On parallelization of the Openfoam-based solver for the heat transfer in electrical power cables

R. ČIEGIS, A. Bugajev, V. Starikovičius

Vilnius Gediminas Technical University, Vilnius, Lithuania
e-mail: rc@vgtu.lt

"Mathematical Modelling and Analysis 2014", May 26-29, 2014, Druskininkai

Benchmark problem

We solve the 2D heat conduction problem for electrical power cables directly buried in the soil

$$\begin{cases} c\rho\dfrac{\partial T}{\partial t} = \nabla\cdot(\lambda\nabla T) + q, & t \in [0, t_{max}], \vec{x} \in \Omega, \\ T(\vec{x}, 0) = T_b, & \vec{x} \in \Omega, \\ T(\vec{x}, t) = T_b, & \vec{x} \in \partial\Omega, \\ T, \lambda\nabla T \ \text{ are continuous}, & \vec{x} \in \Omega, \end{cases}$$

We have used 2D geometry for our benchmark problem. Three cables are buried in the soil as shown in Fig. **??**.

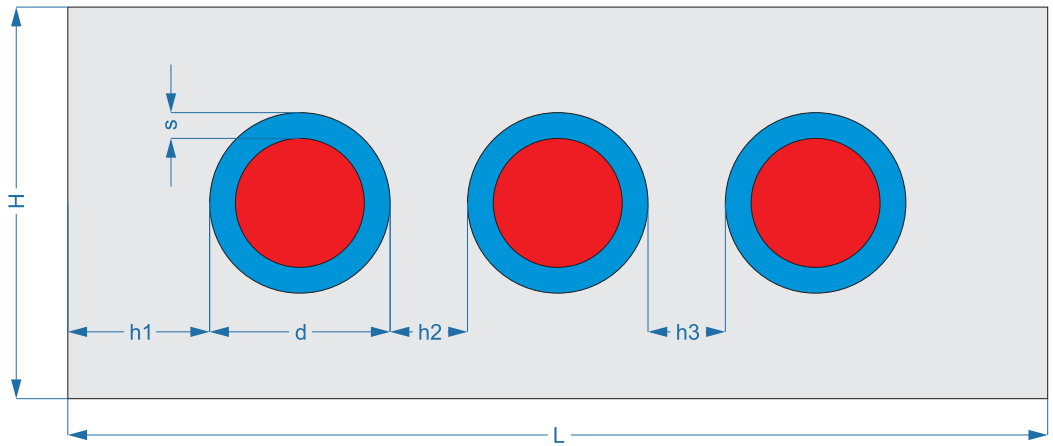

FIGURE: 2D geometry of benchmark problem: three cables in the soil.

The red area is metallic conductor, the blue area is an insulator and the gray area marks the soil.

- OpenFOAM (Open source Field Operation And Manipulation) is a C++ toolbox (library) for the development of customized numerical solvers for partial differential equations (PDEs).

- OpenFOAM (Open source Field Operation And Manipulation) is a C++ toolbox (library) for the development of customized numerical solvers for partial differential equations (PDEs).
- OpenFOAM uses the Finite Volume Method (FVM) with co-located arrangement of unknowns.

- OpenFOAM (Open source Field Operation And Manipulation) is a C++ toolbox (library) for the development of customized numerical solvers for partial differential equations (PDEs).
- OpenFOAM uses the Finite Volume Method (FVM) with co-located arrangement of unknowns.
- We have constructed our numerical solver by a slight modification of the standard *laplacianFoam* solver.

Parallel OpenFOAM-based solver

- Parallelization in OpenFOAM is robust and implemented at a low level using MPI library.

Parallel OpenFOAM-based solver

- Parallelization in OpenFOAM is robust and implemented at a low level using MPI library.
- Solvers are built using high level objects and don't require any parallel-specific coding. They will run in parallel automatically.

Parallel OpenFOAM-based solver

- Parallelization in OpenFOAM is robust and implemented at a low level using MPI library.
- Solvers are built using high level objects and don't require any parallel-specific coding. They will run in parallel automatically.
- There is no need for users to implement standard steps of any parallel code: decomposition of the problem into subproblems, distribution of these tasks among different processes, implementation of data communication methods.

The main drawback:

Users have very limited possibilities to modify the generated parallel algorithm if the efficiency of the OpenFaom parallel code is not sufficient.

- OpenFOAM employs a common approach for parallelization of numerical algorithms: domain decomposition.

  The mesh and its associated fields are split into sub-domains, which are allocated to different processes.

- OpenFOAM employs a common approach for parallelization of numerical algorithms: domain decomposition.

  The mesh and its associated fields are split into sub-domains, which are allocated to different processes.

- Parallel computation of the proposed finite FVM algorithm requires two types of communication:

  a) local communications between neighboring processes for approximation of the laplacian term on the given stencil;

  b) global communications between all processes for computation of scalar products in IC iterative method.

Domain Decomposition

- OpenFOAM supports four methods of domain decomposition, which decompose the data into a non-overlapping sub-domains: simple, hierarchical, scotch and manual.

Domain Decomposition

- OpenFOAM supports four methods of domain decomposition, which decompose the data into a non-overlapping sub-domains: simple, hierarchical, scotch and manual.
- We have used Scotch library, which is a tool for graph and mesh partitioning, similar to well-known Metis library.

Domain Decomposition

- OpenFOAM supports four methods of domain decomposition, which decompose the data into a non-overlapping sub-domains: simple, hierarchical, scotch and manual.
- We have used Scotch library, which is a tool for graph and mesh partitioning, similar to well-known Metis library.
- Users can specify the weights of the sub-domains, what can be useful on heterogeneous clusters of parallel computers with different performance of processors.

Scalability Analysis

- We estimate the costs of the sequential algorithm to compute a solution at one time step as

$$W = (c_1 + 1000c_2)J,$$

J is the total number of finite volumes in the mesh,
c_1 estimates the costs of computation of coefficients of the discrete linear system,
c_2 estimates the costs of one IC iteration.

- Let us assume that p homogeneous processes are used in computations. Then computation costs of the parallel algorithm can be estimated as

$$T_p^1 = (c_1 + 1000c_2)\lceil J/p \rceil.$$

Here we are not taking into account that $c_1 = c_1(p)$, $c_2 = c_2(p)$ are not constant for the parallel algorithm, they can decrease for $p > 1$ due to better cashing of smaller size parallel subproblems.

We assume that the largest number of data items send between neighbour processes can be estimated as $c_3\sqrt{J}$ and let M be the largest number of neighbours for some process.

Then the communication costs can be estimated as

$$T_p^2 = 1000\big[r(M)(\alpha + \beta c_3\sqrt{J}) + R(p)\big].$$

Here α denotes the message startup time and β is the time required to send one element of data.

Thus the total complexity of the parallel algorithm is equal to

$$T_p = (c_1 + 1000c_2)\lceil J/p \rceil + 1000\Big[r(M)(\alpha + \beta c_3\sqrt{J}) + R(p)\Big].$$

Thus the total complexity of the parallel algorithm is equal to

$$T_p = (c_1 + 1000c_2)\lceil J/p \rceil + 1000\Big[r(M)(\alpha + \beta c_3\sqrt{J}) + R(p)\Big].$$

The scalability analysis finds the rate at which the size of the sequential algorithm W needs to grow up with respect to the number of processes p in order to maintain fixed the theoretical efficiency of the parallel algorithm $E_p = W/(pT_p)$.

For a given efficiency E the isoefficiency function $W = g(p, E)$ is defined by the implicit equation

$$W = E/(1-E)\, H(p, W)\,.$$

The total overhead of the proposed parallel algorithm is given by

$$\begin{aligned} H(p, W) &:= pT_p - W \\ &= (c_1 + 1000c_2)\big(p\lceil J/p\rceil - J\big) + 1000p\big[r(M)(\alpha + \beta c_3\sqrt{J}) + R(p)\big]. \end{aligned}$$

For a given efficiency E the isoefficiency function $W = g(p, E)$ is defined by the implicit equation

$$W = E/(1-E)\, H(p, W)\,.$$

The total overhead of the proposed parallel algorithm is given by

$$\begin{aligned} H(p, W) &:= pT_p - W \\ &= (c_1 + 1000c_2)\big(p\lceil J/p\rceil - J\big) + 1000p\big[r(M)(\alpha + \beta c_3\sqrt{J}) + R(p)\big]. \end{aligned}$$

Asymptotical isoefficiency functions due to local and global communications have the same order $W = O(p^2)$.

Parallel Performance Tests

The computations are done on Vilkas cluster at VGTU.

- 8 Intel Quad i7-860 processors (2.80 GHz) and 8 Pentium 4 processors (3.2 GHz) are interconnected via Gigabit Smart Switch.

Parallel Performance Tests

The computations are done on Vilkas cluster at VGTU.

- 8 Intel Quad i7-860 processors (2.80 GHz) and 8 Pentium 4 processors (3.2 GHz) are interconnected via Gigabit Smart Switch.
- Intel Quad i7-860 processors are not fully homogeneous.

Parallel Performance Tests

The computations are done on Vilkas cluster at VGTU.

- 8 Intel Quad i7-860 processors (2.80 GHz) and 8 Pentium 4 processors (3.2 GHz) are interconnected via Gigabit Smart Switch.
- Intel Quad i7-860 processors are not fully homogeneous.
- Vilkas cluster is quite heterogeneous and additional weighted load balancing is included into the mesh distribution step.

TABLE: CPU times of the sequential algorithm for different sizes of the problem and different processors of Vilkas cluster: *i*7-*x* and *q*-*x* denote the *x*-th Intel Quad i7-860 and Pentium 4 processor, respectively.

J	128000	254892	512000	1018488	2048000
*i*7-0	36.7	82.7	201.9	415.9	909.2
*i*7-1	36.6	82.9	202.0	415.5	893.3
*i*7-2	36.7	82.4	198.6	413.0	887.9
*i*7-3	35.2	77.7	177.0	362.3	772.0
*i*7-4	36.8	82.4	198.3	415.8	887.6
*i*7-5	36.8	83.6	199.0	412.6	874.4
*i*7-7	36.1	81.1	190.7	390.2	839.3
*i*7-8	36.9	83.2	200.3	417.8	868.1
q-8	66.2	151.3	336.5	677.5	1441
q-13	66.3	153.9	341.3	678.2	1442

- Due to memory cashing effects, the CPU time of the OpenFoam solver increases over-linearly with respect to the size J of the discrete problem.

  A theoretical model for OpenFoam solver

$$T_p^1(J) = \max_{x \in G} T_0(x, J/p).$$

- Due to memory cashing effects, the CPU time of the OpenFoam solver increases over-linearly with respect to the size J of the discrete problem.

  A theoretical model for OpenFoam solver

$$T_p^1(J) = \max_{x \in G} T_0(x, J/p).$$

- Intel Quad i7-860 processors are approximately 1.6 times faster than Pentium 4 processors. In addition some Intel Quad i7-860 processors are till 1.15 times faster than the remaining processors.

TABLE: CPU times of the parallel OpenFoam algorithm for different sizes of the problem and different sets of processors.

J	128000	254892	512000	1018488	2048000
*i*7-0, *i*7-1	18.8	39.3	86.7	204.9	420.2
*i*7-3, *i*7-7	18.4	38.3	86.1	193.3	394.4
q-8, *q*-9	30.7	73.2	162.0	337.5	689.5
*i*7-3, *q*-8	25.5	67.41	155.4	331.6	683.6
*i*7-0, 1, 2, 4	11.8	20.4	41.3	89.4	209.4
*i*7-3, 5, 7, 8	11.4	20.4	40.9	90.0	204.3
q-8, 9, 12, 13	23.4	37.9	79.4	168.5	347.6
8 *i*7 nodes	8.8	13.6	22.4	43.6	94.3
8 *q* nodes	22.1	28.0	44.3	86.7	179.5
16 nodes	17.3	20.5	26.2	42.7	85.4

TABLE: CPU times of the parallel OpenFoam algorithm for the adaptive mesh decomposition algorithm. Here $i7(w)$ and $q(w)$ denote Intel Quad i7-860 or Pentium 4 processors and w denotes the relative speed.

J	512000	1018488	2048000	8192000
$i7$-3(1), q-8(1) q-9(1)	102.0	222.0	461.1	
$i7$-3(2), q-8(1) q-9(1)	89.7	189.9	381.3	
$i7$-3(1.87), q-8(1) q-9(1)	88.8	183.0	371.8	
8 $i7(1)$, 8 $q(1)$	26.2	47.7	85.4	363.6
8 $i7(1.5)$, 8 $q(1)$	24.3	35.5	69.4	288.2
8 $i7(1.6)$, 8 $q(1)$	24.5	35.7	66.1	279.9

TABLE: CPU times of the parallel OpenFoam algorithm for different numbers of processors n_d and $n_c = 2$ cores per node. The size of the problem is $J = 2048000$ elements.

	1×1	1×2	2×2	4×2	8×2
i7	772.0	566.5	288.2	142.7	64.8
q	1441	1161	573.6	270.2	122.7

TABLE: CPU times of the parallel OpenFoam algorithm for the adaptive mesh decomposition algorithm. Here $i7(w)$ and $q(w)$ denote Intel Quad i7-860 or Pentium 4 processors and w denotes the relative speed of these processors.

J	$1i7$	$8i7$	$8(i7+q)$	$8(i7(1.6)+q)$	$16i7$	$16(i7(1.6)+q)$
$2 \cdot 10^6$	772	94.3	85.4	66.1	64.8	40.5
$8 \cdot 10^6$			363.6	280	280	202.4