Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 108C (2017) 1622–1631

Procedia Computer Science

International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland

Transforming a Local Medical Image Analysis for Running on a Hadoop Cluster

Marco Strutz[1], Hermann Heßling[1], and Achim Streit[2]

[1] HTW Berlin, University of Applied Sciences, Wilhelminenhofstraße 75A, D–12459 Berlin, Germany
Email: marco.strutz,hessling@htw-berlin.de

[2] Karlsruhe Institute of Technology (KIT), Steinbuch Centre for Computing (SCC), Hermann-von-Helmholtz-Platz 1, D–76344 Eggenstein-Leopoldshafen, Germany
Email: achim.streit@kit.edu

Abstract

There is a progressive digitization in many medical fields, such as digital microscopy, which leads to an increase in data volume and processing demands for the underlying computing infrastructure. This paper explores scaling behaviours of a *Ki–67 analysis* application, which processes medical image tiles, originating from a WSI (*Whole Slide Image*) file format. Furthermore, it describes how the software is transformed from a Windows PC to a distributed Linux cluster environment. A test for platform independence revealed a non-deterministic behaviour of the application, which has been fixed successfully. The speedup of the application is determined. The slope of the increase is quite close to 1, i.e. there is almost no loss due to a parallelization overhead. Beyond the cluster's hardware limit (72 cores, 144 threads, 216 GB RAM) the speedup saturates to a value around 64. This is a strong improvement of the original software, whose speedup is limited to two.

Peer-review under responsibility of the scientific committee of the International Conference on Computational Science

Keywords: distributed computing, big data, virtual microscopy, virtual microscopy, biomarker, whole slide imaging

1 Introduction

In the area of digital pathology, digital microscopy offers promising potentials such as 3D tissue reconstruction [2]. In contrast to conventional microscopy, pathologist will perform diagnoses by investigating images on computer screens. These images are generated by high-resolution scanning of tissue slides [5]. To aid diagnoses, pathologists typically use a single workstation to run various applications and analyses. To handle image data, *whole slide imaging* (WSI) offers a standard for digital slides [6]. It allows to save and retrieve image data at different file sizes using a single file. By resolutions of more than 20 GigaPixels per file, current analysis applications are confronted with hardware limitations, though. Such enormous resolutions are

1877-0509
Peer-review under responsibility of the scientific committee of the International Conference on Computational Science
10.1016/j.procs.2017.05.227

producing data in Gigabyte or Terabyte ranges. The layout of a whole slide image stack is illustrated in Figure 1.

Processing a digital slide already requires many gigabytes of memory before any kind of analysis on these data can run. Today, due to limited memory resources of a single workstation a digital slide is often processed only at a reduced resolution. In addition, a single image is split into equally sized tiles, which afterwards are processed in batch jobs on Windows workstations.

A main drawback of these two workarounds is their limited scaling property. They will always be affected by the performance limits of a single desktop and the steady increasing computing demands of virtual slides.

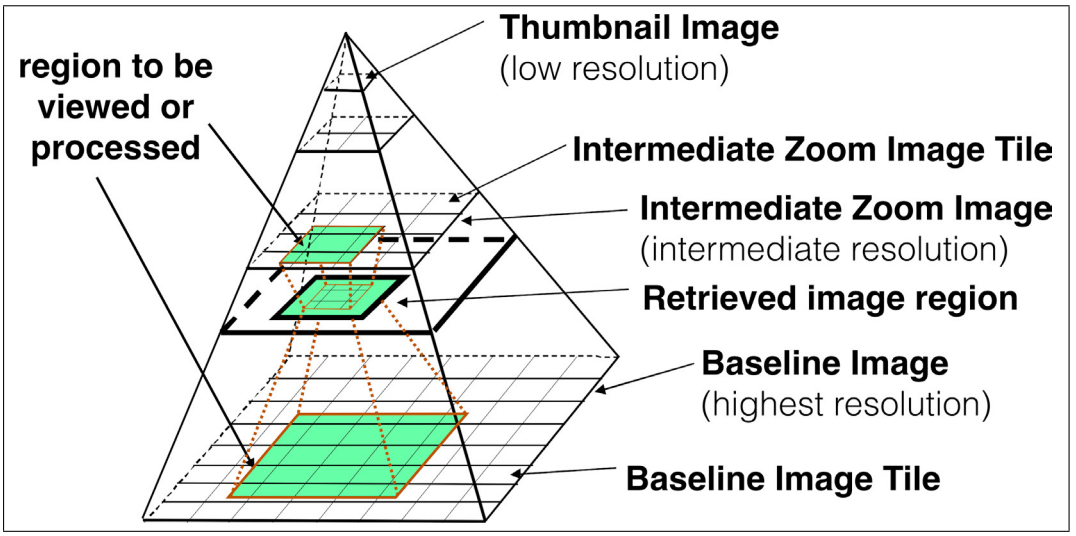

Figure 1: Layout of a whole slide image stack according to the DICOM Standards Commitee (Supplement 145).

2 Distributed Computing and Image Tiling

A well-established way to overcome the bottlenecks of image-processing on single workstations is provided by the methods of parallel and distributed computing. For example, a problem domain can be reduced by dividing it into smaller parts and then handling the parts separately. Each part is processed by a single or many computing nodes and a single node processes one or many parts depending on the design. A speedup can be expected if an algorithm is adapted in a way that it benefits from the underlying design. Speedup is given by the ratio of the execution time without parallelization divided by the execution time on a parallel environment.

Splitting a single image into many tiles is a common approach to enable a parallel processing [14, 11]. This often creates boundary problems. In Figure 2, the centre of a single cell nucleus is located at the edge of two neighbouring tiles. For example, a cell segmentation on those two images tiles needs to consider overlapping cell nuclei appropriately. A new component has to be introduced into the algorithm which is able to handle this particular case. Solving boundary problems is not part of this paper though. Instead, the focus is set to present first results on how a selected locally running medical image analysis software can be executed on a distributed cluster. Boundary problems will be explored in subsequent investigations.

There are various software frameworks and engines supporting a distributed processing. Often, they require a setup of homogeneous hardware components to operate such as *Thrill* [3] ("High-Performance Algorithmic Distributed Batch Data Processing with C++"). However, not every computing cluster is exclusively equipped with homogeneous hardware. Rather heterogeneous components are available or a mix of both setups. This poses a problem as the selection of appropriate systems and software components is limited by the mix of available hardware. Therefore a solution for those heterogeneous layouts is needed which cannot be converted to homogeneous components but still want to benefit from novel parallel processing concepts such as *Map Reduce* [7] or *Resilient Distributed Datasets* [13].

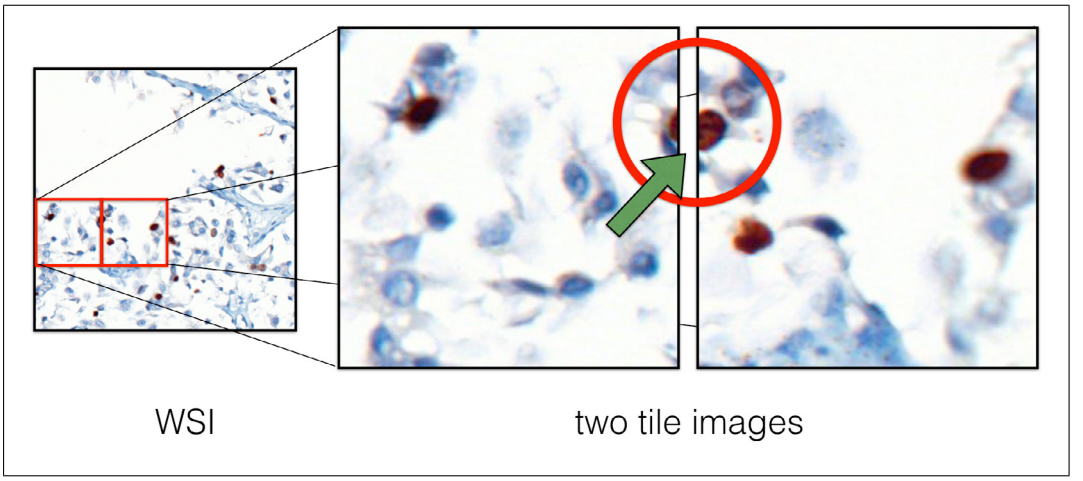

Figure 2: The shown tile images are originated from a WSI with Ki67–DAB and Hematoxylin background stain. At the edge of both tiles a cell nucleus overlaps. This is a typical boundary problem and needs to be taken into account when parallelizing segmentation algorithms.

3 Ki–67 Analysis

The protein Ki–67 has been established as one of the most important biomarkers for cell proliferation in breast cancer [8, 9]. High Ki–67 values indicate high tumor growth and have direct impact on the patient's treatment. Several automated image analysis methods for identifying Ki–67-positive and negative tumor cells have been developed [4]. There are various software frameworks which implement a Ki–67 analysis. They differ in speed, accuracy, and correctness of their computed output scores. Only a few of them were validated with respect to their output results. For example, the computed results of a cell segmentation can be compared with a manual segmentation performed by a pathologist. The application used in this work to perform Ki–67 analyses is developed by VMScope GmbH. It is used at Charité – Universitätsmedizin Berlin (Germany). See [10] for more details on this application's validation. The software itself is optimized for running on a single workstation and, in particular, not designed to run on a distributed environment. Furthermore, it is developed to run on Microsoft Windows, is written in C# and uses .NET build-in methods of concurrency such as *Parallel.For* instructions. The Ki–67 analysis software takes an image as input data and outputs a score which represents the rate of cell proliferation for tumor cells. The score ranges between 0 to 1 and spans three classes: class A (low tumor growth) with values < 0.15, class B (medium tumor growth) with

values of 0.15 to 0.35 and class C (high tumor growth) with values ≥ 0.35. At runtime, the software detects the available number of CPU cores and uses multiple threads for execution if more than a single core is present, i.e. the available CPU cores can be utilized during execution, in principle. The efficiency is analyzed in the following section.

3.1 Identifying the Parallel Ratio

Profiling thread states allows to identify how many threads are being used during the execution of an application and how the computing load is distributed among these threads. Microsoft Visual Studio [1] has been gathering the average core utilization from the analysis of randomly picked image tiles. The patterns of thread usage look similar for all pre-selected image tiles. Figure 3 displays such a pattern based on metrics gathered for a Ki–67 analysis of a single image tile. The portion of parallely executed threads (≥ 2) is not exceeding half of the total runtime. In addition, the application is running single threaded for the second half of execution time. Consequently, the application is most of the time not utilizing all available CPU cores.

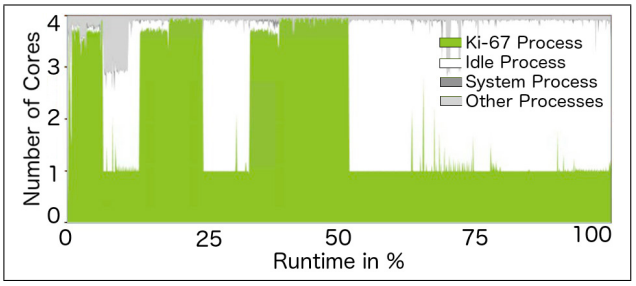

Figure 3: The average core utilization (green) of the analyzed Ki–67 process is shown by the y-axis. The x-axis represents the relative execution time line of the traced application. The *Utilization View* of Microsoft VisualStudio has been used as profiling utility.

3.2 Determining Memory Consumption

Analysing the memory consumption of an application reveals potential bottlenecks and may help to develop optimization strategies for a parallel execution. A Ki–67 analysis was run on a set of image tiles on a Linux workstation using Mono [12] as execution framework. The memory consumption during execution was recorded for each tile and *VmRSS* (Virtual Memory Resident Set Size) values are collected at a sampling rate of one second. A graphical representation of all measured values is presented in Figure 4. It displays the memory consumption of every image tile for a WSI (comprising 3639 tiles in total). The relative execution time is plotted on the x-axis. For example, if an execution takes four minutes then fifty percent equals two minutes. The y-axis represents all tiles by number and are mapped to a unique image tile. These values are also sorted in ascending order based on values of the z-axis. The relative deviation of memory consumption in percent is plotted on the z-axis. Zero represents the mean value during execution for a single tile. Each tile has a different mean value. For example, a deviation of 100 % indicates the memory consumption for the corresponding x-value. As shown in Figure 5, the average memory consumption of all tiles is 618 MBytes. We observe a similar pattern of memory consumption for all tiles. At the beginning of runtime, the memory consumption rises and saturates at its maximum value. Then it keeps a threshold and finally

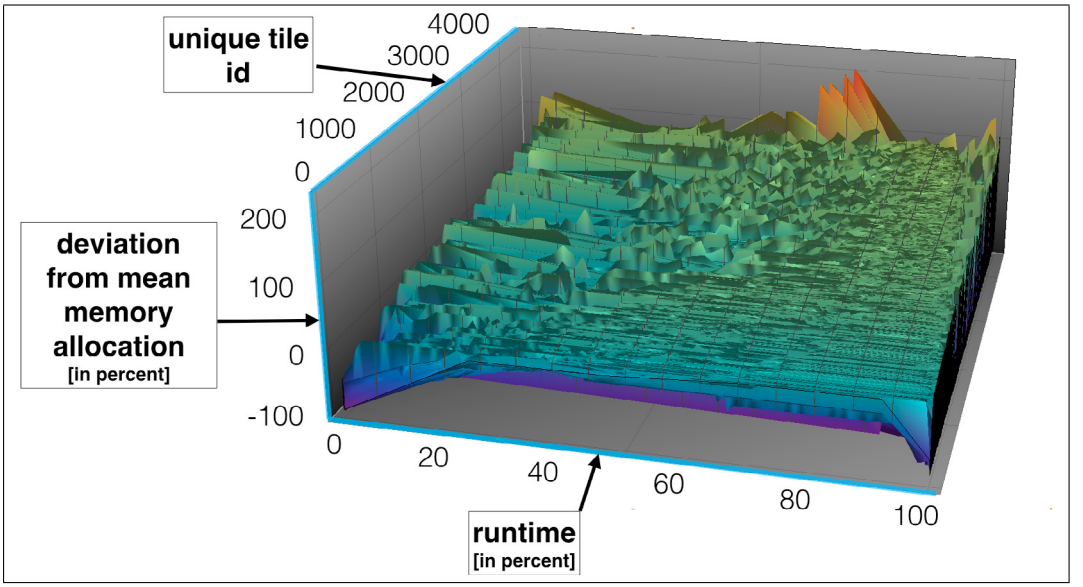

Figure 4: Memory consumption of a typical Ki–67 analysis. The x-axis represents the relative execution time in percent. The z-axis represents the relative memory consumption (VmRSS) compared to the mean consumption of the according tile. The y-axis represents the ID of a single tile.

at the end of runtime memory is released again. Also, the peak value deviates across all image tiles. The quantity memory consumption may be helpful in determining the Ki–67 score of an image tile, in particular of those images tile that show a long plateau.

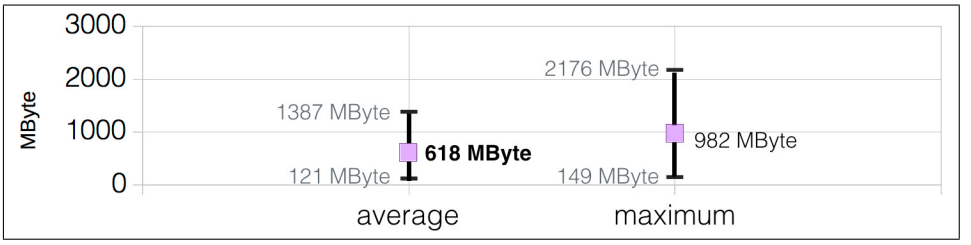

Figure 5: In absolute values, the average memory consumption of the Ki–67 analysis software for our dataset is 618 MByte per tile, with a minimum deviation of 80 % (equals 121 MByte) and a maximum deviation of 224 % (equals 1387 MByte). The average of the maximum memory consumption of all tiles is 982 MByte, with a minimum deviation of 85 % (equals 149 MByte) and a maximum deviation of 221 % (equals 2176 MByte). The variation in memory occupation is mainly a result of how the analysis program is handling image contents and qualities of various input files.

Iterations

Threads	1	2	3	4	5	6	7	8	9	10
1	0.083	0.214	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083
2	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083
3	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083
4	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.214

Figure 6: An example for non-deterministic results produced by the Ki–67 analysis software. The same image is processed with 1, 2, 3 and 4 threads and with 10 iterations for each thread. In total, 40 runs are performed. In 95 % of all iterations, the Ki–67 analysis results in a score of 0.083 [green background] representing the class 'low tumor growth'. In 5 % of all iterations, the score 0.214 [yellow background] is computed and represents the class 'medium tumor growth'.

3.3 Testing For Platform Independence

When executing an application on heterogeneous platforms one has to verify for correctness of results. Even computation across different CPU architectures may differ. After running first tests with the Ki–67 application we observed a discrepancy between Windows and Linux for the same set of input images. A detailed study of the source code showed that this is due to the handling of intermediate computations which creates time-bound results. For further computations, a reference set of image tiles is created. Half of this set contains randomly selected of image tiles based of a WSI. The other half is based of the same WSI as well, but contains image tiles satisfying specific conditions. Such a condition can be an image having just a single cell nucleus and hardly no other textures. Then, the Ki–67 application processes both images sets on Windows and Linux. Afterwards, the obtained scores are compared for both platforms. Analysing the results has revealed non-matching scores. By examining the source code of the application three functions could be identified which caused non-deterministic results as illustrated by Figure 6. The classification of a very same image changes by varying the number of processing threads and also when not changing any execution parameter at all. Indicators to look at are parallel statements of .NET such as *lock* and *Parallel.For*. These are the main structures which are used by the application for achieving a local parallel processing. The way by which the *Parallel.For* code block is handling intermediate computations produces time-bound results. These results vary by the number of threads the *Parallel.For* code block is using. To fix the issue all three functions were restricted to use a single thread. It is a valid method to reduce any impact of changing the underlying algorithm. First benchmarks indicates a performance loss of less then five percent. Modifying the underlying algorithms of the three affected functions could have introduced other side effects which may not be intended by the original author of the software. To summarize, the modified Ki–67 application is now producing identical results on both operating systems Windows and Linux.

3.4 Benchmarking

The benchmark is used to examine how the Ki–67 applications performs when running on a distributed environment as shown in Figure 7. The overall processing time of an image set is measured. In preparation, a typical WSI is split into equally sized image tiles. These tiles results in a total dataset of 5 GB. Each tile has a dimension of 1024×1024 pixels and is encoded as PNG file format. The original application sequentially processes all images as illustrated in Figure 8..

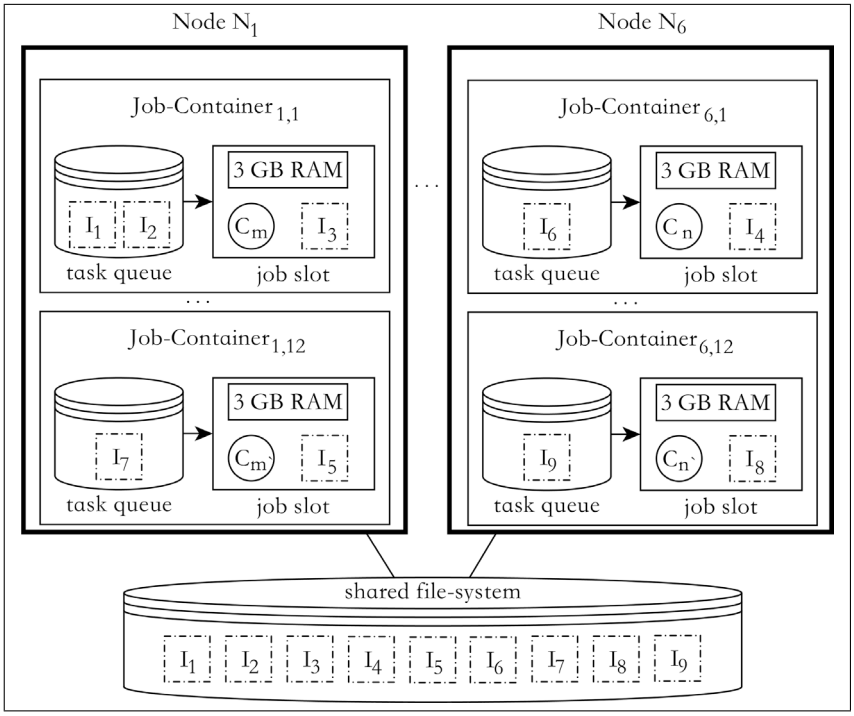

Figure 7: Cluster set-up used for benchmarking. It illustrates an example for job allocation and processing of a nine image tile dataset (I_1 to I_9) running in parallel across six nodes (N_1 to N_6). A CPU core (C) is assigned to each job slot. Each job slot spawns a new Ki–67 application and sequentially processes image tiles from its task queue.

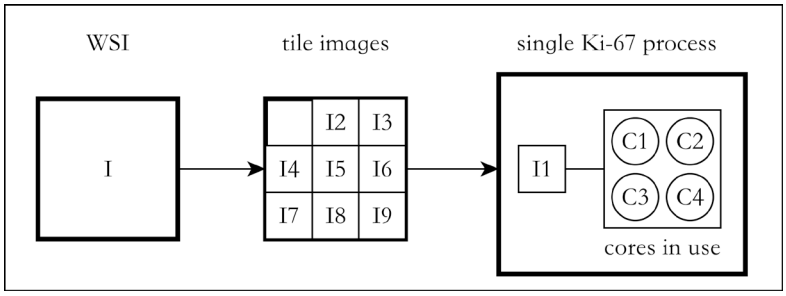

Figure 8: Schema of a Ki–67 analysis running on a single Windows PC. It processes single image tiles by using all available CPU cores provided by the execution environment.

Figure 9 shows the schema how image tiles are being processed by running Ki–67 processes in parallel. The image tiles are subdivided into two sets. Each set is now executed by a separate Ki–67 instance. Afterwards both sets are subdivided again and four parallel running instances of the Ki–67 application sequentially processing their assigned image batches. This schema is continued till 512 separate data sets are processed. The number of parallel running Ki–67 instances is determined by the underlying scheduling software. A cluster of six nodes is used, providing 72 CPUs (144 Threads) and 216 GByte of memory (3 GB per core). The

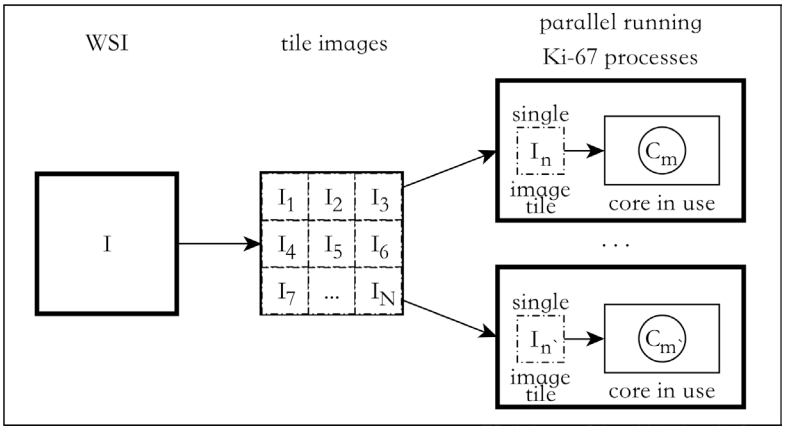

Figure 9: Cluster execution schema of a parallel Ki–67 analysis. Each process uses a single CPU core. In contrast to Figure 8, many tiles are processed in parallel across their assigned cluster nodes.

job scheduling is managed by *YARN2* of *Apache Hadoop* and all datasets are stored within the distributed filesystem *HDFS* (Hadoop Distributed File System). Several test runs are conducted while changing the number of requested container available to process all image tiles. After a test run is committed, YARN2 provisions resources accordingly. Within a single container a Ki–67 application is instantiated and sequentially processes its assigned image tiles. Afterwards, the YARN2 scheduler detects completed jobs and collects its return values. We are using specifically prepared input files to influence the parallel ratio within Hadoop. Each file comprises an equally sized sub-set of 3639 images where each element represents a HDFS encoded file-location path. The number of input files reflects the number of partitions Hadoop is using for a test run. For example, two input files result in the provisioning of two job-container as long as sufficient resources are available. Both running in parallel and each container sequentially processes about 1820 image tiles. Four input files result in four job-container, and so on. Only for a single input file our Hadoop environment behaves differently. Instead of a single one, two job-container are created. This produces similar measurements for a single and for two input files. Due to this anomaly, the processing times for one and two job-containers are equivalent. A determination of the speedup has to take this anomaly properly into account. This is most easily done by effectively doubling the processing time for one container, i.e. we calculate the speedup from the formula

$$S(n) = \frac{2T(1)}{T(n)}$$

where $T(n)$ denotes the total time for processing all tiles using n containers. The results of our speedup measurements are shown in Figure 10. At low degrees of parallelization, the speedup evolves parallel to the boundary of the regime of super-linear speedup. The speedup for the number of requested container saturates for values greater than 64. This is related to the cluster's hardware limit and the way of Hadoops job provisioning. By assigning 3 GB of memory for a job, Hadoop divides the total available memory (216 GB) by three which leads to a upper limit of 72 containers. Exceeding this values leads to an over-provisioning of image tiles. A decline in speedup for $n = 64$ requested container is observed. This seems to be a systematic effect as the error (standard deviation) is systematically smaller than the observed effect.

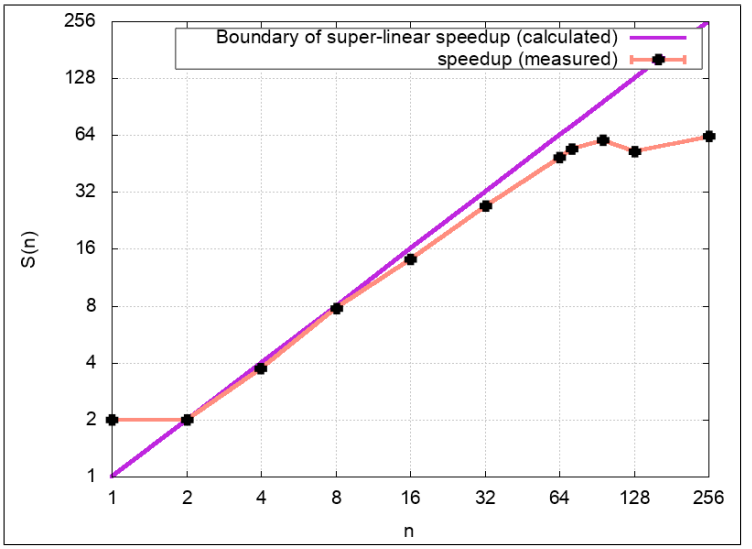

Figure 10: Runtime measurements of the Ki–67 application which is executed in parallel across a six node *Hadoop* cluster. In total, 96 container are being used at maximum load in parallel where each container is sequentially processing a stack of image tiles. For reference, the boundary of the region of super-linear speedup is indicated. The speedup anomaly at $n = 1$ is set out in the text.

4 Conclusion

A reference framework for Ki–67 analyses is explored in detail. It is shown that more than 50 % of the time it is running as a single threaded process, i.e. its maximum speedup is smaller than 2. Although the application is being developed to run on a *Microsoft .Net* platform it is now able to be executed on a *Linux* environment and shown to produce the same results. To verify this assertion, the source code was analyzed in detail. It turned out that by changing the platform from Windows to Linux the original algorithms were affected by non-deterministic behaviours. The root of this flaw is identified and fixed. This fix has only a minor impact on the performance. The speedup of the modified Ki–67 software is determined by performing several benchmarks in a distributed environment. It is shown that the speed increases linearly with the degree of parallelization and that the slope of the increase is nearly one, i.e. the efficiency of the parallelization is almost unaffected. However, the speedup saturates above the cluster's hardware limit of 72 cores. There may be a relation between the content of image tiles and the memory consumption during processing a Ki–67 application. In conclusion, due to various runtime analyses and precise source code optimizations, the selected Ki–67 application is now able to run in a distributed environment and a strong improvement in speedup from 2 to 64 is obtained. It should be noted that the observed upper limit in speedup is merely due to the available hardware. The methods presented in this work allow to improve the speedup even more by using a more powerful cluster. So far, fixed sizes for image tile are used. Next steps are to investigate, how various tiling strategies influencing the Ki–67 analysis scores and its performance figures. Working on different image tiles often creates boundary problems which also can lead to a misclassification. This issue will also be explored in an upcoming investigation.

References

[1] Visual Studio 2015. Concurrency visualizer: Utilization view. 2017.

[2] Ingrid Augusto, Douglas Monteiro, Wendell Girard-Dias, Thaisa Oliveira dos Santos, Simone Letícia Rosa Belmonte, Jairo Pinto de Oliveira, Helder Mauad, Marcos da Silva Pacheco, Dominik Lenz, Athelson Stefanon Bittencourt, Breno Valentim Nogueira, Jorge Roberto Lopes dos Santos, Kildare Miranda, and Marco Cesar Cunegundes Guimarães. Virtual reconstruction and three-dimensional printing of blood cells as a tool in cell biology education. *PLOS ONE*, 11(8):e0161184, aug 2016.

[3] T Bingmann. Thrill: High-performance algorithmic distributed batch data processing with c++. aug 2016.

[4] Daniel G Booth, Masatoshi Takagi, Luis Sanchez-Pulido, Elizabeth Petfalski, Giulia Vargiu, Kumiko Samejima, Naoko Imamoto, Chris P Ponting, David Tollervey, William C Earnshaw, and Paola Vagnarelli. Ki-67 is a PP1-interacting protein that organises the mitotic chromosome periphery. *eLife*, 3, may 2014.

[5] Gloria Bueno, M. Milagro Fernández-Carrobles, Oscar Deniz, and Marcial García-Rojo. New trends of emerging technologies in digital pathology. *Pathobiology*, 83(2-3):61–69, apr 2016.

[6] DICOM Standards Committee. Digital imaging and communications in medicine (dicom) supplement 145: Whole slide microscopic image iod and sop classes. page 8, aug 2010.

[7] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[8] M. Dowsett, T. O. Nielsen, R. A Hern, J. Bartlett, R. C. Coombes, J. Cuzick, M. Ellis, N. L. Henry, J. C. Hugh, T. Lively, L. McShane, S. Paik, F. Penault-Llorca, L. Prudkin, M. Regan, J. Salter, C. Sotiriou, I. E. Smith, G. Viale, J. A. Zujewski, and D. F. Hayes. Assessment of ki67 in breast cancer: Recommendations from the international ki67 in breast cancer working group. *JNCI Journal of the National Cancer Institute*, 103(22):1656–1664, sep 2011.

[9] E. C. Inwald, M. Klinkhammer-Schalke, F. Hofstädter, F. Zeman, M. Koller, M. Gerstenhauer, and O. Ortmann. Ki-67 is a prognostic parameter in breast cancer patients: results of a large population-based cohort of a cancer registry. *Breast Cancer Research and Treatment*, 139(2):539–552, may 2013.

[10] F. Klauschen, S. Wienert, W. D. Schmitt, S. Loibl, B. Gerber, J.-U. Blohmer, J. Huober, T. Rudiger, E. Erbstosser, K. Mehta, B. Lederer, M. Dietel, C. Denkert, and G. von Minckwitz. Standardized ki67 diagnostics using automated scoring–clinical validation in the GeparTrio breast cancer study. *Clinical Cancer Research*, 21(16):3651–3657, dec 2014.

[11] Tahsin Kurc, Xin Qi, Daihou Wang, Fusheng Wang, George Teodoro, Lee Cooper, Michael Nalisnik, Lin Yang, Joel Saltz, and David J. Foran. Scalable analysis of big pathology image data cohorts using efficient methods and high-performance computing strategies. *BMC Bioinformatics*, 16(1), dec 2015.

[12] Mono Project. Cross platform, open source .net framework. 2017.

[13] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 15–28, San Jose, CA, 2012. USENIX.

[14] Norman Zerbe, Peter Hufnagl, and Karsten Schlüns. Distributed computing in image analysis using open source frameworks and application to image sharpness assessment of histological whole slide images. *Diagnostic Pathology*, 6(Suppl 1):S16, 2011.