Evaluation of Image Pixels Similarity Measurement Algorithm Accelerated on GPU with OpenACC

Ibrahim Mundher Abdulqader, Kim Chuan Lim
Centre for Telecommunication Research & Innovation (CeTRI), Faculty of Electronic & Computer Engineering, Universiti Teknikal Malaysia Melaka
ibm.shaikhly@gmail.com, kimchuan@utem.edu.my

***Abstract*—OpenACC is a directive based parallel programming library that allows for easy acceleration of existing C, C++ and Fortran based applications with minimal code modifications. By annotating the bottleneck causing section of the code with OpenACC directives, the acceleration of the code can be simplified, leading for high portability of performance across different target Graphic Processing Units (GPUs). In this work, the portability of an implemented parallelizable chi-square based pixel similarity measurement algorithm has been evaluated on two consumer and professional grade GPUs. To our best knowledge, this is the first performance evaluation report that utilizes the OpenACC optimization clauses (collapse and tile) on different GPUs to process a less workload (low resolution image of 581x429 pixels) and a heavy workload (high resolution image of 4500 x 3500 pixels) to demonstrate the effectiveness and high portability of OpenACC.**

***Index Terms*—chi-square; OpenACC, pixel similarity measurement; tile and collapse clauses.**

I. INTRODUCTION

Over the last decade, researchers and developers have been increasingly using General Purpose Graphics Processing Unit (GPGPU) to accelerate the computation of scientific applications and simulations, by offloading a section of the code that contains the heaviest and parallelizable computation on the thousands of computation cores within the GPU [1].

To make use of GPGPU, special application programming interfaces (APIs) are required to extend the functionality of the widely used programming languages such as C/C++ and Fortran, with CUDA library being an example of such APIs [2].

The acceleration of applications with CUDA results in a low portability of performance for applications, that is the ability to accelerate applications on different GPUs without sacrificing performance due to optimization issues, as CUDA is only compatible with Nvidia GPUs and is unsupported on GPUs from other vendors.

Using OpenCL, an alternative to CUDA addresses the low portability issue, however, both CUDA and OpenCL require a lot of effort to port and re-write existing applications.

To overcome these two hurdles, several companies including Nvidia and PGI have collaborated to develop OpenACC [3], an API that uses directive based programming to simplify the acceleration of newly written and legacy applications, with studies showing promising potential and good performance as demonstrated in [4], [5] and [6].

Image processing is one field of study that can benefit greatly from the acceleration on GPUs, due to the variety in image sizes, with one of the widely used algorithms in image processing being the Pixel Similarity Measurement algorithm, which is widely used in implementing image processing techniques including feature extraction and image segmentation [11], [12], [17], [18], [19], [20], [21], [22], [23].

One method for implementing a pixel similarity measurement algorithm is by using Chi-Square test to compute the dis-similarity of each pixel with the 8 neighborhood pixels, followed by comparing the computed values to find the smallest dis-similarity value which represents the most similar pixel value.

In this work, the benefits of combining the OpenACC API with image processing applications are presented in the form of accelerating a developed chi-square based pixel similarity measurement algorithm on two different GPUs.

The contributions of this paper are as follows:

- The GPU kernel implementation of the parallelizable chi-square based pixel similarity measurement algorithm.
- The evaluation of the kernel portability by accelerating the kernel on a consumer grade (GTX 1060) and professional data center grade (Tesla K40c) GPUs.

The remainder of this paper is structured as follows. Section II is the literature review that explains the architecture of OpenACC. Section III is the methodology section that discusses the kernel implementation and how the utilization of GPU resources could affect the acceleration performance. Section IV discusses the hardware specifications and the OpenACC-aware PGI compiler. Section V discusses the experimental results obtained from profiling the kernel execution. Finally, Section VI concludes the paper findings.

II. LITERATURE REVIEW

OpenACC is a directive based programming library developed by Nvidia, PGI, CAPS and CRAY in an effort to create an API standard that provides ease of use and portability across different devices and compilers. The API allows developers to utilize the high-level directive language to specify which sections of the Host (CPU) code are to be offloaded onto the Device (GPU) and accelerated, by using two sets of directives [7], [8].

The first set contains the data management directives, which allows developers to create a data region that automatically manages the transmission of input/output data between the Host and Device, while also allowing for manual control over how to implement the data region.

The second set contains the parallelism directives, that consists of two directives, the first is the *kernels* directive which has been designed for easy acceleration on the GPU by having the compiler to automatically specify the number of

blocks and threads allocated for the code execution.

While the second is the *parallel* directive, which enables the developers to manually specify the allocated number of blocks and threads, thereby allowing for a finer-grained control over the parallelization of the kernel on the thousands of GPU computing cores.

Each of the parallelism directives can be optimized with one of two optimization clauses, the Tile and Collapse clauses. The *tile (M, N)* clause breaks the specified loop into tiles of (*M*, *N*) loops, with "*M*" being the size of the inner loop while (*N*) is the size of the outer loop [15], [16].

While *collapse (x)* clause transforms the subsequent specific number of loops into a single loop thereby creating a single iteration space across all the nested loops which increases execution parallelization of the loop with an increased iteration per GPU computing core.

Meaning when *collapse (2)* clause is used, 2 of the subsequent nested loops with trip counts of *X* and *Y* will be transformed into a single loop of *X***Y* [14], [16].

Currently, there are only three compilers that are OpenACC aware, the PGI compiler which will be used, the CRAY compiler which only works on CRAY systems, and the GCC compiler though only offering partial support with full support of OpenACC is currently being implemented.

III. Methodology

A. Kernel implementation of parallelized pixel similarity measurement algorithm

The chi-square based similarity measurement algorithm consists of two parts as can be seen on Figure 1. The first part is the computation of the dis-similarity between the central pixel and its 8 neighbors by using the chi-square formula seen in (1), in which x^2 is the dis-similarity value, with higher values x^2 mean the more unlikely that the two pixels in comparison are similar, while P_c and P_n represents the values of the central and the neighboring pixels respectively.

$$x^2 = \frac{(P_C - P_N)^2}{(P_C + P_N)} \qquad (1)$$

The second part is the computation of the most similar neighboring pixels, and is achieved by comparing the computed dis-similarity values of the 8 neighbors to find the smallest value that represents the most similar neighboring pixel.

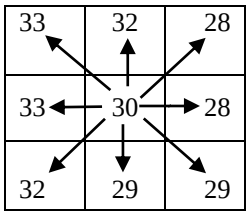
To achieve full parallelizability of the code, eight 2-D arrays are utilized to store the results of the dis-similarity computation between the central pixel with the one of the neighboring pixels as seen on Figure 2; this also improves the memory utilization by allowing the GPU to achieve a coalesced memory access that improves the acceleration performance [13], while allowing for data reusability if needed (modern GPU comes with large memory, therefore, it is acceptable to sacrifice the memory usage to improve the computation).

The code implementation of the GPU kernel seen in Figure 3, shows that each array has been named after the corresponding neighbor that is compared to the central pixel, i.e. the dis-similarity between the central pixel and the bottom left pixel is stored in bot_left array.

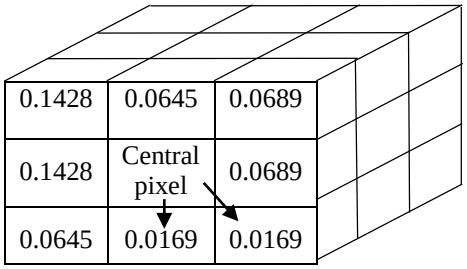
The kernel is then annotated using the *"*#pragma acc kernels loop" without the quotation marks, and optimized using either "collapse (2)*"* or *"*tile (32,4)*"* clauses rather than manually specifying the number of blocks and threads to be allocated for the kernel execution to improve the performance.

The values of tile (32,4) were chosen to ensure that all threads within a warp are allocated to the same row (warp = 32 threads, 64 warps per SM), thereby improving the performance of the kernel; the same goes for the use of collapse (2) due to the code containing two loops.

The data region is implemented lastly to manage the transmission of input and output data between the Host and Device, which limits the transmission of data to only what the Device requests from the Host, thereby improving the performance.



(a) First Part: Computing the dis-similarity between the neighboring pixels and the central pixel with chi-square.



(b) Second Part: Computing the most similar pixels by finding the smallest dis-similarity value of the 8 neighbors.

Figure 1 Computation task of the pixel similarity measurement (graphical illustration).

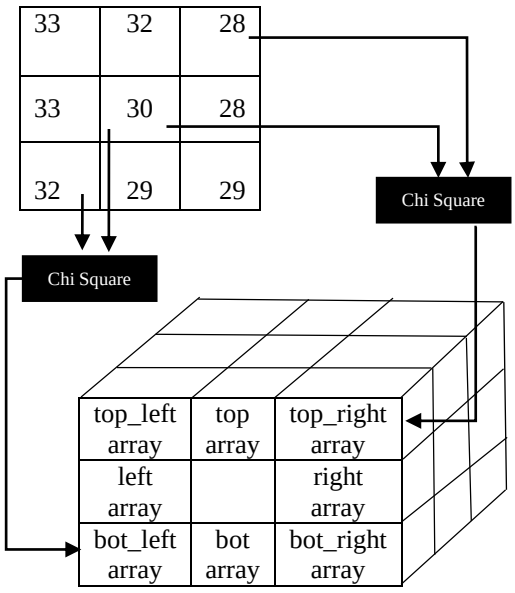


Figure 2 The use of multiple arrays as temporary storage to enable parallel code execution.

B. Using Nvidia Visual Profiler to obtain the insight of GPU utilization

The acceleration performance of the parallelized kernel depends greatly on the utilization of the GPU resources (number of computing cores per GPU, memory and GPU-

```
#pragma acc data copyout(left[:height+1][-1:width+2],bot[:height+1][-1:width+2])
#pragma acc data copyout(thresh_array[:height-1][:width])
#pragma acc data copyin(array[:height+1][-1:width+2])
#pragma acc data copyout(bot_left[:height+1][-1:width+2],bot_right[:height+1][-1:width+2],top_right[:height+1][-1:width+2],top_left[:height+1][-1:width+2],top[:height+1][-1:width+2],right[:height+1][-1:width+2])
{
#pragma acc kernels loop collapse (2) independent
   for (int i=1; i<(height); i++)
    {
        for (int j=0; j <( width); j++)
        {
            if(array[i][j] >0)
            {
                top_left[i-1][j] =       (pow((array[i][j]-array[i-1][j-1]),2.0)) / (array[i][j]+array[i-1][j-1]);

                top[i-1][j] =            (pow((array[i][j]-array[i-1][j]),2.0)) / (array[i][j]+array[i-1][j]));

                top_right[i-1][j] =      (pow((array[i][j]-array[i-1][j+1]),2.0)) / (array[i][j]+array[i-1][j+1]);

                left[i-1][j] =           (pow((array[i][j]-array[i][j-1]),2.0)) / (array[i][j]+array[i][j-1]);

                right[i-1][j] =          (pow((array[i][j]-array[i][j+1]),2.0)) / (array[i][j]+array[i][j+1]);

                bot_left[i-1][j] =       (pow((array[i][j]-array[i+1][j-1]),2.0)) / (array[i][j]+array[i+1][j-1]);

                bot[i-1][j] =            (pow((array[i][j]-array[i+1][j]),2.0)) / (array[i][j]+array[i+1][j]);

                bot_right[i-1][j] =      (pow((array[i][j]-array[i+1][j+1]),2.0)) / (array[i][j]+array[i+1][j+1]);

                thresh_array[i-1][j] =   min(top_left[i-1][j],min(top[i-1][j],min(top_right[i-1][j],min(right[i-1][j],
                                         min(left[i-1][j], min(bot_left[i-1][j],min(bot[i-1][j],bot_right[i-1][j])))))));
            }
        }
    }
}
```

Figure 3 C++ implementation with OpenACC pragma of the chi-square based pixel similarity measurement kernel (with collapse clause of 2 loops)

Host data transaction bandwidth), which can be obtained by profiling the kernel execution with Nvidia Visual Profiler.

Depending on the acquired utilization levels, there are 3 likely reasons why the kernel performance is being limited [10] which are tabulated and shown in Table 1, the first being the latency, which causes a low utilization for both compute resources and memory bandwidth, while the second reason is the insufficient compute resources in the GPU which causes high compute utilization and low memory bandwidth utilization.

While the third reason results in high memory bandwidth utilization with low compute utilization, and is caused by the insufficient memory bandwidth, or the kernel implementation containing too much dependencies.

Table 1: Kernel performance limiters

Compute Utilization	Memory Utilization	Likely limitation
Low (less than 40%)	Low (less than 40%)	Latency Bound
	High (more than 70%)	Memory Bound
High (more than 70%)	Low (less than 40%)	Compute Bound
	High (more than 70%)	-

IV. Experimental Setup

A. *Acceleration Devices*

Two GPUs of different architectures are used to perform the performance and portability evaluation for the kernel, the is a consumer grade GTX 1060 GPU based on the Pascal architecture, and consists of 1280 single precision computing cores and 40 double precision cores distributed on 10 Stream Multiprocessors (SM) with compute capability 6.1

While the second GPU is a professional grade Tesla K40c GPU, based on the Kepler architecture and is designed primarily for GPGPU appliances. The Tesla K40c contains 2880 single precision computing cores and 960 double precision computing cores distributed on 15 SMs with compute capability 3.5 [8], [9].

The performance and memory bandwidth of both GPUs seen in Figure 4 shows that both GPUs capable of achieving similar single precision FLOPs performance, with significant advantage for the Tesla K40c over the GTX 1060 in both double precision FLOPs and memory bandwidth.

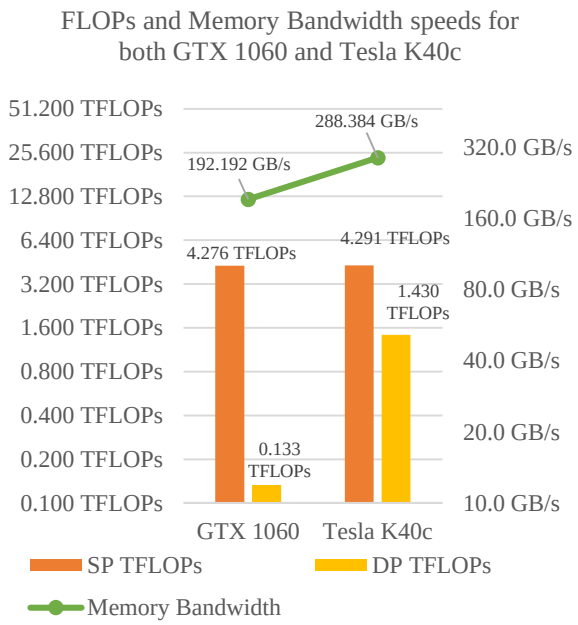


Figure 4 Single precision and Dobule precision FLOPs and memory bandwidth performances for both GPUs obtained from Nvidia's visual profiler (measured in Tera FLOP/s for the compute and Gigabytes/s for the memory).

B. *PGI Compiler*

This compiler is created by Portland Group inc. which is a part of Nvidia as of 2013, and supports C/C++ and Fortran programming languages as well as two directive based parallel processing modules, the first being the OpenACC 2.5 standard with support of acceleration on GPUs, Multi-core CPUs and Coprocessors. While the second parallel processing module is OpenMP 3.1 that allows for the acceleration on multi-core CPUs only.

The compiler initiates the acceleration of the code by analyzing the application and data structures and scans for the annotated sections of the code in order to implement the optimization required in order utilize the hardware threading

capabilities and SIMD vector features available in modern accelerators.

For the setup, PGI compiler v.17.4 (community edition) and CUDA 8.0 are being used for both GPUs, with GTX 1060 running on Ubuntu 16.04 with Linux kernel 4.11, while the Tesla K40c is running on Ubuntu 14.04 with Linux kernel 4.4, while the other hardware specifications do not affect the kernel performance.

V. Results

The kernel is compiled with the PGI compiler using the terminal command line "pgc++ *-fast -acc -Minfo=accel -ta=tesla,ccXY,maxregcount:32 source.cpp -o output*" with XY referring to the compute capability for the GPU (cc60 for GTX 1060, cc35 for Tesla K40c). The performance evaluation tests are conducted by profiling the execution of the kernel optimized with both collapse and tile clauses during the processing of two images on both GPUs; the first image has a low resolution of 581x429 pixels, while the second image has a high resolution of 4500x3500 pixels.

A. *Evaluation of the kernel optimized with OpenACC's collapse clause*

Figure 5 shows the compute and memory resources utilization of kernel optimized with collapse clause, obtained by profiling the kernel performance limiters with Nvidia's visual profiler, during the processing of large image in Tesla K40c. The compute and memory resource utilization of Tesla K40c together with GTX 1060 is tabulated and shown in Table 2.

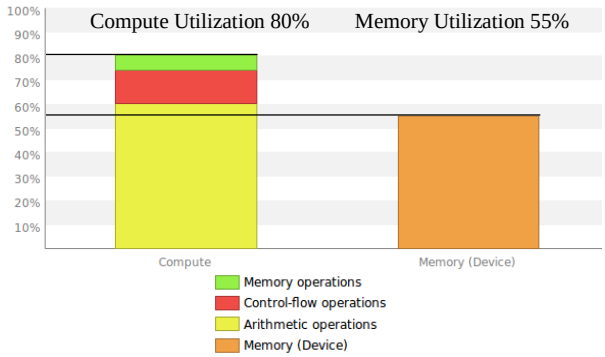


Figure 5 Utilization of Tesla K40c resources during the execution of the kernel optimized with collapse clause to process the high-resolution image

From Figure 5, it can be concluded that the kernel heavily making use of the compute resources (double precision computing cores) and does not make full use of the available memory bandwidth. This comes as a result of using multiple arrays in the kernel, allowing for a coalesced memory access, thus reducing the memory bandwidth required of accessing the global memory.

On the other hand, the same kernel running in GTX 1060 has saturated all the available compute resources (GTX 1060 has 40 double precision compute cores as compared to 960 double precision compute cores in Tesla K40c) and therefore is considered as compute bound.

The achieved execution times of the kernel optimized with collapse clause, obtained with Nvidia Visual Profiler, are presented in Table 3 (see sample of kernel execution timeline of GTX 1060 in Figure 6 when processing the image with the resolution of 581 x 429). The effects of having insufficient number of computing cores can be clearly seen with the large difference of execution time between GTX 1060 and Tesla K40c when processing both images, as seen on Table 3.

Table 2: Compute and memory resources utilization of both GPUs when processing both images with collapse-optimized kernel (collapse (2)).

GPU	Image Size	Compute Utilization	Memory Utilization	Kernel classification
GTX 1060	*Small*	95%	35%	Compute Bound
	Large	95%	35%	
Tesla K40c	*Small*	75%	55%	-
	Large	80%	55%	

Table 3 Execution times for test images on both GPUs with collapse (2) clause

Image Size	GTX 1060	Tesla K40c
Small	0.6769ms	0.172ms
Large	32.099ms	7.227ms

B. *Evaluation of the kernel optimized with OpenACC's tile clause*

The GPU utilization for GTX 1060 and Tesla K40c when processing both small and large images using the kernel optimized with tile clause and their kernel execution time is shown in Table 4 and Table 5, respectively.

Table 4: Compute and memory resources utilization of both GPUs when processing both images with tile-optimized kernel (tile (32,4)).

GPU	Image Size	Compute Utilization	Memory Utilization	Kernel classification
GTX 1060	*Small*	95%	35%	Compute Bound
	Large	95%	35%	
Tesla K40c	*Small*	78%	55%	-
	Large	79%	65%	

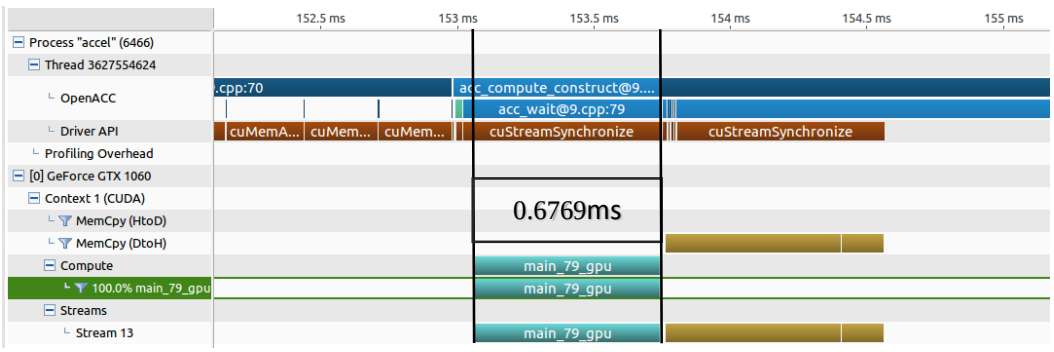


Figure 6 Execution timeline for the kernel optimized with collapse clause on GTX 1060

Table 5: Execution times for test images on both GPUs with tile (32,4) clause

Image Size	GTX 1060	Tesla K40c
Small	0.693ms	0.168ms
Large	32.192ms	7.769ms

The execution of the kernel optimized with tile clause in GTX 1060 is also compute bound due to insufficient number of the double precision compute cores. However, the same kernel did not saturate the GPU resources of Tesla K40c.

Using the obtained kernel execution time in both Table 3 and Table 5, the absolute kernel processing time difference (in percentage) of pixel similarity measurement kernel optimized with both collapse and tile clause were calculated and as shown in Table 6. The difference of kernel execution time small image (less workload) varied around 2.3% for GTX 1060 and 0.2% for Tesla K40c, however, significant difference was observed for large image (heavy workload).

Table 6: Comparing of kernel optimized with tile and collapse clause (absolute difference of kernel execution time in percentage)

Image Size	GTX 1060	Tesla K40c
Small	2.37%	0.2%
Large	2.90%	7.49%

Further analysis, profiling the memory operations conducted by the kernel during the processing of the large image on Tesla K40c, is conducted to understand the performance difference between the two optimizations. The results of profiling, which can be seen on Table 7, shows that the collapse optimization uses a higher L1 cache/shared memory bandwidth and less transactions than the tile optimization, while using less global memory bandwidth (device memory) and transactions.

These results mean that the collapse achieves a much better cache utilization by achieving high bandwidth speeds and requiring less transactions to finish the operations, and at the same time, not requiring many accesses to the global memory, which is the slowest memory in the GPU.

Table 7: Achieved memory/cache bandwidth and transactions of tile and collapse on Tesla K40c when processing the large image.

	Collapse (7.227ms)		Tile (7.769ms)	
Memory Type	Number of Transactions	Bandwidth GB/s	Number of Transactions	Bandwidth GB/s
L1/Shared Memory	16,240,643	272.056	22,515,034	267.874
L2 Cache	65,693,876	292.178	70,687,198	292.198
Device Memory	38,199,354	169.894	46,750,138	193.25

VI. CONCLUSION

In this work, a chi-square based pixel similarity measurement algorithm has been implemented, optimized and accelerated with OpenACC using kernels directive and optimized with both tile and collapse clauses.

Two different GPUs have been used to evaluate the performance and portability of the algorithm, the first is a consumer grade Nvidia GTX 1060 GPU, while the second is a professional datacenter grade Nvidia Tesla K40c accelerator GPU.

The performance evaluation has been conducted by profiling the kernel execution on both GPUs during the processing of two images of small and large sizes respectively, with the results showing a bottleneck of performance on GTX 1060 due to the insufficient compute resources (double precision computing cores), while the Tesla K40c faced no bottleneck of performance.

The results also showed similar performance for both optimization clauses on both GPUs, with collapse clause performing slightly faster that tile in all but one test, while the cause for the performance difference is the better GPU cache memory utilization for the collapse clause.

From the obtained results, it can be concluded that the proposed kernel implementation of the chi-square based similarity measurement algorithm is highly portable.

ACKNOWLEDGEMENT

This work was supported by the research grant no. GLuar/MIROS/2017/FKEKK-CeTRI/I00026.

REFERENCES

[1] Fermi Compute Architecture Whitepaper, "NVIDIA's Next Generation CUDA Compute Architecture: Fermi", [online] available at https://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf [accessed 15 August 2017].

[2] Nvidia CUDA documentation, [online] available at http://docs.nvidia.com/cuda/ [accessed 15 August 2017].

[3] J. Larkin, "Introduction to OpenACC", [online] http://on-demand.gputechconf.com/gtc/2015/presentation/S5192-Jeff-Larkin.pdf [accessed 15 August 2017].

[4] J. Kraus, M. Schlottke, A. Adinetz, and D. Pleiter, "Accelerating a C++ CFD code with OpenACC," *Proc. WACCPD 2014 1st Work. Accel. Program. Using Dir. - Held Conjunction with SC 2014 Int. Conf. High Perform. Comput. Networking, Storage Anal.*, pp. 47–54, 2014.

[5] I. K. Ikeda, F. Ino, and K. Hagihara, "An OpenACC Optimizer for Accelerating Histogram Computation on a GPU," *Proc. - 24th Euromicro Int. Conf. Parallel, Distrib. Network-Based Process. PDP 2016*, pp. 468–477, 2016.

[6] K. Komatsu, R. Egawa, S. Hirasawa, H. Takizawa, K. Itakura, and H. Kobayashi, "Migration of an Atmospheric Simulation Code to an OpenACC Platform Using the Xevolver Framework," *Proc. - 2015 3rd Int. Symp. Comput. Networking, CANDAR 2015*, pp. 515–520, 2016.

[7] S. Lee and J. S. Vetter, "OpenARC: Extensible OpenACC compiler framework for directive-based accelerator programming study," *Proc. WACCPD 2014 1st Work. Accel. Program. Using Dir. - Held Conjunction with SC 2014 Int. Conf. High Perform. Comput. Networking, Storage Anal.*, pp. 1–11, 2015.

[8] NVIDIA Kepler Architecture Whitepaper, "NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110" [online] https://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf [accessed 15 August 2017].

[9] Tesla K40 GPU active accelerator [online] https://www.nvidia.com/content/PDF/kepler/Tesla-K40-Active-Board-Spec-BD-06949-001_v03.pdf [accessed 15 August 2017].

[10] Guided Performance Analysis with the NVIDIA Visual Profiler, [online] http://on-demand.gputechconf.com/gtc/2013/webinar/gtc-express-guided-analysis-nvidia-visual-profiler.pdf [accessed 15 August 2017].

[11] A. N. Selvan, "Boundary Extraction in Images Using Hierarchical Clustering-based Segmentation", *Sheffield Hallam University Research Archive (SHURA), 2011*, [online] http://shura.shu.ac.uk/4050 [accessed 15 August 2017].

[12] J. Lin, D. A. Adjeroh, B. H. Jiang, and Y. Jiang, "K2: Efficient alignment-free sequence similarity measurement using the Kendall statistic," *Proc. - 2016 IEEE Int. Conf. Bioinforma. Biomed. BIBM 2016*, no. 2, pp. 1128–1132, 2017.

[13] Peng Wang, "Introduction to OpenACC, 2017", [online] http://web.stanford.edu/class/cme213/files/lectures/Lecture_14_openacc2017.pdf [accessed 15 August 2017].

[14] OpenACC Programming and Best Practices Guide - June 2015, [online]http://www.openacc.org/sites/default/files/inline-files/OpenACC_Programming_Guide_0.pdf [accessed 15 August 2017].

[15] S. Sawadsitang, J. Lin, S. See, F. Bodin, and S. Matsuoka, "Understanding Performance Portability of OpenACC for Supercomputers," *Proc. - 2015 IEEE 29th Int. Parallel Distrib. Process. Symp. Work. IPDPSW 2015*, pp. 699–707, 2015.

[16] J. Larkin, "Introduction to compiler directives with OpenACC", Nvidia Developer Technologies, [online] http://on-demand.gputechconf.com/gtc/2015/presentation/S5192-Jeff-Larkin.pdf [accessed 15 August 2017]

[17] X. Chen and C. Qi, "Document image super-resolution using structural similarity and Markov random field," *IET Image Process.*, vol. 8, no. November 2013, pp. 687–698, 2014.

[18] M. Wael and A. S. Fahmy, "Classification of Cardiac Magnetic Resonance Image Type and Orientation" *Int. Conf. Image Process.*, vol. 978, no. 1, pp. 2232–2235, 2014.

[19] C. Jian, Y. Bin, J. Hua, Z. Lei, and T. Li, "Interactive image segmentation by improved maximal similarity based region merging," *2013 IEEE Int. Conf. Med. Imaging Phys. Eng.*, no. c, pp. 279–282, 2013.

[20] E. Golkar, A. A. Abd. Rahni, and R. Sulaiman, "Comparison of Image Registration Similarity Measures for an Abdominal Organ Segmentation Framework", *Conference on Biomedical Engineering and Sciences, 8-10 December, Miri, Sarawak, Malaysia, 2014 IEEE*, pp. 442-445.

[21] E. Fida, J. Baber, M. Bakhtyar, and M. J. Iqbal, "Automatic Image Segmentation Based on Maximal Similarity Based Region Merging," *Digit. Image Comput. Tech. Appl. (DICTA), 2015 Int. Conf.*, pp. 1–8, 2015.

[22] W. Song, M. Li, P. Zhang, Y. Wu, L. Jia, and L. An, "Unsupervised PolSAR Image Classification and Segmentation Using Dirichlet Process Mixture Model and Markov Random Fields With Similarity Measure" pp. 1–13, 2017.

[23] WK Yeo, DFW Yap, KC Lim, DP Andito, MK Suaidi, TH Oh, "A feedforward neural network compression with near to lossless image quality and lossy compression ratio" IEEE Student Conference on Research and Development (SCOReD), pp. 90-94, 2010.