44

A single-code software model for Multiphysics analysis-engine on parallel and distributed computers with the PHYSICA toolkit

P.Chow[1], C.Bailey[2], K.McManus[2], C.Addison[1] & M.Cross[2]

INTRODUCTION

In reality, all engineering processes are multiphysics. It is the simplifying assumptions used previously - because of past numerical algorithms and computing technology - that constrained the models. Now with advances in computer technology, modelling techniques and numerical algorithms (e.g. domain decomposition methods - applied with good success in linear and non-linear solvers, exchange of quantities between non-matching grids, and parallel computations), it is no longer the case. Today's engineers have the tool, in Multiphysics software, to tackle tough engineering problems with interacting physics. Employing it on parallel computers ensures the solution is deliverable in a practical timeframe.

In this paper, we use the PHYSICA toolkit [phy] to provide an overview of a single-code software model strategy of an "*Analysis-Engine*" for Multiphysics simulations on parallel and distributed computers. A mould-ingot casting case that includes analysis of thermal convection, solidification and stress is used to illustrate the approach.

1 Fujitsu European Centre for Information Technology Ltd, 2 Longwalk Road, Stockley Park, Uxbridge, Middlesex UB11 1AB, U.K. email: chow@fecit.co.uk
2 School of Computing & Mathematical Sciences, University of Greenwich, Wellington Street, Woolwich, London SE18 6PF, U.K.
Eleventh International Conference on Domain Decomposition Methods
Editors Choi-Hong Lai, Petter E. Bjørstad, Mark Cross and Olof B. Widlund

MODELLING INTERACTING PROCESSES

Considering a simple geometry sample case from the metal casting process, see Figure 1 for a Lead-Ingot casting. Table 44 gives the physical phenomena and related analyses used in an industry funded mould thickness investigation [C.B]. Individually, there are very good analysis codes to solve each of the physical phenomena. Few have the capability for a number of them together, and fewer still do it all.

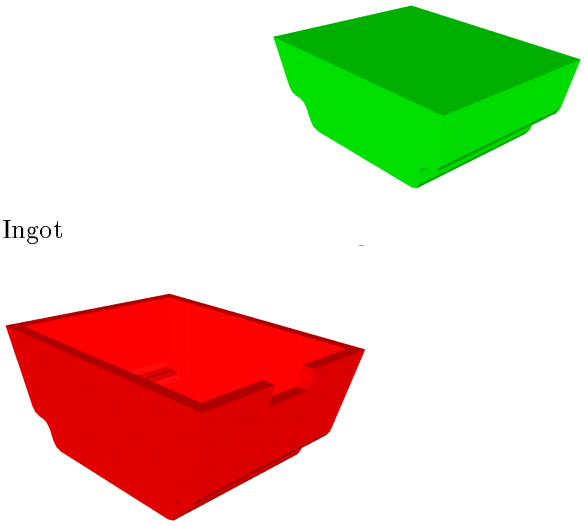

Figure 1 Casting sample case

Table 1 Physics and analysis for the sample case

Physical phenomena	**Type of Analysis**
Thermal convection	Fluid flow analysis
Cooling, solidification (Liquid-Solid)	Thermal and phase change analysis
Residual stress	Solid mechanics analysis
Deformation and gap formation	Contact analysis
Time-dependent	

The analysis elements in Table 44 provide the foundation for a Casting analysis-engine. More complex casting processes could also have:

- Mould filling
- Multiphase (Gas-Liquid-Solid)
- Chemical reactions

- Electromagnetic (mechanism for fluid flow control)
- Microstructures (predicting voids, cracks, etc.)

Putting together a computational analysis-engine for such spectrum of phenomena is not a simple task. It commonly involves a multidisciplinary team or groups using different numerical methods and techniques. The next section gives the PHYSICA toolkit's open software framework that can help with such engine construction under a single-code paradigm.

OPEN SOFTWARE FRAMEWORK

Apart from software related component such as input-output, error handling, etc., a common component in all computational mechanics codes is some kind of a mesh or computational lattice representing the domain for which the unknown variables are solved and holding of relevant information (e.g. material properties, boundary regions). For 3-dimensional space, all these quantities commonly relate to one of four mesh entities: nodes (points), edges, faces and elements. And with the size of models anticipated, meshes into millions of elements and 10 or more unknowns, parallel processing needs to be part of the strategy for such a computational software framework.

Multiphysics modelling projects commonly requires a multidisciplinary team or collaborating groups that may geographically located miles apart. For international projects this can mean participating organizations from different continents of the world. So it is easy to understand the advantages of a common and open software framework for such programs. Also for some collaborative programs code ownership is an important issue. In such cases, open software frameworks that have no provision to address this can have the opposite effects i.e. they can hinder collaboration.

From the above points the following elements have been identified as core and essential components to achieving a good open software framework:

- Data repository system
- Common integration interface
- Safeguard developers' investments

The following subsections give an overview of these matters for a computational mechanics analysis-engine toolkit called PHYSICA [phy].

Software Model

The model for this mesh-based open analysis-engine software framework can be referred to as a Single-Code Component-Based paradigm (single executable). Much of the object-oriented approach has been included in the software design, and for addressing maintainability and extendibility. Common programming language inter-operability is made possible with a Data Repository System (DRS) and a Component Integration Interface (CII), within the framework - DRS and CII are covered in the subsections below.

Table 2 Level of abstraction

Level	**Likely Components**
L4 (Model)	Fluid Flow, Residual Stress, Heat Transfer, Solidification, Contacts, etc., & Coupled Models.
L3 (Advanced)	General Equation for Finite Volume and Finite Element, Solution Procedures, Linear & Non-linear Solvers, Domain Coupling library, etc.
L2 (Standard)	System Matrices, Material Properties, Math Libraries, etc.
L1 (System)	Management Systems (Memory, IO, Database), Parser, Parallel Library, Geometry/Mesh Tools and Libraries, etc.

Table 44 shows the abstraction levels of the present framework. Level-1, the System level, is where system related components are found and most computer experts or system developers contribute, such as inter-communication in parallel processing. Level-2, the Standard level, sees the input from numerical analysis and math library developers. Level-3, the Advanced level, sees the components for numerical methods such as Finite Element and Finite Volume methods, and domain decomposition methods such as in solvers for systems of equations and domain coupling/mapping algorithms - predominately of computational scientists and engineers. Finally Level-4, the Models level, sees the various analysis engines/modules at the component level. Most can perform analysis individually but the potential power comes when model developers couple them together to solve challenging problems. If generic robust "Gluing" technology (coupling) is available, and has been applied to the model components, then Coupled-Model construction with existing component is straightforward. Customized models can be put together and tuned for a particular application.

Data Repository System

Provide the mechanism for storing and accessing of data within the analysis-engine framework. This is where software components would normally take data for processing and put the resulting data on the DRS.

With majority of the data (e.g. field variables) relating to some mesh entities - Points, Edges, Faces and Elements - it is convenient to treat them as mesh-entity objects, and for the DRS to manage them accordingly. Access to the DRS needs to be efficient and inter-operable for popular scientific programming languages. Meeting these objectives limit the options. Therefore, the mesh-entity objects are arrays with each array index location holding the relevant data, e.g. element material type (element object), nodal temperature (point object).

The allocation (and de-allocation) of memory areas for objects (and other data) is also manage by the DRS. A name (text string) is required for allocation, this provide the mean for software components to query objects and data in the DRS.

Component Integration Interface

An N-pin connector model is used to support a generic way of connecting components to the analysis-engine framework, akin to the common 25-pin sockets for connecting external devices to computers. Like the 25-pin sockets not all the pins need to be present, only the ones that are needed by the device. Here the pin equates to function or subroutine calls in the software. The "Pin-function" is given in Table 44 with name association for each pin in the N-pin connector. The last two arguments of the Pin-function are pretty obvious but the Data Repository Arrays (DRA) require explanation.

Depending on the implementation language for the DRS, such as C or Fortran, the arguments for DRA can be different. For a C implementation it can be argument free. This is not possible with standard Fortran77, which lacks dynamic memory allocation (such feature is in some vendors' extensions to the standard but not all and commonly not compatible). Therefore, the seven standard data type arrays (Real, Integer, Character, Logical, Complex Number, Double Real and Double Complex) are on the argument list, such as the example given below.

¡Pin-function name¿ (Ra, Ia, CHa, La, Xa, DRa, DXa, Error Code, Failed Flag)

Table 3 An N-pin connector

`<Pin-function name> ( [DRA], Error Code, Failed Flag )`
⇔ Banner
⇔ Default Settings
⇔ Error Handling
⇔ Read Command Script
⇔ Initialisation
⇔ Before Solution Procedure
⇔ Standard Procedure
⇔ Start of Time-Step
⇔ Sweep Entry (Start, Middle, End)
⇔ Convergence Status
⇔ End of Time-Step
⇔ Users' Own Procedure
⇔ After Solution Procedure
⇔ Output Data
⇔ General Equation Source Terms

Safeguarding Developers' Investments

Using a Component paradigm with both DRS and CII, the code ownership issue is addressed. A "Component Template" that corresponds to the CII's N-pin connector with private and public sections for data and functions is where all components are based. The template is a component itself in the software framework and is referred

to as the User-Template component. The syntax for the Pin-function in the User-Template component is

USER_¡Pin-function name¿ ([DRA], Error Code, Failed Flag)

Hence, by changing the name USER to some other, for example name of the component, a new component is created. Together with a "Component Socket Definition" file (a simple text file with the needed Pin-functions) accompanying the binary file, integration to the analysis-engine framework is straightforward and can be automated in a plug-and-go fashion.

Like any software packages the technology or know-how of the process is encapsulated in binary form, with clear instruction of input/output data, purpose of processing and any limitations. The software component approach is no different from this paradigm, only now is at the component level. Like libraries, it is aim towards integration. But unlike conventional scientific libraries the component can be an entire analysis program under the analysis-engine framework. Thereby, authorship is at the component level but maintain all the rights of a library.

Parallel and Distributed Model

The PHYSICA toolkit use the Single Program Multiple Data (SPMD) paradigm for parallel and distributed processing - each processing element runs the same program operating on a local subset of the model domain. For more details see the reference by K.McManus et al [KMS97].

Domain Decomposition with Message Passing

The partitioning of the model domain (i.e. the mesh) to sub-meshes is done with an overlapping domain decomposition procedure. A local-global numbering scheme is used to map between the local (sub-partition) and global (undivided) meshes. Communication is required between the processing elements to exchange information, predominately for the overlapping regions, to solving the model.

The mesh partitioning is done using a version of JOSTLE [jos], a graph partition code, to divide the mesh. And for the inter-processors message passing (e.g. overlapped regions) is with the portable communication library, CAPlib of CAPTools [cap]. These elements plus others (construct the mesh-graph for JOSTLE, local-global renumbering, etc.) forms the Parallel component.

The objective with parallel processing is to obtain the same solution (within round-off errors) much faster than processing in scalar, else there is no advantage. The total execution time will tell you if you have gained or not. Genuine scalable parallel solvers/codes will guarantee speed-ups over the scalar, but bottlenecks commonly occurring at Input/Output can have a big impact on the overall execution time. This is very much system dependent and until parallel I/O is standardized and commonly available, this bottleneck remains.

The I/O bottleneck is one of two challenges that are actively being researched. The other is dynamic load balancing. In the case of Casting applications, the computational

load imbalance can come from, moving surfaces, contact areas, adaptive meshing, solidification, and possibly all simultaneously. In the case of solidification with thermal convection and residual stress analysis, the workload is totally imbalance even with a balance mesh partition. Because of the physical states - Liquid, Solid, and phase transition - the workload for each is different. The heavy loads are with partitions having the phase transition, these needs to compute both the liquid and solid states and the phase transition.

A Simple Programming Model

With the parallel and distributed model used, a Simple Programming Model (SPM) within the analysis-engine framework is becoming possible. A SPM component can be created building on the Parallel, Geometry and DRS components.

Initially providing functions for performing overlap updates for all mesh-entity objects and global operations (such as summations). These functions takes the object name (those in the DRS), access the DRS and do the inter-communication using functions of the Parallel and Geometry components, to complete the operation. With these simple functions developers can put together codes/components to integrate with the framework that can take advantage of parallel processing. The component is at an advanced stage for inclusion with the analysis-engine framework.

CONCLUDING REMARKS

Demand is increasing in computational mechanics to have the ability to simulate engineering processes with interacting physics - going beyond the simplifying assumptions used in previous models. Putting together such an Analysis-Engine for Multiphysics simulations is not simple, often requiring a multidiscipline team that commonly involves different numerical methods and codes written in different programming languages. Domain decomposition technology is one key component in such analysis-engine.

With large models anticipated parallel processing need to be part of the software strategy for such analysis-engine. An open software framework using a single-code component-based software model can help with the analysis-engine construction. Two essential elements for such open framework are Data Repository System and Component Integration Interface. In the case of multiple working groups, code authorship is at the component level.

REFERENCES

[cap] CAPTOOLS. University of Greenwich (http://www.gre.ac.uk/ captools). London, U.K.

[C.B] C.BaileyComputational modelling of mould design in lead ingot casting. in press.

[jos] JOSTLE. University of Greenwich (http://www.gre.ac.uk/ wc06/jostle). London, U.K.

[KMS97] K.McManus, M.Cross, and S.Johnson (1997) Issues and strategies in the

parallelisation of unstructured multiphysics codes. In *Proceedings of Parallel and Distributed Computing for Computational Mechanics.*

[phy] PHYSICA. University of Greenwich (http://physica.gre.ac.uk). London, U.K.