IDE-JASMIN
An Interactive Graphical Approach for Parallel Programming in Scientific Computing

Liao Li, Zhang Aiqing, Yang Zhang, Wang Wei and Jing Cuiping
Institute of Applied Physics and Computational Mathematics, No. 2, East Fenghao Road, Beijing, China

Keywords: Parallel Programming, Integrator Component, Integrated Development Environment, Structured Flow Chart.

Abstract: A major challenge in scientific computing lays in the rapid design and implementation of parallel applications for complex simulations. In this paper, we develop an interactive graphical system to address this challenge. Our system is based on JASMIN infrastructure and outstands three key features. First, to facilitate the organization of parallel data communication and computation, we encapsulate JASMIN integrator component models as user-configurable components. Second, to support the top-down design of the application, we develop a structured-flow-chart based visual programming approach. Third, to finally generate application code, we develop a powerful code generation engine, which can generate major part of the application code using information in flow charts and component configurations. We also utilize the FORTRAN 90 standard to assist users write numerical kernels. These approaches are integrated and implemented in IDE-JASMIN to ease parallel programming for domain experts. Real applications demonstrate that our approaches for developing complex numerical applications are both practical and efficient.

1 INTRODUCTION

"J Adaptive Structured Meshes applications Infrastructure" (JASMIN) is an infrastructure for structured mesh and SAMR applications for numerical simulations of complex systems on parallel computers (Mo et al., 2010). Tens of large-scale numerical applications have been ported or directly developed on JASMIN. Many of these programs can efficiently utilize thousands of processors. Using JASMIN, numerical applications developed on personal computers can run on large-scale parallel computers efficiently without any modifications.

JASMIN facilitates problem solving with encapsulation of parallel computing and communication in a wide variety of application domains used in science and research. JASMIN provides patch-based data structures and parallel programming interfaces to shield the details of parallel computing from the users. Object-oriented component design techniques (Parker, 2002) are used for capturing these characteristics of complex parallel operations (Sarkar et al., 2009). JASMIN deals with portability and operating system issues, data access and manipulation, respectively. Using JASMIN, users need implement several subclasses of strategy classes in C++ language though they mainly focus on writing numerical subroutine in FORTRAN.

The limited popularization of JASMIN is explained by the fact that most application domain scientists still prefer to code their programs using FORTRAN other than unacquainted high level computer languages. Understanding and writing object-oriented code in C++ hinder the domain researchers from developing applications on JASMIN.

An integrated development environment (IDE) facing JASMIN, named with IDE-JASMIN, is developed to resolve these problems. IDE-JASMIN provides an interactive graphical approach for parallel programming. The idea of parallel design combined with sequential coding is achieved in the visual programming environment.

The organization of this paper is as follows. First, we briefly describe the design goals of IDE-JASMIN. Then, we detail the design and implementation of the key parts including parallel component models and structured flow chart editor.

In section 4, the implementation of the integrated development environment is presented. In section 5, we introduce a three-dimensional program for the simulation of laser plasma intersections, which is developed on IDE-JASMIN.

2 DESIGN GOALS

To support rapid design and implementation of parallel numerical applications on JASMIN, IDE-JASMIN is designed as a visual programming environment and provides a rapid programming approach based on component configuration and assembling techniques. It is designed with the following features:

- Easy development of parallel numerical applications for domain experts by configuring and assembling parallel integrator components. Users break down their complex algorithms into smaller and manageable parts, configure and assemble corresponding integrator components through intuitive GUI, without worrying about parallel computing details.
- Easy creation and reuse of configured integrator components. Users only need to create an integrator component from an appropriate component model and use it throughout the application.
- Powerful code generation to generate tedious part application code and free users from unnecessary exposure to complex features of C++ language. All C++ part as well as numerical kernel interfaces are automatically generated, users only need to write simple numerical kernels in FORTRAN 90 language.
- Versatile tools to facilitate the whole application development. IDE-JASMIN integrates FORTRAN editing, code compiling and debugging, parallel execution as well as visualized data analysis features into a single system.

3 DESIGN FEATURES

IDE-JASMIN is an integrated development environment tailored to the needs of high performance numerical application developers. The IDE provides a wide variety of component models to describe a computational process and enable users to build their numerical applications by assembling these components. In the following subsections we introduce our component models and component-assembling approach.

3.1 Parallel Integrator Component Model

Parallel integrator components are proposed in JASMIN to encapsulate parallel computing details on patch-based data structure for structured mesh applications (Mo, 2009). These components cover a wide range of parallel computing patterns including communication and numerical computing. Users have to subclass in C++ language the *algs::StandardComponentPatchStrategy* strategy class and implement several abstract interfaces. In these interfaces, users have to manually fetch data fields via Patch object and pass them to numerical kernel often written in FORTRAN language. This approach involves tedious amount of coding and often error-prone for domain experts, who usually are not familiar with complex object-oriented C++ language.

IDE-JASMIN encapsulates JASMIN parallel integrator component models as configurable user interface elements. Some frequently used components are listed below:

- *InitializeIntegratorComonent:* initialize variables
- *NumericalIntegratorComponent:* execute numerical algorithms on specified variables
- *DtIntegratorComponent:* compute time step value
- *MemoryIntegratorComponent:* explicit allocate and deallocate memory for variables
- *CopyIntegratorComponent:* copy values between variables
- *ParticleCommComponent:* communicate data in particle simulations

In IDE-JASMIN, we provide several types of GUI for configuration of these component models. For example, we provide a communication configuration GUI where user can select which variables shall be communicated and how to communicate. We also provide a computation GUI where user can define numerical kernel interfaces and select which variables shall be passed to the kernel. When configuring integrator component models using our GUI, users do not have to write any code hence knowledge of C++ language is not required.

In fact, creating and using a parallel integrator component in IDE-JASMIN needs only three simple steps:

1. Select an appropriate component model
2. Configure necessary attributes of the component
3. If necessary, define numerical kernel interface and set variables to pass to the kernel.

3.2 Structured Flow Chart Editor for Assembling Integrator Components

Structured flow chart (SFC) is often used by programmers as a tool to design and visualize algorithm logic. A flow chart is the graphical representation of control flow of an algorithm. Tia Watts develops “SFC Editor”, a graphical tool for algorithm development (Tia, 2004). SFC editor differs from other flow chart creation software in the ways that it represents directly C language control structures and can generate automatically C pseudo code from the flow charts.

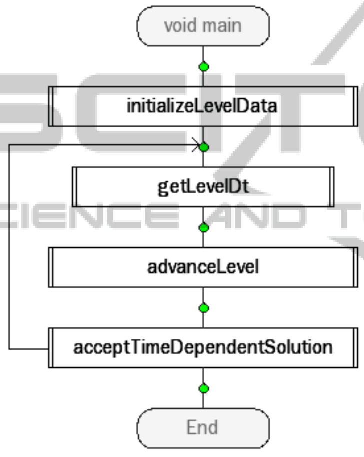


Figure 1: The predefined main flowchart of an application.

IDE-JASMIN implements a SFC editor to facilitate parallel programming. We take the SFC idea further and adapt it to fit the need of rapid parallel application development. IDE-JASMIN extends SFC editor’s sequential control structures to include parallel integrator component models and linear solver models. We also extend supported data types to cover all variables in JASMIN infrastructure. These extensions are necessary to support full application development in scientific and engineering computing.

We demonstrate our SFC approach by two examples. A predefined flow chart for application main control flow is illustrated in Figure 1. This flow chart defines the main procedure of a numerical simulation. It organizes four sub flow charts in a loop structure, where *initalizeLevelData* defines how the variables shall be initialized, *getLevelDt* calculates time step value and feeds it to *advanceLevel* which in turn does all number crouching jobs, the result is then copied back to appropriate variable by *acceptTimeDependentSolution* for next iteration. Details of *advanceLevel* are illustrated in Figure 2, where several integrator components are created and used to organize the actual computation.

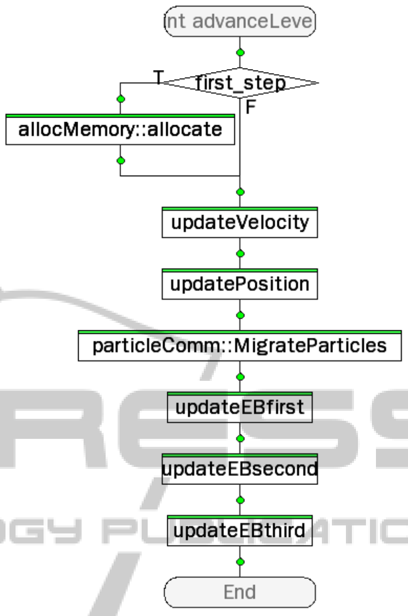


Figure 2: Integrator assembling in a flow chart.

As indicated by the above examples, our integrator-component-based SFC editor is helpful on the numerical algorithm top-down implementation. It decouples complex application to simpler and more manageable parts, which are represented as SFC elements. Using this approach, application developers only need to focus on configuring and assembling integrator components.

4 IMPLEMENTATION

IDE-JASMIN is developed in C++ using Qt graphics toolkit (Qt, 2012). It is designed to make numerical application development easy and fast. Backing up by a powerful code generation engine, IDE-JASMIN’s GUI comes up with four modes, including two edit modes, a debug mode and a running mode. Switching modes changes window UI and features. Figure 3 shows the main window of flowchart editor mode in IDE-JASMIN.

The left part of the UI is flowchart control panel. Elements in the left-upper corner can be dragged into the flowchart to organize the algorithm logic. Flowchart browser in the left-lower corner presents the flowchart structure. The middle part of the UI is used to view and edit flowchart. The right part provides additional editable attributes of current selected flowchart item. Edit operations take effect immediately when losing focus.

4.1 Visual Programming

Two edit modes are provided in IDE-JASMIN. One is SFC editor with component assembling abilities. The other is a kernel editor for Fortran 90 numerical kernel development. To create a parallel program, user selects required component model or structured block in the control panel, drag it onto the connection point in the flowchart, and configure possible attributes.

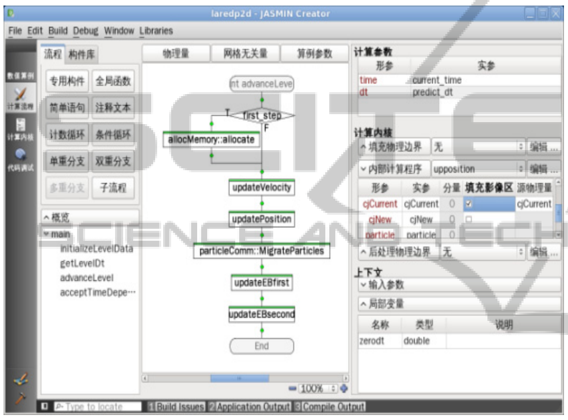
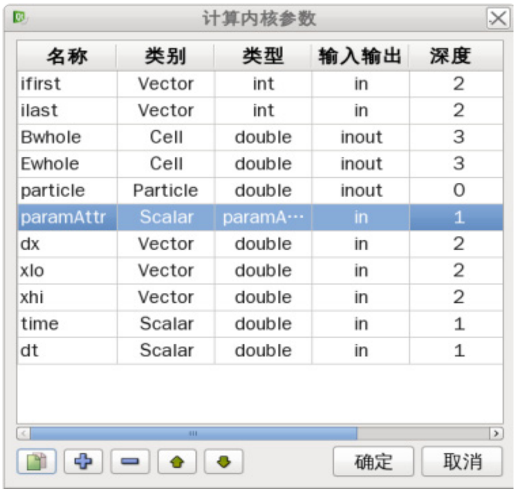


Figure 3: The flowchart editor mode of IDE-JASMIN.

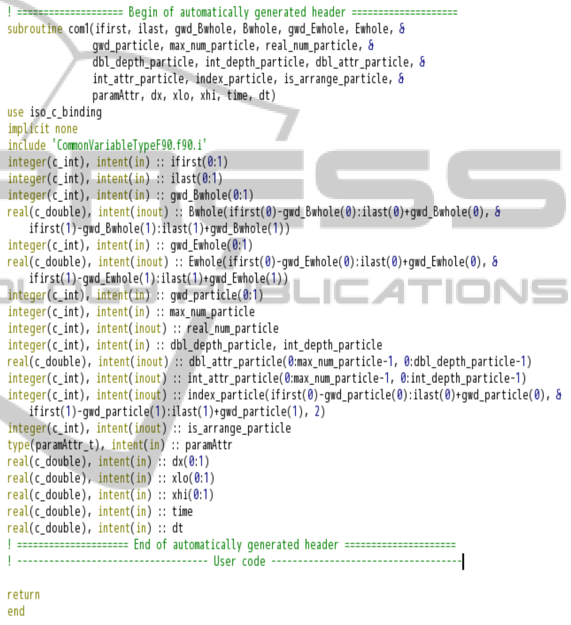
The SFC editor provides a programming abstraction: users can uses predefined integrator components to build their applications regardless of their programming experience. In this way, it promotes productivity of both experienced and novice developers.


计算内核参数

名称	类别	类型	输入输出	深度
ifirst	Vector	int	in	2
ilast	Vector	int	in	2
Bwhole	Cell	double	inout	3
Ewhole	Cell	double	inout	3
particle	Particle	double	inout	0
paramAttr	Scalar	paramA···	in	1
dx	Vector	double	in	2
xlo	Vector	double	in	2
xhi	Vector	double	in	2
time	Scalar	double	in	1
dt	Scalar	double	in	1

Figure 4: The kernel parameter definition GUI of IDE-JASMIN.

The kernel editor, in the other hand, provides users a convenient approach to write numerical kernels in Fortran 90 language. The editor provides a intuitive GUI where users can define kernel parameters by simply click and select. Figure 4 shows how it looks like. Users define abstract kernel parameters using domain specific data structures encapsulated in JASMIN and we generate automatically a valid Fortran 90 header for it (see Figure 5 for an example). This approach frees users from error-prone and tedious type definitions and variable declarations.


```fortran
! =================== Begin of automatically generated header ===================
subroutine com1(ifirst, ilast, gwd_Bwhole, Bwhole, gwd_Ewhole, Ewhole, &
            gwd_particle, max_num_particle, real_num_particle, &
            dbl_depth_particle, int_depth_particle, dbl_attr_particle, &
            int_attr_particle, index_particle, is_arrange_particle, &
            paramAttr, dx, xlo, xhi, time, dt)
use iso_c_binding
implicit none
include 'CommonVariableTypeF90.f90.i'
integer(c_int), intent(in) :: ifirst(0:1)
integer(c_int), intent(in) :: ilast(0:1)
integer(c_int), intent(in) :: gwd_Bwhole(0:1)
real(c_double), intent(inout) :: Bwhole(ifirst(0)-gwd_Bwhole(0):ilast(0)+gwd_Bwhole(0), &
  ifirst(1)-gwd_Bwhole(1):ilast(1)+gwd_Bwhole(1))
integer(c_int), intent(in) :: gwd_Ewhole(0:1)
real(c_double), intent(inout) :: Ewhole(ifirst(0)-gwd_Ewhole(0):ilast(0)+gwd_Ewhole(0), &
  ifirst(1)-gwd_Ewhole(1):ilast(1)+gwd_Ewhole(1))
integer(c_int), intent(in) :: gwd_particle(0:1)
integer(c_int), intent(in) :: max_num_particle
integer(c_int), intent(inout) :: real_num_particle
integer(c_int), intent(in) :: dbl_depth_particle, int_depth_particle
real(c_double), intent(inout) :: dbl_attr_particle(0:max_num_particle-1, 0:dbl_depth_particle-1)
integer(c_int), intent(inout) :: int_attr_particle(0:max_num_particle-1, 0:int_depth_particle-1)
integer(c_int), intent(inout) :: index_particle(ifirst(0)-gwd_particle(0):ilast(0)+gwd_particle(0), &
  ifirst(1)-gwd_particle(1):ilast(1)+gwd_particle(1), 2)
integer(c_int), intent(inout) :: is_arrange_particle
type(paramAttr_t), intent(in) :: paramAttr
real(c_double), intent(in) :: dx(0:1)
real(c_double), intent(in) :: xlo(0:1)
real(c_double), intent(in) :: xhi(0:1)
real(c_double), intent(in) :: time
real(c_double), intent(in) :: dt
! =================== End of automatically generated header ===================
! -------------------------------- User code ---------------------------------

return
end
```

Figure 5: Automatically generated Fortran 90 header for numerical kernel.

4.2 Code Generation

Flowchart and kernels as well as other user input are stored into a database, which are then feed into code generation engine to generate full application code. Flowchart defines the high level flow of the application and database keeps all information of integrator components, variables and numerical kernels. The code generation engine generate valid JASMIN application using JASMIN API and predefined application templates. This approach ensures JASMIN APIs are properly called and variables are properly defined and correctly passed around.

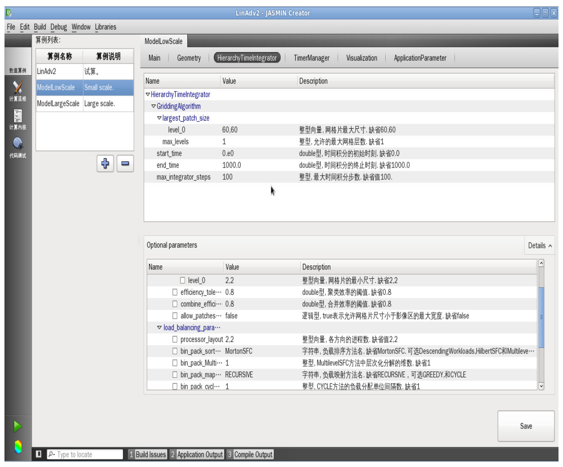
The generated source code is feed into a source project manager, which manages the build and debug process. Errors in build process are mapped back directly to flowcharts in SFC editor and users can find flowchart errors much easier. In fact, users do not make much programming mistakes when using SFC editor and the majority of mistakes are empty attribute configurations which can be easily validated prior to build. Debug information are also mapped back directly to flowchart and Fortran kernel, making it easy to spot bugs.

4.3 Simulation and Visualization

Scientific computing programming differs from other programming area in that users have to run simulations to validate their applications. Often it requires a lot of work to setup simulation configuration, run simulations, collect results and visualize data. To make life easier for scientific and engineering computing application developers, we provide a running mode in IDE-JASMIM and integrate visualization software in it.

In the running mode, users are provided with configuration tables where an physics model of a numerical simulation can be easily configured. Figure 6 shows part of the GUI for physics model configuration. Users only need to fill in mandatory parameters such as grid geometry and time integrator steps. The code generation engine will generate a valid input file for a simulation. The GUI also provides a physics model browser to help manage different physics models.

Besides physics model configuration, IDE-JASMIN supports running parallel numerical simulation directly in the IDE and number of parallel processes is specified in the GUI. The running of a simulation is confined to a separate directory to leave the directory structure clean and data collection is done automatically.



算例名称	算例说明
LinAdv2	试算。
ModelLowScale	Small scale.
ModelLargeScale	Large scale.

Name	Value	Description
HierarchyTimeIntegrator		
GriddingAlgorithm		
largest_patch_size		
level_0	60,60	整型向量, 网格片最大尺寸. 缺省60,60
max_levels	1	整型, 允许的最大网格层数. 缺省1
start_time	0.e0	double型, 时间积分的初始时刻. 缺省0.0
end_time	1000.0	double型, 时间积分的终止时刻. 缺省1000.0
max_integrator_steps	100	整型, 最大时间积分步数. 缺省值100.

Name	Value	Description
level_0	2,2	整型向量, 网格片的最小尺寸. 缺省2,2
efficiency_tole···	0.8	double型, 聚类效率的阈值. 缺省0.8
combine_effici···	0.8	double型, 合并效率的阈值. 缺省0.8
allow_patches···	false	逻辑型, true表示允许网格片尺寸小于影像区的最大宽度. 缺省false
load_balancing_para···		
processor_layout	2,2	整型向量, 各方向的进程数. 缺省值2,2
bin_pack_sort···	MortonSFC	字符串, 负载排序方法名. 缺省MortonSFC. 可选DescendingWorkloads,HilbertSFC和Multileve···
bin_pack_Multi···	1	整型, MultilevelSFC方法中层次化分解的维数. 缺省1
bin_pack_map···	RECURSIVE	字符串, 负载映射方法名. 缺省RECURSIVE，可选GREEDY,和CYCLE
bin_pack_cycl···	1	整型, CYCLE方法的负载分配单位间隔数. 缺省1

Figure 6: The physics model configuration GUI of IDE-JASMIN

IDE-JASMIN integrates JAVIS visualization platform to visualize simulation data. Users select variables to be visualized directly in the IDE and a python script is then generated and delivered to JAVIS visualization engine. The visualization result is then generated and delivered to users by a pop-up window when user triggered the visualization action. Different ways to inspect the simulation results are then immediately available from IDE-JASMIN.

5 APPLICATION

We apply IDE-JASMIN to develop uniform rectangular grid applications. Several parallel applications are built on our platform, including two JASMIN example applications. We also reconstruct a real world parallel numerical simulation application, LARED-P, on IDE-JASMIN.

For the above applications, we find that our developers finish building their applications in about one tenth the time required to write the whole application by hand. Our experiments suggests that developers with little or zero C++ knowledge can easily grasp the key idea of JASMIN and think more naturally when building their applications. Direct definition and use of physical variables, intuitive specification of algorithm logic by flowcharts, together with automatic code generation, save developers the burden of complex C++ language details and features. Both novice and experienced developers can build professional parallel numerical applications in almost identical speed.

LARED-P is a three-dimensional application for laser-plasma intersection simulation using Particle-In-Cell method (Pei and Zhu, 2009). It involves solving two equations with about a dozen integrator components. Flowchart in Figure 2 shows parts of these procedure:

- Numerical components *updateVelocity and updatePosition* for particle movement
- Numerical integrator components *updateEBfirst, updateEBsecond* and *updateEBthird* for Maxwell equation.

Over half of LARED-P soure code is automatically generated, including about 1350 lines C++ code and 290 lines Fortran code.

Figure 7 shows the visualization results of LARED-P application developed on our platform. It simulates 10 billions of particles on 2048 processor cores.

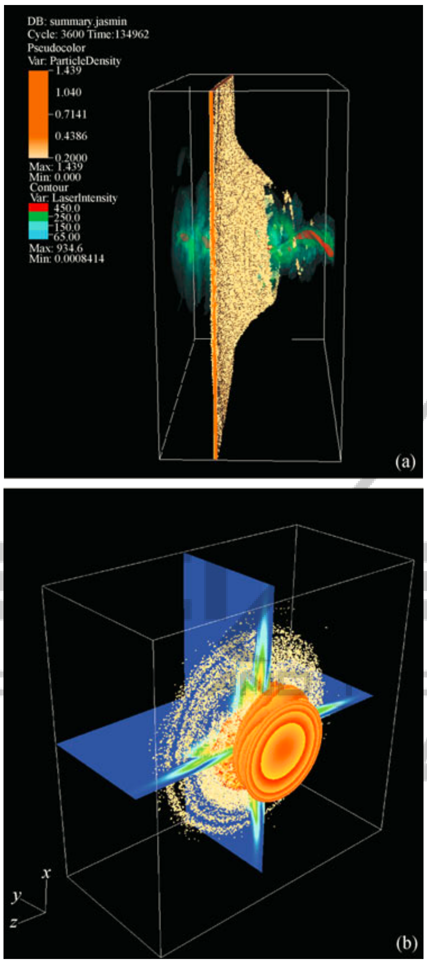


Figure 7: LARED-P simulation results: (a) Particle distribution snapshot; (b) Laser density contour.

6 CONCLUSIONS

We propose an interactive approach for high productivity parallel scientific and engineering numerical simulation applications, design and implement it in IDE-JASMIN. Our approach combines together domain-specific data structure declaration, SFC based algorithm logic definition, text kernel editing as well as automatic application generation, on top of JASMIN infrastructure, to ease parallel numerical application development. Real world applications show that our approach greatly reduced programming complexities.

In order to fully support application development, IDE-JASMIN integrates physical model definition, application source building, debugging, running and visualization. IDE-JASMIN is an ongoing project. Our next plan to fully support different application types defined in JASMIN and support fully reuse of already built components.

ACKNOWLEDGEMENTS

This work is under the auspices of the National Natural Science Foundation of China (61033009), National Basic Key Research Special Fund (2011CB309702) and National High Research and Development Program of China (2012AA01A309). Thanks for many JASMIN users who gave us some good suggestions for improvements the GUI of IDE.

REFERENCES

Mo Z. Y., Pei W. B., 2009. Scientific computing application codes. *Physics (in Chinese)*.

Mo Z. Y., Zhang A. Q., 2010. JASMIN: A parallel software infrastructure for scientific computing. *Front. Comput. Sci*. China.

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

Parker, S. G., 2002. A component-based architecture for parallel multi-physics PDE simulation. *In Proceedings of the International Conference on Computational Science-Part III*. Springer-Verlag.

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

Qt, 2012. *http://qt-project.org/*

Sarkar V., Harrod W., Snavelg A Z., 2009. Software challenges in extreme scale systems. *Journal of Physics: Conference Series*.

Tia W. 2004.The SFC Editor a graphical tool for algorithm development. *Jounal of Computing Science in Colleges*.