Leiden University

BSc Computer Science

Analysing Electron Tomography with IMOD on the LLSC

Name:	Simon R. Klaver
Studentnr:	1140760
Date:	Thursday 25th June, 2015
1st supervisor:	Fons J. Verbeek
2nd supervisor:	Kristian Rietveld

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

State-of-the-art Electron Microscopes, such as large Transmission Electron Microscopes like the Titan Krios, can automatically process multiple samples in sequence without human intervention between different samples. Because of this, more image series can be produced in a shorter timespan. Therefore, it is deemed necessary to also be able to complete the further processing of the images that result from the microscope in shorter time. A possible means of reducing the processing time is by parallelising the software used for the further processing of image series. Within this project, we have investigated the parallelisation of Electron Tomography software, in particular IMOD, on the Leiden Life Sciences Cluster (LLSC) available at LIACS. It was investigated how a configuration for parallel execution of the software could be optimized to realize minimal execution times for each processed image set. We have observed modest speed-ups and have gained understanding in how this goal could be further accomplished.

Contents

1 Introduction

The Netherlands Center for Electron Nanoscopy or, in short NeCen, is a research facility which applies electron tomography on tiny samples such as cells, viruses, and bacteria. This process enables researchers to make 3D configurations of these tiny living organisms, which can be used to understand them better which is necessary for future studies. To achieve this they have a state of the art electron microscope which produces images of the samples. These images can then be formed to a 3D configuration. The electron microscope used is called the Titan Krios[1], which has a width of about 1 meter and a height of about 3 meters (see figure 1). It is connected to a computer, so images taken can be immediately stored on a hard disk and be processed right afterwards.

The scope of the project is to enable software facilitating this process, called

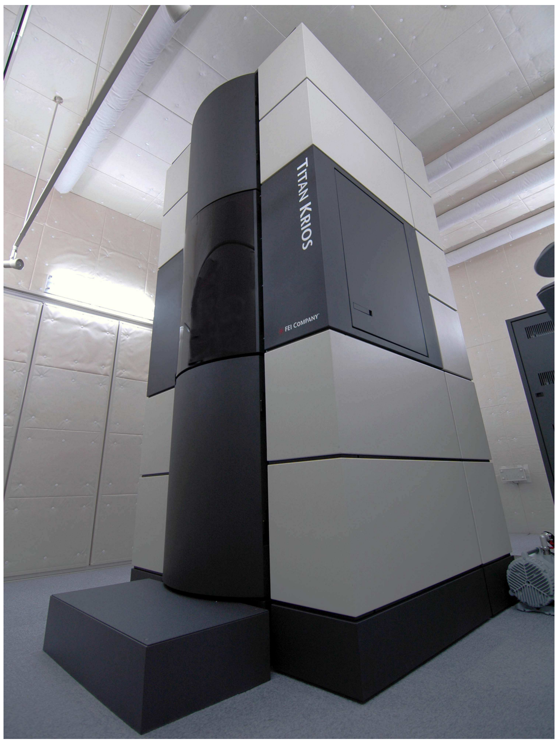


Fig. 1: Titan Krios

Source: `http://www.oist.jp/news-center/photos/titan-krios`

IMOD to run on a computing cluster, called the *Liacs Life Sciences Cluster*, which is situated at the *Leiden Institute of Advanced Computer Science*. In the case this is confirmed to work and confirmed to produce valid output, research will focus on the optimization of the process on the *LLSC*, by utilizing the availability of parallelism in both the software and the *LLSC*. This leads to the research question: How to implement *IMOD* on the *LLSC*, and how to improve it when it works?

First in Section 2, information is provided on certain terms mentioned and general background on the project. In Section 3 the possible solutions and the work on them is made clear. Then results are given in Section 4 and finally the paper is concluded with Section 5, in which recommendations are given about the best configuration and the development of the used software.

2 Background

This section contains the background of the project. First we will give a general image of electron microscopy and electron tomography. Then we will explain the terms IMOD, LLSC, and Torque. Lastly, we will grant insight in the current situation, and thus why this project as started.

2.1 Electron Microscopy

Electron microscopy enables visualizing objects on sub-cellular scale, which are not visualizable by devices relying on the capturing of rays of light in the visible spectrum: Wavelengths for light in this spectrum vary from 400 to 700nm, which is from violet to red, and therefore can only be used to capture up to a resolution of 200nm[2]. Objects on sub-cellular scale however are much smaller, and usually vary in the tenths of nanometers. Electron microscopes can capture these tiny samples, and with a specialized electron microscope even capturing objects of around 50pm is reported[3], which is managed by sending electrons through the sample instead of regular light. As electrons collide with the sample, they either go through or bounce off. The amount of electrons that go through the sample and with that the amount of electrons that arrive at the bottom determine the densities of different parts of the sample, which therefore can be used to visualize the sample. As at the bottom either a capturing device or electron-sensitive layer, such as phosphor, is placed which captures the electrons which went through the sample, this visualizes the sample. Invented roughly eighty years ago[4], these microscopes have been improved a lot, as the earliest were relatively small and could for example not store their output on a computer.
The main setup of an electron microscope is as follows: At the top, an electron cannon will shoot electrons in a beam through the microscope, which is best imagined as a giant tube. Roughly halfway a sample will be placed, along with 3 sets of lenses. The first set, the *condensor* lenses, focus the electrons slightly to keep them in a beam, as they tend to scatter when shot at high velocity. The second set, the *objective* lenses, either converge the beam further to ensure as many electrons as possible hit the sample or only let the electrons pass which are bound to hit the sample. This differs along with different types of electron microscopy. The third set, the *projector* lenses, diverge the beam again to allow the electrons to scatter over the phosphor screen or CCD camera at the bottom. A CCD camera, or Charge-Coupled Device camera, can detect photons, or in this case electrons, and converse these to electrical charges, which are then used to produce an image. In modern times, the CCD camera is more popular as images taken with a camera can be directly stored on a computer. See Figures 3 and 4[5] for methods of attaching a CCD camera to an electron microscope.

2.2 Electron Tomography

Electron tomography is the process in which multiple tilted images (such as in figure 5) of the same sample are combined into a 3D-model visualizing the sample, enabling a better study of it than what could be done with a 2D-model. This is done by taking the multiple images acquired in tilted fashion and looking for corresponding points on which they can be aligned. The best way to

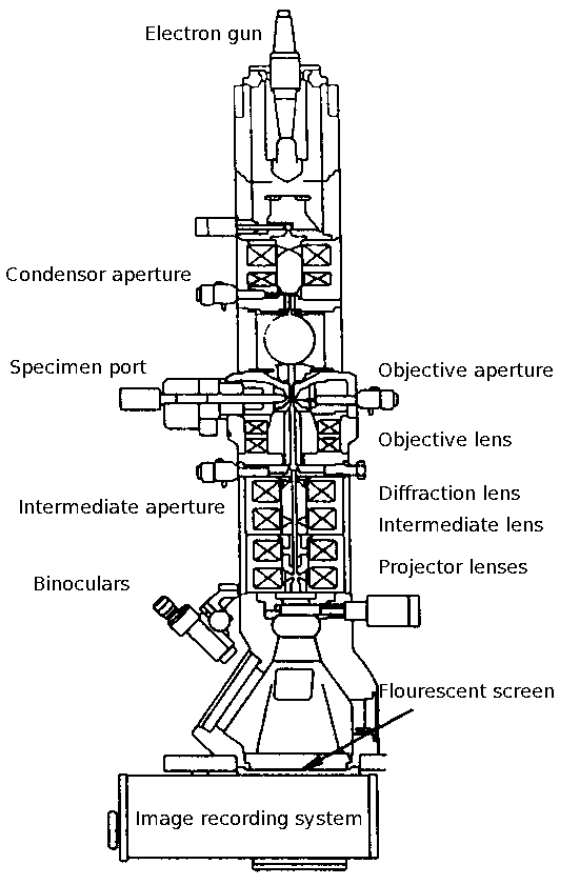


Fig. 2: Standard layout of a transmission electron microscope

Source:
"http://commons.wikimedia.org/wiki/File:Scheme_TEM_en.png"

ensure these points can be found easily is by adding goldbeads to the sample before acquiring images of it. The beads will have to be small as they must not block the view on the sample, but still be large enough to be measured as beads and not as noise. A diameter of roughly 1% of the size of the sample will do fine. These beads will then be tracked through the different images by hand or through software, and after tracking them through the images an aligned model will be formed.

Another way to align the images is by using patch tracking. This is particularly useful when there were no goldbeads in the sample when acquiring images, when there are not enough beads to align the picture on those, or any other reason due to which aligning on the beads does not work. Patch tracking divides each image in patches and matches them to the other pictures and their patches to find out which patch of the other images corresponds most to the selected patch of the current image. This will be repeated for several different patches in several different images, but in the end the images will be aligned corresponding

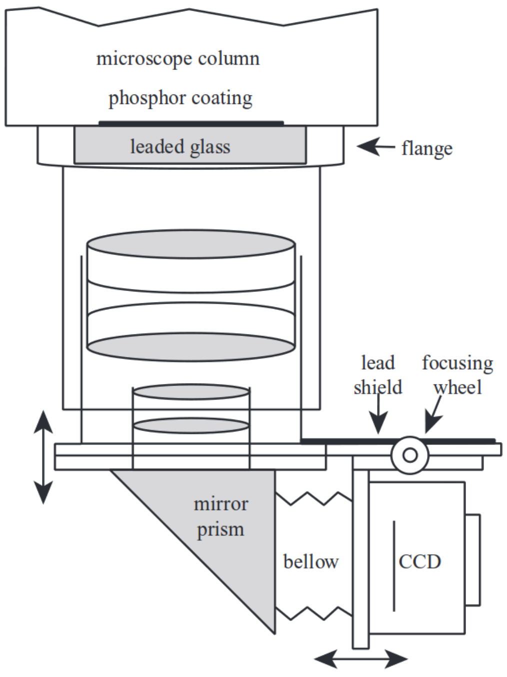


Fig. 3: Example of a CCD camera (1)[5]

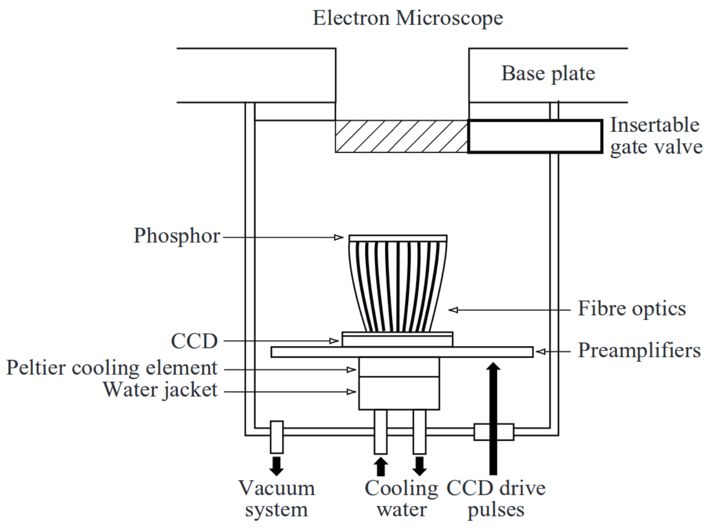


Fig. 4: Example of a CCD camera (2)[5]

to the different tracked patches. An advantage of this second method is that, assuming the goldbeads were left out on purpose, a 3D-model will be made of the sample without any beads blocking the view. The goldbeads, when used, can also be removed through different software which fills the space of a bead

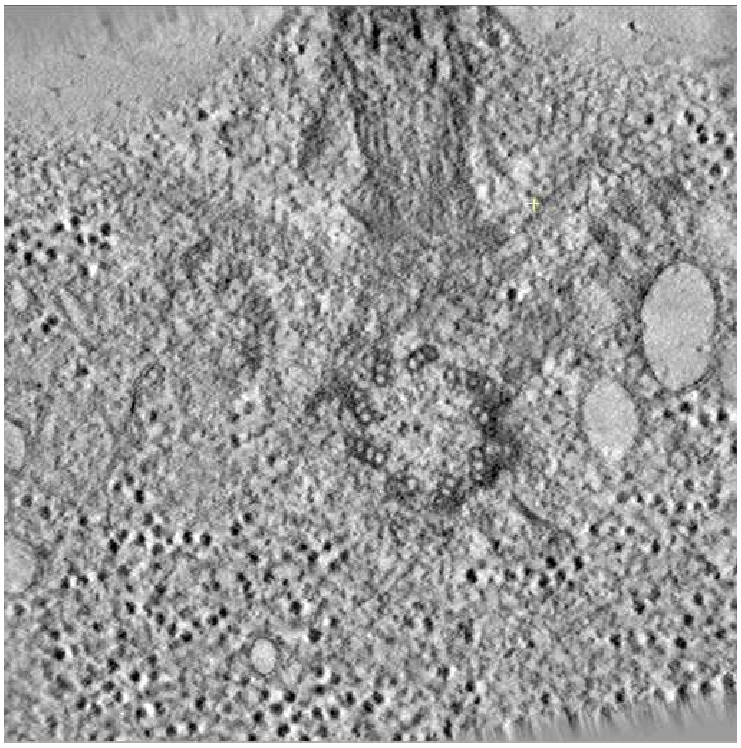


Fig. 5: Tilted image from the Etomo tutorial[6]

up with an average of it's surroundings, but without adding them in the first place there will be actual sample in that place to view. A disadvantage however is that this method is generally less precise, even with fine-tuned parameters.
The alignment of images also depends on the type of containment the sample is put in when an image is acquired with an electron microscope. In general, there are two types of containment used at NeCEN, plastic and ice. The main differences between containing the sample in either ice or plastic are the cost and the quality. A plastic embodiment is cheaper as it costs less labour than an embedding of ice, but it can drastically lower the quality, as plastic is more susceptible to the heat generated in the process. This may seem counter-intuitive, as ice would generally melt from heat and plastic would in general keep in shape, but on the small scale of electron microscopy even a small change can change the protein configuration and therefore alter the sample. When the plastic embodiment is subject of any heat source, whether external or even just the electrons sent through it, it may slightly deform. Ice does not have this problem, as in general the heat it is subject to is not enough to make it melt, while it will also not deform. A sample does however take more effort to be prepared in ice than in plastic, and is therefore more expensive than embedding the sample in plastic. Due to this plastic is a lot more used as a sample embedding in research.

2.3 IMOD

The process of electron tomography requires specific software to perform. There is a lot of software which can be used for this purpose, but as NeCEN and other affiliated instances were already using a certain software package and we could get their help to understand how this software works, IMOD[7][8] was chosen. Developed in the early 1990s, the software utilizes partly other, older, software in its own software structure which also offers a graphical user interface called Etomo. Since the initial release numerous changes have been made, such as an improved user interface and multithreading. Since release 4.7.3, released around 3/31/14, a script called *batchruntomo* was introduced, which automates the process of building a tomogram with a set of options given at start.

2.4 LLSC

LLSC is an abbreviation for the LIACS Life Sciences Cluster. It consists of a number of server machines, or nodes, obtained by LIACS which have been configured as a cluster-computer. The total number of active nodes in the cluster is 24. The clock frequency is 2.66 GHz, and there are 9 nodes with 8 processors per node and 15 nodes with 4 processors per node, plus 5 nodes which are used for specific functions such as user space, development and management. Each node has 16 GB RAM and approximately 655 GB total hard disk size, distributed over 6 disks: 3 disks of 146 GB and 3 disks of 73 GB. Aside from that, there is also a file server which can be used by all nodes as storage.

2.5 Torque

Torque[9] is the resource manager with built-in scheduler which runs on the LLSC and enables the users to submit jobs to the cluster: A job is a program or a collection of programs to run in select order with possible given options on how the cluster should accommodate these programs. For example the amount of nodes and their type needed, the amount of time the job can at most use, and the priority the job should have over other submitted jobs. Other options include the dependencies between different jobs, or interactivity between multiple jobs in a job-array. The scheduler then tries to fit submitted jobs as well as possible with the amount of available nodes and jobs already running at that moment: Jobs submitted with a higher priority than the ones already running will not interrupt those, but will be run earlier than other already queued jobs with lower priority. This first of all enables a user to submit multiple of these jobs at the same time, without having to check every so much time if a new job can be submitted as the previous one finished. Secondly it enables multiple users to independently of each other submit jobs on the cluster, which according to their priority and/or position in the queue will execute in a certain order, and which will not interfere with each other for they do not use the same nodes, space and/or processors. Lastly, a scheduler takes away the pain of distributing the resources over the available jobs by hand.
The built-in job-scheduler of Torque has shown to be suboptimal in previous work[10] and Maui[11] has shown to be a suitable replacement scheduler. The Maui scheduler comes with its own utilities, however with some altering of

the configuration PBS/Torque utilities can also be utilized. This enabled the continued use of already developed scripts.

2.6 Old implementation

NeCEN currently has a cluster dedicated to processing tomography with IMOD. It consists of about eight relatively old server nodes configured as a small cluster, but as the *LLSC* consists of many more nodes it should be possible to significantly reduce computation times compared to the NeCEN computer-cluster. With the switch to the *LLSC*, for convenience and minimizing possible delays due to needed input, the goal is to fully automate this process on the *LLSC*. This is instead of the current manual usage of IMOD on the old cluster. Another point is that the process of putting in parameters in the different screens of Etomo was mostly redundant. Quite a lot of the parameters were the same for all data sets, and most of the parameters could be known at the start of a process. Images in the stack to leave out, for example, could be determined by just looking at the image stack before the process, and does not require to first run some programs on the stack to for example remove X-rays. Still, in Etomo these parameters needed to be filled in every single time an image stack was put in, even for sets already processed once, and for most of the parameters one needs to sit through all the previous steps before having a chance to put them in. Figure 6 shows the different tools used in the different categories of Etomo's GUI, which is shown in figure 7.

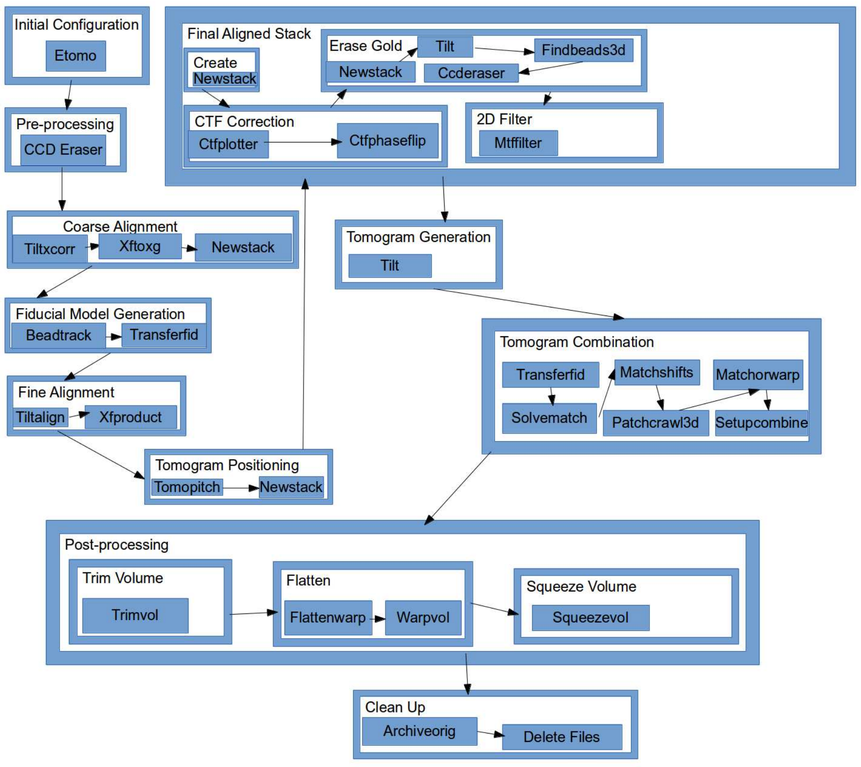


Fig. 6: Flowchart Etomo

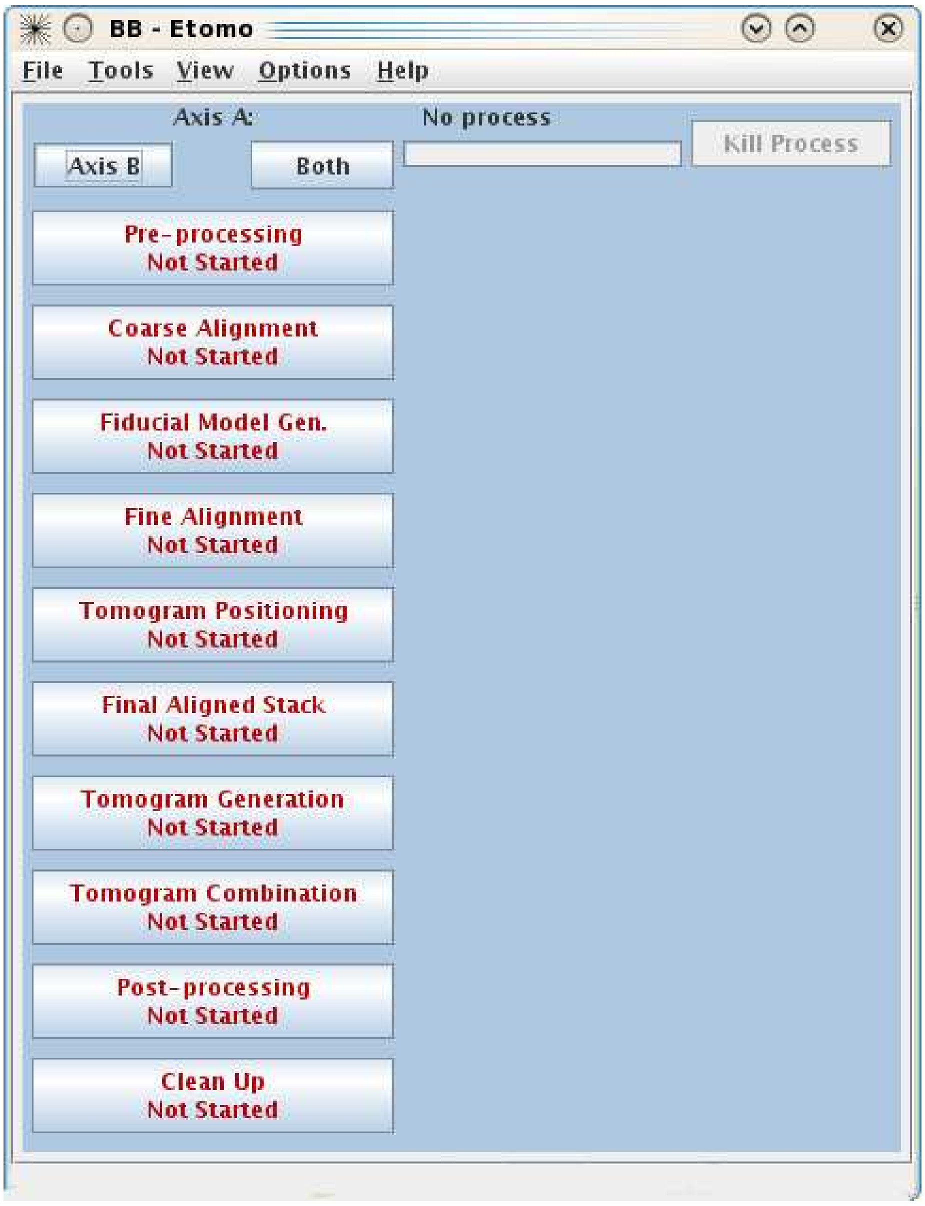


Fig. 7: Etomo Main Window (GUI)

3 Implementation

In this section the implementation of the project is discussed. At first, the initial solution is given. Secondly, we will talk about the re-evaluation of the development plan, then about the outcome of that which is called *batchruntomo*. Lastly, we will talk about the experimental setup which is used to run experiments.

3.1 Initial solution

3.1.1 The idea behind the initial solution

The initial solution to automate *Etomo* was to take the original software and wrap it up in a script to ensure the tools which had to be used were used in the correct order and with the correct parameters. This way a tomogram would be generated exactly the same way it could be generated when processed manually using the GUI, however now the tasks would be automated. The parameters to be used by the script should be easy to fill in and easy to change afterwards for different data sets to be processed. This can for example mean having the parameters all in one file which is read by the script, or having tiny scripts for each of the used tools and hardcode the parameters in each. One could of course give all the parameters as arguments when calling the script, but given the amount of needed parameters for the process this is not feasible.
Next to the script there should be measures to ensure the script runs correctly each time and preferably in a consistent way. At least in the developing stage of the script this would surely help, but also with the actual data sets and the parameters, as for one it would be quite a hindrance if the script would not be consistent in succeeding on certain data sets, and for two as many data sets need to be run in order without having to worry about any needed user-guided actions to occur in the meantime.
The script itself would consist of multiple smaller scripts, each as a wrapper around one of the tools used in a normal processing of a data set with the GUI, with in each small script the parameters hardcoded, mainly because this was the easiest way and secondly as the only test set present at the time was the tutorial test set, which needed the same set of parameters on each run. Each small script would then be called by a larger script, wrapping around all the smaller scripts, to ensure the scripts are called in the correct order.

3.1.2 Implementing the initial solution

Part of the process of making these scripts and the overlapping script is to analyse the order in which the small scripts need to be called, and on a second notion to investigate which scripts could possibly be executed in parallel when they are independent of each other. In hindsight the latter was not possible, because the tools had to be run in sequential order as shown in Figure 6. Another thing to take notice of was multiple uses of a tool on different moments in the process. For example *newstack*, which recreates the data set to ensure changes made to different subsets of the data set are applied to the alignment set, so other tools using the alignment set would use the changed set instead of the original. It was probably made a separate tool to enhance modularity, as it is used on different places in -almost- the same way each time, whereas differences apply to having

multiple alignment stacks, in some cases, probably to enhance the quality of the overall process.
A third thing was the presence of performance improvement tools already present or the ability to improve tools to enhance performance. For example using the *.com*-files which were used by *Etomo* to communicate parameters to the tools, but which we did not use as writing the parameters down in the code was easier. Within the GUI, these are used as carriers of parameters, but it was expected these files also held parameters which were not given in advance by the user, such as values determined by the output of a tool. The parameters for the tutorial data set could be easily deduced from the GUI, which was used to process it already a few times, and the parameters required to call the tools were the same as asked for by the GUI: The names corresponded and therefore they could be determined to be corresponding easily. An error occurred however after implementing a few tools: The *.preali* file could not be found, and instead only a *.ali* file seemed to be made where *ali* is an abbreviation for alignment. With this particular example about the *.preali* file, it was the tool *newstack* which had more functionality, as it can produce either a *.ali* and a *.xf* file, or a *.preali* and a *.prexg* file, and a parameter could enable the second two parameters while the first two extensions were the default behaviour. Other problems include the seeming need of the automating of a tool called *3dmod* which could not be automated as it required user input, and whether *transferfid* would be needed or not in the eventual implementation.
Due to errors and uncertainties like these the development of the script and subscripts went slower than expected. Example of this is the *3dmod* problem mentioned earlier, as the user input only seemed to be clicking certain buttons which could not be automated, and therefore was too big of a hindrance to keep. Instead requiring user input was also considered, but rejected for the extra logic to program. The implementation of the script wrapping *patchcrawl3d* presented a bigger problem, as no clear parameters were given there and neither to be found anywhere, whereas the GUI seemed to have found a way around. So to say: the GUI seemed to use it without any parameters and still succeeded, but using it that way as an user outside the GUI and hoping it will have default parameters for itself does not work. A seeming solution to that problem was the usage of the *solvematch.com* *.com*-file, which seemed to handle quite some tools including but not limited to *patchcrawl3d* and *solvematch*. The automatic processing of this file, in which also no clear parameters were given for *patchcrawl3d* but which seemed to work for the GUI, would be done with *subm* or *submfg*, where *fg* means *foreground*, so whether the tools called in the *.com*-file processed with *subm*(*fg*) should be run on either the foreground or the background. However, it still did not work, for one because *patchcrawl3d* still gave errors about missing parameters, and for two because *solvematch* just refused to give clear way with the correct parameters and kept insisting the residual error was too high.

3.2 Re-evaluation of development plan

As the problems occurring, mentioned in the previous section, were not easily and quickly solved, the choice was made to contact the original developer and spokesperson of IMOD; David Mastronarde, and ask him a few questions regarding our problems. The questions asked can be found in Figure 8, and their respective answers in Figure 9. Note that the numbers before the answers refer to the first, second, and third question asked.

Dear David,

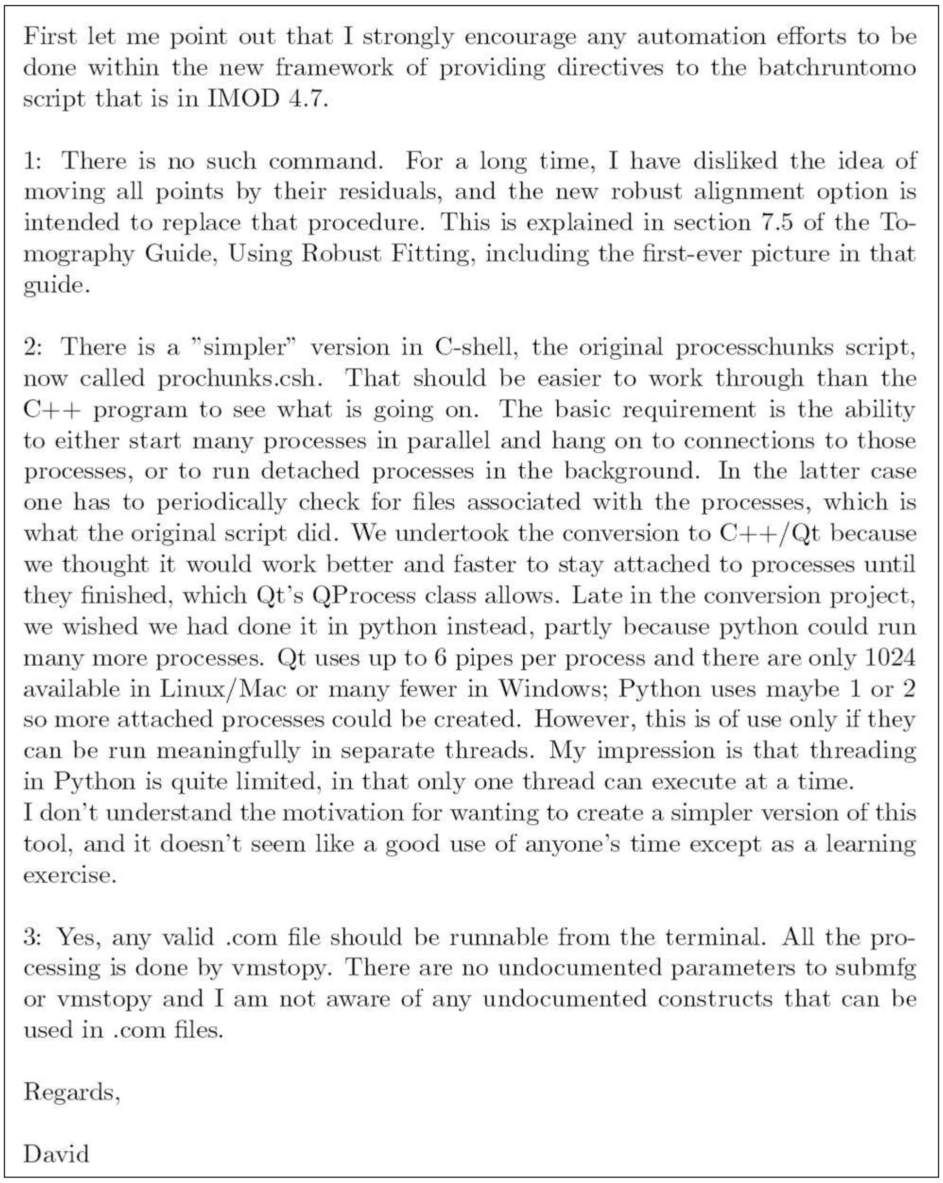
I have a few questions for you about IMOD from a collaborating undergraduate student we have working on automation of image processing using IMOD, in order to process as fast as possible the enormous amounts of tomographic data that we get from the Titan Krios microscopes. He had a few questions that I could not answer. I was hoping you could shed light on a few questions he had:
"What is the 3dmod command line command to fix the big residuals, and possible other options, such as moving all the points by their residual?"
"On which tools does the parallel manager rely, and in which manner, as if someone would want to implement a (simpler) version of it in i.e. Python?" (oddly specific)
"Should the execution of a .com file with subm(fg) be possible in the terminal without any pre-processing of the .com-file or any given but undocumented parameters?"

Best regards,

Roman Koning

Fig. 8: The e-mail with questions to David Mastronarde.

The first question was about *3dmod*, specifically about how to automate it to fix the residuals and move the bead-traces by this residual. This means that when the tool used to point out the beads, at that time *Raptor*, gives estimations of were the beads are positioned, *3dmod* will track these beads through the stack and based on the performance of these estimations in comparison with other estimations in the stack a conclusion is drawn whether a bead could be at its estimation, and a new estimation is given. In most cases the new estimations are better than the first ones, and as the process can be repeated over and over it would be nice if it could be automated in some way. In the first answer *autofidseed* was mentioned, of which we had no knowledge at that point in time and is discussed later in this paper.
The second question was about implementing parallelism in the script, as some tools were able to make use of multiple processors and nodes which was implemented in the 'parallel manager'. Being able to, in a way, copy this process and use it in the script could speed up the process a lot, but as it had little importance at that time a positive answer would probably not on itself have made a change. Main problem with implementing this in Python was that the original code is in Java and very complex, due to which it was not readable at all. The answer on this question mainly meant he saw no gain in re-implementing the parallel manager and apparently did write the manager in C++, which on its turn means the Java code was probably only a front-end.
The third question was about the tool *subm*, which is mentioned earlier, and



First let me point out that I strongly encourage any automation efforts to be done within the new framework of providing directives to the batchruntomo script that is in IMOD 4.7.

1: There is no such command. For a long time, I have disliked the idea of moving all points by their residuals, and the new robust alignment option is intended to replace that procedure. This is explained in section 7.5 of the Tomography Guide, Using Robust Fitting, including the first-ever picture in that guide.

2: There is a "simpler" version in C-shell, the original processchunks script, now called prochunks.csh. That should be easier to work through than the C++ program to see what is going on. The basic requirement is the ability to either start many processes in parallel and hang on to connections to those processes, or to run detached processes in the background. In the latter case one has to periodically check for files associated with the processes, which is what the original script did. We undertook the conversion to C++/Qt because we thought it would work better and faster to stay attached to processes until they finished, which Qt's QProcess class allows. Late in the conversion project, we wished we had done it in python instead, partly because python could run many more processes. Qt uses up to 6 pipes per process and there are only 1024 available in Linux/Mac or many fewer in Windows; Python uses maybe 1 or 2 so more attached processes could be created. However, this is of use only if they can be run meaningfully in separate threads. My impression is that threading in Python is quite limited, in that only one thread can execute at a time.
I don't understand the motivation for wanting to create a simpler version of this tool, and it doesn't seem like a good use of anyone's time except as a learning exercise.

3: Yes, any valid .com file should be runnable from the terminal. All the processing is done by vmstopy. There are no undocumented parameters to submfg or vmstopy and I am not aware of any undocumented constructs that can be used in .com files.

Regards,

David

Fig. 9: The e-mail with answers from David Mastronarde, on the questions as in the e-mail in Figure 8.

the usage of it, as the seeming lack of options did hinder the usage in our script. However, as was asked about the usage of it without any undocumented needed pre-processing or undocumented parameters, the answer was quite unsettling as it did not seem to work under normal circumstances. In hindsight this was probably due to errors earlier in the process which were missed, and in some way corrupted *.com*-files. After this e-mail conversation decision was made to focus on the usage of *batchruntomo* on the cluster and perform a performance analysis instead of focusing on the development of a script able to automate the process of tomography.

3.3 Batchruntomo

In later versions of *IMOD* such a script is already shipped: *batchruntomo*. This software is technically the same as the proposed method, as it is a wrapper around the different tools which should be used in the process. Furthermore, this script already includes features such as checks in the code for already existing folders and data, and the possibility to send an e-mail when the process is finished. The developers also had found solutions for the many problems we encountered when trying to implement our script: For example the *3dmod* problem, where it was not possible to automate *3dmod* as it required buttons to be clicked, and there was no standard function to instead evade this clicking. In *batchruntomo*, this is solved by not using *3dmod* at all, while also not being dependant on *Raptor* due to that, as the developers implemented *autofidseed* which works about as good as when using *3dmod* and therefore much better than *Raptor*, at least in our case.
As there was no tutorial available at that moment describing the usage of *batchruntomo*, step one consisted of for one discovering the functionalities, and for two the correct way to use these functionalities and the script as a whole. The version which brought *batchruntomo* had, as apparently is common with the tools in the *IMOD* package, a man page[12] included about the tool describing the options and the manner to call it in a terminal. However, as found out quickly enough, that was really only about the terminal usage, and there was more work needed before a data set could be run with it (by now this has changed and the man page shows information on the templates, directives, and their usage). Initial problems experienced with the tool were for one that the choice of options was not complete as defining where the output should be put did not have a default, and for two as there were certain dependencies missing. These dependencies were required files to be filled by the user and consist of three general templates and one data set specific directive, which all contain parameters to run *batchruntomo*.
The template files hold parameters which do not need to be dedicated to a specific data set, while the dedicated directive file can hold almost all, but is mainly used for parameters such as those dedicated to the name of the data set and whether one or two axis should be used. The parameters are of the form <parameter>=<value>. All possible parameters, their type and whether they can be placed in a template file are listed online[13]. There are three template files, in the hierarchy of scope, system and user. Parameters appearing in later files overrule the corresponding parameters given in previous. The scope template is meant to be dedicated to the used electron microscope. Here one can state a default set of directives for the microscope it is dedicated to, such as the voltage of the microscope which in general will not be overruled in later templates. The system template, a step higher in the hierarchy, is meant for a specific computer system. For example, when multiple data sets from one microscope are processed on multiple computers, some may be slower than others and therefore less heavy -on the computer- parameters may be used such as having *beadtrack* running less iterations. And finally, there is the user template, which is dedicated for a certain user on a system: a user can have certain de-

faults which another user does not, such as having patch tracking enabled. In our case, as there was only one microscope, one system and one user, one of the template files was filled in while the others were left empty (it does not matter which template is filled in and which is empty for this situation). The template files do not need to be stored in certain directories, as their locations are to be given in the directive file, however there are default locations to store them: The system and scope template are by default stored in respectively the folders SystemTemplate and ScopeTemplate within the *$IMOD_CALIB* directory. The user template is by default stored in the *.etomotemplate* folder in the *$HOME* directory.
Having successfully determined how *batchruntomo* is operated and can be configured, it was time to test how the program would behave with a data set. At first, tries were made with the tutorial set for *Etomo* on a stand-alone computer to be able to determine basic errors such as problems with arguments in a familiar environment. After seemingly successfully running *batchruntomo* on this stand-alone computer with the tutorial set, it was tried on the cluster. As the data set will be moved to the destination given each test, there is a need to copy the data set each time from a backup-directory to the place one does want *batchruntomo* to get the data set from. To achieve this, the initial script for automating testing did some unnecessary work: It copied a test set from a fileserver determined to the storage of the datasets called *Rosalind* to a special *Input* folder on the main fileserver, after which *batchruntomo* moved it to a folder called *Result* and there ran its tools on this dataset. As this progressed each round for possibly hundreds of rounds the time needed for a complete experiment became enormous. To reduce this time *batchruntomo* was slightly rewritten to *batchruntomo.no-move*, which does not move the dataset but instead only copies it. In *batchruntomo.no-move* the line *os.rename(source, dest)* was replaced by *shutil.copy(source, dest)* in the functions *deliverAncillary* and *deliverStack*.

3.4 Experimental setup

Measurements were done on a representative dataset. After running the dataset a few times with these parameters on the cluster the results were validated and found to be correct. The parameters for this set can be found in the appendix, Section A. With these batch and template files the dataset was run several hundreds of times with different architectural configurations as parameters to determine the optimal architectural configuration to run the tool on the cluster. An architectural configuration is meant as a combination of nodes and processors per node as available resources for a job. The script with which the jobs initially were submitted is shown in Figure 10.

The script lets each architectural configuration run 10 times, using between 1 and 9 nodes per job and using between 1, 2, 4 and 8 processors per node, giving a total of 360 runs. Singular characters in the script are variables which just mean to represent a number in either string or integer format, to make sure all the jobs are getting the right parameters and no job is launched with the same number: This is used to differentiate between jobs more easily.
Job options are written to a 'jobfile', which mainly means they are written to an external file which qsub, the queue submission tool for *Torque*, can find and use instead of relying on the stdin. Options for qsub are denoted as *#PBS ...*,

```
#!/bin/python

import sys, os, shutil
from subprocess import call

greaterIterations = 10 #to ensure some spread data
iterations = 10

if len(sys.argv) > 1:
 iterations = int(sys.argv[1])

resultDir = "/home/tomography/ResultatenSimon"

#remove previous results
if os.path.isdir(resultDir):
 shutil.rmtree(resultDir)
os.mkdir(resultDir)

qbeta = resultDir + "/Qbeta"
q = 1
for n in [1,2,4,8]: #amount of cores
 o = str(n)
 p = q * 100
 q += 1
 #amount of iterations per node/core config
 for i in range(0, greaterIterations):
  j = i*10
  for k in range(1,10): #amount of nodes
   l = str(k)
   m = str(j + k + p)
   os.mkdir(qbeta + m)
   jobname = "qbatch" + m + ".job"
   with open(jobname, "w") as jobfile:
     jobfile.write("#!/bin/sh\n\n
          #PBS -l nodes=" + l + ":tomography:ppn=" + o +
              ",cput=5:00:00,walltime=5:00:00\n\ncd
              $PBS_O_WORKDIR\n\npython nodefileToAdoc
              $PBS_NODEFILE " + o + "\n\nsh runqbatch.sh "
               + m + " " + l + " " + o)
   call(["qsub", jobname])
```

Fig. 10: Script with which jobs are submitted. Small changes were made between different experiments, to for example also account for the amount of threads or to increase the cpu- and walltime.

which in the script in Figure 10 only applies to the *-l* option, describing the amount and type of resources needed for the job. In this script, the amounts of nodes (*nodes=*) and processors (*ppn=*, or *processors per node*) are determined by values given by for-loops, with as static string *tomography* denoting that only nodes with the classification string *tomography* may be used for the job, and *walltime* and *cput* which are respectively denoting the maximum amount of real time a job may run and the amount of CPU time a job may run. Both are set to high values to prevent any job being stopped before it is finished. *cd $PBS_O_-WORKDIR* ensures the job is being run in the right directory. *nodefileToAdoc* is a python script which made sure the nodes with their corresponding amount of available resources was written to a *cpu.adoc* file, of which we at that time were unaware *batchruntomo* would not look at. At last, the command is given to run the shellscript *runqbatch.sh*, which is shown in figure 11.

```
#!/bin/bash

i=$1
nodes=$2
procs=$3

startTime=$(date +%s)

Batchruntomo.no-move -ro 140502_Qbeta_tomo2_0 -current /
    home/user/tomo/Qbeta -deliver /home/tomography/
    ResultatenSimon/Qbeta${i} /home/user/tomo/
    batchQbetatest.adoc

endTime=$(date +%s)
elapsedTime=$(($endTime - $startTime))
echo "Qbatch ${i} with $nodes nodes and $procs processors
     needed $elapsedTime"
```

Fig. 11: Script with which *batchruntomo* was executed when a submitted job was executed.

This shellscript takes as arguments the number which will be in the jobname, the amount of nodes and the amount of processors per node available. It will also log the system time before and after the execution of *batchruntomo* so that the runtime of *batchruntomo* can be computed. In the end, these values are put out in the output file of the job, where all output processed within the scope of the job will be put. *batchruntomo* has the following command line arguments:

- The name of the dataset with parameter option *-ro*.
- The directory the dataset is currently in with parameter option *-current*.
- The directory the dataset will go to with parameter option *-deliver*.
- The batch file with the options for the current dataset, which can be preceded by the parameter option *-directive* although the last element is considered to be the batch file.

The used batch file is shown in the appendix, Section A, and as stated there the template files which also will be used are linked in the batch file.

```
qsub -n -l nodes=1:tomography:ppn=8,cput=5:00:00,walltime
    =5:00:00 runqbatch.sh
```

Fig. 12: The command with which qsub is called directly from the command line. To achieve multiple values of nodes and processors per node for-loops can be added in Bash.

```
qsub -n -l host=core-014,nodes=core-014:ppn=8,cput
    =5:00:00,walltime=5:00:00 runqbatch.sh
```

Fig. 13: The command with which *qsub* is called directly from the command line. Notice that ccore-014 is now both the host and all the nodes *qsub* will allocate for the shell script.

Later on the usage of Python scripts to execute *qsub* was discontinued and instead of generating a jobfile the command with which *qsub* was called was directly executed in the command line as seen in Figure 12. Note that there are no variables denoting the amount of nodes or processors per node: This could easily be done by adding for-loops, however at the time we switched there was no need to do so as the commands were run on a single node. When using this option, the command becomes as in Figure 13. Here *host=* means the shell script will be run from the specified node, instead of from the location from which the command is issued which in this case is the fileserver. Therefore it will also look for the resources which are possibly requested by the shell script on that specific node, such as the template files when running *batchruntomo*.

4 Results

In this section the results given by the experiments will be presented, and the story of why certain experiments were performed. At first we will talk about the initial experiments, then we will continue discussing the effects of network I/O. Then we will look at the performance of a single node setup, followed by discussing the fluctuations in the results, and lastly we will look at the performance of a multiple node setup.

4.1 Initial experiments

Experiments have been performed with all possible configurations described in Section 3.4. The results are shown in Figure 14. Each bar shown in this figure is the average time resulted from the architectural configuration consisting of the amount of nodes on the x-axis and amount of cores per node corresponding to the color of the bar. The average is taken of a varying amount of samples: of some configurations there are ten successful samples, of others there were only two, or even zero as has the 9 nodes 1 core per node. The reason of this lack of samples is that a lot of the spawned processes had aborted. The exact reason is unknown to us as the error messages were ambiguously only stating the fact that it aborted and where. The only pattern to be found within these error messages was the fact that it was always *processchunks* which failed and not always at the same spot. *processchunks* is the tool of *IMOD* which manages the spawning and regulating of parallel processes. An abort with this tool indicates there is an error in either the spawning of the parallel processes or the communication between them. As the spawning of parallel processes seemed to work fine as it almost always passed the first few tools used, of which *newstack* which is run in parallel, it is very well possible the error came forth due to communication errors between the parallel processes.
A possible explanation for this communication error has to do with the amount of instances of *batchruntomo* running at the same time and the I/O generated by each of these instances which had to be processed by the filesystem. As this communication was not only between nodes, but also with the fileserver, and with every communication the dataset or part of it had to be transmitted to either a node or to the fileserver there are a lot of GB (waiting to be) transferred every second. As the queue of outstanding I/O requests grows larger and larger some processes might give a time-out on the transmission and abort. This would also explain the timing of each abort in the instances of *batchruntomo*: As it is completely dependant on the current load of the fileserver the abort can happen anywhere in the process.

These I/O problems could also explain the time differences between different architectural configurations, as the overhead on the transmission of all the instances differs between different points in time, some configurations might take a longer time than other, which also is seen in Figure 14.

4.2 Effects of Network I/O

To test the effects of the I/O overhead, at first neglecting the time needed to transfer the dataset on and over to the nodes and back to the fileserver,

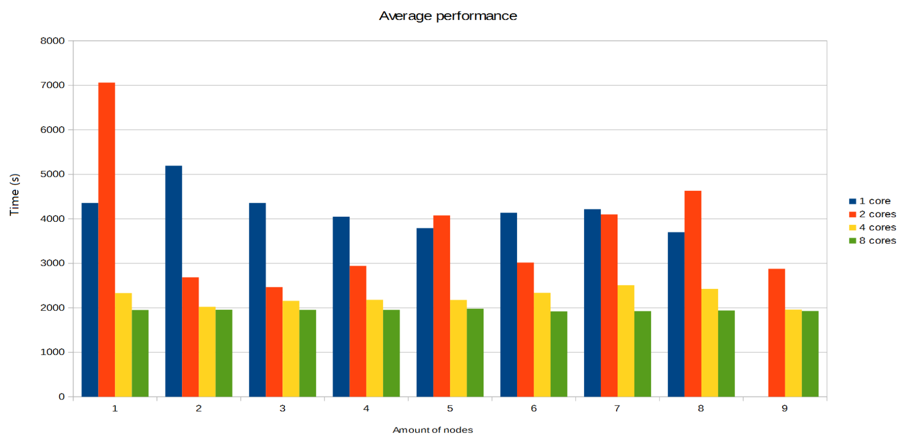


Fig. 14: Average runtimes of the first experiment. The absence of the '1 core' bar at '9 nodes' on the x-axis is because all of these runs aborted. The high fluctuations were at this point less an issue than the high times.

batchruntomo was run again with the same dataset and parameters but now only for the configurations with 8 cores. Constraining the experiment to only include the configurations with 8 cores was as these configurations had the lowest times in the first experiment and were therefore most probable to also be recommended in the end conclusion. This experiment consisted of spawning one instance of *batchruntomo* at a time, with every next instance spawning two hours after the previous spawned. As each instance would take less than 40 minutes, which was the highest time it took in the previous experiment, this would give enough time for each instance to finish on its own without having to wait for the I/O of the other instances.
The results of this second experiment are plotted in Figure 15 and show no improvement over the instances which, by our hypothesis, were severely affected by I/O performed by other instances and therefore allowed for a speed-up in that regard. Therefore was concluded the main reason of the high differences between the times of each configuration is the I/O of each individual instance, still depending on the state of the cluster, but not as much depending on the I/O of the other instances as expected, at least for configurations with 8 cores.

4.3 Single Node Performance

To determine the effect of the interaction with the fileserver, experiments were performed with a single instance on a single node with a single core and a multiple amounts of threads. Performing the experiment on a single node is achieved by copying the dataset to a node's local storage, and then issue *batchruntomo* to take the dataset from and deliver it to this node. To completely eliminate the fileserver from the experiment, the directives and templates also had to be stored locally, and the PATH variable had to be changed to point to the new *IMOD_CALIB_DIR* folder. Tests were run in the way used earlier, using *Torque* with the *Maui* scheduler to schedule the processes as jobs on this single node, just because that way the process would not shut down if the connection was

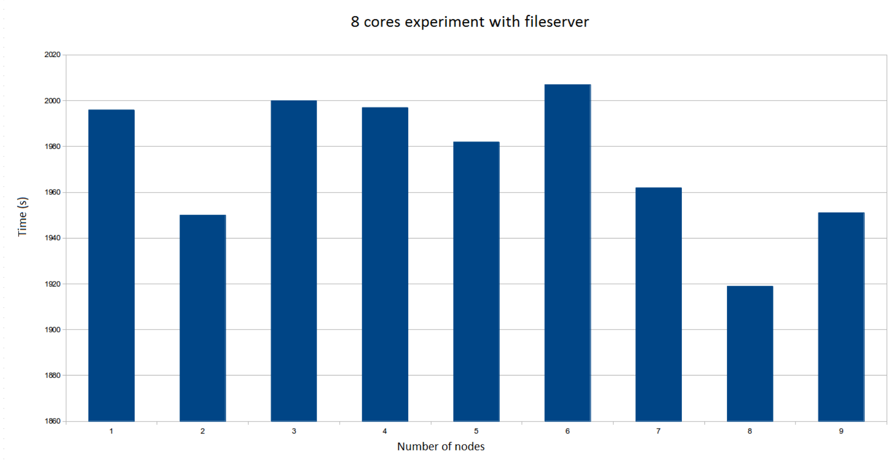


Fig. 15: Average runtimes of the 8 cores experiment with the fileserver. Each run was done two hours after the start of the previous, making sure the runs did not interfere with each other.

broken. To specify the number of threads used to *batchruntomo*, *~/.bashrc* was altered by adding the line *export OMP_NUM_THREADS=1*, where *1* could be changed by the amount of threads we wanted to use, and sourced afterwards. This knowledge was shared with us by the developer of *batchruntomo*, as can be read in Figure 4.3. In the script, we did this by adding the code in Figure 16.

```
sed -i '5s/.*/export OMP_NUM_THREADS=${j}/' ~/.bashrc
source ~/.bashrc
```

Fig. 16: Bash code to specify the number of threads used by *batchruntomo*.

Ensuring the use of only 1 core on this node was done by not using 'chunking', as chunking allows *batchruntomo* to be run with multiple cores. To enable this option the parameter *-cpus*, followed by a number corresponding with the number of cores one wants to allow *batchruntomo* to use, has to be added to the execution command of *batchruntomo*. However, when not adding this parameter to the call of *batchruntomo* it will use only 1 core.

The results were surprising. Not only was it faster without the fileserver, it was a factor 6 faster than the quickest runtime observed up till now: $\frac{1919}{320.2} \approx 6$. The durations of the tools of which *batchruntomo* consists are plotted in Figures 19, 20 and 21. The legend is shown in Figure 18.

There are two parts to this.
1) Batchruntomo pays no attention to cpu.adoc or the IMOD_PROCESSORS setting. The number of cores it uses for parallel operations that are chopped into chunks is controlled by the -cpus entry. Without this entry, it uses one core for these operations.
2) More likely you are seeing uncontrolled thread usage by other operations that involve a single run of a program that is partially utilising parallelism with OpenMP. Newstack and tiltxcorr are two examples of this. Here the number of threads can be limited by the environment variable OMP_NUM_THREADS, so you would need to run
export OMP_NUM_THREADS=4
This will then limit the thread usage of any other programs run with that setting in effect (i.e., in the same terminal.
Just like when running these operations through the interface, Batchruntomo doesn't limit the thread usage of these programs. That seems wrong to me, and it ought to limit them somehow. At least for the case of purely local processing, and I could see making the default limit be the same as the number of cores specified. It is not clear what to do when multiple machines are specified with a few cores on each.

Fig. 17: Mail about *cpu.adoc* from David Mastronarde

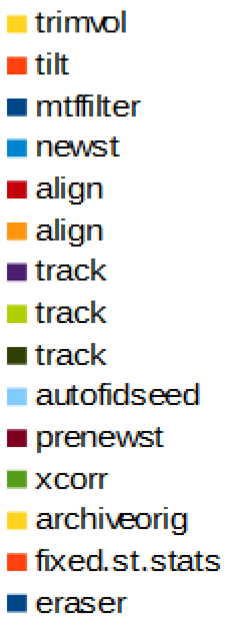


Fig. 18: The legend of the experiments 'without fileserver'.

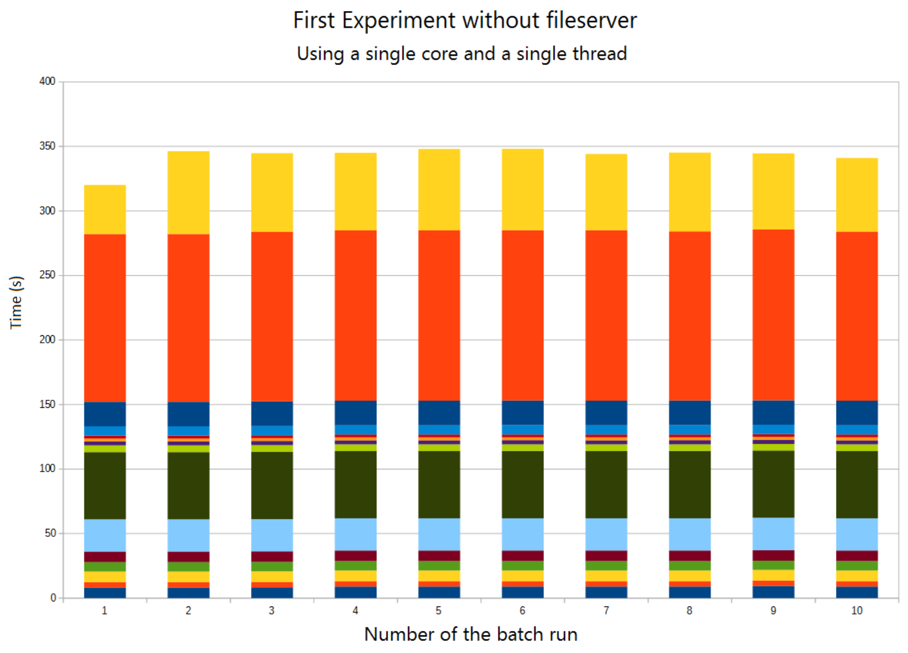


Fig. 19: First experiment without fileserver, utilizing a single node, a single core, and a single thread.

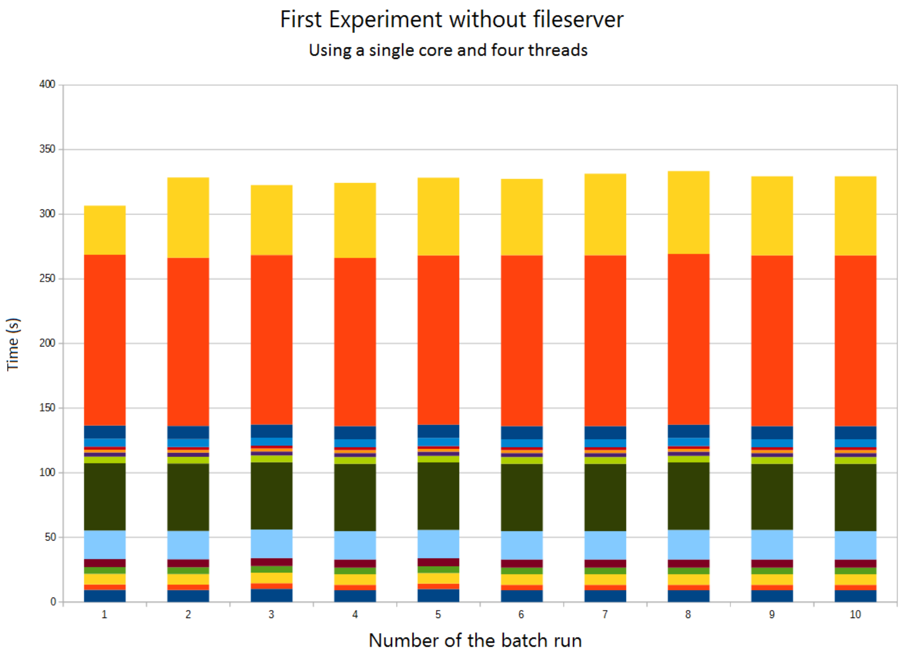


Fig. 20: First experiment without fileserver, utilizing a single node, a single core, and 4 threads.

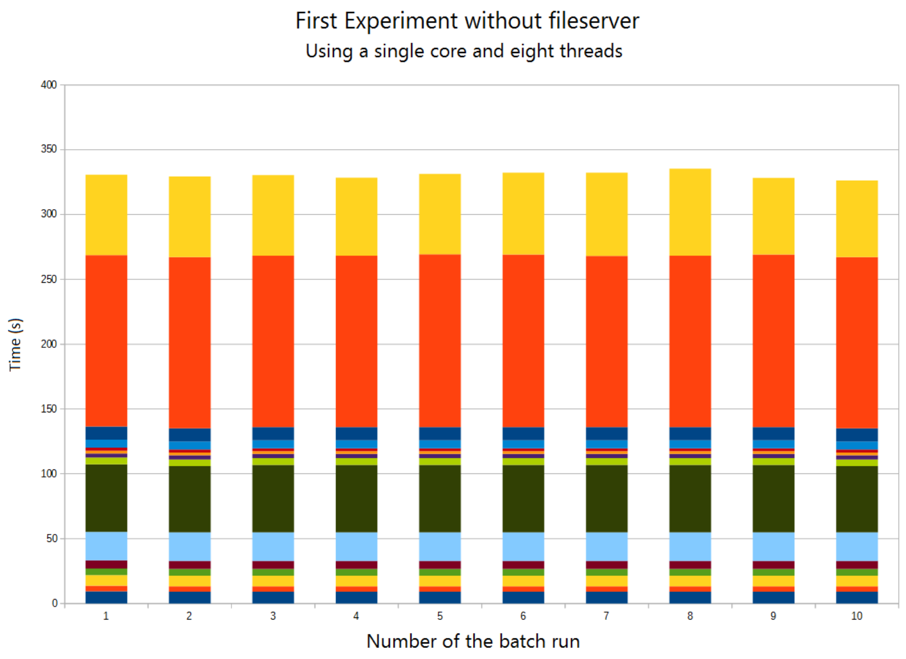


Fig. 21: First experiment without fileserver, utilizing a single node, a single core, and 8 threads.

As can be observed some lines are a bit variable, but there is a more heavy fluctuating first bar in comparison to the others in Figures 19 and 20. To test whether the fluctuations were consistent a second experiment was performed which was the same as the previous experiment but had more runs: each bar in Figures 22, 23, 24 and 25 is the average of 10 instances instead of a single instance. This shows the fluctuations are persistent, on which we will continue in the next Section.

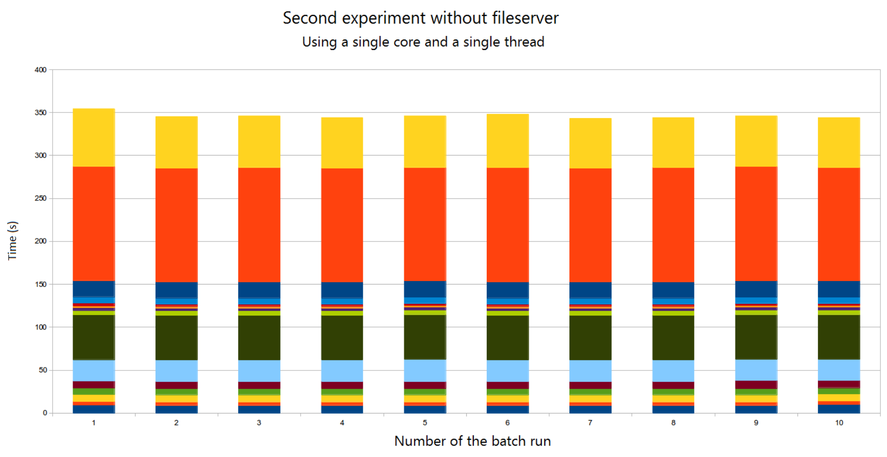


Fig. 22: Second experiment without fileserver, utilizing a single node, a single core, and a single thread. Each bar is the average of 10 instances.

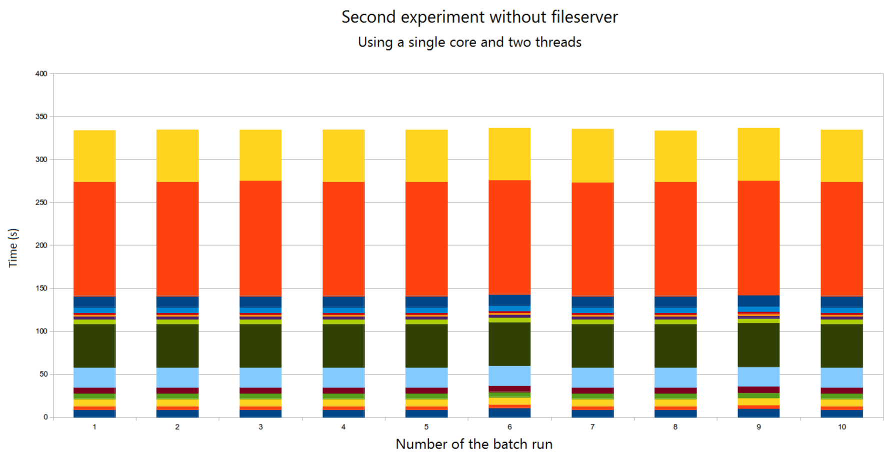


Fig. 23: Second experiment without fileserver, utilizing a single node, a single core, and 2 threads. Each bar is the average of 10 instances.

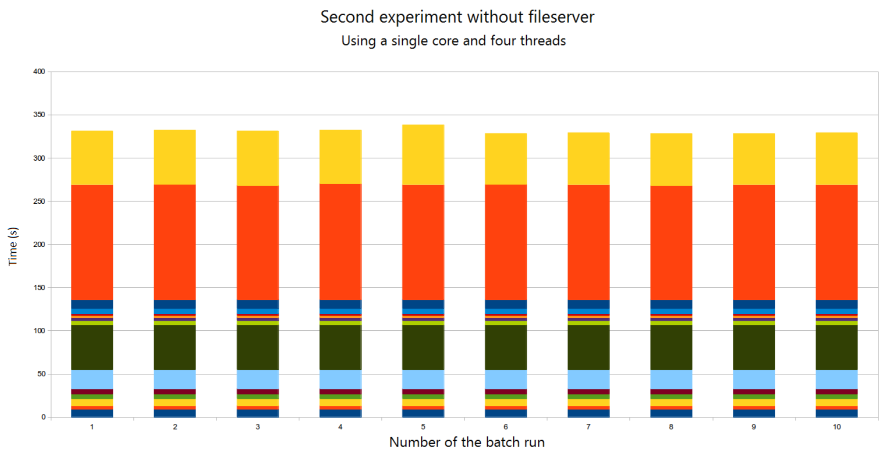


Fig. 24: Second experiment without fileserver, utilizing a single node, a single core, and 4 threads. Each bar is the average of 10 instances.

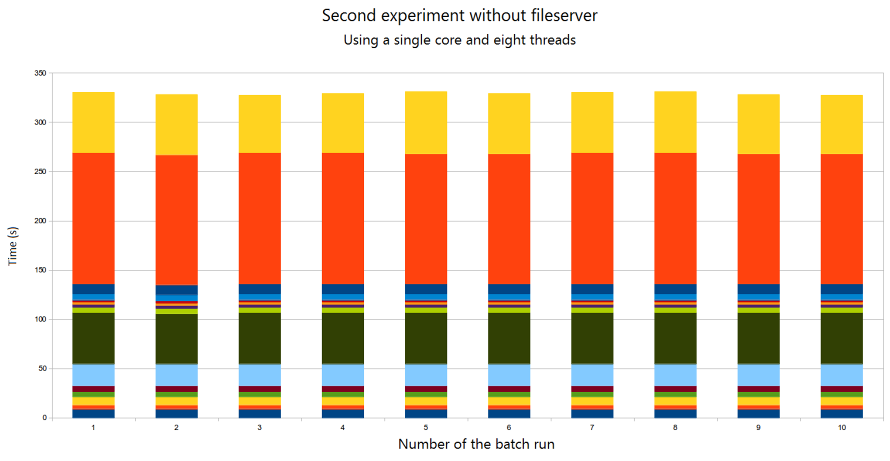


Fig. 25: Second experiment without fileserver, utilizing a single node, a single core, and 8 threads. Each bar is the average of 10 instances.

After that, chunking was enabled to see how much faster that would make *batchruntomo* on a single node. Enabling chunking was, as said before, done by adding the *-cpus* parameter to the command, altering the command to as seen in Figure 26.

```
Batchruntomo.no−move −cpus ${amountOfCores} −ro 140502
    _Qbeta_tomo2_0 −current /home/user/tomo/Qbeta −deliver
     /home/tomography/Resultaten/Qbeta${number} /home/user
    /tomo/batchQbetatest.adoc
```

Fig. 26: Command with which *batchruntomo* can be called while using chunking.

We therefore ran an experiment with 1, 2, 4 and 8 cores and 1, 2, 4 and 8 threads. The experiment with 1 core and 1 thread already are seen in a previous graph, but are also added in this graph as reference material. The results of this experiment are in Figure 27. In this graph the amount of cores used is shown in ascending order on the x-axis, the time used in seconds on the y-axis, with the different colored bars representing the amount of threads used, in ascending order. Each bar is an average of 5 batches in the experiment with the same architectural configuration.

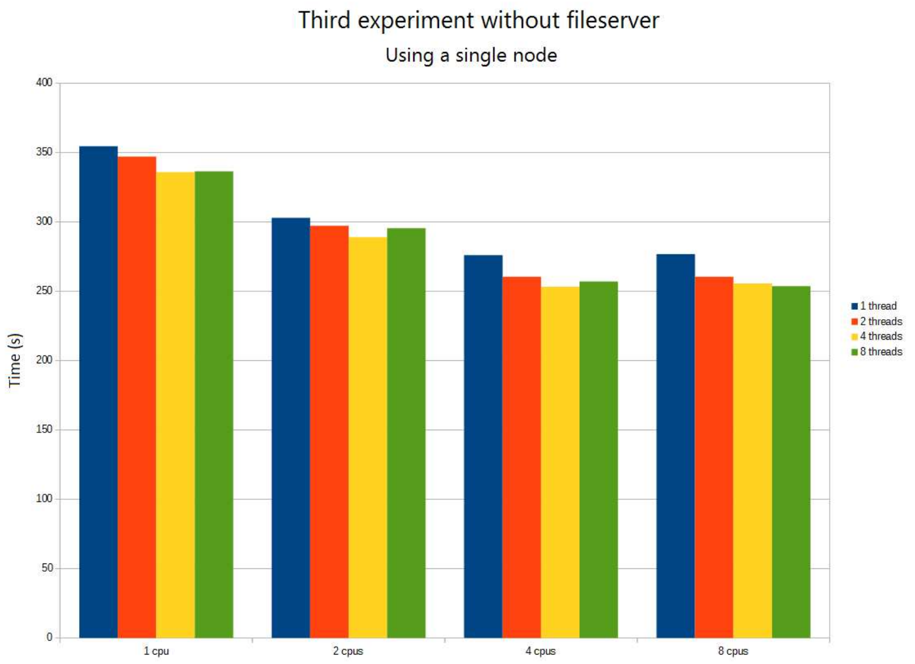


Fig. 27: Experiment on a single node which makes use of multiple amounts of threads and multiple amounts of cores (so it is both using and not using chunking), without making use of the fileserver.

4.4 Fluctuations

The fluctuations found in the results are caused by *trimvol*, as the other tools of *batchruntomo* at most differ about a second in time between different batches while *trimvol* may differ up to about 20 seconds. *trimvol* is “a command-line interface to the programs *findcontrast* and *newstack*”[14], of which *newstack* is used earlier in the process and already is known to be close to constant in time. *findcontrast* therefore was thought to be the source of the fluctuations, as it functions to find “the absolute minimum and maximum pixel values within the selected volume”[15] of an image. However, the code of *findcontrast* did not use random-function or anthing alike, and therefore was neither the source of the fluctuations.

Ensured it is not a random-function causing the fluctuations, the probable cause of the fluctuations becomes I/O. First thought is the influence of I/O operations, both scheduled but not yet performed and performed but not yet written to memory (only stored in buffer). The I/O operations which are scheduled but not yet performed can be checked using the shell command

```
cat /sys/block/sdb/stat
```

and then looking at the ninth column which contains the number of these operations at that moment. As this number was always zero, this could neither be the case. The amount of dirty pages is returned in *kB* with the command

```
cat /proc/meminfo | grep "Dirty"
```

A dirty page is a page, or a piece of information, which is stored in a buffer but not written to the memory: the time required to write these pages to the memory can take multiple seconds, depending on the size. If this would happen sometimes during the execution of trimvol and otherwise not, this could explain the fluctuation. Therefore a new experiment was performed, in which the dirty pages were synced with system call *sync* right after *trimvol*. To implement this in *batchruntomo* the following code was used:

```
import ctypes
libc = ctypes.CDLL("libc.so.6")
sync = libc.sync()
```

as it uses *Python2*.
As shown in Figure 28 and more specifically the times of *trimvol* in Figure 29, this seems to lessen the fluctuation, but does not lessen it to the extent where it can be concluded that dirty pages are the prime cause of the fluctuations. Even more, the amount of dirty pages does not seem to even correlate with the fluctuations: Another experiment, seen in Figures 30 and 31 shows respectively the times and the amount of dirty pages (and the pending I/O operations, which was always 0), shows the lack of this.
As a next step to explain the fluctuatons, we turned to the buffer cache. The buffer cache would be filled with the data of the process in the first batch. If the buffer cache is not cleared, or the files of the first batch are not removed (whereas removing the files also accomplishes the buffer cache to be cleared), then the buffer cache is available for subsequent runs. This could lead to different run times for subsequent runs, depending on how the page/buffer cache algorithm is tuned. To get an idea whether these effects can be measured, another experiment was done where the results are deleted right after running a batch. The results of this experiment are shown in Figures 32 and 33. As can be seen, the fluctuations lessen drastically when the cache is cleared in this way, to the extent where we can conclude the fluctuations are caused by the cache usage of the latter batches when the cache is not cleared.

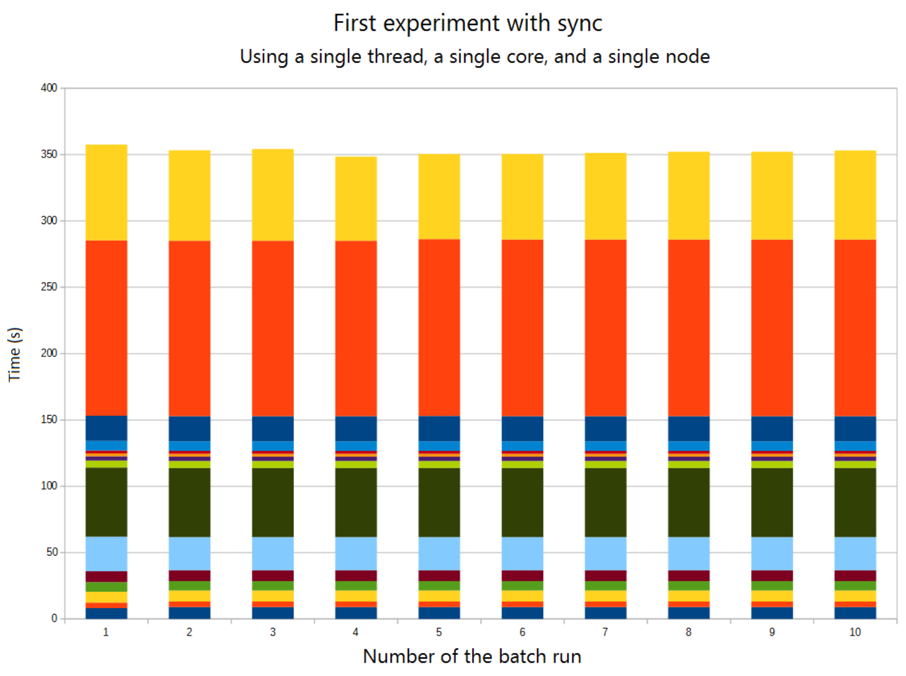


Fig. 28: First experiment with sync, using a single thread, a single core, and a single node. In this experiment sync is added right after the execution of trimvol, and added to the time which is measured for trimvol.

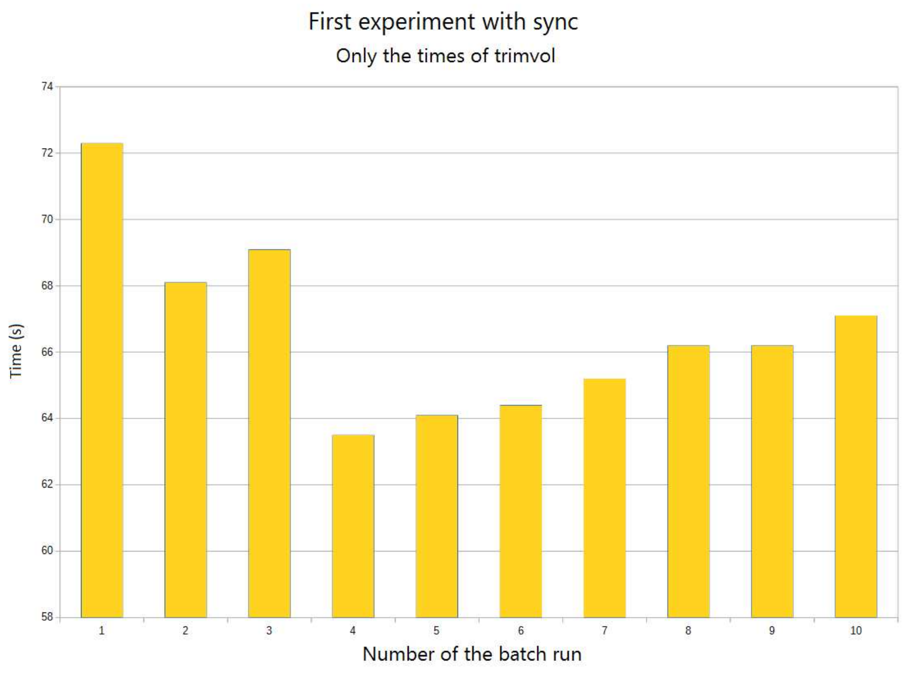


Fig. 29: First experiment with sync, using a single thread, a single core, and a single node. In this graph only the times of trimvol, including the sync, are shown.

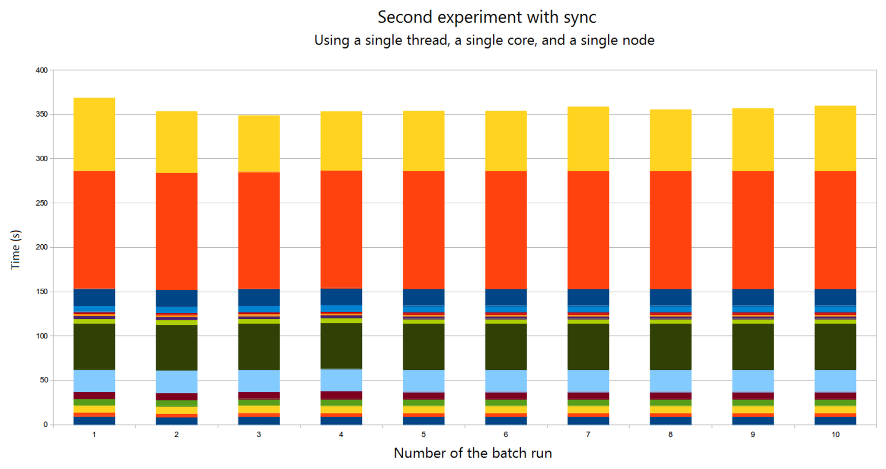


Fig. 30: Second experiment with sync, using a single thread, a single core, and a single node. This experiment is run to see if there is any correlation between the time and the amount of dirty pages and outstanding I/O processes, which are seen in Figure 31.

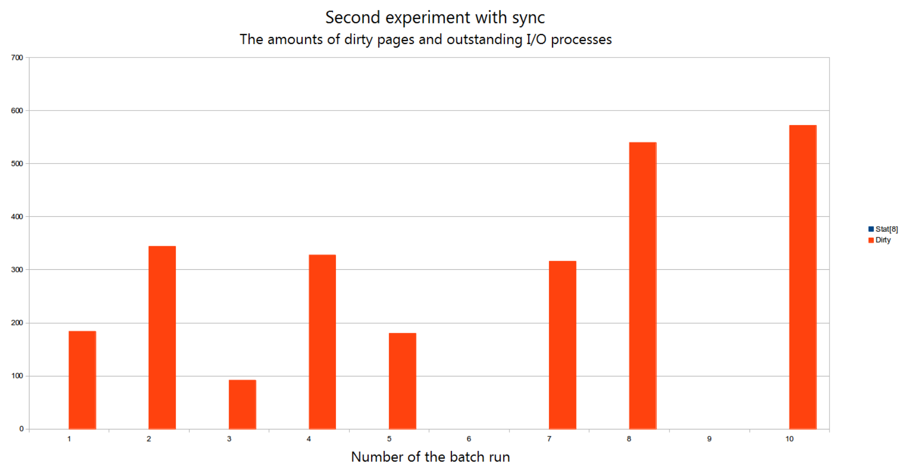


Fig. 31: Second experiment with sync, using a single thread, a single core, and a single node. This experiment is run to see if there is any correlation between the amount of dirty pages and outstanding I/O processes and the time, which is seen in Figure 30. Note that the orange bar is showing the amount of dirty pages in *kb*, and the non-present blue bar would show the amount of outstanding I/O processes.

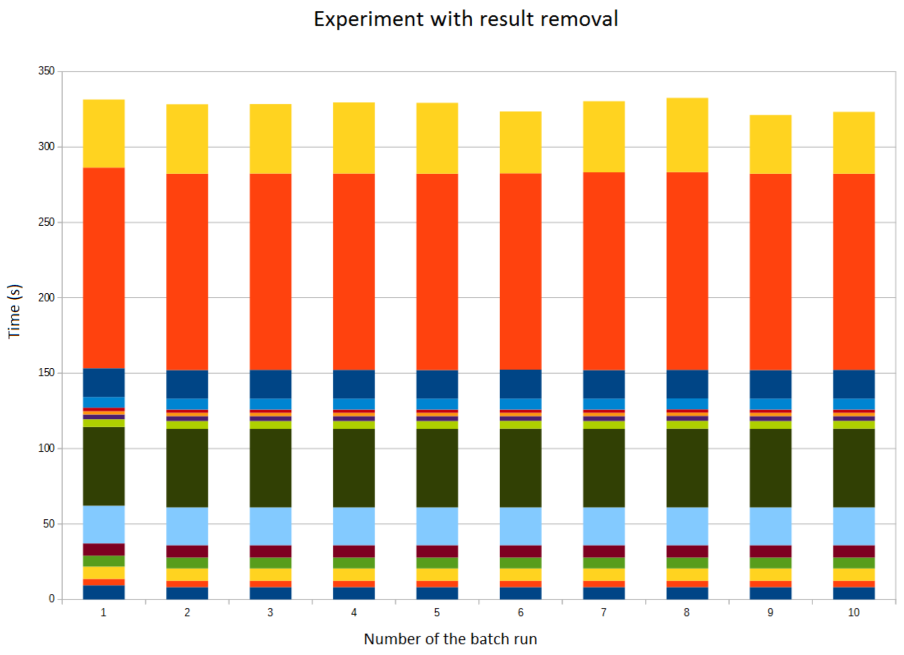


Fig. 32: This graph shows an experiment in which the results are immediately removed after a run of *batchruntomo*, to clear the buffer cache. That way, subsequent runs cannot use the cache to predetermine their results.

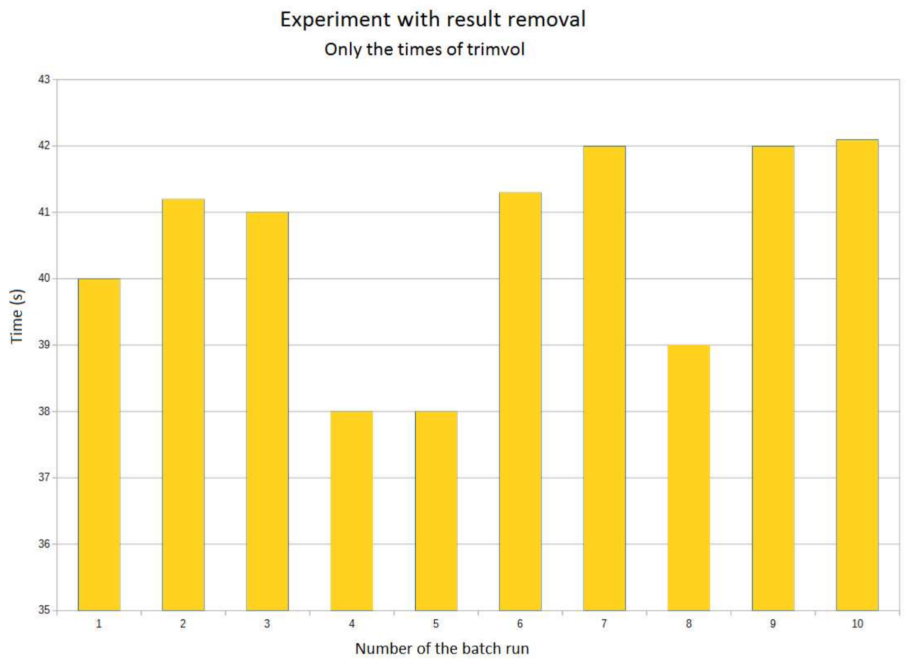


Fig. 33: This graph shows the same experiment as in Figure 32, however with only the times made by trimvol.

4.5 Multiple Node Performance

Having determined the cause of the fluctuations, the next step was to determine the effect of running *batchruntomo* on multiple nodes without the use of the fileserver. As chunking did result in a great improvement in speed and therefore a great reduction of time the different runs took, the use of multiple nodes could also have a comparable effect. Running *batchruntomo* on multiple nodes is done by having one node as a host, the same way as when running on a single node, and then making clear to *batchruntomo* which nodes it can use with which amount of cores by means of the -*cpus* parameter. This means the data is at the beginning stored at the host node, and will be processed on the host node as well. The code of a script which runs an experiment with *batchruntomo* on multiple nodes can be found in Figure 35. Notice how the nodes to be used with their respective amount of cores to be used are a string of format <node>#<amount of cores>, and then multiple of these behind each other with comma's in between. The results of this experiment are seen in Figures 34, 36, and 37.

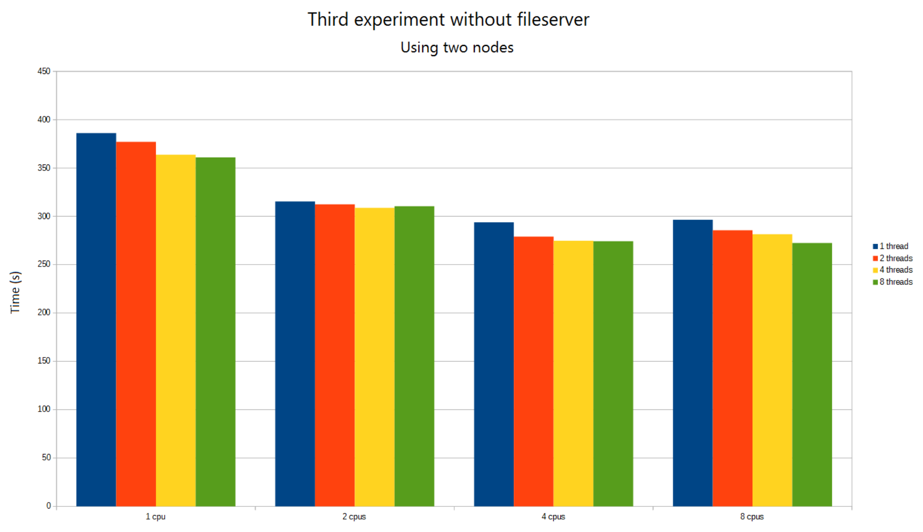


Fig. 34: Third experiment without fileserver, in which 2 nodes are used in combination with either 1, 2, 4, or 8 cores and 1, 2, 4, or 8 threads to show the effect on the time of an execution when using multiple nodes.

```
#!/bin/bash

declare -a totalCpulist=("core-015" "core-016" "core-017"
     "core-018" "core-020" "core-021" "core-012" "core-013
    ") #all nodes on the cluster with 8 cores

for l in 2 4 8 #amount of nodes
do
    for k in 1 2 4 8 #amount of cores
    do
        cpulist="core-014#${k}" #the host node, which is
            therefore preferred to be used each run
        n=$((l-2))
        for m in $(seq 0 ${n})
        do
            cpulist="$cpulist,${totalCpulist[m]}#${k}"
        done
        for j in 1 2 4 8 #threads
        do
            sed -i '5s/.*/export OMP_NUM_THREADS=${j}/' ~/.
                bashrc
            source ~/.bashrc
            for i in {1..5}
            do
                rm -rf /scratch/sklaver/resultaten/QbetaSync$
                    {i}${j}
                mkdir -p /scratch/sklaver/resultaten/
                    QbetaSync${i}${j}
                startTime=$(date +%s)
                batchruntomo.no-move.sync -cpus ${cpulist} -
                    ro 140502_Qbeta_tomo2_0 -current /scratch/
                    sklaver/testset -de /scratch/sklaver/
                    resultaten/QbetaSync${i}${j} /scratch/
                    sklaver/batchQbetatest.adoc
                endTime=$(date +%s)
                elapsedTime=$(($endTime - $startTime))
                echo "Qbatch ${i} with ${k} cpus, ${l} nodes
                    and ${j} threads needed $elapsedTime"
            done
        done
    done
done
```

Fig. 35: Bash script in which *batchruntomo* is executed multiple times, 5 times for each combination of an amount of threads, an amount of cores, and an amount of nodes.

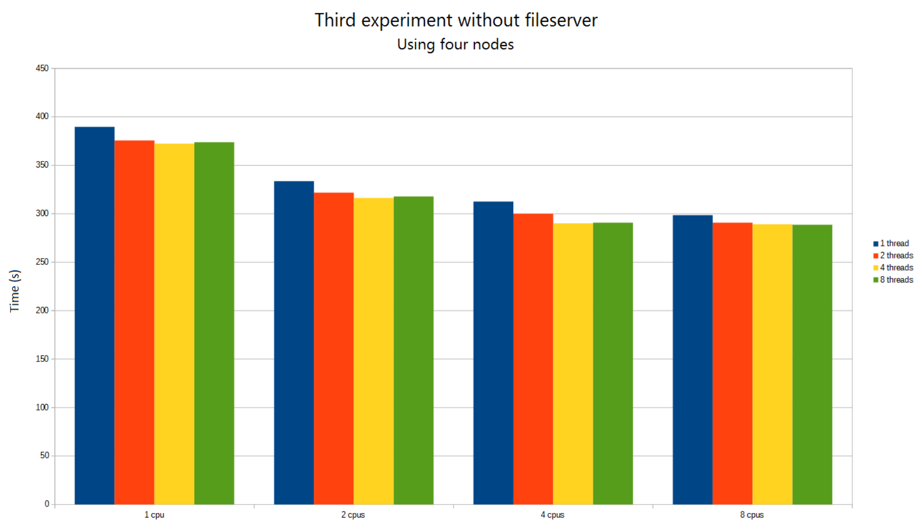


Fig. 36: Third experiment without fileserver, in which 4 nodes are used in combination with either 1, 2, 4, or 8 cores and 1, 2, 4, or 8 threads to show the effect on the time of an execution when using multiple nodes.

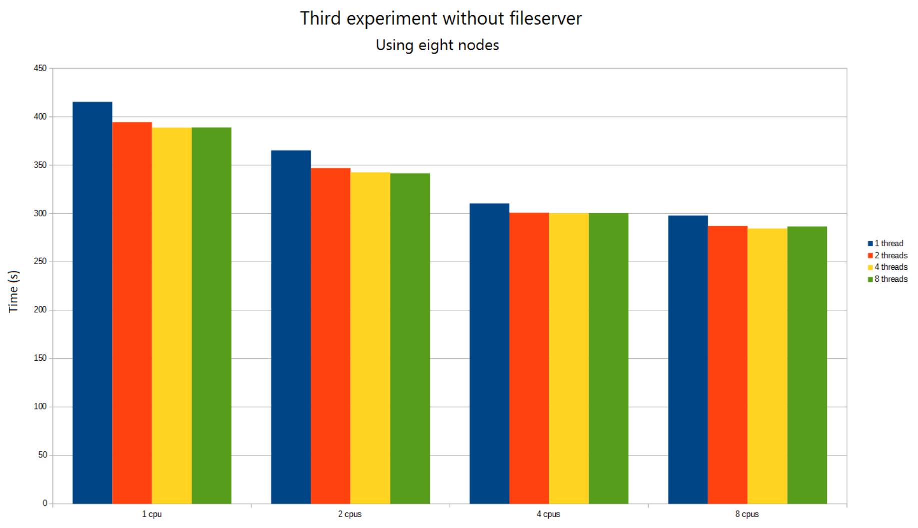


Fig. 37: Third experiment without fileserver, in which 8 nodes are used in combination with either 1, 2, 4, or 8 cores and 1, 2, 4, or 8 threads to show the effect on the time of an execution when using multiple nodes.

5 Conclusion & Discussion

In this section we will conclude the project. First we will give a recommended configuration, drawn from our conclusions when looking at the results. Then we will discuss the possible improvements for *batchruntomo*.

5.1 Recommended configuration

Comparing the results of the experiments, first to notice is the large difference in performance when *batchruntomo* has to use the fileserver as seen in Figure 14, compared to having the data stored locally such as in the experiment of Figure 22. Further investigation is necessary to find out how the fileserver is to be effectively utilized. Then, in the results of Experiment 1 and Experiment 2 without chunking, the results are fluctuating in total time, which was explained in Section 4.4, but above all there was no way to conclude the optimal amount of threads to be used without chunking, as there was no configuration with an amount of threads which clearly made *batchruntomo* faster compared to the other configurations. However, when using chunking, a speedup was present, as for example shown in Figure 27. Therefore we already recommend using chunking when using *batchruntomo*, of which the speed-up however stagnates between 4 and 8 cores. Multiple threads also stagnate in speed-up between 4 and 8 threads, giving a first conclusion that the use of multiple cores and threads above 4 is not useful.
When Figures 34, 36 and 37 are compared with Figure 27 we can see the lowest times for the different amounts of nodes:

1 node: about 250 seconds with 4 cores and 4 threads.

2 nodes: about 270 seconds with 4 cores and 4 threads.

4 nodes: about 290 seconds, with 4 cores and 4 threads.

8 nodes: about 290 seconds, with 8 cores and 4 threads.

Slight fluctuations might be the case, as mentioned before, but the results show a general trend, even more as the bars were each averages of five batches in the experiments. Therefore we can see that the addition of multiple nodes does not decrease the time used with multiple cores and threads available. Important to note is that the experiments were concluded with a dataset of about 2.6GB, meaning that with larger datasets the configurations could perform differently. Aside from possibly minor improvements in speed when using multiple nodes, or up to 8 cores, with larger datasets we do not believe they will vary much from our results.
The conclusion therefore is that the optimal configuration for the used dataset, regarding usage of at least as possible nodes, cores, threads, and therefore power, while minimizing the time needed for *batchruntomo* is the configuration which consists of 1 node, 4 cores and 4 threads, as higher amounts of resources did not show improved runtimes.

5.2 Possible improvements for batchruntomo

While *batchruntomo* works correctly when given the right parameters for a certain dataset, the user guide on it is incomplete and therefore too confusing. The man page does show some hints on how to fill in the templates and directives, but does not contain much information on the possible parameters themselves. Because of this the options to put in the directives and templates could be described more to inform the user of its effects and the proper value to give to the parameter in certain conditions. As of now, only with *Etomo* next to it and a few years of experience in using the software help one suffice in understanding the directives and the proper parameters to be filled in for certain data sets, and then it still may fail on the first few tries.
Another possible improvement, also in regard to the first one, is a graphical interface for *batchruntomo*, in which one can give in the different parameters for the different directives in an overlookable way, whereas the filling in of the parameters in a text file, which is the proper way to do it as of now, can be frustrating and confusing. The interface can than also hold fields for the parameters to give in in the terminal, including a simple file browser to look up a file and automatically let it be recognized by the program, as many of the probable users of this software will not know too much of the system they are working on and how the program behaves on it as the researchers who are doing tomography generally do not have too much knowledge of computer systems. As of *IMOD* beta-version 4.8.22 there is a graphical interface for *batchruntomo*, but as it does not seem to work correctly yet and is more an indication of what it will become, there is no attention given to it by us.
A third improvement can be the error handling of *batchruntomo*, or *IMOD* in general: As the errors differ between errors regarding the images to errors regarding the system, quite some errors are not understood by people working with it. Of course, excessive testing at the spot where the error occurs or simply asking it to a developer may help, but if the errors given would not just describe the error but also a possible solution this could help greatly. Or, more distant but still satisfying, a complete list of possible errors, for example sorted on the different tools and programs which give them, with an explanation of what went wrong and how to solve it in most cases. As the data sets are usually quite large and the directive files hold a lot of directives and corresponding parameters, it might be confusing which parameter is actually faulty, while testing if one has corrected it properly may also take quite some time, as not everyone has a large cluster available.

References

[1] "Titan Krios: They are for sale."
http://www.fei.com/products/tem/titan-krios/.

[2] E. Abbe, "Beiträge zur theorie des mikroskops und der mikroskopischen wahrnehmung," *Archiv für mikroskopische Anatomie*, vol. 9, no. 1, pp. 413–418, 1873.

[3] R. Erni, M. D. Rossell, C. Kisielowski, and U. Dahmen, "Atomic-resolution imaging with a sub-50-pm electron probe," *Phys. Rev. Lett.*, vol. 102, p. 096101, Mar 2009.

[4] dr. ing. Reinhold Ruedenberg, "Apparatus for producing images of objects," 1931.
http://worldwide.espacenet.com/searchResults?PN=DE906737.

[5] "Ccd detectors in high-resolution biological electron microscopy."
http://www.psu.edu/dept/hafenstein/pdfs/cdb37aa0a720e380f80e67c38ed8f97b.pdf.

[6] "Etomo tutorial."
http://bio3d.colorado.edu/imod/doc/etomoTutorial.html.

[7] D. N. M. James R. Kremer and J. R. McIntosh, "Computer Visualization of Three-Dimensional Image Data Using IMOD," 1995.
http://bio3d.colorado.edu/imod/paper/.

[8] "IMOD Official Homepage."
http://bio3d.colorado.edu/imod/.

[9] G. Staples, "Torque resource manager," in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, SC '06, (New York, NY, USA), ACM, 2006.

[10] N. van Veen, "Deploying single particle analysis." http://www.liacs.nl/assets/Bachelorscripties/Inf-Studiejaar-2013-2014/20913-2014Neal-van-Veen.pdf, 2014.

[11] "Maui scheduler." "
http://www.adaptivecomputing.com/products/open-source/maui/".

[12] "Batchruntomo manpage as of 23-4-2015."
http://bio3d.colorado.edu/imod/doc/man/batchruntomo.html.

[13] "Batchruntomo directives as of 23-4-2015."
http://bio3d.colorado.edu/imod/doc/directives.html.

[14] "Trimvol manpage as of 23-4-2015."
http://bio3d.colorado.edu/imod/doc/man/trimvol.html.

[15] "Findcontrast manpage as of 23-4-2015."
http://bio3d.colorado.edu/imod/doc/man/findcontrast.html.

Appendix

A Parameters Batchruntomo Experiment

Directive Adoc

```
#Created by Roman Koning 18 july 2014 for testing Qbeta
    dataset
#Arguments that are ONLY present in batch.adoc are
    denoted here, for all (including these) see the user.
    adoc file

#Arguments to copytomocoms

setupset.copyarg.name=140502_Qbeta_tomo2_0

#Other setup parameters

setupset.scopeTemplate=
    /home/user/ImodCalib/ScopeTemplate/scopeKrios1.adoc
setupset.systemTemplate=
    /home/user/ImodCalib/SystemTemplate/systemCryo.adoc
setupset.userTemplate=
    /home/user/.etomotemplate/userRoman.adoc
setupset.scanHeader=1
setupset.datasetDirectory=
    /home/user/tomo/Qbetatest

#Preprocessing

runtime.Preprocessing.any.removeXrays=1
runtime.Preprocessing.any.archiveOriginal=1

#Beadtracking

runtime.BeadTracking.any.numberOfRuns=5

#Auto seed finding

comparam.autofidseed.autofidseed.MinGuessNumBeads=5

#Aligned stack choices

runtime.AlignedStack.any.correctCTF=0
runtime.AlignedStack.any.eraseGold=0
runtime.AlignedStack.any.filterStack=1

#Aligned Stack Parameters

runtime.GoldErasing.any.extraDiameter=2

```

```
#Reconstruction

runtime.Reconstruction.any.extraThickness=100
runtime.Reconstruction.any.doBackprojAlso=1

#Postprocessing

runtime.Trimvol.any.reorient=1
```

Scope Template Adoc

```
#Made by Roman Koning, 16 july 2014,
#for Titan Krios 1 using GIF at 300 keV

#Arguments to copytomocoms

setupset.copyarg.voltage=300
setupset.copyarg.Cs=2.7

#We have not yet recorded noise files for the Titan Krios
    1
setupset.copyarg.ctfnoise=
    /home/user/NoiseFiles/ImodCalib/CTFNoise/F20/
        F20_bin1_linux.cfg
```

System Template Adoc

```
#Made by Roman Koning, 18 july 2014,
#for Titan Krios 1 using GIF at 300 keV
#This file only contains arguments that are in both
#batch and template files and therefore is (or at
#least should be) fully complementary (without
#double or missing entries) to the batch.adoc

#Arguments to copytomocoms

setupset.copyarg.dual=0
setupset.copyarg.montage=0
setupset.copyarg.pixel=0.72
setupset.copyarg.gold=10
setupset.copyarg.userawtlt=1
setupset.copyarg.extract=1
setupset.copyarg.binning=1
setupset.copyarg.twodir=0
setupset.copyarg.binning=1
setupset.copyarg.defocus=-5000

#Preprocessing

comparam.eraser.ccderaser.PeakCriterion=10
comparam.eraser.ccderaser.DiffCriterion=8
comparam.eraser.ccderaser.MaximumRadius=2.1

#Coarse alignment

comparam.xcorr.tiltxcorr.FilterRadius2=0.25
comparam.xcorr.tiltxcorr.FilterSigma2=0.05
runtime.Fiducials.any.fiducialless=0
comparam.prenewst.newstack.ModeToOutput=

#Tracking choices

runtime.Fiducials.any.trackingMethod=0
runtime.Fiducials.any.seedingMethod=1

#Beadtracking

comparam.track.beadtrack.LightBeads=0
comparam.track.beadtrack.LocalAreaTracking=0
comparam.track.beadtrack.SobelFilterCentering=1
comparam.track.beadtrack.KernelSigmaForSobel=1.5
comparam.track.beadtrack.RoundsOfTracking=5

#Auto seed finding

```

```
comparam.autofidseed.autofidseed.TwoSurfaces=1
comparam.autofidseed.autofidseed.TargetNumberOfBeads=50
comparam.autofidseed.autofidseed.TargetDensityOfBeads=50
comparam.autofidseed.autofidseed.ExcludeInsideAreas=0
comparam.autofidseed.autofidseed.AdjustSizes=1

#Alignment

comparam.align.tiltalign.SurfacesToAnalyze=1
comparam.align.tiltalign.LocalAlignments=0
comparam.align.tiltalign.MagOption=3
comparam.align.tiltalign.TiltOption=0
comparam.align.tiltalign.RotOption=-1
comparam.align.tiltalign.BeamTiltOption=0
comparam.align.tiltalign.ResidualReportCriterion=3

#Tomogram Positioning

runtime.Positioning.any.wholeTomogram=1
runtime.Positioning.any.binByFactor=1
runtime.Positioning.any.thickness=800

#Aligned Stack Parameters

runtime.AlignedStack.any.linearInterpolation=1
runtime.AlignedStack.any.binByFactor=1
comparam.ctfplotter.ctfplotter.InvertTiltAngles=0
comparam.ctfcorrection.ctfphaseflip.InvertTiltAngles=1
comparam.mtffilter.mtffilter.LowPassRadiusSigma=0.35 0.05
runtime.GoldErasing.any.binning=1
comparam.golderaser.ccderaser.ExpandCircleIterations=2

#Reconstruction

comparam.tilt.tilt.THICKNESS=400
comparam.tilt.tilt.RADIAL=0.35 0.05
runtime.Reconstruction.any.useSirt=0

#SIRT parameters

comparam.sirtsetup.sirtsetup.LeaveIterations=2 4 6
comparam.sirtsetup.sirtsetup.ScaleToInteger=-32000 32000
comparam.sirtsetup.sirtsetup.RadiusAndSigma=0.4 0.05
```

B Protocol to processing a dataset with batchruntomo on a cluster

1. Think of the amount of nodes, cores, and threads which have to be used for the dataset. If it is around 2.6GB in size; 1 node, 4 cores and 4 threads will do fine. Also configure *~/.bashrc* or equivalent to set the amount of threads to be used.

2. Fill in the template files and batch file accordingly to how *batchruntomo* should process the dataset.

3. Copy the dataset to a node on which it will be processed. Make sure the node has enough cores to use, so at least the amount of cores determined in step 1. Also check this for all nodes to be used: When a node has not a sufficient amount of cores, use the maximum amount of cores available instead.

4. Determine the locations on the host node to be used. That is: Make sure the folder in which the result has to be deposited exists and is not already filled with a result of the same dataset. Also make sure the templates are in the right directories as configured in the batch file. Lastly make sure there is enough space left on the node to complete the process.

5. Run *batchruntomo* with the right parameters: Specify the nodes and their amounts of cores correctly, the location from which to take and where to move (or copy) the dataset, and the directive file. Possibly also define to which e-mail address a notification should be sent when the process finishes.

6. Check the results. If they are insufficient, revert to step 2. If the process took too long at that, revert to step 1.