2010 12th International Workshop on Cellular Nanoscale
Networks and their Applications (CNNA)

Autonomous long distance transfer on SIMD cellular processor arrays

Marc Geese and Piotr Dudek
School of Electrical and Electronic Engineering
The University of Manchester
United Kingdom
Contact: marc.geese@postgrad.manchester.ac.uk or p.dudek@manchester.ac.uk

***Abstract*—In image processing the tasks of rotating, mirroring and scaling the image are often required. These operations necessitate data transfer between distant elements in the image. SIMD processor arrays commonly support communication within the direct neighborhood only. In this paper a method for implementing long-distance data communication is proposed. Inspired by lattice gas cellular automata models and marching pixels, the developed method leads to an efficient pixel routing algorithm. Further, the use of autonomous elements leads to an emergent behavior that has been studied. Promising results for the above tasks have been achieved for simulations on nearest-neighbor connected processor arrays.**

I. Introduction

Many low-level vision problems can be solved efficiently by the use of pixel-per-processor cellular SIMD architectures [1], [2], [3], [4], [5]. But these cellular processor arrays lack fast long distance communication, as the cells are only connected in the nearest neighborhood. Especially the often used tasks like mirroring, rotating and zooming of an image need information transferred to distant elements on the grid. Therefore we developed a method to achieve this goal in an efficient way, by making use of the "Marching Pixels" (MPs) that were proposed by Fey [6]. For routing the information with MPs, these are equipped with a program code that should be executed by the processing elements (PEs) at the grid position. Because of the general possibilities of such an individual program, each MP gains autonomous behavior and is able to alter the carried information as well as its grid position. Fig. 1 shows the concept of moving MPs on a PE-grid. It should be noted, that this concept is related to lattice gas cellular automata models [7].

II. Considerations for SIMD architectures

The transfer of the above described concept of MPs to a typical SIMD architecture is not straightforward. As described, each PE may be required to execute a different MP program or none, at a given time. The common SIMD processor array architecture requires that all processors execute the same instruction at the same time, or are switched inactive.

As a solution to this problem, we store an index for a MP program locally and then activate only the corresponding processors to execute the program. The program is then broadcast to the entire array in SIMD mode. Of course this approach leads to the situation that different MP programs are issued sequentially, which results in a performance penalty. To improve performance, one can search for identical parts of different MP programs that then can be issued to more than one type of MPs at the same time. Nevertheless one gains most performance on SIMD architectures if one uses only few different types of MPs at a time.

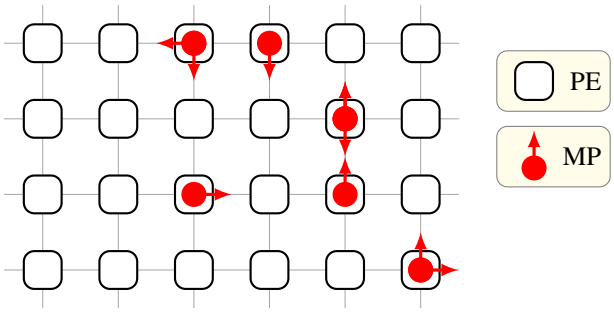


Fig. 1. Data and/or program packets (MPs) are moving in a grid isomorphic with a 2D SIMD processor array

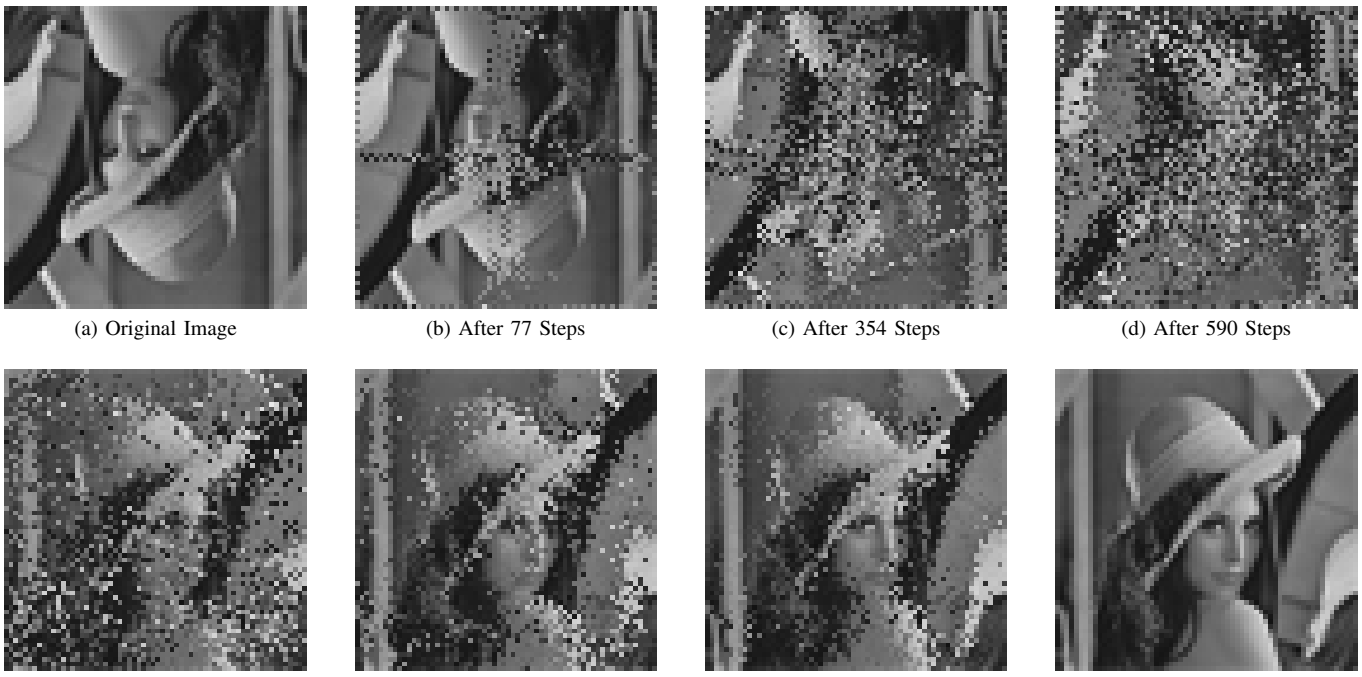
Further, the requirement of local memory of a processing element is critical, as usually only little memory is available. But as it will be shown below, the usage of further local memory leads to an improvement of processing time. Therefore one needs to consider carefully, how much local memory one can spare to gain a higher processing time.

III. March to target applications

A simple but often needed task for a MP is to carry some information (e.g. the pixel's grey level intensity) to a given destination on the grid. These kind of MPs form the basis for long distance data transfer applications. As example applications, one can think of scaling, rotating or mirroring an image. To realize the MPs one needs to store the grid position in each PE and provide the MP with their target destination address plus the information to carry along. Each of the mentioned information, PE's position, destination address and carried information, needs to be adapted to the corresponding image processing task.

The "pixel routing" is then performed. The program, executed on the SIMD array, compares the selected MPs target address with the local processing elements address and moves the MP in the required direction. In detail, this is done by

978-1-4244-6678-8/10/$26.00 ©2010 IEEE



(a) Original Image (b) After 77 Steps (c) After 354 Steps (d) After 590 Steps

(e) After 1770 Steps (f) After 2360 Steps (g) After 4720 Steps (h) FinalState, after 21568 Steps

Fig. 2. A diagonal flip of an Image using MPs. Pictured after different iterations the algorithmn. 2a shows the original image and 2h shows the final result.

checking sequentially for each possible direction, e.g. whether move to north, or east, or south or west. Once a MP is moved to a position where another MP is resident, the MPs positions could be switched. Alternatively one could allow a certain number of MPs to be resident in a single grid position. However, with current cellular SIMD architectures, there are usually restrictions on the local memory available, which puts a constraint on how many pixels can be simultaneously resident at a single grid location.

However, allowing only one MP per grid position leads to another handicap, as two neighboring pixels cannot be allowed to move at the same time. If a MP moves one step and reaches a location where another MP resides, they need to switch positions. A problem occurs, if the second MP itself wants to move to a position, where yet another MP is resident. If one considers, that all MPs switch their positions, the third MP should be switched to the position of the first MP. Fig. 3a and Fig. 3b show what the desired result should be. But as all MPs are executed on a locally connected in SIMD array, the third MP can't be transferred further than within its neighborhood, which would be necessary to succeed. Consequently, applying the MP's switching behavior would lead to corrupt data which is shown in Fig. 3c to 3e.

To solve this problem we considered to detect the groups of MPs that march in the same direction. This information would allow to implement an advanced mode of switching the MPs. For example if one considers the problem from Fig. 3 only pixel 3 needs be be saved in the Temp register and be moved position 1. But the detection of these "group-movements" and the additional necessity of the distinction between MPs that are the first and the last of one group would lead to an extra overhead in memory and processing time.

Algorithm 1 March to Target MPs

```
Require: All MPs have their target destination addresses
Require: All PEs know their grid position
Require: A CheckerboardMask
  repeat
    {invert the mask:}
    CheckerboardMask = NOT(CheckerboardMask)
    {... start with checking for a march to north}
    where MP.Target.X < PE.Position.X there
      where CheckerboardMask there
        SwitchPostionWithNeighbor(North);
      everywhere
    everywhere
    {...then check for south}
    where MP.Target.X > PE.Position.X there
      where CheckerboardMask there
        SwitchPostionWithNeighbor(South)
      everywhere
    everywhere
    ... ... ;
    {Now repeat the above with the Y-Coordinate:}
    { (East and West)}
  until ((TaskIsDone) or (LimitIsReached))
```

A simpler solution to solve this problem is to apply a checkerboard mask, activating half of the cells only at a given time. This way all possible target destinations for a MP are

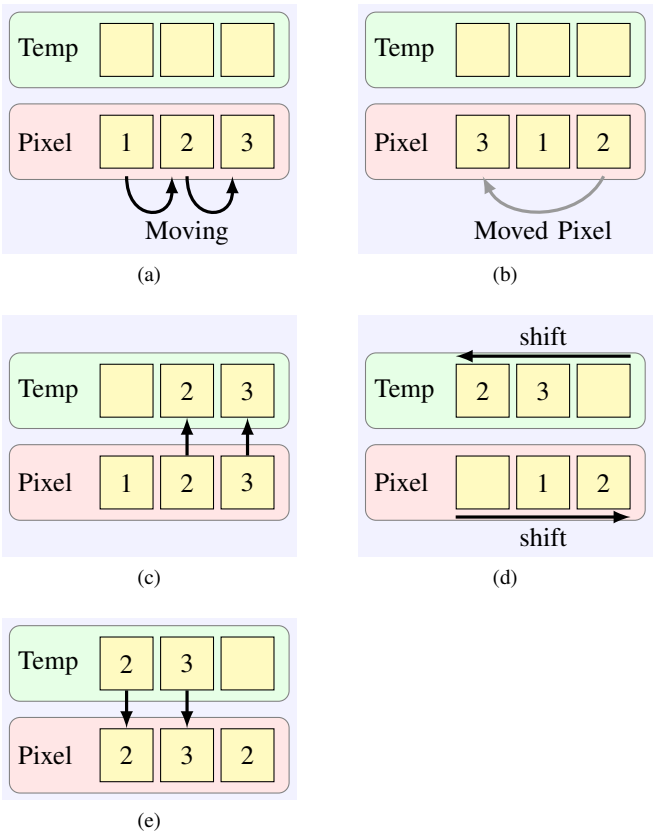


Fig. 3. SIMD copy problem with two neighbors switching positions. (a) moving two neighboring Pixels, (b) the desired result, (c)- (e) the error occurring in SIMD mode: First, saving the information from the target positions to a temporary register. Second, the array is shifted according to the task and last, the saved information is restored faulty.

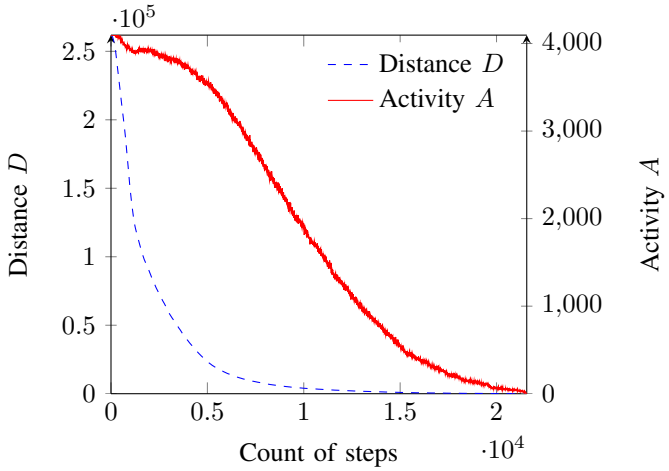


Fig. 5. The number of active MPs A and summed Manhattan distance D for mirroring the 64×64 image in Fig. 2.

set inactive and provide a correct information for swapping the MPs. The checkerboard mask solution is the one we choose to implement, because the overhead in memory and processing time of the "group-movement" method seemed to overwhelm the advantages of that method.

The pseudo-code in algorithm 1 provides the described behavior for processor array architectures. The SIMD array processing conditional flow control commands *where, there, elsewhere, everywhere* are used instead of *if, then, else, endif*. They indicate that the operation is based on the "activity-flag" of the processors in the array, which provides the possibility to disable chosen PEs of the grid.

In the end the algorithm checks the number of MPs that have not reached their target destination. If all MPs have reached their target, the program finishes (*TaskIsDone*). Alternatively the program can also be ended after a fixed number of steps (*LimitIsReached*). This exception needs to be handled, because there is no guarantee that the program will finish the task.

Figure 2 shows different stages of MPs, that were set to flip a 64×64 image on its diagonal and Figure 5 shows the corresponding distance D and activity A. The distance D is defined as the summed up Manhattan distance of all MPs and the activity A is defined as the number of MPs that have not reached their destination. Both values are plotted against the number of steps of the "march" program. One step equals a check for moving to all possible directions with a a given checkerboard mask applied (see Algorithn 1).

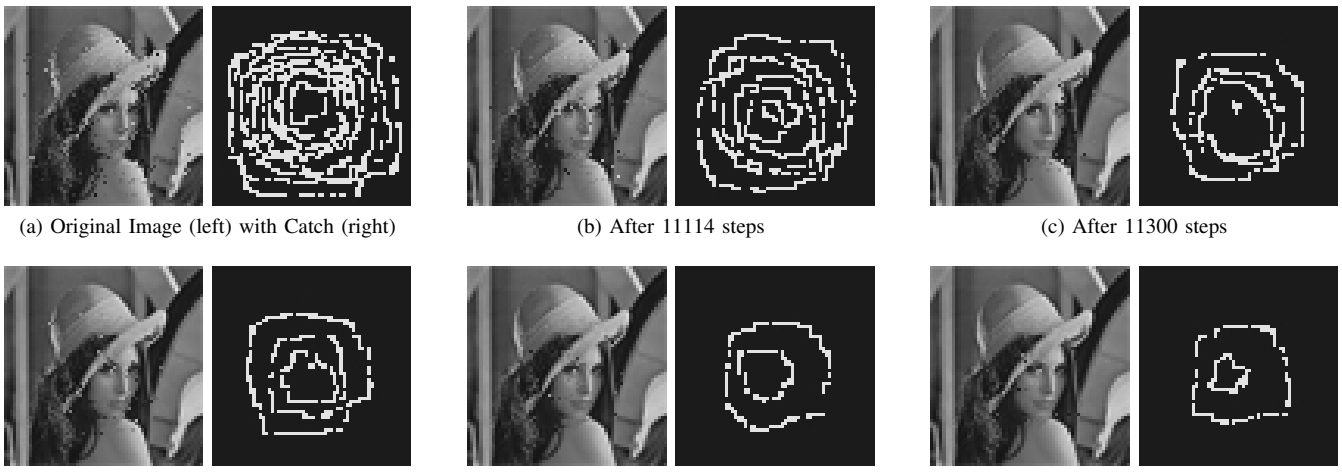
As it can be seen, the simple algorithm is not optimal and takes a very large number of steps to complete the task. Since in the above example, we allow only one MP per grid point and set every pixel in the image as a MP, this leads to a very high density of MPs and therefore to a high number of MP collisions on the grid. Eventually most of the pixels reach their destination, except a small number of MPs which are trapped in "circles of activity". These become visible by plotting only the active MPs at different states of the program, as shown in Fig. 4. These circles are formed by many closed loops of switching pixels, within a distance of about $1-2$ neighborhood diameters.

Within further iterations of the program, the circles collapse slowly. These activity circles emerge irrespective of the order of execution of various program components. They appear to be a general phenomena for an extremely high density of pixels on a routing grid. In the end, the algorithm that tries to use MPs that swap their positions fails to converge quickly if all positions of the grid are occupied. In the example of Fig. 2 took about $22 \cdot 10^3$ steps to finish.

With certain states of the array we observed that the program did not converge to a desired result. In these cases, the wanted result can often be reached by rearranging the execution of the different program parts, such as "inverting the mask", "marching to north", "marching to south", etc. .

A. Improvement with Catch Register

The improvement of processing time can be achieved by reducing the number of MP collisions, which can be reached by clearing the MPs from the grid that have already reached their destination. The information, that has been carried by the MPs needs then to be stored in an extra register. This procedure leads of course to some extra execution time of



(a) Original Image (left) with Catch (right)

(b) After 11114 steps

(c) After 11300 steps

(d) After 11523 steps

(e) After 11626 steps

(f) Final state (11679 steps)

Fig. 4. Circles of activity at the end of the mirroring task from Fig. 2.

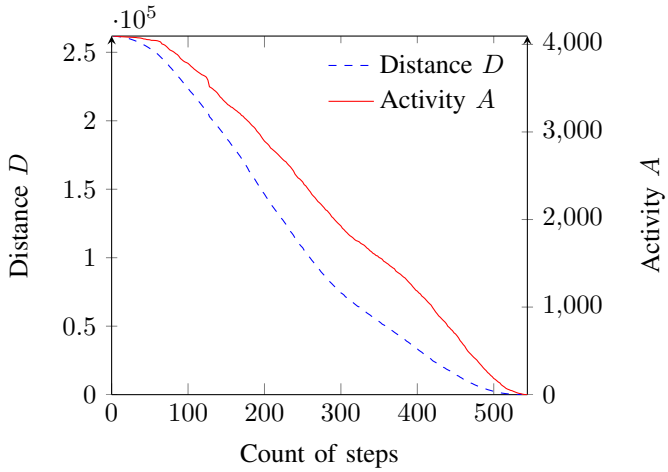


Fig. 7. The number of active MPs A and summed Manhattan distance D for mirroring the 64×64 image in Fig. 6.

the MPs program, but this can be neglected by the improved convergence. In the given example, the number of steps could be reduced from $22 \cdot 10^3$ to 544. Fig. 6 shows the MPs with their carried information as well as the fast filling "catch" register at different stages and Fig. 7 pictures the distance D and activity A as explained before. In addition to the fast convergence, there has not been observed any case of non-convergence while a catch register was used.

B. Improvement with fewer active MPs

A consequent development of the "catch"-register method is to reduce the number of pixels from the beginning of the routing. For this method, we applied a checkerboard mask and then copied only half of the MPs to the grid. After all these MPs had reached their destinations, we released the second half of the MPs and waited again for the program to finish. With this method a "catch"-register is a necessity because the second half of the MPs needs to be put on an empty grid, otherwise they might copy onto MPs from the first wave. The main advantage of this method is that it leads to even fewer collisions between the MPs than in the approach with just a "catch"-register. But again, this method suffers from consuming more local memory because the MPs need to be stored in an extra register before they are set active. Further, the reduction of collisions is not that significant any more, for example in the 64×64 diagonal mirroring task the overall number of steps to convergence is just reduced by 24 from 544 to 520 steps.

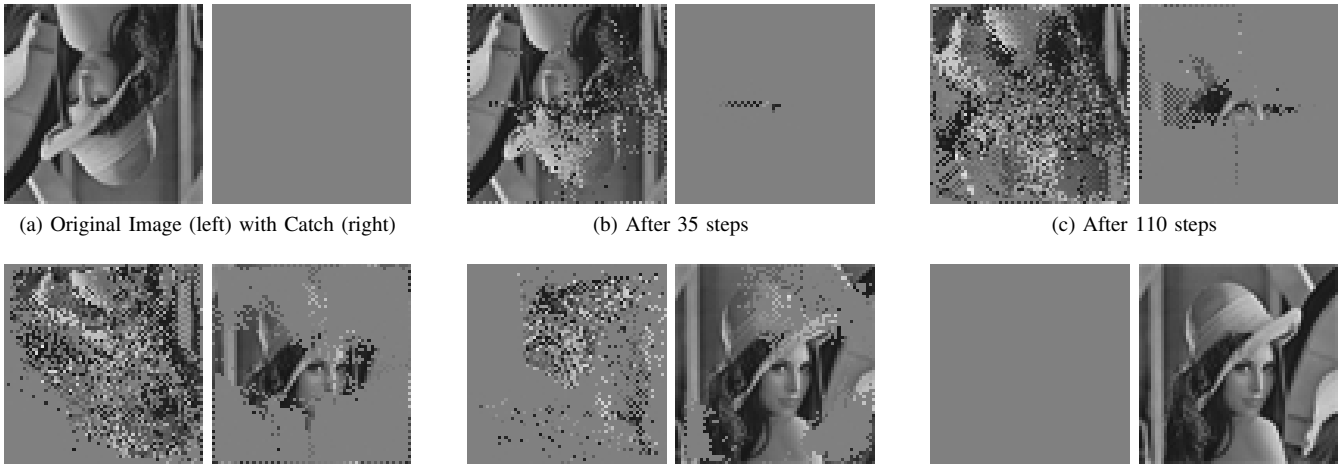
Of course there are various other possibilities to reduce the collisions, that have not yet been investigated. Therefore a better trade-off between local memory demands and speed might be possible.

IV. Further Applications

Further, a simple form of zooming an image can be achieved by MPs as well. To gain a low quality downsized image, we used the following method: First, we made only one pixel in a given $N \times M$ neighborhood a MP and set it to march to the corresponding grid position of the downsized image. Of course this is not the best approach, as effectively the original image is sampled at discrete points, but it is very useful if one needs a fast approximation of a downsized image.

An improvement can be achieved by performing pixel binning by placing an "averaging" MP in the center of a neighborhood. These "averaging" MPs collect all the information from MPs that move to their position and erase these MPs afterwards. In a second step the averaged information is set to march to their target position in a downsized image. Fig. 8 shows this method at different stages. The result is a downsized image of a higher quality, especially if the considered neighborhood for averaging is large.

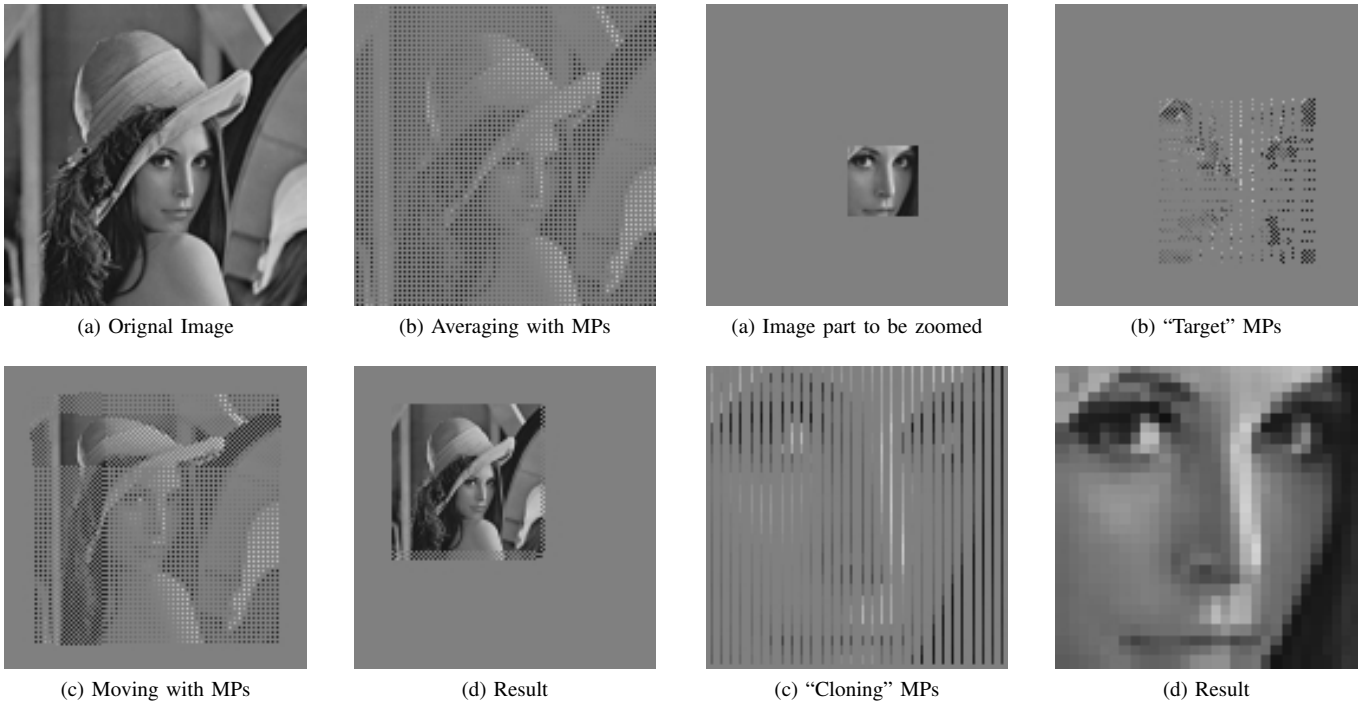
For scaling an image up, one uses again a two stage method. First, one sets each MP in the interesting part of the image to march to a position on the grid that corresponds to the center point of a neighborhood of the upscaled image. For



(a) Original Image (left) with Catch (right) (b) After 35 steps (c) After 110 steps

(d) After 200 steps (e) After 400 steps (f) Final state (544 steps)

Fig. 6. Diagonal Flip of a Test-Image using a "catch" register. Pictured after different executions of the "march"-program. 6a shows the original Image and 6f the final result.



(a) Orignal Image (b) Averaging with MPs

(c) Moving with MPs (d) Result

Fig. 8. A two stage downscaling method of an image. Using first "averaging" MPs and then "target" MPs.

(a) Image part to be zoomed (b) "Target" MPs

(c) "Cloning" MPs (d) Result

Fig. 9. A upscaling method with two stages. First "target" MPs and second "cloning" MPs.

example one could set the MPs to spread equally over the whole array. In the second stage, the MPs are then made "cloning" MPs. These MPs duplicate by copying themselves to their neighborhood. They stop to copy if another cloning MP is resident at that location they want to copy to. The final effect is similar to a pixel size increase. Fig. 9 illustrates the described method. We suppose that a more elaborate method could be also developed, using "interpolating" MPs to obtain a more smooth version of the scaled up image.

V. Conclusion

MPs present a practical solution for solving problems of long distance information transfer in nearest-neighbor connected processor arrays. Although, it cannot be guaranteed, that they are the most efficient way to deal with a given task, nor that system of MPs converges in all cases. In particular we discussed that the trade-off between the local memory requirements and the convergence time needs to be evaluated for each given task. In general, the use of more local

memory seems to cause increased convergence times. Finally, the proposed MP implementation on SIMD architectures leads to an easy configurable and effective tool for many image transformation tasks.

ACKNOWLEDGMENT

The authors would like to thank David Barr for the development and adaption of the APRON Simulation environment [8], where all the simulations took place. This work was funded by EPSRC, grand no. $EP/D029759$. We also thank the Office of Naval Research (ONR) for their financial support.

REFERENCES

[1] A. Lopich and P. Dudek, "Architecture of a vlsi cellular processor array for synchronous/asynchronous image processing," in *ISCAS*, 2006.

[2] S. Carey and P. Dudek, "General-purpose 128x128 simd processor array with integrated image sensor," *Electronic Letters*, vol. 42, June 2006.

[3] A. Rodriguez-Vazquez, G. Linan-Cembrano, L. Carranza, E. Roca-Moreno, R. Carmona-Galan, F. Jimenez-Garrido, R. Dominguez-Castro, and S. Meana, "Ace16k: the third generation of mixed-signal simd-cnn ace chips toward vsocs," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 51, May 2004.

[4] M. Laiho, J. Poikonen, P. Virta, and A. Paasio, "A 64x64 cell mixed-mode array processor prototyping system," *Cellular Neural Networks and Their Applications, 2008. CNNA 2008*, July 2008.

[5] A. Nieto, V. Brea, and D. Vilarino, "Simd array on fpga for b/w image processing," *Cellular Neural Networks and Their Applications, 2008. CNNA 2008.*, 2008.

[6] D. Fey and D. Schmidt, "Marching-pixels: a new organic computing paradigm for smart sensor processor arrays," *Proceedings of the 2nd conference on Computing frontiers*, pp. 1–9, 2005.

[7] D. A. Wolf-Gladrow, *Lattice-gas cellular automata and lattice Boltzmann models: an introduction*. Springer, 2000, p. 39.

[8] D. R. W. Barr and P. Dudek, "Apron: A cellular processor array simulation and hardware design tool," *EURASIP Journal on Advances in Signal Processing*, no. ID 751687, p. 9, 2009.