ISSN 2039-7941
anno 2022 numero 441

RAPPORTI
TECNICI INGV

Strong scaling analysis of the code GALES on 3D fluid, solid, and FSI problems

441

ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA

REGISTRAZIONE AL TRIBUNALE DI ROMA N.174 | 2014, 23 LUGLIO

ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA

ISSN 2039-7941
anno **2022** numero **441**

RAPPORTI
TECNICI INGV

Strong scaling analysis of the code GALES on 3D fluid, solid, and FSI problems

Deepak Garg, Paolo Papale

INGV | Istituto Nazionale di Geofisica e Vulcanologia, Sezione di Pisa

Accepted 2 August 2021 | *Accettato 2 agosto 2021*

How to cite | *Come citare* Garg D., Papale P., (2022). Strong scaling analysis of the code GALES on 3D fluid, solid, and FSI problems. Rapp. Tec. INGV, 441: 1-28, https://doi.org/10.13127/rpt/441

Cover | *In copertina* Load-balanced mesh partitioning on 4 processors where each processor shares the same number of elements

441

INDEX

Abstract

GALES is a general-purpose multiphysics finite element software developed at INGV Pisa, and it consists of various solvers for computational fluid dynamics (CFD), computational solid dynamics (CSD) and fluid-solid interaction (FSI). The software has been verified and validated on several benchmarks. The software is intended to be used for high-performance computing (HPC) on cluster machines and has been in regular use to study 2D volcanology problems such as magma transport and rock deformation. Recently, the numerical code has been extended to 3D. In this report, we present the implementation of GALES for a parallel environment and perform a strong scaling study on various solvers of GALES for 3D problems.

Keywords Strong-scaling; Parallel computing; Finite element method

Introduction

Computational problems often take a large amount of execution time when solved on a single computer. High-performance computing (HPC) is a paradigm where many processors work simultaneously to produce exceptional computational power and to significantly reduce the total computational time [Flanagan et al., 2020; Dowd and Severance, 2010]. Numerical codes are parallelized for the parallel computing in which large problems can often be divided into smaller ones, which can then be solved at the same time [Gottlieb et al., 1989; Asanovic et al., 2006]. A parallel numerical code is often checked for its performance through a strong-scaling test i.e. the execution time should reduce by adding more hardware [Blake et al., 2006; Muller et al., 2015]. In this work, we perform a strong-scaling test on the GALES code. The results obtained by this work will provide an indication of the number of resources to be requested for the size of a particular job.
GALES is a multi-physics 2D/3D finite element method (FEM) code developed at INGV-Pisa [Longo et al., 2012a; Garg et al., 2018a; Garg et al., 2018b]. The code is written in object-oriented C++ and is parallelized using OpenMpi for parallel computing on HPC environment. GALES deploys massively parallel unstructured, implicit solvers for computational fluid dynamics (CFD), solid dynamics (CSD) and fluid-solid/structure interaction (FSI) dynamics problems. Although GALES can be used in a wide variety of problems, primarily it was developed to study the mixing of multicomponent and multiphase magmas and their transportation in the crust [Longo et al., 2006]. FSI solver developed in GALES allows to directly link deep magma dynamics in geometrically complex systems to the geophysical signals recorded on the Earth's surface.
GALES is comprised of mainly three packages: GALES-CFD, GALES-CSD and GALES-FSI. The packages include multiple solvers for different kind of problems.
GALES-CFD contains three solvers: SC-ICO, SC-CO and MC-miscible. SC-ICO and SC-CO are, respectively, for the incompressible and compressible flow of a single-component fluid. MC-miscible solves for multicomponent miscible compressible and incompressible flows. The solvers can be used for transient as well as steady flow problems. To solve the flow problems on deforming domains such as free flow and moving boundaries, all the solvers are implemented in the arbitrary Lagrangian-Eulerian (ALE) frame of reference [Souli et al., 2001]. For fixed domain problems, we solve the problems in the Eulerian setting which is obtained by setting mesh velocity equal to zero in the ALE model. Our fluid solvers are based on the

Galerkin least square based stabilized finite element method [Masud et al., 1997; Tezduyar et al., 1992]. Lagrange interpolation functions are used to discretize space and solution variables. We use Euler's backward method for temporal discretization. The Navier-Stokes equations (conservation of mass, momentum, energy and mass fraction) are solved through a monolithic approach [Hauke and Hughes, 1998; Longo et al., 2012a].
GALES-CSD include four solvers: ED for elastodynamics, ES for elastostatics, VED for visco-elastodynamics and VES for visco-elastostatics [Hughes, 1987]. In all solvers, we use the standard Galerkin method for the spatial domain and the generalized alpha-method for the time discretization [Chung and Hulbert, 1993]. We have implemented several constitutive models which include Hooke's law for an elastic material, Saint-Venant Kirchoff and Neo-Hookean model for a hyperelastic material [Simo and Hughes, 1998], and generalized Maxwell model for visco-elastic materials.
For FSI problems, we chose a partitioned approach in GALES, which allows us to combine any CFD and CSD solvers effortlessly [Schafer et al., 2006]. We use Dirichlet-Neumann conditions to establish the kinematic and dynamic equilibrium at the fluid-solid interface [Farhat et al., 1998]. A set of classes have been implemented to establish the coupling between fluid and solid solvers.
The code has been applied to flows ranging from validation benchmarks [Garg et al., 2018a; Garg et al., 2018b; Longo et al., 2012a] to cases of practical interest [Longo et al., 2012b; Papale et al., 2017, Garg et al., 2019, Bagagli et al., 2017]. In this work, we do not provide a detailed description of the physical models and mathematical formulations for different solvers of GALES. Instead, we present the strong scaling results of solvers for 3D meshes. The computations have been performed on the Marenostrum supercomputer in Barcelona Supercomputing Centre (BSC) within the project GALES-3D (2010PA5625) in the frame of PRACE preparatory access A (https://prace-ri.eu/).

1. Computational Hardware

The Marenostrum supercomputer is a Lenovo system composed of 48 SD530 Compute Racks. A single rack holds 72 Lenovo stark compute nodes and 2 Lenovo G8282 switches. Each node is composed of 48 Intel Xeon Platinum cores. Thus, a single rack contains 3456 cores. The total number of computational processors is 165888 and the total memory of the cluster is 384.75 TB. The current measured Linpack Rmax Performance of the cluster is 6.2272 Petaflops. Within a rack, the 3456 computing cores are interconnected through a 100 GB Intel Omni-Path (OPA) Full-Fat Tree high-speed interconnection network.
The different racks are interconnected by a 10 GB Ethernet network. As per MareNostrum user's guide (http://www.bsc.es/support/MareNostrum4-ug.pdf), for PRACE projects, 19200 is the maximum number of cores to submit the jobs, out of the total available 165888 cores of the cluster. For additional details, we refer the reader to the technical details page of Marenostrum (https://www.bsc.es/marenostrum/marenostrum/technical-information).

2. Parallel implementation in GALES

All solvers in GALES are implicit and based on the finite element method. In the framework of the FEM, computations are carried out at the element level which is well suited for parallel computers as the computational load can be well balanced by distributing an almost equal number of elements on different processors [Vollaire et al., 1998]. Figure 1 displays a load-balanced mesh split over four processors. Typically, an implicit FEM solver spends most of its execution time in the following two steps during each non-linear iteration:

1. Element level computations and assembly of the linear system of equations
2. Solution of the linear system

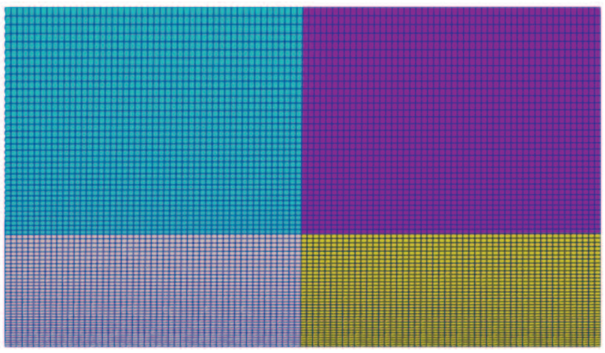
Figure 1 Load-balanced mesh partitioning on 4 processors where each processor shares the same number of elements.

Either of these steps can be performed in parallel. In the following, we present the details of GALES implementation to perform these steps.

2.1. Mesh generation

Computational meshes are generated by GMSH, which is a popular open-source widely used software [Geuzaine and Remacle, 2009]. The mesh file includes the information on the nodes coordinates, element topology, edge connections between nodes and the boundary data. For a simple domain geometry (box, sphere, etc.), GMSH is quite fast, taking less than a minute, in creating up to several million unstructured tetrahedral elements. However, on complex domains for more than 10 million elements, we notice that GMSH takes a significant amount of time (of order hours). To read the meshes in GALES we rewrite them in a different format using a python script. In the new mesh file, each node is identified with its coordinates and the boundary type. Similarly, each element stores the information about its nodes as well as an extra tag that classifies if the element lies on the boundary or in the interior of the domain.

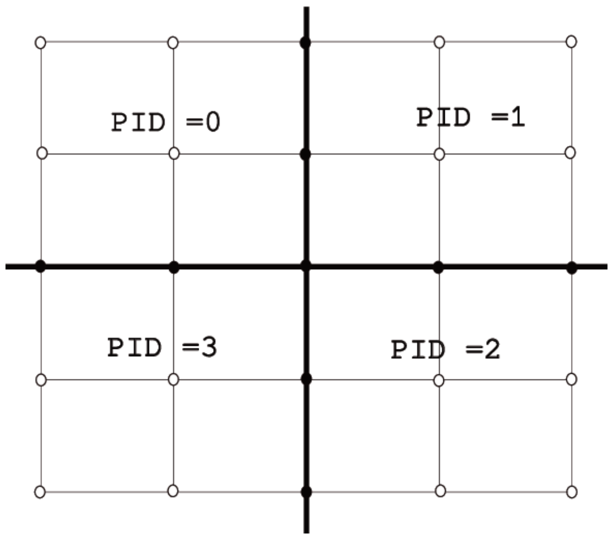

Figure 2 A sketch showing mesh partitioned over 4 processors represented by processor ID (PID) numbers. Solid dots indicate the shared nodes among different processors.

2.2 Mesh partitioning

In an HPC environment, typically, the computational mesh is partitioned into multiple parts. Each part is assigned to a processor (process) that carries out all the numerical operations corresponding to that part of the mesh. In GALES, we use the element based domain partition strategy, in which each element is assigned to a unique processor, but nodes can belong to multiple processors if they belong to the element lying on the boundary between different sub-domains (we call them inter-processor boundary nodes or shared nodes, see Figure 2). By assigning whole elements to each processor, we can minimize data dependency during the assembly procedure. To partition the mesh on different processors, we use the METIS library, which uses the multilevel heuristic graph partitioning approach and assigns an almost equal number of elements to processors [Karypis and Kumar, 1999]. That is crucial for load-balancing and the performance of a parallel application. The METIS library also minimizes the number of inter-processor boundary nodes, hence reduce the amount and the cost of inter-processor communication. Overall computational time is the sum of execution and communication times. The execution time is directly proportional to the number of partitions and the load-balance on processors. It involves the element level integral computations at the Gauss quadrature points and their assembly into a global tangent matrix and the right-hand side residual vector. The communication time depends on the number of nodes lying on inter-processor boundaries. The partitions coming from METIS are optimal in reducing execution as well as communication time. METIS partitions the mesh in serial and is fast up to several millions of elements, taking only a few seconds. In this work, we use meshes of up to 50 million elements. For larger meshes consisting of billions of nodes, METIS may be overwhelmingly computationally expensive. In future, we will move to a more optimal library such as ParMETIS [Schloegel et al., 2002] (an MPI based parallel extension of METIS) or Zoltan [Boman et al., 2012].

2.3 Element integrals

For the element level computations, we construct vectors and matrices from the boost library. The library provides a range of convenient functions for the arithmetic of vectors and matrices. Boost library is more convenient to use for the element level computations in the frame of FEM. Some of the frequently used operations are matrix-matrix multiplication and matrix-vector multiplication, transpose of a matrix and the inverse of a matrix, the norm of a vector and the dot product of vectors.

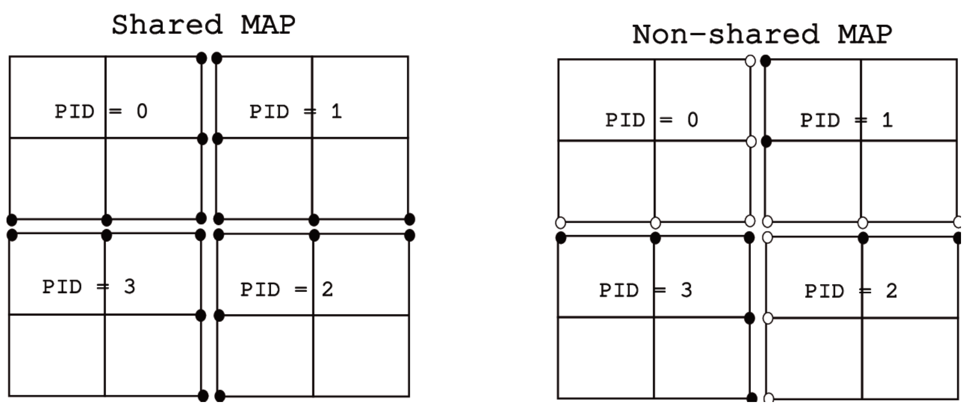

Figure 3 Illustration of shared and non-shared maps. In shared map the solid dots indicate that the node is included for the map generation. In the non-shared map the solid dots indicate the owner PID of that node whereas the hollow ones indicate the non-owner PID.

2.4 DOFs distribution on processors

The distribution of degrees of freedom (DOFs) across the processors is taken care of by suitable mappings. In GALES, we construct maps with the *Epetra_Map* class (https://trilinos.github.io/epetra.html) of the Trilinos library (https://trilinos.github.io/index.html). The maps encapsulate the details of distributing data over MPI processors. For a given global id (GID) list of the DOFs on processors, an *Epetra_Map* class object generates a corresponding local id (LID) list of the DOFs on each processor which are identified with the corresponding processor id (PID). In GALES, we create two different distributed maps that we call shared map and non-shared map.
The shared map is based on the element distribution. To construct the map, on each PID, we collect the GID list of the DOFs of the nodes of the elements. The GID list also includes the DOFs of the nodes that lie on the inter-processor boundaries (see Shared MAP in Figure 3). In the shared map an element lying on the inter-processor boundary is entirely independent of its neighbouring element that lies on a different PID. Hence, no inter-processor communication is needed when we need to extract the DOFs of an element. That is particularly relevant for the transient problems as the solution obtained at the previous iteration or at the previous time step is often needed in the implicit methods to construct the linear system for the current iteration. In GALES, we use the shared map to initialize the vector of DOFs for the application problem and use it extensively to carry the solution forward in transient simulations.
The non-shared map is based on the DOFs of the mesh nodes. Before creating the map each node lying on inter-processor boundaries is assigned uniquely to an owner processor. This is to ensure that the aggregate of DOFs over owner PIDs is equal to the total DOFs of the computation mesh and is independent of the mesh parts. In our implementation, if a node lies on the inter-processor boundary then it is assigned to be owned by the processor who has the highest PID number (see Non-Shared MAP in Figure 3). The maps themselves are distributed and do not store all GIDs on a single processor. This ensures memory scalability because no individual processor has to save all the data.

2.5 Linear system of equations

In all solvers, FEM discretization results in a linear system of algebraic equations, $Ax = b$ where A is the sparse tangent matrix, x is the solution vector, and b is the non-linear residual vector. The dimensions of the linear system are proportional to the number of DOFs of the collective mesh. The global sparse matrix A and global vectors b and x are constructed as distributed objects whose entries lie across multiple processors. To define them, we use the *FE_CrsMatrix* and *FE_Vector* classes of the Epetra package. The data entries in *FE_CrsMatrix* are stored in a compressed row format. The *FE_CrsMatrix* and *FE_Vector* classes are specially designed for the finite element type procedures that allow assembling the element level contributions into the global system. The classes automatically handle the details about the data layout, the storage format, and the number of the ghost nodes and their corresponding location. The element level data can be added to the global linear system through three functions, *InsertGlobalValues*, *ReplaceGlobalValues* and *SumIntoGlobalValues*. The objects of the classes are created by passing the DOFs distribution described by the non-shared map. In our implementation, we use the same data distribution for A, x and b. For shared DOFs, the data are accumulated locally by keeping a copy of the data on the processor. The assembly procedure is completed by calling the *GlobalAssemble* function, which gathers any shared data into the non-overlapping partitioning defined by the non-shared map. *GlobalAssemble* is a collective method that reorganizes the data to be classified as off-processor, on-processor and on inter-processor boundaries. Accordingly, the communication patterns are established to transfer the data from a non-owner processor to the owner as specified by the non-shared map. The reorganized data structure is further used for the matrix-vector product in the linear solver. To communicate off-processor data of distributed matrix and vectors, *Epetra* provides the *Import* and *Export* classes. By specifying the source and the target map, the content of an object is transferred to a second object with a different distribution. The operations of import and export use the gather and scatter functions from MPI. *GlobalAssemble* function implicitly creates the *Import/Export* objects and uses them for off-processor communications.

2.6 Linear solver

In GALES, the linear system of equations $Ax = b$ is solved with the GMRES method from the Belos package of Trilinos [Bavier et al., 2012] (https://trilinos.github.io/belos.html). GMRES is a Krylov subspace-based iterative method that minimizes the residual of the linear system. The method is best known for its robustness and efficiency for solving a large and sparse system of equations. To generate the GMRES solver, we use the Belos factory class with which any available solver can be constructed by passing the solver name at the run time. GMRES solver requires to specify an initial guess of the solution, the subspaces number, restarts, iterations and the tolerance for the relative residual. For all numerical tests in this study we set these numbers as subspaces = 200, restarts = 5, iterations = 300 and residual tolerance = 10^{-6}. GMRES method often requires a good preconditioner which could effectively lower the condition number of the matrix and achieve convergence with reasonable computational effort. We use the incomplete lower-upper factorization (ILU) preconditioned version of the GMRES method. The preconditioner is generated from the Ifpack package.

3. Results

Strong scaling is widely used to check the ability of a numerical code to deliver results in less time when the amount of resources is increased. The parallel performance of the solvers is quantified by comparing the actual speedup with the ideal speedup for a given set of processors. The actual speedup and the ideal speedup are defined as

$$\text{Actual speedup} = \frac{t_r}{t_N} \qquad \text{r} < \text{N}$$

$$\text{Ideal speedup} = \frac{N}{r}$$

where t_r is the computational time taken by r processors, and t_N is the computational time taken by N processors, with $N > r$. Parallel efficiency is computed as the ratio of the actual speedup and the ideal speedup for a given number of processors:

$$\text{Efficiency} = \frac{t_r}{t_N}\frac{r}{N}$$

An ideal scalable software should result in a linear speedup. However, this is hardly achieved in real situations. For a given mesh the parallel efficiency almost always decreases as we increase the number of processors. In this work, we consider a parallel efficiency is satisfactory if it is greater than or equal to 0.6. In the following, we test the performance of GALES solvers. All tests are performed with 3D tetrahedral elements.

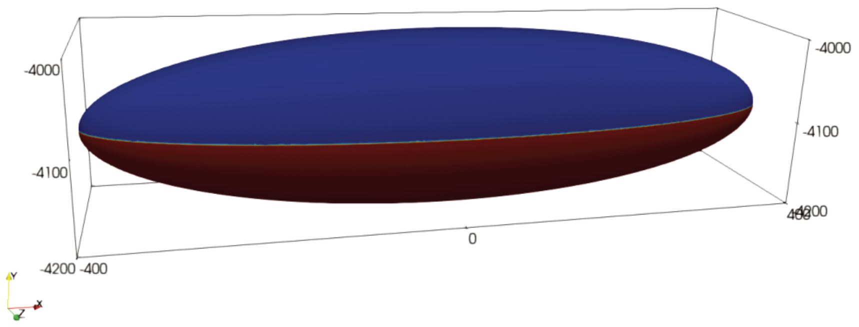

Figure 4 Setup for andesite-dacite magma mixing.

3.1 Multi-component miscible flow

As mentioned above, GALES was primarily developed to study geophysical problems. Specifically, to study the dynamics of underground multi-component magmas. As a matter of fact, most of the simulations carried out by GALES study magma dynamics in complex geometries. Therefore, this solver is the most frequently used one in GALES. To test the performance of the solver, we choose a typical two-magma mixing setup. A prolate ellipsoid domain with the longer axis set towards the x-axis is placed at 4 km depth. The lengths of the semi-axis in x, y, and z directions are 400, 100 and 100 respectively (see Figure 4). The lower half of the domain is filled with andesite magma, and the upper half is with dacite magma, each with its own properties deriving from their actual composition determined by ten major oxides plus the two major volatile species water and carbon dioxide. Although the andesitic melt is in general denser than the dacitic melt, the larger volatile content usually associated to deeper, gas rich magmas makes the two-phase gas-melt andesitic mixture lighter than the dacitic one. Initial and boundary conditions are set as in [Garg et al., 2019].
The simulation is run on 5 different meshes with progressively increasing size. The mesh models are listed in Table 1. For a two-fluid case, there are 6 unknowns per node: pressure, three components of velocity, temperature and mass fraction of one component. We run the test cases for 10 time iterations with a fixed time step of 0.01 s. At each time step, 3 non-linear iterations are performed.

Test	#nodes	#elements	#dofs
1	48478	266661	290868
2	175001	1004750	1050006
3	1951658	11782329	11709948
4	3341888	20317220	20051328
5	7802061	47581941	46812366

Table 1 Mesh models for andesite-dacite mixing.

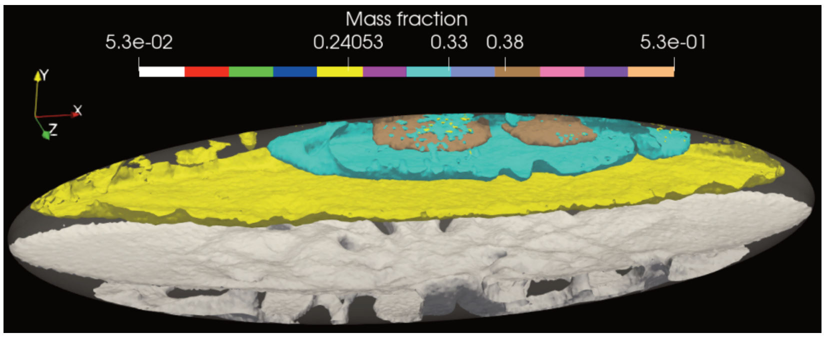

Figure 5 Andesite-dacite mixing results for mesh 3: segregated layers of magmas displayed as contours of mass fraction at time $t = 785$ s.

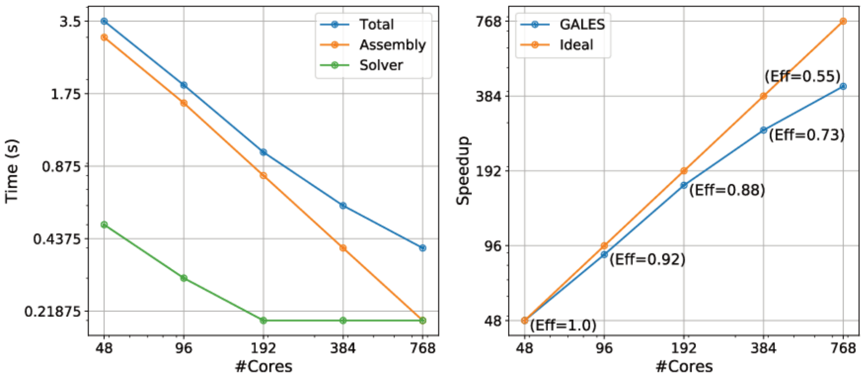

Figure 6 Andesite-dacite mixing: scaling results for mesh #1.

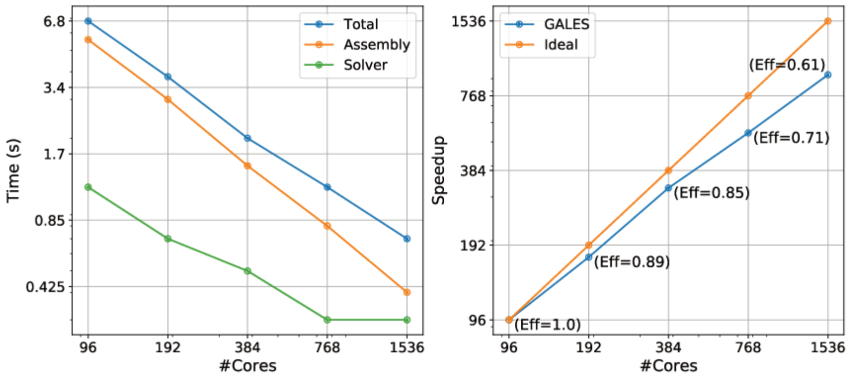

Figure 7 Andesite-dacite mixing: scaling results for mesh #2.

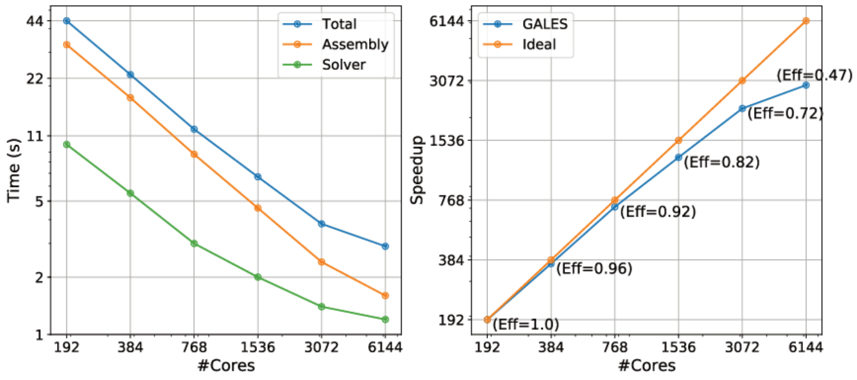

Figure 8 Andesite-dacite mixing: scaling results for mesh #3.

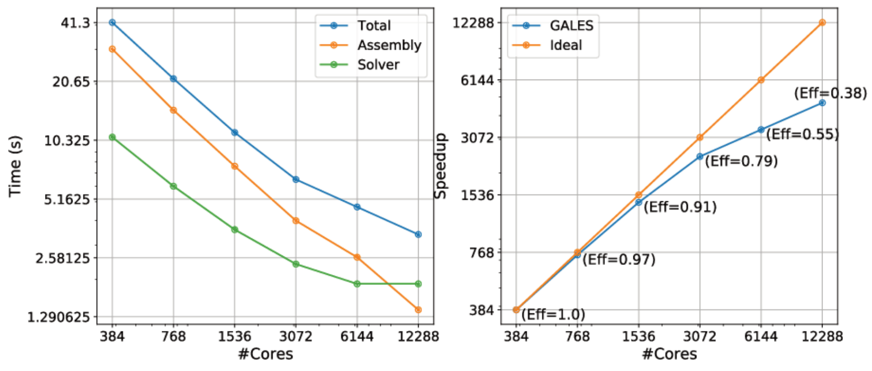

Figure 9 Andesite-dacite mixing: scaling results for mesh #4.

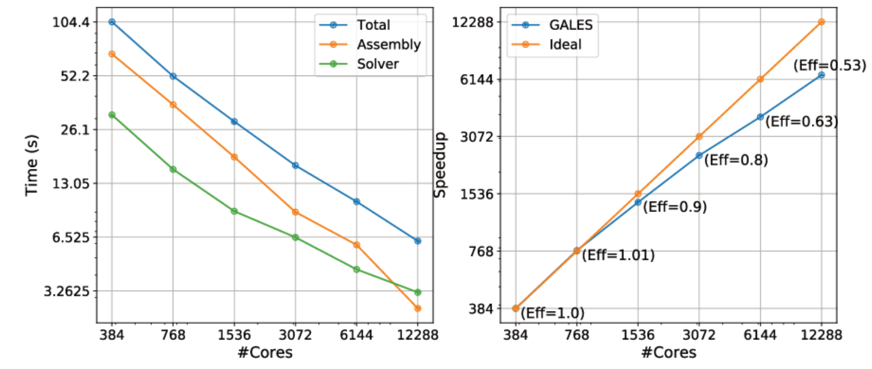

Figure 10 Andesite-dacite mixing: scaling results for mesh #5.

Figure 5 displays the contours of mass fraction for five different values at $t = 785$ s when a steady state is reached. We do not discuss the detailed results on the magma mixing here; instead, we present the parallel performance of the solver. Figures 6-10 show the strong scaling results for each test case. The left panel in the figures displays the time taken by the assembly procedure, the linear solver and the total time. We report here the average time taken for a non-linear iteration over 10 time steps with a variance of 0.056. We do not cover the initialization phase of the simulation in the analysis as it involves reading the mesh and constructing several class objects to be used for subsequent times. The right panel in the figures plots the speedup and the efficiency as a function of number of cores.

For increasing mesh size (Figures 6-10), we get a good scaling efficiency up to 3072 number of processors. For the mesh #1 (Figure 6), we can deduce that the code scales well up to 384 cores giving a parallel efficiency of 0.73. For more than 384 cores the efficiency starts decreasing. A similar conclusion can be drawn for other mesh models. For case #5 (Figure 10), the code scales very well up to 6144 cores with parallel efficiency of 0.63.

Parallel performance primarily depends on load balancing and inter-processor communication. Our linear solvers are distributed and use peer-to-peer communication to solve the system. Table 2 and Table 3 display the partition statistics for meshes #1 and #5, respectively, for different number of processors. Specifically, we display the average number of elements on a processor and the average percentage of inter-processor boundary nodes. These results are computed from the load balanced partitions generated by the Metis software. A load balanced partition reduces the overall execution time when we increase the resources. However, by increasing the number of processors, the percentage of boundary nodes lying at inter-processor boundaries also goes up, which means that the processors need to communicate more data. This substantially increases the communication time and results in a reduction of the parallel efficiency.

#cores	#elements	boundary nodes (in %)	scaling efficiency	computational time T_r (s)
48	5555	37	1.0	3.5
96	2777	46	0.92	1.9
192	1389	56	0.88	1.0
384	694	65	0.73	0.6
768	347	75	0.55	0.4

Table 2 Partitioning statistics for mesh #1 for andesite-dacite mixing: the second column displays the average number of elements on a core. The third column lists the average percentage of the number of nodes lying on inter-processor boundaries.

#cores	#elements	boundary nodes (in %)	scaling efficiency	computational time T_r (s)
384	123911	19	1.0	104.4
768	61955	23	1.0	51.8
1536	30977	30	0.91	27.9
3072	15488	35	0.80	16.4
6144	7744	42	0.63	10.3
12288	3872	50	0.53	6.2

Table 3 Partitioning statistics for mesh #5 for andesite-dacite mixing: the second column displays the average number of elements on a core. The third column lists the average percentage of the number of nodes lying on inter-processor boundaries.

Note that in case #1 for 768 cores, the average number of elements on a core becomes less than the total number of cores employed. Likewise, in case #5 for 12288 cores, there are 3872 elements per core. This is not an ideal situation for a parallel setting and therefore expected to degrade the performance. However, up to 6144 cores, the efficiency stays good.

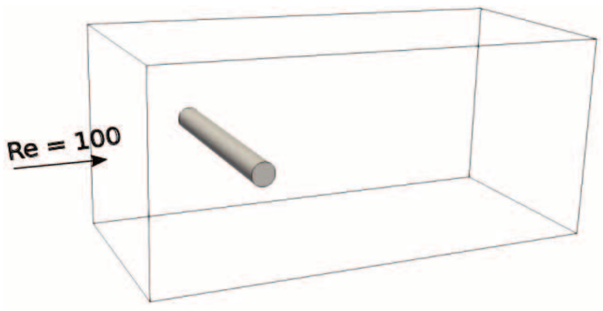

Figure 11 Flow past a fixed cylinder at Re 100: the problem set up.

3.2 Incompressible flow

For incompressible flow, we perform a scaling test on the flow past a fixed cylinder at Re = 100. The test case is one of the most common standard benchmarks for code validation in incompressible flow. The computational mesh is composed of 892108 nodes and 5339591 tetrahedral elements, giving rise to a total of 4460540 DOFs. Figure 11 displays the schematic sketch of the problem and Figure 12 the scaling results in terms of execution time and speedup. The results are averaged over 10 time steps and 3 non-linear iterations per time step.

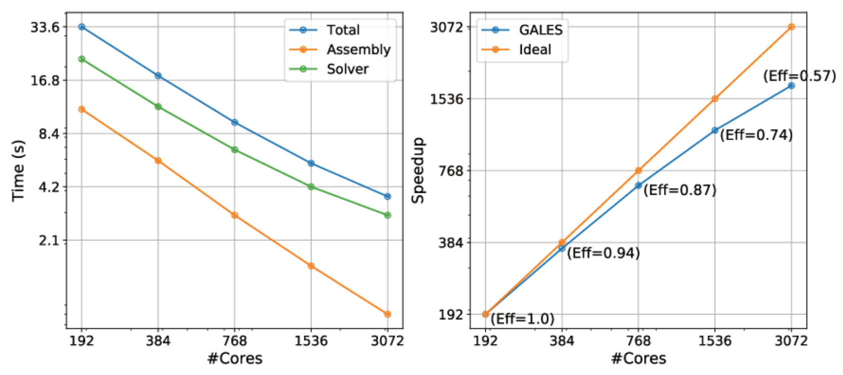

Figure 12 Flow past a fixed cylinder at Re 100: the scaling results.

We notice that for this problem the solver scales well up to 1536 cores for this mesh size. Again, in this case, one can see the performance degrades mainly when the average number of elements per core becomes smaller than the number of processors. A second factor that contributes to the lower performance is the increase in inter-processor boundary nodes (see Table 4).

#cores	#elements	boundary nodes (in %)	scaling efficiency	computational time T_r (s)
192	27810	27	1.0	33.6
384	13905	34	0.95	17.8
768	6952	42	0.87	9.7
1536	3476	50	0.74	5.6
3072	1738	59	0.58	3.7

Table 4 Partitioning statistics for flow past a fixed cylinder at Re 100: the second column displays the average number of elements on a core. The third column lists the average percentage of the number of nodes lying on inter-processor boundaries.

3.3 Compressible flow

For compressible flow, we chose the test case of Mach 3 flow over a flat plate. The flow features result in a shock and a boundary layer. The computational mesh is composed of 664983 nodes and 3964360 tetrahedral elements. The total number of unknowns to be solved is 3324915. Figure 13 displays the problem setup. The scaling results are presented in Figure 14. The results are averaged over 10 time steps and 5 non-linear iterations per time step.

For this case we get a good scaling efficiency of up to 3072 cores even if the average number of elements per core goes as low as 1290 and the percentage of boundary nodes increases to 62% (see Table 5). For 6144 cores, the scaling efficiency decreases to 0.43.

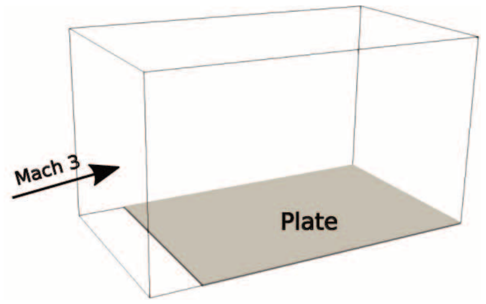

Figure 13 Flat plate at Mach 3: the problem set up.

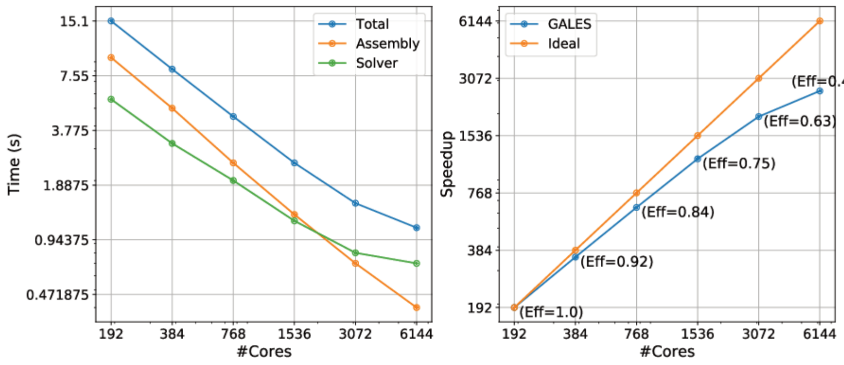

Figure 14 Flat plate at Mach 3: the scaling results.

#cores	#elements	boundary nodes (in %)	scaling efficiency	Computational time T_r (s)
192	20648	30	1.0	15.1
384	10323	37	0.98	8.2
768	5161	45	0.84	4.5
1536	2580	53	0.75	2.5
3072	1290	62	0.63	1.5
6144	645	71	0.43	1.1

Table 5 Partitioning statistics for *flat plate at Mach 3*: the second column displays the average number of elements on a core. The third column lists the average percentage of the number of nodes lying on inter-processor boundaries.

3.4 Elastostatics

For the elastostatic solver we perform the scaling test on a Mogi model problem. The Mogi model is often used to compute the surface deformation at active volcanoes as well as to invert the measured surface deformation and retrieve lumped parameters characterizing the source, such as the change in volume or change in pressure. The simulation domain consists of a cube with side length 100 km.

A magma chamber of radius 1 km is placed at 4 km depth which acts as a source pushing outwards with a force deriving from an applied 20 MPa overpressure (see Figure 15). Table 6 lists the three different meshes used to run this case. The scaling results are presented in Figures 16-18.

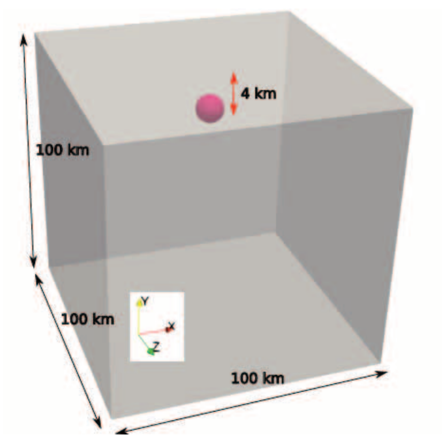

Figure 15 Problem setup for Mogi model.

Test	#nodes	#elements	#dofs
1	225802	1129778	677406
2	4033243	20649633	12099729
3	8406255	43121526	25218765

Table 6 Mesh models for Mogi test.

We notice that for all test runs the time taken by the GMRES solver is always more than the time taken to assemble the linear system. This is different from the case of the fluid solvers where GMRES was efficient to solve the problems and took less time in delivering results. This suggests that GMRES solver with ILU preconditioner is not an optimal choice for elastostatic problems. Since the finite element discretization of elastostatics equations results in symmetric positive definite system, the conjugate gradient method with Incomplete Cholesky Factorization will be a good candidate to be tested for this solver. In our current analysis we have not yet tested the conjugate gradient method and we plan to do that in our future work.

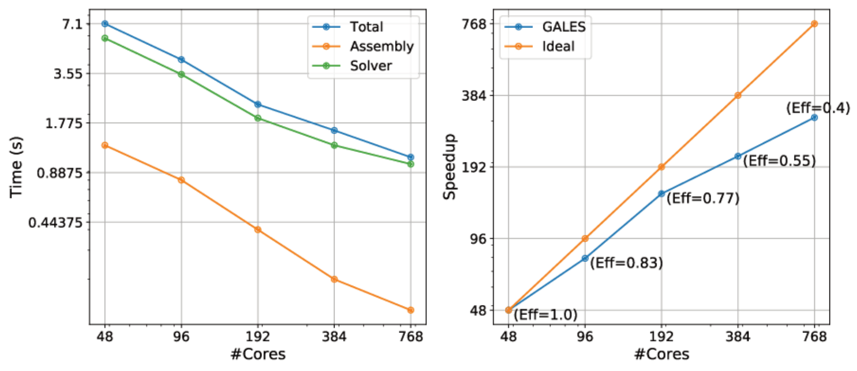

Figure 16 Mogi model: scaling results for mesh #1.

Figure 17 Mogi model: scaling results for mesh #2.

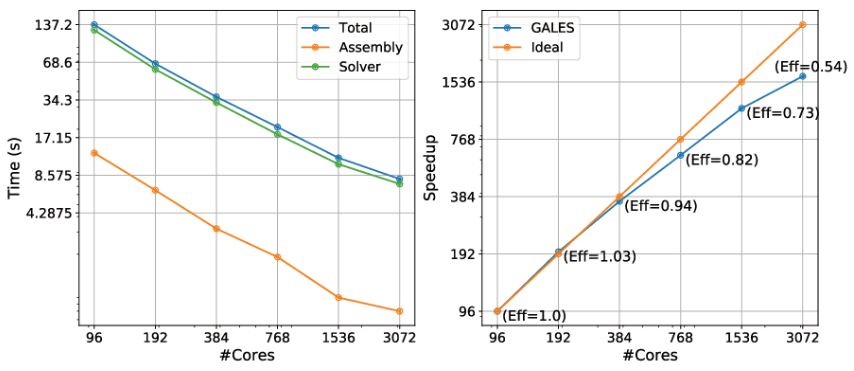

Figure 18 Mogi model: scaling results for mesh #3.

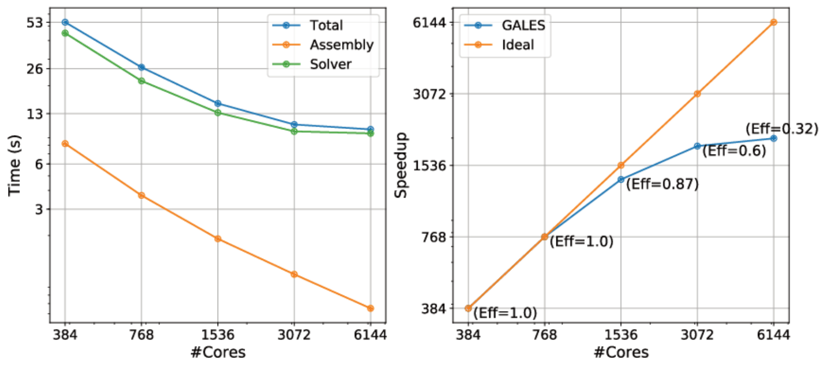

Table 7 displays the mesh statistics for mesh model #2. In general, we notice that the performance is lower in comparison to the fluid cases. With 3072 cores, the scaling efficiency already decreases even though the number of elements per processor is relatively high and the number of inter-processor boundary nodes is still reasonable.

#cores	#elements	boundary nodes (in %)	scaling efficiency	Computational time T_r (s)
96	215100	6	1.0	137.2
192	107550	9	1.0	66.9
384	53775	13	0.94	36.3
768	26887	18	0.82	20.8
1536	13443	25	0.73	11.8
3072	6123	36	0.54	8.0

Table 7 Partitioning statistics for mesh #2 Mogi model: the second column displays the average number of elements on a core. The third column lists the average percentage of the number of nodes lying on inter-processor boundaries.

3.5 Elastodynamics

For the elastodynamics solver, we use the same setup of the Mogi model with mesh number 3. We run it for 5 time iterations with a time step of 0.001 s. The non-linear iterations per time step are fixed to 3. Figure 19 displays the scaling results, which are averaged over all iterations. The results are similar to the ones we obtained in the elastostatic case.

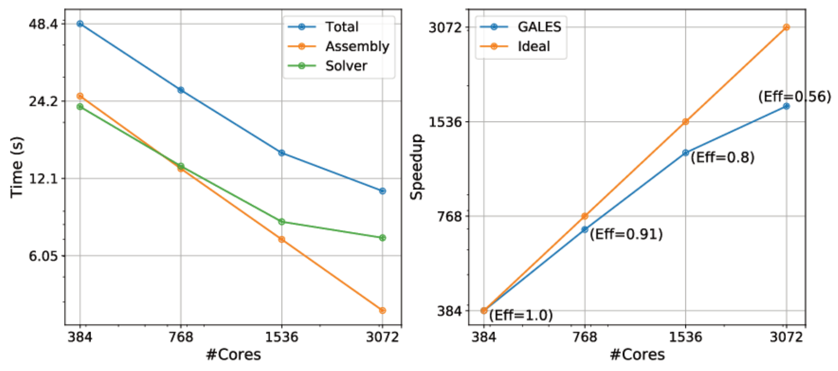

Figure 19 Mogi model: scaling results of elastodynamics case.

3.6 Fluid-Solid interaction

For fluid-solid interaction (FSI), we test the solver on two different problems. In the first problem, an elastic plate is placed in a channel. The fluid flow is incompressible at Re = 100. A schematic sketch of the setup and the scaling results are presented in Figures 20 and 21. The fluid mesh is made of 406813 nodes and 2364631 elements, and the solid mesh contains 51170 nodes and 239488 elements. The time reported in Figure 21 is the average time for one time iteration which includes multiple non-linear iterations of fluid and solid solvers done at the outer level. Furthermore, both fluid and solid solvers use multiple non-linear iterations at the inner level to achieve convergence.

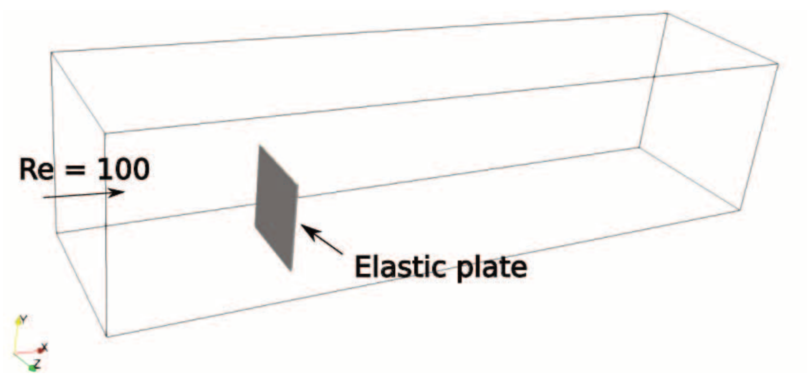

Figure 20 Plate in channel: problem setup.

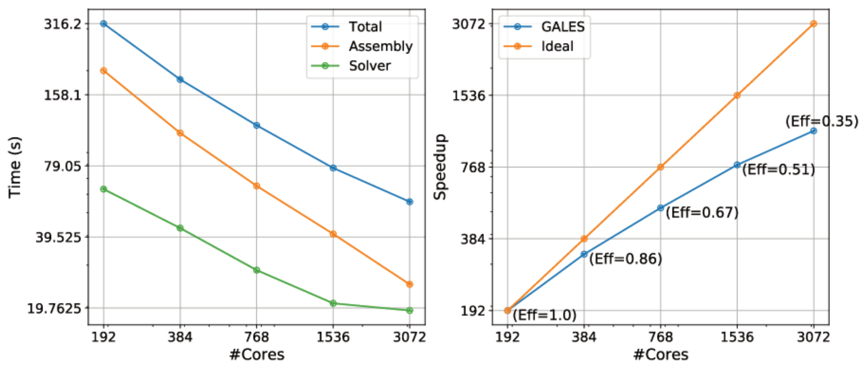

Figure 21 Plate in channel: scaling results.

#cores	#elements (fluid)	#elements (solid)	scaling efficiency	Computational time T_r (s)
192	12315	1248	1.0	316.2
384	6608	624	0.86	183.5
768	3304	312	0.67	117.3
1536	1652	156	0.51	77.4

Table 8 Partitioning statistics for plate in channel: the second and third columns displays the average number of elements on a core for fluid and solid domains, respectively.

Table 8 lists the mesh partitioning statistics in terms of both fluid and solid meshes. We see that for this case, the solver scales well up to 384 cores, giving a parallel efficiency of 0.86, which is considered good for an FSI solver. At 768 cores, the number of elements per core for the solid problem decreases to 312, which is even less than the total number of cores. Nevertheless, the efficiency of 0.67 is still good. This is because of the fluid solver, which still has a high number of elements per processor. Also, the above tests show that the fluid solvers are better in scalability than the solid solvers. The efficiency decreases to 0.51 when the number of elements per core for the fluid domain approaches the number of processors.
The second test case has the same setup as in the Mogi model problem (Figure 15). The only difference is that now the chamber is filled with andesite and dacite magmas separated by an interface and interacting with the surrounding rocks. Table 9 lists the mesh configurations for two different cases. The scaling results are displayed in Figures 22-23.

Test	#nodes (fluid)	#elements (fluid)	#nodes (solid)	#elements (solid)
1	20571	110862	31805	156769
2	276778	1624750	177836	912171

Table 9 Mesh sizes of different run cases for magma-rock interaction.

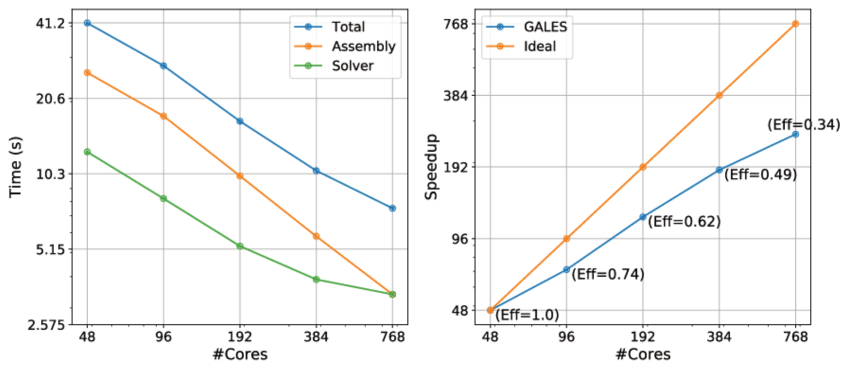

Figure 22 Magma-rock interaction: scaling results for mesh #1.

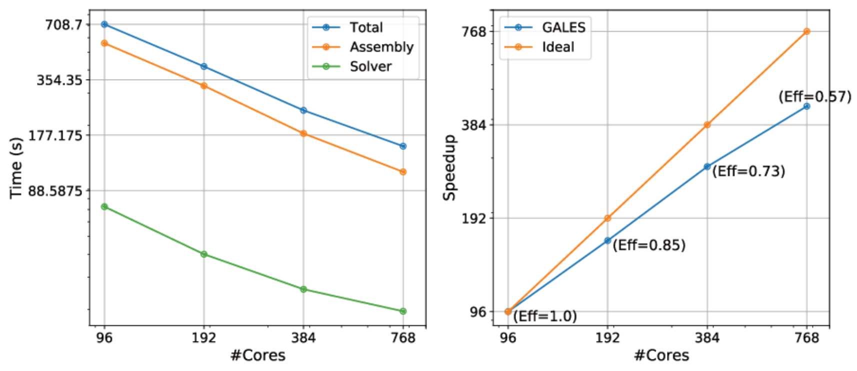

Figure 23 Magma-rock interaction: scaling results for mesh #2.

We notice that for mesh 1 with 192 cores, we get a parallel efficiency of 0.62. For this case, the satisfactory scaling limit (parallel efficiency ≥ 0.6) is 192. Similarly, for mesh 2, one can see that the scaling limit is 384 cores.

The FSI test cases show a little lower performance than individual fluid and solid solvers. This is in general true for any segregated FSI solver. The reason for that is the extra communication of data. Besides the peer-to-peer communication between different processors for individual fluid and solid linear solvers the extra communication comes from the boundary conditions at fluid solid interface ensuring coupling at each iteration step.

Within a time step, an FSI problem requires multiple non-linear iterations between fluid and solid solvers. At each non-linear iteration, the forces and the kinematic variables need to be at equilibrium at the fluid-solid interface. The coupling is achieved by imposing Dirichlet and Neumann boundary conditions at the fluid-solid interface. After solving the fluid problem, the forces are computed and applied at the fluid-solid interface to the solid problem. Similarly, after solving the solid problem, the velocities and deformation need to be transferred to the fluid and moving mesh problems, respectively. In the parallel environment, often the fluid and solid nodes that share the interface reside on different processors. Since coupling must be guaranteed at each non-linear iteration of a time step, the segregated solver inherently involves this extra communication cost. Such an additional communication cost is unavoidable and usually causes the lower performance of the code. A circumvent to this problem is to use a monolithic solver in which fluid and solid problems are solved simultaneously. This is currently under consideration for further developments.

Conclusions

In this report we present the results pertaining to strong scaling of the GALES code. After a description of the parallel implementation of the code, we test the performance of the fluid, the solid, and the FSI solvers for different problem configurations and mesh size. We demonstrate that GALES successfully solves problems containing several million 3D elements. We remark that in this study we test cases up to approximately 50 million unknowns. However, the code can be effectively used for even larger problems (up to 2 billion unknowns). A good speedup is observed for the fluid solvers, for which the performance mainly decreased when the number of elements on a core becomes less than the number of employed processors. We also show that an increase in the number of inter-processor boundary nodes negatively impacts the strong scaling performance. We notice that in case of solid problems, the GMRES solver with ILU preconditioner takes a much longer time than the assembly time. That results in a total longer time to solve the problems. Potential solutions will be explored in our future work. Finally, we test the performance of the FSI solver, which turns out to be lower than that for the fluid and solid solvers alone. Also in this case, we are identifying solutions and believe there is room for substantial improvements in code performance. Additional work is underway to study the bottlenecks in the code and analyze the parallel performance through advanced tools such as Extrae, PAPI and Paraver.

Acknowledgement

We acknowledge that the results of this research have been achieved using the PRACE Research Infrastructure resource MareNostrum based in Spain at Barcelona Supercomputing Centre (BSC).

References

Asanovic K. et al., (2006). *The landscape of parallel computing research: A view from berkeley*. Technical report, EECS Department, University of California, Berkeley.

Bavier E. et al., (2012). *Amesos2 and belos: Direct and iterative solvers for large sparse linear systems*. Scientific Programming, 20(3), 241-255.

Bagagli M. et al., (2017). *Signature of magmatic processes in strainmeter records at campi flegrei (Italy)*. GEOPHYSICAL RESEARCH LETTERS, 44, 718-725.

Blake G. et al., (2006). *Blue matter: Strong scaling of molecular dynamics on blue gene*. Computational Science-ICCS2006, Springer Berlin Heidelberg, pp. 846–854.

Boman G. et al., (2012). *The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: Partitioning, ordering, and coloring*. Scientific Programming, 20(2), 129-150.

Chung J. and Hulbert J.M., (1993). *A time integration algorithm for structural dynamics with improved numerical dissipation: The generalized-alpha method*. J. Appl. Mech., 60(2), 371-375.

Dowd K. and Severance C., (2010). *High Performance Computing*. Rice University.

Farhat C. et al., (1998). *Load and motion transfer algorithms for fluid/structure interaction problems with non-matching discrete interfaces: Momentum and energy conservation, optimal discretization and application to aeroelasticity*. CMAME, 157(1-2), 95-114.

Flanagan J. et al., (2020). *A review of a distributed high performance computing implementation*. Journal of Information Technology Case and Application Research, 22(3), 142-158.

Garg D. et al., (2018a). *Computation of compressible and incompressible flows with a space-time stabilized finite element method*. Computers and Mathematics with Applications, 75(12), 4272-4285.

Garg D. et al., (2018b). *Modeling free surface flows using stabilized finite element method*. Mathematical Problems in Engineering, 2018(6154251), 1–9.

Garg D. et al., (2019). *Long-lived compositional heterogeneities in magma chambers, and implications for volcanic hazard*. Scientific Reports, 9(3321), 1-13.

Geuzaine C. and Remacle J.F., (2009). *Gmsh: a three-dimensional finite element mesh generator with built-in pre-and post-processing facilities*. International Journal for Numerical Methods in Engineering, 79(11), 1309-1331.

Gottlieb A. et al., (1989). *Highly parallel computing*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA.

Hauke G. and Hughes T.J.R., (1998). *A comparative study of different sets of variables for solving compressible and incompressible flows*. CMAME, 153(1-2), 1-44.

Hughes T.J.R., (1987). *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Prentice Hall Inc., Engelwood Cliffs, New Jersey 07632.

Karypis G. and Kumar V., (1999). *A fast and high quality multilevel scheme for partitioning irregular graphs*. SIAM Journal on Scientific Computing, 20(1), 359-392.

Longo A. et al., (2006). *Numerical simulation of convection and mixing in magma chambers* *replenished with co2-rich magma*. GEOPHYSICAL RESEARCH LETTERS, 33, 1-6:L21305.

Longo A. et al., (2012a). *A finite element galerkin/least squares method for computation of multicomponent compressible incompressible flows*. Computers and fluids, 67, 57-71.

Longo A. et al., (2012b). *Magma convection and mixing dynamics as a source of ultra-long-period oscillations*. Bull. Volcanology, 74, 873-880.

Masud A. and Hughes T.J.R., (1997). *A space-time galerkin/least-squares finite element formulation of the navier-stokes equations for moving domain problems*. CMAME, 146(1-2), 91–126.

Muller A. et al., (2015). *Strong scaling for numerical weather prediction at petascale with the atmospheric model numa*. The International Journal of High Performance Computing Applications, 33(2), 411-426.

Papale P. et al., (2017). *Pressure evolution in shallow magma chambers upon buoyancy-driven replenishment*. Geochemistry, Geophysics, Geosystems, 18(3), 1214-1224.

Schafer M. et al., (2006). *An implicit partitioned method for the numerical simulation of fluid-structure interaction*. Lecture Notes in Computational Science and Engineering, Springer, Berlin, Heidelberg.

Schloegel K. et al., (2002). *Parallel static and dynamic multi-constraint graph partitioning*. Concurrency and Computation: Practice and Experience, 14(3), 219-240.

Simo J.C. and Hughes T.J.R., (1998). *Computational inelasticity*. Springer, NewYork.

Souli M. et al., (2001). *Arbitrary lagrangian eulerian and free surface methods in fluid mechanics*. CMAME, 191, 451-466.

Tezduyar T.E., (1992). *Stabilized finite element formulations for incompressible flow computations*. Advances in applied mechanics, 28, 1-44.

Vollaire C. et al., (1998). *Parallel computing for the finite element method*. The European Physical Journal Applied Physics, 1, 305-314.

Coordinamento editoriale e impaginazione
Francesca DI STEFANO, Rossella CELI
Istituto Nazionale di Geofisica e Vulcanologia

Progetto grafico e impaginazione
Barbara ANGIONI
Istituto Nazionale di Geofisica e Vulcanologia

ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA