Benchmarking DOUG on the Cloud

Oleg Batrashev, Satish Narayana Srirama, Eero Vainikko
Institute of Computer Science, University of Tartu
J. Liivi 2, Tartu, Estonia
{olegus, srirama, eero}@ut.ee

***Abstract*—Large systems of linear equations with sparse matrices arise often in scientific computing problems and engineering tasks. For efficient solution of such problems iterative techniques like preconditioned Krylov subspace methods are used. Domain decomposition preconditioners are good for reducing the number of iterative steps together efficient parallelisation of the problem. While cloud computing infrastructure has become quite attractive also for the HPC community, this paper gives an overview of DOUG (Domain Decomposition on Unstructured Grids) implementation with the focus of important parameters for parallel performance on computer clusters as well as on the SciCloud (Scientific Computing on the Cloud) environment. We describe the used methods and perform a number of tests for benchmarking the application on both environments.**

***Keywords*-cloud computing; parallel scientific computing problems; domain decomposition; Krylov subspace methods;**

I. Introduction

Cloud computing [6] is a style of computing in which, typically, resources scalable on demand are provided "as a service (aaS)" over the Internet to users who need not have knowledge of, expertise in, or control over the cloud infrastructure that supports them. The provisioning of cloud services can be at the Infrastructural level (IaaS), Platform level (PaaS) or at the Software level (SaaS). Cloud computing suits well in solving parallel scientific computing problems, with its promise of provisioning virtually infinite resources. Thus to take the benefits of the cloud, we tried to move several of our parallel application to the cloud. One such application is DOUG (Domain decomposition On Unstructured Grids). DOUG is an open source software package for parallel iterative solution of very large sparse systems of linear equations with up to several millions of unknowns.

While running the parallel applications on our private cloud, SciCloud [11], it was realized that the transmission delays in cloud environment to be the major problem for adapting HPC problems on the cloud [10]. However, the transmission delays were not so significant for DOUG when compared to pure Conjugate Gradient (CG), as DOUG has much better computation-communication ratio than pure CG. Nevertheless, while porting DOUG to the cloud, it was also observed that, changing some parameters of DOUG may make it effective for the cloud. Considering this hypothesis, we studied varying various parameters of DOUG and conducted the benchmarking to see the effects. The details are produced in the paper and it is organized as follows.

Section II gives the description of DOUG package and some preliminaries needed for describing the problem of solving large systems of linear equations with sparse matrices. In Section III we give an overview of two-level preconditioner we are using. The next Section discusses the partitioning and aggregation strategies for efficient parallel implementation. Section V focuses on parallel implementation of the given methods and is followed by parallel performance tests on cluster and SciCloud in Section VI. Section VII concludes the paper and describes the future research directions in the field.

II. Parallel scientific applications

As already mentioned, cloud computing suits well in solving parallel scientific computing problems, with its promise of provisioning virtually infinite resources. To verify this, along with several parallel applications, like NAS PB (NASA Advanced Supercomputing Parallel Benchmarks) CG, embarrassingly parallel applications, we also tried to move our DOUG to the SciCloud. SciCloud is a Eucalyptus based private cloud built on University of Tartu, HPC cluster. With the SciCloud, students and researchers can efficiently use the already existing resources of university computer networks, in solving computationally intensive scientific, mathematical, and academic problems. Traditionally, such computationally intensive problems were targeted by batch-oriented models of the GRID computing domain. The SciCloud project tries to achieve this with more interactive and service oriented models of cloud computing that fits a larger class of applications. The analysis of moving scientific computing application to the cloud had given us a chance to have a clear look at the comparison of running the scientific and mathematical problems in a cluster and on the cloud. However, this paper address porting DOUG to SciCloud.

A. DOUG (Domain decomposition On Unstructured Grids)

DOUG is a software package for parallel solution of large sparse systems of linear equations typically arising from discretizations of partial differential equations to be solved in 2D or 3D regions which may consist of materials with high variations in physical properties. DOUG is developed at the University of Tartu, Estonia and the University of Bath, England since 1997. It was first implemented in FORTRAN 77 but has been completely rewritten in Fortran 95. To achieve good parallel performance, DOUG uses automatic load-balancing and parallelization being implemented through MPI and overlapping communication and calculations through non-blocking communication whenever it is applicable.

Basically, DOUG uses an iterative Krylov subspace method to solve linear systems in a form

$$Ax = b,$$

where the matrix A is sparse and the dimension of A is very large. Due to large dimensions of A the convergence rate is by far too slow. For that reason, a preconditioner is used. A good preconditioner transforms the original linear system into one with the same solution, but with the transformed linear system the iterative solver needs much smaller number of iterations [7]. DOUG implements the Preconditioned Conjugate Gradient (PCG), the Stabilized Bi-Conjugate Gradient (BiCGstab), the minimal residual (MINRES) and the 2-layered Flexible Preconditioned Generalized Minimum Residual method (FPGMRES) with left or right preconditioning.

The general algorithm used for creating the sub-problems that can be assigned to separate CPUs is called domain decomposition. The idea of domain decomposition is to decompose the problem into n sub-problems, usually with some overlap, each of which will be solved on one CPU. In a way, it is similar to a divide and conquer scheme but with domain decomposition there is communication on the borders of the sub-problems (overlaps) involved. Usually, communication is measured to be more costly than CPU time and therefore the decomposition algorithm tries to minimize the cross-sections between neighboring subdomains.

In domain decomposition, instead of solving the global system of equations, smaller problems are solved on each sub-domain, solutions of which are combined together to form an approximation to the original problem. The common practice is to use domain decomposition as a preconditioning step for Krylov subspace methods such as the conjugate gradient method or the method of generalized minimum residual [2]. DOUG employs 2-level preconditioning in which a coarse matrix is used which approximates the global matrix on a suitable chosen coarser scale. This reduces the total work of a preconditioned Krylov method (like PCG) to almost a constant number of iterations independent of the matrix A size.

Recently, the development and research has been focused around aggregation based domain decomposition methods. Major progress has been made in determining an upper bound for the condition number of the preconditioner in case of highly variable coefficients. This allows a better estimate of the error and thus enables the solver to finish in less iterations. This approach has been implemented in DOUG and it has been shown experimentally that it is of superior speed to comparable methods. [8], [9]

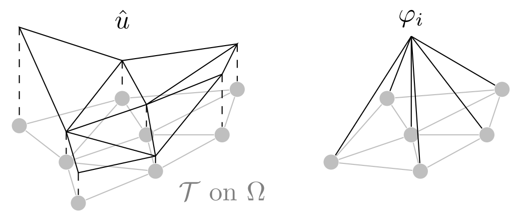
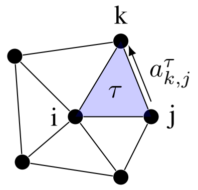


Figure 1. Piecewise linear function $\hat{u}$ and node i basis function



$$A^\tau = \begin{pmatrix} a^\tau_{i,i} & a^\tau_{i,j} & a^\tau_{i,k} \\ a^\tau_{j,i} & a^\tau_{j,j} & a^\tau_{j,k} \\ a^\tau_{k,i} & a^\tau_{k,j} & a^\tau_{kk} \end{pmatrix}$$

Figure 2. Mesh element and corresponding element matrix

B. Preliminaries

The original boundary-value problem is described by elliptic partial differential equation

$$-\nabla \cdot (\alpha \nabla u) = f,$$

defined on the physical domain Ω, where $u = u(x)$ and $x \in \Omega$. The system of linear equations $A\mathbf{u} = \mathbf{b}$ to be solved is usually generated using Finite Element method (FEM). In FEM the domain Ω is discretized into finite elements $\tau \in \mathcal{T}$ (e.g. triangles for 2D) which comprise the conforming mesh with nodes x_i indexed with $i \in \mathcal{I}(\Omega)$. Instead of the actual solution $u(x)$, FEM looks for its approximation $\hat{u}(x)$, which is piecewise linear function with respect to $\mathcal{T}$ (i.e. linear on each element τ). This function is expressed through *fine grid basis functions* φ_i (hat functions) as (see Figure 1)

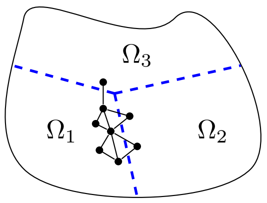
$$\hat{u} = \sum u_i \varphi_i \,.$$

Each function φ_i equals to 1 at node x_i, is 0 at all other nodes and linearly fades to 0 in immediate neighbour elements. Vector $\mathbf{u}$ values u_i define the function $\hat{u}$ through the basis functions $\{\varphi_i\}$.

Thus, each node in the mesh contains one unknown u_i and elements of the mesh describe the relations between the unknowns with *element matrices* (see Figure 2). The *global stiffness matrix* A is the composition of element matrices, where $A_{ij} \neq 0$ denotes a relation between the unknowns u_i and u_j. This, among other implications, causes to fetch z_j value to calculate y_i in parallel matrix vector multiplication $\mathbf{y} = A\mathbf{z}$. Hence only unknowns of the neighbouring nodes are being related, the matrix A is sparse and reflects the connectivity of the mesh grid.

III. DOUG: TWO-LEVEL PRECONDITIONER

DOUG uses two-level preconditioner $M^{-1} = M^{-1}_{AS} + M^{-1}_{C}$: *Schwarz preconditioner* on subdomains and *coarse grid preconditioner*. The methods of both levels rely on partitioning of the mesh grid. The partitioning process



$$R_1 = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

Figure 3. Subdomains and subdomain Ω_1 restriction matrix

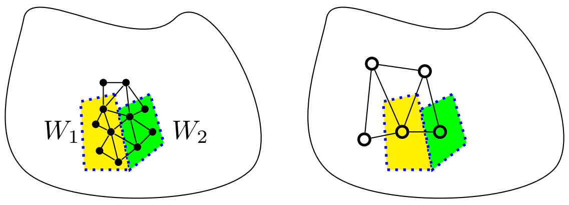


Figure 4. Aggregates and coarse grid

details are described in Section IV. Here we describe how preconditioners are constructed given the partitions.

A. Additive Schwarz preconditioner

Schwarz methods are based on the idea of decomposing the domain of initial problem into subdomains Ω_i, so that sub-problems are solved independently and the result is combined from the solutions on the subdomains (see Figure 3). Define by $\mathcal{I}(\Omega_i)$ the index set of nodes in the subdomain Ω_i $(i = 1, ..., n_d)$, where n_d is the number of subdomains. Then for $p \in \mathcal{I}(\Omega_i)$ and $q \in \mathcal{I}(\Omega)$ the subdomain injection matrix R_i of shape $|\mathcal{I}(\Omega_i)| \times |\mathcal{I}(\Omega)|$ selects the values in the subdomain nodes

$$(R_i)_{pq} = \delta_{pq} = \begin{cases} 1 & \text{if } p = q \\ 0 & \text{otherwise.} \end{cases}$$

Local matrix $A_i = R_i A R_i^T$ is just a minor of A corresponding to rows and columns of indices from $\mathcal{I}(\Omega_i)$. Non-overlapping Schwarz preconditioner with complete local solves is defined with

$$M_{AS}^{-1} = \sum_{i=1}^{s} R_i^T A_i^{-1} R_i.$$

In general case the subdomains may overlap, so that one node belongs to several subdomains. With the Additive Schwarz method we just add the solution values of the overlapping nodes and the previous formula stays the same for the overlapping case.

B. Coarse grid preconditioner

This method uses coarse grid which is constructed from fine grid aggregates $W = \{W_i | i = 1, \ldots, n_a\}$, where $n_a$denotes the number of aggregates. Each aggregate W_i consists of several fine grid nodes x_k and each coarse grid node corresponds to one aggregate (see Figure 4). The coarse space restriction matrix R_0 of shape $|W| \times |\mathcal{I}(\Omega)|$ maps fine grid values to the coarse grid

$$(R_0)_{iq} = \begin{cases} 1 & \text{if } x_q \in W_i \\ 0 & \text{otherwise.} \end{cases}$$

The meaning of the restriction matrix R_0 is to specify combinations of the fine grid basis functions φ_i to form *coarse space basis functions*

$$\Phi_i(x) = \sum_{q \in \mathcal{I}(\Omega)} (R_0)_{iq} \varphi_q(x) \; x \in \Omega, \; i = 1, \ldots, n_a.$$

The same approach of approximating $u(x)$ with a piecewise linear function is transferred to the coarse level with the new smaller basis. The nodes x_q where $\Phi_i(x_q) \neq 0$ is called *coarse space basis function* Φ_i *support* and it coincides with the aggregate W_i nodes.

Coarse matrix $A_0 = R_0 A R_0^T$ defines the problem on the coarse grid. Coarse grid method restricts the values to the coarse grid, solves the problem and interpolates the solution back to the fine grid

$$M_C^{-1} = R_0^T A_0^{-1} R_0. \tag{1}$$

In general, aggregates may overlap, in which case $\sum_{i, x_q \in W_i} (R_0)_{iq} = 1$ must still hold for each q. The overlap for aggregates is achieved by smoothing process described in Section IV-C.

Highly-variable coefficient: Overlap of the aggregates significantly reduces the number of iterations. However, if there is a jump in the coefficient on the boundary of two aggregates

$$\alpha(x_p) \ll \alpha(x_q), x_p \in W_i, \; x_q \in W_j, \; i \neq j, \; A_{pq} \neq 0$$

the smoothing is much less efficient. The idea of "smart" smoothing is to avoid overlap in such places.

IV. Partitioning and Aggregation

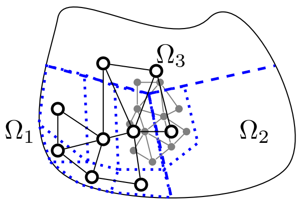
This section describes DOUG partitions and their overlaps as well as partitioning algorithm of smoothed aggregation, implemented in DOUG.

A. Partitioning

Two-level preconditioner in DOUG requires two types of partitions: subdomains for the Schwarz preconditioner and coarse space basis function supports for the coarse grid preconditioner. The latter are supposed to be smaller and the former should be as large as server memory allows but not too large, because matrix factorization time for a subdomain increases non-linearly. In DOUG each MPI process handles exactly one subdomain. Further, for parallel implementation it is convenient to have conforming subdomain and support boundaries, so that (see Figure 5)

$$\forall W_i \, \exists! \Omega_j \; W_i \subset \{x_k | k \in \mathcal{I}(\Omega_j)\}$$

Hence in DOUG 2-phase process generates both partition types: first, small partitions on the fine mesh are generated, then larger partitions are created by joining fine mesh partitions. Both subprocesses may be accomplished



a) coarse grid is defined by the aggregates on the fine grid,
b) subdomains Ω_i are defined by the aggregates on the coarse grid

Figure 5. 2-phase partitioning in DOUG

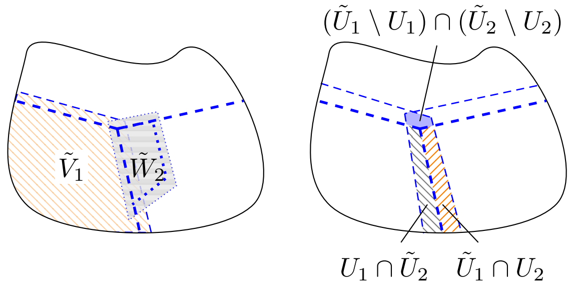


Figure 6. Expanded aggregates and process region overlaps

by either any ready graph partitioning library, like Metis, or DOUG aggregation algorithm. The following terms are used for the partitions

1) *fine aggregates* W_i – partitions of the fine grid, which serve as basis function supports in the coarse grid preconditioner;
2) *coarse aggregates* $V_i = \{x_k | k \in \mathcal{I}(\Omega_i)\}$ – partitions of the coarse grid, which serve as subdomains in the Schwarz preconditioner.

In fact, 3-phase partitioning process is implemented in DOUG, because it involves distribution of values between processes. The details are described in Section V-A.

B. Overlaps and process region

After aggregates are generated DOUG expands the aggregates by adding several layers of neighbouring nodes using matrix A values. The following grid regions are formed (see Figure 6):

- $\tilde{V}_i$ – nodes in the subdomains of the overlapping Schwarz preconditioner with the overlap n_o,
- $\tilde{W}_i$ – nodes in coarse space basis function support after n_s smoothing steps.

On each process i all local regions are combined and single *process region* is formed, which also comes in two variations:

- interior $U_i = V_i$
- with overlap $\tilde{U}_i = \tilde{V}_i \cup \left(\bigcup_{j, W_j \subset V_i} \tilde{W}_j\right)$
  - If $\max(n_o, n_s) = 0$ then $\tilde{U}_i$-s are extended by one layer using A as an adjacency matrix. This is needed for parallel matrix-vector multiplication.

The general assumption in DOUG is that the values in the process region $\tilde{U}_i$ are supposed to be valid after each parallel operation. This may require receiving data from other processes if the values in the overlap are changed on a remote process. Constructing and handling the overlaps for all regions is essential part of DOUG code.

The overlaps for process i with process j are (see Figure 6): *inner* $U_i \cap \tilde{U}_j$, *outer* $\tilde{U}_i \cap U_j$, and *total* $\tilde{U}_i \cap \tilde{U}_j$. Note that total overlap may be larger than the union if inner and outer overlaps by the overlap on third subdomains $(\tilde{U}_i \setminus U_i) \cap (\tilde{U}_j \setminus U_j)$.

C. Smoothed aggregation

In the center of the coarse problem creation and partitioning of the problem into the subdomains in a "smart" way taking account the parameter jumps, is the Smoothed Aggregation Setup (see Algorithm 1). On input there is given a large sparse matrix A. The method is flexible in a way that no discretizations grid coordinates are needed to be given. The aggregation idea is somewhat similar to the Algebraic Multigrid method coarsening strategies, where the coarse problem is built on top of aggregated nodes.

Algorithm 1 Smoothed Aggregation Setup

```
Input: Matrix $A$ defined on nodes $\mathcal{N}$, $\varepsilon \in [0,1]$, aggregate
    size bounds $a_{min}$, $a_{max}$, aggregation radius $r$ and
    number of smoothing steps $n_s$
Output: Set of (in case $n_s > 0$ overlapping) aggregates
    $W = \{W_j : j = 1, ..., n_a\}$, Restriction matrix
    $R_0 : \mathbb{R}^{|\mathcal{N}|} \to \mathbb{R}^{n_a}$, Interpolation matrix $R_0^T$ and coarse
    matrix $A_0$
1. Scale the matrix $A := (\text{diag}A)^{-1/2} A\ (\text{diag}A)^{-1/2}$
2. Filter out weak connections from matrix $A$ for which
    $|A_{ij}| < \varepsilon \max_{k \neq i} |A_{ik}|$;
3. Initialise $C := \emptyset$; $\mathcal{F} := \mathcal{N}$; $j = 0$
repeat
    4. $j := j + 1$; choose a seednode $x_j$ from $C$ (or randomly
       from set $\mathcal{F}$ if $C == \emptyset$)
    5. Set layer $L(0) := \{x_j\}$; $\mathcal{F} := \mathcal{F} \setminus L(0)$ and $W_j = L(0)$
    6. for $i = 1 : 2r + 1$
        (a) Set layer $L(i) := \bigcup_{x_k \in L(i-1)} (\{x_\ell : A_{k\ell} \neq 0\})$
        (b) If $i \leq r$, add to $L(i)$ all $x \in \mathcal{F}$ that are connected
            through $A$ to at least 2 nodes in $L(i)$, set
            $\mathcal{F} := \mathcal{F} \setminus L(i)$ and set $W_j := W_j \cup L(i)$.
    7. Find $i_{max} := \operatorname{argmax}_{i \in \{r+1,...,2r+1\}} |L(i)|$ (i.e. the largest layer)
        and add to $C$ all $x \in L(i_{max})$ of shortest path length
        from $x_j$
until $\mathcal{F} == \emptyset$
8. set $n_a := j$
9. Merge any aggregate $W_j$ that is too small (i.e.
    $|W_j| < a_{min}$ ) with a connected neighbouring aggregate
    $W_k$ (subject to the requirement $|W_j \cup W_k| \leq a_{max}$; it
    may be necessary to split up $W_j$ to achieve this) and
    shrink $n_a$ accordingly.
10. Form the aggregate projector operator $P : \mathbb{R}^{|\mathcal{N}|} \to \mathbb{R}^{n_a}$,
    where $P_{jk} = \{1 \text{ if } x_k \in W_j \text{ or } 0, \text{ otherwise}\}$
11. Form the restriction operator $R_0 = PS^{n_s}$, with
    $S = (I - \omega A)$ (applying $n_s$ times a damped Jacobi
    smoother); aggregates grow by $n_s$ layers as well, forming
    overlaps.
12. Form the coarse problem matrix $A_0$ through sparse matrix
    multiplication $A_0 = R_0 A R_0^T$
```

Such nodes are close to each other on the fine level, forming a set of nodes not further than $2r$ connections away from each other through matrix A nonzero values, where r is called the aggregation radius.

For the algorithm setup phase 1, the actual matrix A gets temporarily scaled down for the diagonal entries to equal one everywhere for faster filtration phase at step 2, depending on the constant $\varepsilon \in [0, 1]$. After the initializations only the strongly connected values play role in the aggregation phase (steps 4-9). The aggregation algorithm is built up in a way that if the entries in the stiffness matrix A do not show substantial jumps in coefficient, the aggregates would look fairly regular in shape (e.g. rectangles in 2D case). This is achieved through special strategy for choosing seed nodes in aggregation algorithm (step 7). At the same time, weak connections are much more likely to end up in different aggregates than the strong ones. This has proved [8], [9] to improve substantially the convergence speed of the two-level Additive Schwarz method preconditioned with the aggregated coarse space solver on top of the original fine grid solvers defined on the subdomains.

For the coarse grid method to better reflect the underlying properties of the actual problem, smoothing technique can be used, as e.g. shown in [8], [9]. The smoother S is given in the Algorithm 1 step 11, which together with the restriction operator P forms the restriction operator R. Step 12 in the algorithm shows, how the coarse level matrix is formed through sparse matrix multiplication.

V. Parallel implementation

Domain Decomposition method as a preconditioner to some Krylov subspace method serves two goals: it helps to reduce substantially the number of iterations needed for the convergence of the iterative method and also is a mean for parallelisation of actual computations. At the same time, distribution of the work between different processors helps to share the load but also introduces the need for communication. The need for communication arises in several different parts of the algorithm:

A. Parallel computation initialization

Crucial to the parallel implementation are the decisions made on different choices closely related to Algorithm 1 different stages. Our implementation is based on the master-slaves model. Distribution of matrix A is done in 3 phases:

1) Master preforms first a rough aggregation of the fine grid, i.e. with omitting Algorithm 1 stages after step 6.
2) Sparse graph structure (preliminary coarse grid) is built on top of the fine aggregates, which is then split into n_p parts using graph partitioning software METIS [5]. Computed coarse aggregates define subdomains $U_i = V_i$. Then process domains $\tilde{U}_i$ are computed by expanding the subdomains with $\max(1, n_o, n_s)$ layers.
3) Thereafter the submatrices of A, limited by $\tilde{U}_i$, are delivered to slave processes, which now perform in parallel the whole Algorithm 1 but only using submatrix values limited by U_i (locally computed aggregates do not extend to the outer overlap). As the result, smoothed fine aggregates $\tilde{W}_i$ are obtained locally, then global numbering of the aggregates is performed using processes ranks.

Subdomain injection matrices R_i $i = 1 \dots n_d$ are obtained directly from $\tilde{V}_i$ and do not require any other treatment, except for re-indexing matrix A to prepare data for complete local solve A_i^{-1} in the Schwarz preconditioner. We use the library UMFPACK [4] as a local solver.

Coarse space restriction matrix R_0 is constructed by the following steps:

1) Matrix $R_0^{(i)} : \left(R_0^{(i)}\right)_{jq} = (R_0)_{jq}$ $W_j \subset V_i$, is constructed locally on each process i by smoothing in step 3) above.
2) The values on the overlap $\left(R_0^{(i)}\right)_{jq} : W_j \subset V_i,\ x_q \in \tilde{U}_i \cap \tilde{U}_k$ are delivered from process i to process k. This is the preparation for the coarse matrix A_0 computation, described below.

For coarse problem the matrix $A_0 = R_0 A R_0^T$ is created by computing local parts and then delivering resulting parts of A_0 to each process in DOUG. Deciding which parts to compute, so that there is no duplicates, is not straightforward. Computing matrix A_0 element

$$(A_0)_{pq} = \sum_k \sum_l r_{pk} \cdot a_{kl} \cdot r_{ql}$$

on process i by the index p (i.e. if $W_p \subset U_i$) is the most obvious way, but it may require a_{kl} and r_{ql} values that lie outside the process region $x_l \notin \tilde{U}_i$. Another possibility (implemented in DOUG) is to assign products $r_{pk} \cdot a_{kl} \cdot r_{ql}$ by index k, so that process i computes only those with $x_k \in U_i$.

B. Dot products

In Krylov subspace methods there are one or several dot products needed to be calculated on each iteration step. Dot products are actually somewhat bad to the method performance – although the calculations are done in parallel, it is not possible to hide the communication needed for gathering the distributed values together behind some other useful calculation. With reducing the number of Krylov subspace iterations also the number of synchronization steps, needed in parallel program, reduces. Therefore, it is crucial to use the best available coarse problem technique to reduce the overall number of iterations and this is why the second level problem solution technique plays an important role.

C. Matrix-vector multiplication

While dot-products would be better to avoid as much as possible in parallel program, communication needed for synchronization of subdomain direct neighbour values after each matrix-vector operation can be hidden in the background while calculating the values not involved in data transfer. Good implementation here is of extreme importance to the resulting parallel performance.

D. Preconditioner application

As the preconditioner used is the two-level Additive Schwarz method, there are two types of communication patterns that occur here.

On fine level, neighbour communication is needed after each fine grid problem solution, pretty similar to the matrix-vector operation case:

1) Local solve on the subdomain $\tilde{V}_i$ is performed.
2) The values of the nodes $\tilde{V}_i \cap \tilde{U}_j$ need to be sent from process i to process j.

On coarse level, information needs to be gathered from all processes to the coarse problem solvers. The coarse problem in DOUG is solved on all processes simultaneously and therefore no communication is needed for interpolation of the coarse solution back to the fine grid. However, current implementation in DOUG is different and requires second communication if $n_s > 0$, the overall algorithm is the following:

1) local coarse vector $y^{(i)} = R_0^{(i)} r^{(i)}$ is computed on each process i
2) all processes gather local coarse vectors into global coarse vector y
3) each process solves the same coarse problem $v = A_0^{-1} y$ independently
4) coarse solution v is interpolated back to process region $\tilde{U}_i$ but only v_j, $W_j \subset V_i$ values are interpolated to z
5) final communication need to be done for the nodes $x \in \left(\bigcup_{j, W_j \subset V_i} \tilde{W}_j \right) \cap \tilde{U}_k$: corresponding z values are sent from process i to process k.

On both levels, it is possible to combine the last step and do the communication for process region overlaps, so values $\tilde{U}_i \cap \tilde{U}_j$ are sent from process i to process j (and vice versa). For efficiency, step 1 of the Schwarz preconditioner is overlapped with step 2 of the coarse grid preconditioner, where all-to-all communication happens in the separate thread.

VI. Performance tests

The size of the grid considered for the analysis is 1536 times 1536, which gives rise to the system of linear equations with about 2.5 millions of unknowns. The matrix of the system contains more than 10 million non-zero elements. DOUG master node first reads data from disk and distributes it to slave nodes, which together with master node use Preconditioned CG algorithm to solve the system.

For testing we had four machines with 2.83GHz quad-core Intel Q9500 processors with 3MBytes of L2 cache, connected with Fast Ethernet network. Cluster has Scientific Linux with kernel 2.6.18 and SciCloud has Ubuntu Linux with kernel 2.6.27. The tests were repeated 3 times and average time was taken, though the run times were very stable.

	$r = 2$	$r = 4$	$r = 4$	$r = 4$
			$n_s = 1$	$n_s = 2$
$n = 2$	70	94	79	68
$n = 16$	71	97	81	70

Table I
NUMBER OF PCG ITERATIONS FOR DIFFERENT CONFIGURATIONS OF THE COARSE GRID PRECONDITIONER

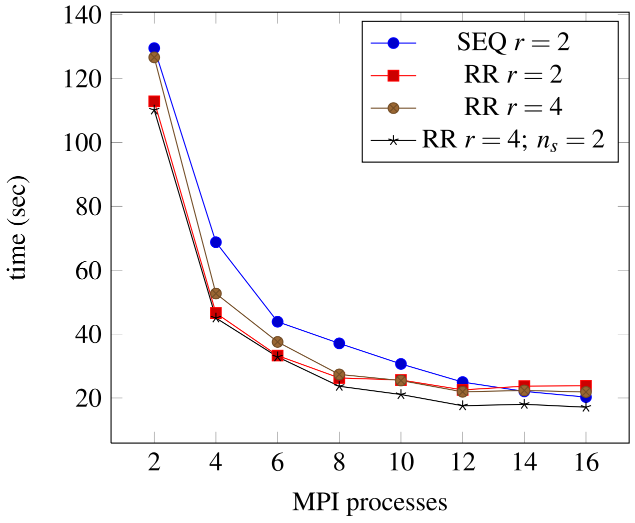


Figure 7. Sequential and round-robin scheduling

A. DOUG on cluster

On cluster we had a choice to schedule processes to cores in sequential (SEQ) or round-robin (RR) order. Sequential order fills all cores of a machine and then switches to another machine in the network while round-robin always spreads processes evenly between the machines. With SEQ the performance is worse, because DOUG process creates additional thread for communication and it requires more than one core. Another aspect is memory bus, which is a critical part of sparse problems, hence putting more processes to a machine degrades performance. This is seen from Figure 7.

We tried different setups primarily changing aggregation radius r and smoothing steps n_s on different number of processors $n = 2, 4, 6, \ldots, 16$. Increasing radius gives smaller coarse problem size, which reduces coarse grid preconditioner communication and solve time, but increases the number of iterations (see Table I). Adding smoothing decreases number of iterations but adds additional communication.

The run times for different number of processes are shown on Figure 7. The best time for 16 processes is observed with the aggregation radius of 4 and 2 steps of smoothing. With the larger number of processes coarse vector distribution becomes more critical, especially if the size of local problem is not large enough. Table II shows that with 16 processes and radius of 2 these sizes are already comparable. Without smoothing though the larger number of iterations nullifies the effect, so smoothing helps to reveal it.

n_d	2	4	6	8	10	12	14	16
sub-problem	1.18	0.59	0.39	0.29	0.23	0.2	0.17	0.15

r	2	4	8
coarse problem	0.097	0.03	0.0088

Table II
LOCAL AND COARSE PROBLEM SIZES IN MILLIONS OF UNKNOWNS

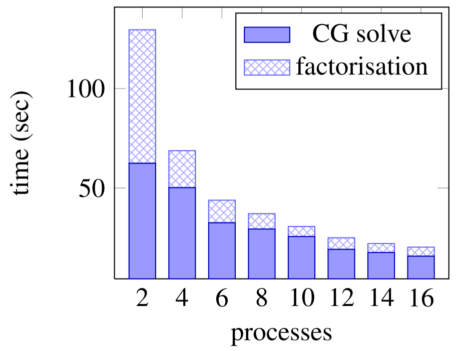


Figure 8. Solve and factorization times

Considerably larger times for 2 processes are because local solves with UMFPACK do not scale linearly, so factorization of the problem with 1.18 millions of unknowns takes a lot of time, even more than the whole CG algorithm (see Figure 8).

Finally, odd behavior with 16 processes, that reveals sequential scheduling is better than round-robin, can be explained by non-uniform communication costs on the cluster, i.e. there are four 4-core machines. For testing this hypothesis we saved coarse aggregate graph: aggregate neighbours and overlaps (see Figure 9), which also defines process communication pattern. Indeed, sequential scheduling eventually gives the best distribution of processes on the machines, while round-robin scheduling places neighbouring aggregates to different machines. The numbering is given by METIS routine and is quite causal. In the future we have to take control over distribution by passing this communication graph to the `MPI_Graph_create` routine or by some other means.

B. DOUG on SciCloud

To run DOUG on SciCloud, machine images with DOUG software have been prepared and the scripts that helped us in preparing the MPI setup are extended for the DOUG. The Cloud is situated on the cluster and uses the same hardware, but different operating system. Our previous tests showed that cloud may introduce communication overhead [11].

Running DOUG program with the given data takes more than 1 GBytes of memory when running on less than 3 nodes, so it was not possible to run it on cloud with 1 GBytes of available memory per VM. The cloud has been tested with 4 to 16 instances running on four 4-core nodes.

The scheduling of processes to the machines on the Cloud is currently not under our control. The expected

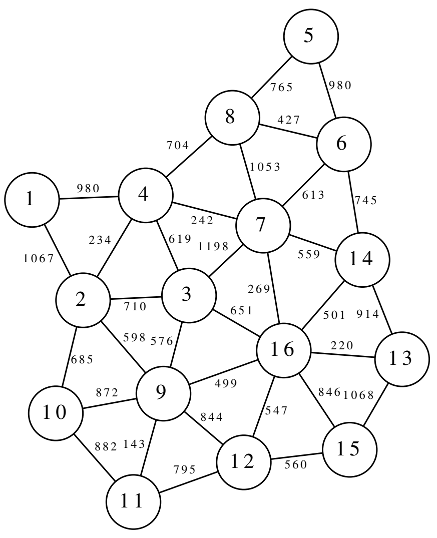


Figure 9. Graph of 16 coarse aggregates and overlap sizes

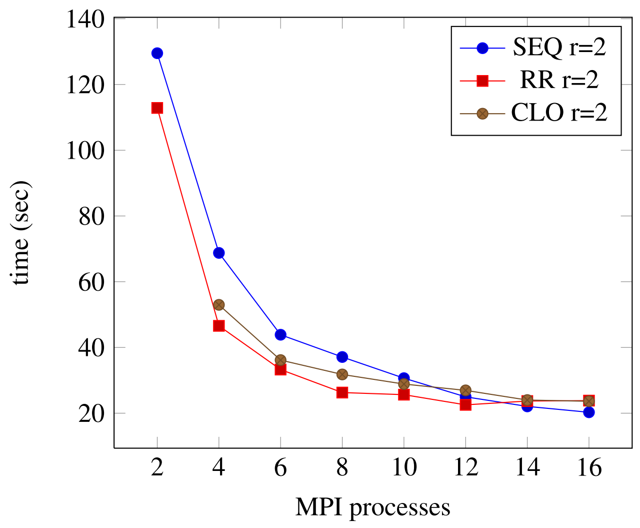


Figure 10. DOUG run times in cluster and cloud cases

behavior is the middle between sequential and round-robin scheduling. This is observed on Figure 10.

The graph shows that cloud does not introduce substantial extra communication overhead with 16 instances and overall performance is the same. However, the overall scaling is not very good for both cluster and cloud cases. This may be due to the optimal DOUG requirement to have 2 cores per processor. Our future research in this domain will address this issue in detail and we are interested in repeating the tests with 2 cores per instance. Our best case also does not show large differences (see Figure 11).

VII. CONCLUSIONS AND FUTURE RESEARCH

From this analysis it can be observed that parallel scientific applications scale reasonable on cloud. However, our experience of moving DOUG to SciCloud reveals some problems with cloud. First, instances in cloud are run on random nodes in the network. This may cause problems

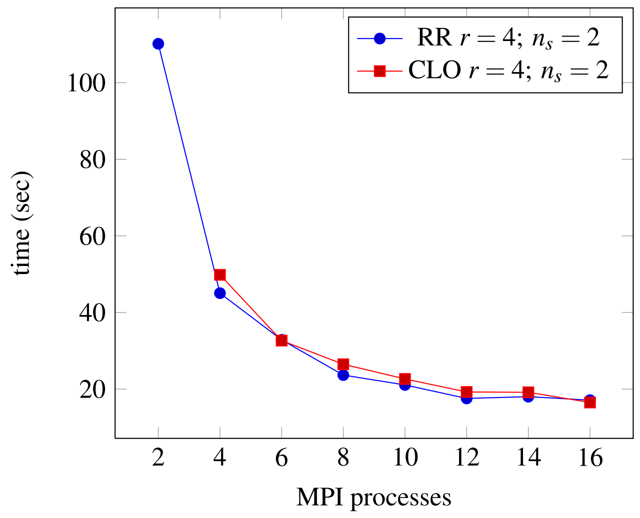


Figure 11. DOUG run times in cluster and cloud cases

of establishing connection from one instance to another. We circumvented the problem with our scripts, which made them initiate on a single cluster or on a single node of a cluster. Public clouds, especially Amazon EC2, also provide such support for HPC, with their Cluster Compute instances, which promise much lesser latency between instances [1]. Second problem is that MPI configuration needs to be created dynamically during run time, which is done by our scripts. However, for others to move their MPI applications to the cloud, additional tools are helpful.

In addition, we have observed that for parallel applications like DOUG on cluster, the process mapping strategies (like round-robin or sequential scheduling above) play substantial role for the parallel application performance. In our future work we intend to develop the possibilities of matching underlying cloud hardware with actual parallel process communication topology graph like it has been studied e.g. in [3] for computer clusters. We are also interested in performing tests with larger than 16 processes.

Acknowledgments

The research is supported by the European Social Fund through Mobilitas program, the European Regional Development Fund through the Estonian Centre of Excellence in Computer Science and Estonian Science Foundation grant No7566 and European Social Fund through the Estonian Doctoral School in Information and Communication Technology.

References

[1] Amazon Inc. High Performance Computing Using Amazon EC2, http://aws.amazon.com/hpc-applications/. Online. URL last visited on 5th Nov 2010.

[2] T. F. Chan and T. P. Mathew. Domain decomposition algorithms. *Acta Numerica*, 3(-1):61–143, 1994.

[3] H. Chen, W. Chen, J. Huang, B. Robert, and H. Kuhn. Mpipp: an automatic profile-guided parallel process placement toolset for smp clusters and multiclusters. In *Proceedings of the 20th annual international conference on Supercomputing*, ICS '06, pages 353–360, New York, NY, USA, 2006. ACM.

[4] T. A. Davis. Algorithm 832: Umfpack v4.3—an unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, 30:196–199, June 2004.

[5] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20:359–392, December 1998.

[6] M. Armbrust et al. Above the clouds, a berkeley view of cloud computing. Technical report, University of California, Feb 2009.

[7] Y. Saad. *Iterative methods for sparse linear systems*. PWS, 1st edition, 1996.

[8] R. Scheichl and E. Vainikko. Additive schwarz and Aggregation-Based coarsening for elliptic problems with highly variable coefficients. Bath Institute For Complex Systems Preprint 9/06, 2006.

[9] R. Scheichl and E. Vainikko. Robust Aggregation-Based coarsening for additive schwarz in the case of highly variable coefficients. In P. Wesseling, E. ONate, and J. Periaux, editors, *Proceddings of the European Conference on Computational Fluid Dynamics, ECCOMAS CFD 2006*, TU Delft, 2006.

[10] S. N. Srirama, O. Batrashev, P. Jakovits, and E. Vainikko. Scalability of parallel scientific applications on the cloud. *Scientific Programming Journal, Special Issue on Science-driven Cloud Computing*, 2011. DOI:10.3233/SPR-2011-0320, (In print).

[11] S. N. Srirama, O. Batrashev, and E. Vainikko. SciCloud: Scientific Computing on the Cloud. In *The 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing(CCGrid 2010)*, page 579, 2010.