Parallel low-level image processing on a distributed-memory system

Cristina Nicolescu and Pieter Jonker

Delft University of Technology
Faculty of Applied Physics
Pattern Recognition Group
Lorentzweg 1, 2628CJ Delft, The Netherlands
email: cristina,pieter@ph.tn.tudelft.nl

Abstract. The paper presents a method to integrate parallelism in the DIPLIB sequential image processing library. The library contains several framework functions for different types of operations. We parallelize the filter framework function (contains the neighborhood image processing operators). We validate our method by testing it with the geometric mean filter. Experiments on a cluster of workstations show linear speedup.

1 Introduction

For effective processing of digital images it is essential to compute the data using a variety of techniques such as filtering, enhancement, feature extraction, and classification. Thus, there is a great need for a collection of image processing routines which can easily and effectively be used on a variety of data. We used in our research an image processing library called DIPLIB (Delft Image Processing LIBrary) [11] developed in the Pattern Recognition Group, Delft University of Technology. It provides a basic set of image handling and processing routines and a framework for expanding the set of image processing routines.

While the library provides the necessary functionality and flexibility required for image processing applications tasks, it is clear that for real-time processing, many important image processing tasks are too slow. For example, a filter operation which removes noise from a 1024×1024 pixel image requires several minutes to complete on a common desktop workstation. This is unreasonable in real-time image processing. A method to speedup the execution is to use existing workstation cluster for parallelism [1, 2, 3, 5]. In [3] the authors present the design and implementation of a parallel image processing toolkit (PIPT), using a model of parallelism designed around MPI. Currently, we are developing a parallel/distributed extension to the DIPLIB. We developed 2 parallel versions of the library on top of MPI [7] and CRL [6], respectively. As a consequence, the code remains portable on several platforms.

The paper is organized as follows. Section 2 presents a classification of low-level image processing operators. Section 3 describes an approach of integrating parallelism in the sequential image processing library. Execution times obtained for the geometric mean filter are presented and interpreted in Section 4. Section 5 concludes the paper and Section 6 presents future work.

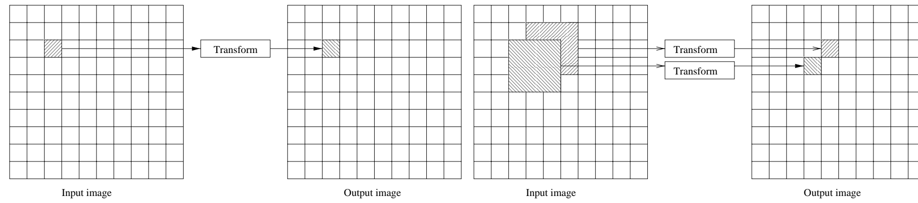

Fig. 1. Point low-level operator

Fig. 2. Neighborhood low-level operator

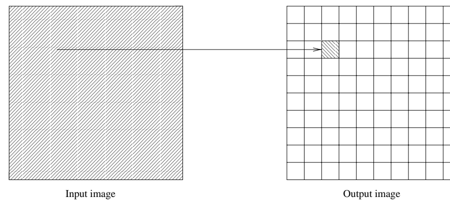

Fig. 3. Global low-level operator

2 Low-level image processing operators

Low-level image processing operators can be classified as *point operators*, *neighborhood operators* and *global operators*, with respect to the way the output pixels are determined from the input pixels [4].

The simplest case are the *point operators* where a pixel from the output image depends only on the value of the same pixel from the input image, see Figure 1. More generally, *neighborhood operators* compute the value of a pixel from the output image as an operation on the pixels of a neighborhood around a corresponding pixel from the input image, possibly using a kernel mask. The values of the output pixels can be computed independently, see Figure 2. The most complex case consists of *global operators* where the value of a pixel from the output image depends on all pixels from the input image, see Figure 3. It can be observed that most of the image processing operators exhibit natural parallelism in the sense that the input image data required to compute a given area of the output is spatially localized. This high degree of natural parallelism exhibited by many image processing algorithms can be easily exploited by using parallel computing and parallel algorithms.

3 Integrating parallelism in an image processing library

To quicken the development of image processing operators, the DIPLIB library supplies several framework functions.

One of the available frameworks is responsible for processing different types of image filters (neighborhood image processing operators). The framework is

intended for image processing filters that filter the image with an arbitrary filter shape. By coding the shape with a pixel table (run length encoding), this framework will provide the filter function with a line of pixels from the image it has to filter. The filter function is allowed to access pixels within a box around each pixel. The size of this box is specified by the function that calls the framework. A description of the framework is presented in Figure 4. Each neighborhood operator calls a framework function which sequentially compute each line of the image. We parallelize the framework approach by data decomposition on a distributed-memory system and in this way we obtain parallelization on all image processing operators (i.e. filters) that are using the framework. So, given a neighborhood image processing operator, the operator calls the framework function on the master processor. The image is distributed by the master processor row-stripe across processors, each processor is computing its part of the image and then the master processor gathers the image back, see Figure 5.

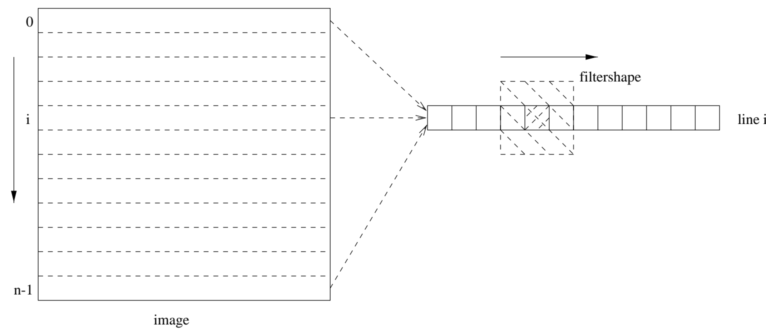

Fig. 4. Library framework function

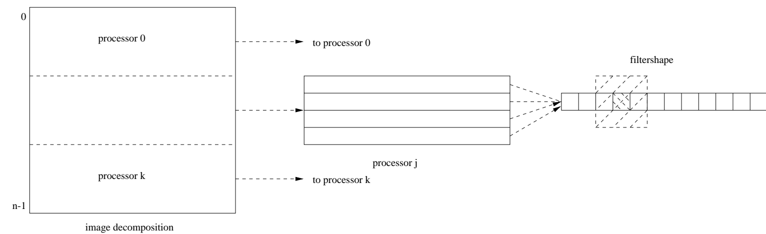

Fig. 5. Library framework function after parallelization

4 Experimental results

We measure the execution time of applying a $k \times k$ window size geometric mean filter on different image sizes. Geometric mean filter is used to remove the gaussian noise in an image. The definition of the geometric mean filter is as follows:

$$GeometricMean = \prod_{(r,c) \in W} [I(r,c)]^{\frac{1}{k^2}}$$

where (r,c) are the image pixel coordinates in window W, $I(r,c)$ is the pixel value and k is the width of the window, measured in pixels.

The filter has been implemented under CRL (C Region Library) [6] and MPI (Message Passing Interface) [7] on a distributed memory system. Two artificial images sizes 256×256 and 1024×1024 are tested on up to 24 processors. The run times in seconds using CRL are tabulated in Table 1. The run times in seconds using MPI are tabulated in Table 2.

The relative speedups computed as $SP(N) = \frac{T(1)}{T(N)}$ are plotted in Figures 6,7,8 and 9 for each image and window sizes of 3×3, 9×9 and 15×15. One may note the sharp increase of the speedup with increasing number of processors for both images. One may also observe that better performance is obtained with larger image sizes and larger window sizes. Thus, the lowest speedup corresponds to the 256×256 image and 3×3 window while the highest speedup to the 1024×1024 image and 15×15 window. The reason is the image operator granularity increases with the image size and the window size. As a consequence, communication time is less predominant compared to computation time and better performance is obtained. Some differences can be noted in the run times between CRL and MPI. This is because on our distributed memory system CRL runs on LFC (Link-level Flow Control) [8] directly while our MPI port uses Panda [9] as a message passing layer and Panda runs on top of LFC, so some overhead appears.

Table 1
Parallel geometric mean filtering execution time (in seconds) with CRL

N	256x256	256x256	256x256	1024x1024	1024x1024	1024x1024
	3x3	9x9	15x15	3x3	9x9	15x15
1	1.10	9.23	24.30	16.93	145.89	398.05
2	0.61	4.68	12.21	8.79	73.19	199.35
4	0.31	2.24	5.86	4.46	36.71	99.99
8	0.17	1.13	2.99	2.34	18.46	50.31
16	0.12	0.61	1.57	1.29	9.42	25.47
24	0.10	0.47	1.13	0.94	6.40	17.19

Table 2
Parallel geometric mean filtering execution time (in seconds) with MPI

N	256x256	256x256	256x256	1024x1024	1024x1024	1024x1024
	3x3	9x9	15x15	3x3	9x9	15x15
1	1.10	9.23	24.30	16.93	145.89	398.05
2	0.69	5.44	14.72	9.12	74.89	202.35
4	0.35	2.72	7.35	5.35	37.25	100.89
8	0.19	1.36	3.66	2.79	19.84	51.08
16	0.13	1.01	1.81	1.48	11.45	26.42
24	0.82	0.97	1.2	1.12	8.23	18.58

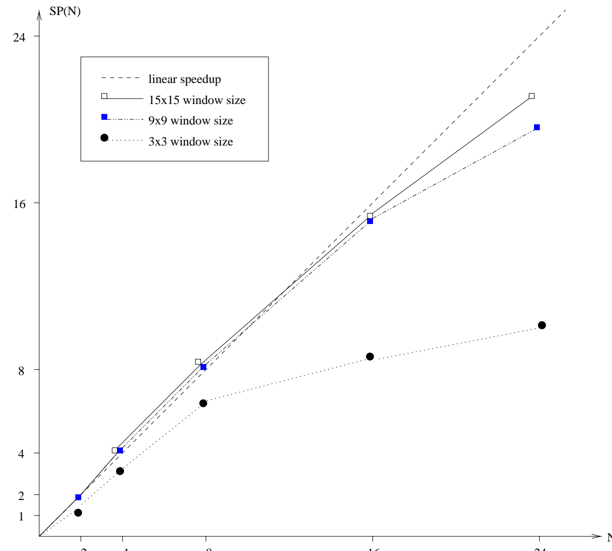

Fig. 6. Speedup of geometric mean filtering on a 256 × 256 image size for different window sizes, with CRL

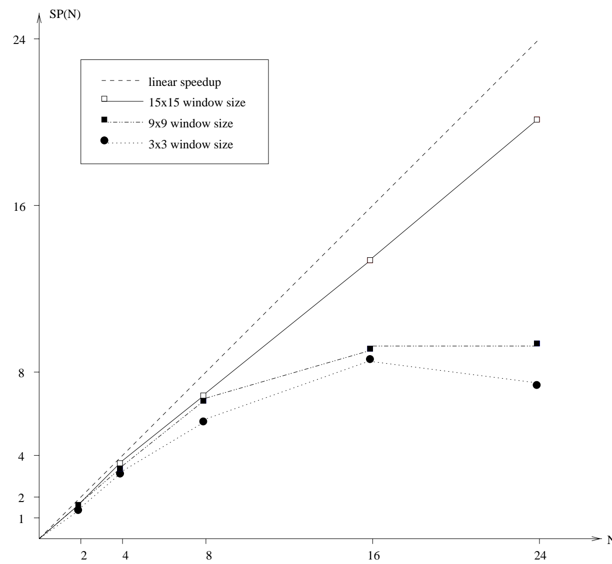

Fig. 7. Speedup of geometric mean filtering on a 256 × 256 image size for different window sizes, with MPI

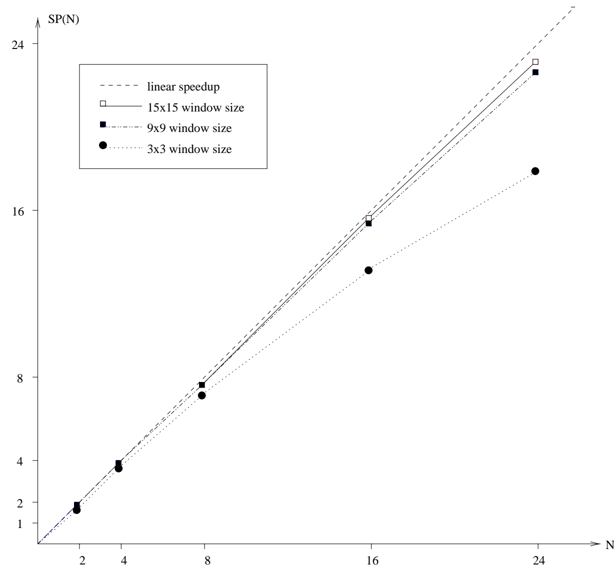

Fig. 8. Speedup of geometric mean filtering on a 1024×1024 image size for different window sizes, with CRL

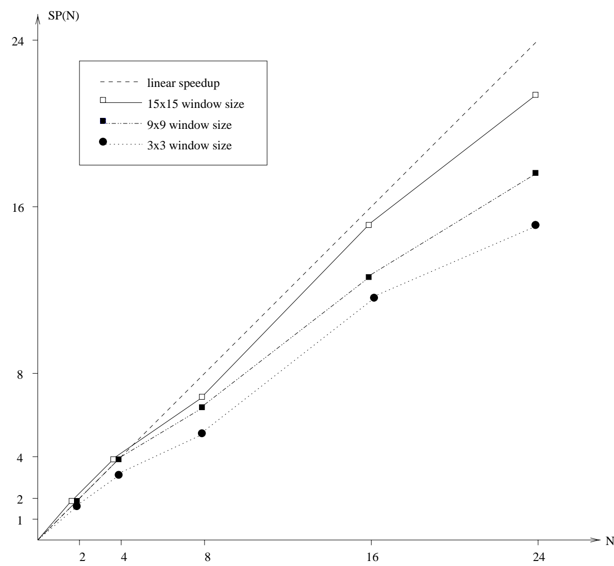

Fig. 9. Speedup of geometric mean filtering on a 1024×1024 image size for different window sizes, with MPI

5 Conclusions

We present a very easy approach of adding parallelism to a sequential image processing library. The method parallelizes a framework function of the library responsible for processing filter operators. Linear speedups are obtained on a cluster of workstations for very intensive computational filters, such as geometric mean filtering.

6 Future work

Nowadays, most of the cluster of workstations consist of workstations containing Intel processors with MMX technology [10]. By exploiting the features added by the MMX technology, an image processing library can be further parallelized. We are working to integrate MMX features into the parallel version of DIPLIB.

MMX technology is designed to accelerate multimedia and communications applications. The technology includes new instructions and data types that allow applications to achieve a new level of performance. It exploits the parallelism inherent in many multimedia and communications algorithms, yet maintains full compatibility with existing operating systems and applications.

The highlights of the technology are:

- Single Instruction, Multiple Data (SIMD) technique
- 57 new instructions
- Eight 64-bit wide MMX technology registers
- Four new data types

A process technique called Single Instruction Multiple Data (SIMD) behaves as a SIMD parallel architecture but at a lower level. Special MMX instructions to perform the same function on multiple pieces of data. MMX technology provides eight 64-bit general purpose registers. These registers are aliased on the floating point (FP) registers. This means that physically MMX registers and FP mantissas are the same but the content is interpreted in a different way depending on the MMX/FP mode. The MMX registers are accessed directly using the register names MM0 to MM7. The principal data type of MMX technology is the packed fixed-point integer. The four new data types are: packed byte, packed word, packed double word and quad word all packed into one 64-bit quantity in quantities of 8, 4, 2 and 1 respectively. For this reason, given an array of element type byte, word or double word the processing of that array with simple operations (addition, subtraction, etc.) will be 8, 4 and respective 2 times faster. In Figure 10 we show an approach of including MMX features in our DIPLIB library. We begin by parallelizing the point framework which is similar to the filter framework, except that we use point operators instead of neighborhood operators. A master processor is distributing the image in a row-stripe way to the slave processors and each slave processor is computing its part of the image by applying the point operator to each line of that part of image. If the slave

processor is enabled with MMX technology we exploit the MMX features of processing in parallel more elements of a line. This part has to be coded using MMX instructions.

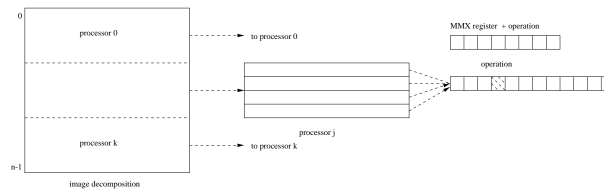

Fig. 10. Adding MMX features to DIPLIB image processing library

References

1. P.Challermvat, N. Alexandridis, P.Piamsa-Niga, M.O'Connell: *Parallel image processing in heterogenous computing network systems*, Proceedings of IEEE International Conference on Image Processing 16-19 sept. 1996,Lausanne,vol.3,pp.161-164
2. J.G.E. Olk, P.P. Jonker: *Parallel Image Processing Using Distributed Arrays of Buckets*, Pattern recognition and Image Analysis, vol. 7, no. 1,pp.114-121,1997
3. J.M. Squyres, A. Lumsdaine, R. Stevenson: *A toolkit for parallel image processing*, Proceedings of the SPIE Conference on Parallel and Distributed Methods for Image processing, San Diego, 1998
4. S.E.Umbaugh: *Computer Vision and Image Processing - a practical approach using CVIPtools*, Prentice Hall International Inc.,1998
5. I.Pitas: *Parallel Algorithms for Digital Image Processing, Computer Vision and Neural Networks*, John Wiley&Sons, 1993
6. K.L.Johnson, M.F.Kaashoek and D.A.Wallach: *CRL: High-Performance All-Software Distributed Shared Memory*, Proccedings of the Fifteenth Symposium on Operating Systems Principles, 1995
7. M.Snir, S.Otto, S.Huss, D.Walker and J.Dongarra: *MPI - The Complete Reference, vol.1, The MPI Core*, The MIT Press, 1998
8. R.A.F. Bhoedjang, T. Ruhl and H.E. Bal: *Efficient Multicast on Myrinet Using Link-level Flow Control*, Proceedings of International Conference on Parallel Processing, pp. 381-390, Minneapolis MN, 1998
9. T.Ruhl, H, Bal, R. Bhoedjang, K. Langendoen and G. Benson: *Experience with a portability layer for implementing parallel programming systems*, Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications, pp. 1477-1488, Sunnyvale CA, 1996
10. J.E. Lecky: *How to optimize a machine vision application for MMX*, Image Processing Europe, March Issue, pp. 16-20, 1999.
11. *http://www.ph.tn.tudelft.nl/Internal/PHServices/onlineManuals.html*

This article was processed using the LaTeX macro package with LLNCS style