Parallel Computing 34 (2008) 694–699

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Parallel processing for image and video processing: Issues and challenges

Alain Merigot [a], Alfredo Petrosino [b,*]

[a] Institut d'Electronique Fondamentale, Université Paris Sud, Orsay, France
[b] Department of Applied Sciences, University of Naples "Parthenope", Naples, Italy

ARTICLE INFO

Article history:
Available online 20 September 2008

Keywords:
Image analysis
Computer vision
Parallel computing
Distributed computing

ABSTRACT

Some meaningful hints about parallelization problems in image processing and analysis are discussed. The issues of the operation of various architectures used to solve vision problems, from the pipeline of dedicated operators to general purpose MIMD machines, passing through specialized SIMD machines and processors with extended instruction sets, and parallelization tools, from parallel library to parallel programming languages, are reviewed. In this context, a discussion of open issues and directions for future research is provided.

1. Introduction

In this paper, a generic framework of the parallel image and video processing is introduced to practitioners both in terms of architectural and programming points of view (Section 2). The idea is to provide a brief review of some key hints and discuss specific related issues, also through the papers reported in this issue (Section 3), highlighting current trends and open issues (Section 4).

2. Framework

Without doubt recent years have seen the emergence of image processing and multimedia as a key technology for modern computing. Besides traditional applications of image processing (robotics, surveillance, visual inspection), many new domains have emerged (video conferencing, video-on-demand, image databases) while others took an important rise (medical imaging, data visualization).

Although an image already consists of a large set of information, many present applications require to process high dimensional image data: time sequences for video application, large datasets for image archiving and retrieval, volumetric slices for 3D medical imaging applications [3]. This gives rise to several progresses in digital signal processing algorithms, while the need of performing complex image analysis in several domain tends to increase the computational requirements for image manipulation. The combination of the important computational effort that is required and the need of fast response time, combined to the inherently data parallel organization of images lead to the use of parallel computation in image processing applications.

* Corresponding author.
E-mail addresses: am@ief.u-psud.fr (A. Merigot), alfredo.petrosino@uniparthenope.it (A. Petrosino).

0167-8191/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.parco.2008.09.009

2.1. The architectural side

For many years, computer scientists have studied parallel architectures for solving image processing problems.

During the 1950s, some proposals for massively parallel architectures appeared [49]. But during the Seventies, it was soon realized that many identical local computations were required, so that it seemed natural to think about replicating the instruction executors, one for each pixel or for each instruction (SIMD and MIMD respectively). This led to the first SIMD architectures (e.g. ILLIAC IV[1], MPP [4], CLIP4 [14]), CLIP7 [15] and the MIMD class (e.g. Cytocomputer [30], ZMOB [42]). At those time (beginning of the1980s) programming was still a hard task for multiprocessor architectures even when using commercial microprocessors like the Z80. The real problem to be solved involved different issues: economical, technological, structural (pipeline/array/reconfigurable) and usability [7]. Moreover, the passage to more sophisticated algorithms of image analysis led to the need for enriching the mechanisms of communication between the elementary processors. Thus, during the Eighties, the researchers considered the use of various topologies allowing to enrich the communications between nodes by a parallel machine: pyramid, hypercube, etc. The most outstanding results in image processing relate to the use of *pyramid* topologies, whose interconnection reflects the data movements present in many algorithms of image processing, in particular the multiresolution processing. Various projects and achievements were thus carried out [5,32,6].

Probably there have been more specialized architectures proposed for images than for any other application, leading to the introduction of specific SIMD instructions in microprocessors, subject to image processing constraints. Most of microprocessors have their multimedia extension now: INTEL with MMX (Multimedia extension)[41] and the second generation Streaming SIMD extensions [22], Sun with SCREW (Visual Instruction Set) [48], HP with MAX (Media ACCelerator) [29], DIGITAL with MVI (Video Motion Instructions) [12], Motorola with AltiVec[38], etc. They differ by the type of authorized data (8/16/32 bits, even floating), by the existence of more or less specialized particular instructions and some details in the instruction set.

On the other side, general purpose architectures and programming concepts have been developed and are presently widely used around the world, becoming an everyday tool for a large number of users. Symmetric shared memory multicomputers are quite common for applications ranging from e-commerce servers to database manipulation. Processor clusters are present in most computer science laboratories and in a growing number of industries.

More recently, the emerging grid technology [39] promises to connect altogether all these resources to provide virtually unlimited computing power. Typically, computing nodes, storage nodes and fast network connections represent the hardware. Two levels of middleware could be considered: the low-level middleware services provide homogeneous access to the underlying heterogeneous components; the upper level middleware provides a user interface for deploying grid-enabled applications with data sharing and distributed computation capabilities. The main aim of grid technologies is to offer a unique environment for sharing data and computing or storage resources through very generic services for deploying large scale applications such as users authorization, authentication and accounting, data replication and fast transfer, and transparent access to computing resources. One of the research advances consist in the development of specialized higher-level layers which take into account the image and video processing specific requirements. Some recent successful experiences in biomedical image processing could be found in [31,34].

Also wireless microsensor networks have been identified as one of the most important technologies for the 21st century, with applications such as infrastructure security, habitat monitoring, and traffic control. Technical challenges in sensor network development include network discovery, control and routing, collaborative signal and information processing, tasking and querying, and security [9].

It is not surprising in these conditions that the image processing community, who has been relying for years on special purpose computers to accelerate their computations, turns towards a mature technology of parallel processing. They can benefit simultaneously from reliable and fast hardware, solid software concepts and programming tools that allow to implement the more and more complex applications required for image and multimedia applications.

Apart from its design and capabilities, the (commercial) success of any computer architecture significantly depends on the availability of tools that simplify software development. As an example, for many users it is often desirable to be able to develop programs in a high-level language such as C or Java. Unfortunately, for many of the architectures so far referred, available high-level language compilers often have great difficulties in generating assembly code that makes effectively use of the capabilities of a parallel machine. To obtain truly efficient code the programmer often must optimize the critical sections of a program by hand. Whereas assembly coding or hand-optimization may be reasonable for a small group of experts, most users prefer to dedicate their time to describing *what* a computer should do rather than *how* it should do. Consequently, many different programming tools have been developed that attempt to alleviate the problem of low-level software design for parallel and distributed computers. In all cases such tools are provided with a programming model that to a certain extent abstracts from the idiosyncrasies of the underlying parallel hardware. However, the small user base of parallel computing within the image processing research community indicates that (as of yet) no parallelization tool has been provided that has a truly familiar programming model.

Because many different image operations incorporate similar data access patterns, a relatively small number of alternative parallelization strategies often need to be considered. These observations have led to the creation of software development tools that are specifically tailored to image processing applications. Such tools may provide more higher-level abstractions to the user than general purpose tools, and are potentially much more efficient as important domain-specific assumptions often can be incorporated.

2.2. The programming tools side

One approach to integrating domain-specific knowledge is to design a specific programming language for parallel image processing specifically, like Apply [20] and Adapt [50].

Apply and Adapt exploit this idea by requiring the programmer to write only the innermost per pixel portion of the computation. The iteration is then implicit and can be easily made parallel; the compiler needs only to divide the images among processors and then iterate the Apply program over the image sections allocated to each processor. Despite the fact that the language was capable of providing significant speedups for many applications, the programming model proved to be too restricted for practical use. To overcome this problem, in Adapt the basic principles of Apply were extended to incorporate global operations as well. In such operations an output pixel can depend on many or all pixels in the input image. Adapt's programming model, however, was not ideal as the programmer is made responsible for data partitioning and merging, albeit at quite a high-level.

An alternative approach was taken in a language named IAL (Image Algebra SIMD Programming Language [37]), based on the abstractions of Image Algebra [43], a mathematical framework for operations and operands able to provide the capability of expressing all image-to-image transformations, without accessing to individual pixels. Operands are basic language operands, images and template.

Typical operations comprise (a) unary over images (sum, product, maximum, minimum, pseudo-inverse, transpose), (b) binary point-to-point (sum, subtraction, multiplication, division, maximum, minimum), (c) binary image-template (convolution, multiplicative maximum, multiplicative minimum, additive maximum, additive minimum), (d) binary template–template (sum, subtraction, multiplication, division, maximum, minimum). The notational adaptability to programming languages allows the substitution of extremely short and concise image algebra expressions for equivalent blocks of code, and therefore increases programmer productivity.

Two extended versions, I-BOL [2] and Tulip [46], provide a more flexible and powerful notation. They treat an image as a tuple of sets. These languages allow access to data at either the pixel or neighborhood level, without being architecture-specific; a number of low-level and intermediate-level vision tasks have been implemented using user-defined neighbourhood functions.

An alternative to the language approach is to provide an extensive set of parallel image processing operations in library form. In principle, this approach allows the programmer to write applications in a familiar sequential language, and make use of the abstractions as provided by the library [47,23,28].

One particularly interesting data parallel library implementation is described by Taniguchi et al. [47], applicable to both SIMD- and MIMD-style architectures, and incorporates a data structure abstraction named DID (Distributed Image Data). The DID abstraction is to be intended as an image data type declaration, without exposing the details of the actual distribution of data.

Although a DID declaration is simple, and easy to understand for programmers unfamiliar to parallel computing, it has the major disadvantage of making the user responsible for the data distribution type.

The alternative library-based environment described by Jamieson et al. [23], provides a fully sequential interface to the user. The main feature is the presence of a set of algorithm libraries, along with abstract information about the performance characteristics of each library routine.

The environment incorporates simple performance models to ensure efficiency of execution of complete applications, but it is too limited in functionality to constitute a true solution, as it supports point operations and a small set of window operations. Two environments that follow a similar approach, but are both much more extensive in functionality, are presented in [25,27].

User transparency environments are described in [36,35].

The main feature of the environment proposed in [36] is its so-called self-optimizing class library, which is extended automatically with optimized parallel operations. During program execution, a syntax graph is constructed for each statement in the program, and evaluated only when an assignment operator is met.

Any syntax graph for combinations of primitive instructions (i.e., those incorporated as a single routine within the library) is written out for later consideration by an off-line optimizer. On subsequent runs of the program a check is made to decide if an optimized routine is available for a given sequence of library calls. An important drawback of this approach, however, is that it may guarantee optimal performance of sequences of library routines, but not necessarily of complete programs.

In MIRTIS [35], programs are parallelized automatically by partitioning sequential data flows into computational blocks, which are decomposed in either a spatial or a temporal manner. All issues related to data decomposition, communication routing, and scheduling are dealt with by using simple domain-specific performance models.

Although, from a programmer perspective, MIRTIS constitutes an ideal solution, its implementation suffers from poor maintainability and extensibility. Also, the provided MIRTIS implementation suffers from reduced portability as the applied communication kernels are too architecture specific.

Surely in this context more successful approaches include skeleton-based parallel programming environments, like SKIPPER-o, SKIPPER-i, SKIPPER-ii and SKIPPER-d [18,8,45]. The main SKIPPER components are: a library of skeletons, a compile-time system (CTS) for generating the parallel C code and a run-time system (RTS) providing support for executing this parallel code on the target platform. The CTS can be further decomposed into a front-end, whose goal is to generate a target-independent intermediate representation of the parallel program, and a back-end system, in charge of mapping this intermediate

representation onto the target architecture. The SKIPPER library of skeletons was built *bottom-up*, from a careful analysis of a large corpus of existing low-to-mid level vision applications hand-coded in parallel C. An alternative approach is proposed in [24]; it adopts both algorithmic skeletons and an Image Application Task Graph (IATG) whose nodes are associated to image processing functions and executed on sets of processors from an available pool of processors, while edges represent communication channels. The IATG is determined starting from the source code, where the required data parallel operations are expressed by skeletons, by a Syntactic and Dependency analyzer, while on the extracted IATG, Mapping and Scheduling algorithms, like those reported in [17], are applied to find the minimum execution time of the application. The skeleton library contains skeletons for low-level, intermediate-level and high-level image processing, adopting distributed bucket processing which allows that only subsets of the initial data is dynamically processed. The underlying idea of the use of the buckets is that to bridge the gap between SIMD architectures, naturally devoted to low-level processing, and MIMD architectures, naturally devoted to intermediate- or high-level processing, while maintaining the heterogeneous way of processing. Typically, during each image scan the data of interest to the specific algorithm is collected in one or more *buckets*, data structures that store the data subsets. During the subsequent bucket processing, data elements are drawn from a bucket, processed in a data parallel mode and, when new data is generated, put into a bucket again.

Differently, in [44] a programming model for the development of time-constrained image processing applications on currently available parallel hardware architectures is reported. The approach is based on the definition of a software architecture that allows transparent sequential implementation of data parallel imaging applications for execution on homogeneous distributed memory MIMD-style multicomputers.

A different approach is reported in [13], consisting in a parallel computing library called ANET, designed for shared memory multiprocessors. The idea is to adopt graph-based representations to cope uniformly with different features that are used in image analysis. This model can efficiently exploit structured information within an image by representing it by means of a subgraph of the mesh that interconnects subsets of adjacent pixels. It can also be hierarchically defined (thanks to the so-called *virtual nets*), and this way describe higher-level features in an image. The computing model [33] treats the irregular structures, such as contours or regions of the images, as connected sets of nodes, called *snet*, where only the links that connect the nodes of these connected sets are conserved. Operations over the graph include some specific computing primitives that gather information along the edges of the graph. The specificity of the model resides on (a) global operations over the graph, performed over any subgraph of the mesh, like connected components, contours, etc. and (b) *asynchronous* local computations by relaxing the data-dependent constraints, managing only the termination detection, instead of using synchronization barriers between the synchronous computations. The model allows real-time execution for a number of complex image processing algorithms, including split and merge segmentation, watershed segmentation and motion detection [11].

3. Papers in this issue

This issue is quite representative of the reported evolution, and the large majority of papers covers the topic of general purpose parallel computing for image processing applications.

The first paper is dedicated to the parallel content-based image retrieval (CBIR) systems. This application aims to retrieve images based on the similarity of their content and is a major issue to deal with huge present and future image archives; the demand of parallel implementation, mainly distributed memory implementations, by adopting a cluster of PCs, is highly required. With this aim, the paper *On Parallel Image Retrieval with Dynamically Extracted Features* by Kao [26], reports wavelet and Gabor-based methods for extracting image features. Techniques for the partitioning of the image formation, parallel execution of the queries, and strategies for workload balancing are explained by considering the parallel image database CAIRO as example.

In *Large-Scale Image Sensing by a Group of Smart Image Sensors* by Oh and Aizawa, [40], an experimental large-scale image sensing system is reported; it uses random accessible spatially-variant sampling (SVS) smart image sensors to reduce the data volume at image acquisition level by spatial/temporal resolution reduction schemes, considering the importance of the regions in the scenes. Specifically, the whole traffic of the system is dynamically controlled by sub-sampling an inactive region (IR), i.e. the region without a significant change, at image acquisition level and transmitting IR with lower temporal rate than the active region (AR), the changed region detected by frame difference. The system, implemented in real environment, is shown to preserve system and network resources.

The paper *A Real-Time Full-Body Tracking and Humanoid Animation System*, by Colombo et al. [10] presents a non-intrusive human body motion tracking and its use for avatar animation. The authors propose and describe a system for vision-based tracking of body posture, which relies on network interconnected asymmetric workstations that are in charge of specific tasks in the tracking system. The selected tasks include (a) low-level image processing and user body tracking on both left and right images; (b) stereo analysis and occlusion handling; (c) body pose reconstruction from stereo data via inverse kinematics and (d) virtual character update based on computer graphics. The system is shown to be able to animate a virtual 3D puppet through body movements and to be efficient both in terms of processing time, precision of the 3D reconstruction, tracking and matching in the presence of occlusions and graphic rendering.

A Distributed Genetic Algorithm for Restoration of Vertical Line Scratches by Isgrò and Tegolo [21] faces a typical problem of digital video restoration, i.e. the restoration of vertical line scratches. The problem, computationally intensive over a video of some hours, is solved over a distributed memory system, specifically a network of workstations supporting heterogoneous operating systems. The authors design a distributed algorithm to solve the problem, by adopting an ad hoc Genetic Algo-

rithm. The achieved experimental results show that the algorithm reports reasonably good results in terms of speedups, accelerating the proposed time expensive optimization method.

Concerning with parallel programming environment, the paper by Jonker et al. *Distributed Bucket Processing: a Paradigm Embedded in a Framework for the Parallel Processing of Pixel Sets* [24] reports a programming framework that could allow to consider simultaneously pixel and object based parallelism. They correspond to fundamental mechanisms in parallel processing (data and task parallelism) and both are important for an efficient use of parallel hardware.

The last papers are concerned with multimedia and attentive vision issues.

The paper by Grover et al. [19], reports studies, thanks to a specific simulator, about the data layout schemes that can improve data delivery in a video server. Specifically, it describes the design and performance of the SAM (Storage Area network Multimedia server) which manages a set of parallel storage devices connected by a SAN with the aim to provide multiple concurrent services such as interactive VOD and video games. The goals of SAM are to support for a heterogenous client population with varying buffer sizes, support for both HDTV and low bit-rate videos, support for interactive control operations such as fast forward, rewind, frame advance and slow play, and deterministic or statistic QoS guarantees.

The paper by Gamba et al. [16] reports foveated solutions that could lead to the implementation of efficient vision hardware. Such approach is based on the foveal/multiresolution search guided both by low-level detection, alerting, and tracking schemes and by high-level interpretation processes. It leads to the use of variable resolution grids, according to the image detail required each time, so exploiting the capabilities of multi-resolution and pyramid computer vision systems. The selection of relevant areas typically consists of a parallel activity of simple operators, on the complete field of view, at a low resolution. This pre-attentive phase is implemented on specialized and massively parallel hardware, while at the most sophisticated levels of detail a sequential analysis of selected image segments or ROIs is required to interpret the scene and track its evolution.

4. Conclusions and open issues

There are a number of open issues related to the applications of parallel processing techniques in image and video processing.

The tendency is to require increasingly sophisticated applications, in increasingly small systems. The microprocessors cannot answer simply this double requirement, and their consumption, their volume and their cost are prohibitive for certain applications (videophone on cellular telephone, distributed monitoring, delocalized visual monitoring in robotics, etc). Usually, relatively simple systems, offering performances coarsely comparable with those of the microprocessors, but with an electric consumption and an obstruction compatible with the applications, are required. They will be probably based on pipeline (possibly reconfigurable) operators, or SIMD meshes of small size.

For a certain number of particular applications, the computing power of microprocessors is currently too low. It is, as instance, the case of the vision of the autonomous vehicles, but also of the exploitation of the new methods of video compression in real time. For these applications, the programmability aspect is essential and will guide architectures mainly. These are able to adopt MIMD structures, or SIMD processing with massive parallelism, even operators specialized for certain operations.

In the longer term, the technological development will allow to integrate these functionalities in all the microprocessors. The current processors integrate already a level of significant parallelism, which it is of pipeline, superscalar, of SIMD extension, etc. With the aid of the applications, the structures of parallel processing that we evoked thus probably will be found within the processors. It is not easy to know today which will be the dominating model: MIMD, SIMD, computation structures with circuit or reconfigurable connections, etc., but there is not any doubt that the future of the applications of vision is in parallelism.

In conclusion, we emphasize that image and video processing represent an important practical domain for parallel computation methods. This special issue provides a systematic view of this field, and hopefully, will lead to further advances and concentrated research efforts along the issues outlined above.

References

[1] George H. Barnes, Richard M. Brown, Maso Kato, David J. Kuck, Daniel L. Slotnick, Richard A. Stokes, The ILLIAC IV computer, IEEE Transactions on Computers C-17 (18) (1968) 746–757.
[2] J. Brown, D. Crookes, A high level language for parallel image processing, Image and Vision Computing 12 (2) (1994) 67–79.
[3] W.R. Brown, J. Terzopoulos, Real-Time Computer Vision, Cambridge University Press, 1994.
[4] K.E. Butcher, Design of a Massively Parallel Processor, IEEE Transactions on Computers C-29 (9) (1980) 836–840.
[5] V. Cantoni, M. Ferretti, S. Levialdi, F. Maloberti, A pyramid project using integrated technology, in: Stefano Levialdi (Ed.), Integrated Technology for Parallel Image Processing, Academic Press, 1985, pp. 121–132.
[6] V. Cantoni, M. Ferretti, Pyramidal Architectures for Computer Vision, Plenum Press, New York, 1994.
[7] V. Cantoni, S. Levialdi, Matching the task to a computer architecture, computer vision, Graphical Image Processing 22 (1983) 301–309.
[8] R. Chapuis, J. Serot, D. Ginhac, J. Derutin, Fast prototyping of parallel vision applications using functional skeletons, Journal of Machine Vision and Applications 12 (6) (2001) 271–290.
[9] C.-Y. Chong, S.P. Kumar, Sensor networks: evolution, opportunities, and challenges, Proceedings of the IEEE 91 (8) (2003) 1247–1256.
[10] C. Colombo, A. Del Bimbo, A. Valli, A real-time full body tracking and humanoid animation system, Parallel Computing 34 (12) (2008) 718–726, this issue.
[11] J. Denoulet, A. Merigot, An architecture based on reconfigurability and asynchronism for real-time image processing, Journal of Real-Time Imaging 3 (3) (2008) 119–130.

[12] Digital Equipment Corp. Alpha 21164PC (PCA57) Microprocessor Hardware Reference Manual, Rapport Technique EC-RADPA-TE, December 1998.
[13] B. Ducourthial, A. Mérigot, Parallel asynchronous computations for image analysis, Proceedings of IEEE 90 (7) (2002) 1218–1229.
[14] M.L.B. Duff, CLIP4, a large scale integrated circuit array parallel processor, in: Proceedings 3rd Int. Conf. on Pattern Recognition, 1976, pp. 728–733.
[15] T. J Fountain, K.N. Matthews, M.J. Duff, The CLIP7A image processor, IEEE Transactions on Pattern Analysis and Machine Intelligence 10 (3) (1988) 310–319.
[16] P. Gamba, L. Lombardi, M. Porta, Log-map analysis, Parallel Computing 34 (12) (2008) 757–764, this issue.
[17] A. van Gemund, A. Radulescu, C. Nicolescu, P.P. Jonker, Mixed task and data parallel scheduling for distributed systems, in: Proceedings of the 15th International Parallel and Distributed Symposium, San Francisco, California, 2001.
[18] D. Ginhac, J. Serot, J. Derutin, Skipper: a skeleton-based parallel programming environment for real-time image processing applications, in: 5th International Conference on Parallel Computing Technologies, LNCS, vol. 1662, 1999, pp. 296–305.
[19] R.S. Grover, Q. Li, H.-P. Dommel, Performance study of data layout schemes for a SAN-based video server, Parallel Computing 34 (12) (2008) 747–756, this issue.
[20] L.G.C. Hamey, J.A. Webb, I-C. Wu, An architecture independent programming language for low-level vision, Computer Vision, Graphics, and Image Processing 48 (2) (1989) 246–264.
[21] F. Isgrò, D. Tegolo, A distributed genetic algorithm for restoration of vertical line scratches, Parallel Computing 34 (12) (2008) 727–734, this issue.
[22] Intel Corp., Intel Architecture Software Developer Manual, Rapport Technique 243190, 1999.
[23] L.H. Jamieson, E.J. Delp, J. Li, C.-C. Wang, F.J. Weil, A software environment for parallel computer vision, IEEE Computer 25 (2) (1992) 73–75.
[24] P.P. Jonker, J.G.E. Olk, C. Nicolescu, Distributed bucket processing: a paradigm embedded in a framework for the parallel processing of pixel sets, Parallel Computing 34 (12) (2008) 735–746, this issue.
[25] Z. Juhasz, An analytical method for predicting the performance of parallel image processing operations, The Journal of Supercomputing 12 (1/2) (1998) 157–174.
[26] O. Kao, On parallel image retrieval with dynamically extracted features, Parallel Computing 34 (12) (2008) 700–709, this issue.
[27] D. Koelma, A software architecture for application driven high performance image processing, in: Proc. Par. Dist. Methods for Image Processing, 1997, pp. 340–351.
[28] C. Lee, M. Hamdi, Parallel image processing applications on a network of workstations, Parallel Computing 21 (1) (1995) 137–160.
[29] R.B. Lee, Subword parallelism with MAX-2, IEEE Micro 16 (4) (1996) 51–59.
[30] R.M. Loughead, D.L. Mac Cubbrey, The cytocomputer: a practical pipeline image processor, in: Proc. 7th Annual Symposium on Computer Architecture, 1980, pp. 217–277.
[31] MedGrid, <http://www.creatis.insa-lyon.fr/MEDIGRID/>.
[32] A. Mérigot, P. Clermont, J. Méhat, F. Devos, B. Zavidovique, A pyramidal system for image processing, in: Virginio Cantoni, Stefano Levialdi (Eds.), Pyramidal Systems for Image Processing and Computer Vision, NATO ASI Series, Springer-Verlag, 1986, pp. 109–124.
[33] A. Mérigot, Associative nets: a graph-based parallel computing model, IEEE Transactions on Computers 46 (5) (1997) 558–571.
[34] J. Montagnat, V. Breton, I. Magnin, Using grid technologies to face medical image analysis challenges, in: First International Workshop on Biomedical Computations on the Grid (Biogrid'03), Proceedings of the IEEE CCGrid03, Tokyo, Japan, 588-59, 2003.
[35] M.S. Moore, A model-integrated program synthesis environment for parallel/real-time image processing, in: Proceedings of Par. Dist. Methods for Image Processing, 1997, pp. 31–45.
[36] P.J. Morrow et al, Efficient implementation of a portable parallel programming model for image processing, Concurrency: Practice and Experience (11) (1999) 671–685.
[37] P.J. Morrow, D. Crookes, P.J. McParland, Ial: a parallel image processing programming language, IEE Proceedings, Part I 137 (3) (1990) 176–182.
[38] Motorola Inc., AltiVec Technology Programming Environment Manual, November 1998.
[39] J.M. Nick, I. Foster, C. Kesselman, S. Tuecke, The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, 2002.
[40] M. Oh, K. Aizawa, Large-scale image sensing by a group of smart image sensors, Parallel Computing 34 (12) (2008) 710–717, this issue.
[41] A. Peleg, U. Weiser, MMX technology extension to the Intel architecture, IEEE Micro 16 (4) (1996) 42–50.
[42] C. Rieger, R. Trigg, R. Bane, ZMOB A new computing engine for AI, Computer Science Tech. Rep. TR-1028. Univ. Maryland, College Park, March 1981.
[43] G.X. Ritter, J.N. Wilson, Handbook of Computer Vision Algorithms in Image Algebra, CRC Press, Inc., 1996.
[44] F.J. Seinstra, D. Koelma, User transparency: a fully sequential programming model for efficient data parallel image processing, Concurrency and Computation: Practice and Experience 16 (2004) 611–644.
[45] J. Serot, D. Ginhac, Skeletons for parallel image processing: an overview of the skipper project, Journal of Parallel Computing 28 (2002) 1685–1708.
[46] J.A. Steele, An abstract machine approach to environments for image interpretation on transputers, Ph.D. thesis, Fac. Science, Queens University of Belfast, N. Ireland, 1994.
[47] R.-I. Taniguchi et al., Software platform for parallel image processing and computer vision, in: Proceedings Parallel and Distributed Methods for Image Processing, 1997, pp. 2–10.
[48] M. Tremblay, J.M. O'Connor, V. Narayanan, L. He, VIS speeds new media processing, IEEE Micro 16 (4) (1996) 10–20.
[49] S. Unger, A computer oriented toward spatial problems, Proceedings of IRE 46 (1958) 1744–1750.
[50] J.A. Webb, Steps toward architecture independent image processing, Computer 25 (2) (1992) 21–31.