Ma ZX, Jin YY, Tang SZ *et al.* Unified programming models for heterogeneous high-performance computers. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 38(1): 211-218 Jan. 2023. DOI: 10.1007/s11390-023-2888-4

Unified Programming Models for Heterogeneous High-Performance Computers

Zi-Xuan Ma (马子轩), *Student Member, CCF*, Yu-Yang Jin (金煜阳), *Member, CCF*
Shi-Zhi Tang (唐适之), *Student Member, CCF*, Hao-Jie Wang (王豪杰), *Member, CCF*
Wei-Cheng Xue (薛伟诚), Ji-Dong Zhai* (翟季冬), *Senior Member, CCF*, and
Wei-Min Zheng (郑纬民), *Fellow, CCF*

Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China

E-mail: mzx22@mails.tsinghua.edu.cn; jinyuyang@tsinghua.edu.cn; tsz19@mails.tsinghua.edu.cn; wanghaojie@tsinghua.edu.cn
greatsincere@mail.tsinghua.edu.cn; zhaijidong@tsinghua.edu.cn; zwm-dcs@tsinghua.edu.cn

Received October 5, 2022; accepted January 10, 2023.

Abstract Unified programming models can effectively improve program portability on various heterogeneous high-performance computers. Existing unified programming models put a lot of effort to code portability but are still far from achieving good performance portability. In this paper, we present a preliminary design of a performance-portable unified programming model including four aspects: programming language, programming abstraction, compilation optimization, and scheduling system. Specifically, domain-specific languages introduce domain knowledge to decouple the optimizations for different applications and architectures. The unified programming abstraction unifies the common features of different architectures to support common optimizations. Multi-level compilation optimization enables comprehensive performance optimization based on multi-level intermediate representations. Resource-aware lightweight runtime scheduling system improves the resource utilization of heterogeneous computers. This is a perspective paper to show our viewpoints on programming models for emerging heterogeneous systems.

Keywords performance portability, programming model, heterogeneous supercomputer

1 Introduction and Motivation

As the development of Moore's Law and Dennard Scaling slows down, high-performance computers based on heterogeneous architectures are becoming the major trend for supercomputers. In the latest TOP500[1] list, nine of the 10 most powerful supercomputers in the world use heterogeneous processors. The first-ranked supercomputer, Frontier, and the third-ranked supercomputer, LUMI, use AMD MI250X GPU. The former top-ranked supercomputers, Summit[2], and Sunway TaihuLight[3], use NVIDIA V100 GPU and SW26010 heterogeneous processors, respectively.

To fully utilize the performance of heterogeneous architectures, each hardware vendor has developed its toolkit, including programming languages, compilers, and private libraries for its accelerators. Therefore, a significant challenge for developers is program portability. When porting applications from one architecture to another, developers must rewrite the code to adapt specific programming frameworks and perform special optimizations to fit the hardware design of each accelerator. For example, Fu *et al.* from Tsinghua University[4] ported Community Atmosphere Model (CAM)[5] of the US National Center for Atmospheric Research to Sunway TaihuLight. It took more than 10 person-years to complete the preliminary portion. Therefore, high porting cost becomes a considerable burden for application developers, which largely limits the development and promotion of high-performance computing.

Perspective
Special Issue in Honor of Professor Kai Hwang's 80th Birthday
This work is partially supported by the National Natural Science Foundation of China under Grant No. 62225206.
*Corresponding Author

To address these challenges, Exascale Computing Project (ECP) has funded several research projects, including Kokkos[6, 7], RAJA[8], and SYCL[①]. Kokkos and RAJA provide a unified programming model based on C++ in an attempt to achieve code portability on different heterogeneous architectures. Kokkos provides a unified programming abstraction in the form of library functions, including data structure and parallel execution primitives. RAJA mainly targets parallel loops and supports automatic equivalent transformations of complex loops, effectively reducing the complexity of writing specific optimization code. SYCL is a C++-based open standard that enables users to develop various applications for different heterogeneous architectures with the same language. Currently, there are multiple implementations of the SYCL standard. For example, Intel oneAPI[9] directly compiles SYCL code into binaries for different target architectures, such as CPUs, Intel GPUs, NVIDIA GPUs, and AMD GPUs.

These programming models enable compiling and executing the same program on different heterogeneous architectures using a unified syntax, which means they achieve code portability on various heterogeneous architectures. However, they fail to achieve performance portability, which means the ability to achieve the same computational efficiency on different heterogeneous architectures with the same code. Pennycook *et al.*[10] defined it as the harmonic mean of the efficiency of different computers running a specific application solving the same problem. Current unified programming models, such as Kokkos and SYCL, are unable to achieve this goal. Lin and McIntosh-Smith[11] compared the efficiency of different homogeneous and heterogeneous computers running the miniBUDE benchmark written in Kokkos[7] and native programming languages, including OpenMP, CUDA, and OpenCL. Although the efficiency achieved by Kokkos matches what is achieved by OpenMP on homogeneous architectures, Kokkos only achieves 64% of CUDA's efficiency on computers equipping NVIDIA GPUs, and only 44% of OpenCL's efficiency on computers equipping AMD GPUs. It shows the limitation of Kokkos to achieve performance portability among these computers.

The main reasons for the above issues lie in two aspects. On the one hand, existing unified abstractions pay more attention to the underlying hardware systems and lack the description of high-level applications, making it difficult to apply many application-related optimizations using these abstractions. On the other hand, it is also difficult for compilers based on such unified programming abstractions to pass high-level application-related optimizations to underlying hardware to optimize for a specific architecture. Therefore, such programming models cannot achieve satisfactory performance portability.

There are mainly three challenges to realizing the performance portability of programming on different heterogeneous architectures.

• *Diversity of Application Workload Characteristics.* Scientific applications have diverse workload characteristics. For example, linear algebra operations are mainly based on dense computing, and thus the optimizations focus on maximizing the computing performance of hardware. While graph computing applications are mainly based on sparse computing, thus optimizations focus on improving the memory access performance of programs. A unified programming model requires the ability to identify different workload characteristics and target optimizations.

• *Complexity of the Heterogeneous Architectures.* Because both hardware and software have complex structures and hierarchies, it is difficult for applications to efficiently utilize different hardware. For example, heterogeneous accelerators usually have complex memory hierarchies and unique hardware units for specific computations. These accelerators are designed in different ways to support different data types with different performances and different precision. For a given application, a unified programming model requires the capability to perform architecture-dependent optimizations for different hardware characteristics.

• *Coupling of Application Optimizations and Architecture Characteristics.* To achieve a higher execution efficiency on heterogeneous architectures, developers usually need to design high-level optimization strategies based on the characteristics of target accelerators. For example, algorithms for the same sorting problem on the CPU and the GPU are distinctly designed and optimized. Optimizations such as this sorting problem need to consider both target architectures and applications, making it difficult for the same set of code to execute efficiently on different architectures at the same time. Therefore, decoupling

[①]The Khronosff SYCL™ Working Group. SYCLTM 2020 specification (revision 6). Standard. Khronos Group, 2020. https://registry.khronos.org/SYCL/specs/sycl-2020/pdf/sycl-2020.pdf, Oct. 2022.

application optimization, and reducing the cost of porting a code to different architectures, are essential goals in developing a novel unified programming model.

The high-performance computing community urgently needs a unified programming model focusing on performance portability on various heterogeneous high-performance computers. To solve the above issues, a unified programming model should address the following aspects: 1) enabling the same code to be compiled and executed directly on different architectures without extra effort; 2) supporting performance portability for domain-specific applications, i.e., the same code can achieve similar computational efficiency on different architectures; 3) providing strong support for a series of broad applications and promoting the widespread usage of heterogeneous high-performance computers.

2 Research Goals

Our research goal is to enable performance-portable unified programming with a unified programming model. To achieve this goal, there are four major challenges to solve: 1) programming language, 2) programming abstraction, 3) compilation optimization, and 4) scheduling system.

Domain-Specific Language (*DSL*). To enable various applications with high-performance portability, optimizations should be applied automatically in compilers for applications on various architectures. These optimizations are both related to applications and architectures.

Optimizations are related to the computational patterns of the domain. For example, stencil computing for fluid simulation may require dedicated 2.5-dimensional tiling optimizations, while sparse computing may require format conversion and sorting of sparse data. To enable these optimizations, the programming model needs to apply different optimizations for different domains.

Optimizations are specific to architectures. For example, when implementing the above 2.5-dimensional tiling optimization, the sizes of the tiles are related to the size of the cache or scratch-pad memory. To achieve performance portability among different architectures, the programming model should be able to perform automatic performance optimization for different architectures.

DSL is a promising way to decouple optimizations of the above two aspects. On the one hand, unified programming models can support multiple domain-specific applications by implementing multiple DSLs and thus enable various workloads. On the other hand, DSLs can represent specific computational patterns of their domain. Thus, domain-specific and architecture-specific optimizations can be decoupled and processed by compilers.

Unified Programming Abstraction. The unified programming abstraction is the bridge connecting applications and architectures. After encoding an application by DSL, we can lower the application to the unified programming abstraction. Therefore, we can further map it to different architectures and achieve portability. To design a unified programming abstraction, we need to consider both the representational capability for applications and the optimization ability for architectures to decouple applications and architectures. Finally, we can provide an extensible programming framework to reduce redundant development for various applications and architectures.

Multi-Level Compilation Optimization. Compilation optimization is one of the key modules to achieve performance portability on different heterogeneous computers. However, existing compilation optimization frameworks suffer from insufficient optimizations. Multi-level compilation optimization is an effective way to solve the above problems. By dividing compilation into several stages, specific optimizations can be applied for each stage, thus achieving comprehensive performance optimizations.

Resource-Aware Lightweight Runtime Scheduling System. To improve the execution efficiency of parallel programs in unified programming models, computational resources should be efficiently utilized. Therefore, it is necessary to study a lightweight runtime parallel scheduling system. The runtime scheduling system needs to efficiently analyze the computational dependencies of parallel programs to automatically discover intra- and inter-task parallelism and explore opportunities for concurrent execution. Considering system resource constraints, the runtime system needs to analyze the computational resource requirements of different tasks to achieve efficient parallel task scheduling.

3 Preliminary Design

In this paper, we give a preliminary design for a performance-portable unified programming model. First, as shown in Fig.1, a unified programming ab-

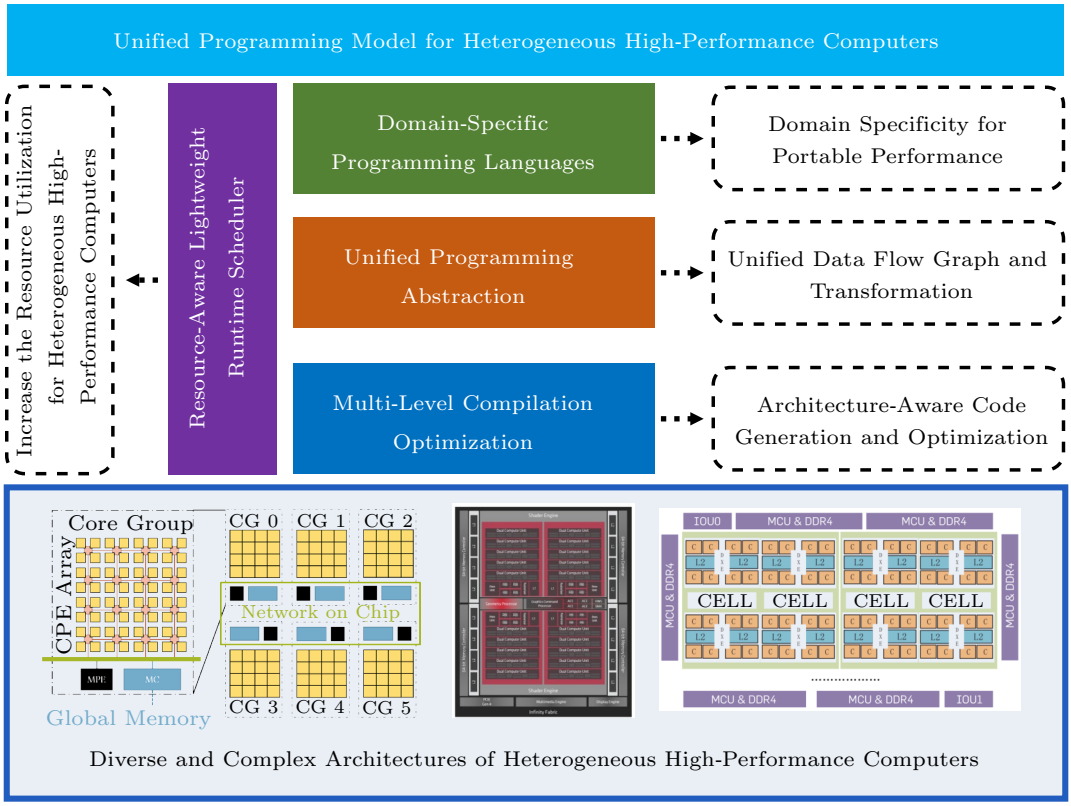

Fig.1. Preliminary design (architecture diagrams are from [12, 13],②).

straction should be provided as the core of unified programming. By abstracting computation, memory access, and concurrent operations, it can clearly describe the behavior of parallel programs on various heterogeneous architectures. This abstraction serves as a bridge between applications and architectures. DSLs should be provided on top of the unified programming abstraction to enable various workloads for unified programming. It should be able to extract domain-specific computational patterns and optimizations, thus allowing applications to be lowered to the unified programming abstraction and optimized by compilers. Moreover, under the unified abstraction, multi-level compilation optimizations are provided to lower the unified abstraction into codes for various architectures. We deploy proper compilation optimizations during compilation to achieve acceptable performance. Finally, to enable efficient execution for parallel programs on heterogeneous supercomputers, there should be a lightweight resource-aware runtime scheduling system to manage the resource according to application requirements and hardware resources.

3.1 Domain-Specific Languages

In order to fully utilize the computing capability of heterogeneous high-performance computers, a significant amount of optimizations, both domain-specific and architecture-specific, need to be applied. To meet this demand, the unified programming model should include DSLs that can be easily extended for different domains. DSLs are designed as a top-level representation of applications, which are programmed by users. Representative studies on high-performance DSLs include GraphIt[14] for graph computing, Halide[15] for image processing, and TVM[16] for deep learning. They express common computational patterns of each domain and enable compilers to fully explore optimization strategies. To achieve portability, compilers should lower DSLs to a shared lower-level representation, and automatically generate transformations on a lower-level representation as domain-specific optimizations. In our design, dataflow graphs can be used as a lower-level representation.

3.2 Unified Programming Abstraction

Unified programming abstraction is the core of the

②https://www.amd.com/system/files/documents/rdna-whitepaper.pdf, Oct. 2022.

performance-portable unified programming model and has the following advantages. 1) The unified programming abstraction decouples applications and architectures by isolating the description for programs and accelerators. Therefore, developers do not need to write a set of optimized code for an application on each accelerator respectively. 2) Through unified programming abstraction, the architecture-independent optimizations can be reused on various accelerators. 3) The programming framework has high extensibility. For a new accelerator, only the lowering from unified abstraction to the target code should be implemented. Developers can reduce the cost of porting significantly. 4) By recording the compilation information on the upper layers, compilers can use these hints for optimizations on the lower layers. Therefore, domain-specific and architecture-specific optimizations can be combined in the compiler without hurting the framework's extensibility.

Unified programming abstraction should be able to describe computation, memory access, and concurrent operations. Since data movement is the main bottleneck of performance and energy consumption of parallel programs on heterogeneous supercomputers, it should be considered as the key of large-scale parallel program optimizations. The dataflow graph is a potential solution for unified abstraction because it can explicitly describe the computation and data movement. Existing studies, such as DaCe[17, 18], implement the abstraction as a stateful dataflow multigraph (SDFG), therefore enabling users to develop applications and port them to achieve high performance.

In our design, a complex application should be described as a data flow graph with a series of transformation operations. There are four major components in this abstraction. 1) *Data Abstraction*. In a unified abstraction, data are described as data objects which contain name, type, size, attributes, and other information. 2) *Computational Abstraction*. Unified abstraction describes computation as multiple tasks. Each task can describe a computational workload which is corresponding to a node in the data flow graph. 3) *Concurrent Operations*. The unified abstraction should provide the description of concurrent operations, which can describe computing and data movement on each level. Thus, the program can easily fit hierarchical heterogeneous architectures. 4) *Transformation Operations*. Since data flow graphs do not contain any architecture information, hardware-specific optimizations should be able to be described through unified abstraction and passed to the compiler. To this end, the unified programming abstraction also provides a series of transformation operations on the data flow graph.

3.3 Multi-Level Compilation Optimizations

The primary purpose of compilation optimization is to explore optimization opportunities for a specific application on a specific architecture. It is key to achieving performance portability on various heterogeneous architectures. Some compilers[19] translate the front-end programming language into an intermediate representation (IR), perform various optimizations on the intermediate representation, and generate binaries for target architectures. However, a single IR cannot provide complete representations for application algorithms and target hardware architectures, which misses numerous optimization opportunities.

Multi-level compilation optimization can effectively solve this problem. By dividing compilation into several stages, multi-level compilation can take different characteristics of applications and hardware architectures into account by building multiple IRs. Thus various optimization opportunities can be explored. MLIR[20] has been a popular project of multi-level compilation in recent years that allows users to quickly design multi-level compilers by defining the IR and transformations between IRs. Several compilers[21, 22] use MLIR to design multi-level compilers for DSLs. In these studies, the IR design is the primary problem of multi-level compilation. Each IR contains specific instructions designed to express the characteristics of the corresponding objects, based on which specific optimizations can be implemented for corresponding objects. Specifically, the multi-level compilation optimization framework should contain IRs at the following levels: generic operation, loop with memory access, and hardware instructions.

3.4 Resource-Aware Lightweight Runtime Scheduling Systems

To efficiently schedule parallel tasks at runtime, a resource-aware lightweight runtime scheduling system is needed. Existing scheduling systems, such as SLURM[23], PBS[24], YARN[25], and Mesos[26], are unaware of the resource requirements of parallel tasks

and thus cannot utilize hardware resources efficiently. Spread-n-Share[27] is able to perceive resources, but its scheduling granularity is too coarse to apply fine-grained task scheduling and it is not able to discover program parallelism automatically. In our design, the scheduling system mainly includes two modules: a computation-dependence-based automated parallelism analyzer, and a resource-aware spread-shared scheduler.

The computation-dependence-based parallelism analyzer can automatically explore parallel opportunities within a task. It can identify all tasks that can execute concurrently within a given time window. Then these tasks are fed into a resource-aware scheduler to decide how to place them on appropriate nodes. The scheduler first analyzes the required types and quantities of resources for each subtask. Then it uses a heuristic scheduling algorithm based on the system's resource constraints to prioritize subtasks with different resource requirements on the same node while guaranteeing that the resources required by all the subtasks on a single node do not exceed the total resources on it.

4 Conclusions

Performance portability is a critical problem in efficiently using high-performance heterogeneous computers. To achieve this goal, we proposed a unified programming model in this paper. This programming model toward this goal has four key techniques. 1) We introduced a domain-specific language to apply domain knowledge for application-level optimizations, and thus we could apply in-depth optimizations for each type of applications. 2) We used a unified programming abstraction to provide a unified view to apply general optimizations for different applications. 3) We used multi-level compilation optimizations to decouple different optimization stages and apply certain optimization strategies at each stage. 4) We used a lightweight runtime scheduling system to automatically discover application parallelism and applied a resource-aware scheduling strategy to improve resource utilization.

References

[1] Dongarra J J, Meuer H W, Strohmaier E. Top500 supercomputer sites. *Supercomputer*, 1997, 13(1): 89–111.

[2] Vazhkudai S S, de Supinski B R, Bland A S *et al.* The design, deployment, and evaluation of the CORAL pre-exascale systems. In *Proc. the 2018 International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2018, pp.661–672. DOI: 10.1109/SC.2018.00055.

[3] Fu H H, Liao J F, Yang J Z *et al.* The Sunway TaihuLight supercomputer: System and applications. *Science China Information Sciences*, 2016, 59(7): 072001. DOI: 10.1007/s11432-016-5588-7.

[4] Fu H H, Liao J F, Xue W *et al.* Refactoring and optimizing the community atmosphere model (CAM) on the Sunway TaihuLight supercomputer. In *Proc. the 2016 International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2016, pp.969–980. DOI: 10.1109/SC.2016.82.

[5] Neale R B, Gettelman A, Park S *et al.* Description of the NCAR community atmosphere model (CAM 5.0). No. NCAR/TN-486+STR, 2010. DOI: 10.5065/wgtk-4g06.

[6] Edwards H C, Trott C R, Sunderland D. Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *Journal of Parallel and Distributed Computing*, 2014, 74(12): 3202–3216. DOI: 10.1016/j.jpdc.2014.07.003.

[7] Trott C R, Lebrun-Grandié D, Arndt D *et al.* Kokkos 3: Programming model extensions for the exascale era. *IEEE Trans. Parallel and Distributed Systems*, 2022, 33(4): 805–817. DOI: 10.1109/TPDS.2021.3097283.

[8] Beckingsale D A, Burmark J, Hornung R *et al.* RAJA: Portable performance for large-scale scientific applications. In *Proc.* the *2019 IEEE/ACM International workshop on Performance, Portability and Productivity in HPC (P3HPC)*, Nov. 2019, pp.71–81. DOI: 10.1109/P3HPC49587.2019.00012.

[9] Reinders J, Ashbaugh B, Brodman J, Kinsner M, Pennycook J, Tian X M. Data Parallel C++: Mastering DPC++ for Programming of Heterogeneous Systems Using C++ and SYCL. Springer Nature, 2021. DOI: 10.1007/978-1-4842-5574-2.

[10] Pennycook S J, Sewall J D, Lee V W. Implications of a metric for performance portability. *Future Generation Computer Systems*, 2019, 92: 947–958. DOI: 10.1016/j.future.2017.08.007.

[11] Lin W C, McIntosh-Smith S. Comparing Julia to performance portable parallel programming models for HPC. In *Proc. the 2021 International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, Nov. 2021, pp.94–105. DOI: 10.1109/PMBS54543.2021.00016.

[12] Ma Z X, He J A, Qiu J Z *et al.* BaGuaLu: Targeting brain scale pretrained models with over 37 million cores. In *Proc. the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Apr. 2022, pp.192–204. DOI: 10.1145/3503221.3508417.

[13] Zhang Y M, Lu K, Chen W G. Processing extreme-scale graphs on China's supercomputers. *Communications of the ACM*, 2021, 64(11): 60–63. DOI: 10.1145/3481614.

[14] Zhang Y, Yang M, Baghdadi R, Kamil S, Shun J. Graphit: A high-performance graph DSL. *Proceedings of*

the ACM on Programming Languages, 2018, 2(OOPSLA): Article No. 121. DOI: 10.1145/3276491.

[15] Ragan-Kelley J, Barnes C, Adams A, Paris S, Durand F, Amarasinghe S. Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In *Proc. the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, June 2013, pp.519–530. DOI: 10.1145/2499370.2462176.

[16] Chen T Q, Moreau T, Jiang Z H *et al.* TVM: An automated end-to-end optimizing compiler for deep learning. In *Proc. the 13th USENIX Conference on Operating Systems Design and Implementation*, Oct. 2018, pp.579–594.

[17] Ben-Nun T, de Fine Licht J, Ziogas A N, Schneider T, Hoefler T. Stateful dataflow multigraphs: A data-centric model for performance portability on heterogeneous architectures. In *Proc. the 2019 International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2019, Article No. 81. DOI: 10.1145/3295500.3356173.

[18] Ziogas A N, Ben-Nun T, Fernández G I, Schneider T, Luisier M, Hoefler T. A data-centric approach to extreme-scale *ab initio* dissipative quantum transport simulations. In *Proc. the 2019 International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2019, Article No. 1. DOI: 10.1145/3295500.3357156.

[19] Lattner C, Adve V. LLVM: A compilation framework for lifelong program analysis & transformation. In *Proc. the 2004 International Symposium on Code Generation and Optimization*, Mar. 2004, pp.75–86. DOI: 10.1109/CGO.2004.1281665.

[20] Lattner C, Amini M, Bondhugula U, Cohen A, Davis A, Pienaar J, Riddle R, Shpeisman T, Vasilache N, Zinenko O. MLIR: A compiler infrastructure for the end of Moore's law. arXiv: 2002.11054, 2020. https://arxiv.org/abs/2002.11054, Mar. 2020.

[21] Gysi T, Müller C, Zinenko O, Herhut S, Davis E, Wicky T, Fuhrer O, Hoefler T, Grosser T. Domain-specific multi-level IR rewriting for GPU: The open earth compiler for GPU-accelerated climate simulation. *ACM Transactions on Architecture and Code Optimization*, 2021, 18(4): Article No. 51. DOI: 10.1145/3469030.

[22] McCaskey A, Nguyen T. A MLIR dialect for quantum assembly languages. In *Proc. the 2021 IEEE International Conference on Quantum Computing and Engineering*, Oct. 2021, pp.255–264. DOI: 10.1109/QCE52317.2021.00043.

[23] Yoo A B, Jette M A, Grondona M. SLURM: Simple Linux utility for resource management. In *Proc. the 9th International Workshop on Job Scheduling Strategies for Parallel Processing*, Jun. 2003, pp.44–60. DOI: 10.1007/10968987_3.

[24] Bode B, Halstead D M, Kendall R *et al.* The portable batch scheduler and the Maui scheduler on Linux clusters. In *Proc. the 4th Annual Linux Showcase & Conference*, Oct. 2000. DOI: 10.5555/1268379.1268406.

[25] Vavilapalli V K, Murthy A C, Douglas C *et al.* Apache Hadoop YARN: Yet another resource negotiator. In *Proc. the 4th Annual Symposium on Cloud Computing*, Oct. 2013, Article No. 5. DOI: 10.1145/2523616.2523633.

[26] Hindman B, Konwinski A, Zaharia M *et al.* Mesos: A platform for fine-grained resource sharing in the data center. In *Proc. the 8th USENIX Conference on Networked Systems Design and Implementation*, Mar. 2011, pp.295–308.

[27] Tang X C, Wang H J, Ma X S *et al.* Spread-n-Share: Improving application performance and cluster throughput with resource-aware job placement. In *Proc. the International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2019, Article No. 12. DOI: 10.1145/3295500.3356152.

Zi-Xuan Ma received his B.S. degree in computer science from Tsinghua University, Beijing, in 2019. He is a Ph.D. candidate in the Department of Computer Science and Technology of Tsinghua University, Beijing. His research interests include high-performance computing and AI Compiler.

Yu-Yang Jin received his B.S. degree in computer science from Beijing Institute of Technology, Beijing, in 2017, and his Ph.D. degree in computer science from Tsinghua University, Beijing, in 2022. He is a postdoctoral researcher in the Department of Computer Science and Technology of Tsinghua University, Beijing. His research interests include performance analysis and optimization for parallel applications.

Shi-Zhi Tang received his B.S. degree in computer science from Tsinghua University, Beijing, in 2019. He is a Ph.D. candidate in the Department of Computer Science and Technology of Tsinghua University, Beijing. His research interests include programming models and optimizing compilers for performance portability and performance productivity.

Hao-Jie Wang received his B.S. degree in engineering mechanics and aerospace engineering, and Ph.D. degree in computer science from Tsinghua University, Beijing, in 2015 and 2021, respectively. He is a postdoctoral researcher in the Department of Computer Science and Technology of Tsinghua University, Beijing. His research interests include AI compiler, program analysis, and high-performance computing.

Wei-Cheng Xue received his B.S. degree in mechanical engineering from Huazhong University of Science and Technology, Wuhan, in 2012, and his Ph.D. degree in aerospace engineering from Virginia Tech, Washington, in 2020. He is a postdoctoral researcher in the Department of Computer Science and Technology of Tsinghua University, Beijing. His research interests include physics informed neural network and high-performance computing.

Ji-Dong Zhai received his B.S. degree in computer science from the University of Electronic Science and Technology of China, Chengdu, in 2003, and his Ph.D. degree in computer science from Tsinghua University, Beijing, in 2010. He is a tenured associate professor in the Department of Computer Science and Technology of Tsinghua University, Beijing. His research interests include performance evaluation for high-performance computers, performance analysis, and modeling of parallel applications.

Wei-Min Zheng received his B.S. and M.S. degrees from Tsinghua University, Beijing, in 1970 and 1982, respectively. He is a professor in the Department of Computer Science and Technology of Tsinghua University, Beijing. He is an academician of the Chinese Academy of Engineering. His research interests include distributed computing, compiler techniques, and network storage.