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:

Extreme-scale parallel computing: bottlenecks and strategies*

Ze-yao MO[1,2]

[1]*CAEP Software Center for High Performance Numerical Simulation, Beijing 100088, China*

[2]*Institute of Applied Physics and Computational Mathematics, Beijing 100094, China*

E-mail: zeyao_mo@iapcm.ac.cn

Received July 7, 2018; Revision accepted Sept. 14, 2018; Crosschecked Oct. 15, 2018

Abstract: Extreme-scale numerical simulations seriously demand extreme parallel computing capabilities. To address the challenges of these capabilities toward exascale, we systematically analyze the major bottlenecks of parallel computing research from three perspectives: computational scale, computing efficiency, and programming productivity. For these bottlenecks, we propose a series of urgent key issues and coping strategies. This study will be useful in synchronizing development between the numerical computing capability and supercomputer peak performance.

Key words: Extreme scale; Numerical simulation; Parallel computing; Supercomputers
https://doi.org/10.1631/FITEE.1800421 **CLC number:** TP311

1 Introduction

Extreme-scale numerical simulation is the process of developing numerical software using the software on supercomputers to reproduce and advance the governing laws of the objective world under investigation, acquiring knowledge from the simulation results, making scientific discoveries, and developing engineering designs based on this knowledge (Reed et al., 2005). Parallel computing research bridges numerical simulation and computer architectures, and supports the high accuracy of numerical simulations by continuously improving the computing capabilities of numerical software. Computing capability is the numerical computation capability obtained by numerical simulations on supercomputers. It includes mainly three ingredients: computational scale, computing efficiency, and programming productivity (Dongarra et al., 2003; Amarasinghe et al., 2011). Computation scale reflects the degrees of freedom required by the numerical simulations that are used to reproduce the governing laws of the objective world under investigation. Computing efficiency is the floating-point efficiency of these simulations on supercomputers. Programming productivity is the amount of investment required to develop numerical applications with this high computing capability. Computing capability includes intelligence, time, and costs, and covers the full lifecycle of numerical software, including development, maintenance, and technical support.

With the advancement of numerical simulation into the era of coupled multiple physics, the governing laws of the objective world require serious quantitative simulations that can predict experimental observations and deliver high-impact policy and decision support. New frontiers of multi-physics simulations need to not only combine multiple application models developed by different teams, but also incorporate data analytics, design and optimization, and uncertainty quantification on the top of traditional

* Project supported by the National Natural Science Foundation of China (No. 91430218) and the National Key Technology R&D Program of China (Nos. 2016YFB0201300 and 2017YFB0202103)
ORCID: Ze-yao MO, http://orcid.org/0000-0003-3280-5682
© Zhejiang University and Springer-Verlag GmbH Germany, part of Springer Nature 2018

forward models (Ashby et al., 2011; Keyes et al., 2013; Johansen et al., 2014). All these requirements ask the peak performance of supercomputers to increase from peta-flops to exa-flops (Lucas et al., 2014) in many significant applications.

With the development of numerical simulation for coupled multiple physics and the increases of supercomputer peak performance, parallel computing research has gradually exhibited three major bottlenecks: limited computational scale, inadequate computing efficiency, and low programming productivity. The computational scale bottleneck stems from the extreme complexity of numerical algorithms, which increases super-linearly with increase in the number of processor cores. The computing efficiency bottleneck results from the rapid evolution of parallel computer architectures, which breaks the mapping strategies of parallel algorithms to computer architectures all the time. The programming productivity bottleneck originates from the increasing complexity of parallel programming, which increases the intelligence, time, and resource requirements of parallel algorithm implementations, and continuously weakens the extensibility, portability, and inheritability of numerical software. These three bottlenecks collectively inhibit the acquisition of computing capability of extreme-scale numerical applications, and keep widening the gap between software development productivity and supercomputer advancements (Johansen et al., 2014).

In this study, we address the above problems in exascale numerical simulation based on our parallel computing research experience in a series of major fundamental research domains, including weapon physics, fusion energy, fission energy, and equipment manufacturing. We systematically analyze the above bottlenecks and propose a series of research-demanding key technology issues and effective coping strategies. This study will be useful in synchronizing improvements in extreme-scale numerical computing capabilities and supercomputer peak performance.

2 Bottlenecks and strategies for computational scale

The governing law of the objective world is often described by mathematical physics equations. After the equation is discretized, the discrete system needs to be solved numerically on supercomputers. The algorithm used is a so-called numerical algorithm. Computational complexity is one key indicator of numerical algorithms. Generally, an algorithm with linear complexity is optimal and can be represented by $t_{N,P} = O(N\log(NP))$, where $t_{N,P}$ is the numerical simulation time, N is the number of degrees of freedom on each processor, P is the number of processor cores, and $N \times P$ is the scale of calculation. As the number of processor cores P increases, $t_{N,P}$ increases only logarithmically, and the computation is scalable with the growth of P.

For time-dependent mathematical physics equations, at each time step, the explicit time-stepping schemes are often of linear complexity. However, for implicit time-stepping schemes, the resultant discrete systems are often expressed as sparse linear or non-linear systems, and need to be solved by either direct methods or iterative methods (Dongarra et al., 2003). Limited by the solver's algorithmic complexity, $t_{N,P}$ may evolve to $O(N^{\alpha}P^{\alpha-1}\log(NP))$, where $\alpha > 1$. The power index α restricts the linear growth of the computational scale with P. For example, if $\alpha = 2$, then $t_{N,P}$ grows linearly with P. That is, when P doubles, $t_{N,P}$ also doubles, and the computation does not scale with P. From another point of view, given fixed $t_{N,P}$, the computational scale would not increase with P. When the peak performance of supercomputers increases from peta-flops to exa-flops, P will increase by about 100 times. Only when the numerical algorithms maintain linear complexity, can computation scale linearly with P.

In the past two decades, many studies have focused on linear complexity algorithms for implicit discretization schemes. These types of algorithms are also commonly referred to as fast algorithms and include the domain decomposition method (DDM) (Dolean et al., 2015), parallel algebra multigrid method (AMG) (Saad and Darwish, 2009), parallel fast multipole method (FMM) (Engheta et al., 1992), fast eigensolvers (Campos and Roman, 2012), fast Fourier transform, and other discrete transform algorithms (Cooley and Tukey, 1965). These algorithms have been implemented in many open source libraries, including PETSc (Balay et al., 1997), Hypre (Falgout and Yang, 2002), Trillinos (Heroux et al., 2005), and SLEPc (Hernandez et al., 2005). However, with the increasing complexity of physical phenomena, the application characteristics need to be fully considered by these algorithms to maintain

linear complexity.

On one hand, when the computational scale increases, the resolution of numerical simulation may increase, and the physical characteristics such as strong discontinuity and strong nonlinearity can have an increasing impact on the numerical properties of the matrices of the discrete systems. Numerical algorithms need to be designed with these characteristics in mind. For example, for numerical simulation of radiation hydrodynamics, when the computational scale increases, the grid resolution increases, and the strong non-linearity of electron, ion, and photon temperatures will increase the emissivity by more than 10 orders of magnitude over a number of grid cells. The stiffness of the discrete system is greatly enhanced, and the parallel algebraic multigrid algorithm needs to perform local coarsening preprocessing to maintain linear complexity (Xu and Mo, 2017). When solving frequency-domain electromagnetic equations, the parallel fast multipole algorithm needs to be customized for the electromagnetic properties of the specific material by local multipole approximations (Cao et al., 2011).

On the other hand, when the numerical simulation paces towards the coupled multi-physics era, computational complexity is increasingly driven by features of the specific application, especially for overall implicit discretization schemes (Keyes et al., 2013). A fundamental scientific problem is as follows: given that the algorithm for the discrete systems of a single physical field has linear complexity, does that of a multi-physics coupled simulation also bear linear complexity? There are currently few theoretical results for this question. However, in case of operator-splitting-based implicit discretizations, the answer is usually no. If globally coupled implicit discretization schemes are used, the matrices of the discrete system are usually asymmetric, and sometimes not diagonally dominant, and require specific algorithms and pre-conditioners to achieve linear complexity. For example, when solving the coupled radiation transport equations in radiation hydrodynamics, the multi-group diffusion equations and multi-group transport equations are usually solved by operator-splitting schemes (Keyes et al., 2013). When solving the 'force-heat-contact' coupled structural mechanics problem, the globally coupled implicit discretization produces a sparse linear system that needs a specific shear-force-aware pre-conditioner to converge (Tian et al., 2018). In the 'magnetic-thermal-force' coupled integrated circuit packaging simulation, the magnetic field, heat transfer, and mechanical response are usually solved using the operator-splitting scheme (Zhao et al., 2014). Similar situations are common in multi-physics coupling applications such as fluid-solid coupling, fusion energy, fission reactor physics, material fracture and damage, surface chemistry, climate change, geodynamics, and accelerator physics (Keyes et al., 2013).

It can be concluded that when the computational scale grows, the application characteristics such as strong discontinuity, strong nonlinearity, and multi-physics coupling will increase the computational complexity of numerical algorithms. Thus, it is urgent to carry out systematic and focused research (Keyes et al., 2013). Two research branches should be considered. One is the common algorithm frameworks, and the other is the application-feature-driven pre-conditioners. At present, algorithm frameworks are fruitful, and need only to be partially improved for exascale computing. Mature frameworks include parallel adaptive mesh refinement frameworks (Dubey et al., 2014), multigrid-preconditioned Krylov subspace iteration method frameworks for sparse linear systems (Saad, 2003), iterative Newton-Krylov preconditioner method frameworks for coupled multi-physics systems (Knoll and Keyes, 2004), and multilayer fast multipole method frameworks for solving frequency-domain Maxwell equations (Darve, 2000). Different from the algorithm frameworks, the application-feature-driven pre-conditioner approach lacks both a theoretical basis and common algorithm modules, and requires further enhancements. The pre-conditioner module can be incorporated into algorithm frameworks for verification and validation, and can be developed in parallel with the algorithm framework, which is helpful in the improvement of programming productivity.

3 Bottlenecks and strategies for computing efficiency

Numerical algorithms need to be redesigned as parallel algorithms to fit the supercomputer architecture before the implementation starts. For parallel algorithm design, computing efficiency is the

core indicator, which can be calculated as the ratio of the floating-point performance obtained to the theoretical peak of involved machine resources. For instance, if a parallel algorithm achieves 200 Tflops on a supercomputer whose peak performance is 1000 Tflops, then the computing efficiency of the algorithm is 20%. Generally, the computing efficiency of one algorithm is considered high when it is above 20% and low when below 5%.

It is well known that there are three major factors that affect the computing efficiency of parallel algorithms: data communication overhead, load balancing efficiency, and single-core floating-point performance. When entering the peta-flops era, the supercomputer architecture converges to a coupled architecture composed of multi-level deep nesting parallelism of general-purpose processors and specific heterogeneous many-core accelerators (Yang, 2012). With the advent of exascale computing, the general-purpose processor part gains more and more nesting levels. It is at six levels at the moment: distributed memory (DM), distributed shared memory (DSM), symmetric multiprocessing (SMP), multi-level cache, instruction level parallelism (ILP), and instruction vectorization parallelism (IVP).

On the other hand, state-of-the-art many-core processors have already integrated more than 1000 cores. Only if the parallel algorithms match this hierarchy of architecture characteristics, can they enjoy high computing efficiency. Fig. 1 shows the mentioned six-level nesting supercomputer architecture, and how a parallel algorithm can adapt to each level of features, which was discussed by Mo et al. (2016) and will not be repeated here.

The data communication overhead comes from three major sources: data transfer between computer nodes, data transfer and cache-coherence protocol overhead among processors and processor cores, and data transfer between general-purpose processors and heterogeneous many-core accelerators (Chung et al., 2011). The load balancing efficiency depends on two major factors: the effectiveness of load balancing between computing nodes, among processors in a single node, and among processor cores, and the extra cost of transferring workload among distributed memory space when migrating imbalanced loads (Liu et al., 2018). The floating-point performance reflects the execution speed of the program within the processor, and is affected mainly by factors such as the multi-level cache hit rate, instruction level parallelism, register reuse, and vectorization (Hennessy and Patterson, 2003). The difference in both parallel algorithms and the load characteristics of numerical simulations can impact the relative contributions of these three factors to computing efficiency. For example, for hydrodynamics simulation on a single-level grid with the Euler equation, the computing efficiency is dominated by data

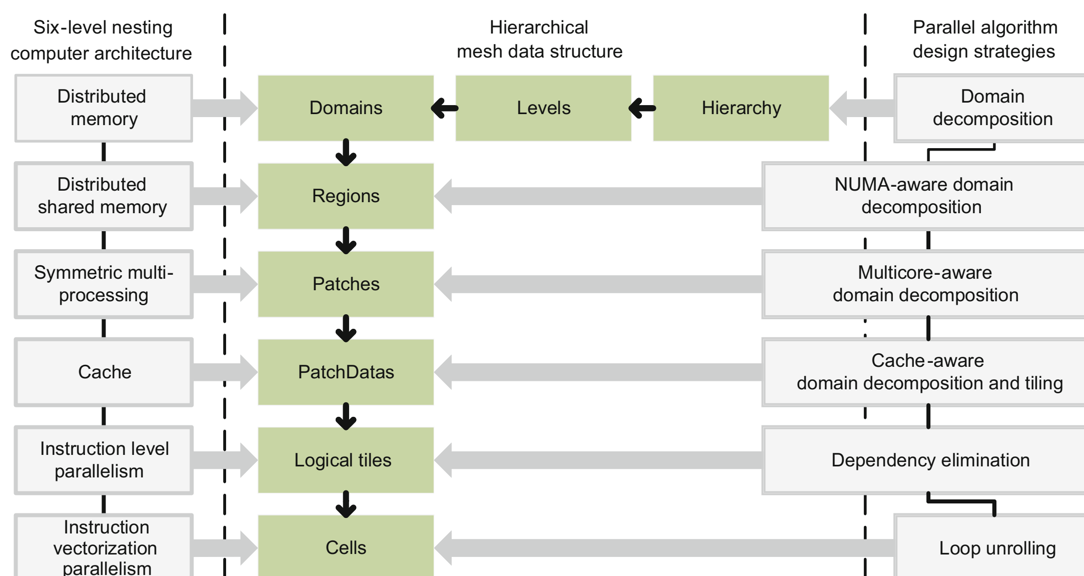

Fig. 1 Six-level nesting hierarchical supercomputer architecture and corresponding parallel algorithm design strategies

communication overhead and single-core floating-point performance. However, for the same algorithm on adaptively refined multi-level grids, the computing efficiency is dominated by the load balancing efficiency (Wissink et al., 2001). Table 1 lists the ways in which computer architecture features can affect the data communication overhead, load balancing efficiency, and single-core floating-point performance.

Computer architecture is a structured abstraction of computer hardware to guide the design and analysis of parallel algorithms. However, there is a gap between abstracted computer architecture and real hardware, and its runtime characteristics. As the peak performance of computers increases, this gap broadens and seriously downgrades the computing efficiency. Thus, how to combine the application code with the hardware features and runtime characteristics of the computer to further minimize data communication overhead, minimize processor idling, and increase single-core floating-point performance, will be a major research topic of performance optimization of parallel algorithms (Sarkar et al., 2016). Practice shows that with the rapid evolution of computer architecture, the impact of performance optimization on the computing efficiency gradually increases. Table 2 shows the contribution of performance optimization to the computing efficiency in the last five-year Gordon Bell Prizes (Rossinelli et al., 2013; Shaw et al., 2014; Rudi et al., 2015; Yang et al., 2016; Fu et al., 2017).

Based on the above analysis, it can be seen that different combinations of numerical algorithms and computer architectures require different parallel algorithms. Accordingly, the resulting complexity of parallel algorithm research is then $O(MN)$, where M denotes the total number of numerical algorithms and N denotes the number of architecture categories. In fact, parallel algorithms are common technologies that can be reused in the parallelization of numerical algorithms and the performance optimization on architectures after proper abstraction and formalization. Concretely, one can abstract the parallelisms of numerical algorithms, and express them as parallel computing patterns on top of common data models. For each combination of parallel computing pattern and computer architecture, efficient parallel algorithms can be designed. Although parallel algorithms need to be changed along with the evolution of computer architectures, the parallel computational patterns and numerical algorithms can stay unchanged, as can the numerical discretization of the mathematical physics equations. Mo et al. (2016) abstracted and refined the data model together with parallel computing patterns for more that 20 types of commonly used parallelisms in parallel numerical algorithms, and discussed methods in the design of efficient parallel implementations on supercomputers. In this way, the design of numerical algorithms and parallel algorithms can be separated and completed by different teams. In addition, the complexity of parallel algorithm research can be reduced from $O(MN)$ to $O(\alpha N)$, where α represents the total number of parallel computing patterns. Similarly, floating-point performance optimization research can be separated from parallel algorithm design and accomplished by different research teams, reducing the complexity from $O(JK)$ to $O(\beta K)$. Here J is the total number of parallel algorithms, K is the total number of computer runtime states, and β is the total number of computer architecture features. In terms of floating-point performance optimization, a large amount of

Table 1 Relationships among the supercomputer architecture and factors that impact computing efficiency

Architecture	Data communication			Load balancing efficiency			Single-core FP	
	Inter-node	Inter-socket	Inter-core	Inter-node	Inter-socket	Inter-core	CPU core	Vectorization
DM	√	–	–	√	–	–	–	–
DSM	–	√	–	–	√	–	–	–
SMP	–	–	√	–	–	√	√	–
Multi-level cache	–	√	√	–	√	√	√	√
ILP	–	–	–	-	–	√	√	√
IVP	–	–	–	-	–	√	√	√
Accelerator	√	–	–	√	–	–	–	√

DM: distributed memory; DSM: distributed shared memory; SMP: symmetric multiprocessing; ILP: instruction level parallelism; IVP: instruction vectorization parallelism

Table 2 Contributions of performance optimization to computing efficiency of last five-year Gordon Bell Prizes

Year	Contribution of performance optimization
2013	2.2× to 3.7× of different components
2014	No baseline
2015	Nearly 6.6×
2016	Nearly 3.5×
2017	3.8× to 8.9× of different components

research work has been conducted on common algebraic operations, including general dense matrix multiplication (GEMM), sparse matrix multiplication (SpMM), sparse matrix-vector multiplication (SpMV), and vector dot product. Libraries like OSKI (Vuduc et al., 2005) and SMAT (Li et al., 2013) have been developed. Due to the ability of adaptive optimization on target computer architecture and high performance, these libraries have been widely used in real-world applications.

4 Bottlenecks and strategies for programming productivity

The supercomputer architecture and parallel algorithm design methods (Fig. 1) complicate parallel programming. Specifically, Table 3 lists parallel programming languages and environments corresponding to computer architecture features. As shown in Table 3, numerical simulation experts need to master a 'four plus one' parallel programming stack, namely 'Process-Thread-Cache-Vector', which is coupled with heterogeneous many-core accelerators. The 'Process' level refers to the parallel programming of message passing between computing nodes. The 'Thread' level refers to two-layer nested shared memory programming between multiple central processing units (CPUs) on the same node and multiple cores within a CPU. The 'Cache' level refers to access optimization of multi-level caches in the CPUs. The 'Vector' level refers to vectorization of loops in the code. Such a complicated parallel programming stack, coupled with heterogeneous many-core accelerated programming techniques, will certainly increase the costs of intelligence, time, and resources, and bring challenges to code extensibility, portability, and inheritability. Under extreme circumstances, when the numerical simulation software is upgraded from peta-flop computers to ten-peta-flop computers and then to hundred-peta-flop computers, codes need to be refactored thoroughly, which makes the inheritance and development of software assets difficult. In fact, such cases have occurred frequently. For example, the codes on general-purpose graphics processing units (GPGPUs) cannot run on the Intel MIC architecture, the codes on Intel MICs cannot fit in domestic processors, and the codes on different domestic processors cannot run on each other seamlessly. At present, it is still a challenge for both parallel computing and exascale computing programming to greatly reduce the risk of code refactoring and to effectively improve the productivity of parallel programming (Amarasinghe et al., 2011).

Reuse of parallel algorithms and programming is an effective way to improve productivity. This innovative technological route was described systematically by Mo (2016) and Mo et al. (2016). The key technologies comprise two aspects: reuse of parallel computing patterns and reuse of numerical algorithm frameworks. The former is based on data models and parallel computing patterns that characterize parallel behaviors. Specifically, the parallel computing patterns are encapsulated as parallel computing components, and computational science experts can

Table 3 Parallel programming languages or environments corresponding to computer architecture features

Architecture	Parallel programming language and environment			Single-core FP programming	
	Inter-node	Inter-socket	Inter-core	CPU core	Vectorization
DM	Process	Process	Process	–	–
DSM	–	Process/Threads	–	–	–
SMP	–	–	Process/Threads	–	–
Multi-level cache	–	–	Cache	Cache	–
ILP	–	–	–	Cache	SIMD
IVP	–	–	–	Cache	SIMD
Accelerator	–	–	Offload	Cache	SIMD/SIMT

DM: distributed memory; DSM: distributed shared memory; SMP: symmetric multiprocessing; ILP: instruction level parallelism; IVP: instruction vectorization parallelism; SIMD/T: single instruction multiple data/thread

develop numerical algorithms by directly configuring and assembling parallel computing components. The latter is based on the separation of the numerical algorithm framework and pre-conditioners. Specifically, the numerical algorithm framework is encapsulated as numerical algorithm components, and the interface of components can be used for the development of pre-conditioner modules by computational experts. Serial programming is sufficient for the assembly of parallel computing components and the implementation of pre-conditioner kernels. All these components are encapsulated into a component library, which is called a 'parallel programming framework'. Based on this programming framework, numerical software can be developed by instantiation, configuration, and assembly of these components, a process that does not involve parallel programming. As a result, based on the parallel programming framework, development of numerical simulation software can obtain massively parallel computing capabilities automatically, and achieve an 'automatic parallelization and high scalability' goal. Mo et al. (2010) took the JASMIN framework as an example to systematically elaborate on the feasibility, effectiveness, and advancement of this technological approach (Mo, 2014, 2015; Mo et al., 2015). This will not be repeated here.

5 Supply-side supporting technology

In previous sections, we systematically analyze key technology issues that need to be solved continuously together with corresponding coping strategies, from the perspectives of computational scale, computing efficiency, and programming productivity. Table 4 summarizes these key technology issues with corresponding coping strategies. As we move towards exascale computing, we need to focus our efforts on the most important topics. Thus, we need to find an effective and quantitative approach for assessing weaknesses in computing capability. Specifically, this evaluation method is called 'supply-side' technology in parallel computing research.

Supply-side technologies are different from the usual performance analysis tools. Supply-side technologies are based on the componentization of numerical algorithm frameworks, parallel computing patterns, performance optimization techniques, and component-based parallel programming, while the usual tools are based on software codes and program modules. With the help of quantitative computing capability evaluation of different computing components, the computing capabilities of the numerical simulation process can be achieved by assessing the computing capabilities of the involved numerical simulation software, which in turn can be assessed by aggregating the capabilities of the involved components. Fig. 2 shows the infrastructure of computing capability evaluation. This infrastructure is divided into five layers: software system, numerical application, numerical algorithm, parallel component, and parallel pattern. Here, the computing capabilities of the software system are supported by many numerical applications. Each numerical application consists of numerical algorithms. Each numerical algorithm consists of parallel computing components. Finally, each parallel computing component involves parallel computing patterns. As a result, the computing capabilities of the software system can be assessed in terms of these five levels. In each level, the assessment first reveals the computing capability bottlenecks of each element, which are then used to guide

Table 4 Key technology issues with corresponding coping strategies in the continuous improvement of computing capabilities

	Computational scale	Computing efficiency	Programming productivity
Key issue	Application-feature-driven design of numerical algorithms	Mismatches among parallel algorithms, performance optimization techniques, and computer architecture features	Parallel computing patterns and parallel algorithm architecture, performance optimization patterns and technical architecture
Target	Linear complexity	Above 20%	Automatic parallelization, high scalability
Coping strategy	Algorithm frameworks, pre-conditioners	Data communication algorithms, load-balancing methods, and performance optimization of floating-point operations matching computer architecture	Programming frameworks, programming by component assembly

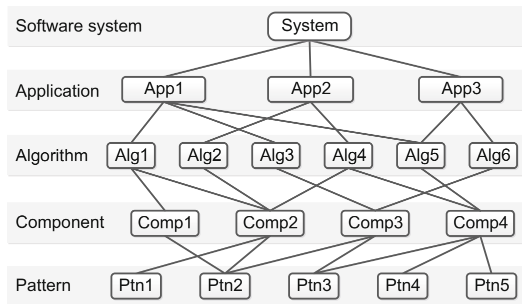

Fig. 2 Architecture of computing-capability evaluation models

the improvement of elements in the next level. In particular, this evaluation model is aimed mainly at the computational capabilities evaluation of software systems, and is thereby applicable to complex numerical simulation systems with multi-physics coupling and multi-scale problems.

Specifically, take the numerical simulation of inertial confinement fusion as an example. The software system is a numerical simulation system of a laser target coupling with radiation hydrodynamics (Pei and Zhu, 2009), which is supported by numerical applications of multi-material fluid mechanics, radiation diffusion, radiation transport, laser propagating, and radiative opacity. Each simulation software comprises different kinds of numerical algorithm components based on explicit or implicit time discretization schemes. Each algorithm component contains multiple different parallel computing components. According to runtime characteristics, each parallel computing component performs the performance optimization of data communication on network, multi-level cache access, and ILP. The quantitative computational capabilities evaluation method needs hierarchical analysis in terms of computational scale, computing efficiency, and software architecture.

6 Conclusions and future work

To address the exascale computing challenge, we have systematically analyzed the parallel computing bottlenecks from three perspectives: computational scale, computing efficiency, and programming productivity. Based on these analyses, we have proposed a series of key technology issues that demand research effort and corresponding coping strategies. For the computational scale bottleneck, the keys are numerical algorithm frameworks, application-driven pre-conditioners, and their coupling bottlenecks. For the computing efficiency bottleneck, the keys would be data communication, load balancing, and floating-point performance optimizations under deeply nested hierarchical parallelism coupled with heterogeneous many-core accelerators. For the programming productivity bottlenecks, domain-specific parallel programming frameworks would be a viable and effective solution. To tackle these bottlenecks with minimal investment, supply-side supporting technology is essential. By and large, with the continuous deepening of multi-physics and multi-scale research and the innovation of supercomputer architectures, extreme-scale parallel computing research will continue to expand and remain full of challenges and opportunities.

References

Amarasinghe S, Hall M, Lethin R, et al., 2011. Exascale programming challenges. Technical Report of the Workshop on Exascale Programming Challenges.

Ashby S, Beckman P, Chen J, et al., 2011. The opportunities and challenges of exascale computing. Summary Report of the Advanced Scientific Computing Advisory Committee Subcommittee.

Balay S, Gropp WD, McInnes LC, et al., 1997. Efficient management of parallelism in object-oriented numerical software libraries. In: Arge E, Bruaset AM, Langtangen HP (Eds.), Modern Software Tools for Scientific Computing. Birkhauser Boston Inc., Cambridge, USA. https://doi.org/10.1007/978-1-4612-1986-6_8

Campos C, Roman JE, 2012. Strategies for spectrum slicing based on restarted Lanczos methods. *Numer Algor*, 60(2):279-295. https://doi.org/10.1007/s11075-012-9564-z

Cao X, Mo Z, Liu X, et al., 2011. Parallel implementation of fast multipole method based on JASMIN. *Sci China Inform Sci*, 54(4):757-766 (in Chinese). https://doi.org/10.1007/s11432-011-4181-3

Chung IH, Lee CR, Zhou J, et al., 2011. Hierarchical mapping for HPC applications. IEEE Int Symp on Parallel and Distributed Processing Workshops and PhD Forum, p.1815-1823. https://doi.org/10.1109/IPDPS.2011.340

Cooley JW, Tukey JW, 1965. An algorithm for the machine calculation of complex Fourier series. *Math Comput*, 19(90):297-301. https://doi.org/10.1090/S0025-5718-1965-0178586-1

Darve E, 2000. The fast multipole method: numerical implementation. *J Comput Phys*, 160(1):195-240. https://doi.org/10.1006/jcph.2000.6451

Dolean V, Jolivet P, Nataf F, 2015. An Introduction to Domain Decomposition Methods: Algorithms, Theory, and Parallel Implementation. Society for Industrial and Applied Mathematics, Philadelphia, USA. https://doi.org/10.1137/1.9781611974065

Dongarra J, Foster I, Fox G, et al., 2003. The Sourcebook of Parallel Computing. Morgan Kaufmann Publishers Inc., San Francisco, USA.

Dubey A, Almgren A, Bell J, et al., 2014. A survey of high level frameworks in block-structured adaptive mesh refinement packages. *J Parall Distr Comput*, 74(12):3217-3227.
https://doi.org/10.1016/j.jpdc.2014.07.001

Engheta N, Murphy WD, Rokhlin V, et al., 1992. The fast multipole method (FMM) for electromagnetic scattering problems. *IEEE Trans Antenn Propag*, 40(6):634-641.
https://doi.org/10.1109/8.144597

Falgout RD, Yang UM, 2002. Hypre: a library of high performance pre-conditioners. Int Conf on Computational Science, p.632-641.

Fu H, He C, Chen B, et al., 2017. 18.9-Pflops nonlinear earthquake simulation on Sunway TaihuLight: enabling depiction of 18-Hz and 8-meter scenarios. Int Conf for High Performance Computing, Networking, Storage, and Analysis, p.1-12.
https://doi.org/10.1145/3126908.3126910

Hennessy JL, Patterson DA, 2003. Computer Architecture: a Quantitative Approach. Morgan Kaufmann Publishers Inc., San Francisco, USA.

Hernandez V, Roman JE, Vidal V, 2005. SLEPc: a scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans Math Softw*, 31(3):351-362.
https://doi.org/10.1145/1089014.1089019

Heroux MA, Bartlett RA, Howle VE, et al., 2005. An overview of the Trilinos project. *ACM Trans Math Softw*, 31(3):397-423.
https://doi.org/10.1145/1089014.1089021

Johansen H, McInnes LC, Bernholdt DE, et al., 2014. Software productivity for extreme-scale science. DOE Workshop Report.

Keyes DE, Mcinnes LC, Woodward CS, et al., 2013. Multiphysics simulations: challenges and opportunities. *Int J High Perform Comput Appl*, 27(1):4-83.
https://doi.org/10.1177/1094342012468181

Knoll DA, Keyes DE, 2004. Jacobian-free Newton-Krylov methods: a survey of approaches and applications. *J Comput Phys*, 193(2):357-397.
https://doi.org/10.1016/j.jcp.2003.08.010

Li J, Zhang X, Tan G, et al., 2013. SMAT: an input adaptive sparse matrix-vector multiplication auto-tuner. *ACM SIGPLAN Not*, 48(6):117-126.
https://doi.org/10.1145/2499370.2462181

Liu X, Yang Z, Yang Y, 2018. A nested partitioning load balancing algorithm for Tianhe-2. *J Comput Res Devel*, 55(2):418-425.
https://doi.org/10.7544/issn1000-1239.2018.20160877

Lucas R, Ang J, Bergman K, et al., 2014. DOE Advanced Scientific Computing Advisory Subcommittee report: top 10 exascale research challenges.
https://doi.org/10.2172/1222713

Mo Z, 2014. Domain-specific programming model for high performance scientific and engineering computation. *Commun CCF*, 10(1):8-12 (in Chinese).

Mo Z, 2015. Progress on high performance programming framework for numerical simulation. *E-Sci Technol Appl*, 6(4):11-19 (in Chinese).
https://doi.org/10.11871/j.issn.1674-9480.2015.04.002

Mo Z, 2016. High performance programming frameworks for numerical simulation. *Nat Sci Rev*, 3(1):28-29.
https://doi.org/10.1093/nsr/nwv086

Mo Z, Zhang A, Cao X, et al., 2010. JASMIN: a parallel software infrastructure for scientific computing. *Front Comput Sci China*, 4(4):480-488.
https://doi.org/10.1007/s11704-010-0120-5

Mo Z, Zhang A, Liu Q, et al., 2015. Research on the components and practices for domain-specific parallel programming models for numerical simulation. *Sci Sin Inform*, 45(3):385-397 (in Chinese).
https://doi.org/10.1360/N112013-00197

Mo Z, Zhang A, Liu Q, et al., 2016. Parallel algorithm and parallel programming: from specialty to generality as well as software reuse. *Sci Sin Inform*, 46(10):1392-1410 (in Chinese). https://doi.org/10.1360/N112016-00144

Pei W, Zhu S, 2009. Scientific computing for laser fusion. *Physics*, 38(8):559-568 (in Chinese).
https://doi.org/10.3321/j.issn:0379-4148.2009.08.005

Reed DA, Bajcsy R, Fernandez MA, et al., 2005. Computational science: ensuring America's competitiveness. Research Report No. ADA462840. President's Information Technology Advisory Committee.
http://www.dtic.mil/dtic/tr/fulltext/u2/a462840.pdf

Rossinelli D, Hejazialhosseini B, Hadjidoukas P, et al., 2013. 11 Pflop/s simulations of cloud cavitation collapse. Int Conf on High Performance Computing, Networking, Storage, and Analysis, p.1-13.
https://doi.org/10.1145/2503210.2504565

Rudi J, Malossi ACI, Isaac T, et al., 2015. An extreme-scale implicit solver for complex PDEs: highly heterogeneous flow in Earth's mantle. Int Conf for High Performance Computing, Networking, Storage, and Analysis, p.1-12.
https://doi.org/10.1145/2807591.2807675

Saad T, Darwish M, 2009. A high scalability parallel algebraic multigrid solver. In: Deconinck H, Dick E (Eds.), Computational Fluid Dynamics. Springer Berlin Heidelberg, p.231-236.
https://doi.org/10.1007/978-3-540-92779-2_34

Saad Y, 2003. Iterative Methods for Sparse Linear Systems (2nd Ed.). Society for Industrial and Applied Mathematics, Philadelphia, USA.

Sarkar V, Budimlic Z, Kulkani M, 2016. 2014 runtime systems Summit. Runtime Systems Report.
https://doi.org/10.2172/1341724

Shaw DE, Grossman JP, Bank JA, et al., 2014. Anton 2: raising the bar for performance and programmability in a special-purpose molecular dynamics supercomputer. Int Conf for High Performance Computing, Networking, Storage, and Analysis, p.41-53.
https://doi.org/10.1109/SC.2014.9

Tian R, Zhou M, Wang J, et al., 2018. A challenging dam structural analysis: large-scale implicit thermomechanical coupled contact simulation on Tianhe-2. *Comput Mech*, p.1-21.
https://doi.org/10.1007/s00466-018-1586-5

Vuduc R, Demmel JW, Yelick KA, 2005. OSKI: a library of automatically tuned sparse matrix kernels. *J Phys Conf Ser*, 16:521-530.
https://doi.org/10.1088/1742-6596/16/1/071

Wissink AM, Hornung RD, Kohn SR, et al., 2001. Large scale parallel structured AMR calculations using the SAMRAI framework. ACM/IEEE Conf on Supercomputing, p.6. https://doi.org/10.1145/582034.582040

Xu X, Mo Z, 2017. Algebraic interface-based coarsening AMG pre-conditioner for multi-scale sparse matrices with applications to radiation hydrodynamics computation. *Numer Linear Algebra Appl*, 24(2):e2078.
https://doi.org/10.1002/nla.2078

Yang C, Xue W, Fu H, et al., 2016. 10M-core scalable fully-implicit solver for non-hydrostatic atmospheric dynamics. Int Conf for High Performance Computing, Networking, Storage, and Analysis, p.1-12.
https://doi.org/10.1109/SC.2016.5

Yang X, 2012. Sixty years of parallel computing. *Comput Eng Sci*, 34(8):1-10 (in Chinese).
https://doi.org/10.3969/j.issn.1007-130X.2012.08.001

Zhao Z, Zhou H, Ma H, et al., 2014. Numerical simulation and verification of electromagnetic pulse effect of PIN diode limiter. *High Power Laser Particle Beams*, 26(6):81-85 (in Chinese).
https://doi.org/10.11884/HPLPB201426.063018