libHPC: Software sustainability and reuse through metadata preservation

Jeremy Cohen, John Darlington, Brian Fuchs
London e-Science Centre
Department of Computing
Imperial College London
South Kensington Campus
London SW7 2AZ, UK
Email: {jeremy.cohen, j.darlington, b.fuchs}@imperial.ac.uk

David Moxey, Chris Cantwell, Pavel Burovskiy, Spencer Sherwin
Department of Aeronautics
Imperial College London
South Kensington Campus
London SW7 2AZ, UK
Email: {d.moxey, c.cantwell, p.burovskiy, s.sherwin}@imperial.ac.uk

Neil Chue Hong
Software Sustainability Institute
University of Edinburgh
James Clerk Maxwell Building
Mayfield Road
Edinburgh EH9 3JZ, UK
Email: n.chuehong@software.ac.uk

***Abstract*—Software development, particularly of complex scientific applications, requires a detailed understanding of the problem(s) to be solved and an ability to translate this understanding into the generic constructs of a programming language. We believe that such knowledge – information about a code's "building blocks", especially the low-level functions and procedures in which domain-specific tasks are implemented – can be very effectively leveraged to optimise code execution across platforms and operating systems. However, all too often such knowledge gets lost during the development process, which can bury the scientist's understanding in the code in a manner that makes it difficult to recover or extract later on. In this paper, we describe our work in the EPSRC-funded libHPC project to build a framework that captures and utilises this information to achieve optimised performance in dynamic, heterogeneous networked execution environments. The aim of the framework is to allow scientists to work in high-level scripting environments based on component libraries to provide descriptions of applications which can then be mapped to optimal execution configurations based on available resources. A key element in our approach is the use of "co-ordination forms" – or functional paradigms – for creating optimised execution plans from components. Our main exemplar application is an advanced finite element framework, Nektar++, and we detail ongoing work to undertake profiling and performance analysis to extract software metadata and derive optimal execution configurations, to target resources based on their hardware metadata.**

I. Introduction

Software development, particularly of complex scientific applications, requires a detailed understanding of advanced technical problems and an ability to translate this information into the generic constructs provided by a programming language. In undertaking this implementation process, the scientist's knowledge is encapsulated within the code in a manner that makes it extremely challenging for others to extract or extend at a later date. While the code may still continue to run as the original developer intended, the lack of any information about how the software is built and operates can make subsequent maintenance of the application very challenging.

This is likely to have a serious impact on its long term sustainability as other software or systems change around it. In modern heterogeneous and distributed processing environments, loss of domain knowledge within code during the software development process can cause further challenges. Information about code implementation/operation and the ability to use this information to configure software inputs, modify the operation of the software or tailor compute platform selection to the code can play a major role in optimising performance.

Large-scale computing facilities present opportunities to scale applications across much larger numbers of resources to achieve faster processing, more detailed results or to handle larger quantities of data. With the emergence of cloud computing infrastructure services and ever increasing network bandwidth that make accessing remote resources more practical, these large facilities are increasingly likely to be remote and owned by a third-party. We suggest that efficiently targeting code to such systems, which may change frequently, can to some extent be automated with the availability of sufficient metadata. We further believe that the long term use and further development of such software can be aided by preserving explicit information about how software is structured and the thinking behind specific implementation approaches and choices.

Efficiently parallelising code to make use of distributed or multi-core platforms can be complex and solutions may only work optimally on the platform(s) that they have been developed for. If end users can avoid the need to handle such low level issues, there is the potential to simplify things for them while also allowing more efficient, automated selection and configuration of resources. A key element in our approach is the use of "co-ordination forms" – or functional paradigms – for creating optimised execution plans from components. These operators provide a high-level application description language that is formed of both domain-specific constructs tailored to specific scientific fields and more generic functional operators. They offer a rich array of options for specifying application semantics.

Work is being undertaken as part of the libHPC project,

a 2-year UK EPSRC-funded research project. The project team consists of two groups of researchers at Imperial College London – computer scientists with experience in developing and working with computational Grid middleware, software components and cloud computing technologies, and domain scientists with high-order finite element method expertise and experience of applying these techniques across a range of scientific domains. Support to ensure the long-term sustainability and applicability of the project outputs is being provided by the Software Sustainability Institute at the University of Edinburgh [1]. The project also has an element of commercial development support to assist in producing a demonstrator.

In this paper, we describe the work that led to the libHPC approach and discuss how it can support the optimised execution of software across a range of computational platforms. We also show how componentised elements of software can be reused alongside different applications within a specific domain. We then describe ongoing work with our main exemplar application, Nektar++ [2], [3], to develop a finite element preprocessing library that can use hardware metadata to optimise the structure of the discrete matrix systems corresponding to a given finite element mesh.

II. Background

Software components, if sufficiently fine-grained, present the possibility of building applications from a group of code blocks that each provide some element of the end functionality. In modern service-oriented architectures [4] this approach is common and workflow management systems such as Taverna [5], [6], [7] and Triana [8], [9] provide the ability to specify and orchestrate dataflow around a system of services which may be remotely located. Remote services are generally static with an instance anchored at a known fixed location. Metadata (for example, Web Services Description Language [10] in the case of Web Services) defines the interface to these services and the format of data required to communicate with them.

The Imperial College e-Science Networked Infrastructure (ICENI) [11], [12] took a slightly different approach, allowing applications to be built from abstract software components that, rather than being live, deployed instances of some service, are simply high-level metadata descriptions. The component model described in [13] defines separate metadata for a component's interface, meaning and behaviour allowing much more knowledge to be captured than in a traditional interface definition. Middleware services then selected suitable execution resources and concrete component instances, deploying the components, potentially on different nodes in different physical locations, and setting up communication paths between them to enable application execution to take place.

An abstract component description provides enough information about a component's capabilities without representing a specific concrete instance of the component. A hierarchical component tree contains the most abstract component definition at its root and concrete component implementations as its leaf nodes. A linear solver component, for example may be realisable using different algorithms, e.g. Jacobi or BiConjugate Gradient, and for each of these algorithms, implementations may be available that are optimised to specific hardware, e.g. an MPI parallel implementation optimised for use on a cluster with low-latency interconnects and a multi-core x86_64 implementation optimised for local execution on a 64-bit multi-core x86-based CPU.

In theory, this approach offers the possibility of automating the building of custom applications, on-the-fly. In reality, a full component ecosystem with sufficiently large numbers of fine-grained components would be likely to take some time to emerge and mature, and components would initially be much more coarse-grained, even to the point of a component being a complete application. This was considered to be one of the problems preventing wider adoption of the ICENI framework. In the libHPC project, we are aiming to take advantage of component hierarchies and metadata to enable selection of suitable resources from those available shortly before runtime, but we also recognise that "components" may be complete applications or significant elements of applications that can be augmented with additional functionality to optimise specific runtime processes.

In addition to using metadata to maintain information about software and hardware platforms, we are also working to simplify the way that end users define their applications or tasks. Rather than having to write low-level code or scripts, a higher-level means of application description would enable scientists/researchers and other end users to describe in more simple terms what they want to achieve and lower the barrier to entry for accessing complex modern computational infrastructure. A key element of this is the concept of coordination forms as described by Darlington et al. [14] which provide a mathematical approach to describing high-level application constructs. In our work, we intend to process these descriptions into a set of suitable software components and ultimately select concrete component instances that are most appropriate for the available computational resources. The aim is to enable end users to specify application functionality in a simpler manner while still obtaining acceptable performance that leverages tested and optimised software components.

The main exemplar application for the initial phase of work carried out as part of libHPC is Nektar++, a high-order finite element framework under ongoing development by teams at University of Utah and Imperial College London. The software is open source and provides algorithms for high-order spectral/hp methods. Nektar++ is written in C++ and is ideal for this work because it consists of a number of software components providing different methods and capabilities, enabling processing of a wide range of inputs. Nektar++ can run serially or in parallel and takes an XML-based input file providing comprehensive metadata describing the problem to be solved. This is ideal since it provides sufficient information to make choices about the optimal hardware platform for a given run of the application. It also provides a good base to convert a high-level application description to.

III. The libHPC framework

The libHPC framework consists of a number of interacting services that handle the process of taking an end-user's high-level description of the task that they want to carry out on their data and undertaking this task in an optimal manner. A number of intermediate steps are carried out in order for the job to be processed and run. These include converting the user's high-level description into a group of one or more concrete processing tasks and selecting suitable hardware from a pool of available resources. Examples of additional steps that may be carried out include the dynamic addition of optimisation libraries to the base code or pre-processing input files based on data semantics.

The multi-layered abstractions that libHPC proposes ensure that low-level implementation details need not be specified in a job description. This, in turn, helps to ensure that application descriptions are not tied to specific hardware features or library implementations that may change or become unavailable over time. Note that it is the application *description* that is generic. When this description is mapped into a concrete execution plan, it will use native code that will execute on selected hardware, however, each execution may result in selection of different code and/or hardware. The high-level application, which represents the end-user's view, is ensured long-term sustainability since as underlying hardware changes or new features become available, new software implementations can be made available and advertised using their metadata to provide the functionality required by the high-level application description.

We recognise that the complete vision set out by libHPC is complex and will take time to develop. While initial implementation work is underway, the libHPC framework is continuing to evolve and we expect ongoing developments to the framework design and structure as further application types from different domains are investigated.

The main functions and services of the libHPC framework are described below. Not all jobs are required to use all elements of the framework. For example, a developer with coding knowledge may prefer to specify their job through a lower-level scripting approach rather than writing a more abstract coordination forms-based description of their job.

- Coordination Forms/DSL Pre-processor: This service takes a high-level application description based on a combination of domain-specific constructs and more generic control and data flow structures. This description is converted into an intermediate job process definition that describes a set of component types and how they are connected. Later in the process, this information will be further concretised by specifying actual component instances and the resources they will execute on.
- Metadata repositories: Hardware and software metadata repositories provide information about hardware resource specification and software capabilities.
- Job Mapper Service: This service is responsible for resolving application requirements against metadata about available hardware and software. The mapping process converts an abstract application description into a concrete plan of how the application will be run. This includes the actual software service(s) that will be used and the hardware that will be used to run the service(s).
- Job Deployment Service: The deployment service is responsible for provisioning resources with the required software before a job can be run, and for managing the running of jobs. The mapper will provide details of the resources and software to be used, the deployment service then needs to ensure that the required software is present on the hardware resources and if not, it needs to be moved onto the resources and configured before the job can be run.
- Node Service: Every hardware resource that is part of a libHPC framework deployment runs an instance of the node service. This service manages resource metadata and regularly synchronises its status to a metadata repository. The node service is also responsible for handling the running of jobs on the local resource.
- Node Manager Service: The node manager service manages a pool of nodes. These may be statically deployed nodes at a given location or they may be dynamically provisioned cloud nodes. In the case of cloud nodes, the service is responsible for starting and stopping the nodes and may advertise resource metadata for cloud resources that do not yet exist and are provisioned when required.

A. Metadata

Metadata is at the core of libHPC. We focus on two types of metadata. Hardware metadata describing processing resources and other computational hardware and software metadata focussing on software applications, libraries or components. The focus of these two different types of metadata differs in that hardware metadata aims to provide a live view of the status of resources at the current time while software metadata focuses on the preservation of historic information that would otherwise be lost.

B. Software

Software metadata aims to preserve as much of a developer's original design intentions as possible. In addition, general software specification information and hardware requirements are also provided. Representation of this metadata can be a challenge. Some of the data must be available in a machine readable format to allow operation of services that select software and match it to suitable hardware. Other information may be targeted at individuals tasked with updating and maintaining software and may be represented in a human readable, free-form format. The key is to support long-term sustainability and usability of software by storing necessary information to ensure it can be effectively used and maintained.

C. Hardware

Hardware metadata differs from software metadata in that it presents a current picture of the fabric of available hardware

resources, their specifications and their current load. While some of this data will be static, such as that defining the specification of resources, the current state of a resource including its available memory, CPU cores, available storage, number of running processes, etc. changes frequently and needs to be regularly published to a hardware metadata repository or obtained directly from the resource. In the case of Infrastructure-as-a-Service (IaaS) platforms, where resources may not actually be available until they're requested from the cloud platform, metadata can be stored in a repository and a node manager service handles the starting of the resources when they are required.

IV. LibFEMpp - a Finite Element Method Pre-Processor component

In addition to the main elements of the libHPC framework, there is scope to develop various domain-specific components to support and optimise the running of applications. Working with the Nektar++ finite element framework, the first of these components to be developed is libFEMpp. This component takes the form of a support library that can be linked against Nektar++ and uses hardware metadata to optimise matrix-vector multiplication routines for the chosen processing platform, combined with optimisations which can be made based on the connectivity information of a finite element mesh.

Given a finite-element problem represented as a mesh of disjoint elements in two or three dimensions, one generally constructs matrices which represent key mathematical operations locally on each element. Where elements are connected, degrees of freedom are duplicated along faces, edges and vertices according to the connectivity of the mesh. How this duplication is removed leads to a rich structure of operator evaluation strategies; generally one may choose to construct a large sparse matrix representing the global matrix for the entire mesh, or alternatively perform element-by-element matrix multiplications and reconstruct the global solution through a mapping. The chosen approach depends on various properties, including the underlying hardware (for example, CPU architecture, cache size and clock speed).

The aim of libFEMpp is to take a hybrid approach which combines aspects of both evaluation strategies to improve performance. To achieve this, elements are grouped into patches which are then coalesced, removing duplicate degrees of freedom inside each patch. The main purpose of libFEMpp is to automate the choice of the number of groups dependent upon the hardware characteristics of the platform. In particular, we target coalesced matrix sizes to be as close to possible to a given cache size. A number of benchmarking tests have been undertaken on different hardware to see how performance changes when working with matrices of different sizes or structures and this information is used to allow libFEMpp to create the most appropriate patches for optimal performance on a particular hardware platform.

Development of libFEMpp is ongoing. The library is written in C with a C++ wrapper and its functionality is portable to other finite element and mesh-oriented codes, in addition to Nektar++.

V. Conclusion

In this paper, we have described on-going work in creating a framework for enabling domain-developers to express design intentions in a way that can be used to improve both application performance and response to user requirements. On the basis of our experience thus far, we would like to propose that this kind of framework offers a particularly promising approach to optimising performance against user requirements in "pay-as-you-go" Cloud environments which are becoming increasingly a part of research computing. To illustrate this idea, we have described current work to build a finite element support library and we are continuing to to build the various elements of the framework with the aim of producing an end-to-end demonstrator.

Acknowledgment

The authors would like to acknowledge the support of the Engineering and Physical Sciences Research Council (EPSRC) in funding the project libHPC: Intelligent Component-based Development of HPC Applications (EP/I030239/1).

References

[1] The Software Sustainability Institute. http://www.software.ac.uk/

[2] P. E. J. Vos, S. J. Sherwin and M. Kirby. From h to p efficiently: Implementing finite and spectral/hp element discretisations to achieve optimal performance at low and high order approximations. *J. Comput. Phys.*, vol 229, pp. 5161-5181, 2010.

[3] Nektar++ spectral/hp element framework. http://www.nektar.info/

[4] C. M. MacKenzie, K. Laskey, F. McCabe, P. Brown and R.Metz (Eds.). Reference Model for Service Oriented Architecture 1.0. OASIS Standard, October 2006. Available at http://docs.oasis-open.org/soa-rm/v1.0/.

[5] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li, and T. Oinn. Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, vol. 34, iss. Web Server issue, pp. 729-732, 2006.

[6] P. Missier, S. Soiland-Reyes, S. Owen, W. Tan, A. Nenadic, I. Dunlop, A. Williams, T. Oinn, and C. Goble. Taverna, reloaded. In SSDBM 2010, Heidelberg, Germany, 2010.

[7] Taverna – open source and domain independent Workflow Management System. http://www.taverna.org.uk

[8] Triana – open source problem solving software. http://www.trianacode.org/

[9] I. Taylor, M. Shields, I. Wang, and A. Harrison. The Triana Workflow Environment: Architecture and Applications. In I. Taylor, E. Deelman, D. Gannon, and M. Shields, editors, *Workflows for e-Science*, pp. 320–339. Springer, New York, Secaucus, NJ, USA, 2007.

[10] E. Christensen, F. Curbera, G. Meredith and S. Weerawarana. Web Services Description Language (WSDL) 1.1. W3C Note. Ariba, International Business Machines Corporation and Microsoft. 2001. Available at http://www.w3.org/TR/wsdl

[11] N. Furmento, A. Mayer, S. McGough, S. Newhouse, A. J. Field and J. Darlington. ICENI: Optimisation of component applications within a Grid environment. *Parallel Computing*, 28(12):1753–1772. Elsevier, December 2002.

[12] A. Mayer, S. McGough, M. Gulamali, L. Young, J. Stanton, S. Newhouse and J. Darlington. Meaning and Behaviour in Grid Oriented Components. In Proceedings of the 3rd International Workshop on Grid Computing. pp. 100–111. Springer-Verlag, November 2002.

[13] A. E. Mayer. Composite Construction of High Performance Scientific Applications. PhD Thesis. Department of Computing, Imperial College, University of London. 2001.

[14] J. Darlington, Y. Guo, H. W. To and J. Yang. Functional Skeletons for Parallel Coordination. *EURO-PAR'95 Parallel Processing*, pp. 55–69, Springer-Verlag, 1995.