

Co-Design of Systems and Applications for Exascale

Edited by

Arndt Bode¹, Adolfy Hoisie², Dieter Kranzlmüller³, and
Wolfgang E. Nagel⁴

- 1 Technische Universität München & Leibniz Supercomputing Centre Garching, DE, bode@informatik.tu-muenchen.de
- 2 Pacific Northwest National Lab., US, adolfy.hoisie@pnnl.gov
- 3 Ludwig-Maximilians-Universität München & Leibniz Supercomputing Centre Garching, DE, kranzlmue@nm.ifi.lmu.de
- 4 Technische Universität Dresden, DE, wolfgang.nagel@tu-dresden.de

Abstract

With more and more machines achieving petascale capabilities, the focus is shifting towards the next big barrier, exascale computing and its possibilities and challenges. There is a common agreement that using machines on this level will definitively require co-design of systems and applications, and corresponding actions on different levels of software, hardware, and the infrastructure. Defining the vision of exascale computing for the community as providing capabilities on levels of performance at extreme scales, and identifying the role and mission of the involved experts from computer science has laid the basis for further discussions. By reflecting on the current state of petascale machines and technologies and identifying known bottlenecks and pitfalls looming ahead, this workshop derived the concrete barriers on the road towards exascale and presented some ideas on how to overcome them, as well as raising open issues to be addressed in future leading-edge research on this topic.

Seminar 22.–25. May, 2012 – www.dagstuhl.de/12212

1998 ACM Subject Classification B. Hardware, D.2 Software Engineering

Keywords and phrases Exascale, Co-Design, Scalability, Power Efficiency, Reliability

Digital Object Identifier 10.4230/DagRep.2.5.71

Edited in cooperation with Christian Straube

1 Executive Summary

Arndt Bode

Adolfy Hoisie

Dieter Kranzlmüller

Wolfgang E. Nagel

License  Creative Commons BY-NC-ND 3.0 Unported license
© Arndt Bode, Adolfy Hoisie, Dieter Kranzlmüller, and Wolfgang E. Nagel

With Petascale computing being a reality today, the focus of the computational science community is already on the next barrier – exascale computing. With systems even more powerful by orders of magnitude, scientists start thinking about the possibilities and challenges. This workshop addressed the many scientific, technological, and financial challenges of exascale level computing with the hypothesis that exascale computing is only possible by co-designing across different levels of software, hardware, and the surrounding infrastructure.



Except where otherwise noted, content of this report is licensed under a Creative Commons BY-NC-ND 3.0 Unported license

Co-Design of Systems and Applications for Exascale, *Dagstuhl Reports*, Vol. 2, Issue 5, pp. 71–92

Editors: Arndt Bode, Adolfy Hoisie, Dieter Kranzlmüller, and Wolfgang E. Nagel



DAGSTUHL
REPORTS

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The workshop program has been composed of a series of short talks, less than 20 minutes on average, and extensive time for discussions. Starting with an overview of the workshop motivation and the general methodologies for co-design, different aspects of co-design have been addressed. This has been followed by talks on modeling, simulation and tools, as well as programming models, runtime support and compilers. The second part addressed the specific problems of system-software for performance, power and reliability and the resulting system architectures, while finally application level aspects of exascale co-design have been discussed between the participating experts from different areas of high performance computing. In all discussions it has been important to tackle a multidimensional combination of major challenges associated with the development of exascale systems and applications from different angles instead of addressing an isolated aspect.

The results of the workshop are manifold¹: Firstly, the vision based on the requirements of the scientific community is thus “to provide exascale capabilities to scientific and engineering applications”, where it is important to notice that exascale means extreme scale or large scale, not the particular barrier of exaflop performance looming ahead. With this vision at hand, the participating experts identified their particular role and mission as follows: “to co-design systems such that they reach exascale capabilities within the given technological and non-technical (social, ...) boundaries”. Each expert has been knowledgeable on a distinct layer of the exascale architecture, the mission requires expertise across all layers, and exascale computing requires involvement from all relevant areas of computer science in order to perform exascale co-design of hard- and software, including also different levels of software working closely together with hardware and the interfacing to the environmental infrastructure. This has led to the definition of co-design, where two or more distinct activities collaborate on and across different layers to design a system architecture for a specific goal.

In summary, the workshop has reflected on the current state of petascale machines providing multiple examples from world-leading machines and using them to derive the barriers on the road towards exascale computing. Looking beyond the current research into the future, where exascale computing will become feasible, we have been trying to identify the exascale roadmap with intermediate goals and pitfalls on the way to exascale, and leveraging the combined forces of computer science to overcome them.

¹ A scientific paper will be created within the next months.

2 Table of Contents

Executive Summary

Arndt Bode, Adolfy Hoisie, Dieter Kranzlmüller, and Wolfgang E. Nagel 71

Exascale Co-Design Methodologies (Overview)

Exascale Co-Design Workshop: Overview and Motivation
Adolfy Hoisie 75

Co-Design: a Holistic Approach on Energy-Efficient Computing
Arndt Bode 75

Another Perspective to Co-Design . . . from one of those who did not raise their hand
Hans Bungartz 75

Performance Engineering based on Knowledge instead of Abstraction
Jan Treibig 76

Exascale Co-Design Methodologies (Modeling, Simulation, Tools)

Tools and Tool Infrastructures for Co-Design
Martin Schulz 77

An Analytical Framework for Algorithm-Architecture Co-Design
Kent Czechowski 77

Co-Design – how to? Generality vs. Specialization in Simulation Frameworks
Sabine Roller 78

Reaching the Pacific Ocean – Co-Design as a long term activity
Michael Resch 78

Exascale System Software (Programming Models, Runtimes, Compilers)

Exascale System Software: Programming Models, Runtimes, Compilers, and Tools
John Mellor-Crummey 80

Performance Observation in Exascale Co-Design
Allen Malony 81

Software Engineering for HPC – A Gap to be Closed
Felix Wolf 81

The classical What-IF-Question: What happens if we are not successful with Hardware or Software on Exascale Level?
Wolfgang E. Nagel 82

Programming must become easier – not more complex!
Bernd Mohr 82

Parallel Programming and Execution Models for Exascale – Evolution or Revolution?
Bettina Krammer 83


Unifying Scalability Infrastructures
Barton P. Miller 83

Using Application Proxies for Co-Design of Future HPC Computer Systems and Applications <i>Alice Koniges</i>	84
Resource (Energy) Saving System Software Stack for Exascale Systems <i>Shinji Sumimoto</i>	84
Exascale Systems (Co-Design with Apps, System-Software for Performance / Power / Reliability)	
Resource Aware Programming <i>Michael Gerndt</i>	85
Relating Exascale to Parallelism, Power and Energy <i>Zhiwei Xu</i>	86
Exascale Systems	
Characterizing Application-Architecture Co-Design by Suitability Functions <i>Vladimir Getov</i>	86
Software Co-Design for Exascale <i>Erwin Laure</i>	86
Exascale Architecture (Design, Execution Models, Power, Reliability)	
Co-Design Challenges of Many-Core Systems-on-Chip <i>Daniel Molka</i>	87
Experiences in Co-Design: Tackling the challenges of Performance, Power, and Reliability <i>Darren Kerbyson</i>	87
Co-Design & Resiliency for Exascale Computing <i>Stephen L. Scott</i>	88
Application Level Issues	
Co-Designing for Online Auto Tuning <i>Jeff Hollingsworth</i>	89
Exascale Co-Design – Data <i>Achim Streit</i>	89
Software-Software Co-Design of Applications and Tools <i>Karl Furlinger</i>	90
How Can We Make Co-Design Really Work? <i>Cherri M. Pancake</i>	91
Participants	92

3 Exascale Co-Design Methodologies (Overview)

3.1 Exascale Co-Design Workshop: Overview and Motivation


Adolfo Hoisie (Pacific Northwest National Lab., US)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Adolfo Hoisie

- Modeling of systems and applications
- Simulation methodologies and frameworks
- Co-Design boundaries: architecture, application, system software, programming models, tools
- Co-Design process

3.2 Co-Design: a Holistic Approach on Energy-Efficient Computing

Arndt Bode (TU München & LRZ Garching, DE)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Arndt Bode

Co-Design is usually considered to be applied to architecture and applications in order to keep exascale systems programmable and scalable. Based on the experience with SuperMUC and other energy efficient HPC systems we advocate to extend the co-design principle to the entire “HPC production chain”: building infrastructure – power supply – climate and cooling – systems hardware architecture – systems software architecture including operating system, programming language, libraries and tools middleware – operating strategies – application and algorithm.


Some arguments for the extension of the co-design paradigm are

- all of the elements of the HPC production chain listed above influence strongly the amount of energy consumed by the solution
- future exascale system cost and technical feasibility are strongly dependent on energy consumption
- many of the elements of the HPC production chain are strongly correlated (example: to what extent does the operating strategy allow the application program to control the clocking of the system)

A discussion on this holistic approach to co-design should bring together all parameters influencing energy consumption and their mutual dependency.

3.3 Another Perspective to Co-Design . . . from one of those who did not raise their hand

Hans Bungartz (TU München, DE)


License  Creative Commons BY-NC-ND 3.0 Unported license
© Hans Bungartz

- Meaning of co-design: not only the “classical” HW-SW one, but also (and maybe even more important) SW-SW co-design, having in mind the complete SW stack for system SW to application software

- How far can/should/must the co-design of systems and applications go? i.e. application-tailored (narrow) systems (MD-Grape, QPACE, ...) vs. application-type systems (BlueGene, e.g.) vs. general-purpose systems
- Maybe a more philosophical issue: From the general application perspective, “exascale” pushes HPC into a narrower niche. Only a few communities have the experience and the codes for petascale, and the number will decrease towards exascale. And only a few communities really need exascale computing, and also this number will decrease towards exascale. Nevertheless, the technologies on the exascale agenda will bring benefit to the non-high-end computing sector, too. We should keep this in mind – in particular with regard to engineering/industrial applications
- Should we make a “funding roundtable” – maybe in the evening? Discussing problems, presenting briefly new initiatives world-wide, reporting the IESP/EESI status, ...?
- algorithmic issues: impact of the architecture on algorithm design, esp. having in mind the big algorithmic paradigms such as hierarchy, adaptivity, dynamics, multi-level – which are all somewhat nasty for exascale ...

3.4 Performance Engineering based on Knowledge instead of Abstraction

Jan Treibig (Universität Erlangen-Nürnberg, DE)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Jan Treibig





Great improvements of the efficiency of applications are possible using a holistic iterative performance engineering approach guided by a deep understanding of the interaction between software and the hardware.

Exascale requires an extreme effort with regard to the efficient use of available hardware. We propose a holistic iterative optimization process, which takes into account code, machine and runtime analysis in order to iteratively create a diagnostic performance model. This is a white box approach in every aspect, leading to a deeper knowledge of the characteristics of the application code and the properties and capabilities of the underlying hardware. The application developer himself is conducting this effort forcing him to get a deeper understanding of the interaction of his code with the hardware. This is in opposite to the widespread belief that the developer should be safeguarded from these details by software abstraction and tooling through a black box approach. It is our attitude that there is no alternative to deep knowledge about what is happening on all levels. Making this knowledge available in a transparent fashion to all participating disciplines is the key for a sustainable optimization effort. All necessary aspects are already available including static code analysis tools, micro benchmarking, profiling tools and hardware performance monitoring. This analysis allows to create an application- and hardware-specific diagnostic performance model, which explains the resulting performance. Since such a model is based on understanding and not on heuristics or statistics it can identify opportunities for possible optimization strategies and provides reliable statements about the attained efficiency of a code on a given architecture. By proposing and teaching this systematic method of performance engineering a great potential for increased efficiency will be leveraged.

4 Exascale Co-Design Methodologies (Modeling, Simulation, Tools)

4.1 Tools and Tool Infrastructures for Co-Design

Martin Schulz (LLNL – Livermore, US)

License     Creative Commons BY-NC-ND 3.0 Unported license
© Martin Schulz





Program development and analysis tools are an essential element of the co-design process. They enable the evaluation of applications and their characteristics and they can help predict the performance impact on future architectures. However, this requires the development of a new generation of tools that allows a more detailed and predictive performance analysis as well as an expansion of the concept of tools to include simulation and emulation environments of future systems to help understand the impact of architectural changes on applications and system software.

The GREMLIN project, which is in its early design stage as part of DOE/Office of Science Co-Design Center ExMatEx, aims at creating such an emulation infrastructure. It consists of a series of small modules, the GREMLINs, that target particular aspects of a system design and either limit targeted resources, such as caches, network bandwidth, or arithmetic units, or introduce faults or process variations. Combined, these GREMLINs will allow us to create exascale conditions on current machines and with that evaluate applications in larger and more realistic settings. The GREMLIN is intended to close the gap between cycle accurate simulation and execution on prototype hardware and is part of ExMatEx’s multi-pronged approach to performance and scalability analysis.

The GREMLIN will itself be built on top of a flexible tool infrastructure that allows us to easily extend and combine the various modules. Further, it enables the transparent integration of performance analysis tools on top of the GREMLIN and provides us comparative performance analysis of different system configurations. Such modular infrastructures are critical to support users at exascale and in particular during the co-design process. They enable us to quickly assemble tools from components, provide mechanisms for interoperability and concurrent execution, and allow us to target specific performance questions without having to rewrite each tool from scratch. This talk will highlight recent efforts in this area, which also reflect a larger trend in the tools community, and how they can help to create new tools, including the GREMLIN tool set.

4.2 An Analytical Framework for Algorithm-Architecture Co-Design

Kent Czechowski (Georgia Institute of Technology, US)


License     Creative Commons BY-NC-ND 3.0 Unported license
© Kent Czechowski

Traditional co-design analysis abstracts away algorithm features entirely (e.g., through an Amdahl’s Law style analysis) or draws conclusions from specific code artifacts (e.g., fixed benchmarks, traces, and machines). Instead, I advocate for a more general algorithm-architecture analysis that explicitly relates characteristics of an algorithm, such as its inherent parallelism or memory behavior, with parameters of an architecture, such as the number of cores, structure of the memory hierarchy, or network topology.

My approach marries abstract algorithmic complexity analysis with a formal representation of architecture design trade-offs. I believe this will enable us to say precisely and analytically how high-level changes to the architecture might affect the execution time of a computation; and, conversely, identify what classes of computation might best match a given architecture. It will necessarily not have the fidelity of cycle-accurate performance estimates possible through detailed simulation. Instead, the strength of the approach is that freedom from the artifacts of current hardware and software implementations, while still obeying technological constraints, may lead to radically new methods and insights into how to achieve performance and scalability in future exascale systems.

4.3 Co-Design – how to? Generality vs. Specialization in Simulation Frameworks


Sabine Roller (German Research School for Simulation Sciences, DE)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Sabine Roller

Simulation software development will at the same time be more individualized and more generalized. More individualized because leading edge software (beyond commercial solvers) needs to specialize. But to be able to specialize, it should not need to start from scratch, but to setup on a generalized basis. This idea stands behind projects like OpenFoam etc. which offer a (community-specific) toolbox in an open-source framework, which can be modified to the individual needs. Nevertheless, this idea runs into its limits at different points. One point is that generalization typically prevents from taking advantages of individual features, especially if the user doesn't overlook the basics, but uses the framework as a black box. Another point, which holds even for highly experienced developers with a deep knowledge, is the dependency on libraries. Currently, simulation software developers often get trapped in scalability issues which are not due to their own software, but due to the underlying libraries like MPI, ParMetis, HDF5. Thus, the key for the future development of HPC software is to resolve the individual view points of application software, middleware, OS software, and operational issues into a holistic end-to-end approach. We need to train more generalists, in addition to the specialists.

4.4 Reaching the Pacific Ocean – Co-Design as a long term activity

Michael Resch (Universität Stuttgart, DE)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Michael Resch

Introduction: In 1893 Frederick Jackson Turner wrote an essay on the significance of the frontier in American history. Referring to a bulletin of the Superintendent of the Census for 1890 he found that the impressive move westwards of the US-American settlers has found an end. In his description of the advance of the frontier Turner identifies 5 barriers that were reached over time – the Alleghenies, the Mississippi, the Missouri, the Rocky Mounties, and finally the Pacific Ocean. With the advent of the settlers at the Pacific Ocean the development of the USA turned mostly inwards and focused on the development of the settled country.

In Supercomputing we have seen a similar breath taking advance. Over only a few decades we have reached megaflops, gigaflops, teraflops, petaflops and are approaching exaflops. While many prepare for the usage of such exascale systems, others start to doubt whether we will be able to reach exaflops or go beyond that barrier. From a technical point of view there is no doubt that exaflops are possible. The driving factor, however, is no longer innovation but rather a massive usage of standard procedures and components.

Rationale: Over about 30 years Moore's law was proven to be right and clock frequencies have increased accordingly. Starting in the early 1990s increase of clock frequency was replaced by increase in level of parallelism. An analysis of the TOP500 shows that this trend is accelerating over the last 5-8 years. A consequence of this hardware race is that software cannot keep pace.

In the discussions about the development of exascale systems co-design plays an important role. By developing software and hardware at the same time one expects to overcome the asynchronies of the two technical development paths. Theoretically this is a reasonable approach. However, the asynchronies remain. The basic fact is that changes in hardware come in steps of 2-4 years. This is about the time horizon for the renewal of a system and an upgrade of existing hardware technology. We are still living in the age of rapidly changing technology when it comes to computers.

Software changes happen at a different speed. A short investigation of system software and application software shows that both types of software follow very similar patterns. First, there is an idea or a basic algorithm. In a second step there is some prototype implementation. Over a time of 3-5 years the software starts to mature. A renewal cycle for basic software typically lasts for about 20 years. Looking at the history of operating systems one might assume even longer cycles. If we take UNIX and Linux together as being based on very similar fundamental ideas, one would say that the life span is in the range of 20-40 years. Similar time frames can be found for application software. Basic concepts get implemented in prototype software. Over time these packages mature and become available to a wider user community. Overall the process of maturing a software approach takes at least about twice as much time as the change in hardware architecture and potentially even longer.

It remains to be seen whether a co-design approach can speed up the software development process. It is certainly going to be beneficial if software developers are very early on involved in the hardware development process. On the other hand, for a long time operating system development of large hardware providers was done in-house, such that an internal co-design was already taking place. Nothing indicates that such an in-house co-design was able to overcome the gap between fast hardware development and slow software development.

From this point of view, a slow down if not a complete stop in increase of speed could be helpful for software development. Just like reaching the Pacific Ocean allowed the US society and economy to focus on internal development, reaching the exascale barrier might allow us to focus on the development of mature software and further improve quality rather than speed of hardware.

5 Exascale System Software (Programming Models, Runtimes, Compilers)

5.1 Exascale System Software: Programming Models, Runtimes, Compilers, and Tools

John Mellor-Crummey (Rice University, US)

License  Creative Commons BY-NC-ND 3.0 Unported license
© John Mellor-Crummey


Exascale architectures will be dramatically different from today's systems in several ways. First, they will feature several orders of magnitude more parallelism. Second, they will come with severe resource limitations including power, memory capacity and bandwidth, bisection bandwidth, and I/O connectivity. Third, to deliver exascale performance in the face of severe resource limitations, exascale systems will employ a complex array of technologies, including heterogeneous cores and deep hierarchical memory systems. Finally, exascale systems will need to cope with faults; the scale of these systems will mean that errors will be several orders of magnitude more frequent than on systems today.

As a result, harnessing the vast computational power of exascale architectures will require fundamental changes to system software, runtime libraries, programming models, tools, and applications. Each layer of the software stack for exascale systems will need to be redesigned to exploit the explosion in parallelism, manage scarce resources, handle failure, and respond to changing application needs. However, we cannot expect developers of applications or other layers of the software stack to understand the nature and scope of changes needed for their software without information about how their existing software falls short of what is needed. Developers must be able to understand where, how, and why their software fails to use a target platform efficiently. The complexity of exascale systems will mean that software may not meet performance expectations for a variety of reasons. At a high level, an application's performance on a large-scale parallel system will depend upon how well it makes use of available resources for communication, computation, and I/O. At a more detailed level, using an exascale platform effectively will require identifying mismatches between software demands for resources and hardware capabilities. Only with detailed knowledge about the nature of shortcomings and a correlation of problems to the software will developers be able to understand how they need to restructure their code to leverage the strengths of a target platform and reduce consumption of scarce resources.

The complexity of hardware and software for exascale systems, as well as the concomitant myriad of potential causes responsible for an impediment to performance or scalability, will make performance tools indispensable for figuring out what to change to resolve problems that will inevitably arise. Building effective tools will require support from the hardware and all levels of the software stack to enable accurate measurement and attribution of performance problems. Tools must support analysis at all levels of the software stack and not just end-user applications. As a result, new techniques will be needed for measurement, analysis, attribution, and presentation of information about the performance of the software stack on exascale systems. Tools will need to identify code regions with are insufficiently parallel, consume excessive energy, contend for scarce resources, introduce large parallel overhead, or pose scalability bottlenecks. Two kinds of support will be needed. First, tools will need to support post-mortem analysis of executions to identify opportunities for code tuning. Second, performance analysis APIs will need to support on-line introspection and control to enable informed decision making by adaptive runtime systems in response to changing workload characteristics that affect parallelism, data locality, and resource consumption.

5.2 Performance Observation in Exascale Co-Design

Allen Malony (University of Oregon, US)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Allen Malony


The exascale co-design challenge demands a new perspective on the role of performance observation as an integral part of the exascale software stack that enables top-down application transformations to be optimized for runtime and system layers by feeding back dynamic information about hardware and software resources from the bottom-up. Performance observation and analysis technology should be an inherent aspect at all exascale levels to make it possible not only to bridge the gap between programming (computation) model semantics and execution model operation, but to deliver opportunities for online, adaptive optimization and control.

The reliance on post-mortem analysis of low-level performance measurements is prohibitive for exascale because of the performance data volume, the primitive basis for performance data attribution, and the inability to reflect back execution dynamics at runtime. With a multi-level exascale programming stack involving high-level transformations, it is necessary to provide richer context for attributions, beyond code locations and simple program events, together with a programmable, hierarchical, and dynamic “performance backplane” with model-driven measurement and analysis, and meaningful mapping back to program performance abstractions.

The perspective can go beyond the exascale software stack to consider how certain performance observation, computational semantics, and feedback support can be implemented in the exascale system architecture and what advantages it may entail. Thinking here could lead to the creation of new hardware technology to specifically to make the exascale machine more performance-aware and performance-adaptive.

5.3 Software Engineering for HPC – A Gap to be Closed

Felix Wolf (German Research School for Simulation Sciences, DE)


License  Creative Commons BY-NC-ND 3.0 Unported license
© Felix Wolf

The idea of co-designing HPC applications together with systems, and system software suggests that the isolated design of applications is already understood – an obvious misconception that becomes clear when browsing standard books on software engineering. Classic plan-based software engineering methodologies usually fail to address the needs of highly research-oriented projects with underspecified functional requirements. In addition, developers of exascale applications will have to satisfy a number of critical non-functional requirements, such as node-level performance, locality, scalability, energy efficiency, and fault-tolerance. Their successful realization presents not only a genuine methodological challenge in itself but aggravates the software engineering crisis of the HPC sector even further. Whereas the development of enterprise software is understood today to a degree where solutions to most of the recurring requirements are embedded in sophisticated generic application containers or platforms such as JavaEE, software engineering for HPC is still in its infancy. We therefore need to rethink our software engineering practices – seeking orientation along recurring problems found across larger classes of applications. This should

start with the cartography of those problem domains, ideally followed by the creation of adequate and easily accessible solutions to be co-designed together with other components of the exascale hardware and software ecosystem.

5.4 The classical What-IF-Question: What happens if we are not successful with Hardware or Software on Exascale Level?

Wolfgang E. Nagel (TU Dresden, DE)


License  Creative Commons BY-NC-ND 3.0 Unported license
© Wolfgang E. Nagel

Since a couple of years, exascale computing is used as the motivation buzzword to highlight the short-comings we have experienced in the development of HPC systems in many years. We have to recognize that technology is driving hardware development and software is following very slowly the given architectural trends. We had success with addressing Linpack performance, nevertheless we failed in many other areas like scalability, sustained applications performance, and I/O.

The talk will summarize a couple of facts and identify some strategies which could help to make reasonable progress in HPC computing, may be as a side effect even to exascale computing.

5.5 Programming must become easier – not more complex!

Bernd Mohr (Jülich Supercomputing Centre, DE)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Bernd Mohr

The current designs of exascale computer systems show two important trends:

- A huge number of execution units
- A three-level HW hierarchy
 1. Nodes
 2. General purpose CPUs
 3. Performance boosters (Accelerators, SIMD vector units, etc.)

Programming such systems currently requires to use a corresponding hybrid model with different programming models for each of the three levels:

1. MPI
2. Shared-memory multi-threading (OpenMP, Pthreads, ...)
3. OpenCL, CUDA, vector intrinsics, ...

Handling of data and work distribution, communication, and synchronization has to be handled explicitly at every level. In addition, currently there are no standardized interactions/interoperability between the programming models on the different levels. This is way too complex.

What is needed?


- Keep the explicit programming with MPI for expressing data distribution, communication, and synchronization between nodes. There is a lot of experience and codes here that are impossible to ignore. In part, PGAS 1-sided communication could be used where it makes sense and is more efficient. MPI should further evolve and needs to clearly

define interoperability with PGAS and intra-node level multi-threading. Tuning of the inter-node level is also done explicitly (with tool support).

- Hardware levels 2 and 3 should be programmed by one portable programming model. Compiler technology, smart runtime systems, and libraries need to hide the underlying complexity of the hardware. This programming model should be implicit: the programmer expresses opportunities of independent code execution of various granularity. Then compilers and the associated runtimes take care of efficient and fault-tolerant (e.g., by reexecuting failed executions) execution. Tuning on this level is also done implicitly and automatically by the runtime balancing performance and power efficiency.

5.6 Parallel Programming and Execution Models for Exascale – Evolution or Revolution?

Bettina Kramer (University of Versailles, FR)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Bettina Kramer

Are we going to have fundamentally new parallel programming models for exascale in a few years from now? Probably not. It seems naive to throw the past 20 years' experience in parallel programming overboard in the hope to find something new and better, ready for exascale, in the next 5 years. Support has to continue for huge amounts of legacy codes, predominantly written in C, C++ and/or FORTRAN, relying on standard APIs for parallelisation such as MPI, OpenMP or pthreads. Models emerging over the last years, e.g. PGAS (UPC, Co-array FORTRAN, ...), cilk, OpenCL, openacc, ... often lack maturity or portability across a wide range of platforms, or still fail to deliver performance on large-scale.

Applications, programming models and, not to forget, the underlying runtime implementations will have to evolve: away from pure MPI codes towards hybrid codes, combining MPI for inter-node communication e.g. with a shared-memory model inside a node or offload directives for hardware accelerators. Interfaces between programming models will have to be well-defined, and pressure on underlying runtime environments will increase to map (hybrid) programming concepts efficiently to the underlying hardware while hiding as much complexity as possible from the user.

5.7 Unifying Scalability Infrastructures

Barton P. Miller (University of Wisconsin – Madison, US)


License  Creative Commons BY-NC-ND 3.0 Unported license
© Barton P. Miller

A key mechanism for the scalability of large (or massive) scale systems is the tree-based communication network, sometimes called a tree-based overlay network (TBON). These TBON's provide the ability to perform command, control, and data gathering and reduction with logarithmic cost. In supercomputers, such networks are used extensively at the hardware level (for diagnostics, initialization, and maintenance), operating system level (for booting the system and launching programs), I/O system level (for scalable file system operations), tool level (for debugging and performance monitoring) and application level (for constructing scalable applications).

Real systems evolve with independent, individual tree-based infrastructures at every level. This is, at best, wasteful and, at worst, generates execution inefficiencies and interference. There is a strong need to a unified, cross-cutting design for such TBON facilities. These facilities need to handle a variety of clients with a variety of topology needs. In addition, different uses of the TBON infrastructure will need different amount of persistence and requirements for fault tolerance. While the challenge for designing such an infrastructure is great, the payoff can be huge.

5.8 Using Application Proxies for Co-Design of Future HPC Computer Systems and Applications

Alice Koniges (Lawrence Berkeley National Laboratory, US)


License  Creative Commons BY-NC-ND 3.0 Unported license
© Alice Koniges

The high performance computing community is in the midst of disruptive architectural changes. The advent of manycore and heterogeneous computing nodes, increased use of vectorization, light-weight threads and thread concurrency, GPUs, along with concerns about energy and resilience, force us to reconsider every aspect of the computer system, software and application stack, often simultaneously. One important toolset to aid the exploration of this complex design space is application proxies. Although addressing the needs of full-scale applications is the final target of any computer system and tools design effort, working directly with these codes early in the design process is practically limited and time-consuming. In contrast, working with application proxies is much easier and can provide tremendous value, if the proxies are properly designed and results properly interpreted.

There are a wide variety of available application proxies including traditional offerings (NAS Parallel Benchmarks, High Performance Linpack, etc.) that have been used over the years to evaluate system performance. For co-design, a more focused effort of proxies called compact apps and miniapp can permit a broader collection of activities, including completely rewriting them in new and emerging language paradigms. We propose to the community that a broad-based international effort of providing and evaluating compact apps and miniapps can give co-design a strong application base. Included in these studies should be linked websites, test cases, and reporting of results on a larger scale.

5.9 Resource (Energy) Saving System Software Stack for Exascale Systems

Shinji Sumimoto (Fujitsu – Kawasaki, JP)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Shinji Sumimoto

Predicted hardware specification for exascale systems will be very power sensitive and the amount of hardware resources, such as memory, processor, interconnect and etc, should be minimized. Moreover, processing not related main calculation processing such as OS, house keeping, monitoring, processing related adaptive functions, should be minimized.

However, current software stacks consume much hardware resources because of full function OS, libraries, and the other software. Therefore, current style of software stack must be re-organized and re-structured.

Some of discussion points are as follows:

- Should we provide current style of software stack for exascale systems?
- How should we provide functions needed for exascale systems?
- How should we design and building software stacks for exascale systems?


My idea for this issues are:

- Building software stack layer using less number of stacks.
- Minimizing provided functions only use at runtime.
- Dividing current software stack functions at runtime to pre-execution functions and real runtime functions, and pre-execution functions are realized as runtime optimization tools.

6 Exascale Systems (Co-Design with Apps, System-Software for Performance / Power / Reliability)

6.1 Resource Aware Programming

Michael Gerndt (TU München, DE)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Michael Gerndt

Resource management on current HPC systems is based on space sharing. The cores of the systems are statically partitioned and assigned to jobs. The jobs have exclusive access to those resources for their entire runtime.


Exascale systems will offer millions of cores and only very few applications will be available to use those resources efficiently over their entire execution. To increase the number of applications that benefit from exascale systems, the resource management should adapt the resources to the scalability of application phases.

Within the Special Research Area TRR 89 Invasive Computing, TUM investigates the concept of resource aware programming in the context of HPC. The invasive computing paradigm allows applications to dynamically invade additional resources, to infect those resources with computational tasks, and to retreat from the resources if they are no longer required. This concept will very well match the current application trend to use adaptive algorithms instead of fixed resolution discretisation to improve the science per flop metric.

While the TRR focuses on an X10 based implementation of this concept for manycore processors, TUM is developing iOMP, an extension of OpenMP with invade, retreat, and infect operations. iOMP will be used to proof the advantages of invasive applications for Scientific Computing on NUMA architectures. In the future, a similar implementation will be investigated for distributed memory systems. Whether MPI is the best choice here is unclear. May be: Clouds as a similar large scale infrastructure with the required resource management.

6.2 Relating Exascale to Parallelism, Power and Energy

Zhiwei Xu (Chinese Academy of Sciences, CN)


License  Creative Commons BY-NC-ND 3.0 Unported license
© Zhiwei Xu

To realize sustained exascale performance for targeted applications, a co-design team needs to explore a design space with millions of options. In addition to benchmarking and meticulous modeling, a third approach is needed: relating performance to architectural contributing factors with a small set of simple equations. These simple equations (rules of thumb) could serve as a first-order approximation to weed out “obviously” undesirable design decisions and to suggest promising research directions. Traditional rules, such as Little’s law, Amdahl’s law and other scaling laws, do not consider power and energy consumption. We propose a simple rule as a starting point for discussion. This equation relates the threads per second performance to number of active threads, Watts per thread, and threads per Joule.

7 Exascale Systems

7.1 Characterizing Application-Architecture Co-Design by Suitability Functions


Vladimir Getov (University of Westminster – London, GB)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Vladimir Getov

Building on previous results this work proposes an abstract model for characterizing the couple application-architecture and a methodology for their co-design. Two sets of parameters are first introduced based on a typical high-end computer architecture. A set of suitability functions are then defined using those parameters. These functions could be used for both optimizing the application-architecture co-design as well as for scalability and comparative performance analysis.

7.2 Software Co-Design for Exascale

Erwin Laure (KTH Stockholm, SE)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Erwin Laure

In order to cope with the challenges of exascale computing, particularly the massive amounts of heterogeneous processing units, the deep memory hierarchies, and the deep and heterogeneous communication facilities, application developers need support in all phases of the application life-cycle, from programming models that allow the construction of efficient, yet portable, applications, over advanced compilation techniques and adaptive runtime environments to online and offline debugging, and performance analysis tools.

Some of the aspects to be tackled include heterogeneous programming models that allow to exploit every bit of parallelism, advanced runtime support through autotuning and dynamic adaptation, and performance tools that not only are capable to handle the enormous data sets





resulting from monitoring peta- and exascale applications but also include a full-system view including components shared with other applications like the interconnect and IO-subsystem.

These tool and method developments need to happen in close collaboration with application developers in co-design teams, as prototyped e.g. by the CRESTA and ScalaLife projects.

8 Exascale Architecture (Design, Execution Models, Power, Reliability)

8.1 Co-Design Challenges of Many-Core Systems-on-Chip

Daniel Molka (TU Dresden, DE)

License     Creative Commons BY-NC-ND 3.0 Unported license
© Daniel Molka





Hardware development is driven by high volume markets (clouds, webserver, etc.) that pay for the R&D. Thus, the influence of the HPC-community on hardware development is limited. Only small changes/additions, that do not jeopardize other areas of application, are realistic. This could include mechanisms to disable/configure certain features (e.g. prefetcher, coherence).

Future HPC systems will likely consist of many compute nodes equipped with manycore processors. Communication characteristics within a node will be totally different from inter-node communication. In order to achieve a high utilization, the fast intra-node data exchange needs to be exploited by software. Therefore a hierarchical approach for programming is necessary, i.e. coarse grained parallelisation to distribute work between the nodes combined with fine grained parallelisation within the (shared memory) node.

Within the nodes cache coherence will probably continue to be the major boundary for scalability. At the moment there is hardly any cooperation of hardware and software in this regard. A lot of knowledge could be extracted from software/runtime regarding the (potential) sharing of certain data. However, the hardware does not use this. Instead operating systems introduce sharing where none is intended (process migration) and the hardware expensively figures out the actual sharing behavior at runtime. A lot of overhead could be avoided if the hardware coherence mechanisms could be bypassed if software can guarantee that the accessed data is not shared.

8.2 Experiences in Co-Design: Tackling the challenges of Performance, Power, and Reliability

Darren Kerbyson (Pacific Northwest National Lab., US)

License     Creative Commons BY-NC-ND 3.0 Unported license
© Darren Kerbyson


In a co-design process two or more factors are optimized in concert to achieve a better solution. Though often considered as the joint design of hardware and software, co-design can taken many forms that can include: algorithms, applications, programming models, runtime systems, and hardware architectures. In addition, as we progress towards exascale systems and applications, the power and resiliency requirements in addition to performance define a

multidimensional optimization space. No co-design process to date has covered all factors in a comprehensive fashion, however some notable cases have addressed a subset of factors and the corresponding trade-offs.

Our own co-design experiences have enabled improvements to be made in performance, energy efficiency, and reliability. In the co-design for performance, we used performance models to determine the best configuration of an application and the best (out of several choices) of architectures that ultimately became the first petascale system – the IBM Roadrunner. In the co-design for energy efficiency, we used dynamic performance models to identify periods of idleness in applications and coupled this with a runtime that could lower power consumption on available resources. In the co-design for Fault-Tolerance, a programming model was extended and used by applications to expose critical data (data required for subsequent recovery from node-level faults) with a runtime system maintaining multiple copies across the system. These experiences will provide a view on the value of modeling for the co-design of future hardware and software.

8.3 Co-Design & Resiliency for Exascale Computing

Stephen L. Scott (Oak Ridge National Lab., US)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Stephen L. Scott

By whatever terms you want to call it – Fault-Tolerance, Resilience, Reliability – this is one broad cross-cutting area where failure has the potential to negatively impact everyone’s pursuit of exascale computing.

Thus it should be considered in the co-design of any hardware, system software, or application targeting exascale computing.

(I will use the term “resilience” to encompass all of the above terms from here forward.)





Today, there seems to be more questions surrounding resilience than there are answers. Certainly part of the problem is the unknown – even to the extent that some dispute that failure will be any more of a problem in exascale than we have today in petascale, or yesterday at terascale. We have trudged along living with unreliable systems to this day due to luck and perhaps more so, because of our own ignorance in recognizing failure versus inconsistency and unknown “issues”.

From past experience, the resilience research community cannot successfully impact computing reliability to a significant degree without directly engaging the wider computing community. Therefore, I would like this session to initiate a dialogue with a broad cross section of folks regarding where the resilience research community should focus its efforts for the greatest impact throughout the co-design process.

9 Application Level Issues

9.1 Co-Designing for Online Auto Tuning

Jeff Hollingsworth (University of Maryland – College Park, US)

License     Creative Commons BY-NC-ND 3.0 Unported license
© Jeff Hollingsworth

For exascale systems, continuous auto-tuning will be mandatory. In the past, it was possible to auto-tune (or perhaps even manually tune) a code for a specific machine. At exascale, it will be necessary for auto-tuning to be a continuous and ongoing activity throughout each execution of the application.

This requirement is driven by several factors. First, the sheer number of threads (on the order of one billion) will mean that dynamic load balancing will almost certainly be required. Second, the presence of energy limits (either in the form of total consumption or thermal throttling due to heat dissipation) will mean that the performance of cores will be dynamically varying. This will mean that the exact performance of the hardware will not be known until runtime. Finally, it is likely that hardware failures and faults will be the norm and not the exception. This will also contribute to dynamic changes the hardware available to run applications.





The applications themselves will also have increasing needs for auto-tuning. For example, Adaptive Mesh Refinement (AMR) techniques naturally have opportunities to expose choices about meshing parameters and frequency of mesh updates.

Limited storage and data migration will likely drive applications to be “super coupled”. Super coupling will mean that not only will multiple physics/chemistry be coupled in a single execution, but previously separate phases of execution such as data staging, analytics, and visualization will be done concurrently with the application execution.

Taken together these trends imply that humans will no longer be able to tune such a system since the reaction time will likely be on the order of milliseconds to seconds and not once per port to a new machine. Thus automation is our only hope.

9.2 Exascale Co-Design – Data

Achim Streit (KIT – SCC, DE)

License     Creative Commons BY-NC-ND 3.0 Unported license
© Achim Streit

Scaling HPC systems and its applications beyond petascale towards exascale computing performance is a large challenge for science and industry today. However not only the raw computing capabilities of such systems and the scalability of the applications need to be addressed: big data also needs to be taken into account in the context of exascale systems.

The data generated by exascale simulations will also be in the exascale regime or beyond and in consequence the analysis of this data will need new paradigms, algorithms, methods and tools as well. Arising questions in this context are: How can data efficiently be transferred from the HPC system to the associated storage hierarchy at the HPC site? To what extent must the storage hierarchy expand with the computing capabilities? Are today’s commonly used technologies and design principles of storage systems mature enough for exascale computing systems? Do the data management software technologies scale with the increasing

demand? How to assure data integrity in light of the large amount of entities (core, memory, disk) in exascale systems?


Focusing more on the users of future exascale systems in the data context brings up the questions of I/O performance to and from the HPC sites (not inside the exascale system): How can users efficiently transfer the presumably also increasing input and output data from remote to the HPC site and vice-versa? Are data transfer and federated data management technologies mature enough for the foreseen large quantities of data to be transferred? Will HPC sites – partly against their will – become data sinks?

As today's and most likely the future HPC ecosystem consists of a heterogeneous set of systems layered in tiers of computing performance, users will always use more than one system during their research lifecycle – due to computing grants, scalability enhancement, or usage of different architectures in multi-scale, multi-physics codes. Consequently one key question is how a federated research data infrastructure looks like? What requirements exist for WAN connections, data movers and federated data management systems? Is it probably easier and faster, if users fedex' disks with large scale data to and from HPC sites?

Finally coming back to the application enabling aspects, one must ask, whether exascale systems will only be usable by an elite set of users and is it enough to focus all efforts only the top of the performance pyramid? Isn't it also mandatory to scale-up the next tier of performance and its users in order to continue the movability of users scaling up their CSE applications – comparable to professional sports vs. grassroot sports (in German: Leistungssport vs. Breitensport).

9.3 Software-Software Co-Design of Applications and Tools

Karl Fürlinger (LMU München, DE)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Karl Fürlinger


Co-Design commonly refers to the synoptic development of hardware and software for a specific application domain. In a similar fashion, benefits can be reaped from integrating various software components which are typically developed in isolation from each other. For example, performance and debugging tools are usually only used when necessary to locate a bug or to tune performance while most applications contain crude performance and correctness indicators (timers, counters) put in place by the developers.

We argue that this situation presents an opportunity for software-software co-design. Historically separate components (the application and the tool) can be turned into a unit that performs the desired function better than a loose coupling would allow. The ad-hoc performance or correctness indicators in applications can be replaced by tried-and-true performance tools technology which supports more advanced performance indicators such as hardware performance counters and has controllable overheads.

For this to be feasible, a lightweight always-on monitoring component has to be developed and an interface to access the data in a standardized way needs to be specified. The resulting performance data can then be made available to a variety of consumers. Firstly, the application itself can benefit from the data in through performance introspection. Algorithms can automatically react to load imbalances or other varying performance conditions. Secondly, other middleware layers and the operators of computing centers can benefit from detailed data of their computing center workloads to drive future procurement decisions, for example.

9.4 How Can We Make Co-Design Really Work?

Cherri M. Pancake (Oregon State University, US)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Cherri M. Pancake

While there are many challenges in moving to exascale, I'm particularly concerned by two issues: (1) ensuring that it will be possible to get sufficient application efficiency to warrant system costs; and (2) coming up with software development models that will allow more than a handful of people to take advantage of the new capabilities.

The workshop topic co-design process is our chance to address the first issue. The only way to ensure that future systems will provide reasonable levels of efficiency on real problems is to engage application developers effectively in the design process. I emphasize the word effectively, because there have been both successes and abject failures with past co-design efforts. As a former anthropologist who has participated on a number of HPC standards groups, I suggest that the most critical aspects of co-design are convening the right groups and ensuring that design truly reflects the issues that each group brings to the table. I would like to discuss ways to involve the “right” users in co-design efforts, as well as how to deal with interpretation and mediation of what are sometimes conflicting viewpoints within a co-design group. Exascale efforts could learn a lot from the experiences of other domains where co-design and standardization already have a long history and are well understood.

My second issue falls under the workshop topics programming models and application level issues. I believe that, rather than relying on our traditional notions of a programming model, it will be necessary to offer software developers a smorgasbord of targeted methods, libraries, and tools. The real challenge is getting users to adopt new methods. Experience at existing scales has shown that they simply are not willing to start from first principles just to use a new model. Instead, users have been attracted to libraries and methods that are clearly related to their application domains, such as higher-level libraries that manage low-level message passing or memory management for them. These provide some level of abstraction – i.e., operations that clearly relate to their application domain – without requiring that they embrace a full scale programming model. They have the added advantage of allowing a single application to combine a mix of techniques. So I think what will be needed for exascale applications is a mix of approaches, where targeted libraries/tools provide higher-level support for different classes of operations such as map/reduce, multiresolution problems, streaming, etc.

Participants

- Arndt Bode
TU München &
LRZ Garching, DE
- Hans-Joachim Bungartz
TU München, DE
- Kent Czechowski
Georgia Inst. of Technology, US
- Karl Furlinger
LMU München, DE
- Hans Michael Gerndt
TU München, DE
- Vladimir Getov
University of Westminster –
London, GB
- Adolfo Hoisie
Pacific Northwest Nat. Lab., US
- Jeffrey K. Hollingsworth
University of Maryland – College
Park, US
- Darren Kerbyson
Pacific Northwest Nat. Lab., US
- Alice E. Koniges
Lawrence Berkeley National
Laboratory, US
- Bettina Kramer
University of Versailles, FR
- Dieter Kranzlmüller
LMU München &
LRZ Garching, DE
- Erwin Laure
KTH Stockholm, SE
- Allen Malony
University of Oregon, US
- John Mellor-Crummey
Rice University, US
- Barton P. Miller
University of Wisconsin –
Madison, US
- Bernd Mohr
Jülich Supercomputing
Centre, DE
- Daniel Molka
TU Dresden, DE
- Wolfgang E. Nagel
TU Dresden, DE
- Cherri M. Pancake
Oregon State University, US
- Michael M. Resch
Universität Stuttgart, DE
- Sabine Roller
German Research School for
Simulation Sciences, DE
- Martin Schulz
LLNL – Livermore, US
- Stephen L. Scott
Oak Ridge National Lab., US
- Christian Straube
LMU München, DE
- Achim Streit
KIT – SCC, DE
- Shinji Sumimoto
Fujitsu – Kawasaki, JP
- Jan Treibig
Univ. Erlangen-Nürnberg, DE
- Felix Wolf
German Research School for
Simulation Sciences, DE
- Zhiwei Xu
Chinese Academy of Sciences, CN

