resources. This approach frees developers from writing explicit MPI or data-transfer code while ensuring scalable, fault-tolerant parallelism.

StarPU [30] defines each computation as a "codelet" with multiple implementations (e.g., CPU, CUDA) and metadata describing data accesses. Its scheduler uses performance models to select the best processing unit for each task and overlaps transfers between host and device memories. On clusters, StarPU can also inject MPI calls based on the task graph, providing a unified runtime for heterogeneous, distributed environments.

HPX [33] implements the ParalleX model in modern C++, exposing familiar abstractions like futures, asynchronous function calls, and parallel algorithms that span both local and remote resources. It manages millions of lightweight threads, uses message-driven execution to hide communication latency, and relies on dataflow constructs for synchronization instead of global barriers. HPX's global address space and runtime performance counters enable adaptive, latency-tolerant scaling across large HPC systems.

Legion [34] focuses on data-centric parallelism by having programmers partition data into logical regions and annotate tasks with explicit read/write privileges. The runtime infers dependencies from these declarations, constructs a fine-grained task Directed Acyclic Graph (DAG), and orchestrates data movement across CPUs, GPUs, and network nodes. A customizable mapping interface separates the concerns of correctness from placement decisions, allowing autotuners or developers to flexibly optimize task and data placement for portable, high-performance execution. This work utilizes PaRSEC, which is described in the following section.

The PaRSEC Runtime System

PaRSEC [47, 48] serves as a generic framework designed for scheduling and managing micro-tasks on distributed, multi-core, and heterogeneous architectures. It integrates a runtime environment, multiple programming models for defining task graphs–applied in dense linear algebra computations [49] and irregular workloads such as mixed-precision techniques [3], low-rank factorizations [50], and sparse operations [40]–along with auxiliary libraries that ease the transition from legacy implementations. Additionally, it incorporates performance analysis utilities [38] to assist in tuning applications for large-scale heterogeneous platforms. Like other task-based runtime systems, PaRSEC represents computations as DAGs of tasks, where each node corresponds to a unit of work–typically operating on tile or block partitions of data–and directed edges indicate data dependencies.

PaRSEC provides users with a collection of Domain-Specific Languages (DSLs), such as the Parameterized Task Graph (PTG) [51], Template Task Graph (TTG) [52], and Dynamic Task Discovery (DTD) [53], to enhance flexibility in algorithmic expression. Among these, PTG, which is used in this paper, encodes the entire DAG in a compact, parameterized format, enabling optimizations such as collective communication reuse and separation of data distribution from execution. This abstraction aligns with the core methodology of our research on adaptive mixed-precision computations, and we employ PTG in this work. PaRSEC's portability extends across heterogeneous architectures. Task kernels written for CPUs can be retargeted to CUDA streams for NVIDIA GPUs or HIP queues for AMD GPUs by simply changing a compile-time backend parameter; no modifications to the PTG definitions or dependency annotations are needed. This zero-touch portability, together with a uniform programming model, ensures that the same application code can be deployed unaltered on multicore CPUs, GPU-accelerated clusters, FPGA-equipped nodes, or future accelerator technologies–dramatically reducing development effort, simplifying maintenance, and future-proofing scientific software against evolving hardware landscapes.

Beyond its DSLs, PaRSEC's runtime features a dataflow-driven scheduling engine that issues tasks as soon as their dependencies are met, exploiting fine-grained parallelism while overlapping communication and computation [53]. It supports GPU offloading, automatically orchestrating data movement between host and device memories to maximize throughput. The framework has demonstrated strong scalability up to hundreds of thousands of cores on leadership-class supercomputers and integrates seamlessly with MPI for inter-node coordination [22–24]. PaRSEC provides a robust, high-performance foundation for expressing and

executing task-based applications at extreme scale. Its domain-specific abstractions, lightweight dataflow runtime, and seamless heterogeneous portability make it a compelling choice for both traditional dense linear algebra and emerging irregular workloads [54]. Its modular design and open-source implementation under the ECP umbrella have fostered a growing community of users and contributors, making PaRSEC a robust foundation for developing next-generation HPC applications.

Algorithm Descriptions

The General Matrix-Matrix Multiplication (GEMM) is a fundamental operation in high-performance computing, deep learning, and scientific computing, which computes the product of two matrices, optionally scaled and accumulated with a third matrix, and is defined as: $C \leftarrow \alpha AB + \beta C$, where A and B are input matrices, C is the output matrix, and α, β are scalar coefficients. This paper adopts the SUMMA (Scalable Universal Matrix Multiplication Algorithm) approach [55], which is a high-performance parallel matrix multiplication algorithm designed for distributed-memory systems, particularly optimized to address communication bottlenecks in large-scale computations. SUMMA follows a row-column computation pattern, whereas Outer-Product GEMM utilizes Rank-1 updates (column vector × row vector accumulation) to optimize performance. This approach enhances data reuse, parallelism, and memory efficiency, making it well-suited for GPU Tensor Cores. Its flexibility to handle matrices of any size and integration into tools like ScaLAPACK [56] make it vital for large-scale math problems in supercomputing and machine learning, where fast distributed operations are crucial.

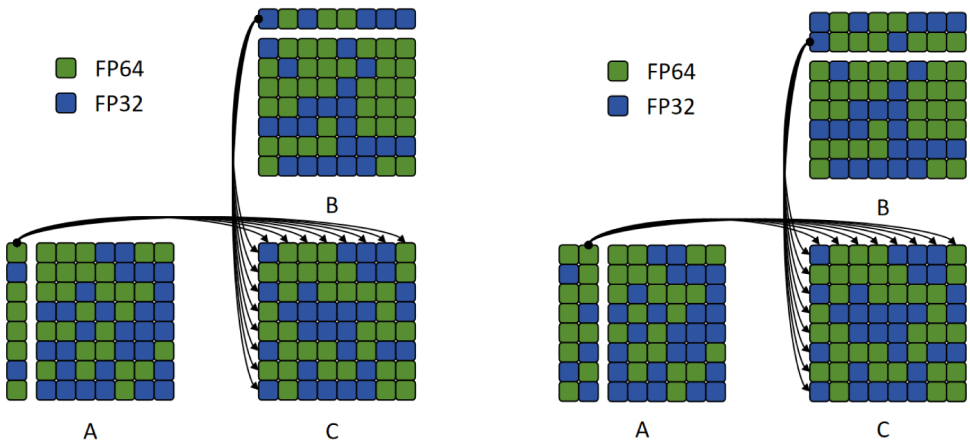
In SUMMA, the dataflow involves iteratively computing rank-k updates to the output matrix C by decomposing input matrices A and B into column and row panels, respectively. For each iteration, a column block $A[:, k]$ and a corresponding row block $B[k, :]$ are loaded into fast memory. Their outer product $A[:, k] \times B[k, :]$ generates a partial matrix, which is accumulated into the target block of C. This approach prioritizes data reuse: the same A and B panels remain resident in high-speed storage across multiple arithmetic operations, minimizing memory bandwidth pressure. The process repeats over all k partitions, with C being incrementally updated until the full matrix product is completed. This dataflow pattern aligns well with tiled architectures and parallel implementations, as it naturally exposes coarse-grained parallelism and reduces synchronization overhead by decoupling block-wise computations.

Algorithm 1 describes the sequential process of the adaptive mixed-precision GEMM following SUMMA approach, where the input matrices $A(i,l)_{m\times k}$, $B(l,j)_{k\times n}$, and the output matrix $C(i,j)_{m\times n}$ are partitioned into tiles/blocks of a fixed size. The outermost loop traverses the reduction dimension k, while the two inner loops iterate over the row index i and the column index j. At each iteration, the algorithm updates the value of the matrix C. The notations $\#$, \$, and $*$ indicate different data representations or precisions for the operands, where the precisions are handled at the fine-grained tile/block level.

1: **for** $l = 0$ **to** k **do**
2: **for** $i = 0$ **to** m **do**
3: **for** $j = 0$ **to** n **do**
4: $C^*(i,j) \leftarrow \alpha\, A^{\#}(i,l) \times B^{\$}(l,j) + \beta\, C^*(i,j)$
5: **end for**
6: **end for**
7: **end for**

Algorithm 1 Tile-centric Mixed-Precision GEMM.

Figure 1 illustrates execution across the first two iterations as defined in Algorithm 1 mixing double precision (FP64 or DP) and single precision (FP32 or SP). The visualization



(a) Iteration $k = 0$. (b) Iteration $k = 1$.

Fig. 1 Demonstration of the first two iterations of the mixed-precision (FP64 and FP32) GEMM. Arrows are representative dependencies that introduce communications

employs color coding to distinguish numerical precisions: green for FP64 and blue for FP32. This study focuses on FP64 and FP32, with plans to extend to additional precision formats (e.g., FP8, FP4, etc.) in future work. Arrows denote representative dependencies arising from data transfer and communication between matrix tiles ($A \rightarrow C$ and $B \rightarrow C$) within a single iteration. Each dependency/communication is associated with a specific datatype. Due to this adaptive tile-centric precision approach, a task may receive data in a precision differing from its operational precision, necessitating potential precision conversion. This conversion is guided by the decision/preicision maps of A, B, and C. In the current strategy, the computational precision is governed by the decision map of C.

Various conversion strategies will be explored in subsequent research. This work employs the receiver-side conversion strategy, meaning that datatype conversion is handled at the receiving end. That is, the datatype in each data flow is dictated by the precision of the stored data. For example, if a tile in A or B is stored in FP64, the transmitted data remains in double precision. Therefore, if a tile of C is stored in FP64, the corresponding tiles of A and B participating in the GEMM operation must first be converted to FP64; similarly, if a tile of C is in FP32, the respective tiles of A and B should be converted to FP32 in advance. For example, in Fig. 1a, during the first iteration at the first row and first column, A transmits a tile in FP64 precision, and B transmits a tile in FP32 precision. However, the corresponding tile of C is in FP32 precision, so A's tile must first be converted to FP32 before performing matrix multiplication. The primary objective of this strategy is to ensure consistency in the data type of each tile within matrix C throughout the computation.

Implementation Details

We implement this adaptive mixed-precision GEMM in Algorithm 1 and leverage the PaRSEC task-based runtime system for task scheduling and efficient parallel execution. The computation follows a dataflow-driven model, ensuring precision consistency across tiles while optimizing data reuse and reducing synchronization overhead in large-scale distributed environments.

Table 1 PaRSEC 's asynchronous, PTG-driven engine for mixed-precision GEMM.

```
// Task class for GEMM
GEMM(m, n, k)

// Execution space
m = 0 .. descC->mt-1
n = 0 .. descC->nt-1
k = 0 .. descA->nt-1

// Parallel partitioning
: descC(m, n)

// Data flow
READ A <-  A READ_A(m, k)                 [ type_remote = decisions_A(m, k) ]
READ B <-  B READ_B(k, n)                 [ type_remote = decisions_B(k, n) ]
RW   C <- (k == 0)             ? descC(m, n)
       <- (k != 0)             ? C GEMM( m, n, k-1 )
       -> (k == (descA->mt-1)) ? descC(m, n)
       -> (k != (descA->mt-1)) ? C GEMM( m, n, k+1 )

BODY [ type = CUDA / HIP / CPU ] {   // List body type on different devices
    // Computations
}

// Task class for reading A, ensuring proper data retrieval; same for READ_B
READ_A(m, k)

m = 0 .. descA->mt-1  // mt: number of tiles in the row dimension
k = 0 .. descA->nt-1  // mt: number of tiles in the column dimension

: descA(m, k)

READ A <- descA(m, k)
       -> A GEMM(m, 0 .. descC->nt-1, k)  [ type_remote = decisions_A(m, k) ]

BODY [ type = CPU ]{
    // Nothing
}
```

The snippet in Table 1 shows a PTG-based PaRSEC implementation of the tile-centric mixed-precision GEMM in Algorithm 1, which (with the analogous READ_B declaration) defines the three task classes (a class of tasks exhibiting similar patterns in both computation and communication)–GEMM, READ_A, and READ_B–where READ_A and READ_B stage the required tiles from A and B, respectively, and GEMM performs the mixed-precision multiply-accumulate. Lines 5–7 declare task class GEMM's execution space: m indexes the row tiles of A and C, n the column tiles of B and C, and k the outer-product step over A's column tiles and B's row tiles. Line 10 indicates how the data/matrix is partitioned, thereby defining the task affinity. Lines 13–18 specify the data-flow annotations. Tasks READ_A(m,k) and READ_B(k,n) (lines 13–14) fetch data A and B into task GEMM(m, n, k) with precisions drawn from pre-provided decision maps, i.e., decisions_A and decisions_B, respectively. C's dependency (lines 15–18) enforces strict k-ordered initialization, accumulation, and final write-back of C's tile. Because line 10 binds GEMM to the local descriptor for C, no remote-transfer tag is needed for C. Lines 20–22 implement the actual computations in the BODY block. The same task definitions can target multiple different architectures, e.g., NVIDIA GPUs, AMD GPUs, and CPUs in this paper; therefore, supporting asynchronous scheduling and fine-grained parallelism across many-core heterogeneous platforms.

Figure 2 illustrates the corresponding DAG for a simple GEMM, where the matrices A, B, and C are each partitioned into 2×2 tiles/blocks. The task affinity of `READ_A`(m, k), `READ_B`(k, n), and `GEMM`(m, n, k) aligns with the data blocks $A(m, k)$, $B(k, n)$, and $C(m, n)$, respectively. Accordingly, the `READ_A`(m, k) and `READ_B`(k, n) tasks load the corresponding data and forward it to the appropriate receiver task(s). Each `GEMM`(m, n, k) task performs the local computation and generates partial results. For example, the `GEMM`(m, n, 0) task performs the local multiplication and then passes the partial result to the `GEMM`(m, n, 1) task in the next iteration via the blue "$C \Rightarrow C$" edge. Each task is scheduled only when all required input data blocks are ready, enabling both tile-level and iteration-level parallelism, and overlapping data movement with computation. This dataflow-driven execution model eliminates the need for global synchronization, thereby improving parallel efficiency. The algorithm is implemented using the PaRSEC runtime system, which serves as the execution backbone. Leveraging PaRSEC enables efficient parallel processing of matrix operations while optimizing the trade-offs between computation, communication, and memory usage, particularly in the presence of imbalanced workloads introduced by the adaptive tile-centric mixed-precision algorithm.

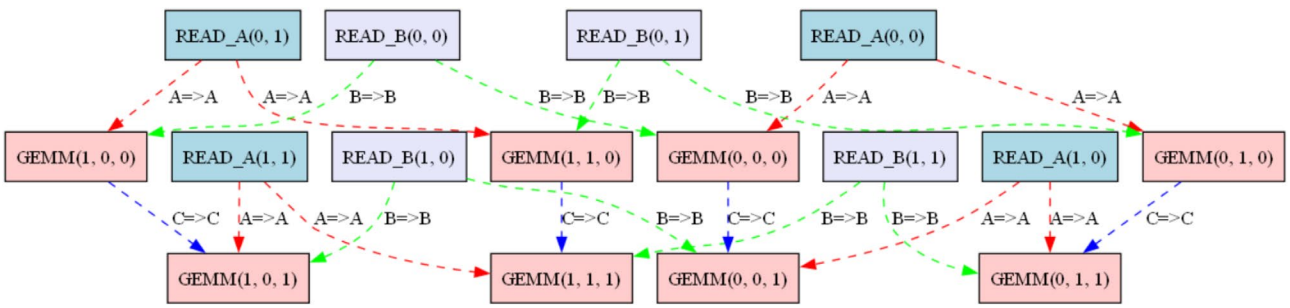


Fig. 2 Corresponding DAG illustrating the dependencies among tasks. `READ_A`, `READ_B`, and `GEMM` are the three task classes listed in Table 1. Different colors of dashed arrows indicate distinct types of data dependencies: red denotes flows between task classes `READ_A` and `GEMM`; blue represents intra-class flows within `GEMM`; green corresponds to flows between task classes `READ_B` and `GEMM`. The symbol $\Rightarrow$ indicates a data dependency between tasks, where $X \Rightarrow Y$ means that data X from the sender task is transmitted to the receiver task as data Y

Performance Results and Analysis

Experimental Settings

The experiments are conducted on various systems.

- `Saturn`: This compute environment is built around dual socket Intel Xeon E5-2650 v3 Haswell-EP processors. Each socket provides 10 physical cores and supports Hyper Threading for 20 hardware threads. The base frequency is 2.30 GHz with turbo boost up to 3.00 GHz.
- `Fugaku`: This system is built on ARM architecture and consists of more than 150,000 compute nodes. Each node is powered by a 48-core A64FX CPU, capable of reaching a peak frequency of 2.2 GHz in boost mode and 2.0 GHz in normal mode, and is equipped with 32 GB of HBM2 memory.
- `Leconte`: It is a GPU-accelerated Intel compute node at ICL, which is powered by two Intel Xeon E5-2698 v4 CPUs, each with 20 cores and a base frequency of 2.20 GHz. The compute node is equipped with eight NVIDIA

Tesla V100-SXM2-32GB GPUs, each with 32 GB of HBM2 memory. The deployed CUDA version is 12.5.

- `Guyot`: It is a GPU-accelerated AMD compute node at ICL, which features two EPYC 7742 CPUs, each with 64 cores running at 2.25 GHz, 2,063 GB of main memory, and eight NVIDIA A100-SXM4-80GB GPUs.
- `Polaris`: It is a Cray-built HPC system equipped with dual-socket AMD EPYC 7713 64-cores processors at up to 2.0 GHz. Its GPU-accelerated compute nodes run on a single-socket AMD EPYC 7543P CPU paired with four NVIDIA A100-SXM4 accelerators, each fitted with 40 GB of HBM2 memory.
- `Hexane`: This is a GPU-based Intel compute node at ICL. It is equipped with two 8-core Xeon Silver 4309Y CPUs clocked at 2.80 GHz, 63 GB of main memory, and a single NVIDIA H100 PCIe GPU. The deployed CUDA version is 12.1.1.
- `Frontier`: This is a GPU-accelerated supercomputer utilizing AMD hardware, comprising 9,408 compute nodes. Each node integrates a 64-core AMD Optimized 3rd Gen EPYC processor alongside four AMD MI250X Graphics Compute Dies (GCDs), supported by 512 GB of DDR4 memory.

We use the term "'a'D:'b'S" to represent the percentage of different precision formats, where D/S denotes double/single precision, and a/b represents the percentage of double/single precision, ensuring $a + b = 100$. For data distribution, we employ a 2D block cyclic scheme with a process grid of size $P \times Q$, structured to be as square as possible. On `Fugaku`, we execute in normal mode due to power restrictions and disable the Sector Cache Optimizations (SCO) due to memory conflicts [22]. Therefore, instead of the theoretical peak (the maximum possible computational performance that the system can achieve under ideal conditions), we use the practical peak performance as an upper bound, defined as the experimental GEMM performance per core multiplied by the total number of cores utilized. Specifically, on Frontier, our configuration utilizes 4 ranks per node in order to mitigate the effects of NUMA-related performance degradation.

Suitable Tile Size Selection

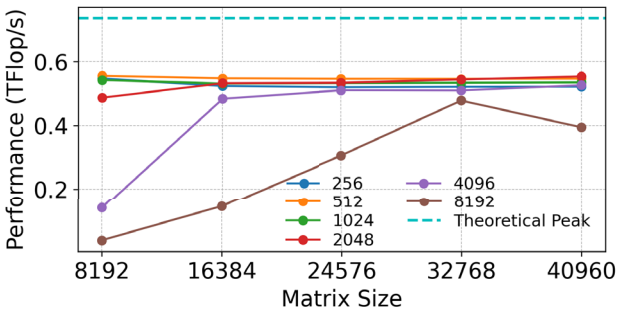
Determining an appropriate tile size is a key concern when designing tile algorithms, as this choice directly influences both computational throughput and the degree of concurrency that can be exploited by the runtime system [38, 57]. The selection process is nontrivial: the optimal tile dimension is not dictated solely by the algorithmic structure but is instead highly sensitive to a combination of architectural and problem-specific parameters. These include, but are not limited to, the number and characteristics of compute resources, network latency and bandwidth, available memory capacity, and the dimensions of the matrices being processed.

Several crucial insights help to elucidate how tile granularity impacts the overall efficiency. As the tile size is reduced, each computational kernel tends to exhibit lower arithmetic intensity, meaning that the ratio of floating-point operations to data movement decreases. This, in turn, heightens the pressure on the memory subsystem and amplifies the overhead associated with task management and synchronization within the underlying task-based execution framework. On the other hand, adopting a larger tile size leads to kernels with higher arithmetic intensity and better use of the compute pipelines, but at the cost of reducing the total number of independent tasks, thereby limiting the potential for parallel execution and possibly underutilizing available resources.

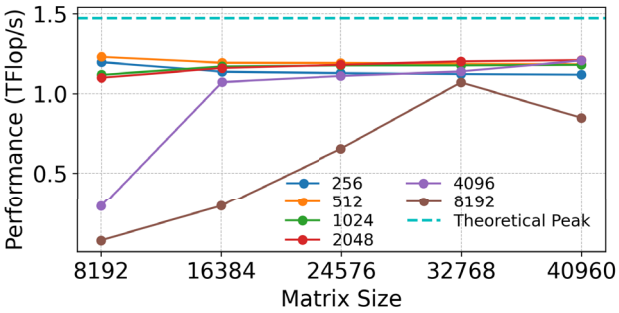
In practice, there exists a tension between these competing effects, and navigating this trade-off is essential to achieving high performance. When assuming that tiles are square and densely populated, and provided that the per-task computational complexity is known, it is theoretically possible to derive an optimal tiling and task decomposition strategy [58]. Such analysis offers guidance on how to balance blocking factors in order to enhance scalability without sacrificing efficiency. However, such theoretical work is out of the scope of this paper and is one of our future works. Therefore, this paper experimentally obtains the most suitable tile size on different architectures in terms of `100D:0 S` and `0D:100 S`, as shown in Fig. 3.

Figure 3 presents the results of the tile-size sweep conducted on a diverse set of representative systems, including one CPU-based platform and three generations of Nvidia GPUs. Separate plots are provided for the `100D:0 S` and `0D:100 S` precision configurations. In each plot, individual lines correspond to a specific tile size and track their performance as the matrix dimensions increase. Across all subfigures, the dashed reference lines labeled 'Theoretical Peak' and 'Practical Peak' denote each device's FP64 theoretical peak (from spec-sheet) and practical peak (DGEMM-measured sustained peak). Unless otherwise noted, the dashed reference lines labeled 'Theoretical Peak' and 'Practical Peak' in all subsequent figures use the same definitions. Generally, for a given tile size, performance tends to improve with increasing matrix size until it reaches a saturation point. For a fixed matrix size, an intermediate tile size typically delivers the highest performance.

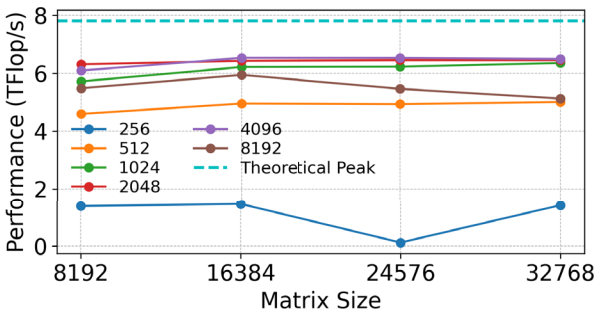
On the CPU-based `Saturn` system, performance under both `100D:0 S` and `0D:100 S` configurations exceeds 80% of the theoretical peak, indicating highly efficient utilization of computational resources. In GPU-accelerated



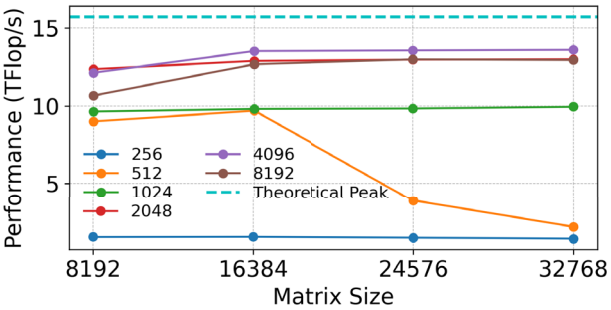
(a) `Saturn: 100D:0S`.



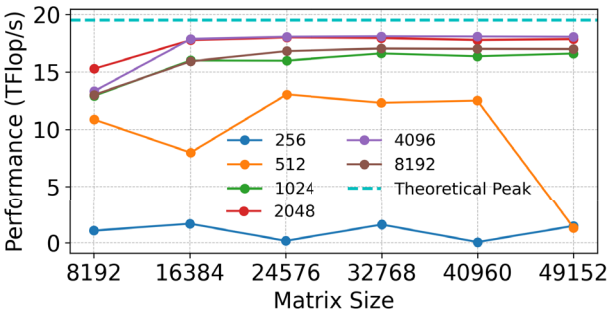
(b) `Saturn: 0D:100S`.



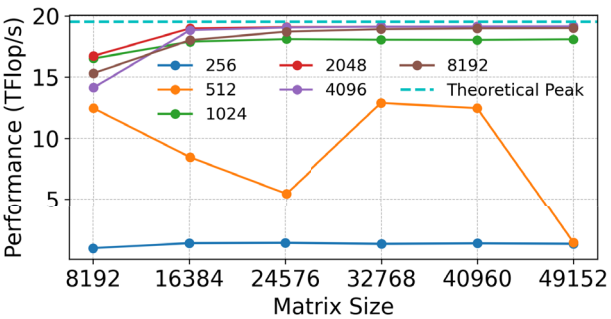
(c) One V100 on `Leconte: 100D:0S`.



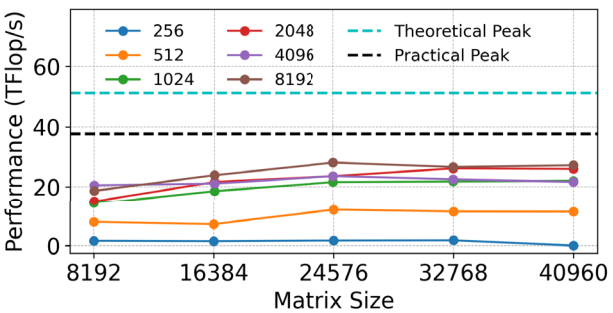
(d) One V100 on `Leconte: 0D:100S`.



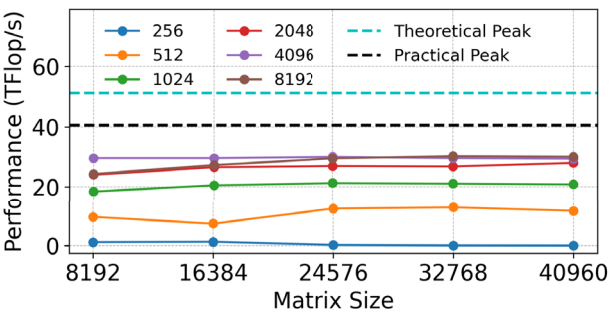
(e) One A100 on `Guyot: 100D:0S`.



(f) One A100 on `Guyot: 0D:100S`.



(g) One H100 on `Hexane: 100D:0S`.



(h) One H100 on `Hexane: 0D:100S`.

Fig. 3 Tile size evaluations across various hardware environments. In each subfigure, each line represents the performance corresponding to a different tile size

experiments, all systems demonstrate strong performance relative to their upper-bound limits. On Leconte, which uses NVIDIA V100 GPUs connected via PCIe, data transfer latency contributes to more modest performance. In contrast, Guyot 's A100 GPUs, equipped with high-bandwidth SXM interconnects, exhibit exceptional throughput: for the 100D:0 S case, performance nearly matches that of 0D:100 S because of the shared theoretical peak of FP64 and FP32, and the latter approaches the theoretical peak with remarkable closeness. Hexane delivers slightly lower performance but aligns well with the system's practical peak. This discrepancy is likely due to the combination of a low-bandwidth PCIe connection and higher computational capabilities on Hexane, which also results in 0D:100 S slightly outperforming 100D:0 S.

Guided by these experimental insights, we select for each configuration the tile size that maximizes performance and use it in all subsequent experiments. After fixing the tile size, we then randomly generate the decision map over the resulting tiles, as demonstrated in the following Sect. 6.3. Since the suitable tile sizes are small compared to the overall matrix dimensions, their impact on the decision map–and consequently on performance–is limited.

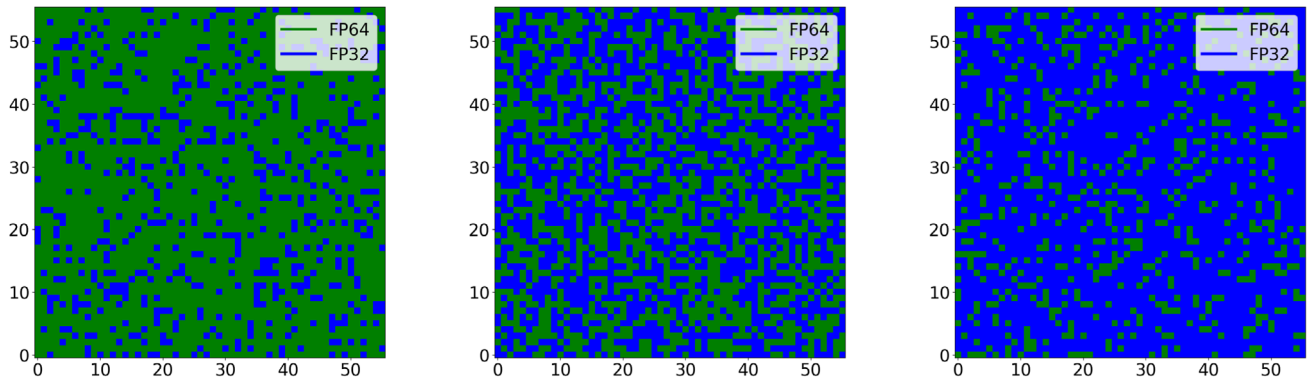
Precision Map of Kernel Execution

Figure 4 visualizes the precision of numerical kernels executed on each individual tile, as referenced in Fig. 1, by presenting a heatmap under different configurations. Each point represents the precision of a tile in FP64 or FP32, randomly generated in our settings. The three subfigures illustrate different distributions of precision in A, B, and C: (a) 80D:20 S, where 80% of the tiles use double precision (FP64) and 20% use single precision (FP32); (b) 50D:50 S, where FP64 and FP32 tiles are equally distributed; and (c) 20D:80 S, where 20% of the tiles use FP64 and 80% use FP32. These configurations exhibit varying degrees of mixed-precision computation, which will be employed in the subsequent performance measurements. To maintain precision information, the precision maps for matrices A, B, and C are pre-initialized. Each precision map is stored as an unsigned 16-bit 2D integer array, where each entry encodes the precision of a corresponding tile. The initialization of matrix data depends on the precision map. During GEMM execution, the precision maps are referenced to check if precision conversion is necessary for tiles in A and B. This approach ensures that the resulting precision of GEMM aligns with the precision specified for the corresponding tile in matrix C. For performance evaluation, precision maps for matrices A,B, and C are randomly generated based on a predetermined ratio of double precision and single precision. Matrix values are then initialized accordingly, adhering to the precision defined in the maps.

Shared Memory Performance

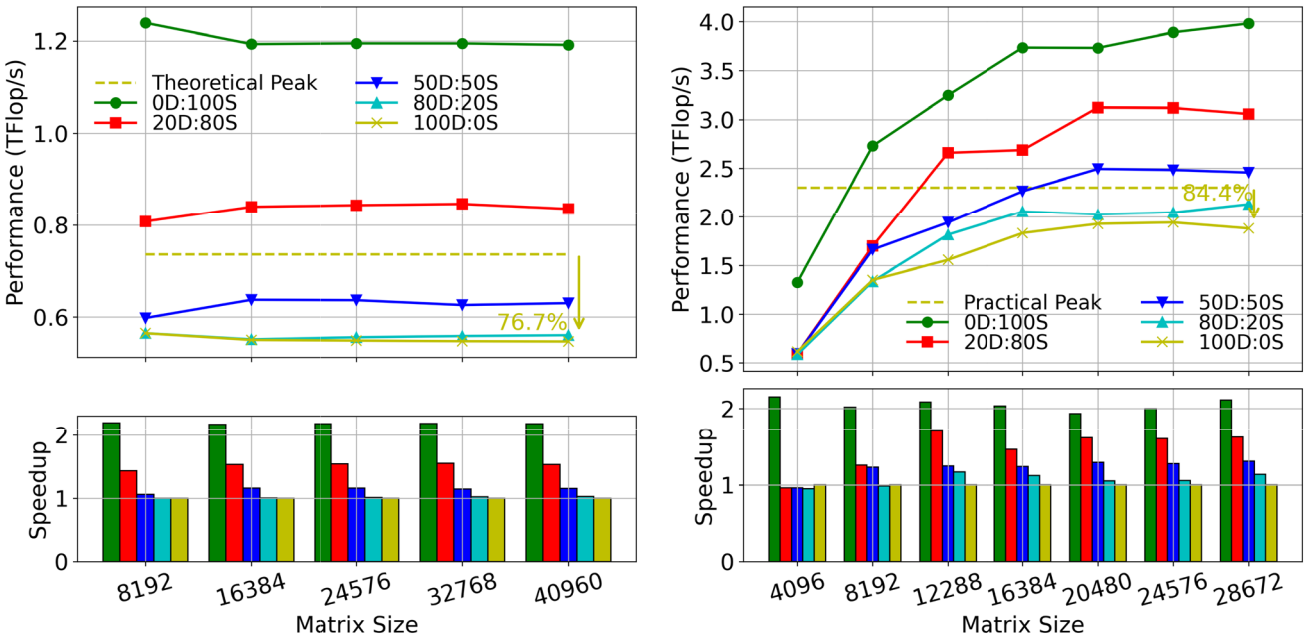
Figures 5, 6, 7 present the evaluations on shared memory across a variety of hardware platforms. Each subfigure illustrates the performance under different precision configurations, along with the corresponding speedup relative to the baseline 100D:0 S. The results demonstrate that varying the precision mix leads to distinct scaling trends, and performance generally improves as the proportion of lower precision computations increases.

Figure 5 show the results on CPU-based systems. On a single Saturn node, as shown in Fig. 5a, the 100D:0 S configuration achieves 76.7% of the theoretical peak performance (the dashed line). In contrast, the 0D:100 S configuration surpasses twice the performance of 100D:0 S, which aligns with the hardware characteristic that FP32 throughput is typically about double that of FP64 on CPUs. The performance on a single Saturn node does not increase with larger matrix sizes, as the tested matrix sizes are already sufficient to saturate the available throughput.



(a) 80D:20S. (b) 50D:50S. (c) 20D:80S.

Fig. 4 Kernel precision heatmap for matrix on one node of Frontier with size 114688 × 114688 and tile size 2048 × 2048



(a) One Saturn node. (b) One node on Fugaku.

Fig. 5 Shared-memory performance on one CPU node of GEMM-MP

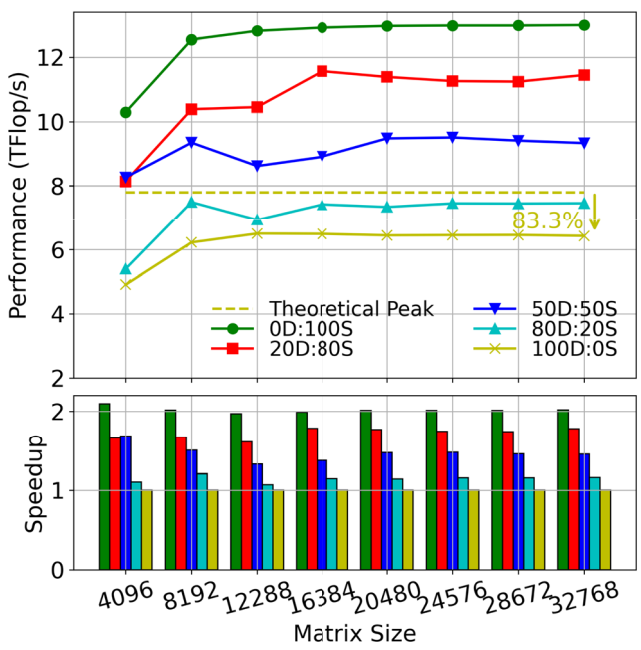
Similarly, on a single Fugaku node, the 100D:0 S configuration attains approximately 1.94 TFlop/s, representing 84.4% of the practical peak. The 0D:100 S configuration achieves nearly double that performance, as depicted in Fig. 5b.

When running on GPU systems, GEMM-MP also demonstrates strong performance. We first examine the performance on a single GPU, as shown in Fig. 6.

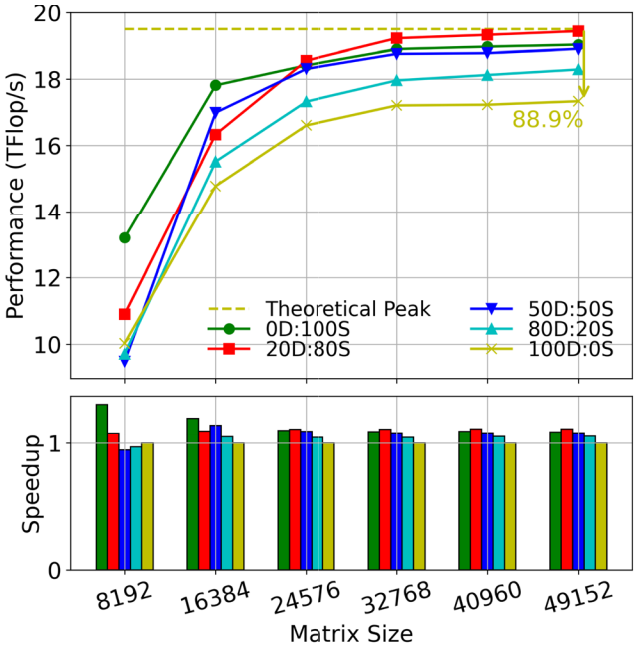
- In Fig. 6a, we present results on a single NVIDIA V100 GPU on Leconte. The 100D:0 S configuration achieves 83.3% of the device's FP64 theoretical peak, while the 0D:100 S configuration reaches 82.8% of the FP32 theoretical peak, indicating balanced efficiency across precision modes.
- Fig. 6b shows performance on a single NVIDIA A100-SXM4-80GB GPU on Guyot, where FP64 and FP32 share the same theoretical peak. Here, the 100D:0 S configuration attains up to 88.9% of the theoretical peak. The observed drop in performance with lower FP32 ratios is likely due to increased data movement and/or the overhead introduced by datatype conversions.
- Similarly, Fig. 6c reports performance on a single NVIDIA A100-SXM4-40GB GPU on Polaris. The 100D:0 S configuration achieves 92.2% of the theoretical peak, highlighting efficient utilization of the hardware.
- Fig. 6d presents results on a single NVIDIA H100 GPU on Hexane, where the 100D:0 S configuration reaches 53.1% of the theoretical peak and 72.2% of the practical peak, showcasing solid performance even on the latest GPU architecture.

Next, Fig. 7 presents the evaluation of GEMM-MP on a single, shared-memory multi-GPU node under the weak-scaling setting, consistent with Fig. 6, where the data volume per GPU remains unchanged throughout.

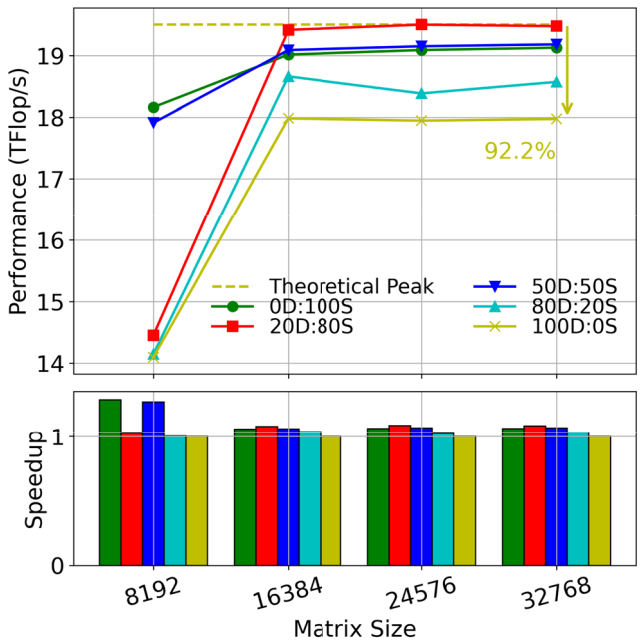
- Fig. 7a reports the performance on eight Nvidia V100 GPUs on Leconte. The aggregate double-precision throughput reaches 51 TFlop/s at a matrix size of 98, 304, corresponding to 81.7% of the theoretical peak of the eight-GPU setup. Notably, the 0D:100 S configuration achieves 102.6 TFlop/s, which is 7.90× the performance of a single Nvidia V100 GPU, indicating a parallel efficiency of 99%.
- Fig. 7b demonstrates the performance on eight Nvidia A100 GPUs on Guyot. The system exhibits nearly linear scaling when compared with the single-GPU performance in Fig. 6b. The 100D:0 S configuration reaches 83.9% of the theoretical peak. The weak-scaling trend from 1 to 8 GPUs is well preserved. Among all configurations, 0D:100 S consistently delivers the highest performance, whereas 100D:0 S trails slightly, achieving 88.9% of peak on 1 GPU and 83.9% on 8 GPUs. Mixed-precision modes, such as 20D:80 S and 50D:50 S, yield competitive performance but remain marginally behind the pure FP32 case due to



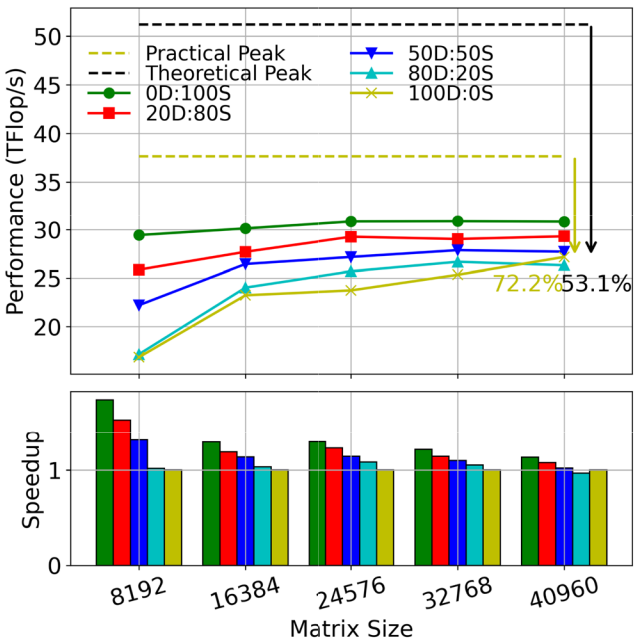
(a) One Nvidia V100 GPU on `Leconte`.



(b) One Nvidia A100 GPU on `Guyot`.



(c) One Nvidia A100 GPU on Polaris.

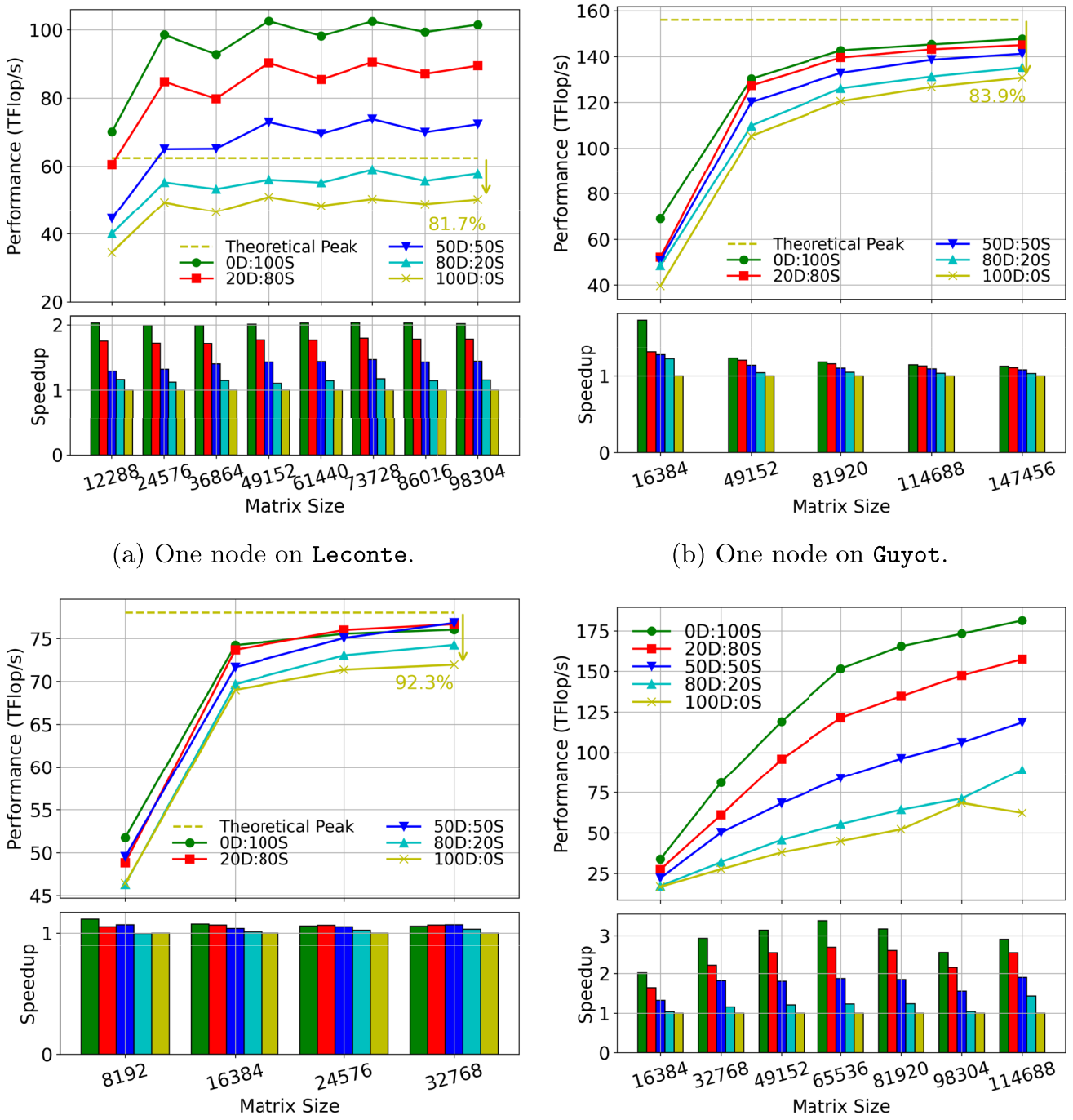


(d) One Nvidia H100 GPU on Hexane.

Fig. 6 Shared-memory performance on one GPU of GEMM-MP

increased data movement overhead from the higher double-precision ratio, which leads to minor performance degradation.

- Similarly, Fig. 7c illustrates strong performance and scalability on one `Polaris` node equipped with four Nvidia A100 GPUs. The `100D:0 S` configuration reaches 92.3% of the theoretical peak, and configurations with a higher single-precision component deliver even better performances.
- Fig. 7d evaluates performance on a single `Frontier` node equipped with four AMD MI250X GCDs. For the `0D:100 S` configuration, the practical peak is around



(a) One node on `Leconte`.

(b) One node on `Guyot`.

(c) One node on `Polaris`.

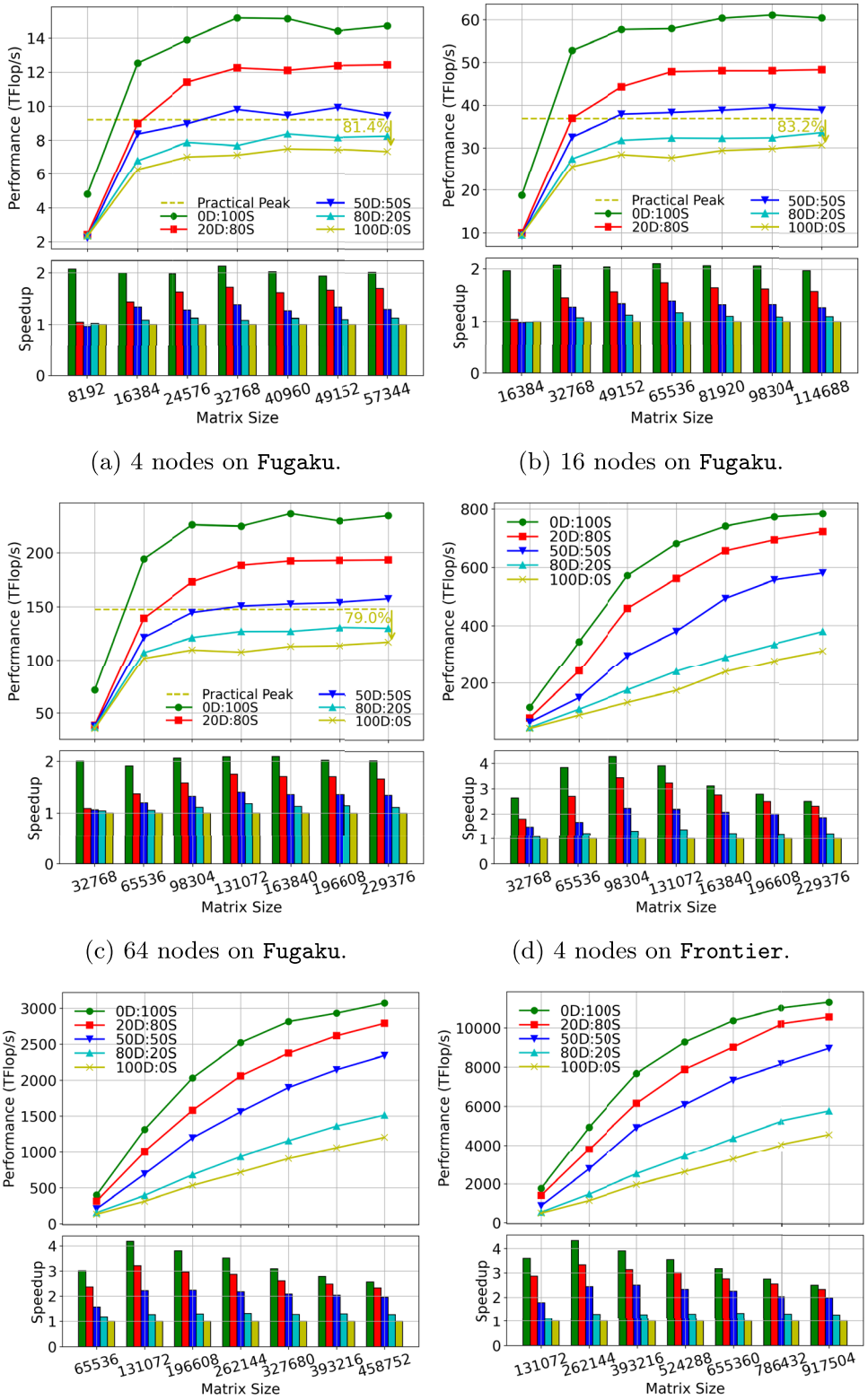
(d) One node on `Frontier`.

Fig. 7 Shared-memory performance on one node of multi-gpus of GEMM-MP

280.0 TFlop/s, with observed performance reaching 181.3 TFlop/s, which accounts for 64.8% of the practical peak. The `100D:0 S` configuration shows somewhat lower performance, which may be attributed to the GPU-NIC topology on `Frontier` and the current stage of GPU-direct communication support in PaRSEC.

Performance Scalability

Figure 8 presents the distributed-memory performance of GEMM-MP on `Fugaku` and `Frontier` using 4, 16, and 64 nodes under various mixed-precision configurations. Focusing first on the pure single?precision mode (`0D:100 S`) on `Fugaku` at the largest matrix size, the



(a) 4 nodes on `Fugaku`.

(b) 16 nodes on `Fugaku`.

(c) 64 nodes on `Fugaku`.

(d) 4 nodes on `Frontier`.

(e) 16 nodes on `Frontier`.

(f) 64 nodes on `Frontier`.

Fig. 8 Distributed-memory performance of GEMM-MP

performance scales from 3.98 Tflop/s on a single node to 14.7 Tflop/s on four nodes, representing a 3.7× speedup with 92.3% parallel efficiency. This continues to 60.4 Tflop/s on sixteen nodes (15.2×, 94.9% efficiency), and ultimately reaches 242.2 Tflop/s on sixty-four nodes (60.9×, 95.0% efficiency). In addition, mixed-precision configurations demonstrate even better parallel efficiency than the pure single-precision baseline. For instance, `50D:50 S` achieves more than 99.9% parallel efficiency, highlighting the effectiveness of precision-aware task orchestration.

Turning to the evaluations on `Frontier`, we again consider the pure single?precision mode (`0D:100 S`) at the largest matrix size. On a single node, the performance reaches 181.3 Tflop/s. When scaled to four nodes, throughput increases to 784.0 Tflop/s (4.4× speedup, 108.1% efficiency), and further scales to approximately 3075.3 Tflop/s on sixteen nodes (16.8×, 106.0% efficiency). At sixty-four nodes, the performance peaks at around 11, 310.3 Tflop/s, corresponding to a 63× speedup and 97.5% efficiency. Similar to the trend observed on `Fugaku`, GEMM-MP on `Frontier` maintains exceptionally high parallel efficiency in mixed-precision modes. Notably, the `50D:50 S` mix surpasses ideal scaling, achieving approximately 117.8% parallel efficiency. This superlinear behavior may stem from the overlap between computation and communication enabled by the mixed-precision design.

All in all, these results underscore the effectiveness of the proposed adaptive tile-centric GEMM-MP in delivering excellent computational efficiency and scalability across shared- and distributed-memory systems on diverse architectural platforms.

Conclusion and Future Work

This work introduces GEMM-MP, an adaptive tile-centric mixed-precision GEMM framework. Built on PaRSEC, the proposed approach efficiently manages heterogeneous tasks that arise from varying precision formats, enabling high-performance execution across both shared- and distributed-memory environments. GEMM-MP exhibits strong efficiency and scalability on homogeneous CPU platforms as well as heterogeneous GPU systems, showcasing its adaptability to a wide spectrum of hardware architectures.

Looking ahead, we plan to enhance the framework by incorporating additional low-precision formats (e.g., FP8, FP4) and devising robust precision selection strategies suitable for both lossless and lossy computations. We also intend to explore various data type conversion techniques to improve performance and numerical stability. Another avenue of development involves constructing a theoretical model capable of predicting optimal tile sizes under different architectural and algorithmic configurations. Beyond kernel-level optimizations, we aim to integrate this mixed-precision framework with alternative runtime systems and evaluate its applicability in real-world HPC and AI workloads.

Acknowledgements This research was supported by internal awards from Saint Louis University (Grant-0001651 and PROJ-000498), as well as by the U.S. National Science Foundation under Award OAC-2451577. For computational resources, this work utilized the compute node at the Innovative Computing Laboratory of the University of Tennessee, Knoxville, the `Fugaku` supercomputer at RIKEN, the `Polaris` supercomputer at Argonne National Laboratory, and the `Frontier` supercomputer at Oak Ridge National Laboratory.

Author Contributions Authors are listed in order of contribution. Qinglei Cao serves as the corresponding author and supervises the overall project.

Funding This work was supported by the U.S. National Science Foundation under Award OAC-2451577.

Data Availibility PaRSEC is publicly available at https://github.com/ICLDisco/parsec. Other source codes used in this study are hosted on GitHub and will be released in the future.

Declarations

Conflict of interest Not Applicable.

Research Involving Human and/or Animals Not Applicable.

Informed Consent Not Applicable.

References

1. Meuer H, Strohmaier E, Dongarra J, Simon, H. The Top500 List; 2024. http://www.top500.org
2. Abdulah S, Cao Q, Pei Y, Bosilca G, Dongarra J, Genton MG, et al. Accelerating geostatistical modeling and prediction with mixed-precision computations: a high-productivity approach with parsec. IEEE Trans Parallel Distrib Syst. 2021;33(4):964–76.
3. Cao Q, Abdulah S, Ltaief H, Genton M,G, Keyes D, Bosilca G. Reducing data motion and energy consumption of geospatial modeling applications using automated precision conversion. In: 2023 IEEE International Conference on Cluster Computing (CLUSTER), 2023;pp. 330–342. IEEE
4. Haidar A, Abdelfattah A, Zounon M, Wu P, Pranesh S, Tomov S, Dongarra J. The design of fast and energy-efficient linear solvers: on the potential of half-precision arithmetic and iterative refinement techniques. In: Computational Science–ICCS 2018: 18th International Conference, Wuxi, China, 2018;11–13, Proceedings, Part I, pp. 586–600 (2018). Springer.
5. Haidar A, Bayraktar H, Tomov S, Dongarra J, Higham NJ. Mixed-precision iterative refinement using tensor cores on gpus to accelerate solution of linear systems. Proc R Soc A. 2020;476(2243):20200110.
6. Netti A, Peng Y, Omland P, Paulitsch M, Parra J, Espinosa G, et al. Mixed precision support in hpc applications: what about reliability? J Parallel Distrib Comput. 2023;181:104746.

7. Nandakumar S, Le Gallo M, Piveteau C, Joshi V, Mariani G, Boybat I, et al. Mixed-precision deep learning based on computational memory. Front Neurosci. 2020;14:406.
8. Walden A, Nielsen E, Diskin B, Zubair M. A mixed precision multicolor point-implicit solver for unstructured grids on gpus. In: 2019 IEEE/ACM 9th workshop on irregular applications: architectures and algorithms (IA3), 2019;23–30. IEEE.
9. Brogi F, Bnà S, Boga G, Amati G, Ongaro TE, Cerminara M. On floating point precision in computational fluid dynamics using openfoam. Futur Gener Comput Syst. 2024;152:1–16.
10. Doucet N, Ltaief H, Gratadour D, Keyes D. Mixed-precision tomographic reconstructor computations on hardware accelerators. In: 2019 IEEE/ACM 9th Workshop on Irregular Applications: Architectures and Algorithms (IA3), 2019;pp. 31–38. IEEE.
11. Chen S, Zhang Y, Wang Y, Liu Z, Li X, Xue W. Mixed-precision computing in the grist dynamical core for weather and climate modelling. Geosci Model Dev Discuss. 2024;2024:1–28.
12. Jia W, Wang H, Chen M, Lu D, Lin L, Car R, Weinan E, Zhang L. Pushing the limit of molecular dynamics with ab initio accuracy to 100 million atoms with machine learning. In: SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, 2020;pp. 1–14. IEEE.
13. Boku T, Ishikawa K-I, Kuramashi Y, Meadows L. Mixed precision solver scalable to 16000 mpi processes for lattice quantum chromodynamics simulations on the oakforest-pacs system. In: 2017 Fifth International Symposium on Computing and Networking (CANDAR), 2017;pp. 362–368. IEEE.
14. Wilkinson JH. Rounding errors in algebraic processes. Philadelphia: SIAM; 2023.
15. Moler CB. Iterative refinement in floating point. J ACM (JACM). 1967;14(2):316–21.
16. Buttari A, Dongarra J, Kurzak J, Luszczek P, Tomov S. Using mixed precision for sparse matrix computations to enhance the performance while achieving 64-bit accuracy. ACM Trans Math Softw (TOMS). 2008;34(4):1–22.
17. Buttari A, Dongarra J, Langou J, Langou J, Luszczek P, Kurzak J. Mixed precision iterative refinement techniques for the solution of dense linear systems. Int J High Perform Comput Appl. 2007;21(4):457–66.
18. Baboulin M, Buttari A, Dongarra J, Kurzak J, Langou J, Langou J, et al. Accelerating scientific computations with mixed precision algorithms. Comput Phys Commun. 2009;180(12):2526–33.
19. Haidar A, Tomov S, Dongarra J, Higham NJ. Harnessing gpu tensor cores for fast fp16 arithmetic to speed up mixed-precision iterative refinement solvers. In: SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, 2018;pp. 603–613. IEEE.
20. Higham NJ. Accuracy and stability of numerical algorithms. Philadelphia: SIAM; 2002.
21. Abdelfattah A, Anzt H, Boman EG, Carson E, Cojean T, Dongarra J, et al. A survey of numerical linear algebra methods utilizing mixed-precision arithmetic. Int J High Perform Comput Appl. 2021;35(4):344–69.
22. Cao Q, Abdulah S, Alomairy R, Pei Y, Nag P, Bosilca G, Dongarra J, Genton M,G, Keyes D,E, Ltaief H, Sun Y. Reshaping geostatistical modeling and prediction for extreme-scale environmental applications. In: SC22: International Conference for High Performance Computing, Networking, Storage and Analysis (ACM Gordon Bell Prize Finalist), 2022;pp. 1–12. IEEE.
23. Abdulah S, Baker AH, Bosilca G, Cao Q, Castruccio S, Genton MG, Keyes DE, Khalid Z, Ltaief H, Yan Song GLS, Sun Y. Boosting earth system model outputs and saving petabytes in their storage using exascale climate emulators. ACM Gordon Bell Prize for Climate Modelling Finalist, 2024.
24. Ltaief H, Alomairy R, Cao Q, Ren J, Slim L, Kurth T, Dorschner B, Bougouffa S, Abdelkhalek R, Keyes DE. Toward capturing genetic epistasis from multivariate genome-wide association studies using mixed-precision kernel ridge regression. ACM Gordon Bell Prize Finalist, 2024.
25. Ichimura T, Fujita K, Yamaguchi T, Naruse A, Wells JC, Schulthess TC, Straatsma TP, Zimmer CJ, Martinasso M, Nakajima K, et al. A fast scalable implicit solver for nonlinear time-evolution earthquake city problem on low-ordered unstructured finite elements with artificial intelligence and transprecision computing. In: SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, 2018;pp. 627–637. IEEE.
26. Vaswani A. Attention is all you need. Advances in Neural Information Processing Systems, 2017.
27. Yu C, Chen T, Gan Z. Boost transformer-based language models with GPU-friendly sparsity and quantization. In: Rogers A, Boyd-Graber J, Okazaki N (eds.) Findings of the Association for Computational Linguistics: ACL 2023, pp. 218–235. Association for Computational Linguistics, Toronto, Canada. 2023. https://doi.org/10.18653/v1/2023.findings-acl.15.
28. Srinivasa Kumar PK. Evaluating full int8 quantization and inference techniques for causal language model. Master's thesis, University of Twente, 2025.
29. Zhao Y, Lin C-Y, Zhu K, Ye Z, Chen L, Zheng S, et al. Atom: Low-bit quantization for efficient and accurate llm serving. Proc Mach Learn Syst. 2024;6:196–209.
30. Augonnet C, Thibault S, Namyst R, Wacrenier P. StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. Concurr Comput Pract Exp. 2011;23:187–98.
31. Duran A, Ferrer R, Ayguade E, Badia RM, Labarta J. A proposal to extend the OpenMP tasking model with dependent tasks. Int J Parallel Program. 2009;37(3):292–305. https://doi.org/10.1007/s10766-009-0101-1.
32. OpenMP. OpenMP 4.5 Complete Specifications, 2015. https://www.openmp.org/wp-content/uploads/openmp-4.5.pdf.
33. Heller T, Kaiser H, Iglberger K. Application of the ParalleX execution model to stencil-based problems. Comput Sci Res Dev. 2013;28(2–3):253–61. https://doi.org/10.1007/s00450-012-0217-1.
34. Bauer M, Treichler S, Slaughter E, Aiken A. Legion: Expressing locality and independence with logical regions. In: International Conference for High Performance Computing, Networking, Storage and Analysis, SC, 2012;pp. 1–11. https://doi.org/10.1109/SC.2012.71. IEEE.
35. Bosilca G, Bouteiller A, Danalis A, Faverge M, Herault T, Dongarra J. PaRSEC: a Programming Paradigm Exploiting Heterogeneity for Enhancing Scalability. Comput Sci Eng. 2013;99:1. https://doi.org/10.1109/MCSE.2013.98.
36. Danalis A, Jagode H, Bosilca G, Dongarra J. PaRSEC in practice: optimizing a legacy chemistry application through distributed task-based execution. In: 2015 IEEE International Conference on Cluster Computing, 2015;pp. 304–313.
37. Jagode H, Danalis A, Dongarra J. Accelerating nwchem coupled cluster through dataflow-based execution. Int J High Perform Comput Appl. 2017;32(4):540–51. https://doi.org/10.1177/1094342016672543. (**1–13**).
38. Cao Q, Pei Y, Herault T, Akbudak K, Mikhalev A, Bosilca G, Ltaief H, Keyes D, Dongarra J. Performance analysis of tile low-rank cholesky factorization using parsec instrumentation tools. In: 2019 IEEE/ACM International Workshop on Programming and Performance Visualization Tools (ProTools), 2019;pp. 25–32. IEEE.
39. Herault T, Robert Y, Bosilca G, Harrison RJ, Lewis CA, Valeev EF, Dongarra JJ. Distributed-memory multi-gpu block-sparse tensor contraction for electronic structure. In: 2021 IEEE

International Parallel and Distributed Processing Symposium (IPDPS), 2021;pp. 537–546. IEEE.
40. Cao Q, Alomairy R, Pei Y, Bosilca G, Ltaief H, Keyes D, Dongarra J. A framework to exploit data sparsity in tile low-rank cholesky factorization. In: 2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2022;pp. 414–424. IEEE.
41. Alomairy R, Cao Q, Ltaief H, Keyes D, Edelman A. Scalable hamming distance computation using accelerated matrix transformations. In: ISC High Performance 2025 Research Paper Proceedings (40th International Conference), 2025;pp. 1–13. Prometeus GmbH.
42. Zhang Q, Alomairy R, Wang D, Gu Z, Cao Q. Leveraging Hardware-Aware Computation in Mixed-Precision Matrix Multiply: A Tile-Centric Approach, 2025. https://arxiv.org/abs/2508.14848.
43. Jouppi NP, Young C, Patil N, Patterson DA, Agrawal G, Bajwa R, Bates S, Bhatia S, Boden N, Borchers A, Boyle R, Cantin P-L, Chao C, Clark C, Coriell J, Daley M, Dau M, Dean J, Gelb B, Ghaemmaghami TV, Gottipati R, Gulland W, Hagmann R, Ho CR, Hogberg D, Hu J, Hundt R, Hurt D, Ibarz J, Jaffey A, Jaworski A, Kaplan A, Khaitan H, Koch A, Kumar N, Lacy S, Laudon J, Law J, Le D, Leary C, Liu Z, Lucke K, Lundin A, MacKean G, Maggiore A, Mahony M, Miller K, Nagarajan R, Narayanaswami R, Ni R, Nix K, Norrie T, Omernick M, Penukonda N, Phelps A, Ross J, Salek A, Samadiani E, Severn C, Sizikov G, Snelham M, Souter J, Steinberg D, Swing A, Tan M, Thorson G, Tian B, Toma H, Tuttle E, Vasudevan V, Walter R, Wang W, Wilcox E, Yoon DH. In-datacenter performance analysis of a tensor processing unit. In: Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA), 2017;pp. 1–12. https://doi.org/10.1145/3079856.3080246. IEEE/ACM.
44. Gupta S, Agrawal A, Gopalakrishnan K, Narayanan P. Deep learning with limited numerical precision. CoRR, 2015. https://doi.org/10.48550/arXiv.1502.02551.
45. Micikevicius P, Narang S, Alben J, Diamos G, Elsen E, Garcia D, Ginsburg B, Houston M, Kuchaiev O, Venkatesh G, Wu H. Mixed precision training. In: International Conference on Learning Representations (ICLR), 2018. https://openreview.net/forum?id=r1gs9JgRZ.
46. Lordan F, Tejedor E, Ejarque J, Rafanell R, Alvarez J, Marozzo F, et al. Servicess: an interoperable programming framework for the cloud. J Grid Comput. 2014;12(1):67–91.
47. Bosilca G, Bouteiller A, Danalis A, Hérault T, Lemarinier P, Dongarra JJ. DAGuE: a generic distributed DAG engine for high performance computing. Parallel Comput. 2012;38(1–2):37–51.
48. Bouteiller A, Herault T, Cao Q, Schuchart J, Bosilca G. PaRSEC: scalability, flexibility, and hybrid architecture support for task-based applications in ECP. Int J High Perform Comput Appl. 2024;39(1):147–66.
49. Cao Q, Bosilca G, Losada N, Wu W, Zhong D, Dongarra J. Evaluating data redistribution in parsec. IEEE Trans Parallel Distrib Syst. 2021;33(8):1856–72.
50. Cao Q, Pei Y, Akbudak K, Bosilca G, Ltaief H, Keyes D, Dongarra J. Leveraging parsec runtime support to tackle challenging 3d data-sparse matrix problems. In: 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2021;pp. 79–89. IEEE.
51. Danalis A, Bosilca G, Bouteiller A, Herault T, Dongarra J. PTG: an abstraction for unhindered parallelism. In: 2014 Fourth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing, 2014;pp. 21–30. IEEE.
52. Bosilca G, Harrison RJ, Herault T, Javanmard MM, Nookala P, Valeev EF. The template task graph (TTG)-an emerging practical dataflow programming paradigm for scientific simulation at extreme scale. In: IEEE/ACM 5th International Workshop on Extreme Scale Programming Models and Middleware (ESPM2), 2020. IEEE.
53. Hoque R, Herault T, Bosilca G, Dongarra J. Dynamic task discovery in PaRSEC: a data-flow task-based runtime. In: Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems. ScalA '17, 2017.
54. Thomadakis P, Chrisochoides N. Runtime support for performance portability on heterogeneous distributed platforms. CoRR, 2023. arXiv:2303.02543 [cs.DC].
55. Van De Geijn RA, Watts J. Summa: scalable universal matrix multiplication algorithm. Concurr Pract Exp. 1997;9(4):255–74.
56. Choi J, Demmel J, Dhillon IS, Ostrouchov S, Petitet A, Stanley K, Walker DW, Whaley RC. ScaLAPACK: a portable linear algebra library for distributed memory computers—design issues and performance. In: Dongarra, J, Madsen, K, Wasniewski, JE (eds.) Proceedings of the Second International Workshop on Applied Parallel Computing: Computations in Physics, Chemistry and Engineering Science (PARA '95), Lyngby, Denmark, August 21-24, 1995. Lecture Notes in Computer Science, vol. 1041, pp. 95–106. Springer, Berlin, Heidelberg, 1996. https://doi.org/10.1007/3-540-60902-4_12.
57. Kurzak J, Anzt H, Gates M, Dongarra J. Implementation and tuning of batched cholesky factorization and solve for nvidia gpus. IEEE Trans Parallel Distrib Syst. 2015;27(7):2036–48.
58. Buttari A, Langou J, Kurzak J, Dongarra J. A class of parallel tiled linear algebra algorithms for multicore architectures. Parallel Comput. 2007;35(1):38–53.