Fractal Image Compression on MIMD Architectures II: Classification Based Speed-up Methods*

Jutta Hämmerle and Andreas Uhl

RIST++ & Department of Scientific Computing, University of Salzburg, Austria

Since fractal image compression is computationally very expensive, speed-up techniques are required in addition to parallel processing in order to compress large images in reasonable time. In this paper we discuss parallel fractal image compression algorithms suited for MIMD architectures which employ block classification as speed-up method.

1. Introduction

Fractal image compression [9, 10, 27, 31] has generated much interest in the image compression community as possible competitor [11] to well established compression techniques (e.g. DCT-JPEG) and to newer technologies (e.g. wavelets). Additionally, hybrid techniques partially based on fractal coding have evolved [3, 4, 41]. One of the main drawbacks of conventional fractal image coding is the high encoding complexity (whereas decoding complexity is much lower) as compared to e.g. transform coding. On the other hand, fractal image compression offers interesting features like resolution-independent and fast decoding, and good image quality at low bit-rates which makes it an interesting candidate for off-line applications (e.g. video-on-demand (VoD), photo or video CD-ROMs, etc.). Additionally, basic building blocks of fractal compression technology may be used in many other fields, like e.g. feature extraction [39], image watermarking [36], and motion compensation in video coding [35].

However, speed-up techniques are necessary in order to accelerate the encoding phase of fractal compression. Two different approaches can be distinguished:

- Sequential techniques [38],
- High Performance Computing.

In this work we discuss parallel fractal image compression algorithms suited for MIMD architectures which additionally employ the sequential speed-up method "block classification" in order to speed up the computation furthermore. It should be noted that the algorithms discussed in this work are not restricted to fractal compression only, but may be applied to all types of applications where block matching operations are involved (like e.g. vector quantization or block-based motion estimation techniques).

2. Fractal Image Compression

Fractal image compression exploits similarities within images. These similarities are described by a contractive transformation of the image whose fixed point is close to the image itself. In our implementation the image transformation consists of block transformations which approximate smaller parts of the image by larger ones, using contractive affine transforms. The smaller parts are called ranges and the larger ones domains. All ranges together (range pool) form

* This work was partially supported by Österreichische Nationalbank, Jubiläumsfonds project no. 6900 and was presented in part at the Fourth International Conference of the ACPC (ACPC'99), Salzburg, February 1999.

a partition of the image. The domains (having twice the height and length of the ranges) can be selected freely within the image and may overlap. For each range an appropriate domain must be found. The domain blocks are transformed to match a given range block as closely as possible.

To compare image blocks (i.e. range and domain) usually rms (root mean square error) is used. The transformation applied to the domain classically consists of the following parts:

- Geometrical contraction (usually implemented by averaging the domain).
- Affine motion (modeled by using the 8 isometries of the square block).
- Gray value adaptation (a least square optimization is performed in order to determine the best values for the parameters describing contrast and brightness modification, respectively).

In a non-adaptive algorithm for each range the block transformation with the smallest rms-error becomes part of the image transformation - no matter how well the range can be covered. In an adaptive algorithm the error of a single block transformation is compared to a predefined maximum error. If the calculated error is larger the specific range is split into smaller blocks which are then covered independently or coded directly in the case of the lowest partition level. So – in contrast to the non-adaptive algorithm – a specific image quality can be guaranteed. As a typical example for this kind of algorithm serves the “adaptive quadtree algorithm” [9, 31].

The search for appropriate transformations is computationally enormously expensive and causes the need for an acceleration of the encoding speed.

In literature, several sequential complexity reduction schemes are presented (for a summary of such techniques see [38]). In this paper we focus on discrete feature extraction methods often also denoted as block classification (see e.g. [9, 28, 29]). The domain pool is divided into several classes according to specific discrete features within the domain blocks. The features of the range determine the class in which optimal matches should be located (i.e. only within this single class the search for the optimal match is performed). The use of discrete feature extraction methods usually slightly reduces the achieved image quality.

3. Fractal Image Compression on MIMD Computers

Fisher [9] points out that the simplest way to implement a fractal encoder in parallel is to assign a piece of the image to each processor element (PE). When running this algorithm without communication after the distribution of the image this corresponds to the use of a smaller domain pool (a localized codebook) and leads to different results (in most cases worse [18]) as compared to the sequential algorithm.

There has already been done a significant amount of work on implementing the encoding phase of the fractal image coding technique on different high performance computing architectures: A reduced version of the algorithm without its adaptive features has been implemented on a pyramidal SIMD architecture [44] using a pixel-based parallelization. An image-block-based parallelization on a SIMD array processor is used for the calculations in [13], several different block-based algorithms for this type of architecture have been discussed and compared in [22, 23]. VLSI chip sets for fractal encoding [21, 6, 7, 8] have been developed as well as parallel algorithms for fractal video coding [34]. A parallel algorithm for the decoding phase has also been developed [33].

A large amount of work has also been already devoted to fractal compression algorithms suited for MIMD architectures preserving sequential coding quality. In a series of papers we identify two major groups of algorithms suited for MIMD computations which trade off memory versus communication demand and are therefore applied according to the relation between memory capacity of one PE and image size [16, 15, 17, 42]:

- **Algorithms Class A** (parallelization via ranges): Within this class it is necessary that at least the entire image can be stored in the memory of each PE. Out of the image-data the complete domain pool can be produced. To each PE a subset of the range pool is assigned – either statically or dynamically – and the PE calculates the best transformations for its range subset. Algorithms of this class may include a dynamic load balancing [15]. According to memory capacities and communication costs further distinctions can be made. For more details see e.g. [14, 26, 32, 43, 45].

- **Algorithms Class B** (parallelization via domains): Within this class of algorithms the domain pool cannot be stored in the memory of one PE - therefore the domain pool is distributed evenly among the PEs. The most efficient algorithms transfer the ranges among the PEs in a pipelined manner in order to carry out the range-domain comparisons of all sub-domain pools. According to synchronization levels and available architectures further distinctions can be made. For more details see e.g. [2, 24, 25, 30].

Depending on the memory capacities of the PEs either algorithms of class A or B are applied. The change from class A to B has to be paid with the de-facto loss of dynamic load balancing possibilities and/or a much higher communication demand. As a consequence, algorithms of class A show a much better scalability.

Therefore, algorithms of class A are well suited for execution on multiprocessors whereas on multicomputers the most suitable algorithm is chosen according to possible memory restrictions (of course preferably out of class A).

4. Classification Based Speed-up on MIMD Computers

The block-classification of the domain pool needs to be performed prior to the actual compression. Therefore, it may be executed in sequential or parallel mode, no matter which class the subsequent fractal compression algorithm belongs to. Parallel algorithms of class A are very efficient and show nearly linear speedup in the order of the number of processors. They may include dynamic load balancing, and the number of PEs can be larger than with algorithms of class B before a bottleneck in the host-process occurs [42]. Therefore, this class is a good basis for combining parallel algorithms and sequential speed-up techniques in an efficient way. When processing large images on distributed memory architectures, unfortunately these algorithms cannot be applied due to memory restrictions. Consequently, the domain pool has to be distributed among the PEs (algorithms of class B). Algorithms of this type can be distinguished according to whether the assignment of parts of the domain pool to each PE is done prior to or after any precalculations. In the first case the domain pool is distributed evenly among the PEs (fixed distribution), in the second case the distribution is performed according to the results of the precalculations (adaptive distribution).

4.1. Parallel Classification

Parallel precalculations for classification methods with predefined classes include the following steps:

1. Distribution of the domain pool (and possibly also the range pool) to the PEs in a way that each PE receives an equal sized part. Each PE receives either the entire pool but uses only a part for precalculations or receives only a part.
2. On each PE: Precalculations for specific sub-domain pool.
3. Redistribution of the domain pool including results of precalculations (or only results if the nodes already possesses the whole pool). This redistribution can either be organized by a host process or by the PE themselves.

Classification methods with image-adapted classes include a kind of image-adapted clustering process. In literature – for example – Kohonen's self-organizing maps [20] or LBG codebook generators [29] are used. This clustering process must be done prior to the classification process and is not discussed here. The steps following these procedures are the same as for methods with predefined classes.

The following parameters determine the efficiency of parallel precalculations: number of

PEs, speed of PEs, communication cost, complexity of precalculations, size of the image, and amount of overlap of the domain pool.

The more PEs are available the more time is wasted if all PEs perform all precalculations. But the results must be redistributed after the precalculations for most of the algorithms discussed. This induces that either each node must communicate with each other (which has a complexity of $O(p^2)$, p=number of PE) or a host process must collect and redistribute the results which might cause a bottleneck in the host. Therefore, the complexity of the entire precalculations on a single PE (also determined by the size of the image) must be compared with the communication cost of distributing values to the PE, complexity of calculations on the PE, and redistributing the results. It turns out that it is often more efficient to perform the precalculations sequentially, since the small computational effort does not justify the parallelization effort (e.g., this is true for the classification applied in our experiments).

4.2. Classification and Class A Algorithms (CA)

The entire image (and consequently the entire domain pool) is residing on each PE. After the precalculation step the ranges may be distributed among the PEs and processed independently. The distribution can be performed dynamically or statically – also, the optimal static distribution may be performed for each quadtree level [15]. This is done similarly to the technique C2B introduced in the next section – knowledge about the class structure of both domain **and** range pool is required for this task.

It should be noted that the communication effort of class CA algorithms remains constant as compared to that of class A algorithms without classification during the fractal encoding itself. On the other hand, computations are reduced significantly for each range (by a factor of $1/c$ in the ideal case, if c denotes the number of classes) for class CA algorithms. Therefore, the choice of a suitable communication structure and load balancing scheme is even more important for class CA algorithms.

4.3. Classification and Class B Algorithms (CB)

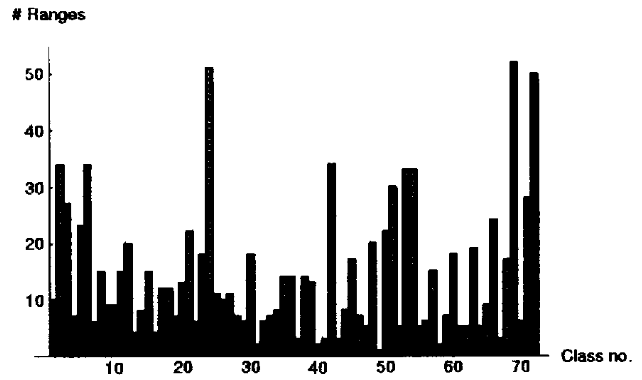
Algorithms C1B – Fixed Distribution

The domain pool is distributed evenly among the PEs. Each PE performs necessary precalculations (i.e. averaging, classification) for its private domain pool. Consequently, parallel precomputations may be performed in this case without the need of subsequent data redistribution. If a range is processed by a PE it is first classified and the corresponding class of the private domain pool is being searched. Subsequently, the range is passed on to the next PE in the pipeline.

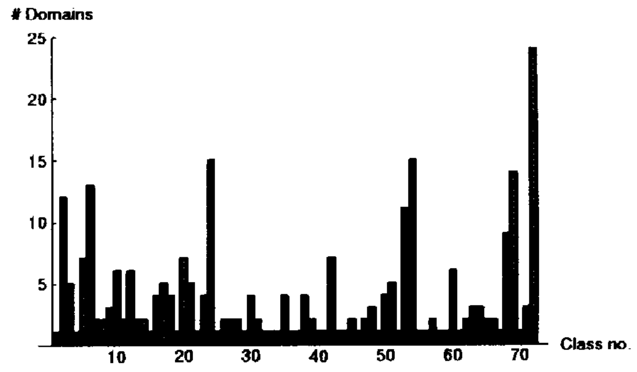
For a specific range a specific class provides domains for an optimal match and only this class is being searched. The sizes of the classes in the private domain pools on the PEs will generally be very different for different PEs. Therefore, the processing time for a given range might be very different on different PEs, which leads to a severe disturbance of the pipelined execution.

Consider the following bad case: On each PE exists one class which contains far more domains of the private pool as compared to the other classes. This large class may be different for different PEs. A PE on which a range is classified into the large class has a significantly higher processing time than most other PEs. This effect may occur for almost each range at different time instances. Since the "slowest" range determines the speed of the pipelined execution the speed-up is limited and - in a worst case - equal to or even smaller than the speed-up of the parallel algorithm without additional sequential speed-up techniques.

The requirement for optimal speed-up concerning discrete feature extraction methods is therefore a uniform size distribution of the domain classes on each PE. The size distribution of the domain classes can be controlled to some extent if discrete feature extraction methods with image adapted classes are being used – if fixed or heuristic classes are used a different strategy is needed for avoiding the load balance problem. However, if fixed distribution is used, it is important not to execute the pipeline in synchronized manner – the unsynchronized pipelined processing mode is capable of compensating the load balancing problem to some extent.



(a) ranges



(b) domains

Fig. 1. Number of ranges and domains for all classes of the first quadtree level (Lena image).

Algorithms C2B – Adaptive Distribution

For this type of algorithms it is necessary to have knowledge about the class structure and size of the classes of both range and domain pool, respectively. The precalculations are done prior to any adaptive data distribution. Contrary to the fixed distribution we do not perform a distribution of the domain pool itself but we distribute the classes among the PEs – both domains **and** ranges of a given class are assigned to a given PE which calculates the optimal match. This means that we neither apply data parallelism via ranges (class A) nor via domains (class B) but via classes.

An optimal distribution of the classes among the PEs therefore depends on:

- Class structure of range pool (i.e. number of ranges in each class),
- Class structure of domain pool,
- relation $\frac{c}{p}$ (c =number of classes, p =number PEs).

Distributing the classes evenly among the PEs would lead to a severe load balancing problem, since

a) the size of the classes (i.e. the number of domains in the classes) is generally very different and

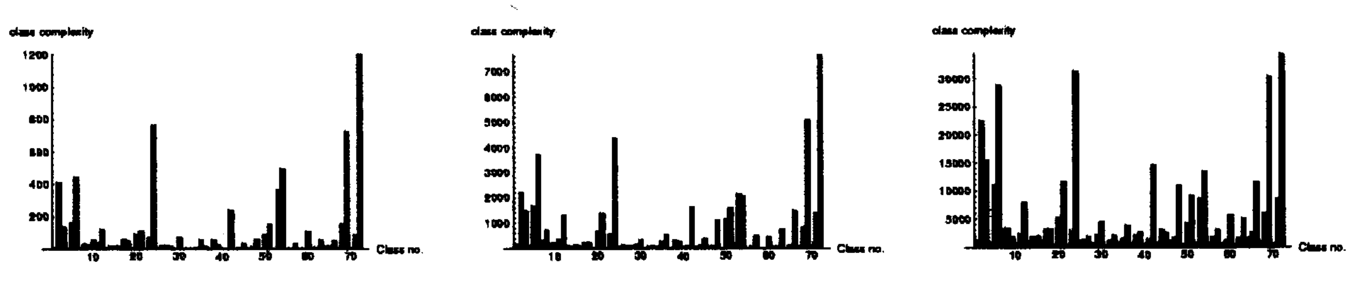
b) the amount of computation inside one class is again very different, since a different number of ranges belongs to each class.

According to the precalculations, we know the number of ranges and domains belonging to a given class (for example see Figure 1).

The use of image-adaptive classes with uniform class sizes would save the computation of the class complexities at the cost of an even more expensive adaptive class generation. However, the number of range-domain comparisons within a class can be determined in any case – we denote this number as "class complexity" (see Figure 2). The total of all class complexities gives the overall complexity of the algorithm. Obviously, we have to distribute the class complexities evenly among the PEs.

These observations lead to the following optimization problem: Given the number of PEs and the precalculated class complexities, distribute the class complexities as uniformly as possible among the PEs. This type of optimization can be interpreted as simple knapsack problem with several knapsacks of equal size. An additional constraint of this optimization may be the memory capacity of the PEs – in the case of very small capacity it may not be possible to distribute the class complexities at once but a step-by-step assignment (possibly dynamically) might be necessary.

An additional issue to be considered is that the calculated distribution of class complexities is valid only for the first quadtree level of the adaptive quadtree decomposition. The distribution of classes of the domain pool of the subsequent quadtree levels among the PEs is different from the distribution of the first level. (For a comparison of class complexity distribution of



(a) first level (b) second level (c) third level

Fig. 2. Class complexities of different quadtree levels (Lena image).

three quadtree levels see Figure 2). Therefore, a class complexity calculation and distribution is required at each quadtree level. These procedures involve a synchronization and centralization step at each quadtree level which limits parallelization efficiency.

Preferably, entire classes are distributed among the PEs according to their calculated complexities. However, this technique may lead to a very unbalanced load distribution (large class complexities may occur, which significantly exceeds one PE's quota, see e.g. [19, p.283]) and is restricted to the case *#PE* $\leq$ *#classes*. In order to overcome these restrictions we must allow the distribution of the class complexity of each class. The decision whether a class complexity is distributed or not is controlled by a tolerance parameter which specifies the allowed deviation from the uniformity of the final complexity distribution. Theoretically, a completely uniform distribution is achievable by setting this parameter to zero. But in this case communication and memory demand rises since many small complexities (the minimal size in our case is one range and the domain pool of one class) have to be distributed.

Although sophisticated techniques may be used for distributing the class complexities among the PEs, the following simple technique has proven to be sufficiently precise. The class complexities are sorted and subsequently assigned to the PEs starting with the largest one. If a class complexity exceeds one PE's computed quota, it is assigned to the PE with the next lower complexity level, and so on. If a class complexity exceeds all PE's quotas, it is assigned to the PE with the current lowest complexity level. The amount of complexity exceeding the PE's quota is assigned again to the set of not yet distributed class complexities. This is done by assigning all domains of this class to the PE, whereas only the corresponding share of ranges is assigned to it. Unfortunately, this technique has to be paid with a slight increase of memory demand on the PEs, since the domains of a single class may be assigned to different PEs for several times.

This technique is applied until all class complexities have been distributed. Depending upon the choice of the tolerance parameter, we accomplish a more or less uniform load distribution.

Note that if for any reason dynamic load balancing is required (e.g. due to changing load conditions in multi-user environments or memory constraints) it is straightforward to create a task pool of any desired structure using the technique described above. We want to emphasize that load balancing is not possible in a sensible way in the case of a fixed distribution.

A final remark on memory requirement: in memory critical systems the fixed domain pool distribution is done by distributing the image itself evenly. On each PE the necessary domains are calculated from this image part, which may dramatically reduce memory demand in return for increase of computational demand if an overlapping domain partition is used. In such a case the described technique of precomputed adaptive distribution is not successful since the technique "higher complexity for lower memory" is not applicable (since the domains of one class will generally be distributed over the entire image). Therefore, precomputed adaptive distribution is restricted to non-overlapping or "little-overlapping" domain partitions (which is a must for large images anyway). For domain partitions with a higher amount of overlap class complexities are distributed in a dynamical way (see previous paragraph) according to the available memory resources on the single PEs.

(a) Lena

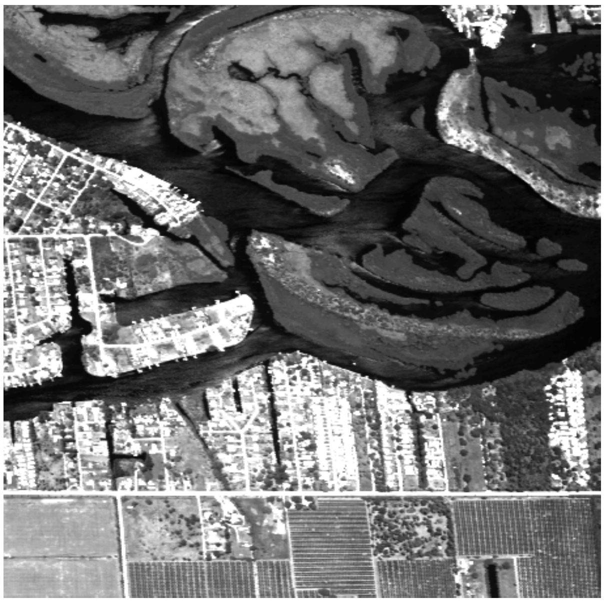

(b) Satellite image

Fig. 3. Test images with 8bpp.

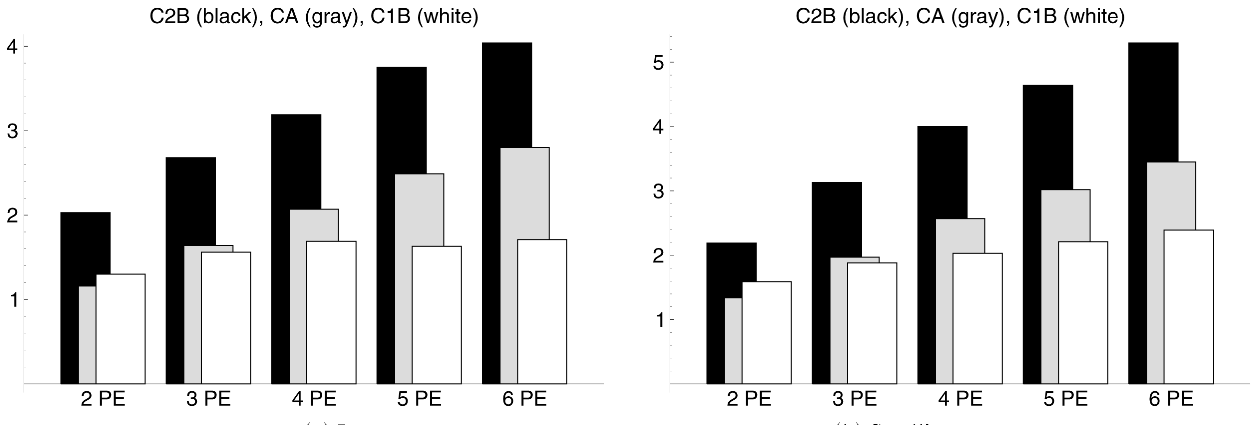
5. Experimental Results

We apply an adaptive quadtree algorithm (starting with range-size 16 $\times$ 16 pixels we partition until size 4 $\times$ 4 is reached, subsequently the range is coded directly) and employ a greyvalue classification scheme which is introduced by Fisher [9]. This scheme uses the ordering of mean and variance values in the four blockquadrants as classification feature and results in 72 classes. This is of course only one possibility for a discrete feature extraction method. Many other techniques exist, which may result in an either higher or lower number of classes (see e.g. [28, 29]). The domains in the domain pool are chosen to be non-overlapping (as it is done in most fractal coders in order to keep the complexity reasonably low) or overlapping with offset 4 (for ensuring higher quality [12]). The necessary precalculations (i.e. domain averaging and classification) are not performed in parallel due to reasons explained earlier (depending on image content precomputations require only 1% or less of the entire sequential processing time).

It should be noted that the validity of our algorithmic considerations is neither affected in principle by the choice of the concrete classification technique nor by the image partitioning method employed (the applicability of the algorithms discussed is, of course, bound to the existence of discrete feature extraction methods for e.g. quadtree, HV, or triangulation [5] based partitioning). Therefore, our results do carry over to a wide range of different algorithms.

The following images with 8bpp are used: the well known Lena image (1024 $\times$ 1024 pixels) and a satellite image (1024 $\times$ 1024 and 2048 $\times$ 2048 pixels) (see Fig. 3.a and 3.b, respectively).

For our experiments we use a FDDI interconnected NOW consisting of 8 DEC AXP 3000/ 400 workstations and a SGI POWERChallenge GR with 2.5 GB memory and 20 MIPS R10000 processors. PVM [40] is employed for implementing a message passing based host/node configuration whereas we use PowerC [1] for shared memory programming (the corresponding algorithm is denoted "PC"). In shared memory programming a sequential algorithm may be transformed into a parallel one by simply identifying areas which are suitable to be run in parallel i.e. in which no data dependencies exist. Subsequently, only local and shared variables need to be declared and parallel compiler directives are inserted. Therefore, shared memory programming can be performed very quickly. On the other hand, message passing requires an explicit programming of each communication event occurring among processors and is consequently very time demanding. However, message passing programs written in e.g. MPI or



(a) Lena (b) Satellite

Fig. 4. Overall Comparison for 1024 × 1024 pixel images on NOW, non-overlapping domain pool.

PVM may be used without changes on different architectures (no matter if multiprocessors or multicomputers) whereas shared memory programming always uses a native programming language.

The number of PEs as depicted in the plots refers to the number of node processes, the vertical axis shows speed-up relative to sequential execution. The host process is executed on a different PE and is not counted (except in the case of algorithm PC where no explicit host process with different functionality exists). If not declared otherwise, algorithm CA is executed with dynamic load balancing using single ranges as tasks and algorithm C2B allows the distribution of class complexities of single classes to a certain extent.

We start with the results of the message passing algorithms suited for multicomputers. Fig. 4 shows a comparison of algorithms CA, C1B, and C2B using non-overlapping domain pool on the NOW. For both images considered there is a fixed and clear ranking: C2B performs best, C1B worst, and CA is in-between. The relation between algorithms CA and C1B is in complete accordance with the case without classification based speed-up [42], but the absolute performance of both algorithms is very poor. Whereas we easily achieve almost linear speed-up for algorithms of class A and at least logarithmic speed-up for algorithms of class B, here speed-up for algorithm CA is very moderate (e.g. 2.6 or 3 using 6 node PEs) and speed-up for algorithm C1B is saturated at 3 node PEs for both images considered (at a value of 1.5 or 1.9, respectively). This effect is due to the significant reduction of computational effort required for algorithms using classification – since the communication amount is not reduced, efficiency (and, as we shall see later, scalability as well) is diminished. On the other hand, algorithm C2B shows considerably better results for both images and especially its relation to C1B entirely conforms with the theoretical considerations. However, it should be noted that algorithm C2B may be "emulated" under algorithm CA conditions (i.e. enough memory to keep the entire image on the node PEs). This leads to an even improved performance since only index-information about the distribution of classes needs to be transferred to the node PEs instead of the data itself.

Obviously, the dynamic task distribution approach in algorithm CA is not competitive in this setting. Therefore, different strategies have been investigated (which have been discussed before for class A algorithms [15]). Whereas a static distribution of 30% or 50% of all ranges and subsequent dynamic range distribution as applied before does not lead to an improvement of the results, sensible static distribution of all ranges sometimes does. Let n denote the number of PEs, r the number of ranges, and $s = r/n$. In the case of block distribution PE$_i$ gets ranges R_{is}, R_{is+1}, R_{is+2}, ... (contiguous image blocks are assigned to the node PEs). This is what would probably be done in an SPMD approach. This strategy behaves very poorly, as it can be seen in Fig. 5.a. The reason is the following: the computation load can be extremely unbalanced if the image has different characteristics in various areas; e.g. variance. If one PE receives many smooth blocks that can be encoded with-

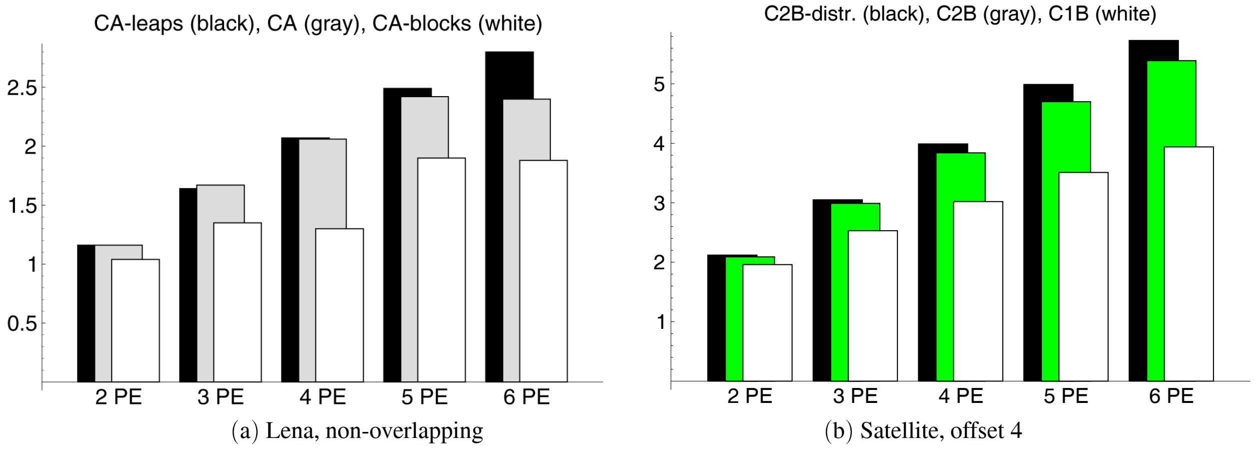


Fig. 5. Different load distribution strategies for CA and the effect of distributing complexity classes for C2B applied to 1024 × 1024 pixel images on NOW.

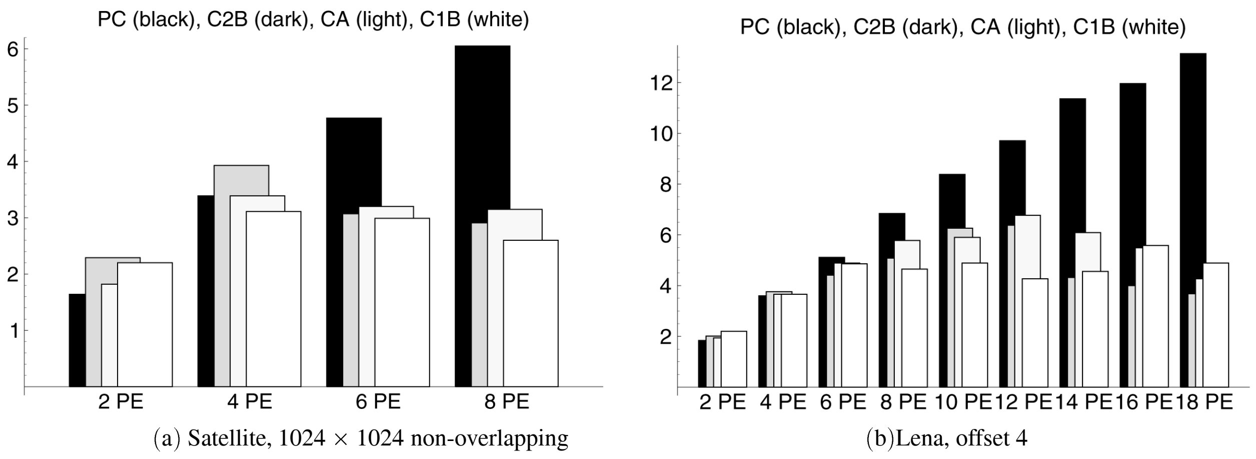


Fig. 6. Scalability of different algorithms for 1024 × 1024 pixels images on the SGI.

out further splitting and another one receives many blocks that have to be split several times, the loads of these two PEs differ significantly.

In the case of leap distribution PE$_i$ gets ranges R_i, R_{i+n}, R_{i+2n}, … (the blocks are assigned to the PEs in interleaved manner and are therefore distributed across the entire image). This strategy leads to a slight improvement of the results of CA using 5 and 6 node PEs, as can be observed in Fig. 5.a. (Note, that in Fig. 4.a algorithm CA is used with leaps.) Of course, the success of this distribution strategy depends on the structure of the image considered. Also, the application of some sort of precalculation (e.g. variance calculation) in order to determine the amount of splitting required fo each range fails, since the class structure of the domain pool is not considered.

In Fig. 5.b we display results concerning algorithms CB only. Note that although performing worst again, the performance of algorithm C1B is improved as compared to Fig. 4. This is caused by the higher amount of computation required for a domain pool with offset 4. The effect of distributing the class complexity of single classes is shown here as well. We notice a small gain in the case of enabling this type of distribution. (The effect would of course be more spectacular if more PEs and a classification with less classes is employed.)

Before discussing the results on the multiprocessor, note the fact that the speed of the PEs of the SGI is about a factor of 4 higher as compared to the PEs of the DEC AXP workstations in the NOW. Fig. 6 shows results for 1024 × 1024 pixels images on the SGI.

Contrasting to the results seen before, we notice a saturation and even decrease of speed-up. For

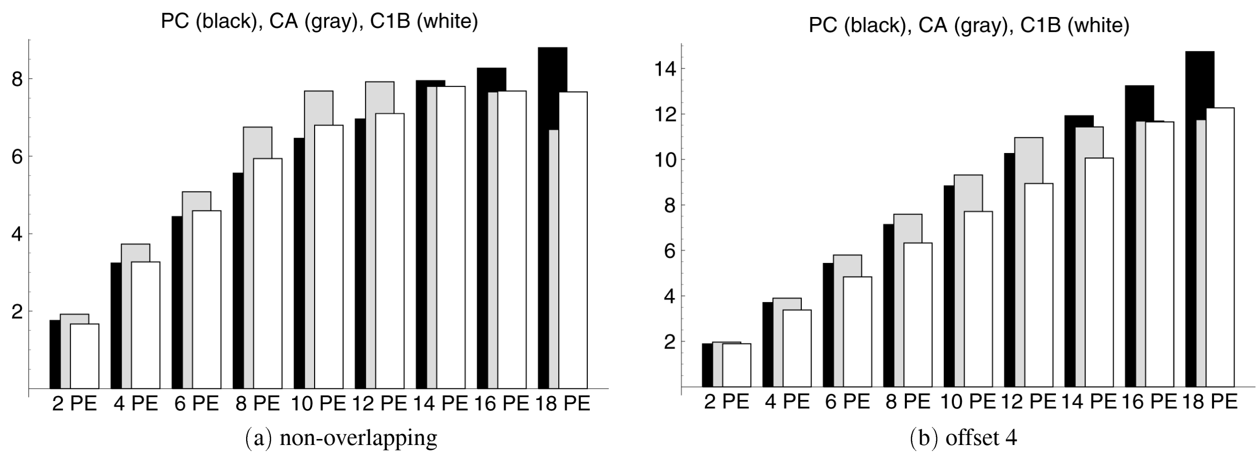


Fig. 7. Scalability of different algorithms for the 2048 × 2048 pixels satellite image on the SGI.

a non-overlapping domain pool (Fig. 6.a) the peak in speed-up for the message passing algorithms is reached at 4 PEs, however, their relative behaviour is preserved to some extent. For 6 PEs and more algorithm PC is significantly more efficient and exhibits good scalability. For a domain pool with offset 4 (Fig. 6.b) a similar but less pronounced tendency may be observed. Obviously, the small overall amount of computations as compared to communication leads to a bottleneck in the host-node configuration very soon. It is interesting to see that algorithm C1B shows almost constant speed-up for 6 PEs upwards whereas algorithms CA and C2B reach a peak at 12 PEs and loose speed-up subsequently.

Finally, we present results for the 2048 × 2048 pixels satellite image on the SGI (Fig. 7). At first sight it is clear that the higher amount of computation required for the large image significantly improves the efficiency and scalability of the message passing algorithms. This effect may be observed as well when comparing non-overlapping and offset 4 domain pools (Fig. 7.a and Fig. 7.b, respectively).

Additionally, the enormous advantage of shared memory programming (algorithm PC) is reduced. However, for a high number of PEs algorithm PC is still best performing. For both domain pool settings a tendency already observed in Fig. 6.b is clearly visible: Whereas algorithm CA is superior for a lower number of PEs, algorithm C1B is performing better using 16 and 18 PEs. This effect is due to the communication structure of algorithm CA where the dominant host process causes a bottleneck at higher PE numbers. On the other hand, the pipelined execution mode of algorithm C1B can cope with this situation better.

6. Conclusion and Future Research

In the discussion of parallelization approaches for classification accelerated fractal image compression algorithms we have seen that all kinds of message passing algorithms show a limited scalability and should therefore be restricted to moderate parallel multicomputers. Within this class of algorithms a parallelization via classes has turned out to be most efficient (algorithm C2B), followed by parallelization via ranges (CA), and domains (C1B). However, algorithm C1B shows the best scalability due to the lack of a dominant host process. On multiprocessors, shared memory programming is clearly superior to message passing if a large number of PEs is employed, especially in the case of low overall computational demand.

Future research will be conducted on how to employ multidimensional nearest neighbour search techniques [37] (which is currently the most efficient sequential speed-up technique for fractal encoding) within parallel algorithms of class A and B.

References

[1] Barr E. Bauer, *Practical Parallel Programming*, Academic Press, New York, NY, USA, 1992.

[2] S. K. Chow and S. L. Chan, A design for fractal image compression using multiple digital signal

processors, In *Proceedings of the International Picture Coding Symposium (PCS'96)*, pages 303–308, Melbourne, March 1996.

[3] G.D̃AVIS, Self-quantized wavelet subtrees: A wavelet-based theory for fractal image compression, In J. A. Storer and M. A. Cohn, editors, *Proceedings Data Compression Conference (DCC'95)*, pages 232–241, IEEE Computer Society Press, March 1995.

[4] G. DAVIS, A wavelet-based analysis of fractal image compression, *IEEE Transactions on Image Processing*, 7(2):141–154, February 1998.

[5] F. DAVOINE, M. ANTONINI, J-M. CHASSERY, AND M. BARLAUD, Fractal image compression based on delauney triangulation and vector quantization, *IEEE Transactions on Image Processing*, 5(2):338–346, 1996.

[6] O. FATEMI AND S. PANCHANATHAN, VLSI chip set for affine based video compression, In V. Bhakskaran, S. Panchanathan, and F. Sijstermans, editors, *Digital Video Compression: Algorithms & Technologies 1996*, volume 2668 of *SPIE Proceedings*, pages 233–242, 1996.

[7] O. FATEMI AND S. PANCHANATHAN, Fractal engine, In S. Panchanathan and F. Sijstermans, editors, *Multimedia Architectures 1997*, volume 3021 of *SPIE Proceedings*, pages 88–99, 1997.

[8] O. FATEMI AND S. PANCHANATHAN, Fractal engine: An affine video processor core for multimedia applications, *IEEE Transactions on Circuits and Systems for Video Technology*, 8(7):892–908, November 1998.

[9] Y. FISHER, EDITOR, *Fractal Image Compression: Theory and Application*, Springer-Verlag, New York, 1995.

[10] Y. FISHER, EDITOR, *Fractal Image Encoding and Analysis*, volume 159 of *NATO ASI Series, Series F: Computer and Systems Sciences*, Springer-Verlag, Berlin, Heidelberg, New York, Tokyo, 1998.

[11] Y. FISHER, T. P. SHEN, AND D. ROGOVIN, A comparison of fractal methods with DCT (JPEG) and wavelets (EPIC), In *Neural and Stochastic Methods in Image and Signal Processing III*, volume 2304-16 of *SPIE Proceedings*, San Diego, CA, USA, July 1994.

[12] TH. FREINA AND A. UHL, Hybrid image coding using fractal prediction, In *Proceedings of the International Picture Coding Symposium (PCS'99)*, pages 133–136, Portland, Oregon USA, April 1999.

[13] S. GIORDANO, M. PAGANO, F. RUSSO, AND D. SPARANO, A novel multiscale fractal image coding algorithm based on SIMD parallel hardware, In *Proceedings of the International Picture Coding Symposium (PCS'96)*, pages 525–530, Melbourne, March 1996.

[14] M. GUGGISBERG, I. PONTIGGIA, AND U. MEYER, Parallel fractal image compression using iterated function systems, Technical Report Technical Report CSCS-TR-95-07, Swiss Scientific Computing Center, May 1995.

[15] J. HÄMMERLE, Load balancing strategies for parallel fractal image compression, In R. Wyrzykowski, H. Piech, B. Mochnacki, M. Vajtersic, and P. Zinterhof, editors, *Proceedings of the International Workshop on Parallel Numerics (Parnum'97)*, pages 72–84, September 1997.

[16] J. HÄMMERLE AND A. UHL, Approaching real-time processing for fractal compression, In J. Biemond and E. J. Delp, editors, *Visual Communications and Image Processing '97*, volume 3024 of *SPIE Proceedings*, pages 514–525, San Jose, February 1997.

[17] J. HÄMMERLE AND A. UHL, Fractal image compression on multiprocessors and multicomputers, In R. Wyrzykowski, H. Piech, and J. Szopa, editors, *Proc. of the International Conference on Parallel Processing and Applied Mathematics (PPAM'97)*, pages 433–441, Zakopane, Poland, September 1997.

[18] J. HÄMMERLE AND A. UHL, Fractal compression of satellite images: Combining parallel processing and geometric searching, In E. H. D'Hollander, G. R. Joubert, F. J. Peters, U. Trottenberg, and R. Völpel, editors, *Parallel Computing: Fundamentals, Applications and New Directions*, number 12 in Advances in Parallel Computing, pages 121–128, North Holland, 1998.

[19] J. HÄMMERLE AND A. UHL, Classification based speed-up methods for fractal image compression on multicomputers, In P. Zinterhof, M. Vajtersic, and A. Uhl, editors, *Parallel Computation. Proceedings of ACPC'99*, volume 1557 of *Lecture Notes on Computer Science*, pages 276–285, Springer-Verlag, 1999.

[20] R. HAMZAOUI, Codebook clustering by self-organizing maps for fractal image compression, *Fractals*, 5 (Supplementary Issue):27–38, April 1997.

[21] Z. L. HE, M. L. LIOU, AND K. W. FU, VLSI architecture for real-time fractal video encoding, In *Proc. ISCAS'96 International Symposium on Circuits and Systems*, Atlanta, GA, USA, May 1996, IEEE.

[22] C. HUFNAGL, J. HÄMMERLE, A. POMMER, A. UHL, AND M. VAJTERSIC, Fractal image compression on massively parallel arrays, In *Proceedings of the International Picture Coding Symposium (PCS'97)*, volume 143 of *ITG-Fachberichte*, pages 77–80, VDE-Verlag, Berlin, Offenbach, September 1997.

[23] C. HUFNAGL AND A. UHL, Algorithms for fractal image compression on massively parallel SIMD arrays, *Real-time Imaging*, 5(6), 1999, to appear.

[24] D. J. JACKSON AND T. BLOM, A parallel fractal image compression algorithm for hypercube multiprocessors, In *Proceedings of the 27th Southeastern Symposium on Sytem Theory*, pages 274–278, March 1995.

[25] D. J. JACKSON AND W. MAHMOUD, Parallel pipelined fractal image compression using quadtree recomposition, *The Computer Journal*, 39(1):1–13, 1996.

[26] D. J. JACKSON AND G. S. TINNEY, Performance analysis of distributed implementations of a fractal image compression algorithm, *Concurrency: Practice and Experience*, 8(5):357–380, June 1996.

[27] A. E. JACQUIN, Fractal image coding: A review, *Proceedings of the IEEE*, 81(10):1451–1465, October 1993.

[28] C. K. LEE AND W. K. LEE, Fast fractal image block coding based on local variances, *IEEE Transactions on Image Processing*, 7(6):888–891, 1998.

[29] S. LEPSØY, *Attractor Image Compression - Fast Algorithms and Comparisons to Related Techniques*, PhD thesis, The Norwegian Institute of Technology, Trondheim, June 1993.

[30] H. LIN AND A. N. VENETSANOPOULOS, Parallel implementation of fractal image compression, In *Proceedings of Canadian Conference on Electrical and Computer Engineering*, pages 1042–1045, Montreal, September 1995.

[31] N. LU, EDITOR, *Fractal Imaging*, Academic Press, San Diego, CA, 1997.

[32] A. OSWALD AND J. BALL, A parallel quadtree approach to fractal image compression, In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'96)*, pages II/914–917, 1996.

[33] S. C. PEI, C. C. TSENG, AND C. Y. LIN, A parallel decoding algorithm for IFS codes without transient behavior, *IEEE Transactions on Image Processing*, 5(3):411–415, March 1996.

[34] A. POMMER, Fractal video compression on shared memory systems, In P. Zinterhof, M. Vajtersic, and A. Uhl, editors, *Parallel Computation. Proceedings of ACPC'99*, volume 1557 of *Lecture Notes on Computer Science*, pages 317–326, Springer-Verlag, February 1999.

[35] A. POMMER, C. HUFNAGL, AND A. UHL, Fractal motion compensation, In S. A. Rajala and M. Rabbani, editors, *Visual Communications and Image Processing '98*, volume 3309 of *SPIE Proceedings*, pages 1050–1061, The International Society of Optical Engineers, January 1998.

[36] J. PUATE AND F. JORDAN, Using fractal compression scheme to embed a digital signature into an image, In *Video Techniques and Software for Full-Service Networks*, volume 2915 of *SPIE Proceedings*, pages 108–118, Boston, MA, USA, November 1997.

[37] D. SAUPE, Accelerating fractal image compression by multi-dimensional nearest neighbor search, In J. A. Storer and M. A. Cohn, editors, *Proceedings Data Compression Conference (DCC'95)*, pages 222–231, IEEE Computer Society Press, March 1995.

[38] D. SAUPE AND R. HAMZAOUI, Complexity reduction methods for fractal image compression, In J. M. Blackledge, editor, *Proc. IMA Conf. on Image Processing; Mathematical Methods and Applications (1994)*, pages 211–229, Oxford University Press, September 1995.

[39] B. A. M. SCHOUTEN AND P. M. DE ZEEUW, Feature extraction using fractal codes, In *Proceedings of the Conference Fractals in Engineering*, Delft, June 1999.

[40] V. S. SUNDERAM, G. A. GEIST, J. DONGARRA, AND R. MANCHEK, The PVM concurrent computing system: evolution, experiences, and trends, *Parallel Computing*, 20:531–545, 1994.

[41] N. T. THAO, A hybrid fractal-DCT coding scheme for image compression, In *Proceedings of the IEEE International Conference on Image Processing (ICIP'96)*, volume I, pages 169–172, Lausanne, September 1996, IEEE Signal Processing Society.

[42] A. UHL AND J. HÄMMERLE, Fractal image compression on MIMD architectures I: Basic algorithms, *Parallel Algorithms and Applications*, 11(3–4):187–204, 1997.

[43] G. D. VECCHIA, R. DISTASI, M. NAPPI, AND M. PEPE, Fractal image compresson on a MIMD architecture, In H. Liddel, A. Colbrook, B. Hertzberger, and P. Sloot, editors, *High Performance Computing and Networking, Proceedings of HPCN Europe 1996*, volume 1067 of *Lecture Notes on Computer Science*, pages 961–963, Springer, 1996.

[44] M. XUE, T. HANSON, AND A. MERIGOT, A massively parallel implementation of fractal image compression, In A. Bovik, editor, *Proceedings of the first IEEE international conference on image processing*, pages II/640–II/644, IEEE Press, 1994.

[45] P. ZINTERHOF AND P. ZINTERHOF JUN., A parallel version of an algorithm for fractal image compression In *Workshop Paragraph 1994*, number 94-17 in RISC - Linz Report Series, 1994.

Received: May 15, 1999
Accepted in revised form: January 21, 2000

Contact address:
Jutta Hämmerle and Andreas Uhl
RIST++ & Department of Scientific Computing
University of Salzburg
A-5020 Salzburg
Austria
e-mail: `jhaemm@cosy.sbg.ac.at, uhl@cosy.sbg.ac.at`

JUTTA HÄMMERLE received the B.S. and M.S. (Computer Science) degrees from the University of Salzburg and is currently working for her own company "Horus Informationstechnologie". Her research interests include parallel processing and compression algorithms.

ANDREAS UHL received the B.S. and M.S. degrees (both in Mathematics) from the University of Salzburg and he completed his PhD on Applied Mathematics at the same University. He is currently Assistent Professor at the Department of Scientific Computing and at the Research Institute for Softwaretechnology. His research interests include wavelets, image processing (with emphasis on adaptive image compression), numbertheoretical methods in numerical analysis, and parallel processing.