Parallel Computing and Optimization with COMSOL Multiphysics

Sol Maja Bjørnsdotter Fossen

Master of Energy and Environmental Engineering
Submission date: July 2017
Supervisor: Robert Nilssen, IEL

Norwegian University of Science and Technology
Department of Electric Power Engineering

Parallel Computing and Optimization with Comsol Multiphysics

Fossen, Sol Maja B.

Master student
Department of Electric Power Engineering
Norwegian University of Science and Technology
Trondheim, Norway
Email: solmajab@stud.ntnu.no

***Abstract*—In this thesis the parallel capabilities of COMSOL Multiphysics are investigated. A description for how one can run COMSOL on a Linux cluster is presented. The speedup was found to be poor for medium-sized simulations with less than 10 million degrees of freedom. The speedup for parametric sweeps were found to be excellent. Particle swarm optimization (PSO) was implemented using LiveLink for Matlab, and run on the supercomputer at NTNU. It was found to perform very well without any tuning of the algorithm.**

I . INTRODUCTION

Motivation

This thesis was motivated by the wish to run larger FEM simulations by the electrical power department.

Research objectives are often restricted by the computational resources available. There is a trade of between the accuracy of a simulation and the simulation time, and simulations will be created with time and memory-constraints in mind. When it comes to FEM software, models are usually simplified in order to achieve a reasonable simulation time. For example by using linear models instead of complex models.

NTNU has a supercomputer (Vilje) at campus which is available for students and phd's, but it is being utilized by master students to a very small degree.

Some reasons that Vilje has not been utilized more is

- No awareness: Students doesn't know it exists, or that it is available for them
- Limited knowledge: students don't know how to use it
- Limited need: Master projects seldom require heavy computations

There is a gap between the electrical power department and the world of cluster computing. The goal of this thesis is to bridge the gap, so that future students are not restricted by the computational resources of a laptop.

COMSOL Multiphysics is the chosen FEM software by NTNU's electrical power engineering department. It is a powerful and versatile simulation software with cluster capabilities.

Scope of Work

The main objectives at the start of this thesis was to gain experience in running COMSOL Multiphysics on the supercomputer Vilje.

The most common types of jobs that requires a great deal of computational resources are

- Large 3D simulations with millions of degrees of freedom
- Time dependent simulations
- Optimization methods that run hundreds or thousands of simulations

A general description of how one can run COMSOL models on a cluster will be presented.

In master projects, time dependent simulations and optimization are perhaps the most common jobs. Of these, optimization is the type of job that will benefit the most from parallel computing. Different optimization methods are reviewed, and a global optimization method is selected for implementation on Vilje. The goal is to 1) describe a method for how one can optimize COMSOL models on Vilje, and 2) use the method to optimize a model from fellow student Charlie Bjørk.

This thesis is partially meant to be a "how to"-manual for future students. Hopefully this work will make it easier for future master students to utilize high performance computing in their COMSOL projects.

II . SCIENTIFIC AND PARALLEL COMPUTING

Scientific computing can be defined as "the study of how to use computers to solve mathematical models in science and engineering". Scientific computing emerged in 1938 when Konrad Zuse build the first programmable computer in order to solve systems of linear equations [1]. Before that scientists had to make many simplifications to be able to solve a problem by hand, and only simple PDE's could be solved accurately.

Since then, scientific computing has become an increasingly important enterprise for researchers and engineers. Most industrial sectors rely on scientific computing when developing new designs and products. The development is driven by the need to solve larger and more complex problems.

Scientific computing combines mathematic models and numerical analysis to solve complex problems. The first step is to use understanding of the physical problem to set up a suitable mathematical model. The model will in most cases consist of differential equations and a number of initial and boundary conditions [2]. Numerical methods and computer science can then be used to solve the system of equations.

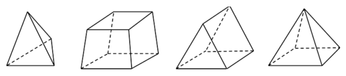

(a)

Fig. 1: Different element shapes [3]

There are many different methods available for solving linear and non-linear PDEs. One of the most practical numerical methods is *finite-element methods (FEM)*.

A. The Finite Element Method

The finite element method was discovered by several people independently, and gained traction in the 1960-70s. One of the main advantages of FEM is that it can handle complex geometry and material topology, and the ability to include general boundary conditions. FEM can solve systems made up of many different geometric and material regions.

The main principle of FEM is to divide a domain into many small, discrete elements, and applying the relevant differential equations to each element. A system of equations is constructed, and the system can then be solved using linear algebra or numerical schemes.

FEM can be divided into five separate steps:

1) **Discretize the domain into finite elements**
   The most common elements are triangular and rectangular linear elements for 2D, and tetrahedral, hexahedral and pyramid elements for 3D [4]. The elements are connected together at nodes.
2) **Select interpolation functions**
   FEM is discrete in nature, meaning that the solution is not computed for every point in the domain. The unknown variables are found at the nodal points. To approximate the variables at all points inside an element, an interpolation function is used. The type of interpolation function depends on the shape of the element. The interpolation function is often a polynomial, where the degree depends on the number of nodes in an element.
3) **Deriving the finite element equations**
   There are two popular methods of deriving the finite element equations, the variational approach and the Galerkin approach [5]. The Galerkin approach is a special case of the method of weighted residuals, and is the most popular due to its greater generality [5]. Boundary conditions are also imposed.
4) **Assembling the element equations**
   Assembly is the process of adding all the element matrices together to form the system matrix [5]. For a model with m unknown nodal values, the system matrix will be an $m \times m$ matrix. The connectivity of the elements is used to assemble all the equations together to a linear system of equations $Ax = b$, where A is called the system matrix, or the stiffness matrix.
   The system matrix will be square, symmetric, singular and in most cases sparse. The sparsity of the matrix depends on the shape of the elements and the connectivity of the nodes. Normally, each node is only connected to the neighboring nodes in the mesh, so most terms in the system matrix will be zero. Some cases can have nodes connected to non-neighboring nodes (for example models using radiative heat transfer) which results in a denser system matrix.
   Assembling the system matrix is one of the most memory-intensive steps when computing a solution. Denser system matrices will require much higher memory. The number of degrees of freedom and the sparsity of the system matrix will determine the memory requirement of a model.
5) **Solving the system of equations**
   The matrix system is solved using either direct or iterative methods.

B. Parallel Computing and Speedup

Depending on the problem, scientific computing is done on everything from personal computers to supercomputers. More powerful computers and more efficient algorithms allows scientists to solve larger and more realistic problems. Supercomputers often use parallel processing, so the development of parallel numerical methods has been an important area of research.

The *speedup* is defined as the ratio of the time T_1 used by one processor, to the time T_P used by P processors.

$$SU = \frac{T_1}{T_P} \tag{1}$$

The speedup for partially parallel problems is determined by how much of the problem can be parallelized.

Let f be the part of the computation, by time, that must be done serially [6]. Then $(1 - f)$ is the amount of time that can be completely parallelized. The run time when using P processors is then

$$T_p = f + \frac{(1-f)}{P} \tag{2}$$

The speedup SU is then

$$\begin{aligned} SU = \frac{T_1}{T_p} &= \frac{1}{f + \frac{(1-f)}{P} \cdot P} \\ &= \frac{P}{f(P-1)+1} \end{aligned} \tag{3}$$

Expression 3 is known as Amdahl's law, and it gives the theoretical maximum speedup for a job. Amdahl's law is plotted in figure 2.

The gained benefit of using a supercomputer is not always as large as one would expect. For high values of f, increasing the number of processors is not going to increase the speedup. Even a slight percent serial part will limit the maximum speedup greatly. If the computational resources goes toward infinity, a 95% parallel job still only has a maximum theoretical speedup of 20.

Problems can be classified according to the degree of parallelism:

- **Embarrassingly parallel problems** are problems which are easy to separate into a number of parallel tasks.

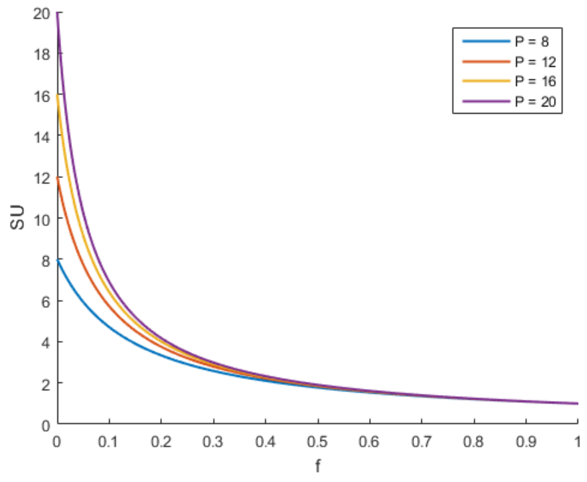


Fig. 2: Amdahl's law

There is little communication between the tasks, and no dependencies between them. A *Parametric Sweep* in *COMSOL Multiphysics* is embarrassingly parallel. Each simulation is completely independent of the others. Of course, each simulation in the sweep has its own degree of parallelism.

- **Partially parallel problems** are problems where part of the problem is parallelizable. All FEM-models are at least partially parallel, because solving a system of linear equations is a partially parallel problem.
- **Inherently serial problems** are impossible to divide into a number of parallel tasks, because they all depend on each other. A good example of this is time dependent studies, where initial values for a time step n is given by time step $n-1$. It is not possible to parallelize the time step simulations, so the speedup be determined by the possible speedup of a single time-step simulation.

For inherently serial problems, throwing more computing power at the problem will gain you little in speedup. In some cases, it might even take longer to solve a problem with more resources. When more processors are added to a parallel job, the computational task for each becomes smaller, and the communication overhead increases. Communication overhead is the time *not* spent on running the job, but on things like distributing the job, collecting the results and communication between nodes. Figure 3 shows how the speedup decreases because of communication overhead.

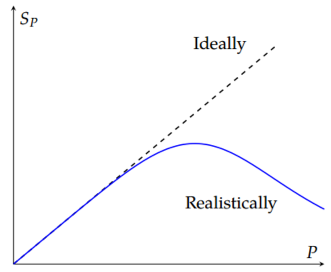


Fig. 3: Ideal speedup vs. realistic speedup [7]

TABLE I: Some terminology regarding supercomputers

Core	The basic unit that performs calculations.
Processor	A processor is the central processing unit in a computer. It contains cores and supporting hardware. A processor can have more that one core.
Node	A physical self contained computer. Each node on Vilje consists of two processors, with 8 cores each.
Cluster	A connected group of computers(nodes), working together.

There are two ways to benefit from parallel computing. With more computational resources one can split a problem into smaller subproblems, reducing the computation time. This is called strong scaling (speedup). Weak scaling, on the other hand, is keeping the work-per-node constant, while increasing the number of nodes. Weak scaling allows you to solve larger problems in a constant amount of time.

C. *Supercomputers (Vilje)*

The term "supercomputer" is changing as computers are rapidly improving. A normal smart phones today could be considered a supercomputer 20 years ago. Some terminology regarding supercomputers is given in table I.

The famous Moore's law (1965) states that the number of transistors per chip doubles every two years. This law held true for many years, but in recent decades the law has been stalling.

In order to increase performance in the post-Moore era, the industry has been moving towards using parallel processing, making specialized chips and reconfigurable chips.

The national e-infrastructure for universities in Norway consists of four supercomputers, "Abel" in UiO, "Vilje" at NTNU, "Hexagon" at UiB, and the new supercomputer "Fram" at UiT. "Fram" was installed at UiT spring 2017, and it will eventually replace Vilje and Hexagon. The new supercomputer is currently rated as one of the worlds top 200 supercomputers [8]. The lifetime of "Fram" is estimated to be four years.

Uninett Sigma2 is responsible for the operation and maintenance of the e-infrastructure for universities.

III . Running Cosmol Multiphysics in Parallel

In this chapter the parallel capabilities of COMSOL are looked into. A detailed explanation of how to run COMSOL on a cluster is presented.

COMSOL Multiphysics is a flexible finite element analysis software geared towards research purposes. The interface makes it easy to set up complex problems without extensive knowledge about the underlying mathematics or physics.

COMSOL Multiphysics has two modes of parallel operation:

- **The shared memory model**
  When running on a personal computer, COMSOL uses shared-memory parallelism. On personal computers all the cores have access to the same memory, and the communication between the cores is as fast as the memory access. COMSOL will by default use all the available cores available.

- **The distributed memory model**
  When running COMSOL on a Linux or Windows cluster consisting of several computers, the distributed memory model is used. The nodes in a cluster do not share the same memory, and the speed of communication between nodes will depend on the physical distance between them. COMSOL uses MPI for node-to-node communication on a cluster. MPI the dominating message passing protocol for parallel computing.

Both memory models are combined to give the best performance. When you start a COMSOL job on a cluster, COMSOL will use the shared memory model within each node, and MPI to communicate between nodes. It is also possible to use MPI (distributed mode) to communicate between the cores within a CPU, but this will be slower than using the shared memory mode.

An advantage of COMSOL Multiphysics is that is will set up and manage the communication between nodes automatically. Little knowledge about parallel computing is required to use it, but in order to get the most out of distributed mode, the model should be set up with parallelism in mind. Choosing the right solver is important in order to best utilize a computing cluster.

A. *Solvers*

There are two main methods to solve systems of linear equations: direct and iterative methods. Direct methods require a fixed, deterministic number of steps to produce a solution. Gaussian elimination and LU factorization are some examples of direct methods. Iterative solvers improve on an initial guess for each iteration. The process can be repeated until the residual Ax-b sufficiently close to $\vec{0}$.

Direct solvers are very robust and will work on most problems, but iterative methods are usually more efficient both in time and memory consumption. Because of this, the default solver uses a direct method only for 2D problems, and for smaller 3D problems.

The initial sparseness of the system matrix will not be maintained when using a direct solver. Many of the zero terms will become non-zero during the solution process [10]. This is called "fill-in", and it is undesirable because it increases the memory requirements and the number of arithmetic operations. Different strategies can be used to reduce the fill-in.

Iterative methods can be challenging to set up for complex multi-physics models, as they are less robust. There is a trade-off between robustness of a solver and their time and memory requirements. The fastest iterative solvers are least robust, and do not work for all cases.

There are many iterative solvers available in COMSOL, but they are all similar to the conjugate gradient method.

Iterative methods rely on good preconditioners to be efficient. A widely used preconditioner is the geometric multigrid (GMG) technique, which can handle a large class of problems [**?**].

Preconditioners can use more time and memory than the iterative solver itself [11].

There are many iterative solvers to choose between in COMSOL, but two of the most widely used are multigrid methods and domain decomposition.

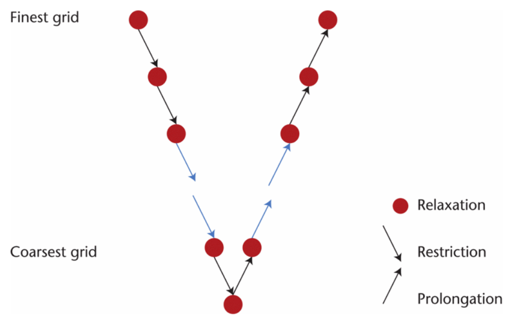


Fig. 4: A schematic description of the full multigrid algorithm [12]

Multigrid Solvers
Basic iterative methods remove the high frequency errors quickly, but use a long time to remove the low frequency errors. The multigrid method is based on the observation that if a problem is transferred to a coarser grid, the highest frequency errors disappear and the low frequency errors turn into higher frequency errors. Transferring the problem to a coarser grid is called *restriction*.

The first step in multigrid methods is to remove the high frequency errors with a basic iterative method, giving the solution x_l. The residual

$$r = b - A \cdot x_l \tag{4}$$

of the solution is computed, and the system is restricted from a mesh with size h to a mesh with size $2 \cdot h$.

The solution on the mesh $n \cdot h$ is then prolongated up to the mesh $(n/2)h$ by interpolation.

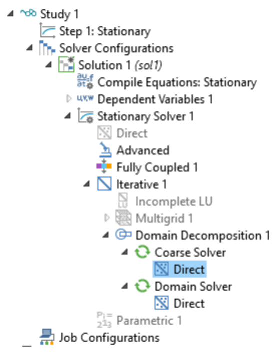
In order to solve the residual on the coarser mesh, the system can again be transferred to a coarser grid. This can be repeated recursively until the system is small enough to solve with direct solvers. This recursive scheme is called the multigrid V-cycle.

Multigrid methods are popular because of their rapid convergence rate. The computational work increases linearly with the number of unknowns.

Domain Decomposition
Domain decomposition works by dividing model domains into sub-domains, and solving the problem for each sub-domain. The total solution is then found by iterating between the computed solutions for each sub-domain, and using the current neighboring sub-domain solutions as boundary conditions [11].

To solve each sub-domain, a "Domain solver" is used. In COMSOL, MUMPS is the default domain solver, but there is a wide range of solvers to choose between. The idea is to divide the the domains into small enough pieces that direct solvers are efficient. Domain decomposition is combined with a global coarse solver to accelerate convergence [13]. Figure 5 shows the domain decomposition tree in COMSOL, with the coarse solver and the domain solver nodes.

Domain decomposition is especially useful for parallel computing, as each sub-domain problem is independent of the others. The default method is the overlapping Schwartz method, where the domains overlap by more than the interface.



```
Study 1
  Step 1: Stationary
  Solver Configurations
    Solution 1 (sol1)
      Compile Equations: Stationary
      Dependent Variables 1
      Stationary Solver 1
        Direct
        Advanced
        Fully Coupled 1
        Iterative 1
          Incomplete LU
          Multigrid 1
          Domain Decomposition 1
            Coarse Solver
              Direct
            Domain Solver
              Direct
        Parametric 1
  Job Configurations
```

Fig. 5: A study tree in COMSOL 5.2 using the "domain decomposition" solver

All iterative solvers (except Incomplete LU) in COMSOL are parallelized. Setting up the solvers to maximize the speedup can be challenging.

From the COMSOL blog [**?**]: "It is also fair to say that setting up and solving large models in the most efficient way possible is something that can require some deep expertise of not just the solver settings, but also of finite element modeling in general."

B. Meshing in Parallel

The free mesher in COMSOL runs in parallel both in shared memory mode, and in distributed mode.

The free mesher starts by meshing the faces in a model, and then moving on to the interior volumes of the domains. After the faces between two domains are meshed, the domains can be meshed independently of each other, and the two jobs can be distributed on the available cores.

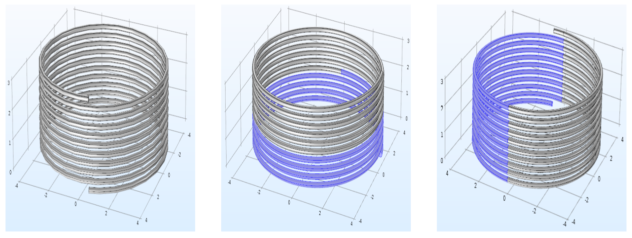
The free mesher will distribute the domains automatically, but it can not divide a domain into sub-domains in order to parallelize the job further. If there is only one domain in the model, which can be the case for some imported CAD-models, there will be limited speedup by using more processors.

Reducing the meshing time by partitioning

In order to parallelize the meshing, the domains can be partitioned manually. To do this, add a *work plane* to the model. A *partition objects* geometry operation can then be used to partition the domain with the work plane. If you only want the partition to affect the meshing, and not the geometry, the geometry operation *Virtual opertions* is useful. It allows you to partition a model only for meshing purposes, without influencing the physics settings of the model.

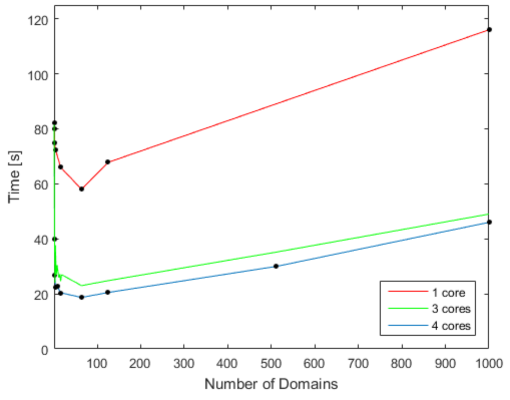
To test the effects of partitioning on meshing time, two simple models were created.

1) Model "Square" consist of a simple 3D-box geometry. The mesh was set to extremely small.

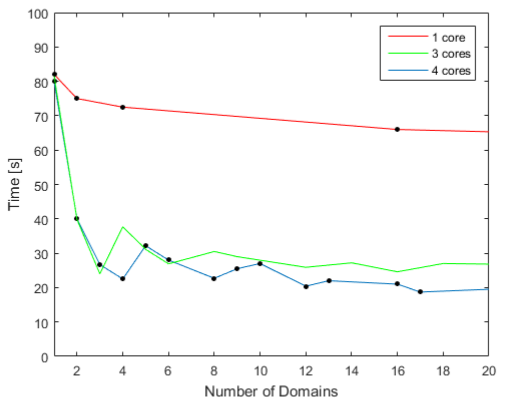


(a) A coil before partitioning (b) The coil after partitioning into 2 domains (c) The coil after partitioning into 20 domains

Fig. 6: A coil geometry before and after partitioning with work planes



(a) Large scale



(b) Small scale

Fig. 7: Meshing time for the model "Box" as a function of number of domains in the model

2) Model "Coil" consists of a simple coil with 10 windings. The mesh set to "Free tetrahedral", and the size to extremely small. See figure 6.

The models were meshed with a different number of partitions to investigate the speedup. The simulations were run on my laptop, which has an Intel Core i7-3720QM CPU with 4 physical cores, and 8 GB of memory. The results are plotted in figure 7 and 8.

In figure 7 b) is is possible to see that a parallel job is most efficient if the number of jobs is divisible by the number of cores. The lines for 4 cores (blue line) and 3 cores

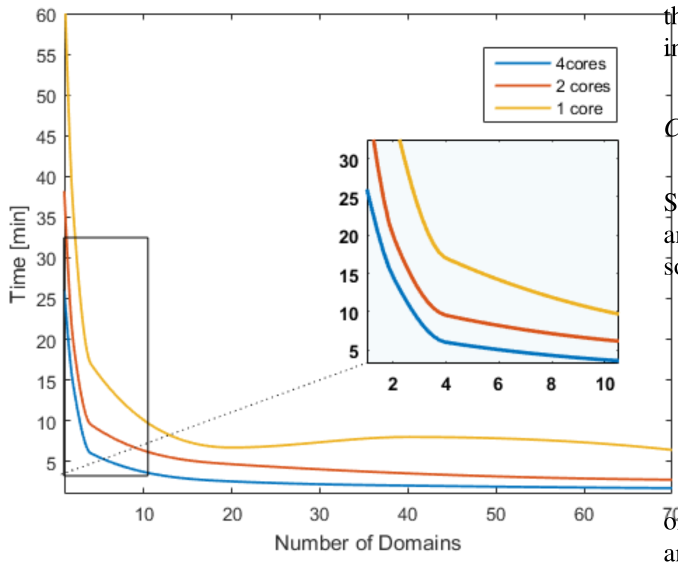


Fig. 8: Meshing time for the model "Coil" as a function of number of domains in the model

TABLE II: Meshing time for model "Coil" for a different number of meshing-nodes in the meshing sequence

20 Domains	20 "Free-tetrahedral"-nodes	5.4 minutes
	10 "Free-tetrahedral"-nodes	4 minutes
	1 "Free-tetrahedral"-nodes	2.5 minutes

(green line), have local minimums at [4,8,12] and [3,6,12] respectively. This effect becomes smaller and smaller as the size of the jobs decrease. For one core, there is a very small speedup in the meshing time when partitioning the model. When the number of domains goes over 100, the meshing time increases. According to COMSOL support, this is because of the increased number of faces in the model.

For the coil geometry, one can see in figure 8 that there is a very good speedup even for the single core case. According to COMSOL support, this is because the faces of the coil have a complicated surface parametrization, which means that the majority of the meshing time is spent generating the surface mesh. When the coil is partitioned the resulting faces get less complex, and the surface meshing time decreases.

COMSOL will mesh a meshing-sequence serially in the order they are listed. This means that if there is, for example, several "Free Tetrahedral" nodes, only one of them will be meshed at a time. As table II shows, the meshing time increases when there are more mesh-nodes in the meshing sequence.

If you are setting up a parametric sweep that only changes one part of the geometry, it is a good idea to put the meshing-node for that part at the bottom of the list. COMSOL will start at the top of the meshing sequence, and check if the geometry corresponding to that mesh-node has changed. If it has, that node and all the nodes under it will be meshed again.

Partitioning the mesh is perhaps more important for models that are being run serially, like in time studies or for some optimization algorithms. Saving meshing time in each step can give a good reduction in the total simulation time. However, this is only important if there are large or complex domains in the model.

C. Running COMSOL on a Cluster

In this chapter, a detailed explanation of how to run COMSOL on a cluster is given. The speedup of single simulations and parametric sweeps is tested and discussed. In the end, the solutions for some common problems are given.

To run COMSOL on Vilje/Fram, there are 4 basic steps:

- Prepare the model
- Create/edit a job script
- Move the files to the cluster, and submit the job
- Move the resulting output file to your personal computer

In addition to this, before you run COMSOL the first time on a cluster you need to log in to COMSOL with a username and password. To do this, write "load COMSOL/5.2", and then start COMSOL. You will be asked for a username and password. After this, press Ctrl+c to close the program. It is important to quit COMSOL because you are not supposed to run programs on Vilje directly. Luckily, you only have to do this once.

1) Preparing the model: Under the "Study" node in COMSOL, there are several options for cluster computing. These are "Batch", "Batch Sweep", "Cluster Computing", and "Cluster Sweep". These nodes are all mainly for running COMSOL on a cluster remotely from your computer, by simply pressing the compute button. However, it is not allowed to run jobs directly on Vilje, so I have not used this functionality. To run a COMSOL job on a cluster, it is *not* necessary to add any of the above study nodes. It is mentioned because this can cause some confusion.

Setting up parametric sweeps
If you are running a large parametric sweep, with hundreds or thousands of simulations, storing all the solutions in a file can take up a lot of RAM. If the file grows too large, it can be cumbersome to work with the file on a personal computer afterwards. To avoid this, use probes to measure the relevant variables for each simulation. Create the probes you need by right-clicking on "Component 1 -> Definitions -> Probes".

In the settings for the parametric sweep, check the box for "Accumulated probe table". COMSOL will collect all the probe values during the sweep, and store them in a table. If the sweep is large, it can be necessary to change the maximum number of rows in the table settings, as the default is 10 000 rows.

To avoid storing all the solutions, go to "Output While Solving" and set "Keep solutions in memory:" to "Only last".

Note: When setting up a parametric sweep, remember to check the "Study Extensions -> Distribute parametric sweep" box.

COMSOL will by default stop a parametric sweep if one of the simulations produces an error. To disable this, go to "Study -> Job Configurations-> Parametric Sweep -> Error" and uncheck the "Stop if error"-box. If a simulation fails, it will simply move on to the next simulation in the sweep.

Before running a COMSOL model on a cluster, there is a few things you can always do to increase the chance of the simulation running smoothly.

- If possible, check that everything works correctly by running a simulation (on a coarse mesh perhaps).
- Press "File -> Compact History"
- Clear the mesh by right-clicking on the mesh and selecting "Clear Mesh". Having a partially meshed model can result in a "Domain already meshed"-error.
- Clear all solutions

2) Setting up the job script: To run a job on Vilje, it must first be submitted to the job scheduler. The scheduler controls which jobs runs on which nodes at any given time. The scheduler will

- Put the job in a queue
- Allocate resources and decide which job to start
- Report back any output and error messages from the job

The order in which the jobs are started depends on how the jobs are ordered (in the queue), how many resources are available, and how many resources the jobs are requesting. On Vilje, the scheduler PBS is used.

To submit a job to the queue, the user must tell the scheduler how many nodes they request, and how long time they estimate their job will take. This is done by creating a *job script*. A job script is a text file containing information for the scheduler, and the commands you want to run.

Running a "normal" COMSOL job without any interaction is called running a *batch job* on a cluster. From my experience, the batch job is the most reliable and stable way of running COMSOL on Vilje. The batch command will take in a file name, run all the studies in the model, and store the solution.

Figure 9 gives an example of a job script for running a batch job. The lines starting with *#PBS* contains information for the scheduler. The line

```
#PBS -l select=2:ncpus=32:mpiprocs=2:ompthreads=8
```

tells the scheduler how many nodes and processors the job is requesting. Change "select" from 2 to the number of nodes you want. "ncpus" determines how many cores per node you are given. If you ask for 1 node, the scheduler will reserve the whole node for your job. That means that there is rarely any point in requesting less than all the cores, as nobody else will be able to use them.

"mpiprocs=2" sets the number of MPI-processes per node. Each node on Vilje has two physical processors which do not share memory, so "mpiprocs" should always be at least 2. If you are running a parametric sweep, this number can be increased depending on how you want to distribute the sweep.

Line 18 in figure 9 starts the simulation. There are two flags which will determine how COMSOL is set up on the nodes:

1) **-nn 20:** The -nn flag tells COMSOL how many COMSOL-processes to start in total. If you are running a batch job without a parametric sweep, set this number to $2 \cdot N$, where N is the number of nodes you are requesting.
2) **-np 8:** This flag tells COMSOL how many physical cores *each* COMSOL process can use. In this case, there are two COMSOL processes per node, and 16 cores per node, so -np is set to $\frac{16}{2} = 8$.

```
1  #!/bin/bash
2  #PBS -N job_name
3  #PBS -A project_account
4  #PBS -l select=2:ncpus=32:mpiprocs=2:
       ompthreads=8
5  #PBS -l walltime=24:00:00
6  #
7
8  module load comsol/5.2
9
10 cd PBS_O_WORKDIR
11
12 w=/work/$PBS_O_LOGNAME/some_folder
13 if [ ! -d $w ]; then mkdir -p $w;fi
14
15 cp model.mph $w
16 cd $w
17
18 comsol -nn 20 -np 8 -clustersimple -mpiarg
       -rmk -mpiarg pbs -f nodefile batch
       -batchlog log.txt -inputfile $model.mph
       -outputfile out.mph
19
20 cp log.txt $PBS_O_WORKDIR
21 cp out.mph $PBS_O_WORKDIR
```

Fig. 9: An example job script for running a COMSOL batch file

The documentation for the rest of the commands in line 18 can be found in the chapter "Running Comsol-> Running COMSOL in parallel -> The Comsol Commands" in the reference manual [11].

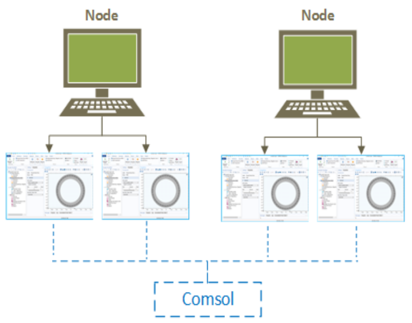
When you are running a parametric sweep on a cluster, the number of COMSOL-processes will decide how many simulations in the sweep are run simultaneously. The parametric sweep will be distributed over the COMSOL-processes, so the number of simultaneous simulations is equal to the $-nn$ value. If you want the parametric sweep to be as fast as possible, set -nn equal to the length of the sweep.

NOTE: The number of MPI processes should be equal to the number of COMSOL-processes per node. Set "mpiprocs" to $\frac{-nn}{N}$. Each COMSOL-process needs a communication interface.

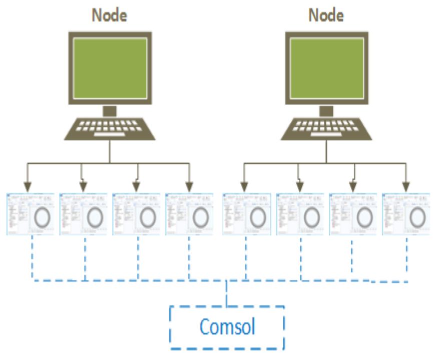
The size of the simulations should decide how many COMSOL processes you start per node. If the simulation is small, you can run many simulations in parallel on one node. For example, if one core is enough for one simulation, you can solve 16 simulations simultaneously per node by setting $-nn$ to $16 \cdot N$, and $-np = 1$. Figure 10 shows an example of two different configurations.

3) Submitting and checking a job:

1) Move your COMSOL file to a folder on your home area on Vilje using WinSCP. See the appendix for some information on WinSCP and puTTY.
2) Create or tweak a job script, and put it in the same folder as your COMSOL file.



(a) COMSOL started with nn=4, np=8



(b) COMSOL started with nn=8, np=4

Fig. 10: Two different ways to distribute a parametric sweep. In figure a), 4 simulations in the sweep are started simultaneously, with each simulation running on 8 cores

3) Log into Vilje with PuTTY and navigate to the correct folder. To submit your job to the queue, type "*qsub yourjobscript.pbs*". The job should now be submitted to the queue.
4) When the job is submitted, you can check the status with *qstat -u username*.
5) When your job is finished, there should be two new files in the folder. One output and one error file, which are named something like jobname.o/e0455732123. If you are using *-batchlog log.txt* in the job script the error file will be empty, as everything is outputted to the log file.
6) If the job finished successfully, use WinSCP to move the COMSOL .mph file back to your own computer. The file should contain a solution.

D. Testing the Speedup of Comsol Models

So what kind of speedup can you expect from COSMOL Multiphysics?

According to [14], for large COMOSL models the maximum speedup is normally in the range of 8 to 13. Generally, models requiring large amounts of memory have better speedup [14] [11]. The speedup is also largely dependent on the mesh, the solver configurations, and the physics.

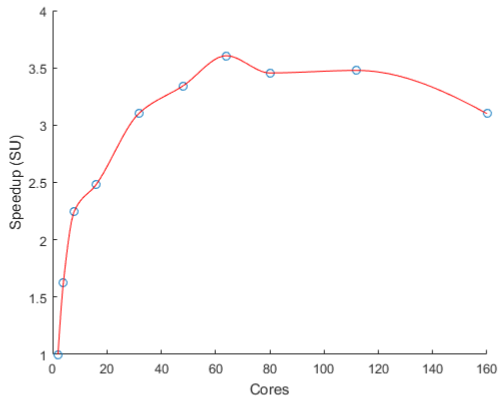


Fig. 11: Speedup for the model "Electronic Enclosure Cooling"

Nodes	NN	NP	Cores	Time
1	1	2	2	11083
1	1	4	4	6808
1	1	8	8	4929
1	2	8	16	4468
2	4	8	32	3567
3	6	8	48	3316
4	8	8	64	3074
5	10	8	80	3207
7	14	8	112	3187
10	20	8	160	3575

TABLE III: Simulation times for the model "Electronic Enclosure Cooling"

1) Speedup of Models: To investigate the speedup, the model "Electronic Enclosure Cooling" from the application library was run on a different number of nodes. The model uses the "Heat Transfer Module" to study the thermal behavior of a computer power supply unit [15]. The model uses the geometric Multigrid solver, and it has 1,2 million number of degrees of freedom.

The speedup is plotted in figure 11. The maximum speedup for this model was found to be around 3,5. The speedup is quite bad, and one of the reasons for this might be that the model is too small.

Some speedup tests were also run on a simple model, "Box.mph". The model consist of a simple 3D box, partitioned into 100 pieces. The mode was run with two different solvers, geometric Multigrid, and MUMPS. It was run for two different mesh sizes, resulting in a DOF of 2,8 and 7,9 million.

It is clear from figure 12 that the iterative solver performs much better than the direct solver. The speedup of the iterative solver was not better than for the direct solver, and the speedup was very small in both cases.

2) Speedup of Parametric Sweeps: To test the speedup of parametric sweeps, a model by my fellow student Charlie Bjœrk [16] was used. A parametric sweep of length 180 was added to the model. The sweep takes approximately 1 hour on my personal computer. The sweep was run in different configurations on Vilje.

Figure 13 shows the speedup of the sweep. The speedup

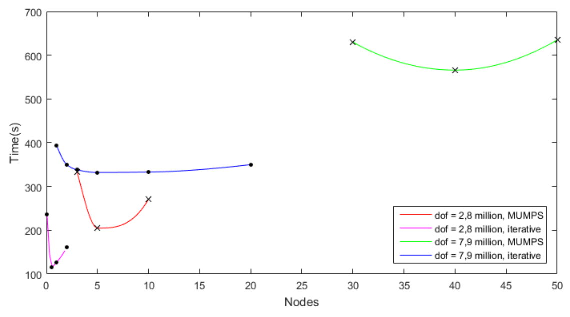


Fig. 12: Simulation times for the model "Box", for different solvers and mesh sizes

TABLE IV: Simulation times for the parametric sweep

Nodes	NN	NP	Time
1	1	8	79 minutes,21 seconds
12	90	2	6 minutes
12	180	1	6 minutes, 57 seconds
6	45	2	x 7 minutes, 25 seconds(?)
23	180	2	5 minutes,54 seconds

is linear in the beginning, then it flattens out as the number of COSMOL-processes reaches the number of simuations in the sweep. Running a parametric sweep in COSMOL is a embarrassingly parallel problem, and the speedup is excellent.

3) Common Problems with Distributed COSMOL: Some of the errors I have experienced with COSMOL, and the solutions to them, are listed below.

- **Problem: Simulation takes unreasonable long time.** When using COSMOL 5.3, often COSMOL would not stop running after finishing a simumlation. It would instead keep "running" until the job exceeded the wall-time. If the top-command shows that all the COSMOL-processes are using 100
- **Problem: Error using the Mphserver on Vilje** It is not possible to use the mphserver on more than one node when using COSMOL 5.2, as there is an bug in the version. Often the error messeage will say something like "Inconsistent numbers of degrees of freedom". According to COSMOL Support, one should use COSMOL 5.3 to fix this. However, I did not manage to get the COSMOL 5.3 Mphserver to run smoothly on Vilje. I you can, use the batch mode.
- **Problem: The probe table is empty after a parametric sweep** Solution: Delete the solver configurations node, and the study-node. Add a new study, the required nodes, and a parametric sweep. **Don't** run a stationary or time-dependent study before you add the parametric sweep. Run a short parametric sweep. Check that a "Parametric Solutions" node shows up under "Solution 1 –> Solver Configurations".
- **Problem: No username detected**
This errors shows up if you accidentally delete your login information in the .comsol folder, or the first time you run a job if you have not created a user.
- **Disk quota exceeded**

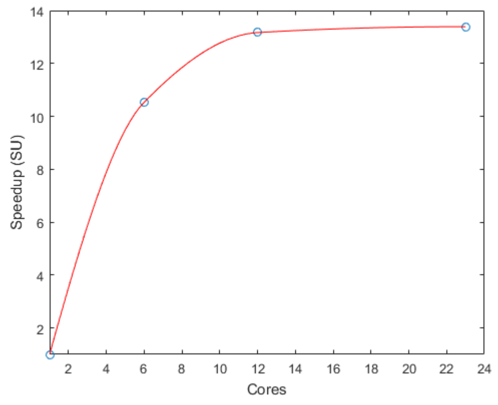


Fig. 13: Speedup a parametric sweep of length 180

Delete some of the hidden COMSOL files on your home area. The files in .comsol/v52/configurations and .comsol/v52/workspace can be deleted.
- **Problem: No batch sweep detected** Sometimes, if you have messed around with the job configurations before adding a parametric sweep, the parametric sweep node will not appear under job configurations. Try deleting the configurations, and creating it again by right-clicking job configurations and selecting "Show default Solver". If this does not work, try deleting configurations, then run a single simulation in the sweep, and try creating it again.

IV . OPTIMIZATION

Optimization is the practice of finding the *best* solution to a problem from all feasible solutions. In most cases, finding a *good enough* solution is often sufficient. How extensively a problem can be optimized is a function of the cost and time of the optimization process. An optimization problem can be expressed as

$$min\, f(x), \qquad x \in \mathbb{R}^n$$

Subject to the constraints

$$G_i(x) > 0$$
$$G_i(x) = 0$$

Where n is the number of control variables, $G_i(x)$ is a set of constraints, and $f(x)$ is the objective function to be minimized. In the case of optimizing a COMSOL model, the control variables can for example parameterize the geometry and materials.

There are two main categories of optimization methods, gradient-based and gradient-free methods. They can also be labeled as stochastic, deterministic or metaheuristic [17].

Table **??** gives some advantages/disadvantages of the two different methods. Gradient-based techniques require the function to be smooth and differentiable. They converge quickly to optima, but can get stuck easily in local optima. Global methods converge slower near optima, but does not get trapped in local optima.

In 1996 and 1997 David Wolpert and William Macready presented the "No Free Lunch" theorem. They observed that the performance of all search algorithms, averaged over all possible problems is exactly the same [18]. This includes blind-guessing. No algorithm is universally better than another on average. This means that there is no search algorithm that always perform better than others on any class of problems. Instead of looking for the best general optimization algorithm, one must look for the best optimization algorithm for the specific problem at hand. Much research has been done the last decades on finding the strengths and weaknesses of different optimization algorithms, and for which problems they excel.

Hybrid algorithms takes advantage of both local and global optimization methods. Hybrid alogitms use a global method to find the area of the global optimum, and then switches to a gradient-based method to find the optimum with fast convergence.

A. *Metaheuristics*

Metaheuristic methods can not guarantee that the solution is the best solution, but are useful as they make few assumptions about the problem, and can handle very large search spaces. Metaheuristics can also solve a wide range of "hard" optimization problems without needing to adapt to each problem [19]. A metaheuristic is useful when a problem can not be solved with an exact method within a reasonable time, and are used in many different areas, from engineering to finance. Metaheuristics can solve problems with several control variables. Metaheuristic methods are very often inspired by nature, and often use random variables.

Metaheristic methods have gained more and more attention the last 30 years.

"The considerable development of metaheuristics can be explained by the significant increase in the processing power of the computers, and by the development of massively parallel architectures. These hardware improvements relativize the CPU time costly nature of metaheuristics." (Boussaid, Lepagnot, Siarry) [19].

Two important processes decide the success of a metahuristic, *exploration* and *explotation*. Exploration is a global sampling of the search space, with the purpose of finding areas of interest. Explotation is the local sampling in these areas in order to close in on the optimal solution. The main difference between metaheristics is how they balance between exploration and explotation [19].

Metaheuristic methods can be divided into single-solution and population based methods. Single-solution based metaheuristics starts with a single point in the search space and moves around, they are also called "trajectory methods". The most "popular" trajectory method is Simulated Annealing. Simulated Annealing was proposed by Kirkpatrick et al. in 1982 and is inspired by annealing in metallurgy. The objective function is thought to be the temperature of a metal, which is then lowered to a state of minimal energy.

Population based meta-heuristics initializes a set of solutions, and moves these through the search-space. One main advantage to population-based methods over trajectory methods is that they can be implemented in parallel. Evolutionary Algorithms (EA's) and Swarm Intelligence (SI) are the most studied groups of population based methods. EA's are based on Darwin's evolutionary theory, and contains, among others, Evolutionary programming and Genetic Algorithms (GA).

Swarm Intelligence algorithms are methods inspired by the social behavior of insects and birds. Typically the individuals will be simple, and need to cooperate to find a good solution [19]. Together the individuals can perform complex tasks, similar to ants and bees. The most studied SI's are Particle Swarm Optimization (PSO) and Ant Colony Optimization. Some other examples are Bacterial foraging optimization, Artificial Immune Systems and Bee Colony Optimization.

In these algorithms the solutions are imagined to be points in a n-dimensional room where each control variable represents one dimension. The solutions (called particles or insects depending on the algorithm) can then fly or crawl through this multidimensional room, often called the *search space*.

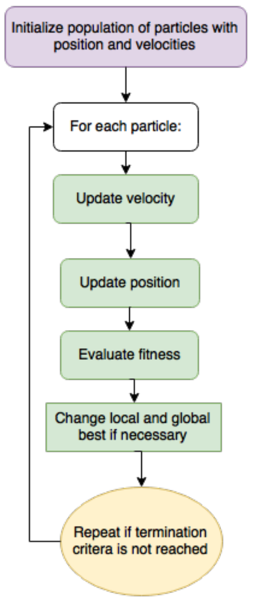

Initialize population of particles with position and velocities
For each particle:
Update velocity
Update position
Evaluate fitness
Change local and global best if necessary
Repeat if termination critera is not reached

Fig. 14: Overview of basic PSO

1) PSO: PSO was introduced by Kennedy, Eberhart and Shi in 1995 as a way to simulate the social behavior of flock of birds or a school of fish. An overview of the generic PSO algorithm can be found in figure 14. PSO is being used on many different problems, from reactive power and voltage control to human tremor analysis [18].

The algorithm works by first initializing a random group of solutions, called *particles*. The values of the decision variables is the position of the particle. Each particle will have a velocity and a position in the search space. For each iteration (called a time step) the particles will move to a new position, and evaluate the fitness of the solution. How a particle moves is dependent on its own memory of the best found position, and the memory of the neighborhood particles. The neighborhood size can vary from a few to all of the particles in the swarm, depending on the implementation.

The best known position found by all the particles is called *global best*, and the best position found a single particle *local* *best*. The particles will move based on its own experience, and the experience of its neighbors.

As a particle is drawn towards another, it might discover new better regions, and will start to attract other particles, and so on. The particles will be drawn towards and influence each other similar to individuals in a social group. Interestingly, if looked on as social individuals, one of the driving factors of the behavior of the particles is confirmation bias. From [**?**] "...individuals seek to confirm not only their own hypotheses but also those of their neighbors".

Two equations are updated for each particle and time step; velocities and positions.

The position is given by:

$$x_p^{i+1} = x_p^i + v_p \tag{5}$$

Where x_p^i is the position vector for the previous time-step, and V_p is a velocity vector. The velocity is given by:

$$\begin{aligned} v_p &= v_p + c_1 r_1 (p_l - x_p) + c_2 r_2 (p_g - x_p) && (6)\\ &= v_{current} + v_{local} + v_{global} && (7) \end{aligned}$$

The new velocity is the sum of three vectors, the current velocity, the new velocity towards the global best position and the new velocity towards the local best position. r_1 and r_2 are normally distributed random numbers $r \sim U(0,1)$. The local and global best positions are given by p_l and p_g. Often a V_{max} parameter is defined, such that if $v_p > V_{max}$ then $v_p = v_{max}$. This is to limit the velocity.

c_1 and c_2 are called cognitive and social scaling parameters [20]. These parameters determine the balance between exploration and exploration. c_1 and c_2 determines how much the movement of the particle will be affected by the local and global best found positions respectively. A large $\frac{c_2}{c_1}$ ratio will restrict the search space quicker and shift the balance towards exploration, as the particles will be pulled fast towards the global best known position. A small ratio means that the balance will shift towards exploration. The values chosen for c_2 and c_1 will effect the convergence significantly.

A parameter that can improve convergence was introduced by Shi and Eberhart [21] in 1998. Adding an *Inertial weight* to the particles gives increased control of the velocity. With inertial weight the expression for velocity becomes:

$$v_p = w \cdot v_p + c_1 r_1 (p_l - x_p) + c_2 r_2 (p_g - x_p) \tag{8}$$

Where $w \ \ [0,1]$ is the weight. w determines how much the particle accelerates or deaccelerates. A large inertial weight will give the particles higher speed and result in a more global search, while a small w makes them search locally [**?**]. The velocity need to be large enough that particles can escape local optima. The benefit of adding inertial weight is that the velocity can be changed dynamically, controlling the transition from exploration to exposition.

Many different dynamic Inertia Weight strategies have been introduced. *Linear Decreasing Weight* will simply decrease the inertial weight for each time step. As the particles have explored the search room, and are closing in on the global maximum, the speed is reduced, moving from a global search to a local search. *Random Inertial Weight* will simply speed up or slow down the particle randomly. [20] found in experiments that random inertia weight was the most efficient strategy (least iterations) for a handful of benchmark problems, while linear decreasing weight gave near optimum results compared to other methods, but required a very high number of iterations.

Random inertial weight is given by

$$w = 0.5 + \frac{rand()}{2} \tag{9}$$

Where $rand()$ is a normal distributed random number in $[0,1]$.

There is no known best population size to initialize a particle swarm [21]. It is important to have *enough* particles for the algorithm to be effective. The particles need to sample the available solutions early, because the search space will become more and more restricted as the particles move towards the global best position. [22] However, having a large amount of particles will limit the number of time steps owing to limited computational resources. [**?**] recommends, based on experience, using between 10-50 particles.

To achieve quick convergence and a small error, it is necessary to tune the parameters w, c_1 and c_2. Small changes to the parameters can give large changes in the behavior of the particles [**?**]. Choosing the parameters is a optimization problem in itself, and the optimal values will be different for each problem.

A well known usual value is $C_l = C_2 = 2$, $c_1 + c_2 =< 4$ and $w = [0.4 - 0.9]$. Pedersen [23] recommends that $c_2 > c_1$.

The size of the neighbor is often set to be the size of the swarm, that is, all the particles communicate with each other. This is called *starstructure*, and has been found to perform well [18]. The main disadvantage of using only one neighborhood is that the particles can not explore more than one optima at a time, they are all drawn towards the same location. How many optima the objective function has should be considered when deciding the size of the neighborhood. For optimization of electrical motors, it seems intuitive that there are several optima, and so using several neighborhoods might be the best solution.

B. *Optimization in COMSOL*

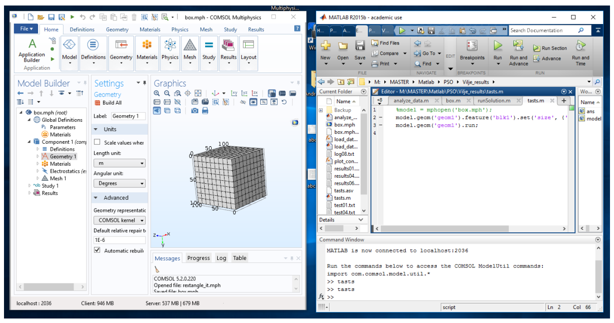
There are several ways to optimize a COMSOL model:

- Using understanding of the model and intuition to tweak the parameters by hand. One can also set up parametric sweeps manually.
- Using the build-in optimization capabilities of COMSOL.
- Using LiveLink for Matlab to implement your own optimization algorithm.

Explain why you do PSO, LIVLINK?

C. *LiveLink for Matlab*

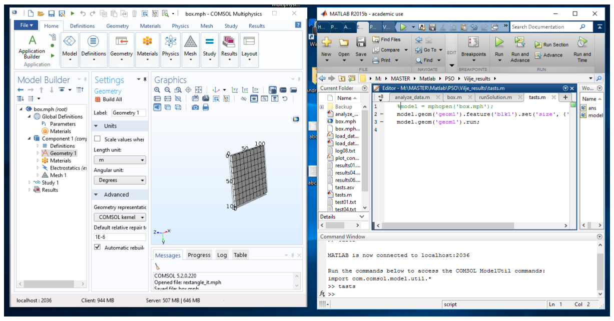
LiveLink for Matlab is a very useful product from COMSOL, that allows you to create, edit and run COMSOL models from Matlab scripts. LiveLink for Matlab is one of COMSOL's "interactive" products, a group of products that allow you to work on COMSOL models through other programs.



```
%model = mphopen('box.mph');
model.geom('geom1').feature('blk1').set('size', {'
model.geom('geom1').run;
```

```
MATLAB is now connected to localhost:2036

Run the commands below to access the COMSOL ModelUtil commands:
import com.comsol.model.util.*
>> tasts
>> tasts
>>
```



```
%model = mphopen('box.mph');
model.geom('geom1').feature('blk1').set('size', {'
model.geom('geom1').run;
```

```
MATLAB is now connected to localhost:2036

Run the commands below to access the COMSOL ModelUtil commands:
import com.comsol.model.util.*
>> tasts
>>
```

Fig. 15: The model "box.mph" before and after running a Matlab script. The figure shows the COMSOL desktop and Matlab side-by side, connected to the same server. COMSOL automatically updates the graphics widow when you change the model in Matlab

LiveLink gives you a lot more control over how you want to set up a model. You can for example use Matlab to run and edit models in a loop, calculate and set geometry parameters, post-process results and so on.

The COMOSL documentation "Introduction To LiveLink For MATLAB" is recommended as a starting point.

When talking about LiveLink, the terms "COMSOL mph-server" and "the COMSOL desktop" is used. The COMSOL desktop is the "normal" desktop program. The COMSOL mphserver works as the interface between COMSOL and Matlab. It is a non-graphical application that can be found it the COMSOL installation folder. The COMSOL mphserver can take in commands from Matlab and execute them.

Connecting to the mhpserver
On Windows, connecting Matlab to the COMSOL mphserver is straightforward. Start the server, and use the command "mphstart" in Matlab to connect to the server. The "LiveLink for Matlab user Guide" in the COMSOL documentation covers this topic nicely.

A nice functionality is the ability to connect the COMSOL desktop to the same mphserver as Matlab. If you are working on a model with LiveLink and want to see the model, you can display the geometry in the COMSOL desktop, see figure 15. This can be done in "File-> Client Server -> Connect to Server", and then "File-> Client Server -> Import Application From Server". The graphics window will then automatically update when you run the Matlab command "model.geom('geom1').run;", which is equal to pressing "Build All" in COMSOL.

When running a Matlab script on a cluster, I found that it is a good idea to run mphstart in a loop. The reason for this is that mphstart will by default try to communicate with the mphserver through port number 2036. If some other process is using this port, the mphserver will connect to the next available port, for example port 2038. If the mphserver is not connected to the default port, Matlab will only be able to connect to it if you specify the correct port number, for example "mphstart(2038)". So if you don't know the correct port, one solution is to simply try different ports until you connect, see figure 16.

```
for port = 2036:2055
    try
        mphstart(port);
        break;
    catch
        s = ['tried port number',
            num2str(port),'
            unsuccesfully'];
        disp(s)
    end
end
```

Fig. 16: An suggestion of how to connect Matlab to the COMSOL mphserver on a cluster

To get a overview of some of the COMSOL commands in Matlab, type "help mli" at the Matlab command line.

One of the best ways to learn using LiveLink for Matlab is to take a COMSOL model, and save it as a Matlab file. Normally it is a good idea to use the function "File -> compact history" beforehand, which removes any redundant history from the model so that the file is easy to read.

The syntax is quite straightforward, the commands in figure 17 will create a model and add a 3D box in the geometry. The first command "mphstart" will connect Matlab with the COMSOL mphserver. The next two lines (line 2 and 3) are always the same, they import the COMSOL class so the COMSOL commands are available.

```
 1 mphstart;
 2 import com.comsol.model.*
 3 import com.comsol.model.util.*
 4
 5 model = ModelUtil.create('Model');
 6 model.modelNode.create('comp1');
 7 model.geom.create('geom1', 3);
 8 model.geom('geom1').create('blk1', '
       Block');
 9 model.geom('geom1').feature('blk1').set
       ('size', {'100' '100' '100'});
10 model.geom('geom1').run;
```

Fig. 17: Example of LiveLink for Matlab syntax

Working with a Model
I found that the easiest way to work with a model through

Matlab was not to save a model as a .m file and edit it, but to use the "mphopen" function, which asks the mphserver to open an existing model. Figure 18 gives an example of the "mphopen" function. Here the model "Box.mph" is loaded by the mphserver. The data in the table with tag "tbl1" is then extracted and returned to Matlab with the "mphtable" function.

When you are working with a model, it is important to use the right tags for the object you want to change. The command "mphnavigate" will open a window in Matlab with all the objects and their tags, but I found it a lot easier to just open the model in COMSOL, and find the tags there. You can find the tags of a study, geometry object, mesh and so on, by clicking the object and going to the "Properties" window.

If you write a Matlab script that interacts with a COMSOL model, make sure that tags you are using in the script still refers to the right objects if you later change the model using the COMSOL desktop.

```
mphstart;
import com.comsol.model.*
import com.comsol.model.util.*

model = mphopen('Box.mph');

str = mphtable(model,'tbl1');
table_data = str.data
```

Fig. 18: Example of LiveLink for Matlab syntax

If you are wondering how to do something, say for example that you want to change a parametric sweep list, you can start by making some change to the list in the COMSOL desktop. If you then save the file as a m. file without using "compact history", the command you want can be found at the bottom of the file. This is often quicker than searching through the documentation for the correct command.

Note: it is always a good idea to run a study in COMSOL before you start working with it in Matlab. Abort the simulation if you want to, the important thing is to start the simulation, because this sets up the "Job Configurations" node.

To run a parametric sweep, the command is "model.batch('p1').run;". "p1" is the tag found in "Study -> Solver Configurations -> Parametric Sweep". So if the "Job Configurations" node is not set up correctly, there might not exist a object in the model with the tag "p1". If "p1" doesn't exist in the model, the run command will give you an error saying it can't find the tag.

When working with models, I found that sometimes the "Job Configurations" node would freeze, and not include any new added studies. If this happens, try to delete the entire study node, and create it again from scratch.

D. Running LiveLink for Matlab on a Cluster

To run the Mphserver with Matlab on Vilje/Fram, you can use a job script that looks something like:

```
1 module load comsol/5.2
2 module load matlab
3
4 COMSOL_MATLAB_INI='matlab -nosplash -
      nodisplay -nojvm'
5
6 comsol -nn 6 -np 8 -clustersimple -
      mpiarg -rmk -mpiarg pbs mphserver
      -silent -tmpdir $w &
7 sleep 8
8 matlab -nodisplay -nosplash -r "
      addpath $COMSOL_INC/mli ,
      yourScript"
```

Fig. 19: An example job script for running the COMSOL mphserver

The & symbol at the end of line 6, tells the Mphserver to run in the background. "Sleep 8" is added because the server takes a little while to start. MATLAB is then started, and asked to run the script "yourScript". The script should contain a "mphstart" command, which will connect the COMSOL Mphserver to MATLAB.

If the Matlab script is small, there is little need to think too much about parallel programming within the script. Any simulations run through the COMSOL mphserver will normally take *much* longer time than running the Matlab script itself. Likely there is more to gain by working on parallelizing the COMSOL model, than from spending time on parallelizing the Matlab script.

COMSOL started out as a GUI-based program, and later batch mode execution was added. The desktop version is more stable, you might quickly run into problems with the COMSOL Server.

V. Setting up PSO

Particle swarm optimization was implemented in Matlab, and applied to a model created by my fellow student Charlie Bjork. The model is a integrated magnetic gear permanent magnet machine, see his thesis for further details [16]. He was interested in optimizing the machine to achieve a feasible machine with the highest possible torque.

In order to run PSO on the model, three things were needed from Bjork:

- A parametrized model
- An objective function
- Lower and upper bounds for the parameters

The problem had 12 dimensions, and it was a mixed problem, with two of the values being integers (number of poles and slots). The problem was defined as:

$$max\, f(x) = |torque|, \qquad x \in \mathbb{R}^n$$

Subject to the constraints

$$
\begin{aligned}
2 &< P_i < 30\\
12 &< N_i < 150\\
P_i + 10 &< N_i \leq 8 \cdot P_i\\
0.01 &< HSR_{iron\_thickness} < 0.10\\
0.01 &< Slot_{thickness} < 0.10\\
&.\\
&.\\
&.\\
0.01 &< AG_i < 0.1
\end{aligned}
$$

Each parameter represents one dimension in the search space, a 12 dimensional space where the length of the "walls" are determined by the parameter bounds. The bounds on each parameter is called the *barriers* of the search space. All of the dimensions, except N_i, had static barriers. N_i was handled by first calculating the new particle positions in dimension P_i, and then setting the barriers for N_i to $[P_i + 10, P_i \cdot 8]$ in every iteration.

The barriers were handled with "bouncing walls", if a particle hits a wall, the velocity will change direction.

The integers were handled by simply rounding off the values.

A. Implementing PSO with Matlab

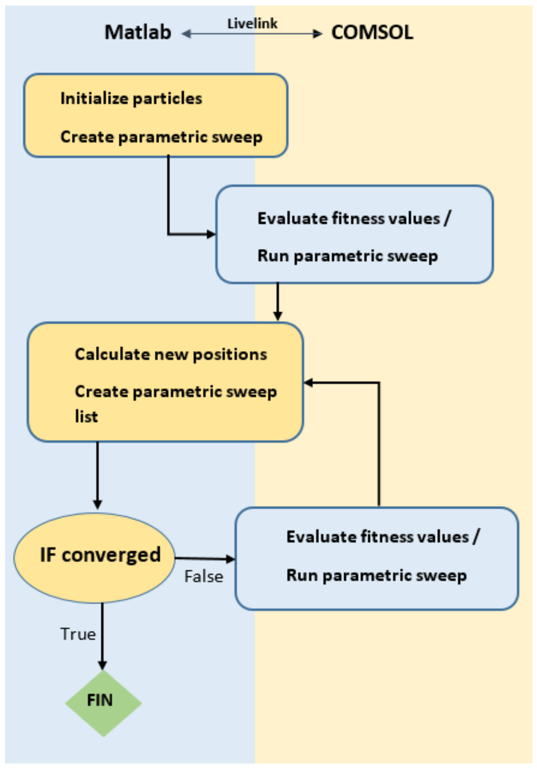


Fig. 20: General program flow

The general program flow of the implementation can be seen in figure 20.

A lot of time was spend trying to implement PSO on Vilje. In order to parallelize the script as much as possible, it would be best if the particles could act independently of each other. That is, if the fitness values of one particles could be evaluated as soon as the particle has moved to a new position. To achieve this with LiveLink, the first attempt was to start one mphserver per particle. However, it was discovered that each instance of the mhpserver uses one license. This means that the number of particles would be limited to the number of licenses, not a feasible solution.

In the end, the only working solution discovered was to use only one mphserver. The particles have to wait until all of them are finished before they can move on to the next step. This is not an optimal solution, especially if there exist some parameter combinations that creates models that are difficult to solve. In that case, many particles could be held waiting each iteration. A time limit on each simulation would help to limit waiting time, but COMSOL does as per today have no functionality for aborting a simulation after a set time. Instead, some care was taken to choose good barriers.

The fitness evaluation was done by creating a parametric sweep in the COMSOL model. For each iteration, the particle positions were calculated and saved in a matrix called "positions". The parametric sweep list was then set to the position matrix as in figure 21. For reasons not entirely understood, it is necessary to change the list in both "Study1->Parametric Sweep" and "Study 1 -> Job Configurations -> Parametric Sweep", or COMSOL gives you an error. It might be because the "Job Configurations" node does not update automatically.

```

model.result.table('tbl10').
    clearTableData;
model.study('std1').feature('param').
    set('plistarr', positions);
model.batch('p1').set('plistarr',
    positions);

model.sol('sol1').runAll;
model.batch('p1').run;
```

Fig. 21: Matlab code showing how you can change a parametric sweep list, and run the sweep using the mphserver

The method in figure 21 worked nicely on my personal computer, but I could not get it to run successfully on Vilje. I used many weeks trying to run the script, and a lot of emails were sent to COMSOL Support. Eventually, COMSOL suggested that there is a bug in COMSOL 5.2, which makes it impossible to run the 5.2 mphserver distributed on a cluster. Their suggested solution was to use COMSOL 5.3, and 5.3 was installed on Vilje.

Tests showed that the 5.3 mphserver distribute sweeps nicely on the cluster. Unfortunately, it turned out that there was another type of bug in the 5.3 mphserver, which caused the program to run forever. COMSOL just recently published a update to 5.3, which will probably fix the bug, but as there

TABLE V: Chosen parameter values for PSO

Parameter	Value
V_{max}	(1/45)*interval length
c_g	2.5
c_p	0.5
S	20-30

was limited time, the update was not tested. To summarize, my experience is that the COMSOL mphserver works well on Windows, but is not very stable on Linux in distributed mode.

In the end, an alternative way to run the sweep was found. Matlab has a system command, which will run a command at the Linux command line, and wait for it to finish. By using this command, you can start a COMSOL batch job from the Matlab script. The COMSOL batch job is the most stable way to run a job, and no problems were encountered with this method.

Figure 22 shows the final code. There are three programs involved, Matlab, the COSMSOL mphserver, and a COMSOL batch process. The mphserver will take the parametric sweep list from Matlab, and set up the sweep. Instead of solving the model, the model is saved as "tempmodel.mph" in line 2. The system command is then used to solve the tempmodel using the "normal" batch job. When the batch job is finished, the results will be saved to the model "outfile.mph". The mphserver will then load the new model, and the results can be extracted.

```
1 model.batch('p1').set('plistarr',
      positions);
2 mphsave(model,'tempfile.mph');
3
4 status = system('comsol -nn 200 -np 4
     -verbose -clustersimple -f
     comsol_nodefile -batchlog log.txt
     batch -tmpdir $w -inputfile
     tempfile.mph -outputfile outfile.
     mph');
5
6 model = mphopen('outfile.mph');
```

Fig. 22: Matlab code showing how the sweep was run using the system command

Hopefully, COMSOL will fix the problems with the mphserver so that this roundabout method will not be necessary in future projects.

If one of the simulations in the parametric sweep fails, there will be a missing line in the probe table. To handle this a function was created to find missing lines in the matrix, and insert a fitness value of 0.

The maximum velocity was set by testing the algorithm and looking at the paths taken by the particles. It was given a value that gave a nice looking path without any large jumps. The chosen PSO parameters can be found in table V.

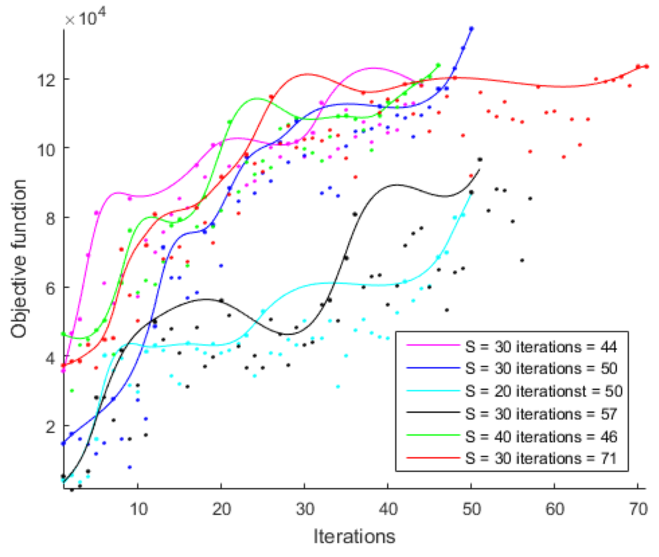
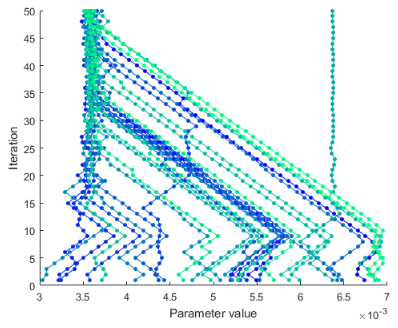
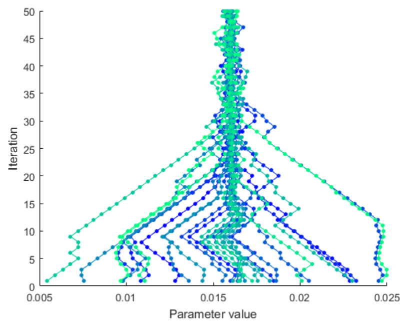


Fig. 23: Results from six runs of PSO on Vilje, using $c_p = 0.5, c_g = 2.5$



(a) "AG_i"



(b) "Stat_PM_thick"

Fig. 24: Particle paths for a run with S=30, showing the paths of each particle for the control variables "Stat_PM_thick" (thickness of stator permanent magnets) and "AG_i" (inner air gap length).

B. Results

PSO was run several times with different number of particles, and the results can be found in figure 23. As the figure shows, it looks like a higher number of iterations would have been beneficial. Looking at the dark blue line, it rises rapidly towards the end of the optimization.

Figure 24 shows the path taken by the particles in two of the dimensions, for one of the runs (dark blue line). For figure b, the particles converge early, at around iteration 30-40. Studying the particle paths for all the dimensions showed that the particles converged early in all the dimensions, except for the one shown in figure 24 a). If the particles had been given enough iterations to converge in all the dimensions of the search room, a better objective function might have been achieved.

The algorithm could also have been tested with several neighborhoods, and with different scaling parameters. The optimization algorithm was not optimized further however, because the results was found to be excellent by Bjork, and because of time restrictions.

A large parametric sweep of the same mode was also run for Bjork on Vilje, with close to 2000 simulations. The results were used to select good parameter values in his iterative optimization approach. PSO delivered a slightly better objective function than his approach. PSO needed to run more simulations in total to achieve a good result (about 10 000 simulations), but it was still a relative small amount of computational resources.

It is surprising that PSO performed so well without any tweaking. The chosen scaling parameters most likely fit the problem very well.

VI . Conclusion

Running COMSOL jobs on Vilje requires some technical know-how, but it should be relatively easy for most students to learn. Very little knowledge about parallel computing is required, as COMSOL will set up communication between nodes automatically.

The COMSOL batch job and the COMSOL mphserver was both run on Vilje. The batch job was found to be the most stable version. The COMSOL mphsever was not run successfully in more than one node.

The speedup of medium-sized single stationary studies was found to be very limited, and according to COMSOL [14] the maximum possible speedup for large models is in the range 8-13. The speedup is generally greater for models with ten to hundred of millions of degrees of freedom.

In some cases, speedup can be improved by partitioning the mesh. The most important thing is to select the right solver. Iterative solvers were found to be much faster for a selected model, but the speedup was in the same range.

The best possible speedup when running COMSOL jobs on Vilje is achieved when parametric sweeps are distributed on the cluster. Population based optimization algorithms will benefit from this, as in each iteration the population individuals can be evaluated by a parametric sweep.

PSO was implemented, and gave good results when applied to a machine optimization case. It is surprising that PSO was able to deliver so good results without any tweaking of the algorithm, as most literature suggests that this is important in order to achieve good convergence. This suggests that PSO might be more powerful than first assumed, or that the chosen parameters fit the problem very well. PSO was found to be a very useful tool for design optimization.

There was many problems with running LiveLink for Matlab distributed on a cluster, and a great deal of time was spent trying to make it work. In the end, an alternative method was found for running PSO in Vilje.

Vilje will be replaced by the supercomputer "Fram" in the near future, but the methods and tips presented in this thesis should be valid and relevant for Fram as well.

VII . Acknowledgments

I would like to thank:
Supervisior Robert Nielssen, for help and encouragment
Boyfriend Andreas Roessland, for moral support
Friend Muhammed Usman Hassan, for his inputs and for creating a positive working environment
Chalie Bjørk for a pleasant collaboration

References

[1] W. Gander, M. J. Gander, and F. Kwok, *Scientific Computing An Introduction using Maple and MATLAB*. Switzerland: Springer, 2014.

[2] G. H. Golub and J. M. Ortega, *Scientific computing and differential equations : an introduction to numerical methods*. Academic Press, 1992.

[3] W. Frei, “Meshing your geometry: When to use the various element types,” https://www.comsol.com/blogs/meshing-your-geometry-various-element-types/, accessed: 2017-07-03.

[4] C. M. Cyclopedia, “The finite element method (fem),” https://www.comsol.no/multiphysics/finite-element-method, accessed: 2017-07-01.

[5] S. J. Salon, *Finite Element Analysis of Electrical Machines*. Kluwer Academic Publishers, 1995.

[6] R. Shonkwiler and L. Lefton, *An Introduction to Parallel and Vector Scientific Computing*. Cambridge University Press, 2006.

[7] E. Rønquist, A. M. Kvarving, and E. Fonn, *Introduction to Supercomputing, TMA4280*, 2016.

[8] “Top500 list - june 2017,” https://www.top500.org/list/2017/06/?page=2, accessed: 2017-07-06.

[9] D. Padua, *Encyclopedia of Parallel Computing*. Springer, 2011.

[10] J. Akin, *Finite Element Analysis with Error Estimates, An Introduction to the FEM and Adaptive Error Analysis for Engineering Students*. Elsevier Butterworth-Heinemann, 2005.

[11] COMSOL, “COMSOL multiphysics reference manual,” 2016.

[12] I. Yavneh, “Why multigrid methods are so efficient,” 2006.

[13] J.-P. Weiss, “Using the domain decomposition solver in COMSOL multiphysics,” https://www.comsol.com/blogs/using-the-domain-decomposition-solver-in-comsol-multiphysics/, accessed: 2017-07-09.

[14] "COMSOL knowledge base: Comsol and multithreading," https://www.comsol.com/support/knowledgebase/1096/, accessed: 2017-07-09.

[15] COMSOL, "Forced convection cooling of an enclosure with fan and grille," 2010.

[16] C. Bjørk, "Design, optimization and analysis of an integrated magnetic gear permanent magnet machine for marine applications," 2017.

[17] S. Genovesi, D. Bianchi, A. Monorchio, and R. Mittra, "Optimization techniques for electromagnetic design with application to loaded antenna and arrays," 2011.

[18] J. Kennedy, R. C. Eberhart, and Y. Shi, *Swarm Intelligence*. Academic Press, 2001.

[19] I. Boussaïd and J. L. ad Patrick Siarry, "A survey on optimization metaheuristics," *Information Sciences, Volume 237, Pages 82-117*, feb 2013.

[20] J. C. Bansal, P. K. Singh, M. Saraswat, A. Verma, S. S. Jadon, and A. Abraham, "Inertia weight strategies in particle swarm optimization," 2011.

[21] R. C. Eberhart, Y. Shi, and J. Kennedy, *Swarm Intelligence*. The Morgan Kaufmann Series in Artificial Intelligence, Mar. 2001.

[22] A. E. Olsson, *Particle Swarm Optimization: Theory, Techniques and Applications*. Hauppauge, US: Nova Science Pub Inc, Mar. 2011.

[23] M. Erik and H. Pedersen, "Good parameters for particle swarm optimization," 2010.

[24] T. R. Bjørtuf, E. Attar, M. Saxegaard, A. Sitko, O. Granhaug, and H. Landsverk, "Dielectric and thermal challenges for next generation ring main units (RMU)," 2013.

[25] A. Pedersen, T. Christen, A. Blaszczyk, and H. Boehme, "Streamer inception and propagation models for designing air insulated power devices," 2009.

[26] R. W. Blower, *Distribution Switchgear*. 8 Grafton Street, London, Great Britain: Collins Proffesional and Technical Books, 1986.

[27] V. Adams and A. Askenazi, *Building Better Products with Finite Element Analysis*. 2530 Camino Entrada, Santa Fe, USA: OnWord Press, 1999.

[28] J. Yström, "Fast solvers for complex problems," 2007.

[29] S. B. Desai, S. R. Madhvapathy, A. B. Sachid, J. P. Llinas, Q. Wang, G. H. Ahn, G. Pitner, M. J. Kim, J. Bokor, C. Hu, H.-S. P. Wong, and A. Javey, "Mos2 transistors with 1-nanometer gate lengths," *Science: Vol. 354, Issue 6308, pp. 99-102*, oct 2016.

[30] ITRS, "International technology roadmap for semiconductors 2.0," ITRS, Tech. Rep., 2015.

[31] COMSOL, "Optimization module, user's guide, version COMSOL 5.2," 2016.

[32] *COMSOL Multiphysics Reference Manual*, COMSOL.

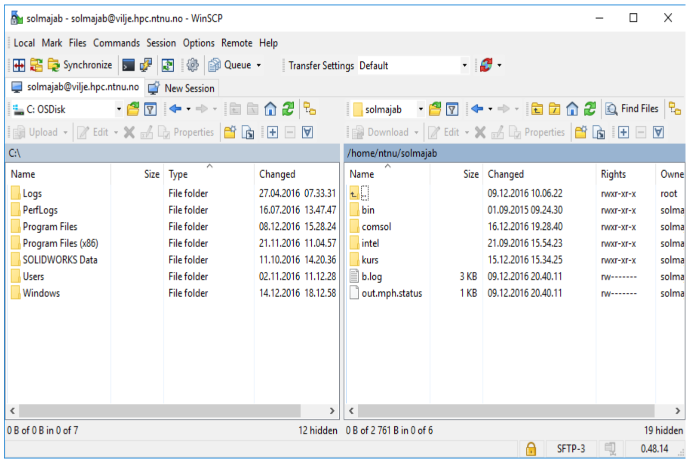
Appendix: Some Practicalities for Windows users

To use Vilje to solve a COMSOL model you need to:

1) Make a user account on Vilje (see hpc.ntnu.no)

2) Get a project account approved (see hpc.ntnu.no)

3) Download a SCP client for Windows. For example winSCP (https://winscp.net/eng/download.php.)
It looks like this:


solmajab - solmajab@vilje.hpc.ntnu.no - WinSCP
Local Mark Files Commands Session Options Remote Help
Synchronize
Queue
Transfer Settings Default
solmajab@vilje.hpc.ntnu.no
New Session
C: OSDisk
solmajab
Find Files
Upload
Edit
Properties
Download
Edit
Properties
SFTP-3
0:48:14

C:\

Name	Size	Type	Changed
Logs		File folder	27.04.2016 07.33.31
PerfLogs		File folder	16.07.2016 13.47.47
Program Files		File folder	08.12.2016 15.28.24
Program Files (x86)		File folder	21.11.2016 11.04.57
SOLIDWORKS Data		File folder	11.10.2016 14.20.36
Users		File folder	02.11.2016 11.12.28
Windows		File folder	14.12.2016 18.12.58

0 B of 0 B in 0 of 7 12 hidden

/home/ntnu/solmajab

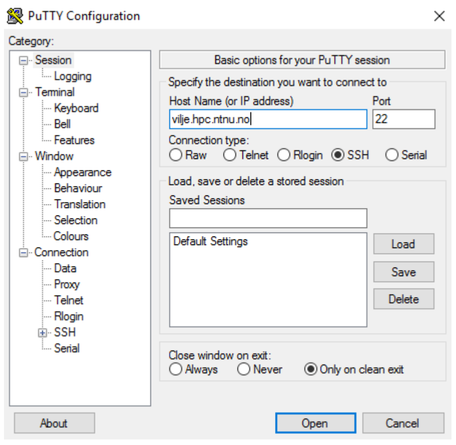
Name	Size	Changed	Rights	Owne
..		09.12.2016 10.06.22	rwxr-xr-x	root
bin		01.09.2015 09.24.30	rwxr-xr-x	solma
comsol		16.12.2016 19.28.40	rwxr-xr-x	solma
intel		21.09.2016 15.54.23	rwxr-xr-x	solma
kurs		15.12.2016 15.34.25	rwxr-xr-x	solma
b.log	3 KB	09.12.2016 20.40.11	rw-------	solma
out.mph.status	1 KB	09.12.2016 20.40.11	rw-------	solma

0 B of 2 761 B in 0 of 6 19 hidden

Fig. 25: winSCP

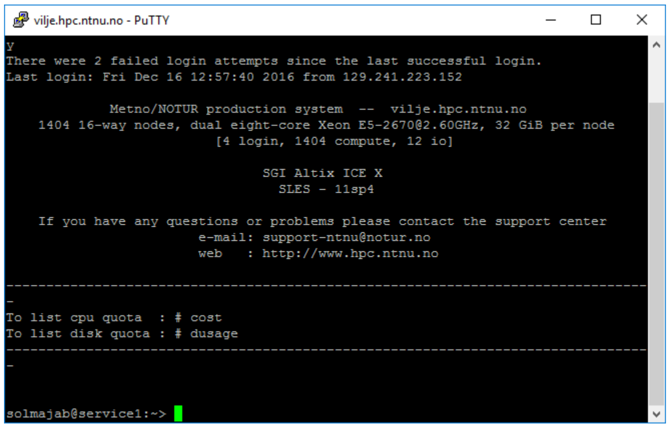
winSCP allows you to move files from your computer to Vilje by simply dragging your files from one folder to the other. The left window is your own computer, navigate to the right folder and drag your COMSOL file across.

4) Download a SSH client. For example puTTY (http://www.chiark.greenend.org.uk/ sgtatham/putty/download.html)



Host Name (or IP address): vilje.hpc.ntnu.no
Port: 22
Connection type: Raw Telnet Rlogin [✓ SSH] Serial
Saved Sessions: Default Settings
Close window on exit: Always Never [✓ Only on clean exit]

(a) puTTY start screen



```
y
There were 2 failed login attempts since the last successful login.
Last login: Fri Dec 16 12:57:40 2016 from 129.241.223.152

             Metno/NOTUR production system  --  vilje.hpc.ntnu.no
    1404 16-way nodes, dual eight-core Xeon E5-2670@2.60GHz, 32 GiB per node
                     [4 login, 1404 compute, 12 io]

                           SGI Altix ICE X
                             SLES - 11sp4

    If you have any questions or problems please contact the support center
                     e-mail: support-ntnu@notur.no
                     web   : http://www.hpc.ntnu.no

--------------------------------------------------------------------------------
-
To list cpu quota  : # cost
To list disk quota : # dusage
--------------------------------------------------------------------------------
-

solmajab@service1:~>
```

(b) How puTTY looks when you are logged in to Vilje

puTTy allows you to log in to Vilje from another computer. To log on to Vilje write "vilje.hpc.ntnu.no" under "Host name", and press open. In the future, this will change as Vilje is replaced by "Fram". You will be prompted to give your username and password.

Vilje runs Unix, and has a command-line interface. To navigate and submit your job, you need to learn a few Linux commands. As a minimum, learn how to navigate.

- **cd** Change directory, feks "cd myfolder"
- **cd..** Move to the parent directory.
- **ls** List the contents of the directory
- **cat myfile** Print the contents of "myfile" to screen
- **vi myfile** Edit the file "myfile" with Vi

To summarize: With winSCP you *move* your files to Vilje, with puTTY you can tell Vilje what to *do* with the files.

It is also very useful to learn how to use a text editing program like Vi/Vim, so you can easily change a job script. Vi has its own set of commands, but you only need a few in order to edit a text file.

- **i** Insert mode will move you from the Vi command line and "into" the text, allowing you to edit the text.
- **Ecs** Use escape to leave "insert mode" and move back to the Vi command line.
- **:q** Quit the program without writing to file.
- **:w** Write to file
- **:wq** Write to file, and quit Vi.
- **gg and shift+g** In "command line mode", these commands will move you to the top or bottom of the text. Useful if you are reading a large log file.