European Exascale Software Initiative: Numerical Libraries, Solvers and Algorithms

Iain S. Duff[1,2]

[1] RAL, Oxfordshire, UK
[2] CERFACS, Toulouse, France

Abstract. Computers with sustained Petascale performance are now available and it is expected that hardware will be developed with a peak capability in the Exascale range by around 2018. However, the complexity, hierarchical nature, and probable heterogeneity of these machines pose great challenges for the development of software to exploit these architectures.

This was recognized some years ago by the IESP (International Exascale Software Project) initiative and the European response to this has been a collaborative project called EESI (European Exascale Software Initiative). This initiative began in 2010 and has submitted its final report to the European Commission with a final conference in Barcelona in October 2011. The main goals of EESI are to build a European vision and roadmap to address the international outstanding challenge of performing scientific computing on the new generation of computers.

The main activity of the EESI is in eight working groups, four on applications and four on supporting technologies. We first briefly review these eight chapters before discussing in more detail the work of Working Group 4.3 on Numerical Libraries, Solvers and Algorithms. Here we will look at the principal areas, the challenges of Exascale and possible ways to address these, and the resources that will be needed.

1 Introduction

Computers with sustained Petascale performance are now available and it is expected that hardware will be developed with a peak capability in the Exascale range by around 2018. However, the complexity, hierarchical nature, and probable heterogeneity of these machines pose great challenges for the development of software to exploit these architectures.

It is widely recognized that the major bottleneck in the exploitation of Exaflop computing lies more in the software than the hardware and also that insufficient attention has been paid in the past to supporting efforts in software development. The hardware costs for this forthcoming generation of high performance computers are estimated to be in the order of $200 million with probably at least $20 million a year in electricity costs to run them. It is against this background that we present relatively modest costs for the software effort that will be necessary if we are to capitalize on the thousand-fold increase in computing power over today's fastest machines.

M. Alexander et al. (Eds.): Euro-Par 2011 Workshops, Part I, LNCS 7155, pp. 295–304, 2012.

Of course, we are working on the assumption that we need to exploit future generation machines and recognize that Exaflop computing is on the horizon. Clearly such power (10^{18} floating-point operations per second) is needed for more accurate and complicated simulations. Applications include: multiphysics, multiscale, inverse problems, and optimization.

This was recognized some years ago by the IESP (International Exascale Software Project) initiative and the European response to this has been a collaborative project called EESI (European Exascale Software Initiative). This initiative began in 2010 and submitted its final report to the European Commission with a final conference in Barcelona in October 2011.

We first give a brief review of the main structure of the EESI project in Section 2 before discussing in more detail the work of Working Group 4.3 on Numerical Libraries, Solvers and Algorithms in Section 3. After outlining the topics covered by the Working Group, we then in separate subsections highlight some major issues raised by our investigation. We conclude this short report in Section 4 by highlighting the areas that we need to address to ensure European competitiveness and by summarizing briefly the resources that will be needed.

This note is based on a talk given at the HPSS 2011 Workshop at EuroPar 2011 in Bordeaux. It is particularly noticeable for its lack of references. This reflects somewhat the deliverable of the Working Group although the main report includes references to several web sites. This main report will shortly be available as deliverable D4.5, referenced through our EU contract number EESI 261513.

2 EESI: European Exascale Software Initiative

The European Exascale Software Initiative (EESI) is supported by the EU under the Infrastructure thematic of Support and Collaborative Action (CSA). The 18 months project began on 1 June 2010. The final conference to present our findings was in Barcelona from 11-12 October 2011.

The organizations that are involved in EESI can be grouped into three categories, viz.

- The *contractual partners* are: ARTTIC, BSC, CINECA, EDF, EPSRC, GENCI, JSC, and NCF.
- The *associated partners* are: CEA, CECAM, CERFACS, CMCC, CNRS, CSC, EMBL-EBI, ENES, EPCC, INGV, INRIA, NAG, STFC, Ter@tec, TOTAL, STRATOS, and Univ Edinburgh.
- There is also an active group of *contributing partners*: Univ Tennessee, Tokyo Inst Tech, SNECMA, Airbus, ESF, IACAT, DEISA, LRZ, OeRC, and PROSPECT.

The main goals of EESI are:

To build a European vision and roadmap to address the international outstanding challenge of performing scientific computing on the new generation of computers (multi-Petaflop now and Exaflop in 2020).

The more specific aims are to:

- 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 programmes in education and training for the next generation of computational scientists.
- Identify and stimulate opportunities of worldwide collaboration.

We are working closely with the IESP (International Exascale Software Project) led by Pete Beckman (Argonne and Chicago) and Jack Dongarra (Tennessee and Manchester). The main funding for IESP is from the DOE Office of Science and the NSF Office of Cyberinfrastructure. The IESP involves researchers from the US, Europe (including Russia), China, and Japan and has held several workshops beginning with one in Santa Fe in April 2009. The workshops have also been attended by researchers from Australia, Saudi Arabia, South Korea, and Taiwan.

The goal of IESP is to "Improve the world's simulation and modeling capability by improving the coordination and development of the HPC software environment". The aim of the many workshops held internationally by the project is to *"Build an international plan for coordinating research for the next generation open source software for scientific high-performance computing"*.

The EESI project is split into five workpackages. WP 1 on administration, WP 2 on international networking, and WP 5 on dissemination. The main part of the project is based around eight working groups in workpackages 3 and 4; four based on applications and four based on underlying technologies. We list the Working Groups in Table 1.

Table 1. EESI Working Groups

WP3: Application Grand Challenges WP Chair: Stéphane Requena (GENCI)		
	Chair	Vice-chair
WG 3.1 Industrial and Engineering Applications	Philippe Ricoux (TOTAL)	Jean-Claude André (CERFACS)
WG 3.2 Weather, Climatology and Earth Sciences	Giovanni Aloisio (ENES-CMCC)	Massimo Cocco (INGV)
WG 3.3 Fundamental Sciences (Chemistry, Physics)	Godehard Sutmann (CECAM)	Jean-Philippe Nominé (CEA)
WG 3.4 Life Science and Health	Modesto Orozco (BSC)	Janet Thornton (EBI)
WP4: Enabling Technologies for Exaflop Computing WP Chair: Bernd Mohr (Jülich)		
	Chair	Vice-chair
WG 4.1 Hardware Roadmaps, Links with Vendors	Herbert Huber (STRATOS-LRZ)	Sanzio Bassini (CINECA)
WG 4.2 Software Eco-system	Franck Cappello (INRIA-UIUC)	Bernd Mohr (Jülich)
WG 4.3 Numerical Libraries, Software and Algorithms	Iain Duff (STFC-RAL and CERFACS)	Andreas Grothey (Edinburgh University)
WG 4.4 Scientific Software Engineering	Mike Ashworth (STFC-DL)	Andrew Jones (NAG)

Each Working Group consists of a Chair, a Vice-Chair, and from ten to fifteen experts, chosen for both topical and geographical coverage. In the case of the Working Group on Numerical Libraries, Software and Algorithms, the composition of the Team is shown in Table 2. The input from all these experts should be acknowledged both in the production of the Working Group report but also in providing the base material for my talk in Bordeaux and this subsequent short report.

Table 2. Composition of Working Group 4.3

Iain Duff	STFC/CERFACS	UK	Sparse Linear Algebra
Andreas Grothey	University of Edinburgh	UK	Cont & Stoch Optimization
Patrick Amestoy	ENSEEIHT-IRIT, Toulouse	FR	Sparse Direct Methods, Solvers
Peter Arbenz	ETH Zürich	CH	Eigenvalues, HPC
Jack Dongarra	Tennessee/Manchester	UK/US	HPC, Numerical LA
Salvatore Filippone	Università di Roma	IT	Numerical Software
Mike Giles	University of Oxford	UK	GPU, CFD/Finance
Luc Giraud	INRIA Bordeaux	FR	Iterative & Hybrid Methods
Thorsten Koch	Zuse-Institut Berlin	DE	Combinatorial Optimization
Bo Kågström	Umeå University	SE	HPC, Dense Linear Algebra
Karl Meerbergen	K.U. Leuven	BE	Preconditioners, ExaScience Lab
Volker Mehrmann	TU Berlin	DE	Linear Algebra, HP Applications
Gerard Meurant	ex-CEA	FR	HPC, PDE solution
François Pellegrini	Université de Bordeaux & INRIA	FR	Partitioning
Julius Žilinskas	Vilnius University	LT	Global Opt, Meta-heuristics

3 Numerical Libraries, Solvers and Algorithms

The area addressed by Working Group 4.3 is very much an enabling technology and so inherits the impact and societal benefits of the enabled applications. Indeed we are very much motivated by the needs of applications and our work is critical to the success of these applications even more so in the forthcoming computing regime than at present.

3.1 Main Areas Covered in the WG 4.3 Report

We started our discussions on which topics to include by basing these on the original Colella's dwarves[1] and extensions of these. The original seven dwarves were: structured grids, unstructured grids, fast Fourier Transform, dense linear algebra, sparse linear algebra, particles, and Monte Carlo, but it was soon apparent that these only cover a limited range of the main areas that we felt we should include. After some discussion, we chose the list:

- Dense linear algebra
- Graph and hypergraph partitioning
- Sparse direct methods
- Iterative methods for sparse matrices
- Eigenvalue problems, model reduction
- Optimization
- Control of complex systems
- Structured and unstructured grids

The above list has been ordered according to a software stack where entries further down the stack use those above them in the stack. This hierarchical structure is important when considering the mapping of our algorithms and libraries to the hierarchically structured computers of the new emerging architectures. For example in the stack:

[1] D. Patterson (2005) Colella, Phillip. Defining software requirements for scientific computing.
`http://www.lanl.gov/orgs/hpc/salishan/salishan2005/david patterson.pdf`

- BLAS
- Dense linear algebra
- Sparse solver
- Hybrid solver
- Optimization/Eigenvalues/Control

the BLAS are at the most basic level and are used extensively by dense linear algebra codes that in turn can be used to factorize submatrices in sparse direct solvers. In turn, the solution of the linear system might be effected by using the sparse direct solver within a hybrid solver, for example to solve subproblems in a domain decomposition approach. Finally the linear system may be solved within the inner loop of optimization, eigensystem, or control software. Note that we need not be preoccupied with achieving Exascale performance at every level. It can often be sufficient, particularly lower in the stack to obtain Peta or even Terascale performance. For example, if the BLAS executes on a multicore node at a Terascale level, the execution of the software at the highest level could well be at Exascale.

3.2 Algorithmic Issues

There were several algorithmic issues that were identified in more than one subtopic and we discuss these in this subsection. We should point out that there are a huge number of algorithms in our portfolio, and the most suitable one will depend not only on the functionality and the target architecture. Clearly, the problem will define the general approach but the structure and size of the problem is also of crucial importance, for example is the problem sparse or in some way structured and can we exploit this structure?

We already mentioned the software stack that gave rise to a hierarchy of library calls with consequent multiple possibilities for exploiting parallelism at many levels. However, even with seemingly tightly defined algorithms or kernels there is often scope for a hierarchically structured algorithm that might match well to emerging computer architectures. An example is in my own field of the direct solution of sparse equations. Here there might be an initial dependency on graph partitioning algorithms and then the construction and use of a computational tree where the units of computation are akin to a dense matrix factorization which in turn uses possibly highly-tuned BLAS algorithms. Further levels of parallelism can be obtained from the context of the sparse direct solver. For example, it could be in the inner loop of an eigensystem solver or at each step of an optimization algorithm. This algorithmic modularity also supports the exploitation of heterogeneous systems.

There are several barriers to the efficient exploitation of parallel architectures. One is synchronization where there can be significant inefficiencies if one thread is held waiting for others to finish. This is epitomized by the fork-join construct and much recent work has sought to remove this bottleneck and express algorithms in terms of a task graph (normally a directed acyclic graph or dag). Several powerful algorithms for both dense and sparse linear algebra use dags as their basis for assigning work to processors and scheduling the computation.

Of course, as has long been recognized, even on older generations of machines, the main bottleneck is not the floating-point execution time but rather the cost of moving data to the arithmetic units that can be increasingly costly on modern very non-NUMA architectures with many levels of memory and cache hierarchy.

One of the main algorithmic tricks to reduce this bottleneck is to block the computation so that the ratio of data fetching to arithmetic is reduced. A simple example of this is the Level 3 BLAS. We note that there can be extra costs in doing this, for example data might have to be reordered or restructured or repartitioned with concomitant costs.

This type of blocking or data rearrangement is often at the heart of so-called communication avoiding algorithms which often rely on blocking to reduce the amount of communication. An allied issue is that of communication hiding where strong attempts are made to ensure that any data movement is masked by simultaneous arithmetic processing on other data. Care has to be taken both to ensure algorithms remain stable and to avoid extra synchronization costs. We note that, if data movement and communication can be reduced then less energy may be needed to effect the computation, a potentially major issue when dealing with such high performance computers. The concept of energy aware algorithms has indeed become a big issue in high performance computation.

3.3 Software Issues

In the previous section, we discussed generic algorithmic issues which should be addressed if we are to obtain good performance on Peta and Exascale machines. We now discuss some software issues, most of which have existed for some time but many have been brought into sharper focus by the development of high performance computing.

Interoperability is always an important feature of software libraries and this is even more true in the current regime. The Exascale setting intensifies the need to draw on multiple areas of expertise and to have various toolkits interacting with each other.

The Exascale target is a moving one so that any software needs to have a good support structure so that it can continually be adapted to meet the demands of new architectural features. Also, partly in order to attract users for the software (and this is by no means guaranteed), it really is important to have and be seen to have long term support over a period of several years or even decades.

Any code supported by the European Union should certainly be open source to promote its widest dissemination within the research and educational communities. This will also contribute strongly to the future development and support of the code and enhancement of it from third parties. The issue of licensing is more fraught and has not really been grasped by the EU. A licence like LGPL allows open dissemination but could hamper its use in applications although it does open the possibility of commercializing the code (sometimes encouraged in EU projects).

Above all, the interface and documentation of the codes are of prime importance partly because the environment in which they will be used is getting increasingly complex, not just from the hardware point of view but also in a software context where codes may be used in effecting stochastic approaches or in the solution of inverse problems.

3.4 Fault Tolerance

Fault tolerance is a major issue in Exascale computation as, even with a high chip yield, a billion core machine will quite likely experience the failure of a core at an intervals that potentially will cause problems in application and numerical library codes. There are various estimates of the severity of this but it is universally agreed to be a significant problem. One thing to note is the difference between MTBF (mean time between failures) that could be in the order of hours or less and MTTI (mean time to interrupt) which is when the code might have to take significant action. The order of this depends on hardware support for handling the chip failures but could be in the order of a day and thus will be less likely to require dramatic remedial action. Certainly our hope is that the vendors will be able to give support for automatically recovering from some faults although the detection of these can be as hard as the subsequent corrective action.

At the level of MPI, there has been some discussion of handling fault tolerance and we encourage further work on FT-MPI.

A standard way of guarding against such a failure is to use checkpointing so that the computation can be restarted if a problem is detected. This is, however, expensive and can be difficult to schedule without introducing extra synchronization points. A faster means of checkpointing data, say to FLASH memories, may help. On some computations, for example sparse direct factorization, checkpointing can be a problem but for sparse iterative methods it might be more easily accommodated.

In addition to checkpointing or when checkpointing is not feasible, there are other algorithmic tricks that can be done. Examples are to do a backward computation from the point of failure (possible for the more simple algorithms) or to perform additional computations that can also be very useful in detecting when there is a problem. This is reminiscent of check sum computations when using hand held computers and indeed some suggested approaches are very close to these earlier methods.

3.5 Uncertainty Quantification

As we move to ever more complicated computation sometimes with inexact data, the issue of uncertainty quantification looms high in the desires of application scientists for support from algorithm and code designers. It is, however, very important to recognize that such issues span very many levels. For example there can be problems or uncertainties with: input data, modelling of

physical phenomena, uncertainty in observed data, approximation of continuous by discrete model, solution of resulting equations, and the effect of finite-precision arithmetic.

Many of these are more in the domain of the application scientist although we can and should provide tools for assisting them in this important quest. For example: stochastic optimization and stochastic partial differential equations, the use of mixed-precision arithmetic and refinement, and software for assessing accuracy.

3.6 Programmability

While the core of our numerical algorithms might continue to be in standard languages like C or Fortran probably using MPI for parallel constructs, it is recognized that there are and perhaps need to be further developments to both make the programming more efficient and robust and also to be able to exploit more complicated parallel architectures. Certainly stronger support for multi-level parallelism is needed, perhaps better combinations of MPI or OpenMP and further developments of MPI to include improved collectives, including sparse and non-blocking collectives and stronger support for fault tolerance.

Other developments of programming languages based on a partitioned global address space (PGAS) parallel programming model are also available and may become more widely used. Prime examples of these are UPC and co-array Fortran.

3.7 Floating-Point Issues

It is all very well to develop algorithms that perform well on high performance computers but there is little point in getting an answer quickly if it is wrong! It is thus particularly important to ensure that any new algorithm is stable. This can be an issue in block algorithms of the kind we mentioned in Section 3.2.

In that section, we also emphasized the problem of data movement and clearly a lower precision floating-point number will require less storage than one at higher precision. Thus if we can compute in lower precision but still get acceptable results then the amount of data handled and potentially moved could be less. This has led to the rediscovery and development of algorithms that perform much of the computation in single precision but have some kind of corrective action, perhaps in higher precision, to ensure that the final accuracy is not compromised. An example of this is the use of iterative refinement when computing the solution of linear equations, where only the residual need be computed in the higher precision. This results in mixed-precision arithmetic. On the other end of the scale, some applications require very high accuracy in some parts of the computation so the mixed arithmetic could also involve some computations in extended precision.

A further issue in floating-point arithmetic is the problem of reproducibility, particularly acute when computing in parallel. This problem is mainly caused by the lack of associativity in floating-point arithmetic so that the sequence

of performing the arithmetic operations can influence the result. This sequence could be changed in a parallel environment because of influences outside the program itself and this indeterminacy mitigates against reproducibility. Getting different results for different runs of the same computation can be disconcerting for users even if, in a sense, both results are correct. Giving an estimate of the backward or forward error in the computation may help but it can be useful if the vendor could have a mode of running which would be far less efficient but could give a better possibility of getting a reproducible result. This would also be very useful for debugging purposes.

3.8 Training

Training is crucial in many ways both in order to develop and maintain the skills necessary to develop the underlying mathematics and numerical algorithms and software but also to train potential application scientists to recognize and use the tools so developed. In addition to the mathematics, the former group will need training in using programming models and basic tools, for example in partitioning or the use of basic algebra kernels.

The report also noted the relationship to already existing European initiatives involving training including DEISA, PRACE, and HPC-Europa.

4 Conclusions

The main conclusion of our Working Group was that the work of European scientists is recognized at a high global level and there is much interaction between individuals and teams in Europe and groups from outside Europe, particularly with the USA. This is true in all the domains addressed by our study. However, there are four main reasons why we cannot be complacent with the current situation and why more resources are required if Europe is to maintain its competitive edge.

The *first* concerns the complexity of moving to the Exascale domain. Although we do not know the detail of the hardware that will be available, we can be certain that the level of parallelism will increase significantly, that machines will be more complex and heterogeneous, and that the hierarchical structure of many current Petascale systems will be even more pronounced. Thus, in common with our colleagues in the US and Japan, we recognize that considerably more effort and manpower will be required to even begin to address this complexity and so additional resources will be necessary just to stay still as it were.

The *second* concern is that most of the high level research in Europe is done by small groups some of which are only just of critical mass. Thus support is needed to strengthen such groups to keep them at or above critical mass, both for today's challenges and those in the future.

The *third* concern that is also primarily a problem in Europe is that the networking of the groups is not at a level to sustain European competence at this next stage. Indeed many groups have closer contact with America than with

their peers in other European countries. Thus there is a great need for support with networking.

The *fourth* concern is the lack of long-term funding to support the maintenance of software libraries, including their porting to new hardware platforms. This is extremely important because application developers will not commit to using software libraries if they are not positive that they will remain supported. In the US, the Department of Energy labs have played a major role in parallel software development, and one of the keys to their success has been the fact that they have the continuity of funding and users trust them to continue supporting the software packages they develop. In the European setting, this could be addressed through a co-design centre that would also involve hardware vendors. Their involvement would be particularly important in the development of kernels optimized to the new architectures. This centre would also house experts with extensive expertise in software engineering and parallel computing who could assume the task of maintaining the software base. Thus the centre could address the two main European weaknesses of fragmentation of research effort and long-term support for software.

In our report, estimates of the number of person years for each topic are presented, broken into subareas. From these, we have calculated that a total of around 2000 person years of effort needs to be funded to keep existing groups at a critical mass and to address the aforementioned Exascale challenges. Our request is thus for general support for all the topics mentioned at a level of a little over 10 million euros per year for the 2012-2030 timeframe. We would envisage support through normal research mechanisms: targeted calls, support for students, postdocs, and engineers, for networking and training. We are also proposing a co-design centre at a cost of roughly 4 million euros a year for the centre including an extensive visitor programme.

To put the request for funding into context note that the amount suggested is **small** relative to the costs of Exascale hardware; perhaps $200 million with around $20 million per year in electricity costs!

Acknowledgements. The talk at EuroPar and Sections 3 and 4 of this report were largely prepared from the report of Working Group 4.3. Thus the involvement of all the people listed in Table 2 is gratefully acknowledged.