Toward an automated parallel computing environment for geosciences
Huai Zhang a,b,*, Mian Liu b, Yaolin Shi a, David A. Yuen c,d, Zhenzhen Yan a, Guoping Liang e

a Laboratory of Computational Geodynamics, Graduate University of Chinese Academy of Sciences, Beijing, PR China
b Department of Geological Sciences, University of Missouri, Columbia, MO 65211, USA
c Department of Geology and Geophysics, University of Minnesota, Minneapolis, MN 55455, USA
d Minnesota Supercomputing Institute, University of Minnesota, Minneapolis, MN 55455, USA
e Institute of Mathematics, Chinese Academy of Sciences, Beijing, PR China

Received 22 January 2007

Abstract

Software for geodynamic modeling has not kept up with the fast growing computing hardware and network resources. In the past decade supercomputing power has become available to most researchers in the form of affordable Beowulf clusters and other parallel computer platforms. However, to take full advantage of such computing power requires developing parallel algorithms and associated software, a task that is often too daunting for geoscience modelers whose main expertise is in geosciences. We introduce here an automated parallel computing environment built on open-source algorithms and libraries. Users interact with this computing environment by specifying the partial differential equations, solvers, and model-specific properties using an English-like modeling language in the input files. The system then automatically generates the finite element codes that can be run on distributed or shared memory parallel machines. This system is dynamic and flexible, allowing users to address different problems in geosciences. It is capable of providing web-based services, enabling users to generate source codes online. This unique feature will facilitate high-performance computing to be integrated with distributed data grids in the emerging cyber-infrastructures for geosciences. In this paper we discuss the principles of this automated modeling environment and provide examples to demonstrate its versatility.

© 2007 Published by Elsevier B.V.

Keywords: Parallel computing; Finite element method; Geodynamics; Cyber-infrastructure; GEON

1. Introduction

With rapid growth of affordable parallel computers, especially the Beowulf class PC- and workstation-clusters, parallel computing has become a powerful tool in the studies of geodynamo (Glatzmaier and Roberts, 1995), seismic wave propagation (Komatitsch and Tromp, 2002a,b), mantle convection (Kameyama and Yuen, 2006; Matyska and Yuen, 2005), lithospheric dynamics (Surussavadee and Staelin, 2006), and other fields of geosciences during the past decade. Powerful as it is, parallel computing is often time-consuming and requires multidisciplinary expertise: the understanding of the fundamental physics governing the geological processes, the ability to mathematically formulate the physics in terms of proper partial differential equa-
tions (PDEs) and initial and boundary conditions, the expertise of developing parallel numerical algorithms to solve the PDEs on parallel machines, and the know-how of benchmarking the model, optimizing the codes for the computer platforms, and visualizing the massive model outputs. Thus, developing parallel computing models usually entails a team effort with combined expertise in physical sciences and software engineering. In some fields where a general model is central to the community, such as air-circulation in the atmosphere or convection in the mantle, collective effort has been devoted to develop community parallel computing models. One example is the MM5 parallel computing software packages (http://www.mmm.ucar.edu/mm5/) for weather forecasting. In other fields, such as lithospheric dynamics, where modeling demands are diverse and complicated, parallel computing remains inaccessible to many researchers.

Geoscientists around the world are taking various measures to overcome this difficulty. For example, the Computational Infrastructure for Geodynamics (CIG, www.geodynamics.org) is a membership-governed organization that develops, maintains, and disseminates open-source software packages for typical geodynamic models, such as mantle convection and global seismic wave propagation. In mineral physics, the Virtual Laboratory for Planetary Materials (VLAB, http://vlab.msi.umn.edu) allows users to run large-scale quantum-mechanical calculations using web-services technology. The GeoFEM project (http://geofem.tokyo.rist.or.jp/) provides a multi-purpose, multi-physics parallel finite element solving environment, in which specific types of models, including seismic wave propagation, fluid dynamics, and structure mechanics, can be plugged into a general supporting system. Escript, developed by The Earth Systems Science Computational Centre (ESSCC) at The University of Queensland, is a system designed to implement PDE-based geoscience models using a computational modeling language based on the object-oriented Python scripts (Davies et al., 2004). Numerous commercial finite element packages, such as PDE2D (http://members.aol.com/pde2d/) and Finite Element program GENERator (FEGEN) (http://www.fegensoft.com), now allow users to generate finite element codes using high-level scripts (modeling languages) in the input files (drivers) that specify the PDEs and model-specific properties.

In this paper, we introduce a prototype of an automated parallel computing environment. Instead of providing static packages of specific models, this system enables geoscientists to model their own problems using high-level scripts, and automatically generate the complete, machine-independent Fortran source codes which can be run on parallel machines. This system may free geoscientists from most part of the time-consuming and error-prone coding for parallel computation. It has the potential to seamlessly integrate data grids with distributed high-performance computing facilities in the new generation of cyber-infrastructures. In the following, we first introduce this system, and then describe the modeling language user interface, and the automation of generating finite element codes. We then provide examples of using this system to model a variety of geological processes to show the versatility of this system.

2. Overview of the automated parallel computing environment

The work flow of geosciences investigation usually consists of the following steps: first, from the observational data we formulate a physical model that attempts to capture the fundamental physics responsible for the observed. Second, we describe the physical model using a set of mathematical equations, usually in the form of partial differential equations, with proper initial and boundary conditions as well as model-specific properties such as the rheology and other physical parameters. Finally, we solve these mathematical equations with the proper constraints; the model is tested and explored by comparing the model predictions with observational data (Fig. 1).

Most physical models in Earth sciences are complicated and require solving non-linear, coupled, or time-dependent PDEs in spatial and temporal domains. For these problems, numerical solutions are often needed, and finite element method (FEM) is by far the most popular method. FEM discretizes the model domain into a finite number of elements with simple geometry; for each element the partial differential equations can be simplified with a set of algebraic equations. Thus, FEM turns a problem of solving a system of partial differential equations with complex initial and boundary conditions into a much easier problem of solving a system of algebraic equations that can be performed by computers.

Although simple in principle, using FEM is usually complicated and time-consuming, especially when dealing with coupled, non-linear problems or parallel algorithms. The tasks can be challenging even to skilled modelers. The automated parallel computing environment to be introduced here is aimed at lowering the technical hurdles of using parallel FEM. This system includes two key aspects: (1) the user interface based
Fig. 1. Flow chart showing how the modeling language based automated parallel computing environment may ease the work flow of scientific investigations in geosciences.

on a high-level, English-like modeling language; (2) an automated system of generating and assembling finite element codes. Similar to C, Java, Fortran and other high-level computer languages, the modeling language is a set of expressions to describe various PDEs and mathematical algorithms using familiar mathematical notations and specific variables. It uses advanced interpreters and interfaces to numerically describe the physical model and instruct the system to generate the desired Fortran source codes. The automation of generating FEM codes is possible because the major part of a FEM code, which is solving a large system of matrices, is similar for many FEM models, thus preexisting subprograms and algorithms can be reused. This approach has been used in some commercial software packages, such as the PDE2D (http://members.aol.com/pde2d), which allows user to answer a set of questions about the model region, partial differential equations, boundary conditions, and preferred solver and graphical output options through an interactive driver; the finite element codes are then generated and compiled automatically, sparing user much labor of finite element coding.

3. Modeling language expressions for partial differential equations

Here we explain the modeling language used in our automated parallel computing environment. All partial differential equations fall into there main categories: elliptic, parabolic, and hyperbolic equations. For designing numerical solutions, we group these equations into another three types: static, one-order partial derivative and two-order partial derivative of \( u \) with respect to time \( t \), respectively:

\[
Lu = f 
\]  
(3.1)

\[
C \frac{\partial u}{\partial t} + Lu = f 
\]  
(3.2)

\[
M \frac{\partial u^2}{\partial t^2} + C \frac{\partial u}{\partial t} + Lu = f 
\]  
(3.3)

where \( u \) denote unknown and \( L, C, \) and \( M \) are the linear or non-linear operator (or coefficient) of \( u, \frac{\partial u}{\partial t}, \) and \( \frac{\partial u^2}{\partial t^2} \), respectively. The right-hand-side term \( f \) can be either linear or non-linear. If \( L, C, M \) or \( f \) is non-linear,
then these PDEs are non-linear. Eqs. (3.2) and (3.3) are also called evolutionary equations.

In FEM, the Ritz and Galerkin methods are commonly used to convert PDEs to a system of high dimensional linear equations, which may then be projected to a lower dimensional system by using variational principles. A variety of numerical techniques can be applied to manage non-linearity at each timestep, such as the Newton–Raphson method (Dettmer and Peric, 2006) for non-linear iterations, the Crank–Nicholson (Honda et al., 1993) and the Newmark schemes (Komatitsch et al., 1999; Zampieri and Pavarino, 2006) for time-dependent equations with respect of time $t$.

In our modeling environment, each term of these equations, as well as the algorithms, can be specified by the modeling language which link them to proper numerical segments and assemble them into a coherent numerical scheme. This works even for systems of coupled physics. We illustrate here the use of the modeling language in three examples.

### 3.1. Describing a linear problem with the modeling language

We first show how to use the modeling language to specify and describe a linear problem, such as the Dirichlet problem (Krishnamoorthy, 1995) defined by the Laplace’s equation in a two-dimensional domain:

\[
\begin{cases} 
-\Delta u = f & \text{(in } \Omega) \\
\ u = u_0 & \text{(on } \partial \Omega)
\end{cases} 
\] (3.4)

where $f$ is given and $u$ is an unknown function of $x$ and $y$ in a two-dimensional connected open domain $\Omega$, and $\Delta$ is Laplacian operator denote the second derivative of $u$ with respect to $x$ and $y$ whose boundary $\partial \Omega$ is a continuous manifold or a polygon.

The weak or variational form of Eq. (3.4) can be written as following,

\[
(\nabla u, \nabla \bar{u})_\Omega - \int_{\partial \Omega} \frac{\partial u}{\partial n} \bar{u} \, ds = (f, \bar{u})_\Omega
\] (3.5)

where $\nabla$ is the gradient, $(\cdot, \cdot)$ the inner product in the two-dimensional plane, and $\bar{u}$ represents virtual displacement of $u$. Furthermore, $\bar{u}$ can be turned into an inner product on a suitable space of “once differentiable” functions of $\Omega$ that is zero on $\partial \Omega$, and (3.5) can be rewritten as:

\[
(\nabla u, \nabla \bar{u})_\Omega = (f, \bar{u})_\Omega
\] (3.6)

In our parallel computing environment, we describe this problem use the following scripts in the modeling language:

```
defi
  disp u
  coor x y
  shap %1 %2
  gaus %3
  func gux guy

  func
  $c6 \ \text{vol} = 1.0d0$
  gux=+[u/x]
  guy=+[u/y]

  stif
  dist=+[gux;gux]*vol+[guy;guy]*vol
  load=+[u]*f*vol

end
```

The ‘defi’ section defines the unknown $u$ and the coordinate. The ‘shap’ section specifies the finite element discretization: the type of the shape function and the node number of each element. For example, replacing ‘%1’ and ‘%2’ with ‘quadrilateral’ and ‘4’ means that we select a four-node quadrilateral element for discretization of the domain $\Omega$. ‘gaus %3’ defines the method of Gaussian quadrature rule in each coordinate (Kronrod, 1965), and ‘func gux guy’ defined two functions, whose expressions are defined in the ‘func’ section; ‘[u/x]’ denotes the derivative of $u$ with respect to $x$. In the ‘stiff’ section, the actual Laplace equations’ weak form expression is given. The symbol $\cdot;\cdot$ denotes the scalar inner product of two functions. The function before the semicolon is the unknown function or its derivative, the function behind the semicolon is the virtual displacement of the unknown function or its derivative. ‘dist’ means
that the element stiff matrix will be calculated and stored in distributed manner. The right hand function is given via ‘load’ keyword.

Having described the PDEs and their weak form (Eqs. (3.4)–(3.6)) in the modeling language, we need to instruct the system what method we wish to use to solve this Laplace’s equation, and other options for the numerical scheme. These instructions are given in two input files: ‘.gio’ and ‘.gen’. The ‘.gio’ file tells the system the element type and auxiliary items, such as input data for material coefficients. The ‘.gen’ file specifies what kind of solving method we want to use. For this example, the

\[
\begin{align*}
\nabla d u_{i+1} + \nabla \bar{u} \rightleftharpoons (3 u_i^2, du_{i+1}, \bar{u}) \rightleftharpoons = -(\nabla u_i, \nabla \bar{u}) \rightleftharpoons - (u_i^3, \bar{u}) \rightleftharpoons + (f, \bar{u}) \rightleftharpoons \\

u_{i+1} = u_i + \Delta u_{i+1}
\end{align*}
\]

contents of the ‘.gen’ file include these lines:

```
defi
   a ell &
START%
   a
SOLV%
```

In the ‘defi’ section, ‘a’ defines a field named ‘a’, and ‘ell’ tell the system to use elliptic type algorithm to solve this problem. This algorithm is one of the standard algorithms built in the system. Users can also define their own algorithms by modifying the ‘.gen’ file.

### 3.2. Describing a non-linear problem with the modeling language

For a non-linear problem, the procedure is slightly different. For the PDEs:

\[
\begin{align*}
-\Delta u + u^3 &= -4 + (x^2 + y^2)^3 \quad \text{(in } \Omega) \\
u &= x^2 + y^2 \quad \text{(on } \partial \Omega)
\end{align*}
\]

The corresponding Galerkin weak form is

\[
(-\Delta u, \bar{u})_\Omega + (u^3, \bar{u})_\Omega = (f, \bar{u})_\Omega
\]  

(3.8)

Integrating by parts on the first term of the left-hand-side of (3.8), we obtain

\[
(\nabla u, \nabla \bar{u})_\Omega + (u^3, \bar{u})_\Omega = (f, \bar{u})_\Omega + \left(\frac{\partial u}{\partial n}, \bar{u}\right)_{\partial \Omega}
\]

(3.9)

where \( f \) is the right-hand-side of (3.7):

\[
f = -4 + (x^2 + y^2)^3
\]

(3.10)

where \( \bar{u} \) represents virtual displacement of \( u \). Furthermore, \( \bar{u} \) can be turned into an inner product on a suitable space of “once differentiable” functions of \( \Omega \) that are zero on \( \partial \Omega \), then (3.9) can be rewritten as:

\[
(\nabla u, \nabla \bar{u})_\Omega + (u^3, \bar{u})_\Omega = (f, \bar{u})_\Omega
\]

(3.11)

To apply Newton’s method (Ypma, 1995), we have to linearize (3.11) by defining the functional \( F \) as:

\[
\begin{align*}
F(u) &= (\nabla u, \nabla \bar{u})_\Omega + (u^3, \bar{u})_\Omega = (f, \bar{u})_\Omega \\
u_{i+1} &= u_i + \Delta u_{i+1}
\end{align*}
\]

(3.12)

the linearized form is

\[
(\nabla u_{i+1}, \nabla \bar{u})_\Omega + (3u_i^2, du_{i+1}, \bar{u})_\Omega + (\nabla u_i, \nabla \bar{u})_\Omega
\]

(3.13)

Add \( F(u_i) \) to each side of (3.13), we have:

\[
(\nabla u_{i+1}, \nabla \bar{u})_\Omega + (3u_i^2, du_{i+1}, \bar{u})_\Omega + (\nabla u_i, \nabla \bar{u})_\Omega
\]

(3.14)

Finally, we obtain

\[
(\nabla u_{i+1}, \nabla \bar{u})_\Omega + 3u_i^2(du_{i+1}, \bar{u})_\Omega = (f + 2u_i^3, \bar{u})_\Omega
\]

(3.15)

where \( u_{i+1} \) denotes unknown value after \( i+1 \) times non-linear iteration, \( u_i \) the unknown value after \( i \) times non-linear iteration and \( \Delta u_{i+1} \) denotes unknown increment value after \( i+1 \) times non-linear iteration, respectively.

In our FEM modeling environment these FEM algorithms can be expressed using the modeling language (Appendix A) similar to the linear problems. Following a definition section, an ‘equation’ section describes the formation of the linear system and the desired linear solvers, such as the Gauss elimination method, or pre-conditionered Krylov subspace BiCG-stable method. Finally, a ‘solution’ describes the algorithm for non-linear iteration schemes, such as the modified Newton–Raphson method (Appendix A).

### 3.3. Describing a non-linear couple problem using the modeling language

A multi-field or coupled problem generally consists of several partial differential equations, and the unknowns are interdependent. In this case a coupled numerical scheme is needed to specify the interconnections between different unknown fields and to control the iteration order. This can be expressed explic-
The keyword ‘defi’ specifies the algorithms for each field and their interrelationship. In this example, there are two fields: ‘a’ and ‘b’. The line “a ell b &” means that the algorithm ‘ell’ (algorithm for solving elliptic type of PDE for a single field, a standard linear static algorithm that is available from the algorithm library) will be used for solving fields ‘a’ and ‘b’. Furthermore, field ‘a’ depends on the results of field ‘b’. Note that the algorithm for each field can be either directly taken from the algorithm library or provided by users.

The second section consists of four command lines. The first two lines initialize fields ‘a’ and ‘b’, respectively. The third and fourth lines solve fields ‘b’ and ‘a’, respectively. The solution of field ‘b’ is obtained at each non-linear step before the calculation of field ‘a’. The character string ‘sin’ in the command lines means that symmetric solver is adopted.

4. Automatic generation of parallel FEM codes

A finite element model usually consists of some basic parts: pre-processing, mesh and data partition, computing element matrix and load vector, assembling and solving large sparse matrices, and post-processing. Many of these parts are similar, even identical, in different FEM models. In an automated FEM modeling systems such as the one introduced here, these “static” parts are written as standardized FEM segment source codes, which can be specified using the modeling language described above, and the final FEM source code may be assembled by plugging the necessary segments of the source codes into a program stencil (Fig. 2).

In our parallel computing environment, the program stencils were designed based on the domain decomposition method. We designed a cluster of program stencils for different types of PDEs. We have also included some of the new program stencils emerged in recent years (Gross et al., 2005) with enriched intrinsic commands.

4.1. Domain decomposition

Parallel computing takes “divide to conquer” strategy to solve complicated problems. In our system we use the domain decomposition method (DDM) (Valli, 1999) to divided the modeling domain into sub-domains, and solve for the unknown fields in these sub-domains simultaneously using multiple computer processors. Fig. 3 illustrates the use of this method in solving a boundary value problem in domain $\Omega$, which is divided into inter-connected sub-domains ($\Omega_i, i = 1, 2, \ldots, n$). In doing so the original problem is turned into a group of relatively simpler and smaller boundary value problems that can be solved simultaneously on different computer nodes. The inner boundaries between the sub-domains, shown by lines $\Gamma_1$ and $\Gamma_2$ in Fig. 3, can be non-overlapped or overlapped, and various preconditioners (Ainsworth, 1996) can be used to treat the inner boundaries (Jianwen Cao, 2005; Korkeev and Jensen, 1999; Rakowsky, 1999; Zumbusch, 1996). An array of methods, including the Language multiplier based substructuring method (Mandel and Tezaur, 1996), preconditioned Krylov subspace iteration (Axelsson et al., 2004; Ito and Toivanen, 2006; Prudencio et al., 2006), parallel direct solvers (Agullo et al., 2006), and multi-grid (Douglas, 1996; Jung, 1997; Lv et al., 2006) and multi-level (Axelsson and Larin, 1998) solvers, can be used to solve for the parallel FEM problems.

4.2. Mesh and data partition

To decompose the model domain for parallel computing, edge- or vertex-based partition methods, with or without overlapping regions, need to be selected in advance. For the partitioning scheme, we also need to define variables for both local and global interfaces. After the computing domain is partitioned, local variables are re-ordered, and then the global ordering of the variables must be updated systematically.

In our parallel computing environment, we have developed a subsystem to directly generate unstructured sparse graph (Michel et al., 1997) from hybrid finite element mesh (Fig. 4). This process can be separated from the parallel computing program as a pre-processing. We have also developed subroutines which can be assembled with the parallel computing program and generate a sparse graph in parallel. We have used the programs Metis and ParaMetis (Karypis, 2003) for the partition of
unstructured sparse graph with or without inner boundary overlapping. The mesh partition and data partition can be executed based on the partition result of unstructured sparse graph.

4.3. Parallel solvers and the solver interface

Large sparse linear systems arising from many geoscientific models may have several millions of equations. Solving such large systems is challenging because of the high demand on computational time and memory. Iterative solution techniques based on Krylov subspace methods and preconditioning methods (multi-grid and approximate LU factorization techniques) are commonly used. Because the Krylov subspace methods require only matrix–vector products, the role of the domain decomposition method (DDM) is restricted to the development of efficient parallel preconditioners. Our system provides two preconditioners that use iterative solvers based on generalized minimal residual method (GMRES) to form an approximate solution of the Schur complement system.

Our parallel computing environment allows user to select different parallel solvers from the catalog, which
includes parallel direct solvers and parallel iteration solvers. We have developed some of parallel solvers, such as the Lagrange multiplier-based domain decomposition method solvers. The system provides these as default solvers. In recent years, numerous large-scale linear system parallel solvers have become available in public domain. These include multifrontal massively parallel sparse direct solver (MUMPS) (Amestoy et al., 2001), SuperLU (Li and Demmel, 2003), Aztec (Shadid, 1992), and parallel Algebraic Recursive Multi-level Solvers (pARMS) (Li et al., 2003). Our parallel computing environment provides interface to these solvers.

In each sub-domain, we first compute the sparse graph of the gross matrix. Then the memory requirement of gross matrix can be determined. We can design different program stencils (see below) to link to different parallel solvers. Users can easily explore multiple variations of a parallel algorithm and select a parallel solver from the menu provided by the system.

4.4. Hierarchical structure of the FEM modeling environment

Fig. 5 shows the hierarchical structure of our automated parallel modeling system. This system has three hierarchical layers. The top layer is the user interface consisting of three input files: ‘.PDE’, ‘.GIO’ and ‘.GCN’. In these files user uses the modeling language to specify, respectively, the PDEs, the element type and other auxiliary items, such as input data for
material coefficients, and the solvers to be used, as discussed in Section 3. The second layer is the source code generator and supporting facilities. After receiving the input files, the system runs a script file, named ‘.fem’ (Appendix B) to generate the program segments and insert them into the program stencils. These codes can be compiled directly and linked with the libraries (libs) on the bottom layer to form executable programs, including program for pre-processing and mesh generation, and post-processing programs. These build-in libraries include sequential and parallel solvers for large-scale sparse linear system, auxiliary libraries of shape functions, Jacobi matrix calculating and its reversion. The libraries for pre-processing and unstructured finite element mesh generation include subroutines for the Delaunay triangulation and quadrilateral unstructured mesh generation in two dimension (Owen and Saigal, 2000; Rebay, 1993), tetrahedral mesh generation in three dimension (Lo and Wang, 2005), Volonoi diagram (dual partition of Delaunay triangulation) for generating sparse graph (Gold and Angel, 2006) and sparse graph partition. The post-processing programs can arrange data in the formats ready for serial or parallel visualization tools, such as GID, OpenDX (http://www.opendx.org), and Para View (http://www.paraview.org).

The system can build and provide finished model libraries with various kinds of PDEs, with choices of shape functions and operators in Cartesian, spherical, and cylindrical coordinates in 1D, 2D, and 3D. From the user menu, one can choose the library files by simply clicking the options from user interface. User has the flexibility to modify the codes with the desired element subroutine and algorithm implementation programs, including shape functions and algorithm expressions within these libraries. The libraries of finished FEM models will include the following:

- Solid mechanics of linear elastic, viscoelastic, and visco-plastic–elastic, finite deformation analysis and Lagrangian multiplier for discontinuous deformation models.
- Fluid dynamics of Navier–Stokes equations, shallow water equations, Stokesian fluid, Newtonian and non-Newtonian fluids.
- Rock mechanics.
- Porous media: steady-state and transit transportation process in porous media.
- Electromagnetic problems: static, time-harmonic, and fully time-dependent dynamics models.
- Typical mathematic models: linear, non-linear, coupled linear and time-dependent mathematical and physical models, with a lot of corresponding algorithm scheme included.

5. Geosciences applications

The automated parallel modeling environment is built on a core of the commercial version for serial FE element modeling (http://www.fegensoft.com), which has been widely used in engineering. The parallel computing environment is still under developing. We show here some examples to show its versatility. A complete example of using the automated parallel computing environment is provided in Appendix D.

5.1. Parallel computing of lithospheric dynamics

We have used this system to model complex lithospheric dynamics in recent years; some of the results are presented by Liu et al. (this volume). In one study we constructed a subcontinental scale model for active tectonics in the western United States. With over half million elements, our 3D lithospheric model allowed inclusion of all major geological boundaries and first-order lithospheric structure in the model. Using our parallel modeling system, we were able to construct and fine-tune this sophisticated model in few weeks that would normally take months even years to build. The model runs efficiently on different PC-clusters, from our small 32-processor cluster to 512-processor large clusters. Such models provide a geodynamic framework to incorporate detailed 3D lithospheric structures, such as those to be expected from the EarthScope (www.earthscope.org), for advanced simulation of 4D continental evolution.

In another example (Liu et al., this volume), we develop a parallel FEM model for the entire San Andreas Fault system. The parallel algorithms built in our system allows simulation with more realistic viscoelasto-plastic rheology for lithospheric deformation in the plate boundary zone (Li and Liu, 2006), and the computing power of parallel machines allowed us to explore the multi-timescale faulting, from rupture in seconds to secular slip in thousands of years, in a single self-consistent model.

5.2. Seismic wave propagation in the Fuzhou basin

We have also used the parallel modeling system to construct a preliminary model for seismic wave propagation. Regional seismic wave propagation can be complicated by heterogeneities in the crust and upper mantle; in sedimentary basins small-scale het-
erogeneities in the sedimentary cover may significantly affect the wave propagation and amplify ground motion (Cardenas-Soto and Chavez-Garcia, 2003). Whereas numerous commercial and public domain software packages are available, such as SPECFEM3D BASIN (http://www.geodynamics.org), they are inadequate if we need to consider complex basin structures.

As part of a national effort in China to study potential earthquake hazards around major cities, we attempted to model wave propagation in the Fuzhou basin, a 50 km × 60 km region in China’s coastal province of Fujian with bustling economy and dense population (Fig. 6). In this case, a high-resolution finite element model is needed to explore the impact on seismic wave propagation by the heterogeneous basin and lithospheric structure. The equation for wave propagation is given by:

\[ \rho \ddot{u} + c \dot{u} - (\mu \nabla \cdot (\nabla \mu) + (\lambda + \mu) \nabla (\nabla \cdot u)) + f = 0 \]

(5.1)

where \( \rho \) is the density, \( c \) the damp coefficient, \( \lambda \) and \( \mu \) the Lame constants, and dots over \( u \) indicate time-derivatives.

For this second-order time-dependent hyperbolic problem, we used Newmark scheme (Artuzi, 2005) in time domain, and incorporated the perfectly matched layer (PML) boundary condition into the formulation to absorb the unwanted radiation out of the computational domain (Obayya et al., 2000; Wang, 2003). It took us only a few days to prepare the input information and generate all source codes need for serial and parallel simulation (see Appendix C for the input files for generating all the source codes of this model). The full 3D numerical results are shown in Fig. 6. This model has about 1.2 million unstructured grid points. The preliminary results show that the maximum amplitudes of ground acceleration are closely related to the heterogeneous basin structure. Thus, the detailed wave propagation modeling may provide some useful guide for urban planning and preparation of earthquake hazards.

5.3. Mantle convection coupled with lithospheric dynamics

In the final example, we use this modeling system to solve fluid mechanics related to mantle convection, coupled with plate tectonics. Tectonic plates may be reviewed as the top thermo-mechanical boundary layer of the convective mantle (McKenzie et al., 1974). However, modeling the coupled system of lithospheric dynamics and mantle convection is challenging because of the different rheology and physical processes involved. We have attempted to model such a system using our parallel modeling environment (Sun, 2002). Approximating the mantle as an incompressible and isoviscous fluid in a spherical shell, the governing equations including the conservation of momentum:

\[ 0 = - \nabla p + \mu \Delta v \]

(5.2)

\[ \nabla \cdot v = 0 \]

(5.3)

and the conservation of energy:

\[ \frac{\partial T}{\partial t} = \nabla \cdot (k \nabla T) - v \cdot \nabla T + q \]

(5.4)

where \( \nabla \) denotes the gradient operator, \( \nabla \cdot \) denotes the divergence operator, \( v \) represents velocities, \( T \) denotes temperature, \( t \) denotes time, \( p \) represents deviatoric pressure, \( \mu \) denotes mean rheology coefficient, \( k \) the thermal diffusivity and \( q \) is the heat source term, respectively.
The lithospheric shell assumes a power-law rheology

\[ \varepsilon = A \bar{\sigma}^n \]  

where \( A \) represents mean viscosity of lithospheric shell, and \( \varepsilon \) represents the mean strain rate,

\[ \varepsilon = \sqrt{\frac{2}{3} \varepsilon_{ij} \varepsilon_{ij}} \]  

In which

\[ \varepsilon_{ij} = \frac{1}{2} (\dot{u}_i,j + \dot{u}_j,i - \dot{u}_k \dot{u}_{k,j}) \]  

and the mean stress is defined as

\[ \bar{\sigma} = \sqrt{\frac{3}{2} \sigma_{ij} \sigma_{ij}} \]  

Solving Eqs. (5.2)–(5.4) requires different iteration approaches because different types of PDEs are involved, and the computational problem is specially suited for parallel computers. Using our parallel FEM modeling environment, we were able to generate a complete Fortran source codes for this model (Sun, 2002).

We used Eqs. (5.2), (5.3) and (5.5) to simulate mantle convection, and Eqs. (5.5) and (5.3) to model lithospheric deformation. Eqs. (5.2) and (5.3) are computed separately using different non-linear algorithms. The coupling between Eqs. (5.2) and (5.3) is fulfilled through exchanging and updating the velocity and temperature fields after each step of convergence. The Lagrange multiplier domain decomposition method (LMDDM) and the Lagrange multiplier discontinuous deformation analyses (LMDDA) were used for interactions between neighbor plates (Sun Xunying and Guoping, 2002). After each time step, the velocity field of the lower layer of the lithosphere will be updated from values derived from mantle convection simulations. The domain decomposition and some of the computing results are shown in Fig. 7. We used 17 computer nodes for this simulation. One of them works as the master node which controls the parallel computing process by distributing data to different slave nodes and gathering and updating the global values after each iteration step and time step. These automatically generated parallel finite element codes are scalable with the processors available and worked well on different parallel machines.

Fig. 7. The model of coupled plate-mantle convection. (a and b) Show the unstructured mesh generation and domain decomposition. (c) Assemble FE mesh. (d) Predicted surface velocity. All the unstructured mesh are generated in parallel by different processors.
6. Discussion

The automated parallel FEM modeling environment presented here can be readily integrated into the emerging cyber-infrastructures for geosciences that are driven by both grid computational technology and the unprecedented growth of geosciences data. The ever-increasing internet connection and fast development of grid middleware technology have stimulated the fast growth of cyber-infrastructure in geosciences, such as the Geoscience Network (GEON) which integrate multidisciplinary data bases with application tools (http://www.geongrid.org), including data manipulation and visualization. Modeling often provides the critical step to turn observation into understanding and knowledge. The automated parallel computing environment introduced here makes FEM modeling easy and accessible. It eases the tasks of finite element modeling and helps scientists to take full advantage of the increasingly affordable parallel and grid computing hardware to solve complicated multi-scale, multi-physical problems. Using the English-like modeling language to describe the PDEs and algorithms, scientists can be liberated from most part of the time-consuming and error-prone tasks of finite element coding. Anyone with a solid understanding of the physics to be investigated and some basic knowledge of FEM modeling can use this system. When incorporated into the emerging collaborative cyber-infrastructures, our modeling system can facilitate collaborative data exploration and interpretation. Fig. 8 shows the basic concept of integrating this automated parallel computing environment with grid-based geoscience cyber-infrastructure.

One of the distinguished advantages of this system is that one can manipulate all the facilities provided by the distributed cyber-infrastructure through grid middleware. Moreover, it allows user to access an intelligent problem-oriented or service-oriented architecture and take full advantage of all the resources, such as geological data, high performance computing power, computing-intensive or memory-intensive model analysis, 4D visualization for multidisciplinary information or computing results, as well as convenient web-based interface.

Fig. 8. Sketch show integration of the parallel computing environment with geosciences cyber-infrastructure. The parallel environment system includes five parts: preprocessing, element subroutine generation, parallel solver, non-linear algorithm and iteration method. It can interface with visualization and other network resources.
All the source code generated, program stencils, libraries of different PDEs and algorithms are accessible to the users. This is an ongoing effort; a preliminary system is available through a Common Gateway Interface (CGI) interface on the website http://www.fegensoft.com, which allow user to become developers. We aspire to make this system an open-source community computing environment in the near future.

7. Conclusions

We have constructed a prototype automated parallel computing environment which automatically generates finite element Fortran source codes from user input using high-level modeling language. This system can spare researchers from much of the time-consuming and error-prone FEM coding, and greatly shorten the times needed for developing most FEM models.

This system consists of three steps commensurate with the current geoscience research models. In the first step, this system translates physical models using English-like modeling language into numerical segments. The parallel computing environment then automatically generates source codes following instructions in the user input files. The numerical results can then be output to visualization facilities.

Contrasting to most problem-specific finite element packages, this system is dynamic. It can be used to solve various kinds of physical problems described by differential PDEs with optional algorithms. The open architecture of this system could facilitate integration of data grids with high performance computing facility in a comprehend cyber-infrastructure for geosciences.

Acknowledgements

This work is supported by the National Science Foundation of China under Grant Numbers 40474038, 40574021, 40374038 and by National Basic Research Program of China under Grant Number 2004cb418406. Zhang’s visit to the University of Missouri-Columbia was partially supported by the NSF-EAR 0225546 grant to Liu. This work was conducted as part of the visualization working group at laboratory of computational geodynamics supported by the Graduate University of Chinese Academy of Sciences, we thank Shi Chen and Shaolin Chen in the visualization working group who provided some of the figures. David A. Yuen thanked NSF for support in CMG and ITR programs.

Appendix A. Finite element modeling language for a non-linear problem

The modeling language expressions for describing the non-linear problem in Section 3.2 are shown here:

```
defi
  disp u
  coor x y
  coef un
  shap %1 %2
  gaus %3
  func gux guy

  func
  gux=+[u/x]
  guy=+[u/y]

  stif
  dist=+[gux;gux]*vol+[guy;guy]*vol
  +[u;u]*un**2.0d0*3.0d0
  load=+[u]*(-4.0d0+(x*x+y*y)**3.0d0)*vol
  +[u]*un**3.0d0*2.0d0

end
```

The scripts are similar to the linear Laplace’s equation in Section 3.1. Only the ‘dist’ and ‘load’ sections are different.

The non-linear algorithm discussed here specify the method of solving the PDEs, including how to linearize a non-linear differential equation, how to discretize the time variable for a time-dependent problem, and how to control the incremental value of each iteration step (i.e. the relaxation factor) and the convergence precision for a non-linear problem. For a coupled multi-field problem, it also includes the iteration sequence and the coupling schedule of different equations by means of coef u.
defi %dof
stifs
mass m
load f
type e
mdty l
step 0
init l
coef u

equation

vect u
read(s,unod) u

matrix = [s]
forc=[f]
solution u

vect u1,u,ue,du
read(s,unod) u1,du

[ue]=[u]-[u1]

Sc6 aa = 0.0
Sc6 ab = 0.0
Sc6 bb = 0.0

%nod

aa = aa+[ue]*[ue]
ab = ab+[ue]*[du]
bb = bb+[du]*[du]

%dof

%nod

$C6 cc = 1.0$

$C6 if (bb.gt.1.0e-20) then$

$C6 rab = sqrt(aa)*sqrt(bb)$

$C6 if (aa.gt.bb) cc = sqrt(bb/aa)$

$C6 if (ab.gt.0.5*rab) cc = cc*2.0$

$C6 if (ab.gt.0.8*rab) cc = cc*2.0$

$C6 if (ab.lt.0.0) cc = cc*0.5$

$C6 if (ab.lt.-0.40*rab) cc = cc*0.5$

$C6 if (ab.lt.-0.80*rab) cc = cc*0.5$

$C6 endif$

$C6 if (cc.gt.1.0) cc = 1.0$

$C6 err = 0.0$

$C6 ul = 0.0$

%nod

%dof
Appendix B. A typical sample of the ‘.fem’ file

In the automated parallel modeling environment, the ‘.fem’ scripts file links the user input files and the libraries to generate all the source codes (Fig. 5). The following is a typical sample of this file:

\[
\begin{align*}
\text{err} &= \text{err} + [\text{ue}]^2 \\
[\text{ue}] &= [\text{ue}] \times c \times c \\
[\text{u}] &= [\text{u}] + [\text{ue}] \\
\text{ul} &= \text{ul} + [\text{u}]^2
\end{align*}
\]

Each ‘#’ marks one section. The ‘#dir’ section tells the system that the problem is non-linear in a two-dimensional Cartesian system. The ‘#schem’ section specifies the PDE to be elliptic type with one unknown. The ‘#nfe’ section defines the non-linear algorithm to be used. The ‘#ges’ section will link all the subroutines and library files related to quadrilateral mesh to the system. The ‘#solv’ section is the user input to specify the solver to be used, ‘sin’ means that we select a symmetric LU decomposition solver. The ‘#method’ section is another user option, here ‘outcore’ tells the system to store the gross stiff matrix on hard-disk, rather than in the RAM.

Appendix C. Input files for the model of seismic wave propagation in the Fuzhou basin

Here we show the complete input files which are used for generating the finite element source codes for the model of seismic wave propagation discussed in Section 5.2. Only two input files are needed here, the file ‘.pde’ for describing the PDE of Eq. (5.1), and the file ‘.nfe’ for defining the algorithms.
C.1. Input PDE file seismic.pde

The `.pde' file contains three main parts. They are indicated by the keywords ‘defi’, ‘func’, and ‘stif’, respectively:

```plaintext
defi
  disp u v w
  coor x y z
  func exx eyy ezz eyz exz exy
  shap %1 %2
  gaus %3
mate pe pv fu fv fw rou alpha 1.0e10;0.3;0;0;0;3000;0.6;
mass %1 rou
damp %1 rou*alpha
$C6 fact = pe/(1.+pv)/(1.-pv*2)
$C6 shear = (0.5-pv)
func
$C6 vol = (1.0d0)
$C6 factvol = fact*vol
exx=+[u/x]
eyy=+[v/y]
ezz=+[w/z]
eyz=+[v/z]+[w/y]
exz=+[u/z]+[w/x]
ey=+[u/y]+[v/x]
stif
  dist=+[exx;exx]*(1.-pv)*factvol+[exx;eyy]*pv*factvol
  +(exx;ezz)*pv*factvol+[eyy;exx]*pv*factvol+[eyy;eyy]*(1.-pv)*factvol
  +(eyy;ezz)*pv*factvol+[ezz;exx]*pv*factvol+[ezz;eyy]*pv*factvol
  +(ezz;ezz)*(1.-pv)*factvol+[eyz;eyz]*shear*factvol
  +[exz;exz]*shear*factvol+[exy;exy]*shear*factvol
load=+[u]*fu*vol+[v]*fv*vol+[w]*fw*vol

end
```
C.2. Input algorithm definition file newmark.nfe

The following lines are the input `.nfe` file for the Newmark algorithm.

defi
stif s
mass m
damp c
load f
type w
mdty l
step 0

equation
vect u1,v1,w1
read(s,unod) u1,v1,w1

matrix = [s]+[m]*a0+[c]*al
forc=[f]+[m]*u1]*a0+[m]*v1]*a2+[m]*[w1]*a3+[c]*u1]*a4+[c]*[v1]*a4

+ [c]*[w1]*a5

solution u
vect u,v,w,u1,v1,w1
read(s,unod) u1,v1,w1

[w]=([u]-[u1])*a0-[v1]*a2-[w1]*a3
[v]=[v1]+[w]*a7+[w1]*a6
write(s,unod) u,v,w
From these two input files the system generated all the source codes for the model shown in Section 5.2. The mass matrix and damp matrix are defined as a lump form by default. The seismic.pde file is similar to the '.pde' file for the linear case in Section 2. The keyword ‘fortran’ tells the system that we want to insert a Fortran program segment into the source code directly. The $o$, $aa$, $a0$, ..., $a7$ are parameters for Newmark scheme (Artuzi, 2005). By default, we use $o = 0.5$ which is common in many applications. The whole scheme of Newmark method is described jointly by the ‘equation’ part and ‘fortran’ sections.

Appendix D. A complete example of using the modeling-language based parallel FEM modeling environment

We provide here a complete example of using the automated parallel FEM modeling environment to solve a physical problem, which in this example is steady-state heat conduction in two dimensions (2D). We show how to use modeling language expressions to describe the partial differential equations and to generate parallel finite element codes in this system. We also provide testing results to validate and benchmark our parallel computing system.

Consider a solid disk with constant thermal properties. The outer rim is imposed to a constant temperature, where the center of the disk is connected to a rod through which heat is conducted away. The 2D steady-state temperature field is given by

\[-\nabla \cdot (k \nabla u) = q \]  

where $u$ is the temperature, $k$ the thermal conductivity, $q$ denotes heat source/sink, and $\nabla \cdot$ and $\nabla$ are the divergence and gradient operators, respectively.

This is a Dirichlet problem. The weak form of (D.1) based on the Galerkin finite element method can be written as

\[(k \nabla u, \nabla \bar{u}) = (q, \bar{u}) \]  

where $\bar{u}$ is the virtual displacement of $u$.

To solve this problem using finite element method, we first specify the problem and the desired solving algorithms in two input files using the modeling language expressions. To describe the PDE and the weak form, we write the ‘heat.pde’ file as following:

```plaintext
Heat.pde

disp u
coor x y
shap %1 %2
gaus %3
mate ek q 1.0d0;4.0d0
func gux guy

func

gux=+[u/x]
guy=+[u/y]

stif
dist=+[gux:gux]*ek +[guy:guy]*ek
load=+[u]*q
end
```
The keywords 'coor', 'shap', 'gaus', 'func' are identical to those we described in Section 3. The keyword 'mate' provides the system default material factors. 'Func' and 'stif' sections are still definition of functions defined after keyword 'func' and the weak form of Eq. (D.2).

We need a '.nfe' file to tell our system explicitly how to solve this equation. Because Eq. (D.1) is a typical elliptic type PDE, we can copy the default '.nfe' file from our system algorithm library and rename it to 'heat.nfe'. We list this file as following.

```
Heat.nfe

defi
stif s
mass m
load f
type e
mdty l
step 0

equation
matrix = [s]
forc=[f]
solution u
write(s,unod) u

end
```

Finally, we need an file that describes the computing domain, the actual finite element type, input and output data structure, etc. This file can be generated automatically by selecting from the user menu provided by the system, here it is named 'heat.gio'.

```
Heat.gio
2dxy 2 2 8 10
lesa 0 2 2 u
lesb 0 1 2 ux uy
#
  element y
lesa
  aeq4 4 ek q
  all2 2 alpha eb
#
  lesb
  bcq4 4 ek q
#
  matedata
  lesa aeq4
  1.0d0;4.0d0;
  lesa all2
  0.d0;0.d0;
#
```

From the input files 'heat.pde' and 'heat.nfe', the system generates automatically the source Fortran codes for serial and parallel finite element computation. The system also generated interface source codes for pre- and post-processing from the 'heat.gio' file.

Take the non-dimensional value of the disk’s radius to be 10, and the non-dimensional temperature on the outer rim to be 120. If \( q \) in (D.1) has a value of 4 and \( k \) equals to 1, then the analytical solution of Eq. (D.1) is

\[
   u = x^2 + y^2 + 20 \quad \text{(D.3)}
\]

Table 1 shows the analytic solution compared with the numerical solutions from serial and parallel FEM computations using the FEM codes generated by the system. The results are shown in Fig. 9. Please see Table 1 for
Table 1
Comparison of analytical and FEM results of Eq. (D.1)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>19.996</td>
<td>31.107</td>
<td>42.219</td>
<td>53.330</td>
<td>64.445</td>
<td>75.557</td>
<td>86.666</td>
<td>97.778</td>
<td>108.89</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>20.003</td>
<td>31.114</td>
<td>42.225</td>
<td>53.334</td>
<td>64.444</td>
<td>75.557</td>
<td>86.666</td>
<td>97.778</td>
<td>108.89</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>30.111</td>
<td>42.222</td>
<td>53.333</td>
<td>64.444</td>
<td>75.556</td>
<td>86.667</td>
<td>97.778</td>
<td>108.89</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>30.111</td>
<td>42.222</td>
<td>53.333</td>
<td>64.444</td>
<td>75.556</td>
<td>86.667</td>
<td>97.778</td>
<td>108.89</td>
<td></td>
</tr>
</tbody>
</table>

1. Analytical solution; 2, serial code, 8842 triangle elements; 3, serial code, 19,086 triangle elements; 4, serial and parallel code, 293,170 triangle elements; 5, parallel code, 809,568 triangle elements.

Fig. 9. These two graphs are description of our computing domain and the parallel finite element computing result, which has a value of 20 at the center of the disk, and increases non-linearly to 120 at the outer rim.

the details of the numerical comparison and benchmark results.

All the parallel and serial source code generated by for this example, and the executable files, are available from http://hpcc.gucas.ac.cn/benchmark/temp.

References


