Science & Technology
Facilities Council

Technical Report
RAL-TR-2012-012

European Exascale Software Initiative: thoughts on software engineering for exascale software development

C Greenough, DJ Worth, LS Chin

August 2012

ISSN 1358- 6254

European ExaScale Software initiative: Thoughts on Software Engineering for ExaScale Software Development

C Greenough, DJ Worth and LS Chin

July 2012

Abstract

This short paper brings together some thoughts on the software engineering needs of ExaScale software development. The work was part of the efforts of Working Group 4.6 (Scientific Software Engineering) of EESI - the European ExaScale Software Initiative [1]. EESI is a *Support Action* co-funded by the European Commission. Its objective is to build a European vision and roadmap to address the challenges of the new generation of massively parallel systems composed of millions of heterogeneous cores which will provide multi-Petaflop performances in the next few years and Exaflop performances in 2020.

Keywords: Software engineering, ExaScale computing

Email: `[christopher.greenough,david.worth,shawn.chin]@stfc.ac.uk`
Reports can be obtained from: `http://epubs.stfc.ac.uk`

Software Engineering Group
Computational Science & Engineering Department
STFC Rutherford Appleton Laboratory
Harwell Oxford
Didcot
Oxfordshire OX11 0QX

Contents

1 Introduction

Software engineering is the process by software is developed. As software becomes more complex and longer lived the importance of these *manufacturing* processes increases. The step from the design and implementation of a simple serial code to one utilising tens of processors was vast requiring different programming models and a whole array of new software engineering tools. As is often the case the systems and development software has lagged far behind the technological developments. Even so, the computational community is already considering systems with thousands of processors. Some applications have reach Petaflop performance and those with sufficient budget are considering ExaScale long before the software tools have caught up.

This short paper brings together some thoughts on the software engineering needs of ExaScale software development. The work was part of the efforts of the Working Group 4.6 (Scientific Software Engineering) of EESI - the European ExaScale Software Initiative. EESI is a *Support Action* co-funded by the European Commission. Its objective is to build a European vision and roadmap to address the challenges of the new generation of massively parallel systems composed of millions of heterogeneous cores which will provide multi-Petaflop performances in the next few years and Exaflop performances in 2020.

Supercomputers are used for computational intensive tasks such as problems involving quantum physics, weather forecasting, climate research, molecular modelling (computing the structures and properties of chemical compounds, biological macromolecules, polymers, and crystals), physical simulations (such as simulation of aircraft in wind tunnels, simulation of the detonation of nuclear weapons, and research into nuclear fusion). A particular class of problems, known as Grand Challenge problems, are problems whose full solution requires semi-infinite computing resources. EESI is looking to provide a computing ecostructure that will support such applications.

The EESI Support Action will :

- Investigate how Europe is located, its strengths and weaknesses, in the overall international HPC landscape and competition
- Identify priority actions
- Identify the sources of competitiveness for Europe induced by the development of Peta/ExaScale solutions and usages
- Investigate and propose programs in education and training for the next generation of computational scientists
- Identify and stimulate opportunities of worldwide collaborations

More information about EESI can be found on the EESI web site: `http://www.eesi-project.eu`.

2 The Scientific Software Engineering Working Group

The core software technologies of today are ill-equipped to handle Peta- and ExaScale systems. One of the major difficulties will be to manage massively parallel systems, composed of millions of heterogeneous cores. The challenge is particularly severe for multi-physics, multi-scale simulation platforms that will have to combine massively parallel software components developed independently from each others. Another difficult issue is to deal with legacy codes, which are constantly evolving and have to stay in the forefront of their disciplines. Scalability and load balancing of scientific applications strongly depend on the quality of the spatial discretisation.

Meshing tools adapted to massively parallel computing, including parallel meshing, mesh healing, CAD healing for meshing and dynamic mesh refinement are still mostly missing software pieces. One example from the field of fluid dynamics serves to illustrate the scale of the problem to run a simulation on one million cores requires a computational mesh with many billions of grid nodes. Most grid generation software is provided by ISVs and is serial. How do we create a mesh with 50+ billion nodes? Assuming we can generate the mesh, how can we partition the grid to run on a million cores? How do we minimise load imbalances?

ExaScale computing does make sense if the entire simulation process is not adapted to ExaScale computing capacities. Thus building a unified Simulation Framework and associated services adapted to massively parallel simulation is mandatory. This includes:

- Meshing tools
- Parallel visualization, remote and collaborative post-processing tools
- Supervising and code coupling tools

Each scientific discipline is faced with different challenges and future software development must solve these problems to enable their research to take advantage of the potential offered by ExaScale computing. In particular, we need to anticipate the bottlenecks we are likely to encounter and plan a software development route that ensures each application chooses the optimal algorithm that best exploits a given architecture for productive science.

One element of the deliberations of Working Group 4.6 is software engineering methods and tools.

3 A Common View

We need to define the scope of this activity and a have a common understanding of our terms. There is no better place to start than by quoting Sommerville (2007):

> Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification to maintaining the system after it has gone into use.

This has not changed as computing systems have grown in power and complexity. The need to apply a systematic and organised approach to software development has not changed. It is still the most effective way to produce high-quality software even though the informal practices of many research groups have lead to world leading software.

The changes of scale from PetaScale to ExaScale may be larger and more complex but the basic software engineering principles remain: requirements, design, implementation, testing and documentation all bounded together by an iteration loop the scale of which is most often determined by the application or the working practices of the developing group. The tools to aid the developer perform each of these steps must reflect the underlying computational model and expose to the programmer only those elements of the programming model that he really needs to see and understand.

4 The Software Engineering Characteristic of ExaScale Systems

The fundamentals of new software development, in software engineering terms, are not going to change radically with a new programming model. The major difference will probably be in the granularity of the process steps and to what these processes are applied. This will be driven by the programming model and by the programming languages available on each system.

The next generation of computing systems the so called ExaScale Systems will have hundreds of thousands of multi-core processors. Processors with tens or hundreds of cores will not be uncommon. Many of these issues were explored (Gropp 2009) with the advent of PetaScale systems and many things have not changed.

We still can view most existing approaches to parallel programming as extensions to existing programming languages. The two approaches currently dominate in scientific computing: OpenMP for a single processing node and the message-passing interface (MPI) for multiprocessors. For systems with thousands to hundreds of thousands of processors, developers typically either use MPI alone (that is, as a Fortran, C, or C++ program that calls MPI-defined routines) or use OpenMP to manage several threads within each MPI process. However, there are still limitations in OpenMP and MPI and the programming models have not changed seriously since their inception. New ideas include the languages developed as part of the DARPA High Productivity Computing Systems program which are designed to alleviate some of these difficulties - including Chapel, Fortress, and X10 - and provide more powerful capabilities have yet to deliver - few systems have production-quality implementations. As a result the associated software engineering tools are primitive and consequently Fortran, C and C++ continued to be the implementation language of choice.

The use of Partitioned Global Address Space (PGAS) programming languages, which let programmers describe the data structure, such as an array or matrix, as distributed across all of a parallel systems nodes has been explored. Out of this work have come Unified Parallel C and Co-Array Fortran. One of the strengths of these languages is their support for distributed data structures and direct access to data on remote nodes through load-store operations. These ideas reduced the need for the programmer to manage his own data structures. However there is still much to be done.

The next phase of hardware developments has brought the GPGPU and the add-on accelerator plus languages such as CUDA and OpenCL but these are really still in their infancy even though heavily used by the community.

In addition, theres been a great deal of research on optimizing and parallelizing compilers. However, these compilers still have trouble optimizing even dense matrix operations for a single thread or process, even for existing languages such as C and Fortran. Ultimately, for these new languages to displace current approaches, they must provide some route to increased performance, whether through new performance-optimization approaches or through hybrid approaches, such as those based on annotations.

Transforming applications from PetaScale to ExaScale systems will require a very large investment in several areas of software research and development (Heroux 2009). Many-cored processor nodes, each with an abundance of parallelism, assembled into vast systems with an increased possibility of system and software faults is the environment of ExaScale computing.

If has been note in numerous roadmaps and scoping studies that: significant scientific results have been achieved in many areas of computational science despite software engineering not because of it. For many, software is a means to an end and not the end in itself. It is becoming more and more apparent to the research community maybe already realised by some communities, in particular the HPC community - that future software will not be, in fact cannot be disposable, it must be sustainable and re-usable. The complexity of the virtual laboratory will be beyond a single mind and lifetime.

It is interesting that the IESP Roadmap spends considerable time on operating systems, compilers and debugging tools for example but little time on programming model, languages, standards and software development tools.

5 Is ExaScale software development different?

The X-stack has been coined by IESP to represent the total computing and software development environment of the ExaScale system.

The section of the IESP road map that has a strong impact on software engineering is that of Development Environments. The development environment will include a whole range of issues: The application development environment is the software that the user has to program, debug, and optimize programs. It includes programming models, frameworks, compilers, libraries, debuggers, performance analysis tools, and, at ExaScale, probably fault tolerance. It is worth noting their comments on programming. Programmability: The applications developers cannot be expected to manage, at a low level, distribution, replication, load balancing, and other issues explicitly in their codes. Complex aspects of distributed services need to be available as high-level APIs to allow end users to optimize their code, perform tuning operations, and improve their applications.

There will be great changes in the resources provided by an ExaScale system and to utilize them effectively the programming models will undergo an evolution. Non-MPI programming models will become common place in ExaScale systems. Data-centric computations - computations in which the data is far more significant and difficult to move - will become increasingly important as simulations begin to include behavioural models and also significant feature extraction (data mining) tasks. Much computing will move from prediction of results and effects to the detection of emergent trends and properties. Programming models will continue to emphasize the management of distributed-memory resources but they will also need to provide a formalism that will enable functional, procedural and declarative approaches to co-exist.

Given the evolution in programming models, we can also expect that individual applications will incorporate multiple programming models. A simple example would be a single application may incorporate components that are based on MPI and other components that are based on shared memory. The particular combination of programming models may be distributed over time (different phases of the application) or space (some of the nodes run MPI; others run shared memory).

Software engineering must reflect these programming models.

6 A Portfolio of Tools

IESP have their X-Stack and so EESI must have its own X-Toolbox (maybe we should consider hte Software Carpentry ideas of Greg Wilson Wilson 2006/2009). In principle we are not looking for the most cutting edge software engineering tools although these maybe the only ones but we need a set of practical and useful tools that make the developers life easier and lead to the development of good quality sustainable software. We can easily build a list of tools in the EESI X-Toolbox:

- IDEs Integrated Development Environments. These might provide an integrated and consistent interface to the following elements.
- Compilers options rich for debugging and performance
- Debuggers capable of selectively monitoring tens of thousands of threads
- System simulators software representations of the hardware seeded with architecture and timing data that could allow estimations to be made about performance
- Performance analysis tools providing information on not only computation performance but memory/cache/register usage and even power consumption.
- Language and architecture intelligent editors that provide auto-completion of standard language elements and APIs and even annotations on potential execution issues
- Static analysis tools (language conformance, complexity, quality etc)
- Automatic documentation tools (e.g. doxygen, robodoc) tools that parse and either prompt the programmer for documentation or generate templates automatically.
- Tools for collaborative working combining SVC systems with develop management SourceForge, CCPForge, Git etc in easy to use forms.

- Version control systems cvs, svn, git etc
- Testing tools tools that will help generate and monitor software testing on the ExaScale.

Much of these will be driven once again by the programming model and the languages provided by the systems. An assumption being made it that the developments will be done by large teams. Clearly this will not always be the case so the tools must be capable and useful to the single user.

Bring together the X-Toolbox under a single environment should be a goal but we must recognised that many IDEs are not used because of the step learning curve they often require and their opaque documentation. IDEs developers dont seem to understand the possibilities of a layered approach. One look at any of the eclipse IDEs or Windows Studio demonstrates this. However integration and interoperability is required for the benefits of the X-Toolbox to be realised.

7 Legacy Software

We will have the problem of legacy software software that has been design for a previous generation of hardware and programming model but still has useful life. This problem has been extensively studied under the Software Modernisation Programme and by many other groups. Post and Kendall (2003) and Decyk and Norton (2001) to mention just a few have graphically described the processes required to migrate legacy software in a language context but similar processes and tools will be required to address to system portability issues.

The main issue must be the balance between some form of re-use/transformation or developing new software.

8 Rewrite vs New

For scientific applications, there may be several reasons motivating a full rewrite:

- change of language, environment, development team,
- change of numerical approach (i.e. mesh-based to particle-based, space-based to frequency based, implicit to explicit,)

In the first case, a full rewrite and incremental evolutions of the code base could in theory lead to the same result, so the choice is more one of cost-vs-risk and HR management issues. In the second case, we replace a solution method that has some advantages and disadvantages by another, which has different trade-offs. Is the science the same, and does it need to be the same? Also, how do we measure the performance increase? For example, if moving from an implicit to an explicit time scheme requires using time steps 100x smaller but the cost per time step is divided by 10, we have still lost an order of magnitude. If this is deemed necessary to go from petaflops to exaflops, we only get 100 out of the 1000x fold increase in performance, so the gain compared to a less-scalable implicit time-stepping may be only one of bragging rights.

Then again, the resulting simplification may unlock other optimization possibilities, and allow further improvements and more science. In a similar fashion, moving from mesh-based to particle-based methods may have many advantages, but though complex physics may be more easily possible, such a move may be require much more computing resources for relatively simple cases.

When using software whose domain of application, validation, and testing includes a broad range of physical resolutions, optimizing for ExaScale by using a different algorithm could actually be detrimental to other applications of the software, while maintaining 2 branches would be unrealistic, not only from a maintainability perspective, but also validation-wise, as each branch would probably be less tested.

This does not mean that we should try to push all legacy applications to the ExaScale level, but that we need to determine whether we recommend focusing on enabling technology for new approaches is a priority, or if we also seek a roadmap for existing applications.

9 The Need for processes and associated tools

It might be that re-writing is the only way in which applications will be able to exploit ExaScale systems. However the weight of legacy software is such that re-use must be addressed. Given a programming model tools need developing that will aid the transformation of existing codes. Although basic re-structing and re-editing (pretty printing) can be achieve if the target language is not very different from the orginal, automatic serious decomposition and re-factoring will probably not be possible. However tools that allow for limited code extraction and transformation should be possible.

If the target language of the new systems in the same or very similar to an original then tools that anotate the source with information on the suitability of of the code may be possible.

10 Verification, validation and testing

The need for these has not changed but these processes are still religated to the background by the need to implement. We have not really learnt the lessons. These steps will reduce the need for major debugging but we are seriously bad at performing these stages. This is not helped by the lack of tools available to those involved in large scale parallel computing.

Even at the PetaScale level there are many application that execute successfully on hundreds of processors yet fail on thousands. It is very difficult to trace the source of the problem in such complex systems.

Basic testing processes such unit and integration testing are used by many teams. These reflect the underlying programming model. Since the complexity of the application in ExaScale computing the need for good testing tools is essential if the developers are not going to spend all their valuable run time debugging.

Verification and validation of basic language elements (string data typing should be a requirement of any new ExaScale programming language) plus interface definitions which can be tested for all library routines should be the starting point.

The developer should not land up debugging the operating system and associated support libraries.

11 Collaboration tools

There is no doubt that the next generation of software will be built by team particularly for ExaScale applications. Collaborative development tools will be required not only to share source code and documentation as we can currently do using svn and Git for example but to share development processes and system configurations we will need better

12 Resourceing Developments

Develop software engineering tools can take considerable resourse both in term of man power and time. Each tools requires as its basis the target programming model and programming language. The requirements of many of these tools are well known but with any change in programming model these willing require refinement.

In Section 6 the basic elements of the X-Toolbox are listed. Each of these could require at least 5 to 10 staff years to design, implement and debug. Even if building on existing tools these will take considerable effort.

13 Conclusions

It was said for PetaScale systems and its hasnt change for the migration to ExaScale systems only the degree has increased (Gropp 2009 with a slight edit) - ExaScale systems arent just bigger versons of the current PetaScale systems. The degree of concurrency as well as increasingly complex processors means that existing alogorithmic and software approaches no longer work. In making use of these vast resources new model of computation will be development and use will be made of co-design. As a consequence for quality sustainable software to be developed these advances must progress hand-on-hand with the development of usable software engineering approaches and tools.

References

[1] The EESI Web site: `http://www.eesi-project.eu/`

[2] I Sommervile, Software Engineering, Addison-Wesley, 8th Edition, 2007. ISBN 13: 978-0-321-31379-9

[3] WD Gropp, Software for PetaScale Computing Systems, Computing in Science & Engineering, Vol 11, 2009

[4] MA Heroux, Software Challenges for Extreme Scale Computing: Going from PetaScale to ExaScale System, International Journal of High Performance Computing Applications September 30, 2009

[5] Norton, D., V. Decyk, Re-engineering legacy mission scientific software, Space 2001 Conference and Exposition Albuquerque, New Mexico, USA, 2001.

[6] D Post and R Kendall, Software Project Management and Quality Engineering Practices for Complex, Coupled Multi-Physics, Massively Parallel Computational Simulations: Lessons Learned from ASCI, Los Alamos National Laboratory, La-UR-03-1274, March 2003

[7] J Dongarra, P Beckman et al., "The International ExaScale Software Roadmap," Volume 25, Number 1, 2011, International Journal of High Performance Computer Applications, ISSN 1094-3420 (latest version at: http://www.ExaScale.org/iesp/IESP:Documents)

[8] G Wilson, Software Carpentry Getting Scientists to Write Better Code by Making Then More Productive, Computing in Science & Engineering, Nov/Dec 2006

[9] JE Hannay, C MacLeod, J Singer, HP Langtangen, D Pfahl and G Wilson, How do scientists develop and use scientific software?, SECSE '09 Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering, IEEE Computer Society Washington DC, 2009

Membership of Working Group 4.6

Chair and Vice Chair Contact details:

Mike Ashworth - STFC - mike.ashworth@stfc.ac.uk
Andrew Jones - NAG - Andrew.Jones@nag.co.uk

Work Group Experts:

Sebastian von Alfthn - HPC algorithms and optimisation - CSC - FI
John Biddescombe - scientific visualisation - CSCS - CH
Ian Bush - HPC algorithms and optimisation - NAG - UK
Iris Christalder - HPC software frameworks - LRZ - DE
Rupert Ford - HPC algorithms and optimisation - University of Manchester - UK
Yvan Fournier - HPC algorithms and optimisation - EDF - FR
Joachim Hein - HPC algorithms and optimisation - Lund University - SE
Mohammed Jowkar - HPC algorithms and optimisation - BSC - ES
Peter Michielse - HPC algorithms and optimisation - NWO - NL
Stephen Pickles - HPC algorithms and optimisation - STFC - UK
Felix Schuermann - Blue Brain project - EPFL - CH
Stephane Ploix - Visualisation - EFD - FR
Christopher Greenough - Software engineering - STFC - UK
Ash Vadgama - Performance Modeling - AWE - UK