Parallelisation and numerical performance of a 3D model for coupled deformation, fluid flow and heat transfer in sedimentary basins

J. B. Haga[*]
[*]Scientific Computing Department
Simula Research Laboratory
e–mail: post@simula.no

A. M. Bruaset[*†], **X. Cai**[*†], **H. P. Langtangen**[*†], **H. Osnes**[‡*] **and J. Skogseid**[§]
[†]Department of Informatics [‡]Department of Mathematics [§]Norsk Hydro ASA
University of Oslo University of Oslo

Summary In this paper, we present some parallel performance results for a 3D simulator of coupled deformation, fluid flow and heat transfer in sedimentary basins. The model parameters are derived from an industry simulator, with realistic material properties and complex irregular grids of up to 1.5 million nodes with 7.3 million degrees of freedom. We have performed parallelisation on the linear algebra level using the ML algebraic multigrid preconditioner with iterative methods in the Diffpack finite element framework. Implementation and speedup results are presented.

Introduction

Sedimentary basins are formed over millions of years, as sediments are deposited in layers of different composition. The current layout of the basin thus shows a history of geological events during its formation. Understanding this history is of general interest to geoscientists, and the underlying physical processes are of great importance to understand the maturation and primary migration of hydrocarbons into oil and gas reservoirs.

Analysis and mathematical modelling of physical processes taking place in sedimentary basins are complicated tasks. Due to the complexity of the governing partial differential equations (PDEs) and the initial and boundary conditions, analytical solutions of realistic problems generally do not exist. Therefore, numerical solution techniques must be applied. The high spatial variability of properties such as permeability and porosity in typical basins means that high resolution of the computational grids is required. The fact that the PDEs are coupled leads to further computational challenges. In addition, the systems must be solved iteratively for the nonlinear couplings. In [7], such a coupled model was proposed and applied to a case of horizontal magma intrusions in a sedimentary succession, using finite element methods on a two-dimensional domain. We have reimplemented and extended this model to cover three dimensional (3D) problems. The number of unknowns required in real 3D cases is very large, which means that parallelisation of the code is necessary. Here we present a method for parallelisation of the model.

We have adopted a minimally invasive method of parallelisation at the linear algebra level, which is detailed in later sections. We present scalability results for the parallel solver, preconditioned with an algebraic multigrid (AMG) preconditioner. These results show that this approach provides good speedup for the range of tested clusters.

The research presented in this paper has been financially supported by Norsk Hydro.

The geomechanical model

Based on certain fundamental physical principles, as well as the continuum hypothesis, a system of three coupled PDEs is derived: One for the fluid (pore) pressure p, one for the displacement field $\boldsymbol{u}$ of the porous medium, and one for the temperature T. Additional mathematical equations, e.g., constitutive relations and boundary conditions, are introduced to fully specify the problem. The governing equations are as follows.

Equation for the fluid pressure

Using the principle of conservation of mass for the fluid and solid phases, and following Darcy's law in which the effect of thermal expansion of the fluid phase is included, we obtain

$$S\frac{\partial p}{\partial t} = \boldsymbol{\nabla}\cdot(\Lambda\boldsymbol{\nabla}p) - \boldsymbol{\nabla}\cdot(\Lambda\rho_{\mathrm{f}}(1-\beta_{\mathrm{f}}(T-T_0))\boldsymbol{g}). \tag{1}$$

Here S is the storage coefficient, Λ is the mobility of the flow, and ρ_{f} is the fluid density. In the temperature term, β_{f} is the thermal expansion coefficient of the fluid, T_0 is a reference temperature, and $\boldsymbol{g}$ is the acceleration of gravity. It should be remarked that the fluid flow in deformable porous media is considered as a quasi-static Biot system, see for example [11].

Equation for the temperature

Based on the principle of conservation of energy, or the first law of thermodynamics, the following advection-diffusion equation for the temperature field is introduced:

$$C\frac{\partial T}{\partial t} + \rho_{\mathrm{f}}C_{\mathrm{f}}\boldsymbol{v}_{\mathrm{D}}\cdot\boldsymbol{\nabla}T = \boldsymbol{\nabla}\cdot(\kappa\boldsymbol{\nabla}T), \tag{2}$$

where $C = (\phi\rho_{\mathrm{f}}C_{\mathrm{f}} + (1-\phi)\rho_{\mathrm{s}}C_{\mathrm{s}})$ is the bulk heat capacity, C_{f} and C_{s} are the specific heats for the fluid and solid phase, ρ_{f} and ρ_{s} are fluid and solid densities, ϕ is the porosity, and κ is the thermal conductivity. The Darcy velocity $\boldsymbol{v}_{\mathrm{D}}$ contains a pressure term (a driving force, coupling it to (1)) as well as a temperature term (buoyancy, which makes it nonlinear),

$$\boldsymbol{v}_{\mathrm{D}} = \phi\boldsymbol{v}_{\mathrm{f}} = -\Lambda(\boldsymbol{\nabla}p - \rho_{\mathrm{f}}(1-\beta_{\mathrm{f}}(T-T_0))\boldsymbol{g}). \tag{3}$$

Equation for the deformation

After neglecting inertia (or acceleration) terms, the principle of conservation of momentum for the sedimentary basin leads to the equilibrium equation

$$\boldsymbol{\nabla}\cdot\boldsymbol{\sigma} + \rho\boldsymbol{g} = 0, \tag{4}$$

where $\boldsymbol{\sigma}$ is the total stress tensor. While the deformations of the sediments are to a large degree plastic, we assume that on a short time scale with small deformations a more computationally efficient elastic model is sufficiently accurate. Applying the assumption of isotropic linear elastic behaviour of the porous medium, the stress components are related to the displacement field through Hooke's law with a linear thermoelastic term,

$$\boldsymbol{\sigma} = (\lambda\boldsymbol{\nabla}\cdot\boldsymbol{u} - \alpha p - \beta_{\mathrm{s}}(3\lambda+2\mu)(T-T_0))\,\mathbf{I} + 2\mu\boldsymbol{\epsilon}. \tag{5}$$

Here, λ and μ are the Lamé material constants, and α is the Biot factor (approaching unity for fully saturated flows). The elastic deformation tensor $\boldsymbol{\epsilon}$ is the symmetricised gradient of the displacement field, $\boldsymbol{\epsilon} = (\boldsymbol{\nabla}\boldsymbol{u} + (\boldsymbol{\nabla}\boldsymbol{u})^{\mathrm{T}})/2$.

Implementation

The present model was originally implemented by Kjeldstad et al. [7] for two-dimensional problems. To incorporate 3D calculations, we did a new implementation of the simulator with emphasis on generality and extensibility. In parallel with the reimplementation of the solver, we have developed software to import models from Petromod, which is one of the leading basin simulation software packages in the oil and gas industry [10]. This has allowed us to use realistic geometries, initial conditions and material parameters for our models, and to some extent also to verify our results.

The serial implementation of the solver uses a standard finite element method for the coupled equations in the Diffpack framework. Details on the handling of complicated initial and boundary conditions are found in [7] and [9].

The coupled equations generate a system of equations with five degrees of freedom per node (p, T and $\boldsymbol{u}$), and it is not uncommon to do simulations on a grid of several million nodes. At these sizes, just the memory accesses associated with one matrix-vector product — the core of any iterative solver — can take a second or more on a single computer.[1] Thus, in order to achieve acceptable performance, parallelisation is required.

Parallelising the solver

Parallelisation is commonly classified into three levels. At the lowest level, instruction level parallelism can be exploited to allow small segments of the program to run in parallel. This level of parallelism is performed routinely and transparently on modern CPUs, but such fine-grained parallelism requires communication for each loop iteration and thus is only efficient in the case of very closely coupled computing units (e.g., different algebraic units on the same processor core).

At the highest level, exemplified by domain decomposition methods [12], the physical domain is split into several subdomains. The system of equations is solved repeatedly but independently on each subdomain. The solutions are then stitched together at the edges by suitable boundary conditions, perhaps aided by a coarse-grid solver to speed up convergence of the long-range elements in the solution. Communication is only required after the subdomains' algebraic systems have been solved, in order to exchange boundary information.

We have chosen the middle ground of parallelising the linear algebra operations used in the solver for the system of algebraic equations. While this approach requires communication for each iteration of the (iterative) linear solver, it has the advantage of being very simple to bolt onto an existing non-parallel implementation; see for example [3], which details the following approach.

The parallel linear algebra implementation in Diffpack [4] allows the user to treat the parallelisation as a black box in simple cases. Ignoring a few lines of initialisation and user input changes, the basic parallelisation is performed in three steps:

1. Introduce a partitioner for the grid. The job of the partitioner is to inspect the connectivity information between grid cells, and partition the work load between the computing nodes to minimise the interprocess communication while roughly balancing the amount of work.

[1] Half a second for 1 million nodes at 6.4GB/s memory bandwidth, assuming an average of 70 entries per matrix row.

2. Chop the global grid into smaller local subgrids, for use in processor-local element assembly.

3. Set up the iterative linear solver as well as the nonlinear solver to use the communication patterns established by the partitioner to exchange values for the boundary points when necessary during the solving process.

This recipe will set up parallel communication and utilise it in the linear and nonlinear solvers, all in about 10 lines of code. However, care must be taken whenever the linear system is accessed directly, as it never exists in full on any one processor. For example, code to read initial conditions and material parameters from a file needs to be made aware of how the nodes and elements are distributed across the processors. This increases the complexity of the parallel solver, but proper use of object-oriented techniques allows this complexity to be separated from the core logic of the simulator.

Parallel preconditioning

For general large sparse systems of equations, preconditioning is vital to the performance of the solvers. Typical discretisations arising from elliptical problems generate matrices with condition number proportional to h^{-2}, where h denotes the average mesh resolution. For Krylov-type iterative solvers this means that the number of iterations to reach a given accuracy is doubled when the resolution is doubled (see for example [1]). To counter this unfortunate situation, it is better to solve instead the preconditioned system

$$\mathbf{CA}\boldsymbol{x} = \mathbf{C}\boldsymbol{b}, \tag{6}$$

where $\mathbf{C}$ mimics the action of $\mathbf{A}^{-1}$ in such a way that the system (6) is better conditioned than the original problem. One approach to this is to look for preconditioners that make the condition number of $\mathbf{CA}$ smaller and less sensitive to the mesh resolution than what is achieved for the operator $\mathbf{A}$ alone.

While it is in principle possible to represent the preconditioner as a matrix $\mathbf{C}$, its action on a vector $\boldsymbol{v}$ is usually implemented either through solving a subsidiary system $\mathbf{M}\boldsymbol{x} = \boldsymbol{v}$ (hence $\mathbf{C} = \mathbf{M}^{-1}$), or procedurally. Either way, the operation must be of complexity no higher than $\mathcal{O}(N)$ to be worthwhile. This rules out the trivial option of parallelising the $\mathbf{C}\boldsymbol{v}$ product in the same way as $\mathbf{A}\boldsymbol{v}$. In fact many standard preconditioners such as Gauss-Seidel and the ILU family of incomplete factorisation methods are known to perform badly in their default modes of operation, and approximate variants are preferred in the parallel case. Finding a good preconditioner is often the main obstacle for parallelisation of serial solvers.

Algebraic multigrid

Multigrid methods are based on the profound observation that even simple iterative methods are very efficient at removing the high-frequency components of the error. Thus, while they may be inefficient or even ineffective as *solvers*, they are very good *smoothers* of the error. The multigrid approach offers a method for transposing all error components into high-frequency ones.

In the original multigrid formulation, known as geometric multigrid (GMG), this transposition is achieved by formulating the problem on a series of grids at successively coarser resolution. The low-frequency components on a fine grid become high-frequency components on a coarser

mesh. By iterating between smoothing on the different grid levels and exact solution at the coarsest level, the solution is reached on the fine grid. A survey of parallel GMG methods is found in [6].

Although conceptually simple, this approach has the drawback of requiring meshes at different resolutions. For general unstructured grids, this is impractical to generate. Algebraic multigrid (AMG), in contrast, does not require grid information at all. The transposition is performed using only the information in the linear system itself, in particular the connectivity information between nodes that is inherent in the coefficient matrix, in a process called *aggregation* [14]. This method is therefore suitable for general meshes and geometries. While the aggregation is usually competitive with or even surpasses the efficiency of the GMG grid hierarchy, it must be generated for each $\mathbf{A}$. Aggregation time is usually of the same magnitude as solving the system. Parallel approaches to the AMG methods are surveyed in [14].

Interfacing Diffpack with the ML parallel AMG preconditioner

The brief introduction above notwithstanding, we can for the present purposes consider AMG as a a black-box preconditioner with promising parallel scalability. We interfaced our existing Diffpack solver with ML [5], which is a mature, open-source AMG solver library suitable for parallel computations.

ML requires the implementation of two interface methods, `getrow` (retrieve a specified row from $\mathbf{A}$) and `matvec` (multiply a vector by $\mathbf{A}$), and also a method `comm` (exchange values between processors) for parallel runs. Diffpack requires a method `apply` (apply the operator $\mathbf{C}$ on a vector). In the non-parallel case the semantics of these methods are clear and the implementation of the glue layer is straightforward. In the parallel case, however, we must be careful so that the two views of local–global matrices and mappings match. In particular, Diffpack is designed to handle an arbitrary amount of overlap between processors, while ML expects every matrix entry to be the sole responsibility of exactly one processor.

In order to achieve this, we modified the standard Diffpack grid partitioner to partition the nodes of the grid instead of the elements, and then to include an element in a subdomain if at least one of its nodes has been partitioned into this subdomain. In other words, the subdomains are partitioned to have a single layer of shared elements on the boundaries. This guarantees that each interior point belongs to just one subdomain, and that each global point is an interior point (a nodal overlap of exactly one).

With this guarantee, the consistent view that ML requires is achieved by just hiding from ML those rows of the Diffpack matrix that correspond to interdomain boundaries (i.e., ghost nodes).

Results

In the following, we first compare the convergence of the algebraic multigrid preconditioner with a few standard preconditioners on the coupled system given in equations (1)–(5). We then evaluate the parallel performance of the AMG preconditioner on multiple processors.

A note on terminology: By *speedup* we refer to the decrease in runtime when running on P processors, compared to running the same workload on one processor. If $T(P)$ is the measured runtime on P processors, the speedup is given as $S(P) = T(1)/T(P)$. The *efficiency* of a given speedup is the fraction of perfect (linear) speedup, $E(P) = S(P)/P$; thus, in a sense, how much effective work each processor contributes to the total.

The *order* of an iterative solver is a measure of how the number of iterations grows with the

problem size. Unpreconditioned Krylov solvers on elliptical problems typically converge[2] in $\mathcal{O}(N^{1/d})$ iterations where N is the total number of nodes in d spatial dimensions. An *order optimal* preconditioner reduces this to $\mathcal{O}(1)$ — the number of iterations is independent of problem size.

The test cases

We have tested the preconditioned BiCGStab[3] solver on a number of test cases of increasing complexity:

SSC[4] A grid of 4x4x4 elements (5x5x5 nodes, 625 unknowns) with constant anisotropic material parameters.

LSC Similar to SSC, with resolution increased to 80x80x80 nodes (2.5M unknowns).

SUC An unstructured grid from a realistic basin model with 170k nodes (850k unknowns) and the same constant material parameters as SSC/LSC.

LUC Similar to SUC, with three times the resolution in x and y directions; 1.5M nodes (7.3M unknowns).

SUV The same grid as SUC, but with realistic, variable material parameters. Some of these parameters are quite jumpy, the permeability may for example change 10 orders of magnitude from one layer of rock to the next.

LUV The LUC grid with the same material parameters as SUV.

In all cases, the convergence criterion for the solver was the reduction of the residual by a factor of 10^{-8}.

While we have included a series of simpler test cases to build some understanding of the behaviour of the various preconditioners under different conditions, our purpose is ultimately to solve systems of the same nature as SUV and LUV; i.e., with realistic material parameters on large unstructured grids.

Hardware

All memory-intensive tests and most non-parallel tests were performed on a 2x2 core[5] Xeon EM64T with 36GB of memory, while all runtime measurements and most parallel tests were performed on a 20+ node cluster of 2x1 core 1.3GHz Itanium-2 IA64 processors with 4GB of memory. The cluster is connected by switched gigabit ethernet. In the cluster tests, most timing results were collected with both cores of each node occupied with similar workloads to keep the memory subsystem load consistent.[6]

Some of the parallel runs were either too big to fit in memory on the given number of cluster nodes, or utilised more than one process per processor. In these cases only the iteration count and not the runtime is reported.

[2] I.e., reduce the residual by a fixed factor ϵ.

[3] Stabilised bi-conjugate gradients, suitable for nonsymmetric matrices [13].

[4] Abbreviation key: **S**mall/**L**arge, **S**tructured/**U**nstructured, **C**onstant/**V**ariable.

[5] That is, two physical processors with two cores each.

[6] The two exceptions were the 4- and 6-processor LUV tests. We found the performance difference to be quite small on this particular architecture, within $\pm 3\%$ when using both cores on one node compared to using a single core on two nodes.

	Iterations					
Precond.	SSC	LSC	SUC	LUC	SUV	LUV
None	>5000	—	—	—	—	—
Jacobi	18	415	1372	2647	487	1029
ILU	5	50	52	300	>5000	>5000
AMG	5	8	40	51	47	42

Table 1: Number of iterations to reduce the residual by 10^{-8} for the different preconditioners. "—" indicates missing measurements, while ">5000" means that the problem did not converge in 5000 iterations.

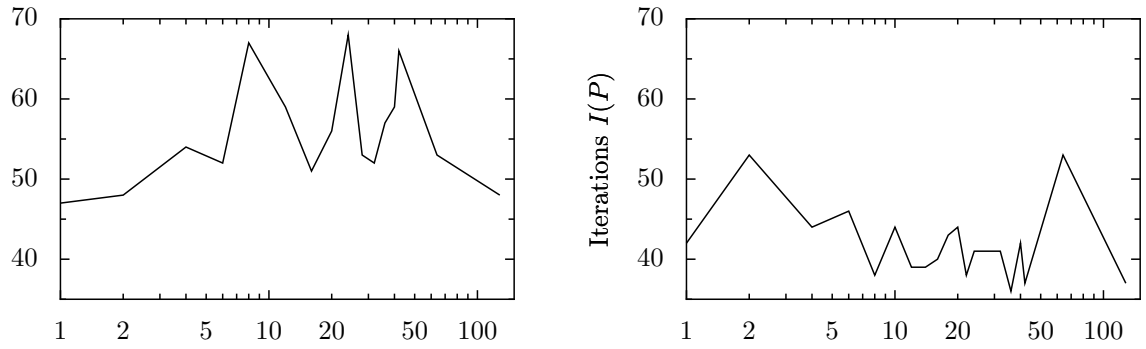

Figure 1: The number of BiCGStab iterations using AMG preconditioning for the SUV (left) and LUV (right) test cases running on multiple processors. The number jumps around somewhat, which is expected when using uncoupled aggregation, but no growth trend is evident.

Serial performance

We tested the coupled system of equations (1)–(5) without preconditioning and with the Jacobi, ILU and AMG preconditioners. On this system, the unpreconditioned solver failed to converge in even the simplest test case (table 1). The Jacobi preconditioner showed dramatically reduced performance on the unstructured grids, although (surprisingly) less so with variable material parameters. The ILU preconditioner showed better performance than Jacobi on all the tests with constant material parameters, but on the unstructured grids the iteration count increased dramatically with the grid size. ILU failed to converge at all with variable material parameters. The only tested preconditioner that performed satisfactorily, with reasonably low number of iterations and low order in all tests, was the AMG preconditioner.

Parallel performance

We show the parallel tests only for the SUV and LUV test cases, because we are primarily interested in real-world performance. Due to high memory requirements, we were not able to test the largest of these on less than 4 processors, and thus speedup and efficiency numbers are not directly available for the LUV case. This complication is handled in the next section.

The number of iterations for the AMG preconditioner shows that the preconditioner performance is acceptable also in the parallel case (figure 1). The aggregation strategy was uncoupled, i.e., aggregates are formed independently on each subdomain. This is the simplest strategy and does not require communication in the aggregation phase, but leads to suboptimal aggregation. Even so, performance loss (measured in number of iterations) seems to be bounded to within 50% or so, with no apparent trend. A number of alternative parallel aggregation strategies are discussed in [14] and supported in ML, but we were unable to get these to work satisfactorily in the time frame of this article.

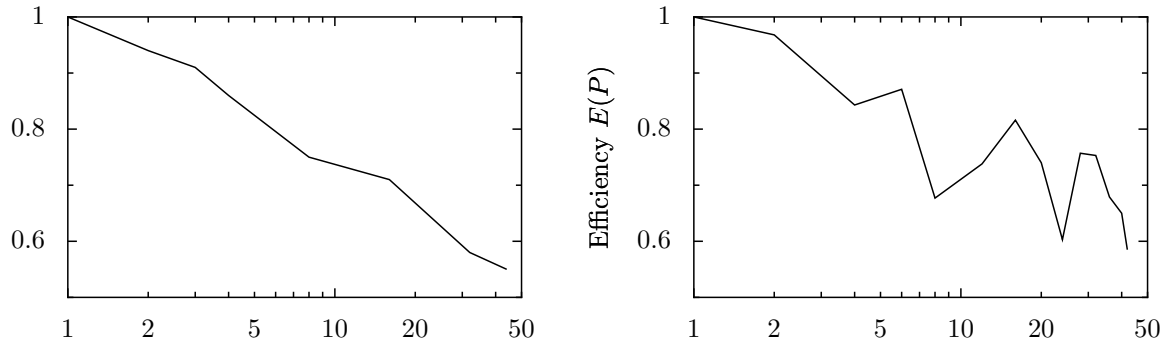

Figure 2: The efficiency results for the SUV case, using the Jacobi (left) and AMG (right) preconditioners, show a similar overall trend. The AMG efficiency is modulated by the number of iterations (as shown in figure 1).

The Jacobi and AMG preconditioners were the only preconditioners to converge in all the serial tests. Jacobi is known to parallelise well, and we compared its performance to that of AMG for the SUV case. Timing results for up to 42 processors show that the performance profile for the two preconditioners are similar when we take into account the varying number of iterations of the AMG preconditioner (figure 2). Since the AMG preconditioner was about 5 times faster than Jacobi in the serial SUV test and 10 times faster in the serial LUV test, we consider only AMG in the following.

For the efficiency numbers, we measured the time for one full iteration of the nonlinear solver: element matrix assembly, aggregation and setup, and linear solve. Together, these account for most of the time spent in the simulator, thus we expect this to be a realistic estimate of the speedup of the entire simulator for long runs.[7] The maximum speedup for the AMG-preconditioned SUV case was 26, running on 40 processors. As we shall see, an estimated speedup of nearly 41 was achieved for the LUV case, running on 42 processors (table 3).

Performance modelling

Measuring the speedup and efficiency of the LUV case directly was not possible, since the high memory requirements[8] of this case meant that we were unable to run it on less than 4 processors. For this reason, and also because we are interested in what performance to expect when running on more processors, a model for the parallel performance is desirable.

In [3], a simple performance model for the parallel computation is given. For a problem with a total of N grid nodes in three dimensions, there are three main costs:

1. The solution of the local problem, with complexity of order $\mathcal{O}(N/P)$,
2. exchange of interdomain boundary values, of order $\mathcal{O}((N/P)^{2/3})$ in 3D, and
3. collective communication, for e.g. inner products, of order $\mathcal{O}(\log P)$ independently of problem size.

We also need to take into account the variability which is due to the AMG preconditioner not converging equally fast for different numbers of processors. If we assume that the setup

[7] Time spent preparing and saving results is the remaining notable contributor, although for very short runs setup time may be significant.

[8] A bit more than 12GB.

	a	b	c	d
SUV	0.248	$6.65 \cdot 10^{-4}$	$2.12 \cdot 10^{-3}$	0.0999
LUV	0.197	$8.01 \cdot 10^{-4}$	$6.32 \cdot 10^{-3}$	—

Table 2: The parameters of (7), found by least squares regression for the two test cases. d was not estimated in the LUV case, instead the SUV value was used. While a and b are of similar magnitudes in the two cases, c (which is associated with the cost of exchange of boundary information) is tripled in the large case. We have not investigated whether this indicates a flaw in the model or just less than perfect measurements and parameter fitting.

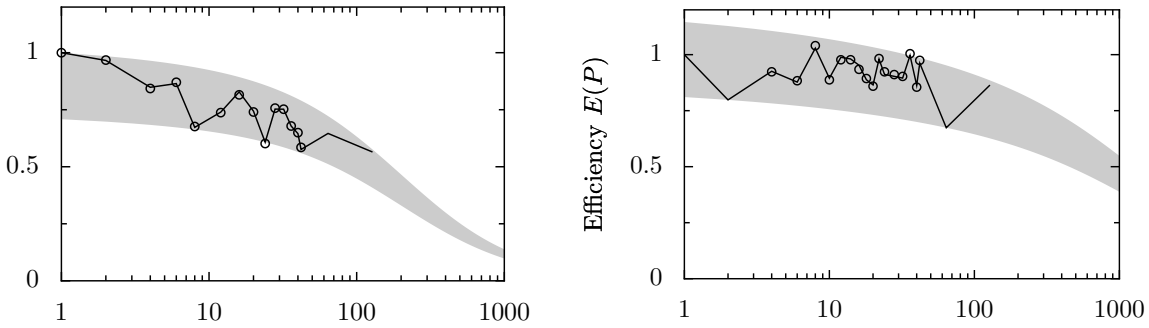

Figure 3: Measured (circles) and modelled (solid line) parallel efficiency of the SUV (left) and LUV (right) cases. The shaded areas in the graphs are the areas between the curves created by inserting the observed minimum and maximum of $I(P)$ into equation (7). Because of the variation in the number of iterations, superlinear speedup is seen for LUV at $P = 8$ and $P = 36$.

phase and the iteration phase of the solver have similar communication patterns, this leads to an equation for the runtime of the solver,

$$T(P) = (1 + aI(P)) \left(b\frac{N}{P} + c\frac{N^{2/3}}{P^{2/3}} + d \log P \right), \tag{7}$$

where $I(P)$ is the number of iterations required for P processors. The coefficients can be found by a least squares regression.

For both the SUV and the LUV cases, we followed a similar routine: First we used about half of the measured runtimes (up to $P = 16$ for SUV, $P = 20$ for LUV; see table 3) and estimated the parameters of (7) from these. We then compared the prediction with the measured values up to $P = 42$, to evaluate the predictive power of the model. The error at $P = 42$ was 4% for the SUV case, 2% for the LUV case.[9] We then used all data points to create a final estimate of the parameters (table 2), which was used to deduce the missing single-processor runtime (in the LUV case) and to extrapolate to larger number of processors (in both cases). The resulting projections are shown in figure 3.

By fitting equation (7) to the timing results for the SUV case (figure 3, left), we find that the parallel performance is good at least up to 100 processors, with the efficiency not projected to drop below 0.5 before the number of processors goes above 128–200. We hope to be able to test this assertion in the near future.

In the LUV case (figure 3, right), we used the modelled performance not only to extrapolate the performance but also to find the single-processor runtime. This value anchors the measured

[9] In the LUV case we only fitted the a, b and c parameters, and used the same d as for SUV. d is independent of problem geometry and partitioning.

	SUV				LUV			
P	I	T [s]	S	E	I	T [s]	S	E
1	47	1516.2	1.00	1.00	42	(11629)	1.00	1.00
2	48	783.2	1.94	0.97	53	—	—	—
4	54	449.4	3.37	0.84	44	3148.2	3.69	0.92
6	52	290.2	5.22	0.87	46	2193.5	5.30	0.88
8	67	280.0	5.42	0.68	38	1397.6	8.32	1.04
10	—	—	—	—	44	1310.3	8.88	0.89
12	59	171.3	8.85	0.74	39	991.7	11.73	0.98
14	—	—	—	—	39	850.0	13.68	0.98
16	51	116.2	13.05	0.82	40	777.9	14.95	0.93
18	—	—	—	—	43	723.1	16.08	0.89
20	56	102.4	14.80	0.74	44	676.7	17.19	0.86
22	—	—	—	—	38	537.9	21.62	0.98
24	68	104.8	14.46	0.60	41	524.8	22.16	0.92
28	53	71.5	21.20	0.76	41	455.9	25.51	0.91
32	52	62.9	24.09	0.75	41	402.5	28.89	0.90
36	57	62.1	24.43	0.68	36	321.6	36.16	1.00
40	59	58.3	26.01	0.65	42	340.0	34.20	0.86
42	66	61.7	24.57	0.58	37	284.1	40.93	0.97
64	53	—	—		53	—	—	—
128	48	—	—		37	—	—	—

Table 3: Measured iteration counts and runtimes, with associated speedup and efficiency values. The LUV single-processor runtime is estimated, not measured; in all other cases, "—" is used to indicate missing measurements. The speedup and efficiency values are relative to the single-processor runtime.

values of speedup and efficiency, since they are both defined in terms of the runtime on one processor. Hence there is in this case a small uncertainty in the absolute positioning of the measured points on the efficiency scale (their relative positions are unaffected).

The performance model predicts that larger problems have better parallel scalability than smaller problems, which we also see in the measurements. Depending on the reliability of the performance model, the SUV case may have adequate performance up to perhaps one thousand processors (figure 3, right), but the extrapolation at such a distance is highly uncertain. Even so, we know that the parallel efficiency gets better as the size of the problem increases. Thus, as even larger simulations are required, scaling into the thousands of processors should be possible.

Conclusion

We have shown that the algebraic multigrid method, as implemented by ML, performs well as preconditioner of the system of equations arising from the coupled equations of heat, pressure and linear elasticity, even when used as black-box software with no tuning to the specific system of equations. Furthermore, the preconditioned BiCGStab solver, parallelised at the linear algebra level using Diffpack tools (with slight modifications to accommodate the AMG preconditioner) scales adequately at least up to medium-sized clusters (10–100 nodes), and likely to even larger clusters for problems with millions of mesh points.

References

[1] A.M.Bruaset *A Survey of Preconditioned Iterative Methods* Addison-Wesley Longman (currently CRC Press), 1995.

[2] A.M.Bruaset and A.Tveito, editors *Numerical Solution of Partial Differential Equations on Parallel Computers* Springer, 2006.

[3] X.Cai, E.Acklam, H.P.Langtangen and A.Tveito Parallel computing In Langtangen and Tveito [8], 1–96.

[4] Diffpack Library for numerical solution of PDEs from inuTech GmbH.

[5] M.W.Gee, C.M.Siefert, J.J.Hu, R.S.Tuminaro and M.G.Sala ML 5.0 smoothed aggregation user's guide Technical Report SAND2006-2649, Sandia National Laboratories, 2006.

[6] F.Hülsemann, M.Kowarschik, M.Mohr and U.Rüde Parallel geometric multigrid In Bruaset and Tveito [2], 165–208.

[7] A.Kjeldstad, H.P.Langtangen, J.Skogseid and K.Bjørlykke Simulation of sedimentary basins In Langtangen and Tveito [8], 611–658.

[8] H.P.Langtangen and A.Tveito, editors *Advanced Topics in Computational Partial Differential Equations* Springer, 2003.

[9] H.Osnes, X.Cai, A.M.Bruaset, H.P.Langtangen, O.Lauvrak and J.Skogseid Simulation of deformation and heat flow in sedimentary basins Internal report, Dec. 2006.

[10] Petromod Petroleum systems modelling software from IES GmbH.

[11] R.E.Showalter Diffusion in deforming porous media *Dyn. Cont. Discr. Impuls. Syst. (Series A: Math. Anal.)*, **vol.10**(5), 661–678, 2003.

[12] B.F.Smith, P.E.Bjørstad and W.Gropp *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations* Cambridge University Press, 1996.

[13] H.A.van der Vorst Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of of nonsymmetric linear systems *SIAM J. Sci. Stat. Comput.*, **vol.13**(2), 631–644, 1992.

[14] U.M.Yang Parallel algebraic multigrid methods—high performance preconditioners In Bruaset and Tveito [2], 209–236.