Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2013, Article ID 764237, 14 pages
http://dx.doi.org/10.1155/2013/764237

Research Article

A Comparative Study on Different Parallel Solvers for Nonlinear Analysis of Complex Structures

Lei Zhang,[1] Guoxin Zhang,[1] Lixiang Wang,[2] Zhaosong Ma,[2] and Shihai Li[2]

[1] *State Key Laboratory of Simulation and Regulation of Water Cycle in River Basin, China Institute of Water Resources and Hydropower Research, Beijing 100038, China*
[2] *Institute of Mechanics, Chinese Academy of Sciences, Beijing 100190, China*

Correspondence should be addressed to Lei Zhang; zhangl@iwhr.com

Received 18 July 2013; Accepted 2 September 2013

Academic Editor: Zhiqiang Hu

The parallelization of 2D/3D software SAPTIS is discussed for nonlinear analysis of complex structures. A comparative study is made on different parallel solvers. The numerical models are presented, including hydration models, water cooling models, modulus models, creep model, and autogenous deformation models. A finite element simulation is made for the whole process of excavation and pouring of dams using these models. The numerical results show a good agreement with the measured ones. To achieve a better computing efficiency, four parallel solvers utilizing parallelization techniques are employed: (1) a parallel preconditioned conjugate gradient (PCG) solver based on OpenMP, (2) a parallel preconditioned Krylov subspace solver based on MPI, (3) a parallel sparse equation solver based on OpenMP, and (4) a parallel GPU equation solver. The parallel solvers run either in a shared memory environment OpenMP or in a distributed memory environment MPI. A comparative study on these parallel solvers is made, and the results show that the parallelization makes SAPTIS more efficient, powerful, and adaptable.

1. Introduction

Complex structures are largely employed in engineering practice in a variety of situations and applications, for example, water resources and hydropower engineering, mining engineering, and traffic engineering. An analysis of such structures is not possible by empirical methods, and moreover in situ experimental studies are costly. Recent advances in numerical techniques have provided the finite element method (FEM) for analysis of much more complex systems in a much more realistic way. The FEM can model the complex behavior of concrete without limitations caused by complexity.

Many authors have studied the nonlinear response of concrete structures using the FEM focusing on the three-dimensional elastoplastic problem [1–12]. Most authors consider the effect of creep [4–8], while others consider cracking [9–12], with regard to nonlinear analysis of structures. Generally, the models are extremely computation intensive. Many engineers complain about the high computational costs. As a result, some nonlinear analysis codes have to be given up. The situation has changed since the arrival of a variety of high performance computers and the advances in parallel computing techniques, that is, parallel algorithms and parallel platforms. Parallel algorithms on different platforms, that is, algorithms utilizing OpenMP and MPI, are studied by [13–16]. Recently, the GPU high performance computing has been popular. A parallel GPU equation solver is introduced into SAPTIS as well. In these respects, we should be able to model more complex effects like hydration, water cooling, cracking, creep, autogenous deformation, and so forth.

The aim of this work is the prediction of concrete behaviors in different situations as well as the improvement of the runtime in the nonlinear analysis program structure analysis program for temperature and induced stress SAPTIS. A composition of models is presented in terms of the former, and a comparative study on different parallel solvers is made for the sake of the latter. The paper does not address the issue in mathematics itself, but focuses on the application of current algorithms. The goal is to develop practical and efficient parallel strategies for nonlinear analysis of complex structures.

In this study, we will arrange the contents as follows. In the first section, a composition of models and its basic

formulations are introduced. Then, the parallel strategies as well as the three parallel solvers are presented. Furthermore, the performance of the newly developed parallel code is tested on different computers. Finally, conclusions regarding the parallelized SAPTIS are reported.

2. Models for Nonlinear Analysis of Concrete Structures

The nonlinear analysis of the model takes into consideration the effects of hydration, water cooling, modulus changes, creep, and autogenous deformation. An appropriate model for these effects as well as the FEM equations is used for building the nonlinear system.

2.1. Thermal Simulation

2.1.1. Hydration Models. Hydration of concrete brings heat. Hereby it can be modeled using five models in SAPTIS, namely, exponential model, hyperbolic model, complex model, hydration degree model, and table model [17]. We put a superscript "+" as heat increasing, and the first four models can be explicitly formulated as follows.

Exponential Model 1:

$$Q^{+}(\tau) = Q_0\left(1 - e^{\alpha\tau}\right). \tag{1}$$

Exponential Model 2:

$$Q^{+}(\tau) = Q_0\left(1 - e^{\alpha\tau^{\beta}}\right). \tag{2}$$

Hyperbolic Model:

$$Q^{+}(\tau) = \frac{Q_0\tau^{\beta}}{\alpha + \tau^{\beta}}. \tag{3}$$

Complex Model:

$$Q^{+}(\tau) = Q_1\left(1 - e^{-\alpha_1\tau^{\beta_1}}\right) + \frac{Q_2\tau^{\gamma}}{n + \tau^{\gamma}}. \tag{4}$$

Hydration Degree Model:

$$\begin{aligned} Q^{+}(\tau) &= Q_{01}\frac{t_e^m}{n + t_e^m} + Q_{02}\left(1 - e^{\alpha t_e^{\beta}}\right), \\ t_e &= \int_0^{\tau} \exp\left[\frac{E_a}{R}\left(\frac{1}{273 + t_R} - \frac{1}{273 + t}\right)\right] dt. \end{aligned} \tag{5}$$

Table Model. Calculate the hydration heats and increments by splines based on experimental data.

In (1) ~ (5), $Q^{+}(\tau)$ is the adiabatic temperature rise regarding hydration at age τ, Q_i $(i = 0, 1, 2, 01, 02)$ are heat parameters, α, β, γ, m, and n are parameters regarding heating rate, t_e is an equivalent age, E_a, R, and t_R are activation parameters.

2.1.2. Water Cooling Models. There are two water cooling models in SAPTIS: one is a fine model and the other is an equivalent model. Derivations and formulations of the two models are described in detail in [18]. The cooling pipes take away heat, which can be written as:

$$Q^{-}(\tau) = T_w + (T_0 - T_w)\,\varphi(\tau), \tag{6}$$

where $Q^{-}(\tau)$ is the adiabatic temperature drop caused by the cooling system, T_w is the temperature of water at entry, T_0 is the initial temperature of concrete, which will be given in (8), and $\varphi(\tau)$ is the thermal distribution function.

In the fine model, a local grid refinement of the cooling pipes is needed. In every single time step, thermal distribution along the cooling pipes is obtained by calculating the heat exchange between the cooling pipes and the concrete, and the thermal field of concrete is therefore obtained. The fine model can accurately simulate the thermal distribution around the cooling pipes, but it needs a grid refinement, which inevitably enlarge the calculation scale.

The equivalent model does not take into consideration the location of pipes. An average concept is exploited by inputting the pipe spacing, the flux, the temperature, and the time of watering and calculating a mean effect of water cooling. The equivalent model can reduce calculation time without local grid refinements and ensure an average precise, but it does not consider the thermal distribution along the cooling pipes and it cannot obtain the thermal gradient near them.

2.1.3. Basic Thermal Model. The equilibrium equation of heat transfer can be written as:

$$\lambda\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2}\right) + c\rho\left(\frac{\partial\theta}{\partial\tau} - \frac{\partial T}{\partial\tau}\right) = 0. \tag{7}$$

The initial condition can be written as:

$$T = T_0(x, y, z) \quad (\tau = 0). \tag{8}$$

The boundary conditions are

$$\begin{aligned} T &= T_b \quad \text{Dirichlet B.C.}, \\ -\lambda\left(\frac{\partial T}{\partial n}\right) &= q \quad \text{Neumann B.C.}, \\ -\lambda\left(\frac{\partial T}{\partial n}\right) &= \beta(T - T_a) \quad \text{Mixed B.C.} \end{aligned} \tag{9}$$

In (7) ~ (9), τ represents age, λ is thermal conductivity [kJ/(m·h·°C)], ρ stands for density [kg/m^3], c is specific heat capacity [kJ/(kg·°C)], a is thermal diffusivity and $a = \lambda/(\rho c)$ [m^2/h], θ stands for total adiabatic temperature incremental and $\theta = Q^{+}(\tau) + Q^{-}(\tau)$ [°C]; T is temperature [°C], T_0 is the initial temperature of concrete [°C], T_b is the fixed boundary temperature [°C], q represents heat flux [kJ/(m^2·h)], T_a denotes ambient temperature in natural convection conditions and adiabatic temperature of boundary layer in forced convection conditions [°C] and β is heat transfer coefficient in surface [kJ/(m^2·h·°C)].

2.2. Creep Simulation. SAPTIS can simulate thermal distribution as well as stress field in concrete casting. Apart from linear, nonlinear, and elastoplastic simulation, creep and hardening of concrete are also required.

2.2.1. Modulus Models. Modulus and strength of concrete increase to peaks with age due to hydration effect after pouring. The increasing rate of modulus and strength relates to temperatures of concrete and environment. To simulate such effect, SAPTIS adopts several modulus models [19].

Exponential Model:

$$E(\tau) = E_c\left(1 - e^{-\alpha\tau^{\beta}}\right). \tag{10}$$

Hyperbolic Model:

$$E(\tau) = E_c\frac{\tau^{\beta}}{\alpha + \tau^{\beta}}. \tag{11}$$

Complex Model:

$$E(\tau) = E_{c1}\left(1 - e^{-\alpha\tau^{\beta}}\right) + E_{c2}\frac{\tau^{\gamma}}{n + \tau^{\gamma}}. \tag{12}$$

Aging Degree Model:

$$\begin{aligned} E(t_e) &= E_{c1}\left(1 - e^{-\alpha t_e^{\beta}}\right) + E_{c2}\frac{t_e^{\gamma}}{n + t_e^{\gamma}}, \\ t_e &= \int_0^{\tau}\exp\left[\frac{E_a}{R}\left(\frac{1}{273 + \tau_R} - \frac{1}{273 + \tau}\right)\right]dt. \end{aligned} \tag{13}$$

Table Model. Calculate the modulus by interpolation based on experimental data.

In (10) ~ (13), $E(\tau)$ is modulus at age τ and α, β, γ, m, and n are parameters regarding hardening rate.

In practical modeling, choose one model from (10) ~ (13), and the results show that any of the models can well simulate the process of modulus growth of concrete.

2.2.2. Creep Model. Zhu [19] proposed a creep formula for loading at different ages after a rigorous derivation:

$$\begin{aligned} C(t,\tau) = {} & \left(A_1 + A_2\tau^{-\alpha_1}\right)\left(1 - e^{-k_1(t-\tau)}\right) \\ & + \left(B_1 + B_2\tau^{-\alpha_2}\right)\left(1 - e^{-k_2(t-\tau)}\right) \\ & + De^{-k_3 t}\left(1 - e^{-k_3(t-\tau)}\right), \end{aligned} \tag{14}$$

where $C(t,\tau)$ represent creep of concrete, τ stands for age, $(t-\tau)$ denotes loading time, and A_1, A_2, B_1, B_2, D, k_1, k_2, k_3, α_1, and α_2 are regression coefficients. An increment method is employed to calculate the creep influence on deformation and stress. Modulus and creep of concrete in several engineerings are shown in Tables 1 and 2.

2.2.3. Autogenous Deformation and MgO Linear Expansion Models. Input the ages of concrete and corresponding strains, and autogenous deformations can be obtained from spline interpolation.

Autogenous deformations exist in MgO concrete and can somehow offset the shrinkage due to thermal decrease, which is beneficial for preventing dam cracking. MgO concrete has been applied in a few projects, and good performance has been achieved [20, 21]. We propose two formulas to calculate autogenous deformations of MgO concrete from practice.

Table 1: Modulus in engineerings.

Engineering	Concrete	Regression equations (E [GPa], τ [d])
Laxiwa arch dam	Normal	$E = 39.05[1 - \exp(-0.51\tau^{0.46})]$
Xiaowan arch dam	Normal	$E = 34.00[1 - \exp(-0.36\tau^{0.34})]$
Xiluodu arch dam	Normal	$E = 39.10[1 - \exp(-0.23\tau^{0.57})]$
Jinping arch dam	Normal	$E = 35.40[1 - \exp(-0.342\tau^{0.402})]$
Ertan arch dam	Normal	$E = 30.24[1 - \exp(-0.222\tau^{0.474})]$
Shapai arch dam	RCC	$E = 26.50\tau/(6.90 + \tau)$
Zhaolaihe arch dam	RCC	$E = 32.50\tau/(8.30 + \tau)$
Bailianya arch dam	Normal	$E = 33.50\tau/(7.30 + \tau)$
Longtan gravity dam	RCC	$E = 44.54[1 - \exp(-0.43\tau^{0.455})]$
Xiangjiaba gravity dam	Normal	$E = 46.57\tau/(1.31 + \tau)$
Jinghong gravity dam	RCC	$E = 39.80\tau/(1.96 + \tau)$
Guangzhao gravity dam	RCC	$E = 3.70[1 - \exp(-0.49\tau^{0.290})]$

Dynamic Formula:

$$\frac{d\varepsilon(\tau)}{d\tau} = \alpha\varepsilon_0\left[1 - \frac{\varepsilon(\tau)}{\varepsilon_0}\right]^{(\beta_1 + \beta_2 T + \beta_3 T^2)} e^{-(\gamma/(T+273))}. \tag{15}$$

Residual Formula:

$$\begin{aligned} \frac{d\varepsilon_g(\tau)}{d\tau} &= \sum_{i=1}^{3} C_i\alpha_i T^{\beta_i}\left[1 - \frac{\varepsilon_g(\tau)}{\varepsilon_{g0}}\right], \\ C_3 &= C_0 - C_1 - C_2. \end{aligned} \tag{16}$$

In (15) and (16), C_0, C_1, and C_2 are parameters related to final expansion in $[\mu]$ and α_1, α_2, α_3, β_1, β_2, β_3, and γ are coefficients.

2.3. Finite Element Formulation

2.3.1. Element Types. SAPTIS is 2D/3D software for general analysis. In 2D space, elements of triangle with 3–6 nodes and quadrilateral with 4–9 nodes are employed. In 3D space, elements consist of tetrahedra with 4–11 nodes, wedges with 6–16 nodes, and hexahedra with 8–21 nodes. The shape functions of these elements can be found in and elaborated upon by [20, 21]. Besides, a one-dimensional (1d) element used for links and prestressed anchors is defined in the software.

Two definitions for this 1d element are emphasized: one is an explicit definition, in which the element is a real bar connected by its two nodes, and it has its own stiff matrix (Figures 1(a), 1(b), and 1(c)). The other is an implicit definition and it treats the element as a virtual bar, which does not have a geometric entity in representation, but its stiff matrix is

Table 2: Creep coefficients in engineerings.

Engineering	Concrete	Coefficients (k_1, k_2, k_3, A_1, A_2, α_1, B_1, B_2, α_2, and D)
Laxiwa arch dam	Normal	0.0178, 0.6483, 0.0266, 9.0540, 42.2266, 31.0377, 0.0000, 43.8010, 0.3994, 20.2991
Xiaowan arch dam	Normal	0.7313, 0.0421, 0.0222, 0.0000, 7.4472, 0.4479, 0.4811, 2.4581, 2.9923, 2.4952
Xiluodu arch dam	Normal	0.0226, 0.7303, 0.1442, 13.0595, 6.6032, 50.5629, 3.0621, 9.2789, 0.0315, 52.3952
Jinping arch dam	Normal	0.7179, 0.0221, 0.8149, 0.0000, 70.0000, 0.4915, 0.0000, 43.8500, 0.3915, 70.0000
Shapai arch dam	RCC	0.0153, 0.4785, 0.7976, 0.0000, 47.1714, 0.2374, 0.0000, 107.4084, 0.4078, 150.0000
Longtan gravity dam	RCC	0.0143, 0.7653, 0.0742, 0.0000, 13.2130, 0.2376, 0.5435, 55.7605, 0.6757, 16.6353
Xiangjiaba gravity dam	Normal	1.0000, 0.0194, 0.0666, 7.6095, 69.9867, 1.0995, 10.9448, 1.7950, 22.1736, 16.2207
Guangzhao gravity dam	RCC	0.0561, 0.6408, 0.0080, 0.0000, 33.7144, 0.3831, 0.0000, 52.9215, 0.6150, 14.8707

Note: unit of creep C in [10^{-6}/MPa]; unit of time t, τ in [d].

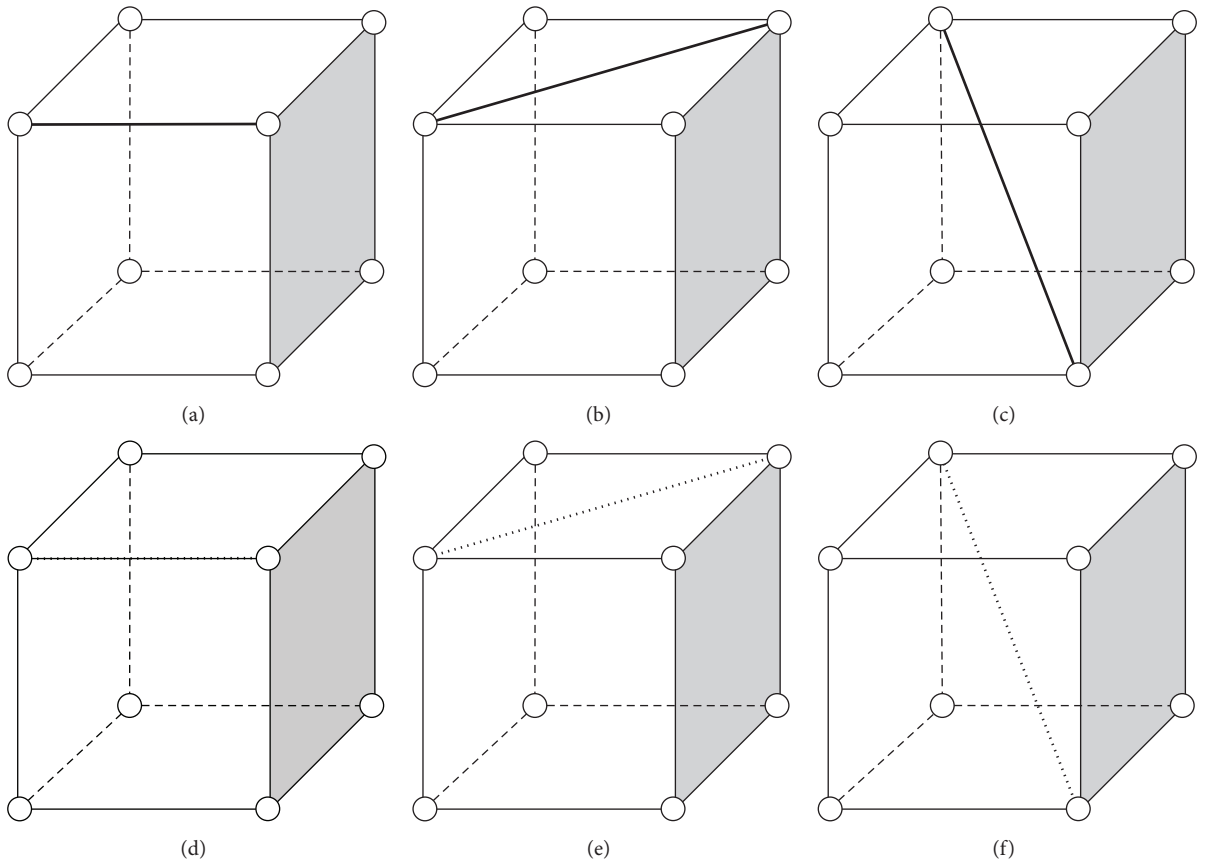


Figure 1: Definitions and connection types of 1d element.

nominally added to its neighboring element(s) (Figures 1(d), 1(e), and 1(f)). In both definitions, there are three connection types in terms of hexahedral neighbors, that is, edge connection (Figures 1(a) and 1(d)), face diagonal connection (Figures 1(b) and 1(e)), and body diagonal connection (Figures 1(c) and 1(f)).

2.3.2. Finite Element Formulation of Heat Transfer. By using the variation formulation, the equilibrium equation of heat transfer in (7) can be transformed into the following matrix form:

$$[C]\left\{\dot{T}\right\} + [K]\{T\} = \{Q\}, \tag{17}$$

where $[C]$ is the matrix for specific heat capacity, $[K]$ is the matrix for heat conductivity, and $\{Q\}$ is the total heat flux vector for internal hydration heat and heat convection.

The matrices of $[C]$ and $[K]$ and vector $\{Q\}$ shown in (17) are

$$[C] = \int_\Omega \rho c [N]^T [N]\, d\Omega,$$

$$[K] = \int_\Omega \lambda \left(\left[\frac{\partial N}{\partial x}\right]^T \left[\frac{\partial N}{\partial x}\right] + \left[\frac{\partial N}{\partial y}\right]^T \left[\frac{\partial N}{\partial y}\right] + \left[\frac{\partial N}{\partial z}\right]^T \left[\frac{\partial N}{\partial z}\right] \right) d\Omega + \int_\Gamma \beta [N]^T [N]\, d\Gamma,$$

$$\{Q\} = \int_{\Omega} [N]^T \rho \left(\frac{\partial \theta}{\partial \tau}\right) d\Omega + \int_{\Gamma} [N]^T q d\Gamma + \int_{\Gamma} [N]^T \beta T_a d\Gamma, \tag{18}$$

where $[N]$ is the matrix for shape function, Ω is the finite domain, and Γ is the boundary of Ω.

2.3.3. Finite Element Formulation of Stress Analysis. An incremental method is employed to analyze the deformation and stress of concrete. The sum of elastic and creep strain in concrete is proportional to a sustained load under ordinary service stress level, while the thermal strain is proportional to the temperature rise. The autogenous deformation in MgO concrete can be worked out by the formulas we propose. The incremental method has been elaborated upon in [12, 22], and we only present its finite element equation as follows:

$$\{\Delta F\}^e = \iiint [B]^T \{\Delta\sigma\} \, dx \, dy \, dz, \tag{19}$$

where $\{\Delta F\}^e$ is the force increment and $[B]^T$ is the strain matrix. $\{\Delta\sigma\}$ is the stress increment caused by elastic, creep, thermal, and autogenous strains, which can be written as

$$\{\Delta\sigma\} = \left[\overline{D}\right]\left(\left\{\Delta\varepsilon^E\right\} - \{\eta\} - \left\{\Delta\varepsilon^T\right\} - \left\{\Delta\varepsilon^A\right\}\right), \tag{20}$$

where $[\overline{D}]$ is the instant stress matrix, $\{\Delta\varepsilon^E\}$ is elastic strain increment, $\{\eta\}$ relates to creep strain increment, $\{\Delta\varepsilon^T\}$ stands for thermal strain increment, and $\{\Delta\varepsilon^A\}$ represents autogenous strain increment.

Put (20) into (19), and we can get an equilibrium form

$$\{\Delta F\}^e = [K]^e\{\Delta u\}^e - \left\{\Delta P^C\right\}^e - \left\{\Delta P^T\right\}^e - \left\{\Delta P^A\right\}^e, \tag{21}$$

where $[K]^e$ is the stiff matrix of element, $\{\Delta u\}^e$ is displacement increment of element, and $\{\Delta P^C\}^e$, $\{\Delta P^T\}^e$, and $\{\Delta P^A\}^e$ are nodal load increments caused, respectively, by creep, temperature, and autogenous deformation. The matrix and vectors can be summarized as

$$\begin{aligned}
[K]^e &= \iiint [B]^T \left[\overline{D}\right] [B] \, dx \, dy \, dz, \\
\left\{\Delta P^C\right\}^e &= \iiint [B]^T \left[\overline{D}\right] \{\eta\} \, dx \, dy \, dz, \\
\left\{\Delta P^T\right\}^e &= \iiint [B]^T \left[\overline{D}\right] \left\{\Delta\varepsilon^T\right\} dx \, dy \, dz, \\
\left\{\Delta P^A\right\}^e &= \iiint [B]^T \left[\overline{D}\right] \left\{\Delta\varepsilon^A\right\} dx \, dy \, dz.
\end{aligned} \tag{22}$$

After the assembly of all element stiffness matrices, a general finite element equation in its global form can be expressed as

$$[K]\{\Delta u\} = \left\{\Delta P^F\right\} + \left\{\Delta P^C\right\} + \left\{\Delta P^T\right\} + \left\{\Delta P^A\right\}, \tag{23}$$

where $[K]$ is global stiffness matrix, $\{\Delta u\}$ is global displacement vector, $\{\Delta P^F\}$ is global increment vector induced by external force, and $\{\Delta P^C\}$, $\{\Delta P^T\}$, and $\{\Delta P^A\}$ are global increment vectors caused by creep, temperature, and autogenous deformation, respectively.

3. Parallelization Strategy

Since the 1980s, a lot of work has been done in parallel and distributed computing for structural analysis. Parallel processing benefits such analysis a lot by using two different strategies [23].

(1) The analysis problem may be subdivided by geometrically dividing the idealization into a number of subdomains: explicit decomposition approach, also called substructure approach.

(2) Alternatively the system of equations for the whole structure may be assembled and solved in parallel without recourse to a physical partitioning of the problem: implicit domain decomposition (IDD) approach, or global approach.

It should be noted that such strategies with domain decomposition techniques are not general solution procedures and are specialized for particular applications. Furthermore, what can be parallelized are [24] (a) input problem characteristics, (b) assembly, (c) boundary conditions, (d) solution of algebraic equations, and (e) postprocessing. The fourth item is the most important to parallelize. Additionally, Gummadi and Palazotto [25] indicated that in a typical linear static FEA, the most consuming operation is the solution of linear equations. We recognize from our own experience that the time used for solving equations can reach 70%~90%. An optimization in linear equation solvers is required to largely improve computing efficiency.

3.1. Linear Equation Solvers. Linear system of large equations is solved by two kinds of solvers, that is, the direct solvers (e.g., Gauss) and the iterative ones (e.g., PCG).

The direct solvers are applicable with accuracy assurance for any nonsingular linear system with an appropriate scale and density. When solving a large sparse system with many 0's, a direct solver is particularly consuming. Additionally it cannot ensure the accuracy in terms of rounding errors. The parallelization for direct solvers is only suitable for tridiagonal matrices, block tridiagonal matrices and banded matrices. For a general sparse linear system, the sparse equation solvers are popular with multifrontal algorithms and supernode technology.

The iterative solvers overcome the shortcomings of the direct ones and retain the sparsity of coefficient matrix. With small storage and computation, they have more obvious advantages especially for large, sparse, asymmetric, and seriously ill-conditioned matrices. At present, the preconditioned conjugate gradient (PCG) solver on behalf of the Krylov subspace solvers is among the most popular iterative solvers in engineering and sciences. Classical iterative solvers (e.g., Jacobi, GS, SOR, and SSOR) with their improved ones are used for preconditions in most situations.

In SAPTIS, four linear equation solvers are adopted:

(1) the preconditioned (i.e., SSOR) conjugate gradient solver,

(2) the preconditioned (e.g., CG, CGS, BiCGSTAB, etc.) Krylov subspace solver,

(3) the sparse equation solver,

(4) the GPU solver.

3.2. Key Technique for Parallel Programming. A parallel procedure is the implement of a parallel algorithm on a parallel computer or cluster. The parallel algorithm is designed for a parallel computing model, which is abstracted from different parallel systems or architectures of computer models. There are three programming models for parallel computing at present: (1) shared memory programming model; (2) message passing programming model; (3) hybrid programming model, that is the hybrid of shared memory and message passing programming models. They correspond to three parallel architectures: (1) shared memory system, for example, Symmetric Multiprocessor (SMP); (2) distributed memory system, for example, clusters and (Massive Parallel Processing MPP); (3) distributed/shared memory system, for example, multicore clusters and SMP clusters.

To implement a parallel procedure, four major problems are faced with. The first to be concerned is what parts of the algorithm can be parallelized. These parts should be decomposed and executed by different processes, and a judgment for decomposition is whether there exists data competition. The second problem is the strategy for decomposition. There are two strategies: one is task strategy, another is data strategy. In task strategy, the procedure is decomposed into different tasks, whose dependence on whom should be noted. Then the tasks are scheduled to avoid mutual interference. Such parallelization strategy belongs to a coarse one and demands high independence of specific problems. In data strategy, the data space of procedure is decomposed into different areas. Each processor takes in charge of its own area. Obviously, this strategy is well suited for parallelization of finite element analysis. The third to be concerned is the programming model. This can be determined by the parallel architecture. The last one is the programming method. In scientific computing, data parallelization strategy is generally employed. As a result, the looping or (Single Program Multiple Data SPMD) programming is adopted. In looping programming, loops without competition are distributed into different processors. In SPMD programming, all processors execute the same procedure, but they use different data, which are transferred and shared among the processors via communication.

In SAPTIS, these problems are solved one by one as follows.

(i) The solution of algebraic equations should be urgent to parallelize.

(ii) For FEA parallelization, the data strategy should be exploited.

(iii) The message passing programming model should be employed if based on MPI, and the multithread programming model should be employed if based on OpenMP.

(iv) Data parallelization strategy determines that the looping or SPMD programming method should be used.

3.3. Implementation. The implementation is based on reference books [26] and work made by [13–16, 27–29].

3.3.1. Parallel Preconditioned Conjugate Gradient Solver Based on OpenMP. The parallel preconditioned conjugate gradient (PCG) solver uses symmetric successive over relaxation (SSOR) technique and runs on the OpenMP-based platform.

The bottlenecks of performance for the PCG solver are the matrix and vector operations like $\mathbf{W}^{-1}\mathbf{b}$ and $\mathbf{W}^{-T}\mathbf{b}$, which costs over 80% of the computation time. Therefore, the parallelization strategy for the PCG solver based on OpenMP is to parallelize the loops containing such operations without data competition by controlling statements of OpenMP. After such programming process, parallelization of matrix-vector multiplication and inner product operation is achieved, and the data formats and compile options are optimized simultaneously.

The PCG solver is parallelized in an algorithm level, which requires frequent communication on computers of distributed memory systems. Thus, such parallel algorithm is more suitable for computers of shared memory systems, and an OpenMP-based platform is employed.

The parallelization procedures for the PCG solver can be summarized as follows:

(i) first, the matrix and vector operations are parallelized, including $\mathbf{AP}$ and $\mathbf{A}^T\mathbf{P}$, where the vector $\mathbf{P}$ is generated by specific algorithm;

(ii) then, the inner product operation is parallelized;

(iii) furthermore, the vectors are updated;

(iv) finally, the preconditions are calculated (if necessary).

3.3.2. Parallel Preconditioned Krylov Subspace Solver Based on MPI. The Krylov subspace K_m is defined by means of the square matrix $\mathbf{A} \in \Re^{n\times n}$, the initial vector $\mathbf{b} \in \Re^n$, and the positive scalar m and $K_m = \text{Span}(\mathbf{b}, \mathbf{Ab}, \ldots, \mathbf{A}^{m-1}\mathbf{b})$. The original system $\mathbf{Ax} = \mathbf{b}$ is reduced by using the orthogonal transformation $\mathbf{Q}$, obtaining the tridiagonal system $T(\mathbf{Q}^T\mathbf{x}) = \mathbf{Q}^T\mathbf{b}$. By the integration of the solution of the tridiagonal system and the generation of the matrices $\mathbf{Q}$ and T, let us obtain a complete algorithm to solve the original system, where we obtain a better solution in each step [29].

The parallel preconditioned Krylov subspace (PKS) solver uses iterative techniques, for example, CG, CGS, BiCGSTAB, GMRES, TFQMR, and so forth, and runs on the MPI-based platform.

For the parallelization of the PKS solver, a domain decomposition method is employed. The "divide and conquer" strategy is used, in which the finite analysis domain is divided into several subdomains and each subdomain is in the charge of one or several processors. All the processors solve the whole problem independently and interactively, and synergistically and simultaneously.

Based on the distributed memory environment of parallel computing, the PKS solver is paralleled by using the SPMD programming method and the message passing programming model. Since the "divide and conquer" strategy is used,

```
extern "C" int VeksInit(UINT32 nElems, UINT32 nGlobalNodes, UINT32 aiElemConn[ ][8]);
// Funcition:                Initialize the solver
extern "C" int VeksSetK(double afK[ ][24][24]);
// Funcition:                Set element stiffness matrices
extern "C" int VeksSetDemF(double afF[ ][8][3]);
// Funcition:                Set element loads
extern "C" int VeksSetU(double afU[ ][3]);
// Funcition:                Set nodal displacements
extern "C" int VeksSetF(double afF[ ][3]);
// Funcition:                Set nodal loads
extern "C" int VeksFixU(char abU[ ][3], double afU[ ][3]);
// Funcition:                Set nodal displacements
extern "C" int VeksGetU(double afU[ ][3]);
// Funcition:                Get nodal displacements
extern "C" int VeksSetC(double afC[ ][8][8]);
// Funcition:                Set element conductivity matrices
extern "C" int VeksSetDemQ(double afQ[ ][8]);
// Funcition:                Set element thermal loads
extern "C" int VeksSetQ(double afQ[ ]);
// Funcition:                Set nodal thermal loads
extern "C" int VeksFixT(char abT[ ], double afT[ ]);
// Funcition:                Set nodal thermal constraints
extern "C" int VeksGetT(double afT[ ]);
// Funcition:                Get nodal temperatures
extern "C" _int64 VeksSolve(_int64 nMaxIter, double fTimeStep, double fUnbRatio, double fEps, unsigned int nPrint);
// Funcition:                Solve equations

// The procedures for GPU analysis
#include "Eks.h"
int nRet = VeksInit(nElems, nGlobalNodes, aiElemConn);
VeksSetK(afK);
VeksSetDemF(afDemF);
VeksFixU(abU, afU);
VeksSetC(afC); VeksSetDemQ(afDemQ);
VeksFixT(abT, afT);
VeksSolve(nSteps, fTimeStep, fUnbRatio, fEps, nStipePrint);
VeksGetU(afU);
VeksGetT(afT);
```

Algorithm 1

Table 3: Parameters for cooling pipe simulations.

Items	Concrete	Plastic pipe	Metal pipe
Materials	C40 concrete	Polyethylene	Steel
Physical properties	$\lambda = 6.26$ kJ/(m·h·°C) $\rho = 2430$ kg/m^3 $c = 0.97$ kJ/(kg·°C)	$\lambda = 1.73$ kJ/(m·h·°C) $\rho = 940$ kg/m^3 $c = 2.3$ kJ/(kg·°C)	$\lambda = 173$ kJ/(m·h·°C) $\rho = 7850$ kg/m^3 $c = 4.8 \times 10^5$ kJ/(kg·°C)
Geometrical properties	$R = 1.5$ m	$R_o = 16.0$ mm $R_i = 14.0$ mm	$R_o = 12.7$ mm $R_i = 11.2$ mm
Initial conditions	$T_0 = 20$°C	$T_0 = 0$°C	$T_0 = 0$°C
Boundary conditions	$q_b = 0$ kJ	$T_b = 0$°C	$T_b = 0$°C

the solution procedures of FEA equations consist of assembly of global matrix and parallel solving of all local equations. The SPMD programming model implies that the source codes of every process are nearly the same, in spite of the differences of data in different areas.

The parallelization procedures for the PKS solver can be summarized as follows:

(i) the global FEA equations are assembled;

(ii) the "divide and conquer" strategy is employed to divide the FEA domain into sub-domains;

(iii) local equations are formed based on sub-domains;

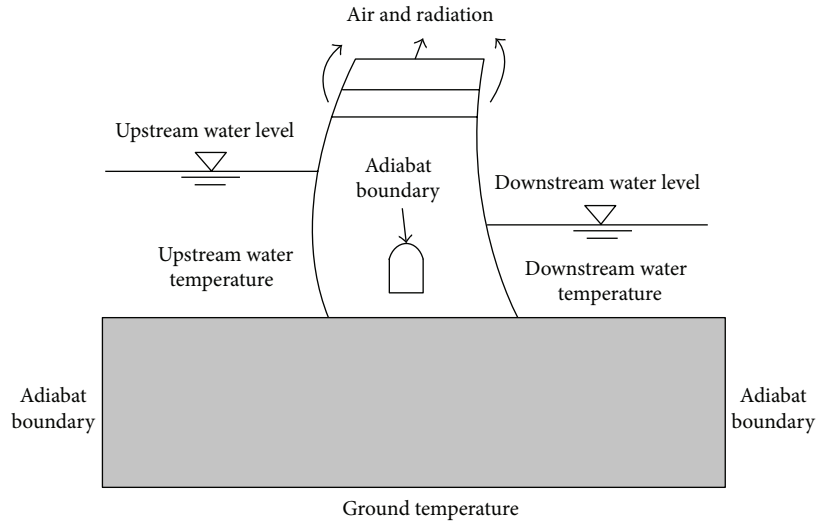


Figure 2: Boundary conditions of a dam.

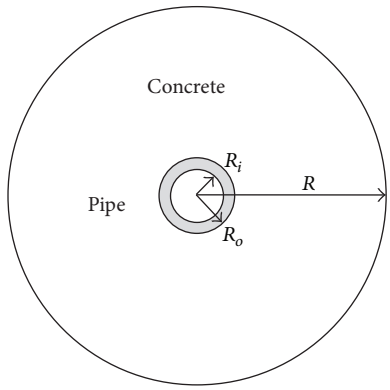


Figure 3: The model of a cylinder water cooling pipe.

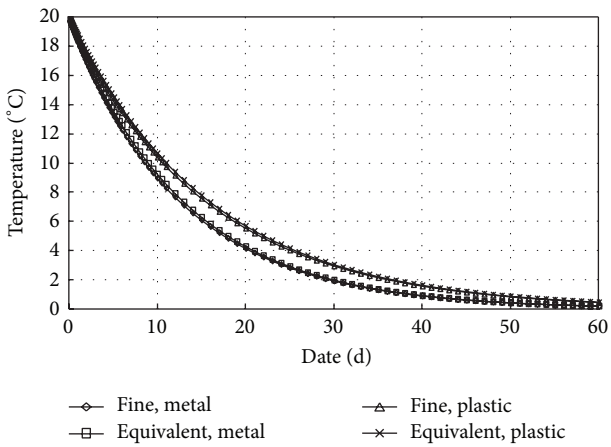


Figure 4: Comparison between the fine and equivalent models.

(iv) iterative techniques, for example, CG, CGS, BiCGSTAB, GMRES, TFQMR, and so forth, are paralleled and used to solve all the local equations.

Table 4: Parameters of ThinkCentre M Q45t.

Computer	ThinkCentre M Q45t
CPU	Intel Core 2
Compute capability	5 million DOFs
Clock rate	2.33 GHz
Processor	Q8200
Number of cores	4
Memory	4 G DDR3

3.3.3. Parallel Sparse Equation Solver Based on OpenMP. The parallel sparse equation (SE) solver uses a direct solving method and runs on the OpenMP-based platform.

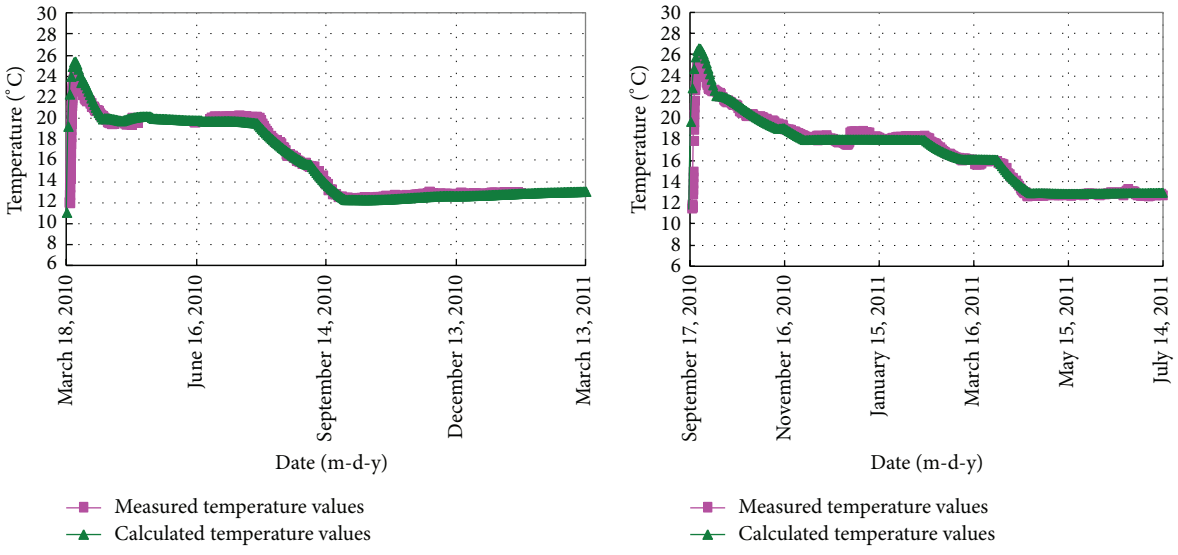
Algorithms based on iterative methods are not always suitable for specific structural analysis, for such algorithms don not always work since the matrix may not be well conditioned during the iterative process. The direct method has fewer problems in achieving solution convergence. In terms of these reasons, a sparse equation solving method is adopted as well in SAPTIS. This is a numerically stable parallel algorithm for solving ill-conditioned linear systems of equations.

The sparse methods are close to direct methods in essence, but they have also big differences. The rearrangements of matrices, incomplete decompositions, and multifront techniques have made the sparse methods more efficient than the direct ones.

The sparse equation solver is parallelized in CSR matrix format, which supports several types of matrices including real/imaginary matrices and symmetric/asymmetric matrices.

The parallelization procedures for the sparse equation solver are similar to those for the PCG solver. In this case, we will omit the discussions.

3.3.4. Parallel GPU Equation Solver. Recently, the high performance computing is quite popular, that is, the GPU computing. The graphics processing unit (GPU) has become



(a) 357 m Elevation (b) 408 m Elevation

Figure 5: Comparisons between computational temperatures and measured ones.

an integral part of today's mainstream computing systems. Over the past decade, there has been a marked increase in the performance and capabilities of GPUs. The advent of GPU computing technology greatly improves the computational performance of numerical simulations and the speedups reach tens to hundreds. Besides, a GPU is small, portable, and cheap. It costs little power.

The parallel GPU equation solver uses an iterative method. The strategies used for GPU implementation can be summarized as follows:

(1) replace the original CPU-based calculation functions with GPU kernels. A kernel is a function that runs on the GPU device;

(2) integrate as many GPU kernels into a large one as possible. This is to ensure the efficiency of GPU codes, for data transfer among different kernels is time-consuming.

Based on the strategies above, the functions shown in Algorithm 1 are introduced into SAPTIS. These functions as well as their library files have enabled SAPTIS to carry out GPU high performance computing.

The analysis procedures by using the GPU parallel solver are summarized as follows:

(1) include the "*Eks.h*" file into the source files;

(2) link the library files *Geks.Lib*;

(3) call the function *VeksInit* to initialize the GPU solver;

(4) call the function *VeksSetK* to transfer the element stiffness matrices into the GPU solver;

(5) call the function *VeksSetC* to transfer the element conductivity matrices into the GPU solver;

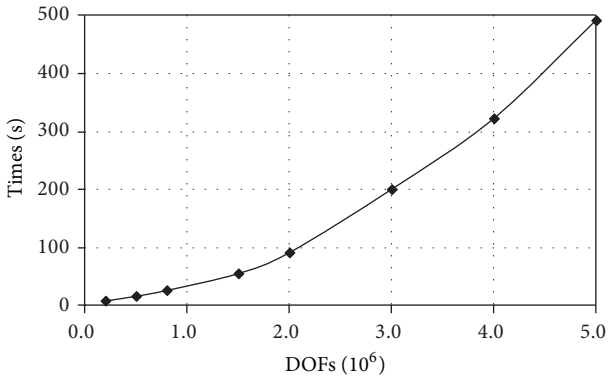


Figure 6: Serial codes test on ThinkCentre M Q45t.

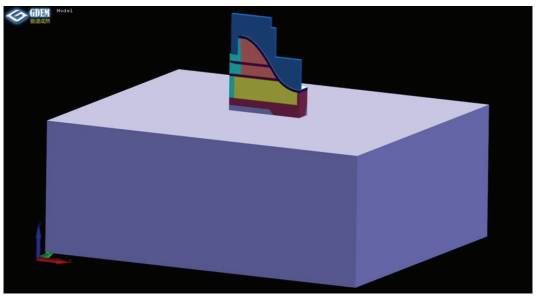
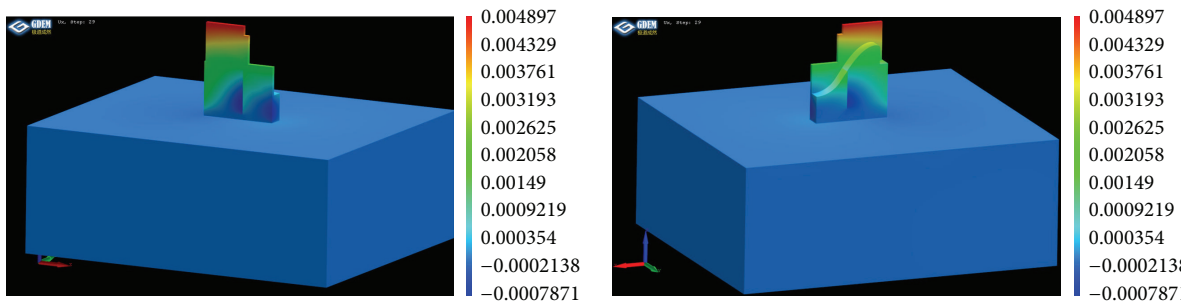


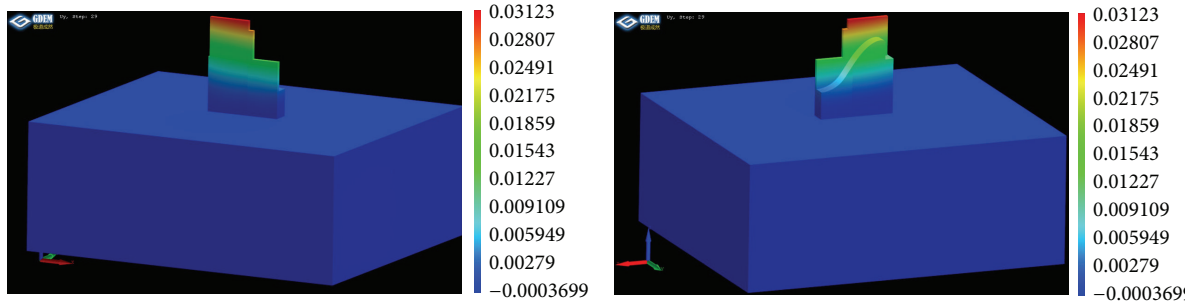
Figure 7: The model used for GPU simulation.

(6) call the function *VeksSetDemF* to transfer the loads vector into the GPU solver;

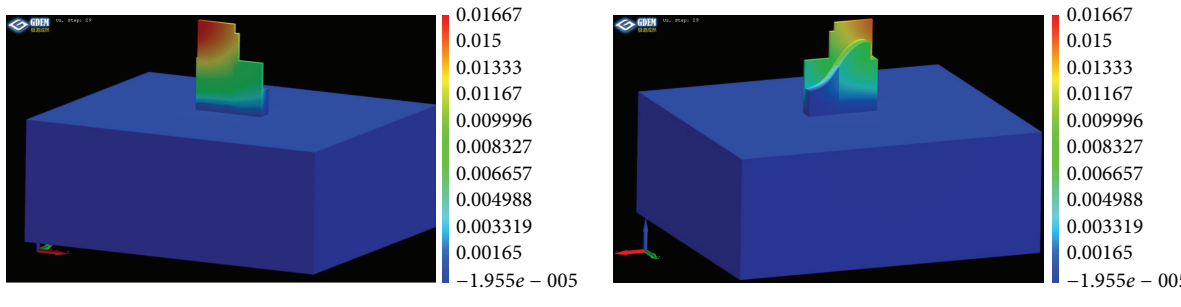
(7) call the function *VeksFixU* to set the displacement constraints;



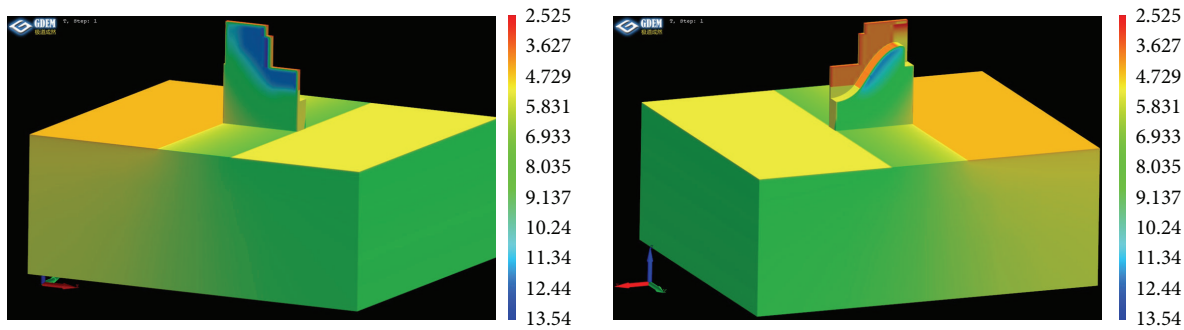
(a) X-displacement



(b) Y-displacement



(c) Z-displacement



(d) Temperature

FIGURE 8: The simulation results using the GPU parallel solver.

(8) call the function *VeksSetDemQ* to transfer the thermal loads vector into the GPU solver;

(9) call the function *VeksSolve* to solve the equations;

(10) call the functions *VeksGetU* and *VeksGetT* to get the displacements and temperatures, respectively.

4. Numerical Examples

4.1. Thermal Field Verification

4.1.1. Cylinder Water Cooling Pipe. Take a piece of circular area, whose radius $R = 1.5$ m (see Figure 3). The fine and equivalent models of FEA are used for comparison and verification. Comparative simulations between a plastic and a

Table 5: Test platform.

Sun Fire 6800					
Operation system	Memory	External memory	Processor	Network	Compiler
Solaris9	16 GB	640 G	Ultra SPARC 1.2 G	Gigaplane	Sun studio 9

Table 6: 380000-order general coefficient matrix test results.

Sections	1 (390 time steps)	2 (469 time steps)	4 (483 time steps)	8 (513 time steps)
1 thread	174.33292s	212.14563s	216.148044s	223.7508s
2 threads		114.81871s		126.267044s
4 threads			61.3276s	66.57s
8 threads				36.64s

Table 7: Test platform.

Shuguang Cluster (single node)					
Operation system	Memory	External memory	Processor	Network	Compiler
RedHat.E.L4.0	8 GB	143 G	XEON3.0 G	Gigaplane	Intel Fortran

Table 8: Test schemes.

Solving methods	Preconditions	
	Global	Local
Method 1 (CGS)	Domain decomposition	ILUT
Method 2 (BiCGSTAB)	Domain decomposition	ILUT
Method 3 (BiCGSTAB)	SYMGS	—

Table 9: 930000-order slightly ill-conditioned matrix test results.

Methods	Time (s)	Iterations	Convergence	Number of threads
Method 1	447.3	297	$1.0E-12$	1
Method 1	220.8	967	$1.0E-12$	4
Method 1	170.2	1103	$1.0E-12$	8
Method 2	150.4	480	$1.0E-12$	8
Method 3	165.4	551	$1.0E-12$	8
Serial codes	656.12	850	$1.0E-12$	1

metal pipe are performed. The parameters for the simulations are presented in Table 3.

The comparisons between the results of the fine and equivalent models are illustrated in Figure 4. The temperature in the fine model is a real value, while the temperature in the equivalent model is and can only be a mean value of the whole pipe domain.

The figure shows that the temperature of the metal cooling pipe is lower than that of a plastic one, and the temperature difference between them reaches 1.5°C on day 15. Thus, the metal pipe takes priority over the plastic one in concrete cooling. Meanwhile, the numerical result shows a great agreement with the analytical solution. The temperatures obtained by these two models are nearly the same, which indicates that the equivalent model can ensure its accuracy when simulating a real engineering problem.

4.1.2. Simulation of a Dam. A hyperbolic arch dam in Southwest China is 285.5 m high. The annual average air temperature in the area is 19.7°C; the highest monthly average air temperature is 27.1°C; the lowest monthly average air temperature is 10.6°C. The annual average ground temperature is 21.4°C.

A simulation for the dam is performed to grasp the real temperature status and thus to develop a measure for preventing cracking. The simulation takes into account the whole process of excavation and pouring of dams and the varieties of influencing factors for dam temperature. The model with its boundary conditions is illustrated in Figure 2.

The simulation results are shown in Figure 5. The temperature of Elevation 357 m is presented in Figure 5(a) while the temperature of Elevation 408 m is presented in Figure 5(b). The comparisons between the numerical results and the measured ones are made, and we know from the figure that these results show a great agreement with each other. Additionally, the results show the real temperature status. These all indicate that the models presented are reasonable and accurate.

4.2. Parallel Computing Examples

4.2.1. Parallel PCG Solver Test on OpenMP

(1) Serial Codes Test. A serial codes test is performed on a ThinkCentre M Q45t computer. Its parameters are shown in Table 4. This computer can solve a problem with more than 5 million degrees of freedom.

The result of the test is shown in Figure 6. The PCG solver runs on a single PC very efficiently with little memory, and

Table 10: Test platform.

Inspur NF5860M2					
Operation system	Memory	External memory	Processor	Network	Compiler
RedHat.E.L6.0	128 GB	2T	XEON3.0 G	Shared memory	Intel Fortran

Table 11: 930000-order slightly ill-conditioned matrix test results.

Methods	Speedup	Time (s)								Number of threads
		Total	Matrix structural analysis	Matrix rearrangement	Symbolic factorization	ILU decomposition	Direct solving	Memory allocation	Other	
Parallel SE	/	**655.03**	0.55	10.23	5.37	630.26	4.39	0.11	4.12	1
	1.92	**340.46**	0.47	10.29	3.57	319.21	2.66	0.13	4.13	2
	3.65	**179.35**	0.47	10.24	2.88	159.99	1.54	0.11	4.12	4
	6.28	**104.29**	0.47	10.30	2.95	85.19	1.14	0.12	4.12	8
	9.31	**70.37**	0.48	10.28	3.77	50.57	1.00	0.13	4.14	16
Serial codes	/	**656.12**	/	/	/	/	/	/	/	1

it can solve big problems. Figure 6 shows the different times solving problems with different DOFs by using the serial SSOR-PCG solver. The result indicates that the sovler takes only 100 seconds when solving a problem with 2 million DOFs and takes 500 seconds when solving a problem with 5 million DOFs. Obviously, it is much faster than many solvers.

(2) Parallel Codes Test. A parallel codes test of the PCG solver runs on a Sun Fire 6800 computer. Table 5 shows the test platform, and Table 6 gives the test results. We can easily know from the results that the speedups increase significantly with the number of threads. The acceleration of speedups drops a little, but the speedups themselves are almost over 80%.

4.2.2. Parallel PKS Solver Test on MPI. Test of the parallel PKS solver is performed on platform shown in Table 7. Methods are shown in Table 8, and the test results are presented in Table 9.

We can easily obtain that the speedups are significant and the parallel solver is quite efficient. This parallel solver can run either on a cross-node cluster or on a shared memory system such as the multicore CPU.

4.2.3. Parallel SE Solver Test on OpenMP. The test platform for parallel SE solver is presented in Table 10, and the results are shown in Table 11. For a 930000-order slightly ill-conditioned matrix, the solving time is quite impressive, and it is only 70 seconds when the solver runs on 16 threads. Since a sparse direct method is employed, the time is mainly spent on incomplete LU decomposition. Only about 20G memory is consumed, which is much less than the Gaussian elimination, but more than iterative methods whose memeroy consumption can be only 3-4 G. We can get from the speedups that the parallel sparse equation solver is very efficient and scalable, and it is well suitable for equations of large linear system.

Table 12: Speedup of GPU580 versus different CPU machines and methods.

CPU machine	32-core CPU	Single-core CPU	Single-core i5 CPU
CPU method	Parallel direct sparse	Serial direct sparse	Dynamic relaxation
GPU speedup	4.2	40.3	445

4.2.4. Parallel GPU Solver Test. An engineering test is performed, using the model in Figure 7.

The boundary conditions are similar to the ones shown in Figure 2. The results are shown in Figure 8.

In this simulation, we test the speedups, which will be shown in Table 12.

5. Conclusions

First, numerical models adopted by the software are presented including

(i) hydration models,

(ii) water cooling models,

(iii) modulus models,

(iv) creep model,

(v) autogenous deformation models taking into account the properties of MgO concrete.

A thermal example for verifying the thermal models is presented. A good agreement is achieved between the numerical results and the analytical ones. A finite element simulation for the whole process of excavation and pouring of dams using these models is made, and the numerical results show a good agreement with the measured ones.

Then, several parallel solvers are introduced with their parallel strategies, consisting of

(1) the preconditioned (i.e., SSOR) conjugate gradient solver,

(2) the preconditioned (e.g., CG, CGS, BiCGSTAB, etc.) Krylov subspace solver,

(3) the sparse equation solver.

The parallelization procedures for the PCG solver can be summarized as follows:

(i) first, the matrix and vector operations are parallelized, including $\mathbf{AP}$ and $\mathbf{A}^T\mathbf{P}$, where the vector $\mathbf{P}$ is generated by specific algorithm;

(ii) then, the inner product operation is parallelized;

(iii) furthermore, the vectors are updated;

(iv) finally, the preconditions are calculated (if necessary).

The parallelization procedures for the PKS solver can be summarized as follows:

(i) the global FEA equations are assembled;

(ii) the "divide and conquer" strategy is employed to divide the FEA domain into sub-domains;

(iii) local equations are formed based on sub-domains;

(iv) iterative techniques, for example, CG, CGS, BiCGSTAB, GMRES, TFQMR, and so forth, are paralleled and used to solve all the local equations.

The sparse equation solver is parallelized in CSR matrix format, which supports several types of matrices including real/imaginary matrices and symmetric/asymmetric matrices. The parallelization procedures for the sparse equation solver are similar to those for the PCG solver.

Last, a comparative study on these parallel solvers is performed. The results show that

(i) the speedups are quite significant;

(ii) the serial and parallel solvers are both very efficient;

(iii) the serial and parallel solvers can both deal with very large problems with more than 5 million degrees of freedom;

(iv) the parallelization makes SAPTIS more powerful and adaptable.

A GPU-based parallel solver has been developed, and the GPU parallelization has made SAPTIS much more efficient.

Conflict of Interests

The authors declare that they do not have any conflict of interests with the content of the paper.

Acknowledgments

The authors would like to acknowledge the financial support of the National Natural Science Foundation of China (51209235), the National Basic Research Program (973 Programs: Grant nos. 2010CB731500, 2013CB036406, and 2013CB035904), the National "Twelfth Five-Year" Plan for Science and Technology Support (2013BAB06B02), the Governmental Public Industry Research Special Funds for Projects of MWR (201201050), and the IWHR Special Research of China (Volume 1118, Volume 1208, Volume 1169, Volume 1268, Volume 1309, Volume 1353, and Volume 1361).

References

[1] A. Saetta, R. Scotta, and R. Vitaliani, "Stress analysis of concrete structures subjected to variable thermal loads," *Journal of Structural Engineering*, vol. 121, no. 3, pp. 446–457, 1995.

[2] O. Omidi and V. Lotfi, "Numerical analysis of cyclically loaded concrete under large tensile strains by the plastic-damage model," *Scientia Iranica A*, vol. 17, no. 3, pp. 194–208, 2010.

[3] M. Nazem, I. Rahmani, and M. Rezaee-Pajand, "Nonlinear fe analysis of reinforced concrete structures using a tresca-type yield surface," *Scientia Iranica A*, vol. 16, no. 6, pp. 512–519, 2009.

[4] Z. P. Bazant, E. C. Rossow, and G. Horrigmoe, "Finite element program for creep analysis of concrete structures," in *Proceedings of the 6th International Conference on Structural Mechanics in Reactor Technology*, Paris, France, 1981.

[5] Z. P. Bazant and R. L'Hermite, *Mathematical Modeling of Creep and Shrinkage of Concrete*, Wiley, New York, NY, USA, 1988.

[6] M. Savoia, D. Ferretti, and C. Mazzotti, "Creep behavior of RC tensile elements retrofitted by FRP plates," in *Proceedings of the 10th IEEE International Conference on Cognitive Informatics (ICCI' 02)*, vol. 2, San Francisco, USA, 2002.

[7] G. C. Fanourakis and Y. Ballim, "Predicting creep deformation of concrete: a comparison of results from different investigations," in *Proceedings of the 11th FIG Symposium on Deformation Measurements*, Santorini, Greece, May 2003.

[8] L. F. Nielsen, "Composite creep analysis of concrete: a rational, incremental stress-strain approach," Tech. Rep., Technical University of Denmark, Copenhagen, Denmark, 2007.

[9] A. Mari and A. Scordelis, "Nonlinear geometric, material and time dependent analysis of three dimensional reinforced concrete and prestressed frames," UC SESM Report 84/12, EERC, Berkeley, Calif, USA, 1984.

[10] R. de Borst and A. H. Van Der Boogaard, "Finite-element modeling of deformation and cracking in early-age concrete," *Journal of Engineering Mechanics*, vol. 120, no. 12, pp. 2519–2534, 1994.

[11] E. Spacone, F. C. Filippou, and F. F. Taucer, "Fibre beam-column model for non-linear analysis of R/C frames: part I. Formulation," *Earthquake Engineering and Structural Dynamics*, vol. 25, no. 7, pp. 711–725, 1996.

[12] H. Ito, I. Maruyama, M. Tanimura, and R. Sato, "Early age deformation and resultant induced stress in expansive high strength concrete," *Journal of Advanced Concrete Technology*, vol. 2, no. 2, pp. 155–174, 2004.

[13] I. Lenhardt and T. Rottner, "Krylov subspace methods for structural finite element analysis," *Parallel Computing*, vol. 25, no. 7, pp. 861–875, 1999.

[14] J. L. Volakis, D. B. Davidson, C. Guiffaut, and K. Mahdjoubi, "A parallel FDTD algorithm using the MPI library," *IEEE Antennas and Propagation Magazine*, vol. 43, no. 2, pp. 94–103, 2001.

[15] S. F. McGinn and R. E. Shaw, "Parallel Gaussian elimination using OpenMP and MPI," in *Proceedings of the 16th Annual International Symposium on High Performance Computing Systems and Applications*, pp. 169–173, IEEE, 2002.

[16] R. Rabenseifner, G. Hager, and G. Jost, "Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes," in *17th Euromicro International Conference on Parallel, Distributed*

and Network-Based Processing, PDP 2009, pp. 427–436, deu, February 2009.

[17] B. F. Zhu, *Thermal Stress and Temperature Control of Mass Concrete (2nd Edition)*, China Electric Power Press, Beijing, China, 2012.

[18] B. F. Zhu, *New Developments on Theories and Techniques of Concrete Dams*, China Water Power Press, Beijing, China, 2009.

[19] B. F. Zhu, *Anthology of Academician Zhu Bofang*, China Electric Power Press, Beijing, China, 1997.

[20] O. C. Zienkiewicz, R. L. Taylor, and J. Z. Zhu, *The Finite Element Method: Its Basis and Fundamentals (6th Edition)*, Butterworth-Heinemann, Oxford, UK, 2005.

[21] X. C. Wang, *The Finite Element Method*, Tsinghua University Press, Beijing, China, 2003.

[22] W. Cheng, "A secondary development and application of ANSYS: simulation of concrete creep," *Guizhou Hydroelectricity*, vol. 23, no. 3, pp. 68–70, 2009 (Chinese).

[23] B. H. V. Topping, *Parallel and Distributed Processing for Computational Mechanics: System and Tools*, Saxe-Coburg, Edinburgh, UK, 1999.

[24] A. K. Noor, "New computing systems and future high-performance computing environment and their impact on structural analysis and design," *Computers and Structures*, vol. 64, no. 1–4, pp. 1–30, 1997.

[25] L. N. B. Gummadi and A. N. Palazotto, "Nonlinear finite element analysis of beams and arches using parallel processors," *Computers and Structures*, vol. 63, no. 3, pp. 413–428, 1997.

[26] R. Menon, L. Dagum, D. Kohr, D. Maydan, and J. McDonald, *Parallel Programming in OpenMP*, Morgan Kaufmann, Burlington, Mass, USA, 2000.

[27] Y. Saad, "Krylov subspace methods on supercomputers," *SIAM Journal on Scientific and Statistical Computing*, vol. 10, no. 6, pp. 1200–1232, 1989.

[28] H. A. Van Der Vorst, "Krylov Subspace Iteration," *Computing in Science and Engineering*, vol. 2, no. 1, pp. 32–37, 2000.

[29] M. L. Romero, P. F. Miguel, and J. J. Cano, "A parallel procedure for nonlinear analysis of reinforced concrete three-dimensional frames," *Computers and Structures*, vol. 80, no. 16-17, pp. 1337–1350, 2002.