Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Perspective:

A vision of post-exascale programming*

Ji-dong ZHAI‡, Wen-guang CHEN

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

E-mail: zhaijidong@tsinghua.edu.cn; cwg@tsinghua.edu.cn

Received July 21, 2018; Revision accepted Sept. 9, 2018; Crosschecked Oct. 15, 2018

Abstract: Exascale systems have been under development for quite some time and will be available for use in a few years. It is time to think about future post-exascale systems. There are many main challenges with regard to future post-exascale systems, such as processor architecture, programming, storage, and interconnect. In this study, we discuss three significant programming challenges for future post-exascale systems: heterogeneity, parallelism, and fault tolerance. Based on our experience of programming on current large-scale systems, we propose several potential solutions for these challenges. Nevertheless, more research efforts are needed to solve these problems.

Key words: Computing model; Fault-tolerance; Heterogeneous; Parallelism; Post-exascale
https://doi.org/10.1631/FITEE.1800442 **CLC number:** TP311

1 Introduction

Currently, peta-scale supercomputers have been built and used for years, and exascale ones will be available for use in a few years. With the development of technology, post-exascale era will finally come. It is time to think about the challenges for future post-exascale systems. In this paper, we discuss the main challenges of post-exascale systems and some potential solutions for these challenges.

We first discuss the three main challenges faced in future post-exascale programming models: (1) supercomputers are becoming more heterogeneous; (2) a large amount of parallelism in applications needs to be exploited to fully use a large-scale supercomputer efficiently; (3) fault tolerance programming will be inevitable in a future post-exascale system. We elaborate each point below.

Heterogeneous architectures have shown increasing popularity for usage in large-scale supercomputers due to the considerations of performance, energy efficiency, density, and costs (Vetter et al., 2011). In the sixteenth top-500 supercomputer list (https://www.top500.org/lists/2018/06/), the top five systems are all heterogeneous. Each node within these systems comprises both general central processing units (CPUs) and heterogeneous co-processors. The CPU is responsible for the control flow of applications, whereas computation-intensive sections are assigned to high-performance co-processors. Even though the hardware provides high peak performance, the programming complexity of heterogeneous applications makes it tricky to fully use the power of systems. Both the implementation and optimization of applications on such systems are difficult. Hence, a productive heterogeneity-oriented programming model is crucial to address post-exascale programming challenges. Normally, heterogeneous programming is responsible mainly for intra-node tasks owing to the architectural features. The workload of co-processors is usually offloaded by CPUs. Co-processors in different nodes usually do not communicate with each other.

The programming challenges on heterogeneous platforms relate mainly to the communication

‡ Corresponding author
* Project supported by the National Key Technology R&D Program of China (No. 2016YFB0200100)
ORCID: Ji-dong ZHAI, http://orcid.org/0000-0002-7656-6428

management of intra-node devices and utilization of complex hardware features of co-processors.

In a heterogeneous system, the CPU and the co-processors have discrete memory systems. Data should be moved from the CPU to the co-processors before it can be processed by cores on co-processors, and the result should be copied back to CPU in the end. The management of data movement between different devices is tedious and error-prone, especially for applications with complicated control flow and data structures.

Co-processors on supercomputers always have a delicate and complicated thread, memory, and intra-device communication hierarchy. We take a graphics processing unit (GPU), which is the most popular co-processor in modern supercomputers, as an example to illustrate this point. In a GPU, threads are organized as warps, warps are organized as thread blocks, and thread blocks are organized as grids. Data can be stored in the global memory and can also be manually cached in shared memory. Meanwhile, there are constant memory, register files, and various caches in the memory hierarchy. Current GPU programming models provide intra-warp register shuffle, intra-block shared-memory-based communications and barriers, and global-memory-based communications. The most recent CUDA (https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html), which is the most popular programming model for NVIDIA GPUs, introduces the 'cooperate group', which makes the thread and communication hierarchy more complex. The utilization of all the above characteristics is always a burden for program developers.

Post-exascale programming will require exploitation of more fine-grained parallelism from applications since there will be more computing nodes or processor cores on a post-exascale supercomputer. For instance, one of the current top supercomputers, Sunway TaihuLight, has more than 10 million processor cores, which means that the future post-exascale supercomputer will have more than 100 million cores. To efficiently execute a program on such a large-scale system using all processor cores, programmers should map a given problem onto over 100 million cores, which is a non-trivial problem for programmers.

Let us look at an example in current supercomputers. Lin et al. (2017) ported the breadth-first search (BFS) onto Sunway TaihuLight (Fu et al., 2016). As Sunway TaihuLight has 40 960 computing nodes and 10.6 million co-processor cores, it requires very fine-grained parallelism granularity to run a program on the whole system. To discover the potential parallelism, researchers propose several novel techniques, including pipelined module mapping, contention-free data shuffling, and group-based message batching, which requires considerable effort. Due to the cap on per-core frequency, a post-exascale supercomputer would have more than 10× nodes than Sunway TaihuLight, which will make it even more challenging to discover any potential parallelism and will burden the programmers.

Fault tolerance has been an important issue for decades and is still an open problem for current peta-scale systems and upcoming exascale systems (Cappello, 2009). As system parallelism keeps increasing, it is highly possible that a post-exascale system will have an even shorter mean time between failures (MTBF) than current high-performance computing (HPC) systems, which is only hours (Schroeder and Gibson, 2010). Therefore, a large-scale program that attempts to make full use of a post-exascale system must consider how to compute in a faulty environment. It will be challenging but inevitable to integrate the specific functionality of failure avoidance, fault detection, fault analysis, and failure recovery into system tools, programming framework, third-party libraries, and applications.

2 Traditional programming model

Programming models can help users conveniently express algorithms and their compositions. As the most widely used programming model, message passing interface (MPI) works well on many modern systems. MPI supports multiple languages such as C, C++, Fortran, and both point-to-point communication and collective communication are supported. Researchers have already started working on exascale MPI (Gropp, 2009; Balaji et al., 2013; Bahmani and Mueller, 2014), but there are still many challenges that need to be addressed such as the increased scale, performance characteristics, and evolving architectural features expected in exascale systems, as well as the capabilities and requirements of applications targeted at these systems.

OpenMP (Dagum and Menon, 1998) is another

widely used programming model that handles shared memory systems. As implementing a shared memory system on a post-exascale supercomputer is not practical, it is not possible to use OpenMP across the whole system. However, using OpenMP along with MPI is a commonly used method in modern systems. OpenMP is used for parallelism within a node while using MPI to handle communication between nodes. MPI plus OpenMP would be still a practical way of writing a program on a post-exascale supercomputer.

CUDA (https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html) is the most commonly used heterogeneous programming model. It provides a set of interfaces to address hardware and software complexities on GPUs. Developers can spend considerable effort to configure programs to better use all hardware features. The problem with CUDA is that even though it adopts a C-based grammar to ease programming, it still requires too much effort to develop and tune a sophisticated program on such a complex platform (a single node with GPUs). This is not to mention that the post-exascale system will comprise millions of such nodes. Another problem with CUDA is that it can run only on NVIDIA GPUs, even though portability is not a critical concern in the HPC area.

OpenCL (Stone et al., 2010) is another programming model for cross-platform, parallel programming of diverse processors. Like CUDA, its syntax is also C-based. It provides uniform interfaces that match the characteristics of a variety of processors (Munshi, 2009). The extensive support for different devices and portability is the main advantage of OpenCL. However, it introduces even more programming complexities than CUDA, which is one of the main drawbacks that make it less popular.

OpenACC (Lee and Vetter, 2012) is a directive-based parallel programming model supported by a wide variety of heterogeneous HPC hardware platforms. It hides the complexity of architectures with a limited set of directives. With little programming effort, developers can realize a portable parallel program on heterogeneous platforms. However, OpenACC has less flexibility compared with CUDA and OpenCL. It is hard for programmers to take full advantage of the hardware with OpenACC. OpenMP is also used on some co-processors (Jeffers and Reinders, 2013). It has similar advantages and disadvantages to OpenACC.

A programming model that provides duplication is a classic approach toward failure avoidance. After running multiple replicas of a program, the majority (or at least one) of those replicas, can hopefully obtain the expected result. Nevertheless, fault tolerance at a low cost is always welcome and desired. Instead of full duplication, partial redundancy provides fault tolerance with fewer resources. Algorithm-based fault tolerance (ABFT) is a technique that redesigns an algorithm and makes it naturally fault-tolerant (Chen, 2013). For matrix operations, additional rows and columns that contain redundant information can help produce a correct result under certain types of system errors (Huang and Abraham, 1984).

Once a failure is encountered, there are two approaches for recovery: roll-back recovery and forward recovery. Roll-back recovery means going back to somewhere before the failure and then redoing the work from that point onward. Checkpoint-and-restart is a classic and useful method for roll-back recovery, but its overhead of flushing bulk of data is worth worrying about (Bland et al., 2012; Tang et al., 2017). A forward recovery approach does not stop or go back to a previous snapshot (Bland et al., 2013). Instead, the program after failure will automatically evolve to a fault-free state. Forward recovery can be achieved by restarting new tasks to catch up with other processes or by re-partitioning the workload among the remaining processes. However, most programming frameworks or libraries in HPC systems do not support forward recovery (or with a high overhead (Bouteiller et al., 2003)). Thus, developers cannot build a forward-recoverable application.

3 Post-exascale programming model

In this section, we provide some potential solutions to address the main challenges faced in post-exascale supercomputers as listed in Section 1.

3.1 Heterogeneity

The programming model on a post-exascale platform should provide interfaces to represent an enough number of hardware features, which is essential for developers to fully use the platform. Meanwhile, it should be efficient enough for programming. One practical and effective approach is to develop a programming model that contains both interfaces

for complex features (called low-level interfaces) as CUDA and OpenCL do, and directives for essential features (called high-level interfaces) as OpenACC does. For complex scenarios, the low-level interfaces can be used to make full use of hardware features, whereas for simple scenarios, the high-level interfaces can be used for efficient development.

The low-level interfaces express enough architecture-specific features and can be used for fine-tuning applications. It will provide a mechanism for implicit data movement between devices as the unified memory technique does in CUDA 6.0 (https://developer.nvidia.com/cuda-toolkit-60). Moreover, the syntax for the low-level interfaces will be C-based, and C++ features will be supported.

The high-level interfaces comprise a set of directives. The directives can be categorized into two types: directives for managing parallelism execution and those for managing data store and movement. The data region manages data, whereas the computing region handles parallelism execution. The inter-device data movement of content in the data region can be accomplished automatically. The loops and code sections in the computing region will be compiled into co-processor kernels that can be run in parallel on co-processors.

The last requirement for the programming model is that it should have a set of powerful libraries for all types of special functions that can run on co-processors, like cuBLAS for CUDA (https://docs.nvidia.com/cuda/cublas). This is not a part of the programming model itself, but it is essential for the efficient development of various applications.

3.2 Parallelism

Most current supercomputers use 'MPI+X' (OpenMP, OpenACC, among others) programming models to describe programs' parallelism, including one of the top supercomputers Sunway TaihuLight. It would still be a practical manner for a future post-exascale supercomputer. Researchers have already started working on exascale MPI (Balaji et al., 2013), but there are still many challenges to be addressed, such as the increasing scale, performance characteristics, and evolving architectural features expected in exascale systems, as well as the capabilities and requirements of applications targeted at these systems. As for X that runs only on a single node, there would be little difference on a post-exascale system as the single node would not change a lot.

Although the programming model would not considerably change on a post-exascale computer, programming would become a big challenge on such a large system. Programmers have to spend much time on programming on current large-scale systems, especially for programs that will be run on more than 100 million processor cores (Lin et al., 2017). We think the emerging domain-specific language (DSL) is a potential good solution to address this challenge. The Halide language is an example of DSL that makes it much easier to write high-performance image processing code on modern machines (Ragan-Kelley and Adams, 2012; Ragan-Kelley et al., 2013), which gives us a guide on the programming model on a post-exascale computer. On a post-exascale system, we may provide programmers with various DSLs for each specific domain so that they can write programs with less effort and higher performance.

Compared with DSL in the HPC area, programming frameworks for big-data applications seem to be more friendly. MapReduce (Dean and Ghemawat, 2008), Spark (Xin et al., 2013), and Gemini (Zhu et al., 2016) provide tidy but powerful interfaces for developers in certain domains. Developing applications on the top of those frameworks is generally easier than using MPI, CUDA, OpenCL, or OpenACC. However, a traditional HPC programmer focuses more on performance rather than productivity. Consequently, a programming model should expose most performance-related details to developers instead of encapsulating everything. Nevertheless, as supercomputers are becoming faster and faster, they should have abundant or even redundant computational resources to support a high-productivity framework, at the price of performance. Simultaneously, a performance-first programming model is always necessary for expert HPC users and high-level framework developers.

There are also some aggressive approaches for writing codes, such as using artificial intelligence (AI) programming or auto tuners. Although machine learning and AI have seen a great leap in recent years, it is still too optimistic to expect an AI HPC developer before it could build a 'Hello World' program in a practical manner.

3.3 Fault tolerance

Hardware errors will become more frequent as a post-exascale system will have more CPUs, memory, discs, switches, and air conditioners than those present today. Consequently, we need to consider as many hardware components as possible for fault tolerance. In the past decades, hardware vendors have made significant efforts to tolerate certain fail-stop errors but lack the tolerance for fail-slow errors. Users of Sunway Taihulight and Tianhe-2 (two supercomputers in China) have reported errors that make the system run slower than usual. If a memory chip has some broken bits, it can still produce a correct result, but at a much reduced speed. Even worse, as the scale increases, 2-bit flipping can occasionally happen but cannot be handled by current devices. In this case, we will probably obtain an incorrect result, without noticing this error at all. In the post-exascale era, duplication will still be an important approach toward failure avoidance. Some may claim that duplication is a waste of resources, but the final goal of any resource is to get the job done. We can use more resources for higher performance, as well as for higher reliability. For ABFT, it will alleviate the burden on developers if most math libraries like Intel MKL and cuBLAS can provide fault-tolerant versions.

On a post-exascale system, novel techniques are required to efficiently detect faults, especially performance faults. To detect performance-related errors, the detection tool should be aware of the system's or program's performance (Tang et al., 2018). Such an error can be detected when we notice that the performance is abnormal. A post-exascale programming model should not only provide the application programming interfaces (APIs) for functionality but also provide some APIs for developers to understand their applications' performance. For silent incorrect execution errors, it is hard or even impossible to detect them from the system's perspective as the underlying system knows nothing about the application's expectation. Therefore, developers should be responsible for the result verification of their applications.

Owing to the large scale of a post-exascale system, new algorithms (Yao et al., 2012) or new devices are necessary for efficient checkpoints. In a post-exascale system, nodes may have local fast storage, such as solid state drives (SSDs) or non-volatile memory (NVM) (Dong et al., 2009). We need a new checkpoint strategy to use those new devices to minimize the performance overhead. To enable forward recovery, we must ensure that the programming framework itself is recoverable; otherwise, it will build a non-stop application on the top of it.

Fault tolerance programming on a post-exascale system is more complex than before, so it cannot be solved solely by the system administrator or the application developer. A co-design from the underlying system, to the programming framework, and to the final application is needed. On the system side, it should monitor the hardware and software status and provide the data through some system calls or libraries. Furthermore, the underlying system tools themselves should be fault-tolerant. On the application side, it should take care of the correctness of its result. Using necessary verification codes in appropriate places will help avoid disasters. We also need an interface between applications and system tools for fault tolerance programming, just as with MPI for communication. The interface should specify what is being provided by the system and what should be done inside the applications.

4 Conclusions and future work

In this paper, three significant challenges of programming have been discussed for a post-exascale system: heterogeneity, parallelism, and fault tolerance. The solutions for a post-exascale system must be aware of the large scale and be specific to architectural features. Considering the complexity and diversity of underlying architectures, we infer that programming frameworks and domain-specific languages will be necessary for future post-exascale programming.

References

Bahmani A, Mueller F, 2014. Scalable performance analysis of exascale MPI programs through signature-based clustering algorithms. 28th ACM Int Conf on Supercomputing, p.155-164. https://doi.org/10.1145/2597652.2597676

Balaji P, Snir M, Amer A, et al., 2013. Exascale MPI. https://www.exascaleproject.org/project/exascale-mpi/ [Accessed on Sept. 10, 2018].

Bland W, Du P, Bouteiller A, et al., 2012. A checkpoint-on-failure protocol for algorithm-based recovery in stan-

dard MPI. European Conf on Parallel Processing, p.477-488. https://doi.org/10.1007/978-3-642-32820-6_48

Bland W, Bouteiller A, Herault T, et al., 2013. Post-failure recovery of MPI communication capability: design and rationale. *Int J High Perform Comput Appl*, 27(3):244-254. https://doi.org/10.1177/1094342013488238

Bouteiller A, Cappello F, Herault T, et al., 2003. MPICH-V2: a fault tolerant MPI for volatile nodes based on pessimistic sender based message logging. ACM/IEEE Conf on Supercomputing, p.1-17. https://doi.org/10.1145/1048935.1050176

Cappello F, 2009. Fault tolerance in petascale/exascale systems: current knowledge, challenges, and research opportunities. *Int J High Perform Comput Appl*, 23(3):212-226. https://doi.org/10.1177/1094342009106189

Chen Z, 2013. Online-ABFT: an online algorithm based fault tolerance scheme for soft error detection in iterative methods. *ACM SIGPLAN Not*, 48(8):167-176. https://doi.org/10.1145/2442516.2442533

Dagum L, Menon R, 1998. OpenMP: an industry standard API for shared-memory programming. *IEEE Comput Sci Eng*, 5(1):46-55. https://doi.org/10.1109/99.660313

Dean J, Ghemawat S, 2008. MapReduce: simplified data processing on large clusters. *Commun ACM*, 51(1):107-113. https://doi.org/10.1145/1327452.1327492

Dong X, Muralimanohar N, Jouppi N, et al., 2009. Leveraging 3D PCRAM technologies to reduce checkpoint overhead for future exascale systems. Int Conf on High Performance Computing Networking, Storage, and Analysis, p.1-12. https://doi.org/10.1145/1654059.1654117

Fu H, Liao J, Yang J, et al., 2016. The Sunway Taihulight supercomputer: system and applications. *Sci Chin Inform Sci*, 59(7):072001. https://doi.org/10.1007/s11432-016-5588-7

Gropp W, 2009. MPI at exascale: challenges for data structures and algorithms. In: Ropo M, Westerholm J, Dongarra J (Eds.), Recent Advances in Parallel Virtual Machine and Message Passing Interface. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-03770-2_3

Huang KH, Abraham JA, 1984. Algorithm-based fault tolerance for matrix operations. *IEEE Trans Comput*, C-33(6):518-528. https://doi.org/10.1109/TC.1984.1676475

Jeffers J, Reinders J, 2013. Intel Xeon Phi Coprocessor High Performance Programming. Morgan Kaufmann Publishers Inc., San Francisco, USA.

Lee S, Vetter JS, 2012. Early evaluation of directive-based GPU programming models for productive exascale computing. Int Conf on High Performance Computing, Networking, Storage, and Analysis, p.1-11. https://doi.org/10.1109/SC.2012.51

Lin H, Tang X, Yu B, et al., 2017. Scalable graph traversal on Sunway Taihulight with ten million cores. Int Parallel and Distributed Processing Symp, p.635-645. https://doi.org/10.1109/IPDPS.2017.53

Munshi A, 2009. The OpenCL specification. 21st IEEE Hot Chips Symp, p.1-314. https://doi.org/10.1109/HOTCHIPS.2009.7478342

Ragan-Kelley J, Adams A, 2012. Halide. http://halide-lang.org [Accessed on Sept. 10, 2018].

Ragan-Kelley J, Barnes C, Adams A, et al., 2013. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. *ACM SIGPLAN Not*, 48(6):519-530. https://doi.org/10.1145/2499370.2462176

Schroeder B, Gibson G, 2010. A large-scale study of failures in high-performance computing systems. *IEEE Trans Depend Sec Comput*, 7(4):337-350. https://doi.org/10.1109/TDSC.2009.4

Stone JE, Gohara D, Shi G, 2010. OpenCL: a parallel programming standard for heterogeneous computing systems. *Comput Sci Eng*, 12(3):66-73. https://doi.org/10.1109/MCSE.2010.69

Tang X, Zhai J, Yu B, et al., 2017. Self-checkpoint: an in-memory checkpoint method using less space and its practice on fault-tolerant HPL. 22nd ACM SIGPLAN Symp on Principles and Practice of Parallel Programming, p.401-413. https://doi.org/10.1145/3155284.3018745

Tang X, Zhai J, Qian X, et al., 2018. VSensor: leveraging fixed-workload snippets of programs for performance variance detection. 23rd ACM SIGPLAN Symp on Principles and Practice of Parallel Programming, p.124-136. https://doi.org/10.1145/3200691.3178497

Vetter JS, Glassbrook R, Dongarra J, et al., 2011. Keeneland: bringing heterogeneous GPU computing to the computational science community. *Comput Sci Eng*, 13(5):90-95. https://doi.org/10.1109/MCSE.2011.83

Xin RS, Gonzalez JE, Franklin MJ, et al., 2013. GraphX: a resilient distributed graph system on Spark. 1st Int Workshop on Graph Data Management Experiences and Systems, p.1-6. https://doi.org/10.1145/2484425.2484427

Yao E, Wang R, Chen M, et al., 2012. A case study of designing efficient algorithm-based fault tolerant application for exascale parallelism. 26th Int Parallel and Distributed Processing Symp, p.438-448. https://doi.org/10.1109/IPDPS.2012.48

Zhu X, Chen W, Zheng W, et al., 2016. Gemini: a computation-centric distributed graph processing system. 12th USENIX Symp on Operating Systems Design and Implementation, p.301-316.