Software Quality Assurance for the Development of JASMIN Infrastructure

Cao Xiaolin, Zhang Aiqing and Liu Qingkai
Institute of Applied Physics and Computational Mathematics, No. 2, Fenghao East Road, Haidian District, Beijing, China

Keywords: Parallel Program, JASMIN Infrastructure, Software Quality Assurance, Software Development Process.

Abstract: JASMIN is a parallel software infrastructure oriented to accelerate the development of parallel programs for large scale simulations of complex applications on supercomputer. Tens of application programs have been reconstructed or developed on JASMIN. With the rising effort needed to develop and maintain JASMIN, it is very crucial to fulfil software quality assurance. Compared with open source or commercial software development, there are four challenges including parallel computing, higher technical risks etc in the development of JASMIN. A Four-phases-twelve-nodes process are presented and widely used for a seriers new software modules development. These modules meet the requirements arising from application programs and improve performance for adapting new supercomputer.

1 INTRODUCTION

The complexity of application systems and that of supercomputer architectures are providing a great challenge for parallel programming in the field of scientific computing. J Adaptive Structured Meshes applications Infrastructure (JASMIN) is a parallel software infrastructure oriented to simplify the development of parallel software for multi-physics peta-scale simulations on multi-block or adaptive structured meshes (Mo, 2010). JASMIN infrastructure is intrinsically different to traditional libraries because it provides data structures, communication algorithms, load balancing strategies and parallel programming interfaces to shield the details of parallel computing from the users. Based on JASMIN, a user can easily develop parallel programs for complex computers.

Tens of large scale application programs have been reconstructed or developed on JASMIN. It is portable for personal computers, high performance clusters, and massively parallel processing computers. On JASMIN, after the program is rewritten on a personal computer, it can successfully run on parallel machines where JASMIN is installed. These programs are suitable for different numerical simulations arising from multi-material hydrodynamics, radiation hydrodynamics, neutron transport, hydrodynamics instability, laser plasma interactions, materials science, climate forecasting, and so on. Many of these programs can efficiently use thousands of processors on TianHe-1A supercomputer (Yang, 2011).

JASMIN has been developed by a parallel computing team including more than ten members since 2004. JASMIN is written in four languages C++, C, Fortran 90, and Fortran 77. It has three layers architecture with hundreds of classes. There are six hundred thousand lines of code in JASMIN V3.0 in 2013. With the rising effort needed to develop and maintain JASMIN, it is very crucial to Software Quality Assurance (SQA).

SQA is necessary to do so in a systematic and quantifiable way(Aiftimiei, 2012). The application of such an approach is combined in the term Software Engineering, the application of the engineering approach to enhance the development process as well as the quality of the resulting software (Lichter, 2010). Since there is no technique of software engineering which can be applied well to tackle all problems, these techniques have to be adapted to specific domains and kinds of software. There are for example special architectures like Enterprise Java Beans for database applications. OpenFOAM (2013) does make use of particular design processes or software project management techniques to enhance the development process.

This paper shows that JASMIN software projects in particular face challenges beyond those regularity faced in common commercial software. These challenges need to be addressed in the software architecture and software development process. A Four-phases-twelve-nodes process for the development of new software module in JASMIN is described. It has been proved that the software development process is very useful to enhance the quality of JASMIN.

2 OVERVIEW OF JASMIN

2.1 Design Objectives

The programming challenges mainly arise from two types of increasing complexity. The first is the computer architecture. The memory wall is still the key bottleneck for realistic performance; the increasing number of cores in each CPU increases the seriousness of this bottleneck. The data structures should be matched to a cached-based memory hierarchy. However, aggregation techniques are indispensable for the construction of computers. More and more cores are integrated into each CPU, and more and more CPUs are clustered into each computing node, and hundreds or thousands of nodes are interconnected. Parallel algorithms should have sufficient parallelism to utilize so many cores. The design of such data structures and parallel algorithms are too professional for the application users.

The second complexity is the application system. Large scale simulations are mainly used to study the characteristics of complex systems. With the increasing of computer capabilities, the complexity of the application system should also increase simultaneously. However, the increasing complexity brings many new problems. Firstly, the fast parallel algorithms are usually required for the solution of large scale discrete systems. Though many such algorithms have been studied, they are seldom mature enough for realistic applications. Secondly, the dynamic variations of physical characteristics often lead to serious load imbalance, challenging the parallel efficiency across thousands of processors. Thirdly, complex systems are often coupled with many small systems and such tight coupling often leads to irregular communication issues. The fast parallel algorithms, load balancing strategies and irregular communications together increase the difficulty of programming for such systems.

JASMIN infrastructures(MO, 2009) are designed toward solving such challenges. The main objective of JASMIN is to accelerate the development of parallel program for large scale simulations of complex applications on parallel computers. The basic ideas described as following:

- **Hides** parallel programming using millons of cores and the hierarchy of parallel computers.
- **Integrates** the efficient implementations of parallel fast numerical algorithms.
- **Provides** efficient data structures and solver libraries.
- **Supports** software engineering for code extensibility.

2.2 Software Architecture

Figure 1 depicts the three layers architecture of JASMIN. The bottom layer mainly consists of modules for high performance computing for SAMR meshes.In this layer, modules are grouped into three sub-layers. The base sub-layer, Toolbox, contains the essential C++ classes for parameter input, memory management, restarting, and input/ output interfaces with the HDF5. The second sub-layer contains two modules, Patch Hierarchy and Patch Data, for data structures and one module, Communications, for communication and load balancing strategies. The top sub-layer, Mesh adaptivity, contains the components for the local mesh refinement or coarsening.

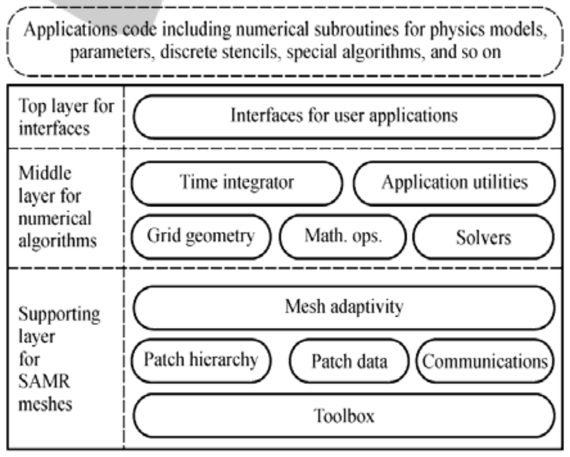
Applications code including numerical subroutines for physics models, parameters, discrete stencils, special algorithms, and so on
Top layer for interfaces
Interfaces for user applications
Middle layer for numerical algorithms
Time integrator
Application utilities
Grid geometry
Math. ops.
Solvers
Supporting layer for SAMR meshes
Mesh adaptivity
Patch hierarchy
Patch data
Communications
Toolbox

Figure 1: Software architecture of JASMIN.

The middle layer of JASMIN contains the modules for the numerical algorithms shared by many applications including computational geometry, fast solvers, mathematical operations on matrix and vectors, time integration schemes, toolkits, and so on.

The top layer is a virtual layer consisting of C++ interfaces for parallel programming. On the top of this layer, users can write serial numerical subroutines for physical models, parameters, discrete stencils, special algorithms, and so on; these subroutines constitute the application program.

2.3 Structure Meshes Supported

Figure 2 depicts the user interfaces for applications programs. The left lists the main classes and strategy classes used by the application programs. The right lists the main modules which should be implemented by the user. The left side of the figure represents JASMIN interfaces for parallel programming, whilst the right hand side shows the user application program. On the left, the class algs::HierarchyTimeIntegrator <DIM> represents the time integration on the patch hierarchy. It should be called by the user main program. In JASMIN, this class depends on the time integration strategy class algs::TimeIntegratorLevelStrategy<DIM>.

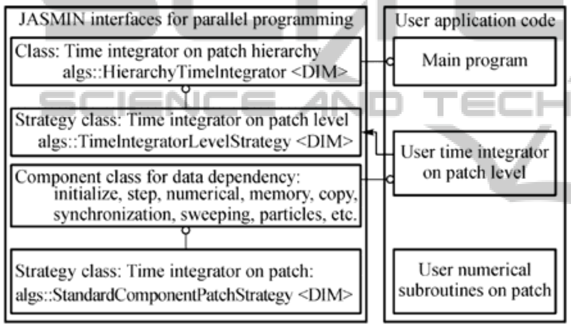
JASMIN interfaces for parallel programming
Class: Time integrator on patch hierarchy algs::HierarchyTimeIntegrator <DIM>
Strategy class: Time integrator on patch level algs::TimeIntegratorLevelStrategy <DIM>
Component class for data dependency: initialize, step, numerical, memory, copy, synchronization, sweeping, particles, etc.
Strategy class: Time integrator on patch: algs::StandardComponentPatchStrategy <DIM>
User application code
Main program
User time integrator on patch level
User numerical subroutines on patch

Figure 2: User programming interfaces of JASMIN.

The user must implement a subclass of this strategy class on a patch level for application program. In return, this implementation should use the integrator components for the descriptions of neighboring relationships. These components cover various data dependencies which occur in different phases of a numerical simulation such as variable initialization, time stepping, numerical computing, memory management, patch data memory copy, synchronization between neighboring patch levels, parallel sweeping, particle computing, etc. Most components contain not only communication but also the numerical computing. Each component calls a strategy class algs::StandardComponent-PatchStrategy<DIM>. The user should implement a subclass of the strategy class at the patch level. In the subclass, the user must implement the numerical subroutines representing the numerical computing on a patch.

3 SOFTWARE QUALITY MANAGEMENT

3.1 Challenges

Scientific software projects (Chhabra, 2010; Jalender, 2012) can be distinguished by characteristics that clearly separate them from other development projects like open source(Franch, 2013) or commercial software development processes. For JASMIN infrastructure, there has fundamental features described as follows:

- **Parallel Computing.** Parallel programs based on JASMIN are developed for multiple cores which go up to massive parallelism in peta-scale supercomputer. Since these codes are both, more complex to implement as well as more difficult to debug and maintain, simulation software, which has to implement parallel functionality incorporates additional technical difficulties.
- **Changing Requirements.** The software architecture has on one side to support efficient algorithms and implementations. On the other side it has to be easily extensible to accommodate for new functional requirements arising from application programs and performance improvement for adapting new generation supercomputer like ten-peta-scale supercomputer.
- **Backward Compatibility.** Subsequent versions of JASMIN need provide the same programming interface that previous versions do, programs written against the previous version will be able to compile and run with the new version. When new version of JASMIN adapted for new generation supercomputer is installed, application programs without any change can run efficiently for large scale simulation.
- **Higher Technical Risks.** When new scientific and/or mathematical methods are implemented one cannot guarantee the feasibility of the problem until nearly complete development of the software. In the case of simulation software the need for parallel computing enlarges the technical difficulties of the software development.

3.2 Software Development Process

A preliminary design process model for the development of scientific software is presented. This model takes the particular challenges of JASMIN in account and promises to tackle these challenges through the combination of an agile approach and stringent application of rules to keep the process manageable.

Table 1: Four-phases-twelve-nodes process for the development of new software module.

Phase	Node	Objectives
Create task (why,what)	Requirements analysis	Real problem in user program; functionality, performance improvement
	Conceptual definition	Concise scientific concepts, clear academic statement
	Architecture design	Systematic sketch; main function; external interfaces
Design task (how)	Test design	Define test cases
	Implement design	Module architecture; data structure, algorithm
	Strategy assessment	Confirm scientific and technical feasibility
Complete task	Coding	Write module code; run and debug code
	Testing	Write test code; do unit and Integration tests
	Version control	Manage source code and test cases.
Apply	Small Release	Do acceptance tests; form installation package.
	Deployment	Make software available on user supercomputers
	Maintenance	Correct faults, upgrade version

The process model is called Four-phases-twelve-nodes(FPTN) process for the development of new software module in JASMIN. There are two kinds of development. One is developing new functionality in order to meet the requirements arising from application programs. Another is performance improvement for adapting new supercomputer. The whole development of JASMIN in each year is consisted of a series of new software modules implemented by small team including 1-3 members. Therefore, the quality of a single module is increasingly vital.

3.3 Example

A Fast multipole method (FMM) solver(Cao, 2011) is a good example for the development of new software module. These main features of the FPTN process will be explained by the development of FMM solver.

3.3.1 Create Task

NODE1-Requirements Analysis. It is necessary to develop parallel code including FMM algorithm in various application fields. Examples include electromagnetic scattering, celestial mechanics and dislocation dynamics. But the lack of efficiently parallelized and user friendly software libraries has hindered the wide-spread use of the FMM algorithm. Then, the analysis and summary of the FMM algorithms and its three applications were described in requirement document. Furthermore, the parallel performance and usage was briefly given.

NODE2-Conceptual Definition. A parallel solver named “FMM solver” was presented. FMM Solver encapsulates the commonness for various applications. It supplies users with abstract interfaces required to implement the individuality with serial mode. The commonness contains distributed storage of multi-levels, intra-level and inter-level data communication, and arrangement of computation etc. The individuality contains various expansion and translation operators. Some concepts in FMM solver, for example M2M, M2L, L2L translation etc, were demonstrated.

NODE3-Architecture Design. The FMM solver has been designed by classes solv::FMMSolver and solv::FMMSolverPatchStrategy. The former creates FMMH, implement algorithms of FMM solver. It includes three public function: initialize-SolverState(), setCoeffID(), and solveSystem(). The later define individuality part as abstract method by using many C++ interfaces: Multi2-MultiShift(), Multi2LocalShift(), Local2Local-Shift().

3.3.2 Design Task

NODE4-Test Design. One test case was designed. It is motion of charged particles in electrostatic field. The case consider a two-dimensional physical model which consists of a set of N charged particles with the potential and force obtained as the sum of pair-wise interactions from Coulomb’s law. The calculation of forces in two ways: via the FMM solver and via direct method. The two calculations were used to compare the accuracy of FMM solver.

NODE5-Implement Design. The main data structure including FMMH, M-coefficients etc was described. The four key algorithms of the FMM solver have been designed. The solver mainly implements the commonness part. (a) Data management on multi-levels in hierarchy: construction, storage, allocation, specification of inter-level and intra-level data dependence relation among cells. (b) Common operations on upward pass and down-ward pass: inter-level and intra-level data communication, arrangement of all translation among cells.

NODE6-Strategy Assessment. These documents of five nodes were reviewed and discussed in order to confirm scientific and technical feasibility. Then, a small team include 3 members was formed for implementing FMM solver, programming test case, real application.

Table 2: Parallel performance with fixed problem size.

Cores	particles	levels	Time(s)	Efficiency
1	1.0×10^6	6	93.83	1.00
4	4.0×10^6	7	98.14	0.95
16	1.6×10^7	8	100.71	0.93
64	6.4×10^7	9	101.12	0.93
256	2.5×10^8	10	101.29	0.92
1024	1.0×10^9	11	103.63	0.90

The other 6 nodes are similar to some classical software development process. The main difference is doing acceptance tests in node10. The tests consist of correctness test and parallel performance test based on the test case in designed in node 4 and implemented in node 8. The correctness test was performed by comparing the accuracy of FMM solver to those of the direct method. For testing the parallel efficiency, we fix problem size on single processor with q=10, and vary the number of processor cores from 1 to 1024. Scalability in table 2 was demonstrated with a parallel efficiency above 90% on 1024 processors.

The FMM solver was applied in a 3-D dislocation dynamics code to calculate interaction force of dislocation lines. A physical model including 30 million dislocation line segments in 128×128×128 cells has been designed for testing parallel efficiency of the FMM solver with 8 levels. We fix problem size, and vary the number of processor cores from 128 to 1024. For p=1024, the parallel efficiency is about 80% compared with p=128.

4 REAL APPLICATION

Since 2013, a series of new software modules have been developed in JASMIN infrastructure based on FPTN process. These modules enhance the functionality and improve the performance of JASMIN with upgrading version from 2.0 to 3.0. Some real applications of JASMIN 3.0 are described as follows.

4.1 Structure Meshes Supported

Figure 3 depicts different structured meshes which JASMIN can support. The first is the patch-based uniform rectangular SAMR mesh where three patch levels are given and are colored with red, green and blue respectively. Usually, a SAMR mesh can contain several patch levels and each level may contains many patches. The finer patch level covers a local region of the next coarse patch level. In Fig. 3, each box represents a region of a patch. The second example is the single block deforming mesh, and the third is the multi-blocks deforming mesh for which nodes always move toward interesting local regions in the process of simulation. In the two or three dimensional geometry, each cell in a mesh is quadrangular or hexahedron respectively. Two blocks should conformingly contact each other along the boundaries. The fourth is the particle mesh for which particles can randomly distribute across the cells of a uniform rectangular mesh. The fifth is the curvilinear mesh suitable for the simulation of climate forecasting. The final example is the three-dimensional body-fitted multi-block deforming mesh for the computation of fluid dynamics.

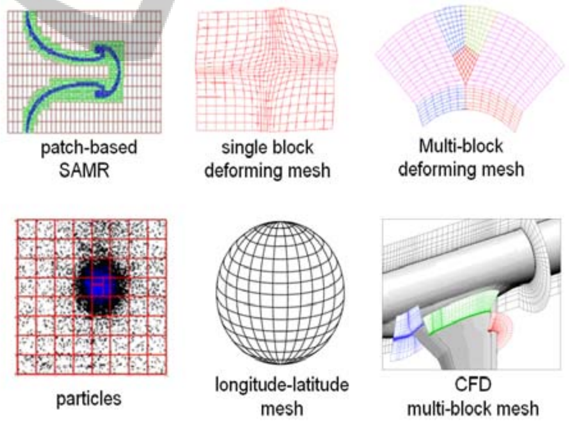

Figure 3: Typical meshes supported by JASMIN.

4.2 Large Scale Simulation

Table 4 shows four large-scale data sets of real application. They have been generated by four application programs reconstructed or developed on JASMIN. These programs have been run on tens of thousands of cores with several or tens of hours on peta-scale supercomputer.

LARED-P is a three-dimensional program for the simulation of laser plasma intersections

Table 3: Simulation scale and parallel performance.

Program	Simulation scale	Efficiency
LARED-P	36000 cores, 2.0×10^{10} particles	45%
LARED-S	32768 cores, 1.6×10^{8} grids	52%
LAP3D	16384 cores, 2.1×10^{9} grids	50%
JEMS-FDTD	30000 cores, 6.1×10^{8} grids	62%

using the method of Particle-In-Cell (Pei, 2009). Left upper map in figure 4 shows the distribution of particles in a snapshot and the related volume rendering of laser intensity. LARED-S is a three-dimensional program for the simulation of radiation hydrodynamics instabilities occurring in the process of radiation driven compression explosion. Right upper map in figure 4 shows the result of the Richtmyer-Meshkov(R-M) instability among materials interface.

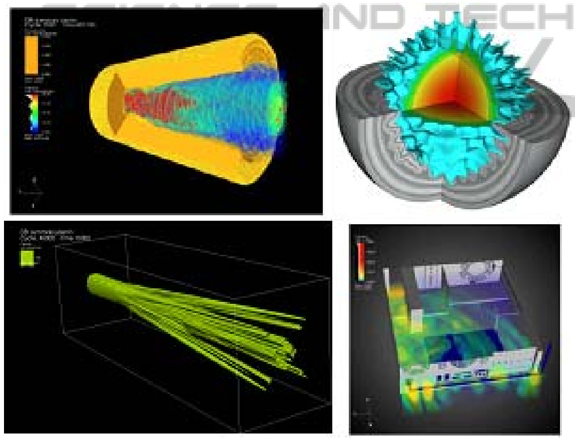
Figure 4: Simulation results of four application programs.

LAP3D is a three-dimensional program for the simulations of filament instabilities for laser plasma intersections in the space scale of hydrodynamics. Left bottom map in figure 4 shows isovalue contour of laser intensity. JEMS-FDTD is a three-dimensional Electronmagnetic solver-finite difference time domain. Right bottom map in figure 4 shows a simulation for high power electromagnetic pulse couples into computer box through small apertures and slots.

5 CONCLUSIONS

A Four -phases -twelve –nodes process for the development of JASMIN infrastructure is described. This process takes the particular challenges of developing scientific software in account. It is widely used for a series new software modules development to keep the software quality of JASMIN manageable. A developing management information system has been customized and installed for these tasks involving tens of persons. A developing environment including code inspection, automatic test system is optimized in order to adapt scientific software efficiently.

ACKNOWLEDGEMENTS

This work was under the auspices of the National Natural Science Foundation of China (Grant Nos. 61033009), the National Basic Key Research Special Fund (2011CB309702) and the National High Technology Research and Development Program of China (863 Program) (2012AA01A309).

REFERENCES

Aiftimiei, C., Ceccanti, A., Dongiovanni, D.,,and Giacomini, F., 2012. Improving the quality of emi releases by leveraging the emi testing infrastructure. *Journal of Physics: Conference Series*, 396(5).

Chhabra, J. and Gupta, V., 2010. A survey of dynamicsoftware metric. *Journal of Computer Science andTechnology*, 25:1016–1029.

Franch. X. and Susi, A., 2013. Managing Risk in Open Source Software Adoption. *Proceedings of ICSOFT 2013, Reykjavik, Iceland*, 29-31.

Jalender B., Govardhan A., Premchand P., 2012. Designing code level reusable software components. *Int. J. Software Engineering & Applications*. 3(1): 219-229.

Lichter A., Hoffmann V., 2010. Processes and Practices for Quality Scientifc Software Projects. *Proceedings ofWASDeTT 2010.*

Mo Z. Y., Zhang A.Q., 2009. User's guide for JASMIN, Technical Report . *https://www.iapcm.ac.cn/jasmine*.

Mo Z. Y., Zhang A.Q., 2010. JASMIN:A parallel software infrastructure for scientific computing. *Front. Comput. Sci. China*. 4(4): 480-488.

OpenFoam, 2013. http://www.openfoam.org/.

Pei W. B., Zhu S.P., 2009. Scientific computing in Laser Fusion. *Physics* (in Chinese), 38(8): 559-568.

Pryscilla, M.D., Ana, C.O., 2013. Improving Quality in Agile Development Processes. *Proceedings of ICSOFT 2013, Reykjavik, Iceland,* 29-31.

Yang X. J., Liao X. K., Lu. K., 2011. The TianHe-1A supercomputer: Its hardware and software. *J. of Computer Science and Technology*. 26(3): 344-351.