Applied Numerical Modeling in Geomechanics – 2020 – Billaux, Hazzard, Nelson & Schöpfer (eds.) Paper: 13-05

Implementation of advanced numerical solvers in *FLAC3D* thermal and fluid implicit formulation

Andrey V. Pyatigorets & David B. Russell
Itasca Consulting Group, Inc., Minneapolis, MN, USA

1 INTRODUCTION

A large number of engineering problems deal with solving heat transfer and/or fluid-flow problems. Very often these problems are coupled with mechanical analysis of solids to capture the physics of the underlying processes. Each of these processes has a characteristic time which describes the typical time needed to transport energy (or information) over the characteristic length (Itasca 2019 a,b,c). For thermal and fluid diffusion processes, the characteristic time is often several magnitudes larger than the time for mechanical processes. In practical applications, however, the timestep in numerical solution of thermal and fluid problems tends to be very small. This is related to the fact that it is limited by the stability of numerical methods used for solving governing differential equations, which are discussed below.

FLAC3D thermal and fluid logic adopt energy-balance and fluid mass-balance equations, respectively, formulated at each node of model grid. The nodal form of these equations is (Itasca 2019 b,c):

for thermal problems: $$\frac{dT_n}{dt} = -\frac{1}{\sum m_{th}^n} Q_{th}^n \quad (1)$$

for fluid-flow problems: $$\frac{dP_n}{dt} = -\frac{1}{\sum m_{fl}^n} Q_{fl}^n \quad (2)$$

where T_n (and P_n) is temperature (pressure) at node n, t is time, m_{th}^n (and m_{fl}^n) is equivalent nodal thermal (fluid) mass and summation is taken over all zones contributing to node n, Q_{th}^n (and Q_{fl}^n) is equivalent out-of-balance heat (discharge).

Equations 1 and 2 are very similar and therefore thermal and fluid logic share the same solution techniques, which include explicit and implicit numerical schemes[1]. Based on this, only discussion pertinent to thermal analysis is provided below. However, all the results presented in this paper, including new solution methods, can be directly adopted to fluid-flow analysis.

In the explicit formulation, the time derivative in Equations 1 and 2 is expressed using forward finite difference and out-of-balance quantities are evaluated at previous time t. Numerical stability of this scheme imposes limitations on the maximum timestep which cannot exceed the characteristic time of the smallest zone in the model. This timestep often tends to be extremely small ($10^{-5} \div 10^{-1}$ seconds) making use of the explicit scheme impractical in many cases.

Implicit formulation partially resolves this restriction by employing the numerically stable Crack-Nicolson method (Thomas 1995). In this formulation, the time derivatives in Equations 1 and 2 are expressed using central finite difference and out-of-balance quantities are evaluated by taking the average between previous

[1] In the fluid module, the implicit scheme only applies to fully saturated fluid-flow simulation.

and current time (Itasca 2019 b,c). By using the Crank-Nicolson method, differential Equation 1 (and similarly Equation 2) is transformed into a system of linear equations:

$$A_{nj}\Delta T_{j\langle t\rangle} = b_{n\langle t\rangle} \ , \tag{3}$$

$$A_{nj} = \delta_{nj} + \frac{\gamma^n}{2} C_{nj}, \ \gamma^n = \frac{\Delta t}{\sum m_{th}^n}$$

Here, C_{nj} is the matrix of thermal coefficients (related to heat capacities) for nodes *n* and *j*, δ_{nj} is Kronecker delta, $\Delta T_{j\langle t\rangle}$ is temperature change from time *t* to time $t + \Delta t$ at node *j*, and $b_{n\langle t\rangle}$ is the RHS vector containing contributions from out-of-balance heat sources.

In the current numerical implementation of the implicit approach, System 3 is formulated and solved locally for every node of the model using the Jacobi iterative method (Itasca 2019 b,c). Although, the Crank-Nicolson method is stable for any positive Δt, the convergence of the Jacobi method is guaranteed only if matrix A_{nj} is strictly diagonally dominant for each node. In practice, this condition sets a limit for maximum calculation timestep as the Jacobi method may diverge for large values of Δt. This often means that the implicit scheme allows the timestep to be only a few-to-hundreds times larger than the explicit scheme, which is not enough for an efficient calculation speed.

New developments presented in this paper overcome the limitations of the Jacobi method and allow using large timesteps only limited by the ability to capture the physics of the processes. The basic idea is to replace the node-based Jacobi iterative solver with an efficient implicit global solver which solves system of equations (3) for the whole model at once at each timestep.

2 FORMULATION OF THE GLOBAL PROBLEM

Contrary to current implementation of the Jacobi method, which operates on local node-zone level, the new approach is based on forming global matrix **M** for the whole model to account for contribution of all nodes at once. Equation 3 can be rewritten in a way more suitable for forming global matrix $\mathbf{M} = M_{nj}$:

$$M_{nj}\Delta T_{j\langle t\rangle} = \frac{2}{\gamma^n} b_{n\langle t\rangle}, \qquad M_{nj} = \frac{2}{\gamma^n}\delta_{nj} + C_{nj} \quad \text{or} \quad \mathbf{M} = \frac{2}{\gamma^n}\mathbf{I} + \mathbf{C} \tag{4}$$

where $1 \le n, j \le k$, *k* is the number of nodes (grid points) in the model, **I** is identity matrix, and $\mathbf{C} = C_{nj}$.

Global matrix **C** does not need to be calculated element-by-element in the new approach; the approach makes use of thermal coefficient matrices $\mathbb{C}$ previously calculated for each zone of the model and used in the Jacobi method (matrix $\mathbb{C}$ describes the contribution of thermal coefficients from node *j* to node *n* within a zone).

A simple example below illustrates the idea of the approach. Consider three triangular 2D zones connected at the common point in the middle (Fig. 1a). For each zone, change in heat ΔQ at each node can be expressed through node temperature change ΔT and local matrix of thermal coeffients $\mathbb{C}$ as:

$$\Delta Q^l = \ \mathbb{C}^l \Delta \boldsymbol{T}, \ l = a, b, c \tag{5}$$

$$\begin{bmatrix}\Delta Q_1^a\\ \Delta Q_4^a\\ \Delta Q_3^a\end{bmatrix} = \begin{bmatrix}c_{11}^a & c_{14}^a & c_{13}^a\\ & c_{44}^a & c_{43}^a\\ & & c_{33}^a\end{bmatrix}\begin{bmatrix}\Delta T_1\\ \Delta T_4\\ \Delta T_3\end{bmatrix}, \qquad \begin{bmatrix}\Delta Q_1^b\\ \Delta Q_2^b\\ \Delta Q_4^b\end{bmatrix} = \begin{bmatrix}c_{11}^b & c_{12}^b & c_{14}^b\\ & c_{22}^b & c_{24}^b\\ & & c_{44}^b\end{bmatrix}\begin{bmatrix}\Delta T_1\\ \Delta T_2\\ \Delta T_4\end{bmatrix},$$

$$\begin{bmatrix}\Delta Q_2^c\\ \Delta Q_3^c\\ \Delta Q_4^c\end{bmatrix} = \begin{bmatrix}c_{22}^c & c_{23}^c & c_{24}^c\\ & c_{33}^c & c_{34}^c\\ & & c_{44}^c\end{bmatrix}\begin{bmatrix}\Delta T_2\\ \Delta T_3\\ \Delta T_4\end{bmatrix}.$$

Note that each matrix $\mathbb{C}$ is symmetric. Summing up nodal heat contributions from each zone connected to a node, the following global system is obtained:

$$\Delta \boldsymbol{Q} = \mathbf{C}\Delta \boldsymbol{T}, \tag{6}$$

$$\begin{bmatrix}\Delta Q_1\\ \Delta Q_2\\ \Delta Q_3\\ \Delta Q_4\end{bmatrix} = \begin{bmatrix}\Delta Q_1^a + \Delta Q_1^b\\ \Delta Q_2^b + \Delta Q_2^c\\ \Delta Q_3^a + \Delta Q_3^c\\ \Delta Q_4^a + \Delta Q_4^b + \Delta Q_4^c\end{bmatrix} = \begin{bmatrix} c_{11}^a + c_{11}^b & c_{12}^b & c_{13}^a & c_{14}^a + c_{14}^b\\ & c_{22}^b + c_{22}^c & c_{23}^c & c_{24}^b + c_{24}^c\\ & & c_{33}^a + c_{33}^c & c_{43}^a + c_{34}^c\\ & & & c_{44}^a + c_{44}^b + c_{44}^c\end{bmatrix}\begin{bmatrix}\Delta T_1\\ \Delta T_2\\ \Delta T_3\\ \Delta T_4\end{bmatrix},$$

where incremental heat and temperature without superscripts denote total nodal quantity.

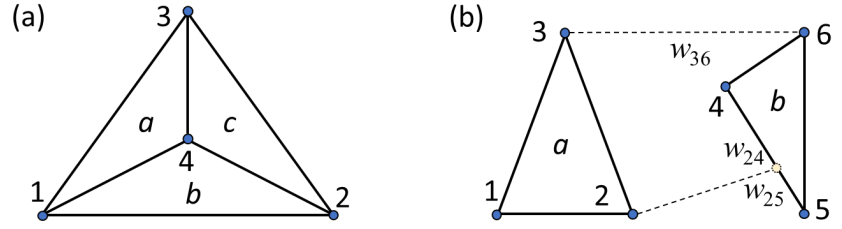

Figure 1. Simple geometries with no attach conditions (a) and with attaches (b).

It can be seen from Equations 4 - 6 that matrices **C** and **M** are symmetric. For this simple case, **M** is a dense matrix; however, in real applications, **M** is very sparse as each node typically accounts for the contributions from the connected zones only. A more complex case of when a node is slaved to another node or a face via attach conditions or interfaces is described next.

Consider the case presented in Figure 1b. Node 2 of zone *a* is attached (slaved) to face 4-5 of zone *b* with weights w_{24} and w_{25}. Node 3 is attached (slaved) to node 6 with weight w_{36}. Similarly to Equation 5, incremental nodal heat in each zone can be expressed as

$$\begin{bmatrix}\Delta Q_1^a\\ \Delta Q_2^a\\ \Delta Q_3^a\end{bmatrix} = \begin{bmatrix} c_{11}^a & c_{12}^a & c_{13}^a\\ & c_{22}^a & c_{23}^a\\ & & c_{33}^a\end{bmatrix}\begin{bmatrix}\Delta T_1\\ \Delta T_2\\ \Delta T_3\end{bmatrix}, \qquad \begin{bmatrix}\Delta Q_4^b\\ \Delta Q_5^b\\ \Delta Q_6^b\end{bmatrix} = \begin{bmatrix} c_{44}^b & c_{45}^b & c_{46}^b\\ & c_{55}^b & c_{56}^b\\ & & c_{66}^b\end{bmatrix}\begin{bmatrix}\Delta T_4\\ \Delta T_5\\ \Delta T_6\end{bmatrix} \tag{7}$$

Due to the presence of attach conditions, temperature at nodes 2 and 3 and incremental heat at nodes 4,5,6 are now inter-dependent. Using this fact and Equations 7, one can obtain:

$$\begin{aligned}
&\Delta T_2 = w_{24}\Delta T_4 + w_{25}\Delta T_5\\
&\Delta T_3 = w_{36}\Delta T_6\\
&\Delta Q_4 = \Delta Q_4^b + w_{24}\Delta Q_2^a = \Delta Q_4^b + w_{24}[c_{12}^a\Delta T_1 + c_{22}^a(w_{24}\Delta T_4 + w_{25}\Delta T_5) + c_{23}^a w_{36}\Delta T_6]\\
&\Delta Q_5 = \Delta Q_5^b + w_{25}\Delta Q_2^a = \Delta Q_5^b + w_{25}[c_{12}^a\Delta T_1 + c_{22}^a(w_{24}\Delta T_4 + w_{25}\Delta T_5) + c_{23}^a w_{36}\Delta T_6]\\
&\Delta Q_6 = \Delta Q_6^b + w_{36}\Delta Q_3^a = \Delta Q_6^b + w_{36}[c_{13}^a\Delta T_1 + c_{23}^a(w_{24}\Delta T_4 + w_{25}\Delta T_5) + c_{33}^a w_{36}\Delta T_6]
\end{aligned} \tag{8}$$

Combining Equations 7 with 8, the following global matrix **C** is formed:

$$\begin{bmatrix}\Delta Q_1\\ \Delta Q_4\\ \Delta Q_5\\ \Delta Q_6\end{bmatrix} = \begin{bmatrix} c_{11}^a & w_{24}c_{12}^a & w_{25}c_{12}^a & w_{36}c_{13}^a\\ & c_{44}^b + w_{24}^2 c_{22}^a & c_{45}^b + w_{24}w_{25}c_{22}^a & c_{46}^b + w_{24}w_{36}c_{23}^a\\ & & c_{55}^b + w_{25}^2 c_{22}^a & c_{56}^b + w_{25}w_{36}c_{23}^a\\ & & & c_{66}^b + w_{36}^2 c_{33}^a\end{bmatrix}\begin{bmatrix}\Delta T_1\\ \Delta T_4\\ \Delta T_5\\ \Delta T_6\end{bmatrix} \tag{9}$$

System of Equations 9 does not involve incremental temperature ΔT_2 and ΔT_3 as they are not independent quantities. After system (9) is solved, incremental heat and temperatures at nodes 2 and 3 can be found from Equations 7 and 8. As in the previous case with no attaches, matrix **C** in Equation 9 is dense, while it is typically very sparse for real engineering problems. If there is a chain of connections between nodes via

attach or interface conditions, this chain must be resolved to use cumulative weights and to form the correct global matrix **M**.

Note again that matrix **M** is symmetric, sparse and positive definite (to reflect the fact that it is equivalent to the heat conductivity matrix in the Fourier's law). In general, the matrix may become close to positive semi-definite as its' smallest eigenvalue may be very small for large timesteps.

3 NEW IMPLICIT SOLVERS

The typical and most efficient approach to solving large sparse symmetric positive definite (SPD) systems is to use preconditioned conjugate gradient (CG) method (Ford 2014, Intel 2019). Direct approaches based, for example, on LU-, QR-, or Cholesky decomposition can be adopted for solving smaller sparse systems and they may be more efficient if the number of iterations in the CG method becomes too large or if the system is semi-definite or indefinite. Both of these approaches are implemented in the current work and they are based on numerical algorithms available in Intel ® Math Kernel Libraries (C-language interface). MKL routines are highly optimized and provide a relatively simple interface (Intel 2019).

3.1 *Conjugate gradient solver*

The conjugate gradient (CG) method adopted in the current work is based on the reverse communication interface (RCI), which implements a group of user-callable routines that are used in the step-by-step solving process (Intel 2019). Based on this logic, several MKL routines are called within a loop until the relative error of the solution falls below a specified limit or until the number of iterations exceeds the specified limit.

Since the system's extremal eigenvalues are calculated to verify whether the CG method can be used (if the system is positive definite), it is also easy to find the system's condition number and evaluate the limit on the number of CG iterations required to reach the desired accuracy. Thus, the iteration limit can be automatically adjusted depending on model parameters, and the solution of an SPD system can always be found up to the desired accuracy. Corresponding stopping criteria and error analysis are provided below.

Let κ be the condition number of matrix **M** defined as the ratio of the largest to the smallest eigenvalues, $\kappa = \lambda_{max}/\lambda_{min}$. Define relative precision of the solution as

$$\varepsilon = \frac{\|x - x_{[n]}\|_M}{\|x - x_{[0]}\|_M} \tag{10}$$

where vector $\boldsymbol{x}$ is the true solution of the system, $\boldsymbol{x}_{[n]}$ is the solution at the n-th iteration, $\boldsymbol{x}_{[0]}$ is the initial guess, and $\|\mathbf{v}\|_M = \sqrt{\mathbf{v}^T \mathbf{M} \mathbf{v}}$ is M-matrix norm of vector **v**. A well-known result for CG convergence estimate is (Saad 2003)

$$\varepsilon \leq 1/T_n\left(\frac{\lambda_{max}+\lambda_{min}}{\lambda_{max}-\lambda_{min}}\right) < 2\exp\left(-2n/\kappa\right) \tag{11}$$

where T_n is Chebyshev n-th degree polynomial and n is the number of iterations in the CG algorithm necessary to reach relative precision ε. Therefore,

$$n \leq \frac{1}{2}\kappa \log\frac{2}{\varepsilon} \tag{12}$$

Provided that the condition number is found, Equation 12 serves as an estimate for the required number of iterations in the CG method and it is implemented in the code for $\varepsilon = 10^{-6}$. The code also caps the maximum allowed number of iterations at 1000. Thus, if for any reason the CG algorithm does not converge to the specified precision within $\min(n, 1000)$ iterations, it is considered that the solver failed to find the solution.

Expression 12 provides an upper estimate for the number of iterations. However, the CG method may converge to the specified accuracy in a smaller number of iterations. Therefore, more precise stopping criterion is implemented in the code and it is also based on Equation 10 for consistency.

Relative error in Equation 10 cannot be directly evaluated as the exact solution is unknown. Taking the initial guess $\boldsymbol{x}_{[\mathbf{0}]} = 0$, it can be shown that

$$\varepsilon_{[\mathrm{k}]} = \frac{\|\boldsymbol{b}-\boldsymbol{b}_{[\mathrm{k}]}\|_{M^{-1}}}{\|\boldsymbol{b}\|_{M^{-1}}} \tag{13}$$

where $\boldsymbol{b}$ is known RHS in Equation 4 and $\boldsymbol{b}_{[\mathbf{k}]} = \mathbf{M}\boldsymbol{x}_{[\mathbf{k}]}$ is RHS obtained at the *k*-th CG iteration. If relative error $\varepsilon_{[k]} \leq 10^{-6}$, iterations stop. While vectors $\boldsymbol{b}$ and $\boldsymbol{b}_{[k]}$ can be easily calculated, the inversion of the global matrix **M** is infeasible; instead, it is approximated with the inverse of the preconditioner matrix as discussed below.

The current work adopts a simple diagonal (Jacobi) preconditioner to improve spectral characteristics of the global matrix and reduce the number of iterations in the CG algorithm. The choice of this preconditioner is natural as it can be cheaply calculated with MKL routines, it does not break symmetry (for central application) or change the sparsity of the system, and it performs well for diagonally dominant systems.

Since the preconditioner is chosen to approximate the original matrix **M**, the inverse of the matrix in Equation 13 can also be approximated through preconditioner **P** as

$$\mathbf{P} = diag(\mathbf{M}) \;\Rightarrow\; M_{ij}^{-1} \approx P_{ij}^{-1} = \delta_{ij}/M_{ij} \tag{14}$$

This approximation is found to work well for error estimation in Equation 13.

MKL routine *"dcg"* is used to get the solution vector using the CG method.

3.2 *Direct solver*

In addition to the preconditioned CG method, the MKL Direct Sparse Solver (DSS) is used in the current work. The user has a choice of selecting the CG or DSS algorithms for the implicit scheme. The advantage of the direct solver is that it is robust (can be used to verify results), it works for indefinite matrices, and may be faster in cases when the CG method requires a large number of iterations ($\gtrsim 500$). This often happens when global matrix **M** is close to being positive semi-definite (the smallest eigenvalue tends to zero). Similarly to MKL CG interface, the DSS interface implements a group of user-callable routines that are used in the step-by-step solving process. The algorithm is based on LU, LL^T or Cholesky (selected automatically) factorization.

MKL routine *"dss_solve_real"* is used to get the solution vector using the DSS method.

3.3 *Calculation of eigenvalues*

Knowledge of the smallest and largest eigenvalues of global matrix **M** allows calculating its condition number and check which of the solvers may be more suitable for the problem. Conjugate gradient method is only suitable for positive definite systems for which $\lambda_{min} > 0$. For some problems, the matrix may be close to positive semi-definite if λ_{min} is small (e.g. $0 \leq \lambda_{min} \lesssim 10^{-5}$). If at the same time, the condition number is large ($\kappa \gtrsim 10^3$), a large number of iterations may be required in the CG method. Therefore, use of the DSS method may be more practical even for large sparse matrices. If $\lambda_{min} = 0$, the CG method may not converge and only the DSS solver should be used. Finally, $\lambda_{min} < 0$ suggests that the system is indefinite or negative definite and this is typically an indication of errors in assembling the global matrix.

Minimum eigenvalue is calculated once for global matrix **M** after it is assembled to verify that selected solver is suitable for the system (if it is not, a warning is issued). After that, if the CG method is selected, minimum and maximum eigenvalues are calculated for the preconditioned global matrix to estimate the required number of iterations. If the found number of iterations is large (> 500) or exceeds maximum limit (> 1000), a warning or error is issued.

Routine *"mkl_*sparse*_d_ev"* from the MKL extended eigen-solver interface is used to obtain the extremal eigenvalues. The routine provides a choice of two algorithms, and it was found that the Krylov-Schur method calculates the extremal eigenvalues significantly faster than another available method, and it is used in this work.

4 IMPLEMENTATION

Before using new implicit solvers, global matrix **M** must be assembled based on nodal (grid point) information and local zone matrices. This process is heavily optimized and multithreaded to use all available CPU cores. Due to the symmetry of the global matrix, only the upper triangular part is assembled in row-by-row fashion and stored in three array variation of compressed sparse row (CSR3) format (Intel 2019).

The global matrix is assembled once only before cycling starts with a specified timestep. Quite often, however, there is a necessity to change the timestep after a certain number of cycles while keeping all other model parameters the same. In order to avoid re-assembling the whole global matrix, the time-independent part of **M**, matrix **C**, is stored in memory and time-dependent addition to the diagonal terms is calculated as needed and added to the diagonal when the timestep changes (see Eq. 4).

On the other hand, if model physical or geometrical parameters change in time (e.g. moving boundaries, large strain), the whole global matrix must be re-assembled (which is also related to the use of CSR3 format). This may negate improvements in overall calculation speed when using new solvers. The existing Jacobi solver should be used for such problems, as well as for problems requiring a very small timestep (it may be overall faster for such case).

In the current implementation, the user has to specify which implicit solver to use. The decision has to be made based on calculation speed and applicability, which depend on problem size, timestep, and if the model physical properties or geometry change in time. For large problems, it is recommended to cycle for a few hundred or thousand steps to estimate the calculation speed of selected solver for a given timestep and, if needed, analyze eigevalues / condition number, which are output in *FLAC3D* console. An automated process of selecting the most suitable solver is planned for future developments.

If the user uses the Jacobi solver and it diverges at some cycle, and the problem permits using the CG solver, new functionality provides the capability to automatically switch to the CG solver and proceed calculations. Figure 2 provides a high level overview of all mentioned operations and functionalities.

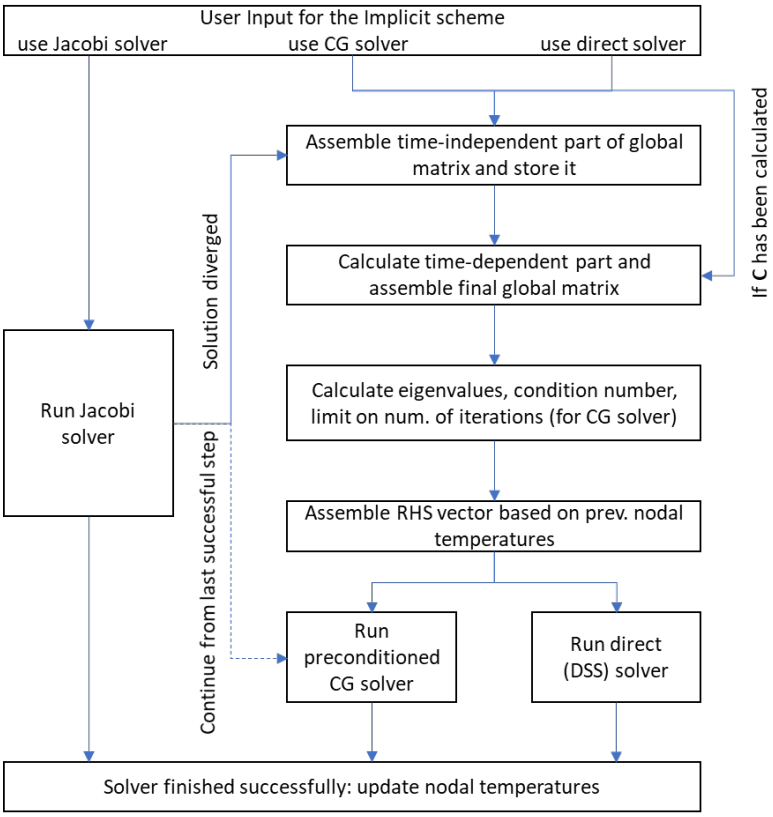

Figure 2. Schematic chart of high-level calling sequence in the implicit scheme.

5 EXAMPLE APPLICATION

The example presented in this section is based on the "Conduction in a Plane Sheet" problem from the *FLAC3D* Thermal Analysis manual (Itasca 2019c). For the detailed description, material properties, and boundary conditions used, refer to the manual. The only modification used in the current example is the geometry of the model: a wedge geometry (Fig. 3a) is created instead of a column of bricks using the command

```
zone create wedge size 15 40 10 point 1 (0.1,0,0) point 2 (0,0,1) point 3 (0,-0.1,0)
```

Constant temperature $T = 100°C$ is applied at the bottom of the wedge while the top is kept at 0°C. After some time, equilibrium state is reached with constant heat flux and unchanging temperature distribution.

The presence of very narrow zones at the tip of the wedge drives the explicit timestep to be very small. The results of a simulation of 15 seconds of heat conduction using the explicit and implicit methods with the Jacobi, preconditioned CG, and direct solvers are presented in Table 1. The implicit solvers are run to reach accuracy of solution of at least 5.0e-4. The results are compared between each other and with the analytical solution and show excellent agreement (Fig. 3b).

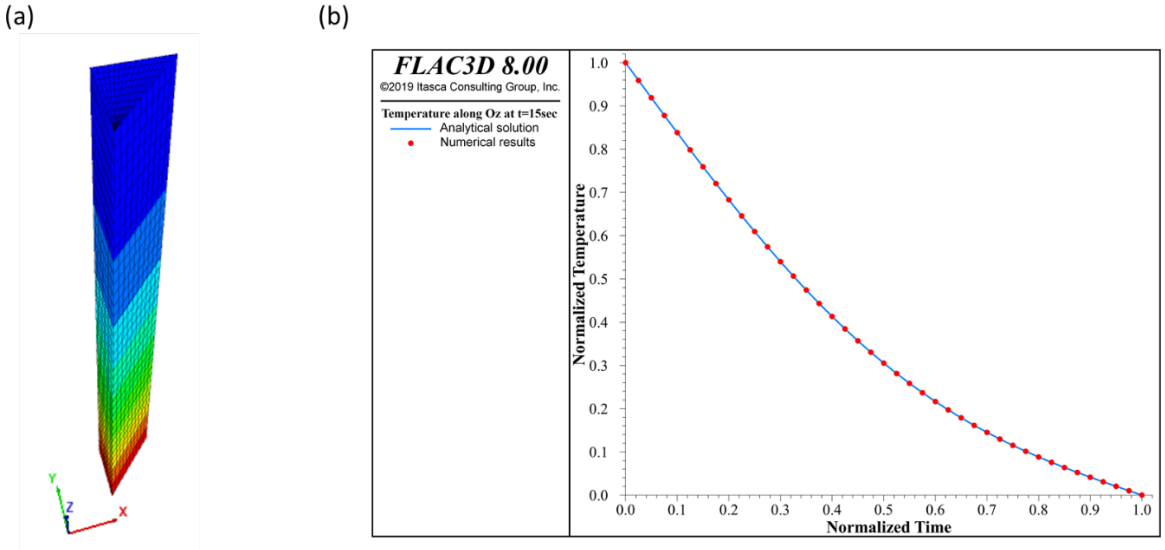

Figure 3. Temperature distribution at 15sec and comparison of numerical (obtained with the preconditioned CG solver) and analytical solutions in vertical direction normalized to $T = 100°C$ and $t = 15$sec.

Table 1. Simulation results of 15sec of heat conduction. The runtime reported in the last row was obtained on a PC with Intel i7 5820 6-core processor and 32Gb RAM.

Scheme / Solver	Explicit	Implicit / Jacobi	Implicit / precond. CG	Implicit / direct
Timestep	1.28856e-5	1e-4	0.5	0.5
Number of steps	1164094	150000	30	30
Error[2]	2e-5	5e-4	9e-5	9e-5
Relative speed-up (runtime)	1X (11m02s)	2.1X (5m13s)	441X (1.5sec)	662X (1sec)

Note that the timestep used in the Jacobi solver is the largest stable timestep; increasing it further makes the solver diverge and calculations fail. The timestep used in the preconditioned CG and direct solvers can be increased further; however, it causes noticeable numerical oscillations in the results close to the bottom boundary (Fig. 4). These oscillations quickly decay as cycling proceeds – this is a known side-effect of the

[2] Error is defined as Euclidean norm between normalized numerical and analytical solutions.

Crank-Nicolson scheme (Østerby 2003). In the current application of the scheme, the oscillations appear as a result of the combination of a large timestep and a small number of iterations (cycles).

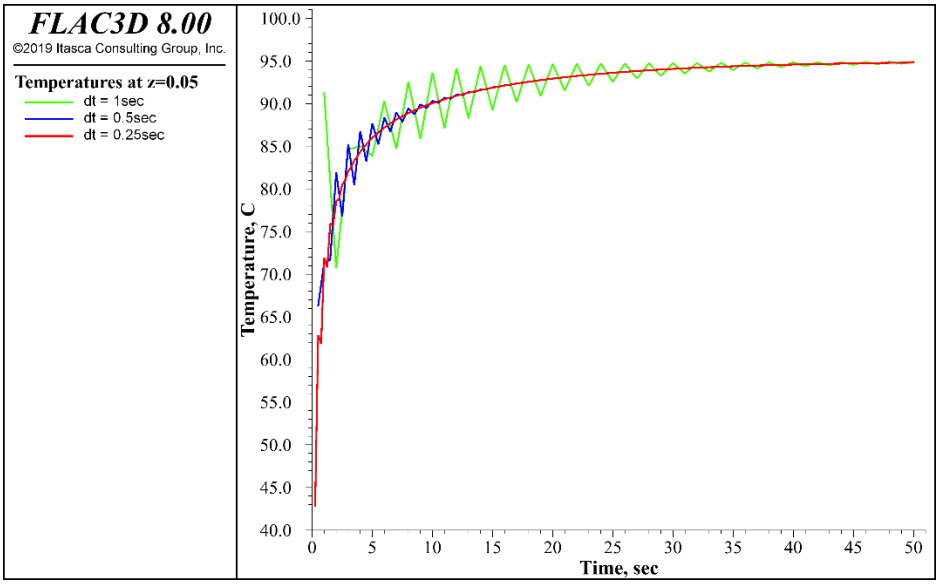

Figure 4. Temperature history at z=0.05 depending on the timestep used in the preconditioned CG or direct solvers.

When using the CG or direct solvers in the implicit scheme, it is recommended to monitor histories near the boundaries of the model or other discontinuities, especially if a large timestep is used in combination with a small number of cycles. The oscillations do not cause the solution to diverge but results may be inaccurate in the oscillatory region. In such cases, the timestep should be decreased or the number of cycles increased.

6 CONCLUSIONS

This work presents an efficient new approach of solving thermal and fluid problems in *FLAC3D* using an implicit scheme with two new solvers: preconditioned conjugate gradient and direct solvers. The new solvers are robust and stable and can operate with large thermal/fluid timestep limited only by the characteristic time of the model (or, in certain cases, by the potential instabilities of the Crank-Nicolson method; see discussion above). The new solvers are capable of handling very large problems with millions of zones, attach and interface conditions, and various boundary conditions. However, the new solvers may be inefficient for problems involving changing geometry or physical properties (e.g. large strain or partial saturation problems) as the global matrix has to be re-assembled each time the model parameters change. The existing Jacobi solver should be used in such cases.

Currently the solvers are implemented for thermal logic and tested on thermal problems involving heat conduction, thermo-mechanical coupling, constant and transient boundary conditions, heat sources/sinks, convective boundaries. Ongoing developments will allow using this new methodology for fluid problems and thermal-fluid coupling.

REFERENCES

Ford, W. 2014. *Numerical Linear Algebra with Applications Using MATLAB.* Chapters12, 21. Academic Press.

Intel Corporation. 2019. *Intel® Math Kernel Library Developer Reference*. Revision: 024, MKL 2019.

Itasca Consulting Group, Inc. 2019 (a). *FLAC3D – Fast Lagrangian Analysis of Continua in 3-Dimensions, Ver. 7.0, Theory and Background*. Minneapolis: Itasca.

Itasca Consulting Group, Inc. 2019 (b). *FLAC3D – Fast Lagrangian Analysis of Continua in 3-Dimensions, Ver. 7.0, Fluid-Mechanical Interaction*. Minneapolis: Itasca.

Itasca Consulting Group, Inc. 2019 (c). *FLAC3D – Fast Lagrangian Analysis of Continua in 3-Dimensions, Ver. 7.0, Thermal Analysis*. Minneapolis: Itasca.

Østerby, O. 2003. Five Ways of Reducing the Crank–Nicolson Oscillations. *BIT Numerical Mathematics*. 43(4): 811–822.

Saad, Y. 2003. *Iterative Methods for Sparse Linear Systems, 2nd Ed.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA

Thomas, J.W. 1995. *Numerical Partial Differential Equations: Finite Difference Methods*. *Texts in Applied Mathematics.* Vol. 22. New York: Springer-Verlag.