12th International Conference on Domain Decomposition Methods
Editors: Tony Chan, Takashi Kako, Hideo Kawarada, Olivier Pironneau, ©2001 DDM.org

27. Minimum Overhead Data Partitioning Algorithms for Parallel Video Processing

D T Altılar[1], Y Paker[2]

Introduction

Data partitioning is important in many aspects, such as computational, load distribution, inter-process communication, load and data overhead considering different applications. In this paper, overhead due to data partitioning is discussed and two algorithms are proposed: Almost Square Tiles (AST) and Almost Square Tiles with aspect ratio (ASTwar). We exploit data parallelism, which is suitable for both SPMD and SIMD type of parallel computing.

The applications are selected from image/video processing arena most of which involve some neighbourhood operations that require surrounding pixels such as convolution or motion estimation. However, this never excludes the applicability of these algorithms to any other parallel applications, including linear or differential equation solvers. Both AST and ASTwar are to minimise the amount of overlapped data by defining a partition pattern that comprises rectangular tiles of similar sizes and having an aspect ratio of around 1.

A detailed explanation of the problem is introduced in Section "Background and Problem". "Approaches to Data Partitioning Problem" provides the reader a brief information about a recently proposed approach by Lee and Hamdi [LH95]. The proposed algorithms are defined in detail, and a brief comparison between the algorithms is given in "Two Proposed Algorithms:AST and ASTwar" Section. The paper concludes with a Section suggesting further research.

Background and the Problem

There are number of research areas in which data partitioning occupies an important role, such as instruction level data parallelism [AAL95], graph partitioning [KQR95], image processing for image space (2-D) or object space (3-D) [LH95, Whi92, LWY94, CQ95]. An extended survey on I/O intensive parallel computing is given in [Bre97] emphasising language support. As mentioned before, we take image/video processing domain to illustrate the partitioning ideas developed. Moreover, data partitioning is a very important issue in real-time video processing because any defined task should terminate within 40ms and acquire new data periodically.

Video processing algorithms we are interested in require neighbourhood pixels or blocks to be transmitted as shown in Figure 1. The original image is initially split into rectangles of size $a * b$. However, for the given application which includes a neighbourhood operation, rectangle is expanded in size by n in both directions.

[1]Department of Computer Science, Queen Mary, University of London, altilar@dcs.qmw.qc.uk
[2]Department of Computer Science, Queen Mary, University of London, paker@dcs.qmw.qc.uk

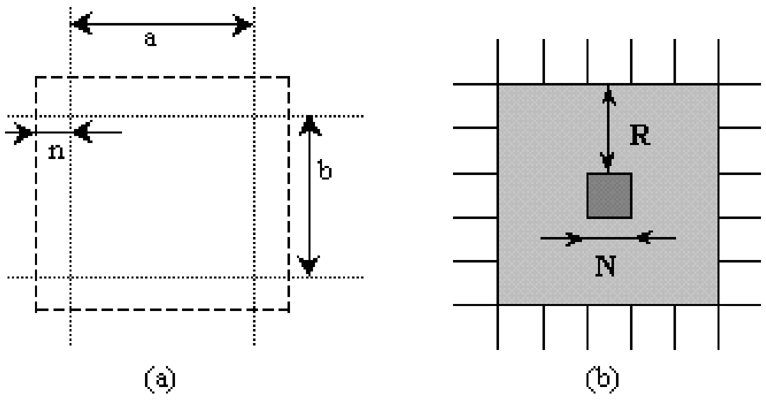

Figure 1: (a): Size expansion of a sub-image of $a * b$ due to the neighbourhood pixels, (b): A core block of *N*N* and search area of (2R+N)*(2R+N)

Figure 1a shows a sub-image of a by b pixels and the required n pixels size expansion. If we are to compute a convolution algorithm, the overall size of the sub-image becomes $(a+2n)(b+2n)$ into an associated coefficient matrix of *2n+1* by *2n+1.* The difference in size in pixels between these two sub-images is given in Eq. 1

$$((a+2n)(b+2n)) - (ab) = 2n(a+b) + 4n^2 \tag{1}$$

As another application, consider motion estimation, which is the most compute intensive part of MPEG video compression: a core block (called "macro block" in MPEG terminology) of N by N from the current frame to be matched with neighbouring blocks of previous frame (Figure 1b) which is a domain of $(2R+N)(2R+N)$ centred on the macro block. Considering the above given example, overhead data becomes significant as R could be up to 16. Thus, comparing with the previous application n could be up to 16 times bigger than a (or b) for this particular application.

When neighbourhood pixels are taken into account, different partition patterns yield different amount of additional data, i.e. data overhead, to be transferred giving rise to a minimisation problem.

Approaches to Data Partitioning Problem

In a recent article [LH95], Lee and Hamdi explain the experimental results of parallel image processing applications on a network of workstations. They exploited image parallelism on a client-server based application model, which they call Host-Node Model. The host splits the image and dispatches to a number of workstations to perform convolution. It is also responsible to collect the distributed sub-images. They consider a one-to-one communication between the host and the other nodes. Nodes are not allowed to communicate among themselves. Above given assumptions on system fit into our model as well.

One of the main concerns they stated in the paper is the impact of the overhead of neighbourhood pixels on the processing time. They proposed a heuristic method for data partitioning which comprises four steps: assuming that t is the number of sub-images (tiles) that the image will split into;

1. If t=1 then fetch another sub-image of whose t¿1,
   if there is no such a sub-image left then terminate.
2. If t is even, divide image into sub-images A and B,
   equally (with the ratio of 1:1)
   horizontally or vertically by keeping overlap minimum.
3. If t is odd, divide image into two sub-images A and B
   with the ratio of (t/2):(t/2)+1,
   horizontally or vertically by keeping overlap minimum.
4. Go to the first step for both sub-image A and sub-image B.

They compared their heuristic method with three standard partitioning methods: cross, column-wise, and row-wise. They indicated that the heuristic method is better than row (or column) partition method but not so good as cross partition. This heuristic method is a divide and conquer type of approach which could lead to undesired partition especially because of the third step of the partition algorithm.

The Core of the Proposed Approach

In order to find a better way of partitioning, we believe the decision should be made considering the original size of the image instead of dividing it into partitions recursively as in the divide and conquer type of approach.

Eq. 1 defines the overhead. If n is a constant as number of partitions, one needs to minimise $a+b$ to minimise the data overhead for $C = a*b$. Since C, load per partition, can be computed for a given n, one can define a generic minimisation problem for the issue: *For a given C, C=a*b , find Min(a+b)* . This is a well known minimisation problem having a solution of

$$a = b = \sqrt{C} \tag{2}$$

Eq. 2 shows that the minimum is achieved for $a = b$, i.e., for a square. In other words, square is the optimal shape for a constant area and minimum circumference. However, it is not always possible to divide a given image into squares of size k for any given number of partitions. Actually it is unlikely to have such a perfect partition except for a few special cases. The partition would comprise a mixture of squares and rectangles of different width and height. For achieving an acceptable solutions the height-width ratio of the rectangles should be close to one.

Two Proposed Algorithms: AST and ASTwar

To solve the above problem, two heuristic algorithms, Almost Square Tiles Data Partitioning Algorithm (AST) and Almost Square Tiles Data Partitioning Algorithm with aspect ratio (ASTwar), have been developed. Let k, the square of an integer, be the

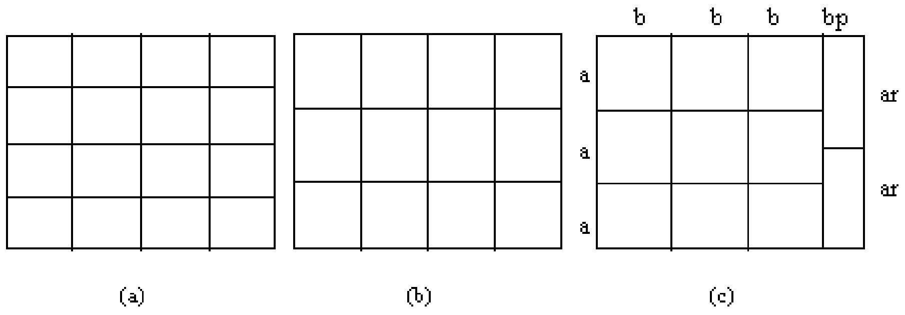

Figure 2: Internal steps of splitting in image of 576*720 into 11 partitions: a=192, ar=288, b=196, and bp=132 pixels.

least number which is greater than or equal to the number of data partitions p to be produced. The frame is split into k tiles with the concern that the height-width ratio of the rectangles should be close to one as much as possible. By changing the width and the height of sub-images afterwards, a partition producing minimum overhead is produced.

Both of the algorithms start by splitting the frame into $k = n^2$, n an integer providing that k is the smallest number greater than or equal to p. There is the possibility of reducing the number of rows (or columns with respect to the aspect ratio) by one for some cases which satisfy $n(n-1) > p$. The algorithm than proceeds to reduce the number of tiles by changing the size of the tiles column-wise.

For example, the above explained steps are shown in Figure 2, for 11 partitions: (a) The frame is split into 16 (4*4) initially although 11 is required, (b) For this particular case, the number of rows is reduced by one since $4(4-1) > 11$, (c) Column-wise changing on the width and reducing the number of tiles to 11 is the latter step of the overall algorithm. This third step comprises computing of height of the rows, i.e., the *a family* consists of *a, ap, ar* and *arp*, and computing width of the columns, i.e, the *b family* consists of *b,bp,* and *bpp* (Figure 3).

Computing the *a family* values is quite simple as number of rows for regular columns are known and number of rows for irregular columns is one less than regulars. *ap* and *arp* are the last tile heights (residues) of the regular and irregular columns respectively. For data balancing the area of tiles should be almost the same, i.e, $a * b = ar * bp$. On the other hand, $width = b * reg\_cols + bp * irr\_cols$. The solution of these two equations gives the value for b and bp.

Almost Square Tiles Data Partitioning Algorithm

In the AST algorithm it is assumed that the width of the frame is equal to or larger than its height. The flow of the developed algorithm can be summarised as follows:

```
 1)  k ← least_greater_or_equal_square(partitions)
 2)  first_square ← squareroot(k)                                          (A)
 3)  cols ← first_square
-----------------------------------------------------------------------------
 4)  if (partitions is a square of an integer) rows ← first_square         (B)
     else if ((rows-1)*cols   partitions) rows ← first_square -1
-----------------------------------------------------------------------------
 5)  irr_col ← cols * rows - partitions                                    (C)
-----------------------------------------------------------------------------
 6)  a ← image_height/rows
 7)  ap ← image_height - a * (rows -1)
 8)  ar ← image_height/num_rows - 1                                        (D)
 9)  arp ← image_height - ar * (rows - 2)
-----------------------------------------------------------------------------
10)  b ← (image_width/((ar/a) * (cols-irr_cols) + irr_cols)) * (ar/a)
11)  bp ← ( image_width - b * (cols-irr_cols) ) / irr_cols                 (E)
12)  bpp ← image_width - b * (cols-irr_cols) - bp * (irr_cols - 1 )
```

Algorithm could be thought in five functional blocks from A to E. Lines 1 and 2 are to determine the maximum number of columns and rows. The number of tiles is assumed to be a square of an integer. If the number of partitions, *partitions*, is a square number, the number of columns, *columns*, and the number of rows, *rows*, would be the same. Set the number of columns for every row(Line 3). By the end of Block A, the number of column which equals the number of rows is known.

Line 4 is to search for the possibility of dividing the data into less rows than the current value of *rows.* If /em partitions is not a square number then there is such a possibility as shown in Figure 2. The final value of the *rows* is set while terminating Block B.

Block C (Line 5)is to compute the number of irregular columns which is one less than the regular ones.

Block D comprises lines to compute the values of the *a family*, i.e., *a,ap,ar,* and *apr.* *a* and *ar* are tile height for regular and irregular columns respectively where *ap* and *arp* are the last tile heights (residues) of the regular and irregular columns respectively. Computing values for the *a family* members is quite simple as *image_height* is known, the number of rows for regular columns is computed in previous blocks, and the number of rows for irregular columns is one less than regulars. The height of a regular tile, a, can be computed by dividing the image height by the number of the rows (Line 6). Line 7 is to check out whether there is a residue row having different height, *ap.* If there is an irregular column, there will be a repeating tile height as well, which is *ar* (Line 8). There might be a residue row having different height than *ar*, which is *arp* (Line 9).

The *b family* members are computed through Block E. Line 10 possesses the solution of two equations to numerate *b* and *bp.* In order to make the areas of most of the tiles equal: *a*b=bp*ar.* Since image width should be covered by columns: $width = cols * b + irr\_cols * bp$. As *a, ar, cols,* and *irr_cols* are computed previously *b*, in Line 10, and *bp*, in Line11 can be numerated. Block E ends with checking out for size of the residue column.

Thus, one could produce at most six different types of tiles through the given algorithm. Tile type names and sizes are (Figure 3):

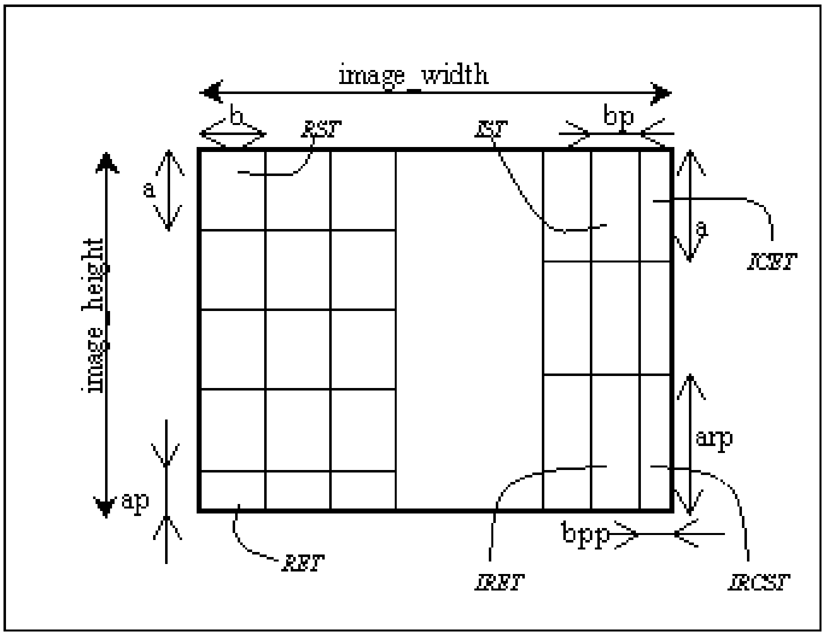

Figure 3: All possible tiles and sizes of tiles to be produced by the proposed algorithms

RST - regular standard ones ($a * b$),
RET - regular excess ones ($ap * b$),
IST - irregular standard one ($bp * ar$),
ICET - irregular column excess ones ($bpp * ar$),
IRET - irregular row excess ones ($bp * arp$),
IRCET - irregular double excess one ($bpp * arp$);

where

a is the standard tile height, ap is the height of the last standard tile in a regular column, ar is the height of irregular column tiles, arp is the height of the last tile in an irregular column, b is the width of the tile of standard regular column, bp is the width of the tiles of irregular columns, bpp is the width of the tiles of the last irregular column.

Almost Square Tiles with aspect ratio Data Partitioning Algorithm

The aspect ratio of the image is taken into account in the ASTwar algorithm. Therefore instead of dividing image into the same number of columns and rows initially, considering the aspect ratio, an image can be divided into different numbers of columns and rows ensuring that widths and heights of rectangles should be as close as possible.

The ASTwar requires the overall ratio for the image. The aspect ratio of the image

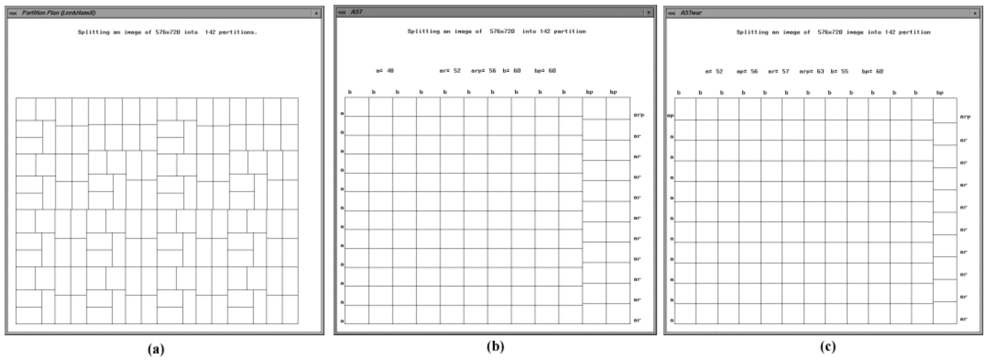
Partition Plan (Lee&Hamdi)
Splitting an image of 576x720 into 142 partitions.
AST
Splitting an image of 576x720 into 142 partition
a= 48 ar= 52 arp= 56 b= 60 bp= 60
ASTwar
Splitting an image of 576x720 image into 142 partition
a= 52 ap= 56 ar= 57 arp= 63 b= 55 bp= 60
(a)
(b)
(c)

Figure 4: Partition patterns for 142 tile: (a)Lee-Hamdi, (b)ATS, (c)ATSwar

is multiplied by the ratio of the number of rows to columns to compute the *overall ratio*: $overall\_ratio = aspect\_ratio\_of\_image * (num\_rows/num\_cols)$

In the ASTwar algorithm *overall_ratio* to be close to 1 where in the AST number of columns is equal to the number of rows as the image ratio is expected as one (or close to one) implicitly. Thus, the ASTwar algorithm is the same as the AST except the block (A) which comprises a loop to set a value for the *overall ratio* as close as possible to 1.

Comparison of the Algorithms

Both AST and ASTwar data partitioning algorithms provide better solution than the one suggested in Lee and Hamdi. The actual values of data overhead for a neighbourhood of 16 pixels are given in Table 1. Even for a neighbourhood of 16 pixels, two proposed algorithms reduce I/0 data amount by upto 10%. Obviously more significant reductions are available for larger values of n.

Partitions	Lee-Hamdi (A)	AST (B)	ASTwar (C)	(A)-(B)	(A-C)	(B-C)
12	78336	74496	74496	3840	3840	0
24	112128	107904	107520	4224	4608	384
110	249792	237200	244544	12592	5248	-7344
130	275072	268672	259440	6400	15632	9232
142	290816	283264	282336	7552	8480	928

Table 1: Actual data overhead in pixels for an image of 576x720 requiring neighbouring pixels of 16.

Partition patterns for 142 partitions are drawn in Figure 4. One should pay attention to the irregularity of shapes in Figure 4a and regularity in Figure 4b and Figure 4c.

Conclusion and Further Research

Two new algorithms, AST and ASTwar, have been introduced to reduce this overhead for data transmission for parallel algorithms requiring neighbourhood pixels. They are both based on the concept that the more tiles are close to squares the less data overhead is to be introduced. Therefore, a global data partition pattern creation, keeping every rectangles height and width as close as possible is the basic approach lying under the two algorithms. ASTwar is slightly different from the first one as it takes the image aspect ratio into account as well. The partition patterns and numerical analysis have shown that the ASTwar algorithm has better performance than the AST algorithm. All of the algorithms are currently being tested for images of different aspect ratios for image/video processing area. These two algorithms for optimal data partitioning are also applicable to other types of parallel applications since optimisation is on overlapped (shared) data. Applying these two algorithms is for parallel numerical solution of partial differential equations is in progress.

References

[AAL95]M. J. Anderson, S. P. Amarasighe, and M. S. Lam. Data and computation transformations for multi-processors. In *Proceedings of the Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PpoPP'95) Santa Barbara CA*, July 19-21 1995.

[Bre97]P. Brezany. *Input/Output Intensive Parallel Computing.* Lecture Notes in Computer Science Series 1220. Springer-Verlag, 1997.

[CQ95]P. E. Crandall and M. J. Quinn. A partitioning advisory system for networked data-parallel processing. *Concurrency: Practice and Experience*, 7(5):479–495, 1995.

[KQR95]M. Kaddoura, C. W. Qu, and S. Ranka. Partitioning unstructured graphs for non-uniform and adaptive environments. *IEEE Parallel and Distributed Technology*, 3, 1995.

[LH95]C. Lee and M. Hamdi. Parallel image processing applications on a network of workstations. *Parallel Computing*, 21:137–160, 1995.

[LWY94]C. Lee, Y. F. Wang, and T. Yang. Static global scheduling for optimal computer vision and image processing operations on distributed memory processors. Technical Report TRC94-23, Dept. of Computer Science, Santa Barbara, CA, December 1994.

[Whi92]S. Whitman. *Multiprocessor Methods for Computer Graphics Rendering.* Jones and Bartlett publishers, Boston, MA., 1992.