Finite element analysis of levee stability for flood early warning systems

Melnikova, N.B.

Publication date
2014

Chapter 3 Implementation, integration into decision support system and performance assessment[2]

3.1 Introduction

Equations of porous flow and deformations in earthen dikes outlined in Section 2.4 were numerically solved by the finite element method. In this chapter we briefly mention finite element method fundamentals. Next, details of implementation of the Virtual Dike module in Comsol software packaged are presented. Description of integration of the Virtual Dike module into the UrbanFlood decision support system is then followed by the parallel efficiency assessment. Performance benchmarks were carried on one computational node of a computer cloud Sara used for Virtual Dike hosting.

Finite element method is widely used for numerical solution of partial differential equations (PDE). It employs space discretization of the original unknown solution function $U(x,y,z)$ by expressing it as a sum of pre-defined basis functions $f_j(x,y,z)$ multiplied by unknown coefficients U_j; the number of terms in the sum equals to the number of nodes in finite element mesh: j=N; and coefficients U_j are nodal values of $U(x,y,z)$ (Potts and Zdravkovic, 1999):

$$U(x,y,z) \approx \sum_{j=1}^{N} f_j(x,y,z) \cdot U_j$$

The original PDE is then reduced to a system of algebraic equations, e.g. by applying the least squares method to the residual integral on a space domain. For physically non-linear problems like plastic flow, the resulting system of algebraic equations is non-linear, due to dependence of stiffness characteristics on the stress level. Nonlinearity introduced by the constitutive behaviour requires incremental solution procedure with linear algebraic system (LAS) at each increment:

$$[K_G]^i \cdot [\Delta U]^i = [\Delta F]^i , \qquad (3.1)$$

where i is number of increment, $[K_G]^i$ is incremental global stiffness matrix, $[\Delta U]^i$ is vector of nodal solution increments (of length N), $[\Delta F]^i$ is vector of nodal load increments.

---

[2] Parts of this chapter have been published in (Melnikova, N.B., Krzhizhanovskaya, V.V., Shirshov, G.S., Shabrov, N.N. (2011). Virtual Dike and Flood Simulator: Parallel distributed computing for flood early warning systems. Proc. of the International Conference on Parallel Computational Technologies (PAVT-2011). Publ. Centre of the South Ural State University, Chelyabinsk, pp. 365-373. http://omega.sp.susu.ac.ru/books/conference/PaVT2011/short/139.pdf)

Equation (3.1) was solved by the modified Newton-Raphson (MNR) iterational procedure (see a detailed specification of MNR procedure in (Potts and Zdravkovic, 1999)).

The finite element method was introduced to slope stability analysis in (Whitman and Bailey, 1967). Zienkiewicz (Zienkiewicz et al., 1975) has proposed to treat divergence of numerical solution as an indicator of actual slope failure, so that simulated failure occurs naturally as the shear strength of the soil becomes insufficient to resist the shear stresses. When perfect plastic yielding occurs in large zones, the stiffness matrix (while solving equations numerically) approaches a singular matrix, and iterations of the finite element solver fail to converge. Physically, it indicates the onset of global plastic yielding, so that present configuration of the dike can not remain stable under the specified load.

3.2 Virtual Dike Implementation

In the *Virtual Dike* module, partial differential equations are solved in the finite element package COMSOL 3.5a. Finite element mesh composed of triangle elements with second order of spatial approximation was used in simulations. Time integration was performed by an implicit backward second order method. The modified Newton-Raphson iterative scheme was used for solving nonlinear algebraic equations at each time integration step. During MNR iterations, systems of linear algebraic equations were solved by direct PARDISO solver from the BLAS library.

Automatic sensor input was implemented via MATLAB script, which reads input from a specified file, starts COMSOL simulation and stores virtual sensor output (Figure 3-1).

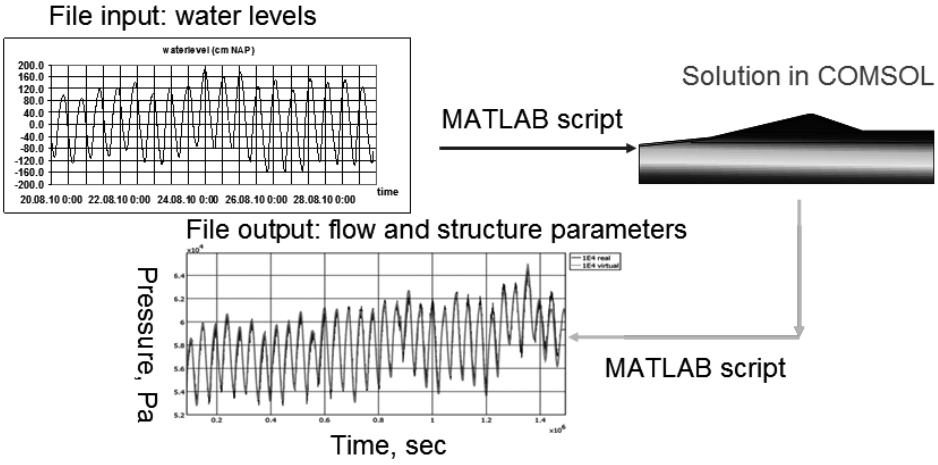


Figure 3-1. Virtual Dike –sensor input/output handling

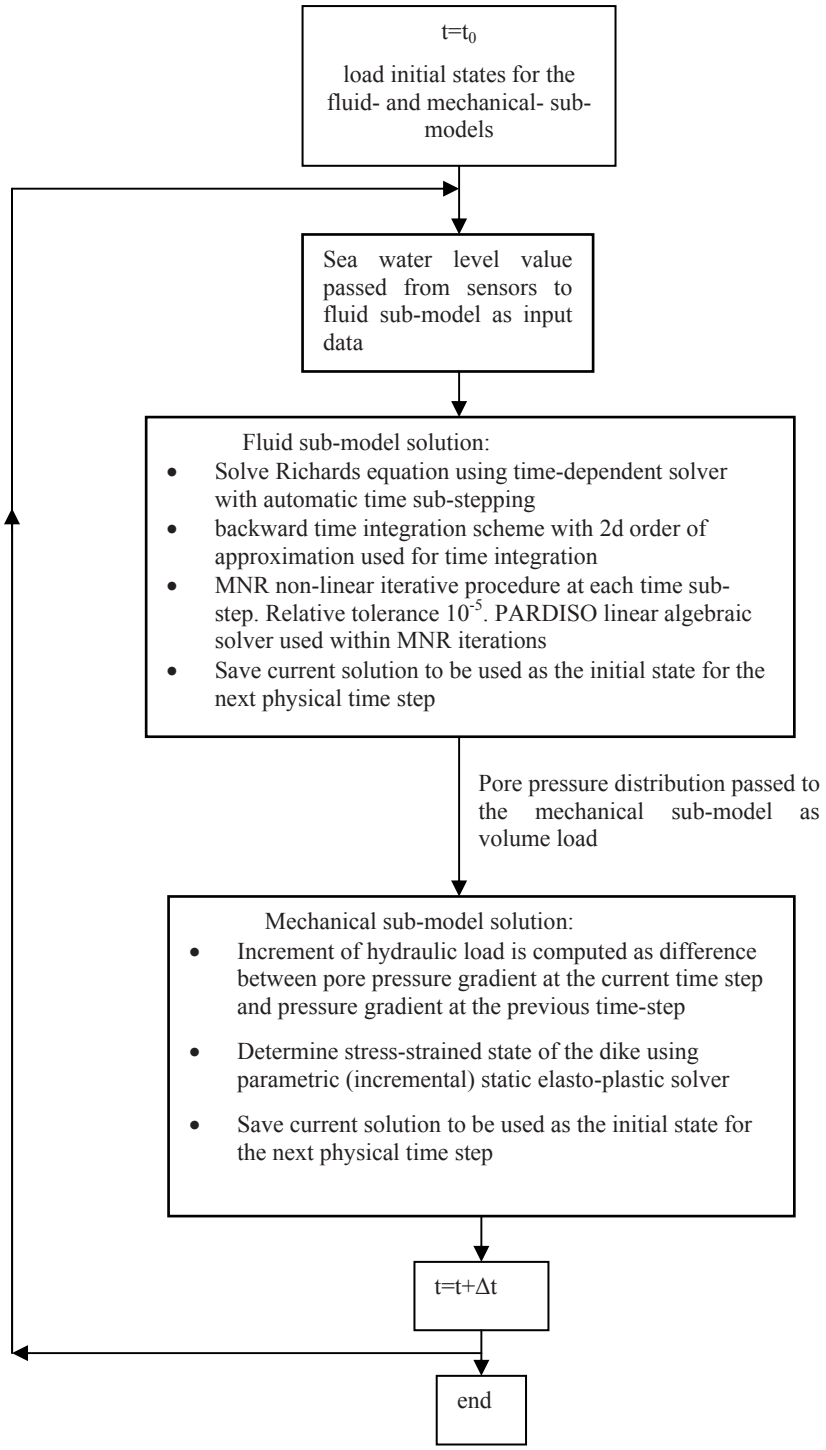


Figure 3-2. Computational workflow in the *Virtual Dike* module

Functionality of the Comsol package supports multi-physics coupling – equations describing the time-dependent fluid sub-model and the quasi-static mechanical sub-model can be united in one system of equations, which will be integrated by a time-dependent solver. However, convergence of the time-dependent solver was poor for the highly-nonlinear mechanical sub-model: integration in time-domain required very small time-steps and finally stopped due to divergence of MNR iterations. Typically plastic deformation problems in Comsol require a steady-state parametric solver for their solution. That's why a two-step solution scheme was used, where the porous flow sub-model was solved by a time-dependent solver; then pore pressure loads were transferred to the mechanical sub-model which in turn was solved by parametric solver. The computational workflow is described in Figure 3-2: the workflow with calls to Comsol routines is processed in a Matlab script; data exchange between the sub-models occurs in the computer RAM memory within the Matlab run-time environment. The loop in Figure 3-2 corresponds to time-stepping; stagger iterations were not used for the solution process, as the problem was considered as one-way coupled.

3.3 Integration into the UrbanFlood early warning system

Different components of the UrbanFlood EWS were running on several servers located in 4 countries: Russia, the Netherlands, Poland and the United Kingdom. The EWS has been designed in accordance with the cloud computer technology. Computer clouds use "virtualization software", which allows multiple operating systems (="virtual machines") to share the same (rented) hardware. The operating systems themselves run various applications (such as *Virtual Dike*, *Flood Simulator, AI* component and others). Furthermore, the virtualisation software allows halting virtual machines (and contained running programs), move these to other computers and continue operations there (Meijer et al., 2012).

A user interface implemented on the Microsoft *Surface* multi-touch table provided access to all EWS components and allowed interactive simulation steering (Figure 3-3). Tests of the EWS performance showed a real-time response of system components to user manipulations.

Figure 3-3. Accessing EWS results via a multi-touch table

All simulation modules and visualization components of the *UrbanFlood* early warning system (Figure 1-1) have been integrated into the global communication platform - Common Information Space (CIS). CIS is an architectural framework providing services to address problems common to all early earning systems as complex software systems: integration of legacy scientific applications, workflow orchestration, allocation of computational resources and robust operation. The key components of CIS are (Balis et al., 2011):

•Integration platform (PlatIn): CIS core technologies for component integration, data exchange and workflow orchestration.

•Metadata registry (UFoReg): a generic service for hosting and querying metadata.

•Dynamic resource allocation service (DyReAlla): a service for dynamic allocation of resources to running Early Warning Systems.

A detailed description of CIS architecture can be found in (Balis et al., 2011) and at http://dice.cyfronet.pl/products/cis.

The *Virtual Dike* module was integrated into the CIS and deployed on a cloud computing resources of the SURFsara BiG Grid High Performance Computing and e-Science Support Centre in Amsterdam, the Netherlands https://www.surfsara.nl/. The cloud is hosted on a 128-core cluster and uses OpenNebula open source cloud computing management toolkit with KVM Virtual machine software. The web-based interface for virtual machines management provided by SARA support to users is shown in Figure 3-4.

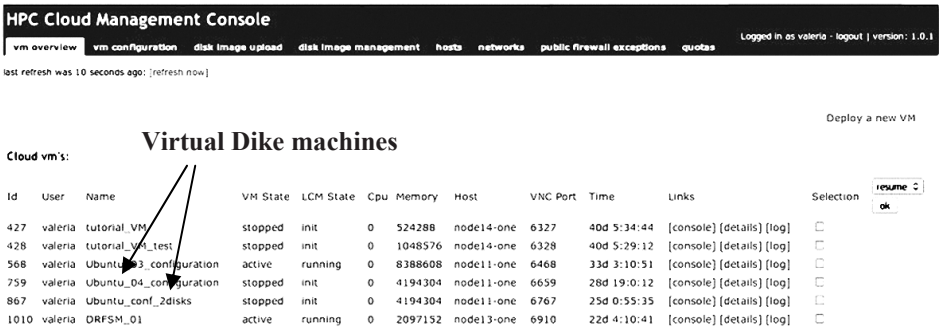


Figure 3-4. Sara cloud management console with Virtual Dike machine instances

Virtual Dike simulations were run under Ubuntu Linux in a shared memory parallel mode. In the EWS workflow, the module ran in real time, receiving water level sensor signal as input data and producing "virtual sensor" signals (flow and structure parameters).

To start a new simulation, CIS launched a new instance of Linux Ubuntu virtual machine with the *Virtual Dike* model and wrote sensor input (sea/river level value) to the specified directory in real-time (the directory was updated every 5 minutes). The output from the *Virtual Dike* was stored in a specified directory on a hard drive, from where it was accessed by the CIS, compared to sensor measurements and visualized at the user front-end.

Monitoring input and output directories for new files and deleting the old files was performed by a Ruby script wrapping the *Virtual Dike* (Krzhizhanovskaya et al., 2012).

3.4 Performance benchmarking

Computational performance benchmarks have been carried in order to estimate the elapsed times for computationally heavy models with very fine mesh. In fact, we have tested efficiency of Comsol software package in application to the uncoupled porous flow analysis.

Besides testing parallel efficiency of the package, we have determined an optimal set of the package settings, allowing further reduction of elapse times. The tested settings include: direct solver algorithm (PARDISO versus UMFPACK), processor usage modes (owner, turnaround and throughput) and BLAS library implementations (MKL, ACML, ATLAS).

The goal of benchmarking was to find the best combinations of computational settings and to estimate feasibility and efficiency of parallel simulations. Parallel performance results have been previously published in (Melnikova et al., 2011).

For a relatively coarse mesh with 60 000 degrees of freedom (DOF) and maximum element size of 1 m, 2D porous flow simulations typically require up to 2 GB RAM in serial mode. Coupled fluid-structure problem requires up to 8 GB of memory in serial mode. The amount of occupied system memory increases when using parallel mode.

Comsol supports two modes of parallel operation: distributed mode and shared memory mode (Comsol Installation and Operations Guide, 2009). Distributed mode employs MPI library for process communications; it can be used on several nodes of a Linux or Windows cluster. Shared memory mode uses common address space for inter-process communication. It can only be used on a single multi-core or multi-processor computational node. In Virtual Dike simulations, one simulation used one computational node of SARA HPC Cloud, with dual quad-core CPU. Shared-memory parallelism has been tested for the uncoupled porous flow model, in the default processor usage mode.

3.4.1 Parallel performance benchmarks

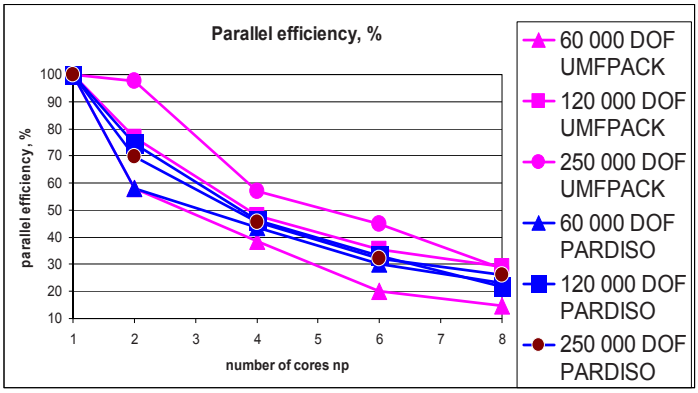
Parallel performance benchmarks were performed using two alternative direct linear sparse solvers: UMFPACK (Unsymmetric MultiFrontal PACKage) and PARDISO (PArallel DIrect SOlver). Results of the tests are presented in Figure 3-5a-b.

PARDISO sparse linear solver is recommended by Comsol as providing the best parallel efficiency in comparison to the other solvers. The results confirm this only for "small" problem size with number of DOF=60 000. Formally, for larger problem sizes, UMFPACK scalability was better (see Figure 3-5a for a plot of parallel efficiency computed in percentage of a real speedup to an ideal speedup (=number of cores np)).

The parallel speedup of each solver first improves as the number of DOF grows (as computational work portion grows higher relative to the portion of information exchange). After that, we get speedup saturation or even decrease for number of cores np equal to 8. The reason of low scalability and cases of decrease in speedup for np=8 may lie [a]: in high

process synchronization costs; [b]: in a relatively high portion of sequential operations included in the algorithm; [c] in a bad-balanced distribution of load among the cores for 8-core case (one of the cores was obviously occupied with system tasks, too). The solver writes down solution on the hard disk with specified periodicity. As the number of DOF grows, this sequential output work increases. Naturally this I/O option could be withdrawn from the simulation benchmarks but it was important to estimate a speedup for real working conditions. The output from the solver gets to CIS and other computational modules via the hard disk files.

(a)



(b)

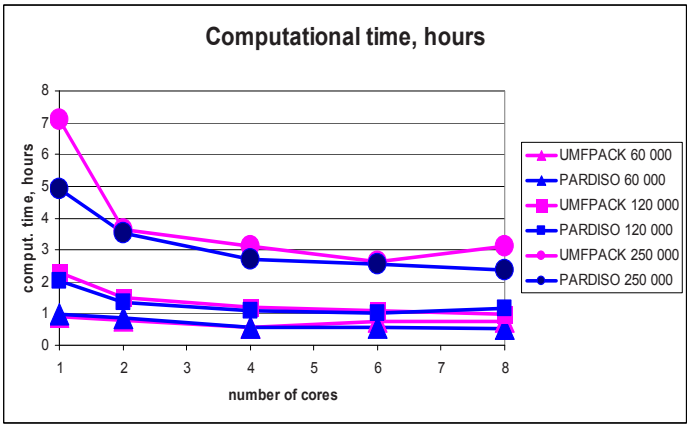


Figure 3-5. Results for 1, 2, 4, 6, 8 cores: (a) parallel efficiency; (b) computational time

Computational time varies from 30 min to 7 hours (Figure 3-5b), depending on the problem size, number of cores and chosen solver. For problems of 250 000 unknowns, PARDISO was almost 50% faster. UMFPACK speed itself was significantly lower for large problem sizes (Figure 3-5b). The slow speed of UNFPACK is explained by the fact that we have a symmetric system matrix in porous flow problems while UMFPACK is designed for a more general case of unsymmetric linear systems, it does not take an advantage of symmetry, which leads to a higher number of operations and slower computational speed. This also explains most probable reason why scalability of UMFPACK for 120 000 and 250 000 unknowns was better – communications were better hidden behind computations, due to a bigger portion of computations, compared to

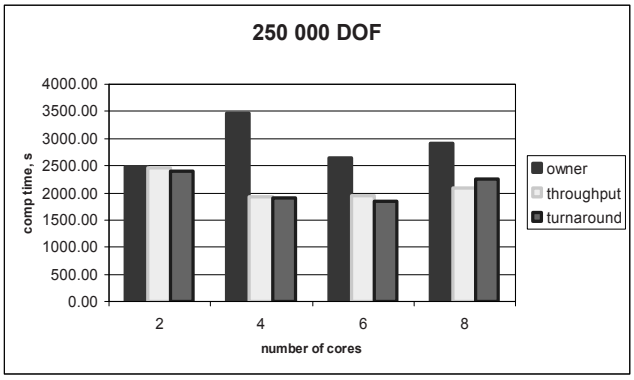
PARDISO. Nevertheless, absolute elapse times for PARDISO were always lower or equal to UMFPACK times, so the general conclusion was to use the PARDISO solver for the *Virtual Dike* module.

3.4.2 Processor usage modes and BLAS libraries efficiency benchmarks

Benchmark results presented below have also been obtained for porous flow problem, using one computational node of SARA. But here Comsol simulation has been coupled with MATLAB for automatic sensor input of water levels. This coupling reduced the performance and parallel scalability of the *Virtual Dike* module.

Comsol supports three processor usage modes: owner (recommended for processors, not occupied with other tasks), turnaround and throughput (both are recommended for multi-tasking regimes). Benchmark results showed that the owner mode is the worst (see Figure 3-6a). Turnaround mode was the best for all cases except the case of using 8 cores, which is not recommended as it increases computational time, comparing to the cases of using 4 and 6 cores.

(a)



(b)

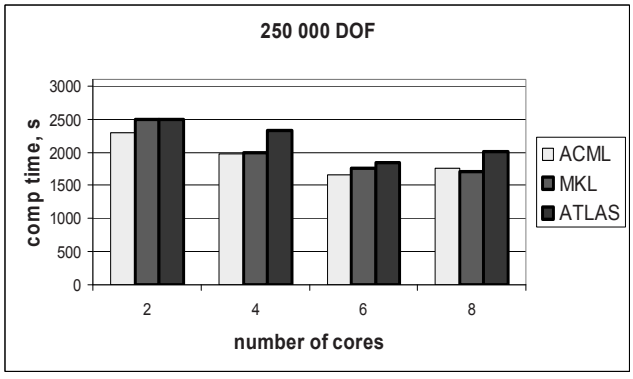


Figure 3-6. Computational time: (a) comparison of processor usage modes; (b) comparison of BLAS versions efficiency

Results of BLAS libraries efficiency are presented in Figure 3-6b. ACML, Intel and MKL BLAS libraries have been tested, for 2, 4, 6 and 8 computational cores. ACML BLAS library was optimal for all cases, except the case of 8 cores. Using 6 cores with ACML library provided the lowest computational time.

3.5 Conclusions

The Virtual Dike module has become the first implementation of a finite element analysis tool for dike stability assessment working with real-time sensor input. The module also produces real-time output from "virtual sensors" and stores it in the common information space of the early warning system. The module was deployed on an 8-core node of the supercomputer cloud SARA (Amsterdam).

According to the computational efficiency benchmarks, two- and four-core parallelism was good enough, with parallel efficiency higher than 50%. For more than 4 cores, synchronisation costs were too high. PARDISO linear solver, "turnaround" CPU usage mode and ACML BLAS library were chosen as optimal settings to get lowest computational times.