792
doi:10.1017/S1431927613005953
Microsc. Microanal. 19 (Suppl 2), 2013

SEM Real-time Image Processing using a GPGPU approach.

N.H.M. Caldwell[1], Y Lei[1], B.C. Breton[1] and D.M. Holburn[1].

[1.] Department of Engineering, University of Cambridge, Trumpington Street, Cambridge CB2 1PZ, United Kingdom

In our prior work, we reported on the first of a series of projects utilizing GPGPUs (General Processing Graphics Processing Units) to improve instrument performance by harnessing the computational capabilities of the tens to thousands of processor cores available in certain graphics cards [1]. This paper reports on another project within the program, where the target application is a subset of real-time image processing. The ideal GPGPU applications involve tasks where the same code must be executed on many data items. SEM image processing is a very good fit. The large digital images generated by SEMs provide worthwhile quantities of data; most image processing functions can be reduced to repeated execution of code across (blocks of) data points in two-dimensional arrays.

The microscope employed for development and testing of this work was a Carl Zeiss 1430VP SEM, which was retrofitted with a low-specification NVIDIA GeForce 8600 GT graphics card with only 32 processor cores. Access to images direct from the SEM, both locally and across a network, was enabled through Carl Zeiss' Application Programming Interface to the SmartSEM™ software. The experimental software was designed and implemented with a combination of the CUDA Toolkit (version 4.x), the NVIDIA GPU Computing SDK (version 4.x), and Microsoft Visual Studio 2008. The NVIDIA 4.x NPP library was also used as this contained a pre-existing set of GPU-accelerated image, video and signal processing functions. In addition, a desktop computer equipped with a NVIDIA GeForce GTX 570 graphics cards (480 processor cores) provided an opportunity to test higher performance ranges. For comparison purposes, Matlab and its Image Processing toolbox, which are both highly optimized for matrix operations and signal processing operations, were used to provide time benchmarks. ImageJ was also used to check the validity of the results from NPP library functions and new code.

Four image processing functions were implemented: thresholding, morphological operations (dilation and erosion), image labelling and Euclidean distance maps. Binary global thresholding (mapping an entire image to two target grey levels according to a single threshold) can be rapidly performed by CPU-only code. The NPP library provided a modest performance gain through computing the image grey level histogram necessary to compute the right threshold. A disadvantage of global thresholding is susceptibility to non-uniform illumination of an image. Local thresholding can avoid this – the scheme used was to convolve the image with a box average filter, subtract the averaged from the original and threshold the difference against a fixed value, thus comparing each pixel against a local value. Multi-level thresholding, e.g. segmenting an image into more than two grey levels, was also implemented. A final option allowed the resulting thresholded image to be rendered in various colours for easier identification of the regions of interest.

For image labelling, a parallelised algorithm for 4-connected pixels was developed to mark connected segments. The image is divided into blocks of pixels. Upon each block, an efficient two-pass sequential procedure is then employed to identify possible labels. The first pass scans the block, from left to right, top to bottom, with each pixel taking its label from its left or top, whichever is valid. If labels differ, the smaller is stored in the lookup table at the position of the larger. The second pass replaces labels with the

equivalents from the lookup table. Then the individual blocks are stitched together and lookup tables resolved in parallel, before a final parallel pass resolves all pixels to their final labels.

Two algorithms were developed for Euclidean distance mapping (a technique used to underpin further image analysis methods such as watershed segmentation and skeletonisation). The first was a naïve progressive search with order N-squared complexity; the second was based on the method of Kolountzakis and Kutulakos [2]. The latter method enables rows and columns of an image to be considered independently, so a CUDA thread can be assigned to a single row, then a single column. For N parallel processors, its complexity could be reduced from N log N to log N. On the SEM's lower specification graphics, the better algorithm was necessary to perform the mapping smoothly in real time.

A basic Microsoft Foundation Class utilising the Application Programming Interface was supplied by Carl Zeiss and modified to include implementations and controls for the new algorithms. OpenGL was used to render the results to the display. The prototype was able to demonstrate real-time processing of images onboard the SEM with the resulting images updating as the specimen stage was moved (see Figure 1(left). Figure 1 (right) shows the performance increases of GPGPU algorithms normalised against MATLAB for 4 square image sizes – note that GPU overheads mean that labelling is faster using MATLAB for small image. Many additional image processing and analysis tasks could benefit from a GPGPU approach and this will be a focus of our future research [3].

References:

[1] N.H.M. Caldwell *et al*, Microsc. Microanal. **18 (Suppl S2)** (2012), p. 1210.
[2] M.N. Kolountzakis and K.N. Kutulakos, Information Processing Letters **43** (1992), p. 181.
[3] NVIDIA GeForce and CUDA are trademarks of NVIDIA Corporation. SmartSEM™ is a trademark of Carl Zeiss Microscopy. This research was supported by funding from Carl Zeiss Microscopy. The authors gratefully acknowledge the assistance of Carl Zeiss personnel, especially Daniel Aldridge, David Hubbard and Stewart Bean.

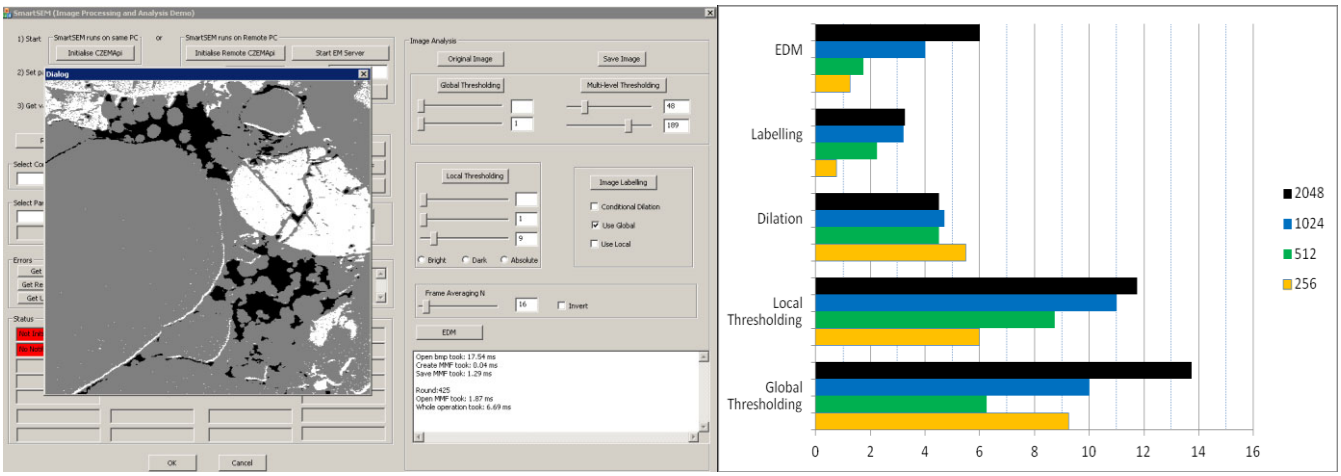

Figure 1: (Left) The software after performing a 3-level thresholding of a Corrie sandstone specimen. (Right) Performance comparisons of GPGPU algorithms relative versus MATLAB (which was normalized at x1)

https://doi.org/10.1017/S1431927613005953 Published online by Cambridge University Press