SF2568: Project Report
Michael Hanke (*hanke@nada.kth.se*)

Omar Elshenawy (*omares@kth.se*)
Thomai Stathopoulou (*thomai@kth.se*)

May 20, 2016

1 Problem Definition

With the digitization of many medical devices, the images generated have grown higher in resolution and larger in quantity. MRI machines, produce a sequence that contains several hundred high resolution images, each image representing a slice of the tissue being filmed. Processing such sequences requires extensive computation. Doing such computations on one processor takes very long time. One of the most common processing procedures is to cluster the sequence of images in 3D, that is, connect different clusters that exist in each slice, with ones that exist in the next slice, which would correspond to segmenting a muscle or a bone. In the current project, a multi-core architecture is explored.

2 Proposed Algorithm

A common approach to solving the problem of detecting 3D clusters is to divide the problem into two parts, generate clusters on each image separately, then connect these clusters across images. This solution is map-reduce compatible. Therefore the problem is divided into two subtasks::

1. Segment each slice (Map)
2. Merge clusters of consecutive slices (Reduce)

2.1 Segment each slice

The first step is to segment each slice. Therefore the slices are distributed in a load-balanced linear data distribution manner. Basically, every processor gets a number of image slices I_p which is calculated as follows:

$$I_p = \lfloor \frac{N + P - p - 1}{P} \rfloor, \tag{1}$$

where N denotes the number of slices, P the total number of processors used and p the current active processor.

Each core computes a sequential k-means on its designated slices. This can be seen in details in Procedure *Cluster* in Algorithm 1.

2.2 Merge clusters of consecutive images

The clusters that were generated in the previous step are then combined together into one set of clusters. This has to be done sequentially in order to maintain stability in the labels. This sequential merging was achieved by having each processor merge the labels of its preceding processor, and send its new set of labels to the following processor. This can be seen in details in Procedure *Merge* in Algorithm 1.

Algorithm 1 3D K-means

```
1: procedure CLUSTER
2:     $N \leftarrow$ number of Slices
3:     $P \leftarrow$ number of Processors
4:     $p \leftarrow$ processor Id
5:     $p\_images \leftarrow$images[range($I_{p-1}, I_{p-1} + I_p - 1$)]
6:     for $i$ in 0, length($p\_images$) do
7:         $labels[i] \leftarrow$ k_means($p\_images$[i])
8:         $blobs[i] \leftarrow$ connected_components($p\_images[i], labels[i]$)
9: procedure MERGE
10:    $c\_blobs \leftarrow$ combine_blobs($blobs[0], blobs$)
11:    if $p \mathrel{!}= 0$ then
12:        receive($previous\_blobs$)
13:        combine_blobs($blobs, previous\_blobs$)
14:
15:    send($p + 1, c\_blobs$)
16: procedure COMBINE_BLOBS(blobs, blobs_list)
17:    $intersections \leftarrow$ intersect_blobs_rectangle($blobs, blobs\_list$)
18:    for $i$ in 0, length($intersections$) do
19:        $new\_blobs \leftarrow blobs[intersections[i]]$
```

3 Theoretical Performance Analysis

Since our approach to the problem was that of an embarassingly parallel problem, we define the time complexity in terms of number of images processed per unit time. For Task 1, the complexity of the algorithm for the sequential version is $O(N)$. Then, the complexity is $O(N/P)$ for the parallel version. For Task 2, the complexity of the algorithm is again $O(N)$, and for the parallel version, it is the same.

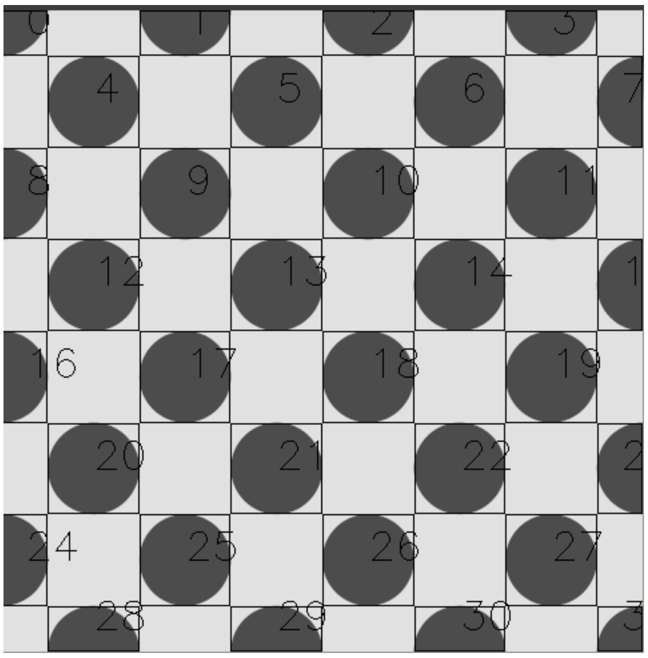

Figure 1: Example of segmentation and blob labelling

4 Implementation Details

In order to be able to run the implemented code and check its performance, we needed to find a dataset containing MRI scans, in quantities and sizes that were large enough, so as to render the parallelization meaningful. However, since finding medical data is not easy, we decided to use a regular image.

We used a single image in two different sizes and replicated it in order to simulate the different slices of an MRI scan, where the number of slices was given as an input to the program.

When running the code locally, we are able to show the segmented slices, where the blobs are indicated by a numbered rectangle, where the number gives the label of the blob. After the merging of the clusters has occurred, the blobs of the different slices that correspond to a larger 3D segmentation have the same label.

When running the program on PDC, since it is not possible to show the images, we save the important information to a file, where each line gives us the slice number, blob label, coordinates and dimensions of the rectangle surrounding the blob.

5 Example

As an example in Figure 1 we can see that the different dots are detected and labelled correspondingly. A total number of 32 blobs has been detected. In the case of different slices (here we only see one), the adjacent blobs would be labelled accordingly. For example, the dot that lies “under” blob 21, would also be labelled as 21 so that we can know that it is one 3D blob.

6 Experimental Speedup

In order to be able to evaluate the performance of our implementation, we ran a number of experiments on multiple core with varying code parameters, such as the number of processors used, the size of the image and the number of slices.

In Figures 2 to 5 we have collected most results of our experiments in compact graphs. The graphs show the run times and speed-ups in relation to the number of processors using increasing numbers of slices for an image sized $[1024 \times 1024]$ and $[768 \times 768]$.

First, we can notice in the graphs, that the different lines start and end on different points. That is, because depending on the size of the image, each processor has a limit of the number of slices it can handle memory-wise.

There is an obvious decrease in the run times with increasing number of processors. However, we can notice that for a constant work load the run times tend to stabilize after a certain addition of processors. That is an indication of Amdahl’s law, showing that there is a certain speed-up that can be achieved for a fixed workload.

We can focus on the lines representing number of slices ranging in $[64 \ldots 256]$, which we assume to be a more realistic simulation of the size of an MRI scan. For these experiments we can see that the parallelization has a very significant positive effect and since the load is quite large, the effect of Amdahl’s law appears only for a small portion of runs for 64 slices.

Finally we have ran the experiments with 24 processors for both image sizes and increasing number of slices on one and two nodes. In Figure 6 we can see that the run time increases linearly as the work load increases. We can also see that running the experiments on two nodes instead of one has a tiny effect in the run time, which indicates that inter-node communication is insignificant for the nature of this problem.

Table 1 summarizes the speed-ups for varying parameters, the ones that we believe are most indicative and most likely to represent a true experiment.

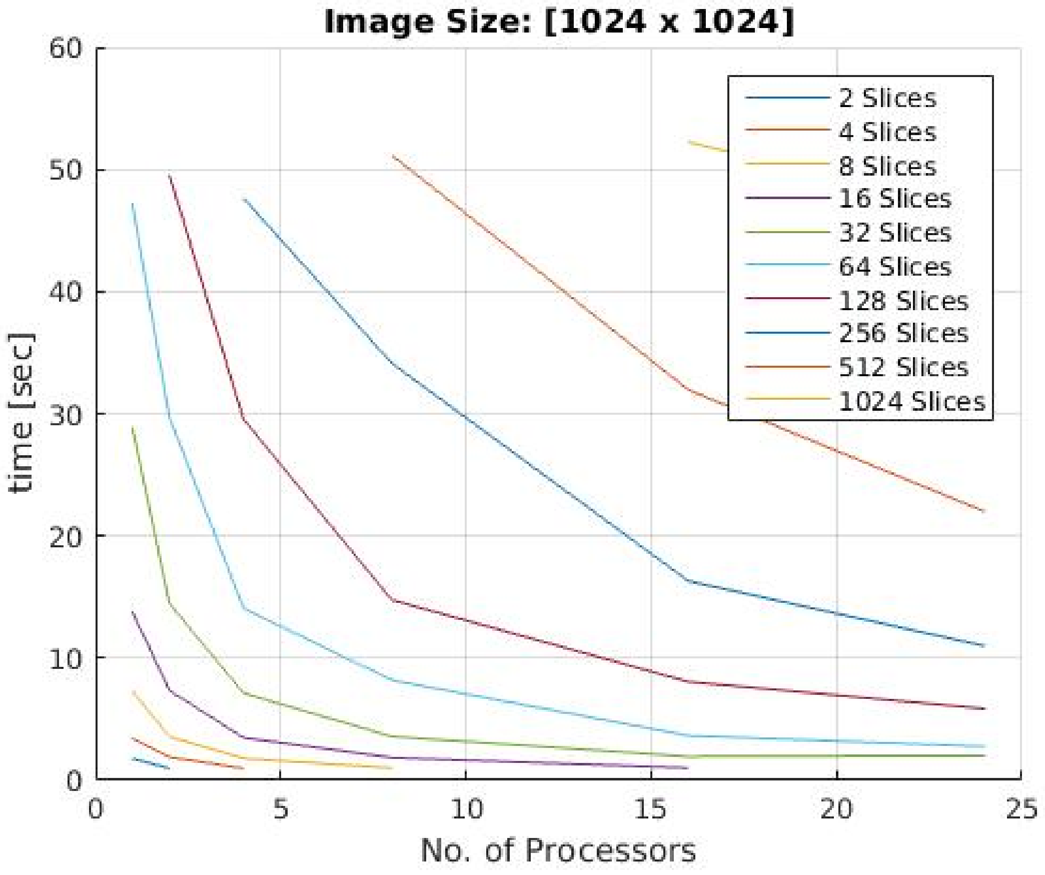

Figure 2: Run time in relation to number of processors for different number of slices. Image size: $[1024 \times 1024]$

Table 1: Speed-ups for different numbers of Processors and Slices

Image Size: $[1024 \times 1024]$						Image Size: $[768 \times 768]$					
64 Slices		128 Slices		256 Slices		64 Slices		128 Slices		256 Slices	
P	S_P	P	S_P	P	S_P	P	S_P	P	S_P	P	S_P
1	1	1	-	1	-	1	1	1	1	1	-
2	1.59	2	1	2	-	2	2.05	2	1.66	2	1
4	3.35	4	1.56	4	1	4	4.73	4	3.83	4	1.89
8	5.77	8	3.36	8	1.39	8	8.4	8	7.36	8	3.35
16	12.9	16	6.15	16	2.92	16	15.93	16	13.13	16	6.59
24	17.1	24	8.47	24	4.33	24	21.62	24	19.79	24	10.11

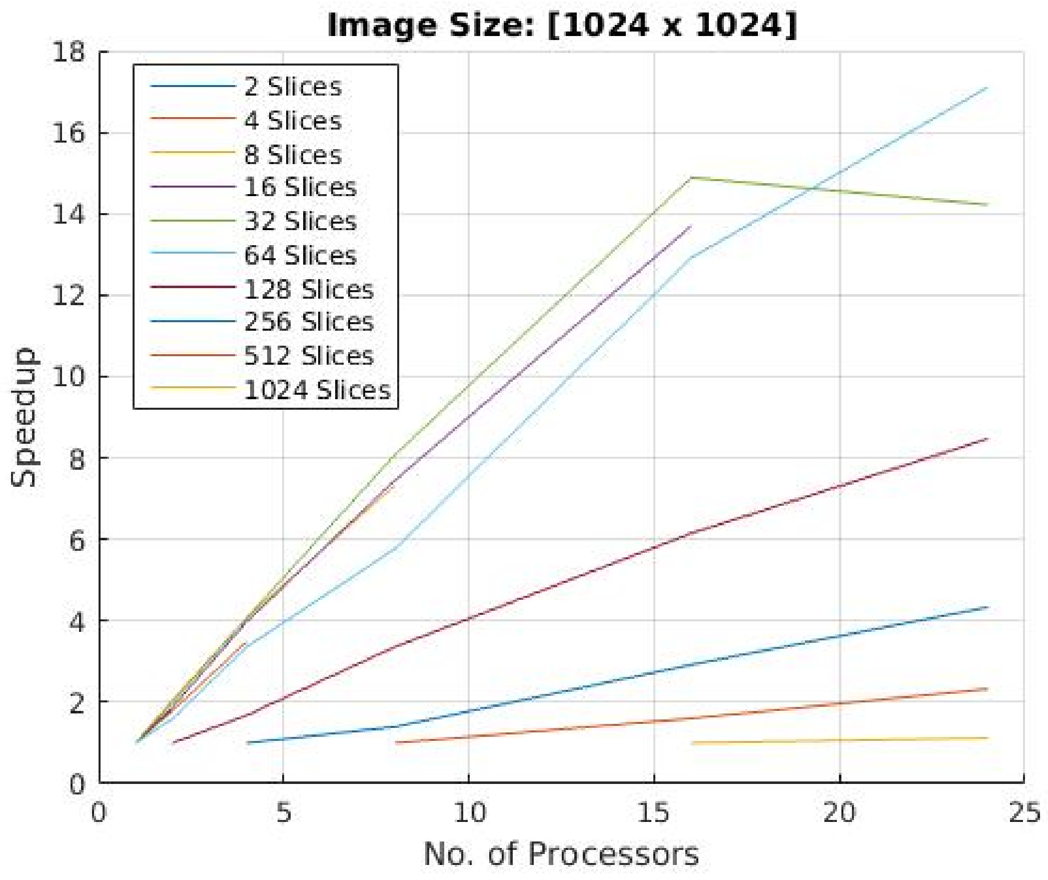

Figure 3: Speed-up in relation to number of processors for different number of slices. Image size: [1024×1024]

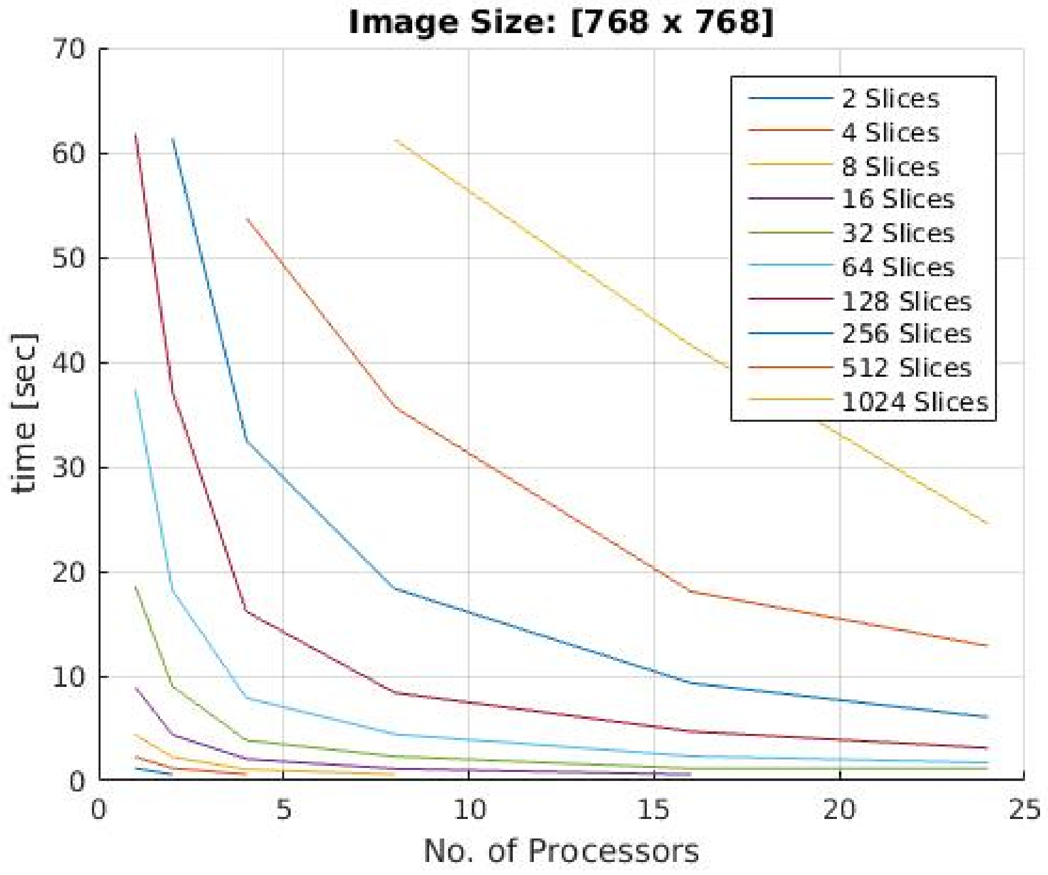

Figure 4: Run time in relation to number of processors for different number of slices. Image size: [768 $\times$ 768]

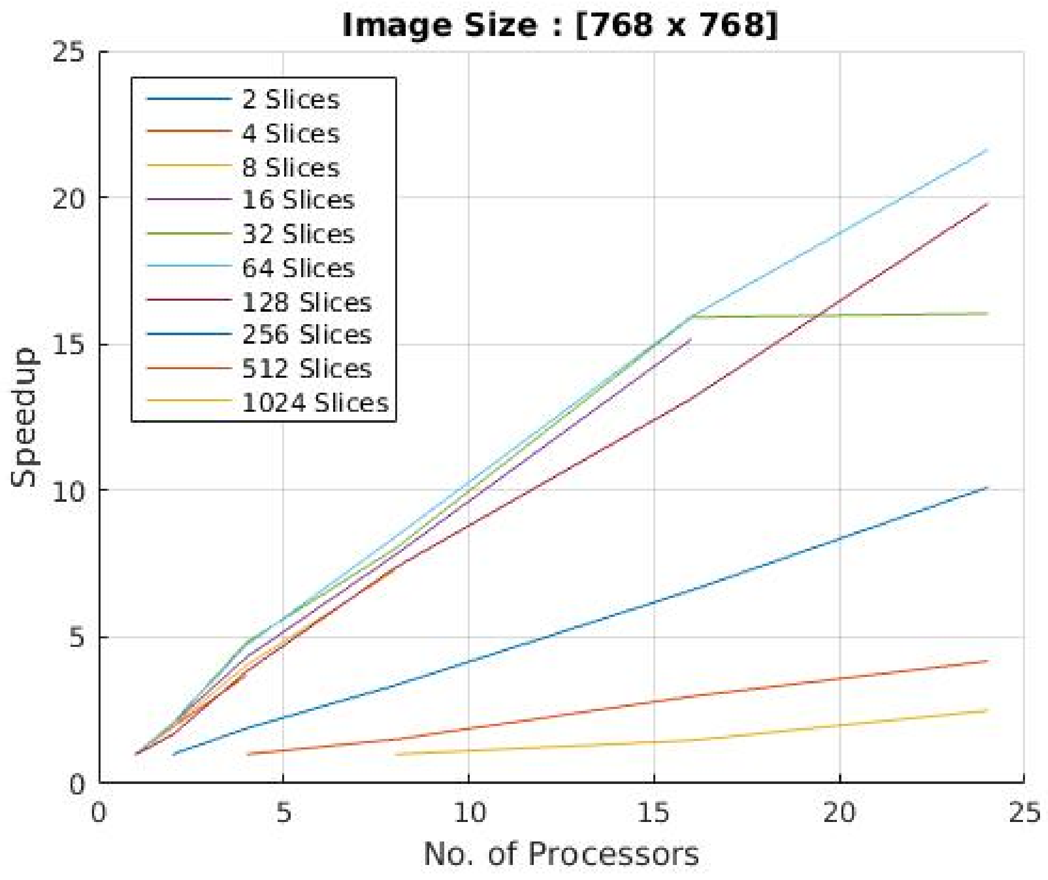

Figure 5: Speed-up in relation to number of processors for different number of slices. Image size: [768 × 768]

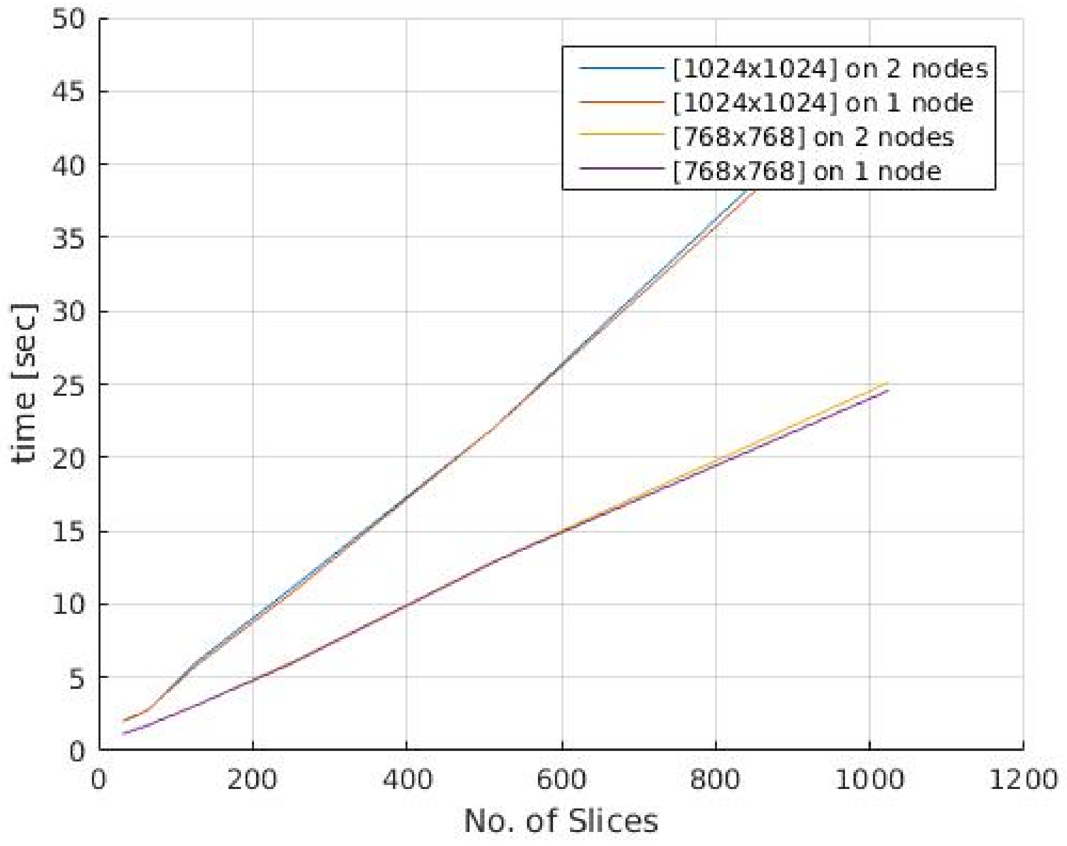

Figure 6: Run time in relation to number of slices using 24 processors on one and two nodes