Maximum subarray algorithms for use in astronomical imaging

S. J. Weddell[a], T. Read[a], M. Thaher[b], and T. Takaoka[b],

[a]Department of Electrical & Computer Engineering, University of Canterbury, Christchurch, New Zealand;

[b]Department of Computer Science & Software Engineering, University of Canterbury, Christchurch, New Zealand

ABSTRACT

The maximum subarray problem is used to identify the subarray of a two dimensional array, where the sum of elements is maximized. In terms of image processing, the solution has been used to find the brightest region within an image. Two parallel algorithms of the maximum subarray problem solve this problem in $O(n)$ and $O(\log n)$ time. A field programmable gate array implementation has verified theoretical maximum performance, however extensive customisation is required restricting general application. A more convenient platform for this work is a graphics processor unit, since it offers a flexible trade-off between hardware customization and performance. Implementation of the maximum subarray algorithm on a graphics processor unit is discussed in this article for rectangular solutions and convex extensions are explored.

Keywords: maximum subarray problem, astronomical image processing, graphics processing unit

1. INTRODUCTION

Efficient algorithms applied to 2D (image) data are of significance to meet real-time performance demands of today's mixed signal technologies. High sample-rates supported by state-of-the-art radio, optical, an X-ray sensor technology often exceed signal processing capabilities of computer hardware. Thus, the adaptation of traditional image processing algorithms to meet these demands, particularly in research fields such as astronomical and medical imaging, is of paramount importance.

A commonly used image processing function is *blob* detection.[1] Blob detection is a generalized term used in computer vision that encompass a variety of methods to detect points or regions within an image that vary from neighbouring regions, such as in brightness or color. Extensions to the maximum subarray problem (MSP) over

Further author information: (Send correspondence to Steve Weddell.)

Steve Weddell.: E-mail: steve.weddell@canterbury.ac.nz, Telephone: 64 3 364 2987

either rectangular, and more recently, convex regions, employing the maximum convex sum problem (MCSP), provide an alternative approach to existing methods of blob detection. However, the ease in which such methods can be supported as a mesh structure, directly implementable in parallel hardware configurations for significant speed-up, is of prime consideration and forms the basis of this article.

The organization of this article is as follows. Section 2 provides a descriptive background and formal definition of the maximum subarray problem (MSP), including K-maximum and convex sum extensions. In Section 3 an overview of suitable hardware platforms for the implementation of these algorithms is given. Section 4 details an image processing application that is suitable for MSP implementation on a general purpose graphical processor unit (GPU). The results of this implementation are presented in Section 5. Lastly, Section 6 concludes with a summary of this article and future work is described.

2. THE MAXIMUM SUBARRAY PROBLEM

The maximum subarray problem (MSP) can be defined as follows. Consider a collection of 1D or 2D array elements, where the selection of a segment of consecutive array elements will form the largest possible sum of all possibilities over the given array. An efficient method is required to return the coordinates, i.e., the location of the maximum subarray. In terms of a 2D array used to represent digital images, the upper bound based on serial processing is cubic or near cubic time.[2]

For example, let a two-dimensional array, $a[1 \cdots m, 1 \cdots n]$, be given as input, then the maximum subarray problem is to maximize the array portion $a[k \cdots i, \ell \cdots j]$ and to return the indices (k, ℓ) and (i, j) representing the top-left and bottom-right positions of the rectangular maximum sum. Let a be defined as

$$\begin{pmatrix} -1 & 2 & -3 & 5 & -4 & -8 & 3 & -3 \\ 2 & -4 & -6 & -8 & 2 & -5 & 4 & 1 \\ 3 & -2 & 9 & -9 & |3 & 6| & -5 & 2 \\ 1 & -3 & 5 & -7 & |8 & -2| & 2 & -6 \end{pmatrix}$$

then the maximum sum, s= 15, is given by the rectangle defined by the upper left corner $(3, 5)$ and the lower right corner $(4, 6)$, as shown by the vertical bars.

The maximum subarray problem was originated by J. Bentley in his *Programming Pearls* of September 1984.[3] An iterative approach was taken by Bentley using Kardane's method on extended rows to find the maximum sub-sequence within a 2D array and was of $O(m^2n)$ time complexity for an $m \times n$ array ($m \leq n$), which is cubic when $m = n$.

A fundamental requirement for the correct operation of these algorithms is that data are represented either as signed integer or floating point, i.e., single or double precision, IEEE 754, such that the array contains both

positive and negative values[3] . For example, if the array contained only unsigned integers the maximum sum would be the entire array. Similarly, if all elements have negative values, the solution is an empty subsequence.

2.1 Parallel 2D prefix-sum method

The parallel prefix sum method employs a list of prefix sums to determine the maximum subarray. Each prefix "sum", s, is updated progressively as both row, i, and column, j, indices iterate through a 2D array, a$[i, j]$. While the prefix sum is accumulated, the minimum of the preceding prefix sums is also maintained. By subtracting the minimum prefix sum from the prefix sum, a candidate is found which may be the solution, "sum", for the maximum subsequence problem. Two registers, $h(\cdot)$ and $g(\cdot)$, act as array pointers and are continuously updated to provide the *scope* of row-wise prefix sum, r$(\cdot)$, and vertical accumulation of the row-wise prefix sum, s$(\cdot)$, respectively. An schematic that shows a graphical representation of the prefix-sum method is shown in Figure 1.

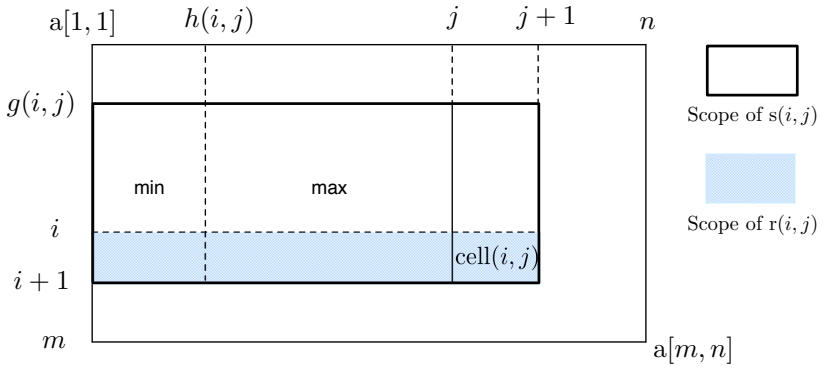

Figure 1. The Prefix Sum Method showing the scopes of r(i, j) and s(i, j), and scope registers, $g(i, j)$ and $h(i, j)$.[4]

The location of the maximum subarray found at each iteration of i and j within the region $[1, \cdots, i; 1, \cdots, j]$ is stored in row and column registers, $(i_1, j_1)|(i_2, j_2)$. These registers represent the top-left and bottom-right location of the maximum subarray, respectively. A formal proof of the maximum subarray problem is presented in the Appendix.

2.2 Parallel 2D mesh-Kadane method

The implementation of the mesh algorithm for a 2D matrix is similar to the prefix sum method. Both algorithms are mesh-based, i.e., a processor and control unit are allocated to each cell; for an image processing application each cell represents a single pixel. Thus, a fine-grained architecture is defined where each cell receives data from above and from the left neighboring cell. A control unit is used to synchronise the transfer of data from neighboring cells. For example, cell $(2, 2)$, as shown in Figure 2, passes to cells $(3, 2)$ and $(2, 3)$ the partial maximum sum, s, of the group of cells directly above and to the left, and the location of the maximum sum

region. After each iteration a diagonal of updated cells propagate from the top left towards the bottom right of the array. A special case is considered for boundary cells. A detail of Bae's original architecture, and based on a systolic array of cells, is shown in Figure 2.

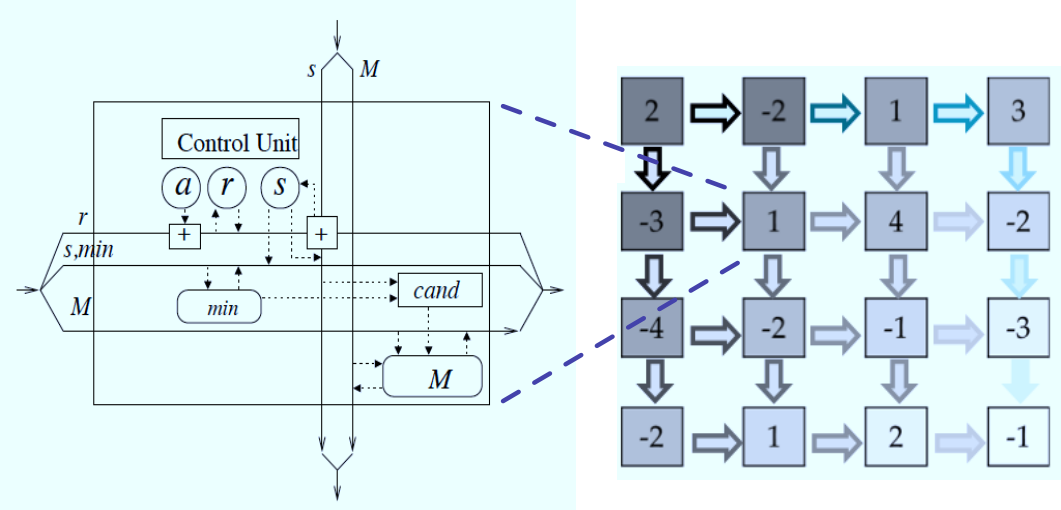

Figure 2. Structural composition for a single cell entity is detailed on the left of this figure, showing the control unit and data path dependencies that are iteratively updated from the top-left to the bottom right of the systolic 2D array shown on the right.[4]

The mesh-Kadane algorithm was designed for a parallel programming environment and therefore is suitable for either GPU implementation or custom VSLI programmable logic. As with the prefix sum, both algorithms are heavily dependent on computer hardware resources; this is discussed in Subections 6.2 and 6.3. A detailed discussion of both serial and parallel variants of the MSP is provided by Bae.[4]

2.3 Formal definition of the mesh algorithm for the maximum subarray problem

In this section we prove the correctness of the algorithm using an extended Hoare logic. Our assertion involves three indices (i, j, k), and several arrays indexed by i, j and k. The algorithm has a sequential structure of the outermost loop, by k. Inside this loop, there is a parallel control structure for all i and j.

In Figure 3, at cell(i, j), a$[i][j]$ is added to row r if $j \leq k$ for all i and j in parallel. Arrays "max", "min", s, and "solution" are similarly updated in parallel. In the following description of the mesh algorithm, max$[i][j]$ is effectively floating, i.e., the left-boundary and right boundary are not fixed. We set up the following six assertions, and prove they are invariants under the parallel execution of the mesh connected cells.

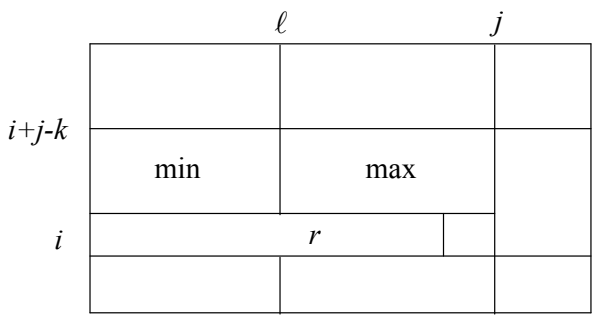

Figure 3. Simplified schematic of Figure 1.

for i =$1, \cdots, n$ and $j = 1, \cdots, k$

P1 r$[i][j]$ is the row sum at (i, j), that is, the sum of $a[i][1, \cdots, j]$.

P2 s$[i][j]$ is the sum of a$[i + j - k, \cdots, i][1, \cdots, j]$.

P3 min$[i][j]$ is the minimum of s$[i][\ell], \ell = 1 \cdots, j$.

P4 max$[i][j]$ is the maximum of the sum of a$[i + j - k, \cdots, i][\ell, \cdots, \ell']$, $1 \leq \ell \leq \ell' \leq j$.

P5 solution$[i][j]$ is the maximum sum of a$[i + j - k, \cdots, i][1, \cdots, j]$.

C1 control$[i][k] = 1$ for $j = 1, \cdots, k$ and control$[i][j] = 0$ for $j = k = 1, \cdots, n$.

k is a time index and (i, j) are space indices. All assertions are true for $k = 0$.

Our goal is obtained from P5 at the end of $k = n$.

Algorithm

cell(i, j) performs the following identical program in a synchronized way. Each cell(i, j) is aware of its indices, or position, (i, j). The program is written in C-like pseudo code.

Initialization

for all i, j **between** 0 and n **do in parallel**

  r$[i][j] = 0$; min$[i][j] = 0$; max$[i][j] = -\infty$; control$[i][j] = 0$; solution$[i][j] = -\infty$;

end

for all i **do in parallel** control$[i][0] = 1$;

Main

for $k = 1$ to $2n - 1$ **do**

  for all i, j **between** 1 and n **do in parallel**

    if control $[i][j-1] == 1$ **then**

      r$[i][j]$ = r$[i][j-1]$ + a$[i][j]$;

      s$[i][j]$ = s$[i-1][j]$ + r$[i][j]$;

      min$[i][j]$ = maximum(min$[i][j-1]$, s$[i][j]$);

      max$[i][j]$ = maximum(s$[i][j]$ − min$[i][j]$, max$[i][j-1]$);

      solution$[i][j]$ = maximum(solution$[i][j]$, solution$[i-1][j]$, solution$[i][j-1]$, max$[i][j]$)

      control$[i][j] = 1$;

    end

  end

end

Proof rules are given in the Appendix.

2.4 Generalization of the maximum subarray problem

Generalizing the mesh algorithm outlined in Subsection 2.2 requires determination of coordinates for the 2nd-, 3rd-, $\cdots$, and K-maximum sum. When the subarray is disjoint, this problem is easily solved by setting each cell containing the maximum subarray to $-\infty$. However, the Kadane algorithm was found not suitable for determining overlapping K-maximum sums. Thus, the *prefix sum* method was developed for such requirements.[5]

Bae originally outlined 43 versions of the maximum subarray solution in the form of 43 algorithms; these are listed from Algorithm 1 to Algorithm 43.[2] The work presented here focuses on four of these algorithms. A description of these algorithms is listed in Table 1.

Table 1. Maximum subarray algorithms considered for hardware implementation.

Classification	Configuration	Description
Algorithm 8	Serial	2D Prefix Sum Method
Algorithm 9	Serial	2D Kadane Method
Algorithm 38	Parallel	2D Prefix Sum Method
Algorithm 39	Parallel	2D Mesh-Kadane Method

In this article we are primarily concerned with parallel implementations of the MSP, i.e., Algorithm 38 and 39, and their suitability for application to astronomical imaging problems. For consistency, the K-maximum sum extension is not considered, i.e., the location of only the brightest portion of an image or sub-image is compared using the four aforementioned algorithms.

3. COMPUTER HARDWARE CONSIDERATIONS

In this section two hardware platforms used to implement MSP parallel algorithms are discussed. An overview of field programmable gate array (FPGA) is firstly given and this is followed by a discussion on GPU computer hardware.

3.1 Field programmable gate arrays

Field programmable gate arrays (FPGAs) are commonly used for specialised hardware functionality and comprise a fabric of combinational and sequential logic, configurable logic blocks (CLB), and block RAM modules. Input Output Blocks (IOBs) provide an interface to external devices. FPGAs have been used to implement concurrent hardware in the form of systolic processor arrays. In this regard they are ideal as a testbed for the implementation of mesh algorithms such as the MSP.

Our initial development used the XC2VP2 device from the Virtex-II Pro series of Xilinx FPGAs to implement the maximum subarray algorithm. This device provided a total of 1,408 slices and utilised 352 CLBs, 216Kb of block RAM and 204 IOBs.[6]

3.2 Graphics processing units

Graphics cards work by having a number of processing units arranged in parallel. Each processing unit is capable of executing a small instruction set. By itself, a single core of a GPU performs slower than a CPU of similar cost. By performing many concurrent operations on a GPU, performance can be increased to exceed that of a CPU.

For a GPU to be more effective than a CPU, as many cores as possible must be utilized. This means that code running on a GPU takes a different form to code targeted for a CPU. GPUs are available on many computers, but often are only used by very specific programs that require graphical processing. This is due to the historic lack of parallel applications, and somewhat related to this, the increased difficulty of programming in parallel. The aim of the latest NVidia graphics cards is to make coding a GPU easier and effective on all platforms.

4. MAXIMUM SUBARRAY APPLICATIONS

Very large scale integration (VLSI) has been used to implement the Shack Hartmann wavefront sensor (SHWS), where wavefront slope is determined iteratively by subdivision of the pupil into zones.[7] However, more efficient VLSI implementations employing the MSP have been used as an alternative to centroiding algorithms for optical wavefront sensing.[6,8] To accelerate processing, centroids of each zone are calculated concurrently using a field programmable gate array (FPGA), where $O(n)$ performance was achieved for small $n \times n$ regions.

Attention has recently been drawn to the need for an efficient implementation to determine the maximum subarray regions of simulated radio telescope images developed for use in the Australian square kilometer array pathfinder (ASKAP) project.[9] A convenient platform for this work is a graphics processor unit (GPU) and our adaptation of the maximum subarray algorithm has shown significant performance gains with an efficient implementation of two parallel algorithms introduced in Subsection 2.4.

In this section the maximum sum algorithm was implemented on a GPU to determine the location of the brightest region (sub-array) from a simulated, radio telescope image. Processing times were reported for various sized images. Two parallel algorithms, suitable for GPU execution, were benchmarked against two serial algorithms that were run on a CPU core.

4.1 GPU Image processing for radio astronomy

The Australian square kilometer array pathfinder (ASKAP) project is a major consortium to develop 36 antennas, each of 12m diameter, in Murchison Western Australia with a baseline of 6 km. The aim of the project is to instantaneously view 30 square degrees of sky and planned operation is 2013. Seven working groups have been established, the first of which (WG 1) is responsible for simulations and imaging. As part of a wider collaboration with New Zealand, the ANZSKA consortium has been established and research on image extraction is being conducted using the maximum sum algorithm as an efficient method to classify source objects.

Data, comprising a simulated sky with added complex bright sources and large-scale diffuse sources, as observed by an ASKAP telescope using models supplied by EMU WG1,[9] were used to test the performance of the MSP using the four algorithms listed in Table 1. Due to memory constraints imposed by the GPU used for this evaluation (GTX 480), the size of the original 6144×6144 image, representing the sky model with source objects for a full continuum observation, was cropped to a maximum size of 4000×4000 pixels.

Figure 4 highlights four regions of interest (ROI) of this sky model. Due to the wide dynamic range of data, four subfigures are used to show underlying structure. Subfigure (a) highlights two bright source objects, the brightest of which was found by each of the four maximum sum subarray algorithms described in Subsection 2.4. To highlight structure within the Continuum, void of 'bright' sources, Subfigure (b) shows the fluctuation

in background intensity. Also shown is the location of the ROI given in Subfigure (c), where the underlaying structure adjacent to the brightest (darkest for inverted) source object is shown. Subfigure (d) shows the intensity variation between a bright object in the top-left of the frame within the Continuum.

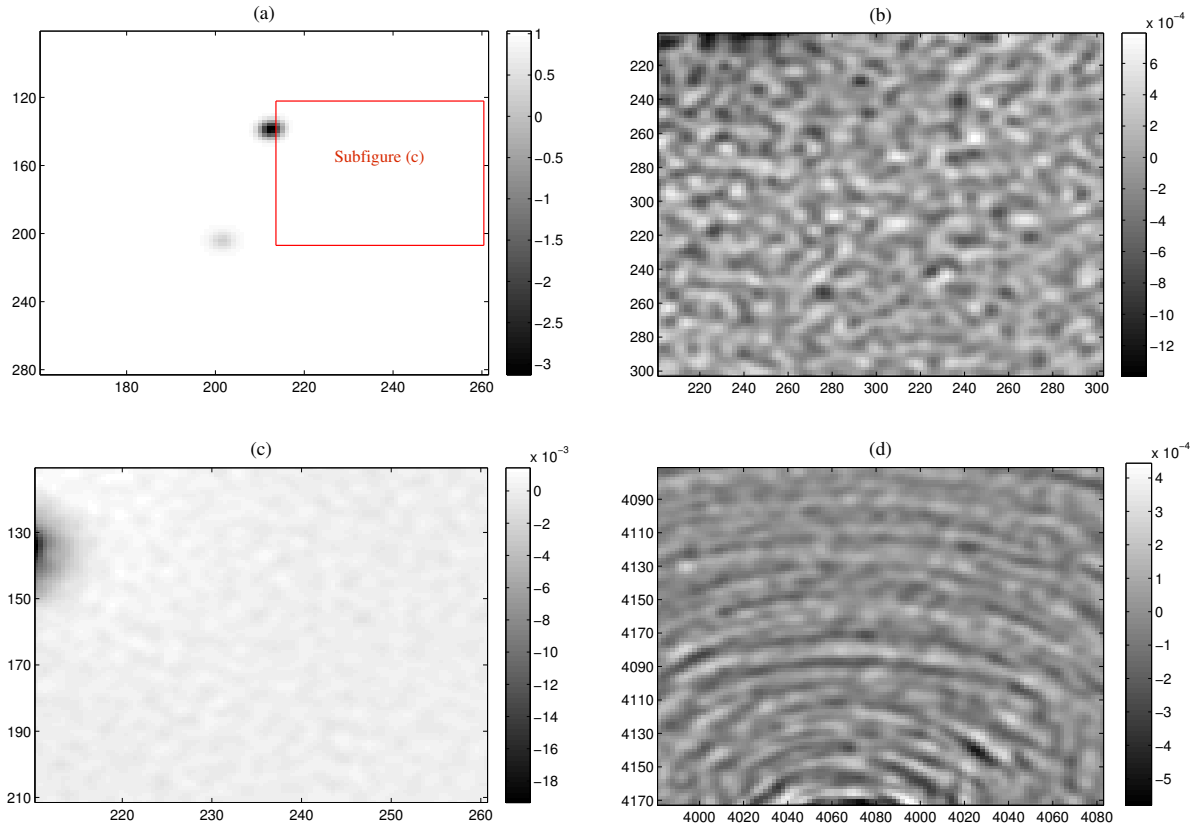

Figure 4. ASKAP simulation of a full continuum observation with added complex bright sources and large-scale diffuse sources using models supplied by EMU WG1:[9] (a) inverted image of two bright sources, where the upper-central object is the maximum subarray of the continuum image, and the rectangle shown is the relative position of subfigure (c); (b) image showing typical structure of a relatively uniform portion of the sky, void of source objects; (c) the highlighted region of Subfigure (a), showing high contrast due to wide dynamic range of data; (d) intensity variations and elliptical structure nearing a bright object in the lower portion of the image.

GPU implementation of the MSP algorithm was achieved using the compute unified device architecture (CUDA). CUDA is a programming language developed for NVidia graphics cards.[10] It is an extension on C and it facilitates parallel programming by using a Kernel to separate CPU operations, GPU serial operations, and GPU parallel operations. Code contained within the Kernel is distributed to a specified number of thread blocks, where each thread block executes the Kernel code concurrently.[11] Multiple copies of each Kernel can be spread

to each thread block by specifying block 'height' and block 'width', and in this way operations can be pipelined. In fact, later models of CUDA GPUs claim to achieve near perfect pipelining when handling a sufficient number of threads.

When the number of thread blocks exceeds the number of logical processors, the thread blocks are queued. Since each thread block can be executed independently of other thread blocks, the number of active thread blocks and the width of the queue can be determined on the fly to suit any graphics card configuration. Operations are also pipelined. With a sufficient number of queued operations, pipelining can become optimally efficient.[11]

4.2 Preprocessing

The current CUDA MSP code can only process images to a maximum of 4000×4000 pixels. This is limited by the number of simultaneous threads supported by the GTX480 (approximately 16 million). There are two alternatives for processing images larger than 4000×4000 pixels. The first is to separate images into multiple overlapping blocks, process each image independently and compare the results. This assumes that the maximum subarray does not extend past the common regions of two sub-blocks.

The second method is to scale the image to 4000×4000 pixels. However, information is lost in this process. The amount of information loss depends on the original size of the image. The suitability of this approach also depends on the data being used. When a region of interest is known *a priori*, cropping an image can be performed as an alternative to scaling.

5. RESULTS

The MSP algorithm was implemented on a GPU and tested using the ASKA simulated telescope Continuum image discussed in Section 4.1. Due to the wide dynamic range of image data, double precision floating point operations were employed. The Continuum image served as the basis for test data, and ROIs of size $20 \times 20, 50 \times 50, 100 \times 100, \cdots, 2500 \times 2500$ and 4000×4000 were generated. The purpose of these tests was to compare the speed between different mesh algorithms over varying memory sizes for both CPU and GPU platforms. The tests can be summarized as follows:

- Algorithms 38 and 39 compiled with CUDA and executed using a GTX480 GPU.
- Algorithms 8 and 9 compiled with GCC and executed using an Intel i7 2.6 GHz CPU.

The results of these tests are shown in Figure 5.

Firstly, these results in Figure 5 show up to a $5\times$ improvement in speedup using either parallel algorithm 38 or 39 on a GPU, over either serial algorithm 8 or 9 executed on a CPU for image sizes in excess of 500×500 pixels.

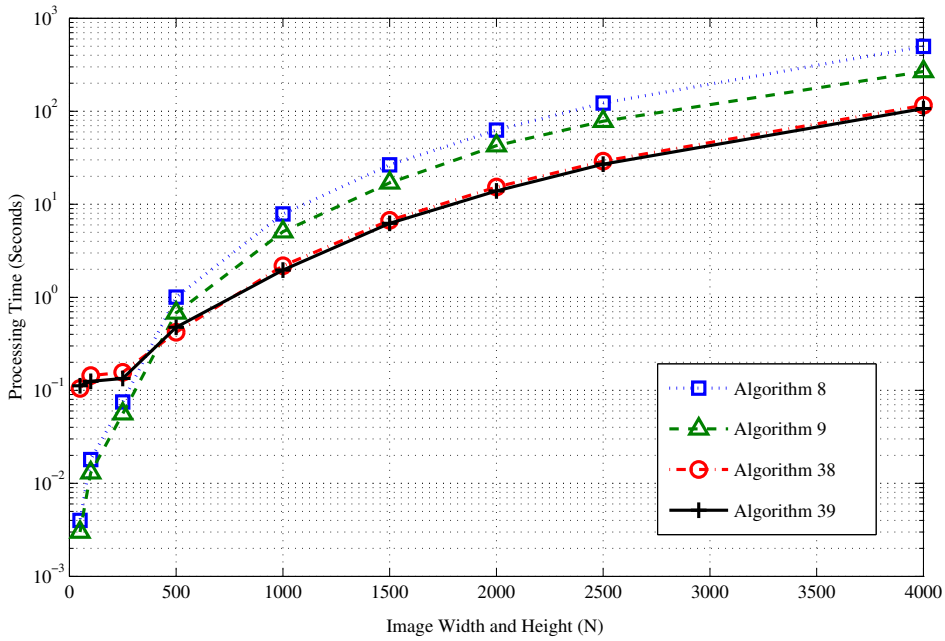

Figure 5. Test results and comparison from GPU compilations of the prefix-sum and mesh algorithms, 38 and 39 respectively, and CPU compilation employing equivalent serial algorithms, 8 and 9, respectively.

Secondly, the difference between either mesh algorithm was minimal. Since the coding of both algorithms is very similar, and memory utilization was identical, this was to be expected. Lastly, the CPU was actually faster for small images compared to the GPU. This is attributed to inefficiencies exhibited by loading GPU memory from main memory for relatively small images.

6. CONCLUSION AND FUTURE WORK

In this article we have described a selection of four efficient algorithms that can be used to solve the maximum subarray problem. These MSP algorithms comprised two serial and two parallel approaches and were applied a set of radio telescope images as a computationally efficient method to determine the brightest region within an image and return the coordinates of this region. The systolic structure of both parallel mesh and prefix-sum algorithms are well suited for implementation on low-cost hardware platforms, such as general purpose graphical processor units. A mathematical proof of the parallel 2D mesh algorithm provides insight into the maximum subarray problem and is used as a basis for further adaptation. These proof rules are detailed in the Appendix.

GPU results were compared with a serial algorithm, compiled and processed on an i7 CPU, and an $5\times$ speedup was reported. Further development of both GPU and FPGA platforms are required to reach a compromise between computational efficiency that was demonstrated using VLSI custom hardware, and the relative simplify

when implementing the MSP on GPUs with minimal hardware constraints. Such extensions will be the result of future work and are described in the remainder of this section.

6.1 Maximum convex sum problem

An extension to the MSP is the Maximum Convex Sum Problem (MCSP), where we generalize the maximum subarray problem to the Maximum Convex Sum Problem (MCSP) by using dynamic programming approach. That is, the shape is not restricted to a rectangle. The convex shape is a more flexible shape to cover various data distribution. The convex shape is defined as a shape that has a centre column linked with"W" and "N" shapes; such a column is called an anchor column,[12–14] as shown in Figure 6. Other researchers call the shape a rectilinear convex shape.[15] In this study we will use the concept of the convex shape, which is not following the geometrical convex shape definition.[12,13] A"W" shape can be described as a region with a top contour inclining or remaining horizontal and a bottom contour declining or remaining flat from left to right,[12] whereas the "N" Shape is a mirror image of the "W".[12,13] The combination of the "W" and "N" shapes is defined as the convex shape for our descriptive purposes. In terms of this article, where two applications have been discussed, the MCSP is more appropriate due to the convex shapes that astronomical objects typically represent.

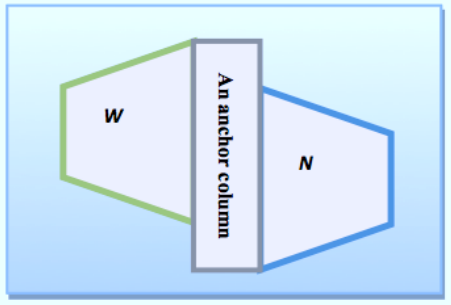

Figure 6. W-N shape (Convex shape).

By our experiments, the convex sum has about twice as much gain over rectangular shapes,[12] Thus, this problem can solve the brightest spot problem more accurately. Our convex sum algorithms simplified Fukuda, *et al.*[15] to solve the maximum convex sum problem in $O(n^3)$ time, when $m = n$, based on dynamic programming. The proof of the simplification is discussed in.[12] We extended the simplified algorithm to the K-Maximum Convex Sum Problem (K-MCSP) with time complexity of $O(kn^3)$ for the disjoint case,[12] whereas the current time complexity for the overlap case is $O(n^3 + kn^2)$.[14] Our final target will be to achieve a parallel algorithm for the K-maximum convex sum problem with $O(n)$ time.

6.2 Optimizing hardware for improved utilization

Solving a complex problem such as the MSP in $O(n)$ time is possible by implementing a mesh algorithm, however it comes at a cost. The demands on complex hardware modules, i.e., adders, comparators, and registers, are such that even a small image can consume the entire resources of programmable logic devices, such as FPGAs, even for relatively small (100×100) array sizes. Simply migrating to a larger FPGA when available is not a solution. Optimizing hardware resources through better interpretation of the hardware description language, in terms of its behavioral definition, is required to meet these demands. Currently, work is continuing on the use of FPGAs that contain up to 2M logic cells. We are also exploring the possibility of cascading processing over multiple devices, essentially resulting in 2D mosaic of VLSI devices.

6.3 GPU memory transfers

For GPU implementation, algorithms 38 and 39 are limited in performance and image size by the memory transfer between thread blocks. These algorithms could be modified to facilitate different thread block layouts, to improve performance , image size, or both. Due to the thread-block limit on the GTX480, image size is limited to just over 4000×4000 pixel images. By arranging the cells so that more than a single cell is computed by each thread, the maximum allowable image size could be increased. Another benefit is that cells that share a thread can transfer data via thread registers, as opposed to relatively slow global GPU memory. To accommodate this change, the algorithm itself would increase in complexity, and therefore the computation time for each cell to complete one step may increase. The trade-off between memory-transfer time and raw computation time is currently under investigation, as well as the efficiency of different cell-to-thread configurations.

7. APPENDIX

Proof rules are given below. For simplicity, cells are described in a one-dimensional manner. We assume the corresponding statement in each cell is executed in a synchronized way. Some of Hoares axioms[16] are extended to concurrent processes below.

Parallel assignment

Let $x_1, \cdots, x_n$ be local variables in cell $1, \cdots$, cell n.

Let ass(i) be defined by an assignment statement $x_i = e_i$. The expression e_i can include variables in other cells.

$$\{\mathrm{P}[e_1/x_1, \cdots, e_n/x_n]\}\ \mathrm{ass}(1) \parallel \cdots \parallel \mathrm{ass}(n)\ \{\mathrm{P}\}$$

$\mathrm{P}[e_1/x_1, \cdots, e_n/x_n]$ is a simultaneous substitution, that is, $e_1, \cdots, e_n$ are independently substituted for $x_1, \cdots, x_n$.

Parallel if - then statement: Let "if B_i then S_i" be a corresponding `if - then` Statement in cell(i).

$$\frac{\textbf{for } i = 1, \cdots, n\ \{\mathrm{B}_i \wedge \mathrm{P}\}\mathrm{S}_i\{\mathrm{Q}\}, \mathrm{P} \wedge \sim \mathrm{B}_i \Rightarrow \mathrm{Q}}{\{\mathrm{P}\}\ \text{if } \mathrm{B}_1 \text{ then } \mathrm{S}_1 \parallel \cdots \parallel \text{if } \mathrm{B}_n \text{ then } \mathrm{S}_n\ \{\mathrm{Q}\}}$$

Parallel execution: Let $\mathrm{S}_i[\ell]$ be the ℓ-th statement of m statements in cell i

$$\frac{\textbf{for } \ell = 1, \cdots, m\ \{\mathrm{P}[\ell - 1]\}\ \mathrm{S}_1[\ell] \parallel \cdots \parallel \mathrm{S}_n[\ell]\{\mathrm{P}[\ell]\}}{\{\mathrm{P}[0]\}\ \text{do in parallel } [\mathrm{cell}(1) \parallel \cdots \parallel \mathrm{cell}(n)]\ \{\mathrm{P}[m]\}}$$

For rule

$$\frac{\mathrm{P}(0),\ \{\mathrm{P}(k-1)\}\ \mathrm{S}\ \{\mathrm{P}(k)\}}{\{\mathrm{P}(0)\}\ \text{for } k = 1 \text{ to } n \text{ do S}\ \{\mathrm{P}(n)\}}$$

We index our assertions with the time index k in the following:

At the beginning of the kth iteration, C1$(k-1)$ holds.
C1$(k-1)$ $\Leftrightarrow$ control$[i][j] = 1$ for $j = 1, \cdots, k-1$ and control$[i][j] = 0$ for $j = k, \cdots, n$.

Proof for C1: If $j \leq k$, $\{\text{control}[i][j-1] = 1\}$ cell(i,j) $\{\text{control}[i][k] = 1\}$
If $j > k$, $\{\text{control}[i][j-1] = 0\}$ cell(i,j) $\{\text{control}[i][k] = 0\}$.
Thus $\{\text{C1}(k-1)$ cell$(1,1) \parallel \cdots \parallel$ cell$(i,j), \cdots \parallel$cell(n,n) $\{\text{C1}(k)\}$.

Proof for P1: P1(0) is true. Suppose P1$(k-1)$ is true.
r$[i][j]$ = r$[i][j-1]$ + a$[i][j]$ is performed for $j = 1, \cdots, k$ in parallel. Then P1(k) is true.

Proof for P2: P1(0) is true. Suppose P2$(k-1)$ is true.
s$[i][j]$ = s$[i-1][j]$ + r$[i][j]$ is performed for $j = 1, \cdots, k$ in parallel. Then P2(k) is true.

Proof for P3: P3(0) is true. Suppose P3$(k-1)$ is true.
min$[i][j]$ = minimum(min$[i][j-1]$, s$[i][j]$) is performed for $j = 1, \cdots, k$ in parallel. Then P3(k) is true.

Proof for P4: P4(0) is true. Suppose P4$(k-1)$ is true.
max$[i][j]$ = maximum(max$[i][j-1]$, s$[i][j]$ − min$[i][j]$) is performed for $j = 1, \cdots, k$ in parallel. Then P4(k) is true.

Proof for P5: P5(0) is true. Suppose P5$(k-1)$ is true.
solution$[i][j]$ = maximum(solution$[i][j]$, solution$[i][j-1]$, solution$[i-1][j]$, max$[i][j]$) is performed for
$j = 1, \cdots, k$ in parallel. Then P5(k) is true.

P5$(2n-1)$ at (n,n) is : solution$[n][n]$ is the maximum sum in a$[1, \cdots, n][1, \cdots, n]$.

Extended proof of $\{\mathrm{P}(k-1)\}\ \mathrm{S}\{\mathrm{P}(k)\}$

Note that in the following proof, "At time k" means "at the end of the k-th iteration".

P1: At time $k-1$, r$[i][j-1]$ is the row sum of a$[i][1,\cdots,j]$ for $j=1,\cdots,k-1$.

At time k, r$[i][j]=r[i][j-1]+$ a$[i][j]$ is performed for $j=1,\cdots,k$.

Thus r$[i][j]$ is the row sum of a$[i][1,\cdots,j]$ for $j=1,...,k$.

P2: At time $k-1$, s$[i-1][j]$ is the sum of a$[i-1+j-(k-1),\cdots,i-1][1,\cdots,j]=$ a$[i+j-k,\cdots,i-1][1,\cdots,j]$.

At time k, s$[i][j]=$ s$[i-1][j]+$ r$[i][j]$ is performed.

Thus s$[i][j]$ is the sum of a$[i+j-k,\cdots,i][1,\cdots,j]$.

P3: At time $k-1$, min$[i][j-1]$ is the minimum of s$[i][\ell,\cdots,j-1],\ell=1,\cdots,j-1$.

At time k, min$[i][j]=$ minimum(min$[i][j-1]$, s$[i][j]$) is performed.

Thus min$[i][j]$ is the minimum of s$[i][\ell,\cdots,j],\ell=1,\cdots,j$.

P4: At time $k-1$, max$[i][j-1]$ is the maximum of the sum of a$[i+j-1-(k-1),\cdots,i][\ell,\cdots,\ell']=$
a$[i+j-k,\cdots,i][\ell,\cdots,\ell']$ for $1\le\ell\le\ell'\le j-1$.

At time k, max$[i][j]=$ maximum(s$[i][j]-$ min$[i][j]$, max$[i][j-1]$) is performed.

Thus max$[i][j]$ is the maximum of the sum of a$[i+j-k,\cdots,i][\ell,\cdots,r]$ for $1\le\ell\le\ell'\le j$.

P5: At time $k-1$, solution$[i-1][j]$ is the maximum sum of a$[i-1+j-(k-1),\cdots,i-1][1,\cdots,j]=$
a$[i+j-k,\cdots,i-1][1,\cdots,j]$.

solution$[i][j-1]$ is the maximum sum of a$[i+j-1-(k-1),\cdots,i][1,\cdots,j-1]=a[i+j-k,\cdots,i][1,\cdots,j-1]$.

solution$[i][j]$ is the maximum sum of a$[i+j-k+1,\cdots,i][1,\cdots,j]$

At time k, solution$[i][j]$ =maximum(solution$[i][j]$, solution$[i-1][j]$, solution$[i][j-1]$, max$[i][j]$) is performed.

Thus solution$[i][j]$ is the maximum sum of a$[i+j-k,\cdots,i][1,\cdots,j]$.

ACKNOWLEDGMENTS

The authors would like to thank David van Leeuwen for configuring and maintaining our *littlered* GPU server, and for his support and part-supervision of two Summer Scholarship students. We have also made use of simulations

of images from the Australian Square Kilometer Array Pathfinder telescope, created by CSIRO Astronomy & Space Science. We thank Dr. Matthew Whiting for his permission to use his *sky models* and in particular the *Continuum* image. Details of where these models can be obtained are provided in the reference section of this article.

REFERENCES

[1] Lindeberg, T., "Detecting salient blob-like image structures and their scales with a scale-space primal sketch: A method for focus-of-attention," *International Journal of Computer Vision* **11**, 283–318 (1993).

[2] Bae, S. E., *Sequential and Parallel Algorithms for the Generalized Maximum Subarray Problem*, PhD thesis, Dept. of Computer Science & Software Engineering, University of Canterbury (2007).

[3] Bentley, J., [*Programming Pearls*], Addison-Wesley, Inc., 2 ed. (2000).

[4] Brae, S. E. and Takaoka, T., "Parallel approaches to the maximum subarray problem," in [*Proc. of the seventh Japan-Korea Workshop on Algorithms and Computation*], 94–104 (2003).

[5] Bae, S. E. and Takaoka, T., "Algorithms for the problem of k maximum sums and a vlsi algorithm for the k maximum subarrays problem," *Parallel Architectures, Algorithms, and Networks, International Symposium on* **0**, 247 (2004).

[6] Weddell, S. and Langford, B., "Hardware implementation of the maximum subarray algorithm for centroid estimation," in [*21st International, Image & Vision Computing NZ 2006 (IVCNZ'06)*], 511–515 (Dec 2006).

[7] Pui, B. H., Hayes-Gill, B., Clark, M., Somekh, M., See, C., Piéri, J.-F. o., Morgan, S. P., and Ng, A., "The design and characterisation of an optical vlsi processor for real time centroid detection," *Analog Integr. Circuits Signal Process.* **32**, 67–75 (June 2002).

[8] Weddell, S., Takaoka, T., Read, T., and Candy, R., "Maximum subarray algorithms for use in optical and radio astronomy," in [*SPIE 8500, Image Reconstruction from Incomplete Data VII*], 85000O–85000O–12 (Oct 2012).

[9] Whiting, M., "ASKAP simulations," (2010). `http://www.atnf.csiro.au/people/Matthew.Whiting/ASKAPsimulations.php`.

[10] Sanders, J. and Kandrot, E., [*CUDA by Example*], Addison-Wesley, Inc. (2011).

[11] Giles, P. M. in [*CUDA Lecture Notes*], Mathematical Institute, University of Oxford (2011). Retrieved August 12, 2011, `http://people.maths.ox.ac.uk/gilesm/cuda/lecs/`.

[12] Thaher, M. and Takaoka, T., "Efficient algorithm for the k-maximum convex sum problem," in [*Proceedings of ICCS 2010*], 1475–1483 (2010).

[13] Thaher, M. and Takaoka, T., "An efficient algorithm for the k-maximum overlapping convex sums," in [*Proceedings of ICCS 2011*], 1288–1295, Elsevier (2011).

[14] Thaher, M. and Takaoka, T., "Improved algorithms for the k overlapping maximum convex sum problem," in [*Proceedings of ICCS 2012*], 754–763, Elsevier (2012).

[15] Fukuda, T., Morimoto, Y., Morishita, S., and Tokuyama, T., "Data mining with optimized two-dimensional association rules," *ACM Trans. Database Syst.* **26**(2), 179–213 (2001).

[16] Hoare, C. A. R., "An axiomatic basis for computer programming," *Communications of the ACM* **12**(10) (1969).

Stephen Weddell received the Bachelor of Applied Science from Curtin University, Perth, Western Australia in 1991 and Master of Applied Science (Eng.) from the same university in 1997. In 2010 he received the Ph.D. in Electrical and Computer Engineering from the University of Canterbury in New Zealand, where he is currently a Lecturer in computer hardware and embedded systems. His current research interests include inverse imaging problems and reservoir computing for classification and prediction of optical wavefront perturbations.

Tristan Read completed a Bachelor of Engineering with Honours at the University of Canterbury, New Zealand, in 2011. Since then he has completed a Summer Scholarship at the same university working on GPU solutions to the maximum subarray problem. He is currently employed as a Software Engineer for Orion Health in New Zealand.

Mohammed Thaher received the Bachelor of Computer Science in 1997 from Al-Isra University, Amman, Jorden. In 2009 he was awarded the Master of Computer Science and Software Engineering from the University of Canterbury in New Zealand. Mohammed worked at IBM-Jorden, Telecom Ltd. in New Zealand. He is currently working to attain the Ph.D. in Computer Science and Software Engineering at the University of Canterbury.

Tad Takaoka received his Ph.D degree in Applied Mathematics from Kyoto University Japan. After working at NTT and a national university in Japan, he joined the University of Canterbury as Professor of Computer Science. He specialises in the design and analysis of algorithms and formal semantics of programming languages.