Universiteit Leiden
The Netherlands

Opleiding Informatica

Optimization of High-Throughput Zebrafish Imaging on a Distributed Computer

Loes Dekker

Supervisors:
Fons Verbeek & Kristian Rietveld

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

14/09/2019

Abstract

This bachelor thesis describes the construction of a pipeline that calculates the volume and surface area of a zebrafish larvae from its axial-view images from three already existing components. These components perform the segmentation of the axial-view images, the optimization of the initially guessed camera parameters and the creation of a 3d reconstruction of the zebrafish larvae. The results from the pipeline seem to be within expectation and are consistent. Different ways to optimize this pipeline were researched. The optimal format to save the profiles for the profile based 3D reconstruction was determined to be compressed numpy arrays. The segmentation of the axial-view images can be run on a multicore computer cluster instead of on GPU with a low estimated slowdown. The biggest bottleneck in the pipeline is the camera parameter optimization. To decrease runtime this step can best run multiple zebrafish larvae parallel. A correlation seems to exist between zebrafish larvae volume and total runtime though the significance of this correlation could not be proven.

Contents

1 Introduction

1.1 The Optimization of the Zebrafish 3D Reconstruction Pipeline

Zebrafish larvae are often used as a model organism in biology thanks to their transparency, fast development, the availability of the whole zebrafish genome and to zebrafish containing the majority of genes known to be associated with human diseases [P+15]. Using the Vertebrate Automated Screening Technology (VAST BioImager) it is possible to construct a 3D model and calculate the volume and surface area of a zebrafish larvae. The VAST BioImager is a microscope mounted system which contains a capillary that holds a zebrafish [G+17]. The capillary is rotated by a stepper motor, which enables the system to take axial-view images of the zebrafish larvae (see Chapter 2).
Reconstructing the 3D model, calculating the volume and calculating the surface area of the zebrafish larvae is done in three steps: the segmentation, the camera parameter optimization and the 3D reconstruction (see Chapter 3) [G+17]. The code which executes these three steps was already written but still needed to be composed in a single pipeline.

1.2 Research Question

The aim of this bachelor research project is to compose this pipeline and to researche how the generation of 3D models from VAST BioImager zebrafish images can be optimized in the different steps of the pipeline on the LIACS Life Science Cluster (LLSC). Several experiments will be conducted to determine what influences the total runtime of the pipeline. These experiments will be focused around a number of sub questions:

- Are there situations in which it is feasible to run the segmentation step on CPU instead of on GPU?
- What is the most efficient way to store the profiles of the zebrafish larvae images generated in the segmentation?
- How can the different components of the pipeline run most efficiently on a distributed computing environment?
- What are the bottlenecks in the pipeline?
- Does the developmental stage of the zebrafish larvae influence the total runtime of the pipeline?

1.3 Thesis Overview

This bachelor thesis research was performed at The Leiden Institute of Advanced Computer Science (LIACS) and supervised by Prof.dr.ir. F.J. Verbeek and Dr. K. F. D. Rietveld. Chapter 2 provides an overview of the different materials used in this research. Chapter 3 gives an in depth explanation of the different components of the pipeline and the modifications necessary to create the pipeline from these components. Chapter 4 contains all the conducted experiments on the (components of the) pipeline. In this chapter the efficiency of different options are explored. In Chapter 5 one can find the conclusions of the research questions, the discussion on the conducted research and the suggestions for further research.

2 Materials and Methods

2.1 The Zebrafish

Zebrafish (Danio rerio) is a freshwater fish native to South Asia [FB]. The zebrafish is an important model for vertebrate development, genomics, physiology, behavior, toxicology, and disease [MP15]. Zebrafish are easy to breed in large quantities and the early embryonic stages are well suited for study through microscopy [G+17]. The transparency of the zebrafish larvae also enables the use of fluorescence marking.

2.2 The VAST BioImager

The VAST BioImager is a microscope mounted system which enables taking high resolution axial-view images of a zebrafish larvae [G+17]. In this setup, the zebrafish larvae is contained in a capillary that is rotated by a stepper motor (see Figure 1). The size step of the stepper motor can be determined by the user, meaning that a range of axial-view images of the zebrafish larvae can be captured in both 84 and 100 images for example. The VAST BioImager is capable of both bright field microscopy and fluorescent microscopy.

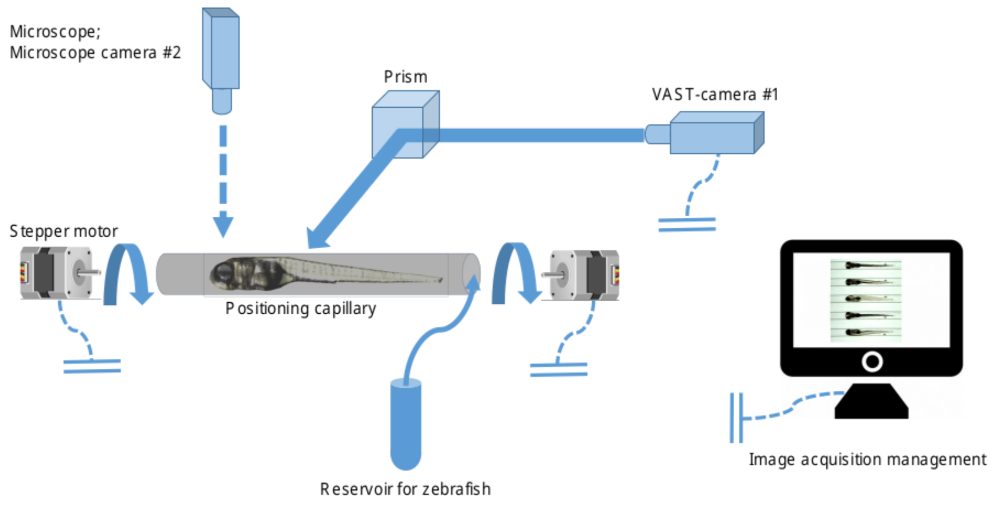

Figure 1: Setup of the VAST BioImager [G+17]

2.3 Compute Infrastructure

2.3.1 The LLSC Environment

The Leiden Life Science Cluster consists of a file server with a capacity of 36 TB, ssh nodes and 20 multicore compute nodes. There are several type of compute nodes available: dual-core Intel Xeon 5150 CPUs at 2.66 GHz, quad-core Intel Xeon E5430 CPUs at 2.66GHz and quad-core Intel Xeon X5450 CPUs at 3.00 GHZ, all with 16 GB RAM. At the start of this research there was no GPU available within the LLSC. The cluster has a Linux operating system installed.
The software installed at the LLSC environment is outdated and is not able to run the scripts in

the pipeline, in particular the segmentation step due to its dependence on Tensorflow. The LLSC will be updated this year to a more recent Linux distribution.

2.3.2 LLSC2.0

Since the LLSC is currently not capable of running the pipeline, a test environment was set up by Kristian Rietveld. This test environment contains two compute nodes, an Intel Xeon 5150 and an Intel Xeon E5430, CPUs that are also in the LLSC.
The software on the LLSC2.0 is up to date. When the LLSC is updated, it will have the same software as the LLSC2.0.

2.3.3 GPU Workstation

LIACS has a GPU workstation (in the department referred to as Puffskein) available that contains two high-end GPUs, namely 2 Titan X GPUs and an Intel(R) Core(TM) i7-6950X at 3.00GHz. The GPUs in this machine are going to be placed in the LLSC environment in the near future and is therefore a suitable testbench for components requiring GPUs that will be deployed on LLSC in the future. A Linux operating system is installed on this GPU workstation.

2.4 Datasets

The LLSC has different datasets containing zebrafish larvae image datasets available for testing. For the experiments conducted in this bachelor thesis zebrafish larvae image datasets from three datasets were used (see Table 1). The raw_data dataset was used to train the classification for the segmentation (see Chapter 3). The wasabi1610 dataset is a result from an experiment on wasting, where starved larvae consume themselves and grow slower as a result.

name	zebrafish image datasets	images/dataset	ages	classification
raw_data	60	84	3dpf to 5dpf	yes
raw_new_data	123	100	3dpf to 5dpf	no
wasabi1610	10	100	unknown (starved)	no

Table 1: The datasets used in this bachelor research project

2.5 The Slurm Workload Manager

Slurm (formerly known as Simple Linux Utility for Resource Management or SLURM) is an open source cluster management and job scheduling tool for Linux systems [slu]. Slurm is installed on LLSC2.0 and will be installed on the LLSC environment when it is updated. The version installed on the LLSC2.0 is slurm 16.05.9.
In this bachelor thesis slurm will be used to conduct several experiments on the cluster computer. The final pipeline will be constructed as a Slurm jobscript.

2.6 The Pipeline

Three scripts have been written that each execute a part of the 3D reconstruction. In this bachelor thesis, the scripts will be referred to as the segmentation step, the camera parameter optimization step and the 3D reconstruction step (see Figure 2). The script for the segmentation step was written by Wilco de Boer [dB19] in `python3` and is an extension of earlier work on the segmentation of zebrafish larvae by Wilco Verhoef [Ver18]. The other two scripts were originally written by Yuanhao Guo [G$^+$17] in `matlab` and were translated by Juliëtte Meeuwsen to `python2` [Mee17]. Kristian Rietveld ported the `python2` code to `python3` code.

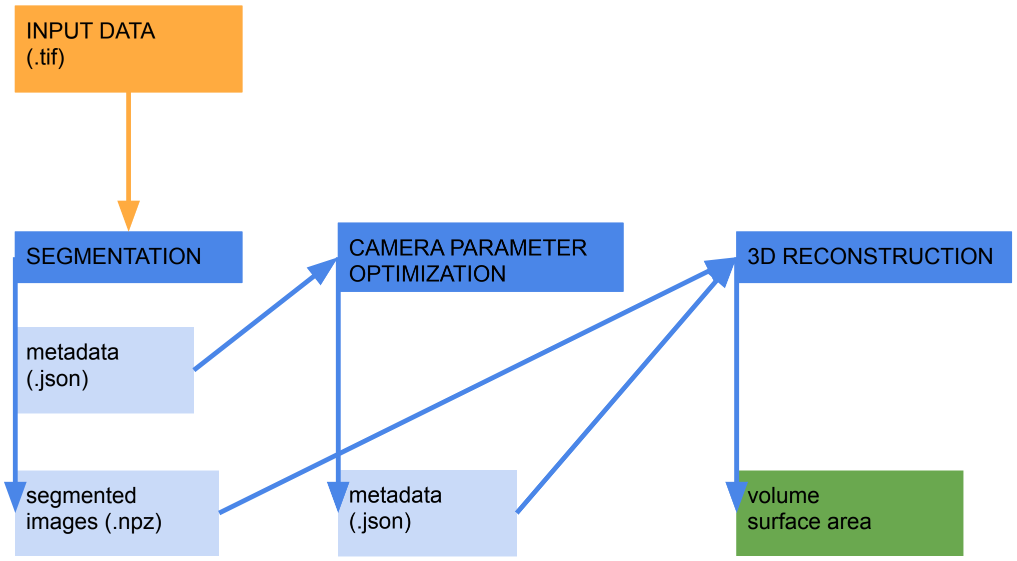

Figure 2: Flowchart of the pipeline construction

3 Implementation

3.1 Segmentation

In the segmentation step of the pipeline, the profile of each axial-view zebrafish larvae image in the given data is calculated. This step is visualized in Figure 3(B). The script takes n zebrafish larvae images in `tif` format as input and delivers the n segmented images (the format of the output is `.png`).
The script uses a convolutional neural network that was trained to predict the edges of the zebrafish larvae. The `python3` package `keras` is utilized for generating the segmented images with a model. This model was made with the segmented images from three different available datasets. These datasets include raw_data (see Chapter 2) in addition to the datasets raw_new_data_good and raw_new_data_manual [dB19]. The first two of these datasets were segmented using the `matlab` code, the third dataset was traced manually.

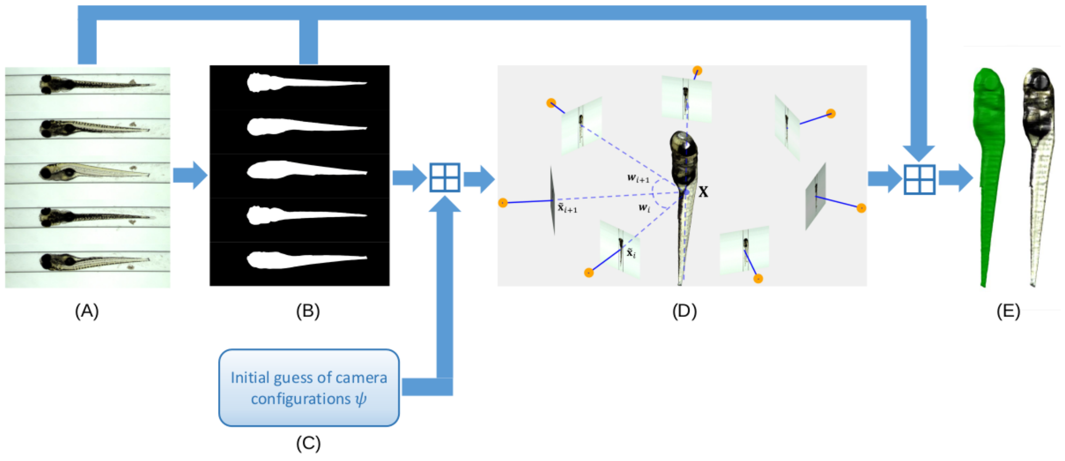

Figure 3: Different stages in the zebrafish larvae 3D reconstruction [G+17]

3.2 Camera Parameter Optimization

The camera parameter optimization step formulates a transformation matrix for every image of the zebrafish larvae in the dataset. This matrix translates every point in the 2D segmented image to a point in 3D space. Every transformation matrix in a single dataset is calculated with the same intrinsic camera matrix. This intrinsic camera matrix is constructed using the focal length of the camera (f), scaling factors, the image center and the skew factor of the camera. Mapping a point in 3D space with coordinates X, Y and Z to a point in 2D with coordinates x and y is done using the formulas $x = (fX)/Z$ and $y = (fY)/Z$ [G+17]. This way the initial guess of the camera configurations are optimized. The optimized parameters are returned in a JSON file.

3.3 3D Reconstruction

The 3D reconstruction and the volume and surface area are generated using the segmented images and the corresponding transformation matrices once the camera parameters are known. It thus uses the JSON file generated in the camera parameter optimization. In the current implementation one fourth of the segmented images are used to create the 3D reconstruction. The 3D representation is generated by back-projecting the segmented images to the 3D world frame so as to obtain the intersections [G+17]. The corresponding 3D representation is built from the collection of voxels visible from the view of every segmented image (see Figure 3(D)). This method was based on the space-carving theorem.
The 3D voxel object is rendered by the marching cubes algorithm so that its surface is made up out of polygons. After this step, the irregularities of the surface are smoothed out using the package `PyPoisson` [Mee17]. The 3D voxel object is used to calculate the volume and surface area.

3.4 Constructing the Pipeline

In order to construct a pipeline from these three components, a number of modifications to the existing components were required.

3.4.1 Modifications to Segmentation

The segmentation script was modified so that it can save the segmented images to disk as numpy arrays and compressed numpy arrays (`.npy` and `.npz`). It was also modified to create a JSON file containing certain meta data. This meta data includes the data type of the segmented images and the path to the segmented images. An option was added to force the script to run on a limited amount of CPU cores for experimentation purposes.

3.4.2 Modifications to Camera Parameter Optimization and 3D Reconstruction

The camera parameter optimization script was modified to not create a JSON file, but use the already existing JSON file created by the segmentation step. A numpy array dataloader and a compressed numpy array dataloader were implemented which are used in both the camera parameter optimization and the 3D reconstruction. This allows these subsequent steps to read the segmented images stored by the segmentation step. The existing JSON dataloader was modified to make it compatible with the `.npy` and `.npz` data types.

3.4.3 Slurm Jobscripts

The final version of the pipeline is a bash script that runs three slurm jobscripts (see Figure 4). There is a jobscript for each step in the pipeline. The latter two jobscripts are submitted immediately after their respective predecessor is submitted to the job queue of slurm. The input for the final bash script is a path to a folder containing the zebrafish larvae that need to be processed. This input is passed on to the three jobscripts. The latter two jobscripts also recieve the job-id of the predecessing job in the pipeline so that job dependencies are created (job dependencies will be explained below). At the start of the pipeline, the correct python environment is activated, which is deactivated at the end of the pipeline. Each jobscript produces log files containing dates, output and possibly errors. The log files of the last jobscript, which handles the 3D reconstruction, contain the calculated volume and surface area.

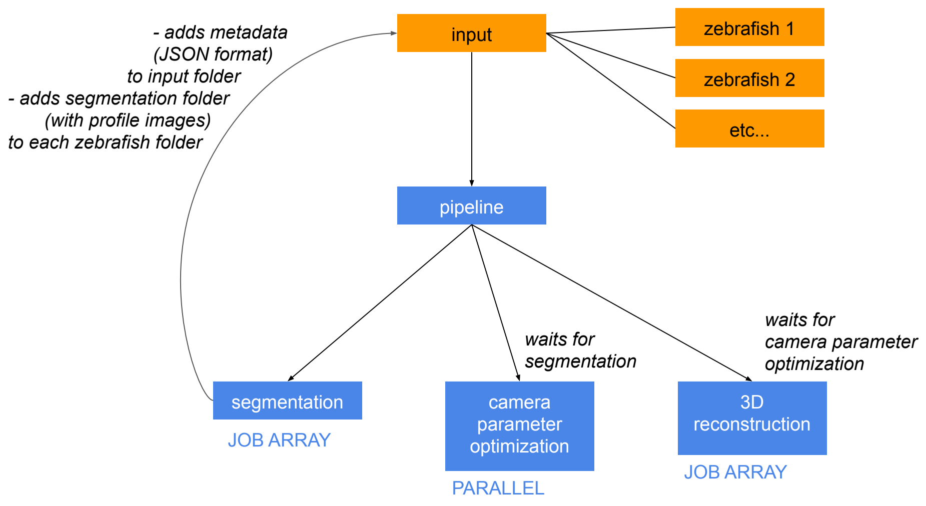
- adds metadata (JSON format) to input folder
- adds segmentation folder (with profile images) to each zebrafish folder
input
zebrafish 1
zebrafish 2
etc...
pipeline
waits for segmentation
waits for camera parameter optimization
segmentation
JOB ARRAY
camera parameter optimization
PARALLEL
3D reconstruction
JOB ARRAY

Figure 4: Overview of the bash script calling the slurm jobscripts

Job Dependencies

The 3D reconstruction step of a zebrafish larvae image set can only start when both the segmentation step and the camera parameter optimization have finished. Job dependencies were used to ensure that each step in the pipeline starts when the former has finished without errors. Job dependencies are easily implemented using slurm. This is done by passing on the job-id of the first job to the dependent job as a parameter in the slurm command that submits the job to the job queue (see a simplified example in Figure 5). In the example code in Figure 5, sbatch is the command that submits a job to the slurm job queue. As can be seen in line 5, the job-id of the segmentation step is saved in the variable `jid1`. The camera parameter optimization is dependent on this job-id finishing without errors (line 6). The 3D reconstruction is dependent on the camera parameter optimization step in the exact same way (line 7).

```
1    #!/bin/bash
2
3    input=$1
4
5    $jid1=$(sbatch segmentation.job $input)$
6    $jid2=$(sbatch --dependency=afterok:$jid1 cameraparam.job $input)
7    $jid3=$(sbatch --dependency=afterok:$jid2 reconstruction.job $input)
```

Figure 5: Simplified batch script showing how the slurm job dependencies are implemented

Parallel Jobs

The camera parameter optimization step is not multithreaded. This means that when multiple camera parameter optimization processes need to be run, they are optimally run parallel. In the slurm jobscript that handles the camera parameter optimization this is handled with GNU Parallel [par]. The input for this jobscript is the folder containing all zebrafish larvae to process. The camera parameter optimization for each zebrafish larvae is then submitted to GNU Parallel (see the example code in Figure 6). In the example code in Figure 6, the srun is the command used to run executables in slurm. Line 5 shows how GNU parallel executes several camera parameter optimization scripts with different zebrafish larvae json files.

```
1    #!/bin/bash
2
3   CAM="path/to/camera/parameter/optimization/script"
4
5   parallel "srun $CAM -j " ::: $set_of_all_zebrafish_json_files_to_process
```

Figure 6: Simplified example of how multiple camera parameter optimization processes are run parallel

Job Arrays

The segmentation step and the 3D reconstruction step are both multithreaded and thus multiple processes should not run parallel. One wants to be able to submit a number of zebrafish larvae as one segmentation job to ensure that the job dependencies stay intact, since the camera parameter optimization step is, thanks to its parallel nature, a single job. The multithreaded segmentation (and 3D reconstruction) jobs for different zebrafish larvae are thus submitted as a single job array. This array contains the jobs for all the zebrafish larvae (see the example code in Figure 7). On line 5 of Figure 7 one can see that the sbatch command is given an array parameter. This array parameter determines that instead of one job, an array of jobs should be submitted to the slurm queue. Within segmentation.job the array task id is used to determine which zebrafish larvae is to be processed.

```
1    #!/bin/bash
2
3   input=$1
4
5   jid1=$(sbatch --array=1-$number_of_zebrafish --parsable segmentation.job $input)
```

Figure 7: Example of how job arrays are implemented in the pipeline

4 Experiments

In this chapter the experiments conducted on (steps of) the pipeline are described. These experiments include the validation of the results, the feasability of running the segmentation step on CPU, the most efficient datatype for the output of the segmentation step, how many parallel processes of the camera parameter optimization are optimal and the possible correlation between zebrafish volume and total runtime.

4.1 Assembling the Pipeline

After implementing the minor changes necessary to turn the three programs into one pipeline (see Chapter 3), two experiments were conducted to test whether the pipeline functions as expected. These experiments were run on the GPU workstation (see Chapter 2). Firstly the volume and surface area for nine different zebrafish larvae image datasets were generated. Secondly three of these datasets were run multiple times to check the consistency of results in between datasets.

Experiment and Results

Ten different zebrafish larvae image datasets have been used (see Chapter 2). This ensures that images that have been used to train the classifier and images that have not been used to train the classifier are tested. From the raw_data and raw_new_data where the age of the zebrafish larvae are known all three age groups were tested. The results can be seen in Table 2.

dataset	**id**	**amount of images**	**age (days)**	**volume (μm^3)**	**surface (μm^2)**
raw_data	3dpf_s003	84	3	$2.498 * 10^8$	$3.395 * 10^6$
raw_data	4dpf_s030	84	4	$2.654 * 10^8$	$3.521 * 10^6$
raw_data	5dpf_s022	84	5	$2.992 * 10^8$	$3.729 * 10^6$
raw_new_data	3dpf_s104	100	3	$2.383 * 10^8$	$3.107 * 10^6$
raw_new_data	4dpf_s111	100	4	$2.314 * 10^8$	$3.173 * 10^6$
raw_new_data	5dpf_s110	100	5	$1.918 * 10^8$	$2.461 * 10^6$
raw_new_data	5dpf_s113	100	5	$2.430 * 10^8$	$3.126 * 10^6$
wasabi1610	2_1	100	unknown	$9.517 * 10^7$	$1.403 * 10^6$
wasabi1610	4_1	100	unknown	$1.264 * 10^8$	$1.545 * 10^6$
wasabi1610	9_1	100	unknown	$1.439 * 10^8$	$2.124 * 10^6$

Table 2: Different datasets of zebrafish larvae image data and the results generated by the pipeline

The pipeline is supposed to yield the same volume and surface for the same zebrafish larvae dataset. Therefore one zebrafish larvea image set was chosen from raw_data, raw_new_data and wasabi1610 and run three additional times. For this experiment the results were not rounded to three decimals, but were left precisely the way the pipeline returned them, so that the slightest deviation in results can be detected, as can be seen in Table 3.

id	run	volume (μm^3)	surface (μm^2)
4dpf_s030	1	$2.65395723 * 10^8$	$3.5205945 * 10^6$
4dpf_s030	2	$2.65395723 * 10^8$	$3.5205950 * 10^6$
4dpf_s030	3	$2.65395723 * 10^8$	$3.52059475 * 10^6$
4dpf_s030	4	$2.65395723 * 10^8$	$3.52059425 * 10^6$
5dpf_s113	1	$2.43041719 * 10^8$	$3.1264265 * 10^6$
5dpf_s113	2	$2.43041719 * 10^8$	$3.12642525 * 10^6$
5dpf_s113	3	$2.43041719 * 10^8$	$3.1264255 * 10^6$
5dpf_s113	4	$2.43041719 * 10^8$	$3.1264255 * 10^6$
2_1	1	$9.517135366412354 * 10^7$	$1.4030285 * 10^6$
2_1	2	$9.517135366412354 * 10^7$	$1.403028375 * 10^6$
2_1	3	$9.517135366412354 * 10^7$	$1.4030285 * 10^6$
2_1	4	$9.517135366412354 * 10^7$	$1.4030285 * 10^6$

Table 3: Different datasets of zebrafish larvae image data and the results generated by the pipeline

4.1.1 LLSC2.0

A subset of the first experiment was repeated on the LLSC2.0 (see Chapter 2) to ensure that the pipeline gives the same results on different systems. For this second experiment all the previously used zebrafish larvae image datasets from raw_data and three of the previously used zebrafish larvae image datasets from raw_new_data were used. The results can be seen in Table 4.

id	amount of images	age (days)	volume (μm^3)	δ ($*10^5$)	surface (μm^2)	δ ($*10^3$)
3dpf_s003	84	3	$2.499 * 10^8$	1	$3.395 * 10^6$	0
4dpf_s030	84	4	$2.654 * 10^8$	0	$3.521 * 10^6$	0
5dpf_s022	84	5	$2.992 * 10^8$	0	$3.729 * 10^6$	0
3dpf_s104	100	3	$2.383 * 10^8$	0	$3.107 * 10^6$	0
4dpf_s111	100	4	$2.314 * 10^8$	0	$3.172 * 10^6$	1
5dpf_s113	100	5	$2.431 * 10^8$	1	$3.126 * 10^6$	0

Table 4: Different datasets of zebrafish larvae image data and the results generated by the pipeline in the LLSC2.0 environment with the difference (δ) for volume and surface compared to the results generated in Table 2

Validation Conclusion

To validate the results from the first experiment, Figure 8 is used, where the expected volume and surface of three, four and five day old zebrafish larvae is shown.
The three zebrafish larvae image datasets from raw_data are all within or very close to the expected results for both the volume and the surface.
The three and four day old zebrafish larvae from raw_new_data are also within or very close to the expected results for both the volume and the surface. The five day old zebrafish larvae are both smaller than they are supposed to be, but both their tails are cut of partly from the picture. This might explain the decreased volume and surface.

The results from the wasabi1610 dataset cannot be validated with Figure 8, since the age of the zebrafish larvae are unknown. The observed results are all however significantly smaller than the expected results for the youngest zebrafish larvae. The zebrafish larvae in this dataset are however starved, meaning that they do not have the resources to grow the way a normal zebrafish would. This means that these zebrafish larvae being smaller than normal zebrafish larvae is to be expected.

The volume calculated in the second experiment is always constant. The surface area does vary slightly in the different runs. This is unexpected, however the biggest difference that occurred in this experiment was 1.0 μm^2, which is insignificantly small. When run on the LLSC2.0 environment, the results are mostly the same as on the GPU workstation. Three results were found that deviated from the results found on the GPU workstation, but only by $1 * 10^5 \mu^3$ (volume) and $1 * 10^3 \mu^2$ (surface area).
From this experiment can be concluded that the pipeline works as expected, though some results are just outside the margin given by Figure 8. Zf_3dpf_s104 and zf_4dpf_s111 were both slightly below the expected volume margin and zf_4dpf_s030 was slightly above the expected surface area margin. The ages of the zebrafish larvae are however not very precise, which probably caused this deviation from the set margins.
The calculated results are consistent when running the pipeline with the same dataset multiple times and are the same on the GPU workstation and the LLSC2.0.

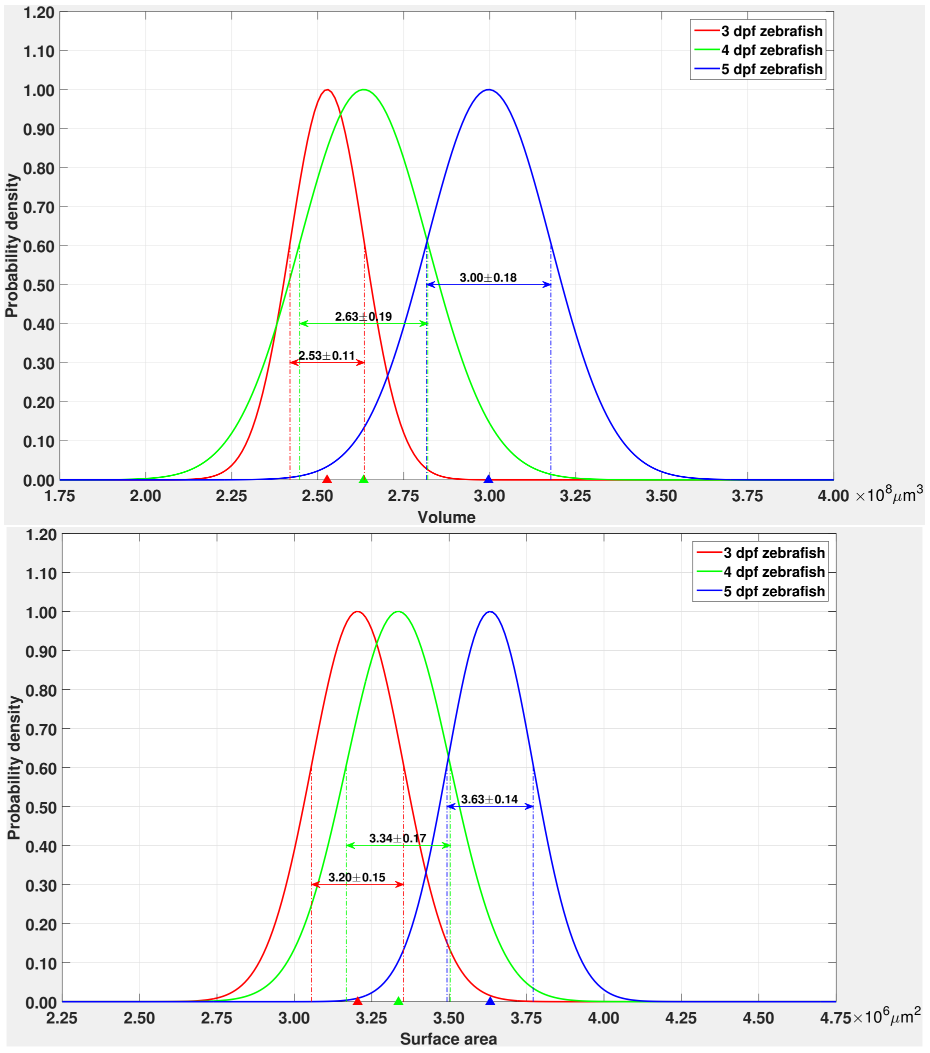

Figure 8: Expected volume and surface of zebrafish larvae aged three to five days [G+17]

4.2 Segmentation: GPU vs CPU

The classifier used in the segmentation step of the pipeline is trained on a machine with GPU, however the segmentation of the images itself does not necessarily need a GPU. The feasibility of running the segmentation on CPU was tested since the GPUs, whether they are included in the LLSC or not, might not always be available because the LLSC is used for many projects beside this pipeline.

4.2.1 Feasibility

In order to test whether it is feasible to consider the use of CPUs for generating segmented images, several tests were run on the GPU workstation (see Chapter 2) that has both a powerful enough GPU to run the segmentation program.
The results from this small experiment will determine whether further more elaborate experiments

will be conducted in LLSC2.0 (see Chapter 2) comparable to the computer cluster (LLSC) where the system will eventually run.

Experiments and Results

For this experiment three zebrafish larvae image datasets were used, one from raw_data, one from raw_new_data and one from wasabi1610. These sets were chosen so that both data that was used to train the classifier and data that has not been used to train the data was represented. Different aged zebrafish larvae were also selected from each dataset, as can be seen in Table 5.

dataset	**GPU**	**CPU multicore**	**CPU single core**
zf_3dpf_s003	0.049	1.251	6.697
zf_4dpf_s136	0.049	1.277	6.689
2_1	0.049	1.281	6.692

Table 5: Average time (s) it takes to segmentate single image without the first image of the dataset

The average time it takes to segment a single image on GPU and different CPU configurations was calculated (see Table 6), where the time it takes to segment the first image was not taken into account, since it takes considerably longer than the rest of the images. These results were calculated using the same three datasets from the experiment in Table 5.

hardware option	**single image**	**stdev**	**first image**
GPU	0.049	0.000	2.128
CPU multicore	1.270	0.025	1.678
CPU single core	6.693	0.032	7.183

Table 6: Average time (s) it takes to segment a single image, the first image and the average standard deviation for the results of Table 5

Based on these results, an estimate was calculated for the average time needed to run a dataset consisting of 100 images on the LLSC (see Table 7).

	GPU	CPU cluster with 20 cores
time (s) 100 image dataset	6.979	33.490 (estimation)
slowdown	1.0x	4.8x

Table 7: Extimate of the expected runtime of a 100 image dataset on a cluster with 20 cores

Feasibility Conclusion

From this experiment it has become clear that it is significantly slower to run the segmentation on CPU than it is to run it on GPU. However, the expected runtime for a dataset of 100 images is still relatively low, especially when keeping in mind that the camera parameter optimization takes about half an hour. Therefore performing segmentation on the CPU is certainly feasible and further tests on the LLSC test environment will be conducted.

4.2.2 LLSC2.0

On the LLSC2.0 (see Chapter 2) the time it takes to run the segmentation on the same CPUs as in the actual LLSC needs to be measured. In this test environment the possibility of distributed computing for the segmentation will also be explored.

Experiments and Results

For this experiment the average time per image was measured on both available compute nodes using the same datasets as in the first experiment of Section 4.2.1. Again, the segmentation of the first image was measured separately since this image takes significantly longer than the rest. The results can be seen in Table 8.

	single image	**stdev**	**first image**
compute node 1 (multicore)	9.561	0.004	10.122
compute node 2 (multicore)	5.990	0.009	5.487

Table 8: Average time (s) it takes to segmentate single image without the first image of the dataset

The segmentation step is a multithreaded script. In a second experiment a single zebrafish larvae image set from raw_data (3dpf) was segmented three times on both LLSC2.0 nodes using different amount of cores (as can be seen in Figure 9). The default bar in this figure is the maximum amount of threads, which is selected when no amount of threads is determined by the user.

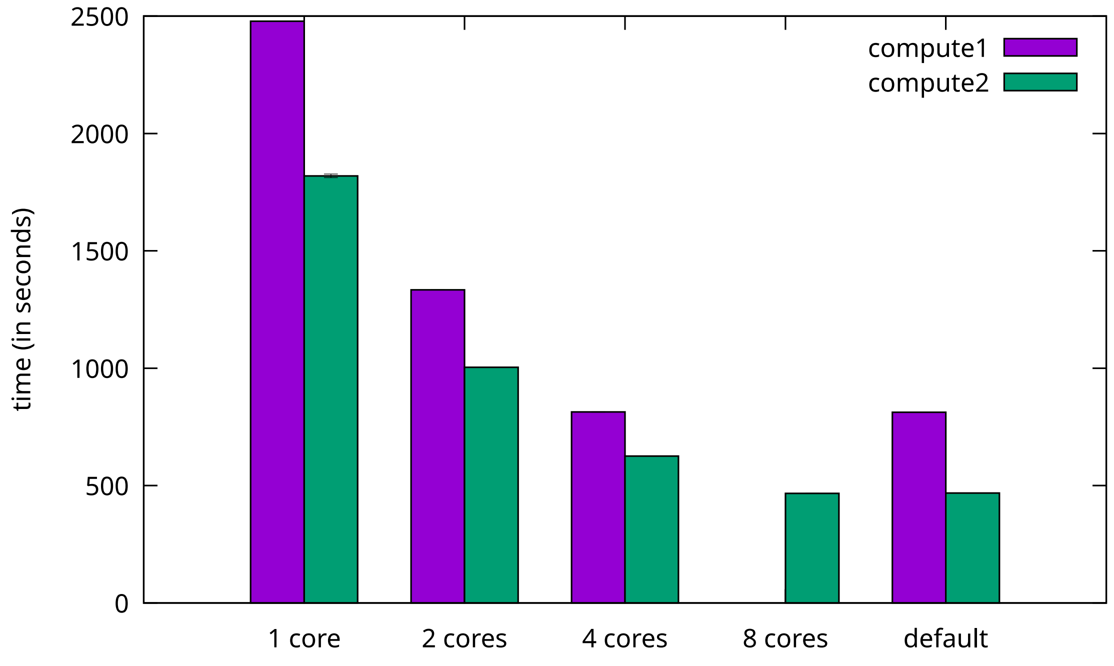

Figure 9: Average time (s) it takes to segmentate zf_3dpf_s003 on different cores on compute node 1 and compute node 2

The speedup of using multiple threads compared to using one thread was calculated for both compute nodes, as can be seen in Table 9 and Table 10.

compute node 1	**1 core**	**2 cores**	**4 cores**	**default**
average time (s)	2478.23	1333.83	813.62	812.67
speedup	1.0x	1.9x	3.0x	3.0x

Table 9: Speedup when using more threads when segmenting zf_3dpf_s003 on compute node 1

Zf_3dpf_s003 was also segmented three times on two 10-core Intel Xeon Silver 4114 CPUs at 2.20 GHz (see Table 11), available at the LLSC (see Chapter 2).

compute node 2	**1 core**	**2 cores**	**4 cores**	**8 cores**	**default**
average time (s)	1819.34	1003.99	625.95	466.54	467.93
speedup	1.0x	1.8x	2.9x	3.9x	3.9x

Table 10: Speedup when using more threads when segmenting zf_3dpf_s003 on compute node 2

	compute node 1	**compute node 2**	**Intel Xeon 4114**
average time (s)	812.67	467.93	22.27
stdev (s)	1.29	0.70	2.28

Table 11: Average time (s) it takes to segmentate zf_3dpf_s003 on different CPUs

Based on these results, an estimation was made on how long the segmentation of a 84 image dataset takes on a cluster computer consisting of 20 x compute node 1 and on a cluster computer consisting of 20 x compute node 2 (see Table 12). The results are compared to the time it takes to run this dataset on GPU.

	GPU	**node 1**	**node 2**	**node 1 x 20**	**node 2 x 20**
average or estimated time (s)	6.29	812.67	467.93	40.63	23.40
slowdown	1.0x	129.2x	74.4x	6.5x	3.7x

Table 12: Average time (s) it takes to segmentate an 84 image dataset

Conclusion CPU vs GPU

The segmentation step of the pipeline was designed on run on GPU, though it can run on CPU. At the start of this bachelor research project, the LLSC did not have acces to GPUs, however it now has the GPUs from the GPU workstation (see 2). These GPUs might not always be available for the segmentation step since the LLSC is used for many purposes by many users. Therefore the feasibility of running the segmentation of the segmentation on CPU instead of on GPU was investigated.

Looking at the average time to perform the segmentation of a single zebrafish image it is very clear that running the segmentation step on CPU is much slower than on GPU (see Table 6 and Table 8).

The runtime of a total dataset varies enormously between different CPUs however, where CPUs that have more cores available are considerably faster (see Table 11). This is to be expected, since the segmentation step is multi-threaded, and means that when enough high quality (modern) CPUs are utilized to segment a number of zebrafish larvae, this might be more efficient than using a single GPU. As can be seen in Table 9 and Table 10, there is a significant speedup when more cores are made available. When comparing the single core performance of the two compute nodes and the single core performance from the CPU from the GPU workstation (see Table 6) one can see that not only the amount of available cores, but also the overall quality of the CPU makes a significant difference in speed. As one can see in Table 12, the expected slowdown when utilizing a complete cluster is not very significant. A more extensive estimation on using different amount of nodes in a computer cluster will be conducted in a dedicated section on distributed computing on the LLSC.

When the LLSC is set up completely, with up to date software and slurm (see Chapter 2) installed, the best way to handle the segmentation step would be as follows. The script will be run on GPU when either one of the GPUs is not used at that moment in time, this would be most efficient since the average time per zebrafish image is smallest when run on GPU. To enable this, either in the bash script, the slurm jobscript or the python3 script itself a check must be implemented that can see GPU usage. When both GPUs are being used, the script is run on CPU.

4.3 Data Types of the Segmented Images

The output of the segmentation step of the program is a `JSON` file containing meta data and a folder containing the segmented images. There are three potential ways in which these images can be saved: `png`, as a `numpy` array (file extention `.npy`) or as a compressed `numpy` array (file extention `.npz`). Saving the images in as small as possible files has two advantages. The first is storage space. For the final application of the pipeline there is no need to keep all the segmented images, however when further optimization will be conducted on the pipeline keeping the images for experimentation might be necessary. The second advantage is that it takes less time to send small files to different nodes in the distributed computer cluster.

Experiments and Results

This experiment was run on the GPU workstation (using GPU for the segmentation step).
In order to test the efficiency of each approach, the elapsed time for each step in the program was measured three times with three different datasets. These datasets are all different in the way that the zebra fish larvae are different ages. The first datasets was used to train the classifier, the other two were not used to train the classifier. The results can be seen in Table 14 and Figure 10.
The average size of a single image in different formats can be seen in Table 13. The average size was determined with the first 10 contour images of zf_3dpf_s003.

	average size	**stdev**
`.png`	3.4 kB	0.048
`.npy`	4.1 MB	0.000
`.npz`	2.0 kB	0.032

Table 13: Average size of a single image for each format

The calculated volume and surface area are consistent between data types.

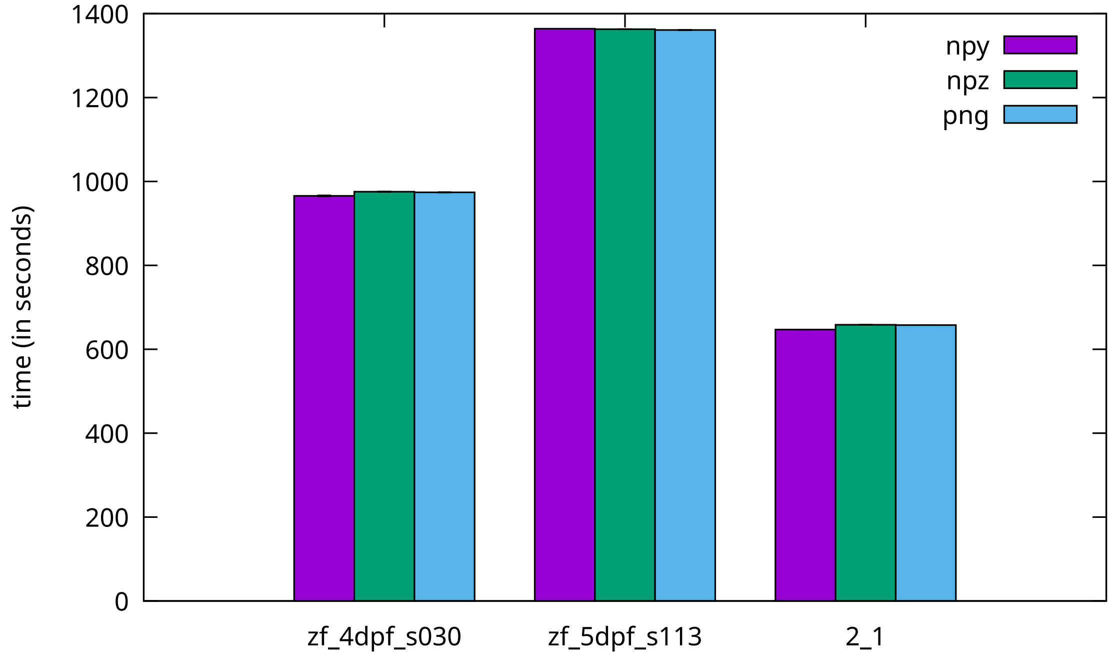

Figure 10: Average time (in seconds) for the whole pipeline

	`.npy`	stdev	`.npz`	stdev	`.png`	stdev
SEG zf_4dpf_s030	18.70	0.08	18.73	0.58	18.83	0.04
SEG zf_5dpf_s113	20.83	0.03	21. 35	0.16	21.30	0.08
SEG 2_1	20.85	0.06	21.30	0.09	20.83	0.11
CAM zf_4dpf_s030	936.30	3.58	946.40	3.46	944.64	1.30
CAM zf_5dpf_s113	1334.61	0.53	1332.84	1.64	1331.11	2.19
CAM 2_1	619.87	0.44	630.97	1.74	630.52	0.23
REC zf_4dpf_s030	10.48	0.79	10.53	0.71	10.68	0.90
REC zf_5dpf_s113	8.86	0.10	8.85	0.37	8.88	0.16
REC 2_1	6.26	0.10	6.20	0.03	6.30	0.03

Table 14: Average time and standard deviation (in seconds) for the segmentation (SEG), camera parameter optimization (CAM) and 3D reconstruction (REC) for the different supported file formats

LLSC2.0

This experiment was repeated on the second computer node of the LLSC2.0 (see Chapter 2). The results can be seen in Table 15 and Figure 11.

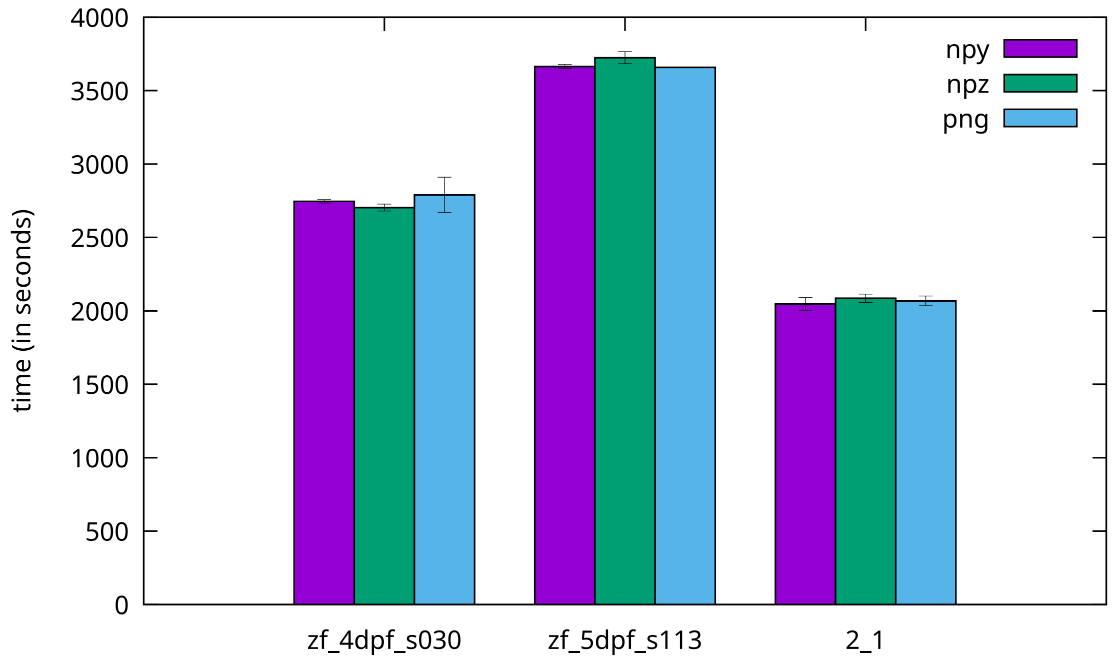

Figure 11: Average time (in seconds) for the whole pipeline on the LLSC2.0

	`.npy`	stdev	`.npz`	stdev	`.png`	stdev
SEG zf_4dpf_s030	471.48	1.84	466.45	0.28	455.61	0.55
SEG zf_5dpf_s113	559.18	0.29	554.81	0.54	540.44	0.31
SEG 2_1	558.57	0.52	553.447	1.35	540.60	0.31
CAM zf_4dpf_s030	2251.97	13.91	2214.54	36.20	2311.52	151.38
CAM zf_5dpf_s113	3084.96	15.08	3149.96	36.30	3097.96	2.46
CAM 2_1	1473.43	37.78	1517.14	24.44	1512.35	29.12
REC zf_4dpf_s030	22.97	0.08	22.48	0.13	22.91	0.46
REC zf_5dpf_s113	20.55	0.31	19.80	0.30	19.99	0.60
REC 2_1	15.25	0.09	15.16	0.05	14.87	0.62

Table 15: Average time and standard deviation (in seconds) for the segmentation (SEG), camera parameter optimization (CAM) and 3D reconstruction (REC) on the LLSC2.0 for the whole pipeline on the LLSC2.0

Conclusion Data Types

All data types yield the same volume and surface result (within a very small range of error). The smallest data type is the compressed numpy array (file extension `.npz`) which is only 2 kB's large. In Figure 10 one can see that the average time difference when running the entire pipeline on the GPU workstation is negligible. There do not seem to be significant differences in the time measurements separate parts of the pipeline either, see Table 14.
On the LLSC2.0 the segmentation step is run on CPU instead of on GPU, this does however not have an effect on the runtime for different datatypes, as can be seen in Table 15. In this case too, there are no significant differences and there is no clear overall best performing data format.
Saving the segmented images in an as small as possible way has two advantages. Firstly the files take less storage space and secondly it takes less time to send the files to different nodes in the distributed computing cluster. Therefore it is the most efficient to use the compressed numpy arrays.

4.4 Camera Parameter Optimization on LLSC2.0

Both the segmentation step and the reconstruction step are multithreaded. It is therefore optimal to give a single zebrafish larvea image set going through one of these steps exclusive use to one CPU node with multiple cores, even when there are several zebrafish larvae image sets to be processed. The camera parameter optimization step is not multithreaded however. When multiple zebrafish larvea image sets are to be processed, it is more efficient to run the camera parameter optimization on these sets in parallel. The optimal amount of sets to run in parallel needs to be determined, as running a lot of sets in parallel can also result in loss of efficiency due to CPUs lacking enough memory bandwith to efficiently running all these processes at the same time.

Experiments and Results

In this experiment the most efficient way to run the camera parameter optimization in parallel was investigated. This was done by running the camera parameter optimization eight times (using zf_3dpf_s003) on different amount of cores. This experiment was conducted on compute node 1 (see Table 16) and on compute node 2 (see Table 17). In these tables each core can run one process at once. When one core is used, there is no parallelism, when four cores are used, four processes run parallel.

	1 core	**2 cores**	**4 cores**
average time (s) per dataset	2085.75	2215.53	3473.84
stdev	28.23	18.73	17.24
total runtime	16686.45	8874.76	6968.83
speedup	1.0x	1.9x	2.4x

Table 16: Average and total time (s) for zf_3dpf_s003 eight times with different levels of parallelism on compute node 1

	1 core	**2 cores**	**4 cores**	**6 cores**	**8 cores**
average time (s) per dataset	1920.66	2093.64	2659.75	3313.87	4804.27
stdev	32.61	125.05	2.00	702.45	17.81
total runtime	15365.67	8410.30	5324.34	5815.31	4820.51
speedup	1.0x	1.8x	2.9x	2.6x	3.2x

Table 17: Average and total time (s) for zf_3dpf_s003 eight times with different levels of parallelism on compute node 2

Conclusion Camera Parameter Optimization

In this experiment it was investigated how many camera parameter optimization processes could optimally run parallel on both the available computer nodes in the LLSC2.0 (see Chapter 2). The average time it takes to optimize the camera parameters increases when the parallelism increases (see Tables 16 and 17), except when six parallel processes are run. The standard deviation is extremely large when six parallel processes are run, because the last two jobs took a significantly smaller amount of time to execute than the first six. This is to be expected, since the average time increases when parallelism increases. The first six processes thus take longer than the last two.
The total runtime keeps decreasing with an increase of parallelism however, both on compute node 1 and compute node 2. Every time the amount of parallel processes is doubled, this decrease in runtime is smaller though. From this we can conclude that it is most efficient to run different camera parameter optimizations in parallel on as many cores as the CPUs have. For the third type of CPU that is available in the LLSC and not in the LLSC2.0, this experiment might need to be repeated, since the speedup is not consistent inbetween both nodes.

4.5 Distributed Computing on LLSC

With the conducted experiments on the segmentation and camera parameter optimization, one can calculate the expected runtime for the pipeline on a multicore computer cluster for a large number of zebrafish larvae to process.
Before the total expected runtime can be calculated, some additional data on the 3D reconstruction is needed.

3D Reconstruction on LLSC2.0

To test the time it takes to execute the 3D reconstruction of a single zebrafish larvae on both compute nodes using all threads, zf_3dpf_s003 was run three times on both compute nodes (see Table 18).

	compute node 1	**compute node 2**
average time (s)	26.17	22.45
stdev (s)	0.09	0.11

Table 18: Average time (s) it takes to execute the 3D reconstruction of zf_3dpf_s003 on compute node 1 and compute node 2

Expectancy Tables

In this section two expectancy tables (see Table 19 and Table 20) will be constructed with the expected runtime for a job of 50 3 day old zebrafish larvae (based on results of zf_3dpf_s003) on a 16-node cluster. The expected total runtime is calculated for different amounts of available nodes. In the first table the cluster consists of nodes with four cores, in the second table the cluster consists of nodes with eight cores (based on compute node 1 and compute node 2).

nodes	segmentation	camera param	3D reconstruction	total runtime	speedup
1 node	40700	90597	1300	132597	1.0x
2 nodes	20325	48783	650	69758	1.9x
4 nodes	10569	27876	338	38783	3.4x
8 nodes	5691	13938	182	19811	6.7x
16 nodes	3253	6969	104	10325	12.8x

Table 19: Average time (s) it is expected to take to process 50 three day old zebrafish larvae on a computer cluster consisting of 16 x compute node 1

nodes	segmentation	camera param	3D reconstruction	total runtime	speedup
1 node	23350	33747	1100	58197	1.0x
2 nodes	11675	19284	550	31509	1.8x
4 nodes	6071	9642	286	15999	3.6x
8 nodes	3269	4821	154	8244	7.1x
16 nodes	1868	4821	88	6777	8.6x

Table 20: Average time (s) it is expected to take to process 50 three day old zebrafish larvae on a computer cluster consisting of 16 x compute node 2

Distributed Computing on LLSC Conclusion

To get an overview of distributed computing on LLSC two expectancy tables were constructed using the experiments conducted on the LLSC2.0. These expectancy tables (see Table 19 and Table 20) simulate the processing of 50 three day old zebrafish larvae on a computer cluster containing 16 nodes. Table 19 assumes this computer cluster contains 16 x compute node 1 and Table 20 assumes this computer cluster contains 16 x compute node 2.
Looking at the results, it is clear that the camera parameter optimization step is the biggest bottleneck in the system. The segmentation also takes a long time when compared to the 3D reconstruction step. There is a significant speedup when more nodes are utilized. The reason the last increase of nodes in Table 20 does not lead to a large increase in speedup is because the camera parameter optimization does not increase. The camera parameter optimization is run in eight parallel processes, meaning that 50 processes go equally fast on 8 and 16 nodes, since only 7 nodes are used. A speedup could be achieved if an equal distribution of parallel processes could be implemented. With a decrease of parallel threads, the individual time per dataset decreases too as was proven in an earlier experiment.

4.6 Time Measurements for Different Age Groups

Zebrafish larvae of different age and developmental stages have different volumes and surface areas. It is not implausible that smaller zebrafish larvae take less time to create a 3D reconstruction of than larger zebrafish larvae. If that is the case, older zebrafish larvae take more time to reconstruct than younger zebrafish larvae if these zebrafish larvae have grown up in the same circumstances.

Experiments and Results

Six zebrafish larvae image sets were selected from raw_data, two for each age group (three days, four days and five days). Two additional zebrafish larvae image sets were selected from wasabi1610, which contains images from mutant zebrafish larvae. From all available datasets, wasabi1610 contains the smallest zebrafish larvae. Each selected image set was processed by the pipeline twice, where the time was measured for each step in the pipeline. Per age group the averages were calculated for each step and for the total runtime of the pipeline. This experiment was run on the GPU workstation (see Chapter 2).

	mutant	**three days old**	**four days old**	**five days old**
Segmentation	21.22	19.17	19.18	19.21
Camera Parameter Optimization	722.47	813.05	883.075	1099.95
3D reconstruction	7.24	10.74	10.82	10.07
Total time	**750.92**	**843.00**	**913.07**	**1129.23**
Standard deviation total time	107.94	3.93	61.26	37.32

Table 21: Average time (in seconds) it takes to run the pipeline for different age groups

In Table 21 one can clearly see an increase in the total runtime with increasing developmental stage. This increase is not apparent in the segmentation step and the 3D reconstruction step, it is the camera parameter optimization step that noticeably increases with size.
In order to get a better understanding of the potential correlation between the volume and the total runtime, another nine additional datasets from raw_data were measured. The results can be seen in Figure 12. The wasabi1610 zebrafish larvae images are not included in this graph.

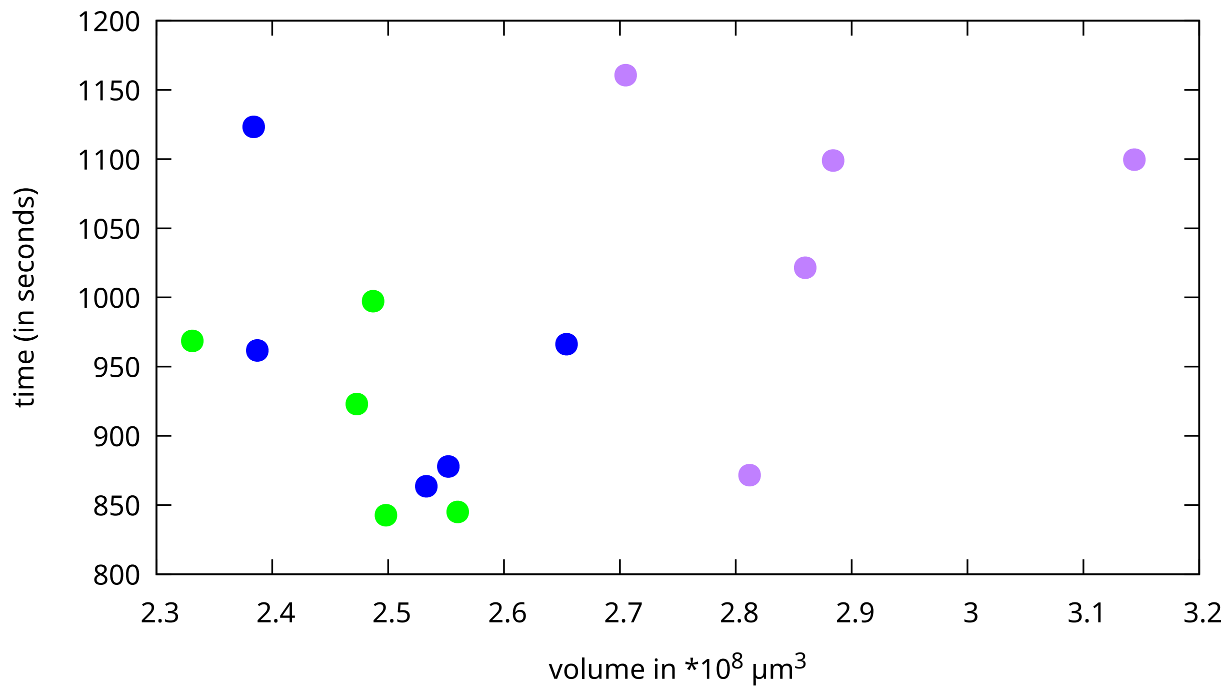

Figure 12: Fifteen zebrafish larvae of different volumes and the corresponding runtime (3dpf: green, 4pdf: blue, 5dpf: purple)

Different Age Groups Conclusion

In this experiment the possibility that runtime increases with volume is explored. In Table 21 one can see the average runtime of zebrafish larvae of different age groups, where older zebrafish have presumably larger volumes. This first experiment was conducted with eight different zebrafish larvae from raw_data and two underfed zebrafish larvae from wasabi1610, which were all run twice. From this experiment a notable increase in runtime was seen.
In Figure 12 one can see the relation between volume and runtime of the second conducted experiment, using only zebrafish larvae images from raw_data. The null-hypothesis is that there is no linear correlation between the volume and the runtime. The chosen p-value is 0.05. Using SPSS [sps] a significance of 0.194 was achieved. This exeeds the chosen p-value, which means that the null-hypothesis, which states that there is no linear correlation between the volume and the runtime, cannot be rejected.

5 Conclusions and Further Research

In this bachelor research project a pipeline was constructed and investigated from three already existing scripts based on research [G+17] which takes axial-view images from zebrafish larvae and calculates the volume and surface area of the zebrafish larvae. The axial-view images of the zebrafish larvae are generated by the Vetrebrate Automated Screening Technology (VAST BioImager). The pipeline consists of a segmentation step, a camera parameter optimization step and a 3D reconstruction step. Experiments were conducted with the pipeline to validate the obtained results, to identify the potential bottlenecks in the system, to test the stability of the runtime for different situations, such as zebrafish image volume and data type of segmented images, to test whether running one step of the program on CPU is a valid possibility and to research the possible gain of using distributed computing.

The results generated by the pipeline were compared to an indication of what size belong to what age groups of zebrafish [G+17]. For the images that were depicting the complete zebrafish larvae, the calculated volume and surface area were close to or within the expected range of the sizes belonging to their respective age groups. The pipeline can therefore be considered to function as expected.
The segmentation step generates a set of segmented images which are used in the 3D step of the pipeline. These images can be saved in different formats. Three potential formats, `.png`, `.npy` and `.npz` were evaluated to check if the writing and reading time is different for different formats. No difference in efficiency was found for the three formats. The format selection was therefore only based on size. `.npz` result in the smallest images.
The segmentation step can be run on both CPU and GPU. Running the segmentation step on GPU is significantly faster than running it on CPU. The runtime of the segmentation greatly increases when run on a CPU with more cores however since it is multithreaded. When 20 multi-core compute nodes are used, the slowdown is approximately between 6.5x and 3.7x. Therefore running segmentation on CPU is a viable option for when the GPU is not available.
The camera parameter optimization is the only step in the pipeline that is not multithreaded. The efficiency of running multiple processes parallel was tested for both the compute nodes available in the LLSC2.0. On both nodes, running highest amount of possible processes parallel was most efficient, however the average time per datasets increases for every time the amount of parallel processes was doubled. The speedup gained by adding more parallel processes was not consistent between both nodes, therefore if the camera parameter optimization is to be run on different CPUs this experiment must be repeated to determine what amount of processes is most efficient.
An estimate of running the pipeline on the LLSC was calculated. From this estimate it became clear that running the pipeline on as many nodes as possible with as many threads as possible is most efficient. It also became clear that the camera parameter optimization takes the longest time and that the 3D reconstruction is by far the shortest process.
The hypothesis that total runtime increases with zebrafish volume was tested. The runtime of the part of the pipeline that seemed to increase with volume was the camera parameter optimalization. After running fifteen datasets of various volumes no significant correlation was found however.

Further research

The largest bottleneck in the system is the camera parameter optimization step. The camera parameter optimization step is the only step in the pipeline that is not yet multithreaded. Investigating a different optimization method, that is able to run multithreaded, is a necessary next step in order to further improve the efficiency of the pipeline.
In order to finalize the pipeline, a system needs to be designed to automatically send the axial-view images from the VAST BioImager to the LLSC. This can be done with a webserver running at the LLSC, which, after storing the axial-view images to the LLSC, can submit a sequence of dependent jobs to run the pipeline with these images to slurm, the job scheduler.
If of interest, the experiment concerning the possible correlation between volume and runtime could be redone with a larger sample set. The sample size used in the experiment in this bachelor research project is relatively small.
When the pipeline is implemented on the LLSC, the experiment conducted on running the camera parameter optimization step parallel needs to be repeated on the CPU type that was not yet tested. The results from that experiment might possibly vary depending on the CPU.

Acknowledgements

I would like to thank Leon Helwerda for taking the time to help me with the GPU workstation and Wilco the Boer for helping me understand the segmentation step.

References

[dB19] Wilco de Boer. Evaluation and improvement of methods and models for segmentation of zebrafish larvae. July 2019.

[FB] Rainer Froese and Crispina B Binohlan. Danio rerio. http://www.fishbase.org/summary/Danio-rerio.html.

[G$^+$17] Yuanhao Guo et al. Three-dimensional reconstruction and measurements of zebrafish larvae from high-throughput axial-view *in vivo* imaging. *Biomedical Optics Express*, 8(5), June 2017. https://doi.org/10.1364/BOE.8.002611.

[Mee17] Juliëtte Meeuwsen. Design and implementation of 3d reconstruction from axial views on the leiden life sciences cluster. August 2017.

[MP15] Breadan M. McCluskey and John H. Postlethwait. Phylogeny of zebrafish, a “model species,” within *Danio*, a “model genus”. *Molecular Biology and Evolution*, 32(3), November 2015. https://doi.org/10.1093/molbev/msu325.

[P$^+$15] Emilie Plantié et al. Model organisms in the fight agains muscular dystrophy: Lessons from *Drosophila* and zebrafish. *Molecules*, 20(4), April 2015. https://doi.org/10.3390/molecules20046237.

[par] https://www.gnu.org/software/parallel/. Date accessed: 27 August 2019.

[slu] https://slurm.schedmd.com/documentation.html. Date accessed: 5 July 2019.

[sps] https://www.ibm.com/nl-en/products/spss-statistics. Date accessed: 2 July 2019.

[Ver18] Wilco Verhoef. On automatic segmentation and classification on age groups of zebrafish larvae microscopy images by deep neural networks. Master’s thesis, Leiden University, September 2018.