The MGAP Family of Processor Arrays

Kevin Acken[1] Eric Gayles[1] Tom Kelliher[2] Robert M. Owens[1] Mary Jane Irwin[1]

[1]Architecture and VLSI CAD Group, Department of Computer Science and Engineering
The Pennsylvania State University, University Park, PA 16802

[2]Department of Mathematics and Computer Science
Westminster College, New Wilmington, PA 16172

Abstract

The Micro-Grain Array Processor (MGAP) is a family of massively parallel SIMD arrays of fine grain processing elements powerful enough to perform complex signal and image processing algorithms in real time. The MGAP was also designed to be compact enough to conveniently fit as an add-on board to a standard workstation at a fraction of the development cost of other comparable parallel machines. In this paper we update the status of the MGAP-2, which became operational in October 1996, and present a comparison of the MGAP-1 and the MGAP-2. We also give performance comparisons of the two designs through three popular image/video compression algorithms: the Discrete Cosine Transform, Motion Estimation, and Fractal Compression.

1. Introduction

Most image compression and image processing algorithms feature large degrees of data-parallelism that can lead to tremendous algorithmic optimizations. The problem is that conventional pipelined SISD computers can not fully take advantage of this data-parallelism due to their inherent sequential architecture. However, SIMD architectures, such as the Massively Parallel Processor [1] and the CLIP [3], are very well suited for data-parallel algorithms due to their ability to support large numbers of processors without the need for complex control logic. Many SIMD computers are also *fine-grain* (or *micro-grain*), meaning that the processors are relatively small. At most, the processors contain a small amount of local RAM, a few dedicated registers, and some minimal circuitry for bit-level arithmetic and logic. This leads to an important area tradeoff decision on how fine-grain to make the PEs, whether it is better to have more simple processors or fewer complex processors.

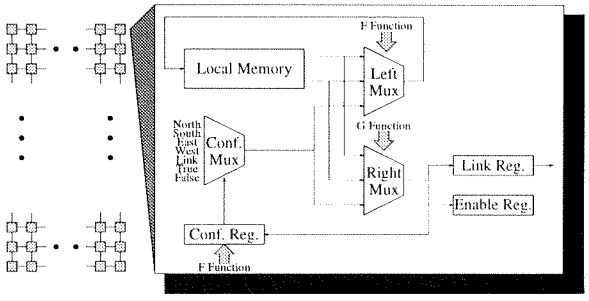

Figure 1. MGAP array structure and Processing Element (PE) structure.

The Micro-Grain Array Processor architecture, introduced in [5], is a massively parallel 2-D SIMD mesh of nearest neighbor connected fine-grain processing elements (PEs)(Figure 1). The key architectural advantage that the MGAP has over other SIMD computers is that each PE has connection autonomy. Connection autonomy allows each processor to independently select a direction in which to receive data. This ability allows the MGAP to group PEs into larger "virtual" processors called digit processors. Figure 2 shows four possible configurations for digit processors. Digit processors solves the area tradeoff issue because MGAP digit processors can be made larger for algorithms with higher point-computation needs, and can be made smaller for algorithms that require higher parallelism. Connection autonomy also provides added flexibility to algorithm design by allowing for arbitrary data flow and mixed precision arithmetic.

In this paper we will present a comparison of the MGAP family of processor arrays. Section 2 with give a detailed description of the MGAP architecture at both the chip and board levels, and also describe the MGAP programming environment. Section 3 will present the MGAP-II's design improvements over the MGAP-I. Section 4 will describe image compression applications that have been performed

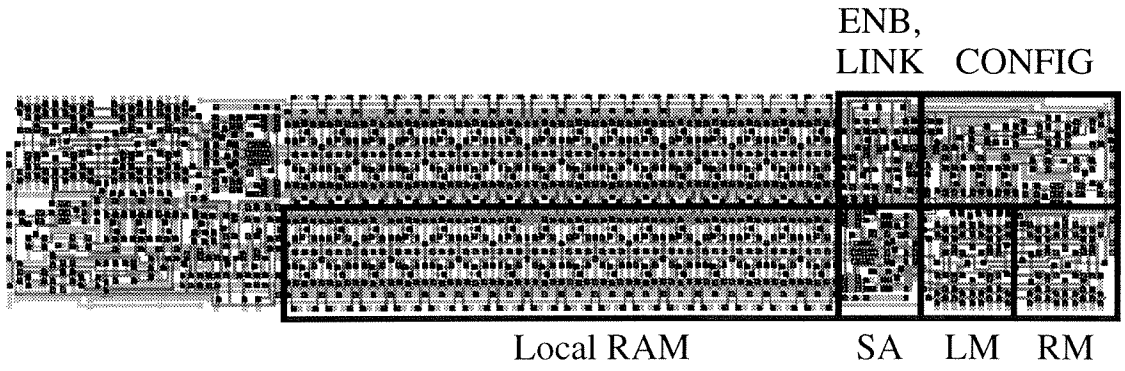

Figure 3. A magic file of a pair of processing elements (metal-2 removed for clarity). Each PE consists of a RAM, sense amp(SA), left function mux(LM), right function mux(RM), configuration mux and register(CONFIG), and enable and link registers.

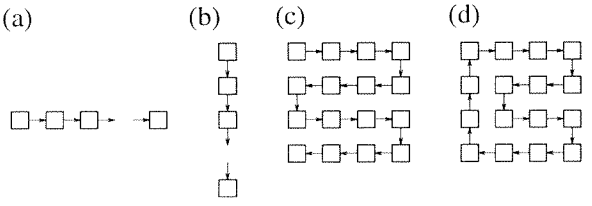

Figure 2. Various PE connections into digit processors: (a) horizontal; (b) vertical; (c) snake; (d) circular.

on the MGAP and section 5 will conclude this paper.

2. The MGAP Architecture

The three primary features of the MGAP architecture are (1) a PE design that allows for connection autonomy; (2) a low-cost board design allowing the MGAP to act as a coprocessor board; and (3) *C++, a high level programming language for the MGAP array. In this section, each of these features will be discussed in detail.

2.1. Processing Element Architecture

Figure 3 shows a block diagram and layout of a pair of PEs. Each PE forms a shape of an 'L' and is paired to the other along its memory cells. This pairing is used to optimize the area required for the global address lines. Each PE consist of three components: a local RAM, bit registers, and functional muxes. The RAM is a multi-ported memory unit that is capable of reading two bits and writing one bit each cycle. The read and write addresses are provided through global address lines, thus every enabled PE will read from/write to the same data addresses relative to its local memory. Each PE contains a configuration area (CONFIG) that has a 7-to-1 mux controlled by a three bit configuration register. The configuration mux selects one of seven bit inputs (north, south, east, or west neighbor's link, local link, or true or false) that will be sent to both the right and left functional muxes. Both the right function mux (RM) and the left function mux (LM) are 3-to-1 muxes that receive two bits from the local RAM and one bit from the configuration mux. These mux structures are each controlled by 8 global function lines (F and G function lines) that represent the truth table lines of a three input function, so any possible three input boolean function can be performed by selecting the appropriate global function lines. The output bit of the left function mux is returned to the RAM where it will be written. The output of the right function mux is sent to the link register. The link register represents the connection to a PEs nearest neighbors and so is fed into each neighbor's configuration mux. The last bit register is the Enable register (ENB), which determines if the PE is active or not. If the PE is disabled, writing to the RAM and to the configuration register are disabled.

There are three basic commands for controlling the PEs: *enable*, *control*, and *mgap*. The *enable* command is used to enable/disable individual processors based on values stored in each PE's local memory. During the enable command, a processor is enabled if its link register (the output of the right function mux) is one, and disabled if it is zero. The *control* command controls the values stored in the configuration registers. With this command, if the link register contains a zero, the three bit configuration register will be written with values stored on selected F function lines. The *mgap* command performs a full calculation within each enabled processor, with inputs determined by global address lines and the local configuration mux.

This design allows for connection autonomy through the selective use of the control and enable commands. More specifically, within each PE, a few memory bits will be allo-

cated to be *mark* bits. The mark bits determine the data flow direction for that PE. The mark bits need to be initialized by loading bit planes containing the initial PE configuration directions onto the array. Each PE is configured by selectively enabling based on the mark bits and using the control command to write the correct direction to the configuration register.

2.2. MGAP Board Structure

The design for the MGAP system is that of a coprocessor board connected to a workstation via a VME or similar bus. The workstation is assumed to be part of a network of such computers, allowing broad access to the array. On the board itself sits the array as well as PLDs and/or field programmable gate arrays which implement the control logic. These programmable devices link signals going across the bus with the array and an on-board memory used for local storage data storage. The board also contains dedicated program memory and a buffering mechanism for data moving between the array and the on-board memory. Finally, programmable ROMS are used to hold startup configuration programs. Figure 4 shows the board level designs for the MGAP family.

2.3. The MGAP Programming Environment

Programming SIMD machines can be a daunting task, often requiring the programmer to use assembly language calls to operate the machine efficiently. To solve this problem, a programming language, *C++, was created that combines the notion of digit processor shape (see figure 2) and MGAP specific library function with C++ to create a very good development platform for writing complex parallel applications. Figure 5 gives an example of a *C++ program that performs signed-digit array additions on the MGAP.

```
private : class word val[128][128];
public :
   class array operator+(class array a, b) {
      array z;
      int i, j;

      for( i = 0; i < 128; i++ )
         for( j = 0; j < 128; j++ )
            z.val[i][j] = a.val[i][j] + b.val[i][j];
      return( z );
   }
};

main(){
   class array A,B,C;

   C = A + B;
}
```

Figure 5. *C++ addition example.

3. Improvements in the MGAP Family

In this section we will first describe specific design decisions of the MGAP-1 architecture and the limitations that resulted from these decisions. We will then describe the design aspects of the MGAP-2 and show how these improved upon the limitations of the MGAP-1.

3.1. The MGAP-1 Architecture

The MGAP-1 was completed and fully operational in fall of 1993. Its primary purpose was to illustrate the micrograined approach to SIMD computing. It contained 16,384 processing elements housed across 32 chips. All of the array chips and the necessary control and I/O logic for the array were placed on a single 9U×400 mm board. This allowed for an extremely compact and cost effective parallel computing engine when compared with other image processing platforms. The board's operating frequency was 25 MHz. The chips were fabricated using 1.2 μm single-poly, double metal CMOS technology. Each PE contained 16 bits of local memory using an eight transistor memory cell capable of three simultaneous operations: two reads and one write. The entire system has a theoretical peak performance of 0.8 teraops. The host system was a SUN-4 workstation on the local network in Penn State's Computer Science and Engineering Department, allowing students and researchers easy access to the MGAP-1.

To simplify the construction processes, the processing array's control logic was completely programmable, providing a high degree of flexibility of which we have taken advantage. The control logic contained an instruction RAM, I/O memories leading to the processor array, and a separate scalar ALU unit. PLDs were extensively used as "glue" logic for instruction and address decoding. These PLDs inputs are rich enough for us to change the instruction set and even the instruction format a fair degree.

Several signal processing algorithms were successfully developed and implemented on the MGAP-1, including edge detection, discrete wavelet transform, dynamic space warping, and a cellular automaton model of lattice gases.

3.2. Limitations of the MGAP-1

The experience running signal and image processing primitives on the MGAP-1 has provided many valuable insights into the requirements of these target applications. It was realized that increasing each processor's local RAM

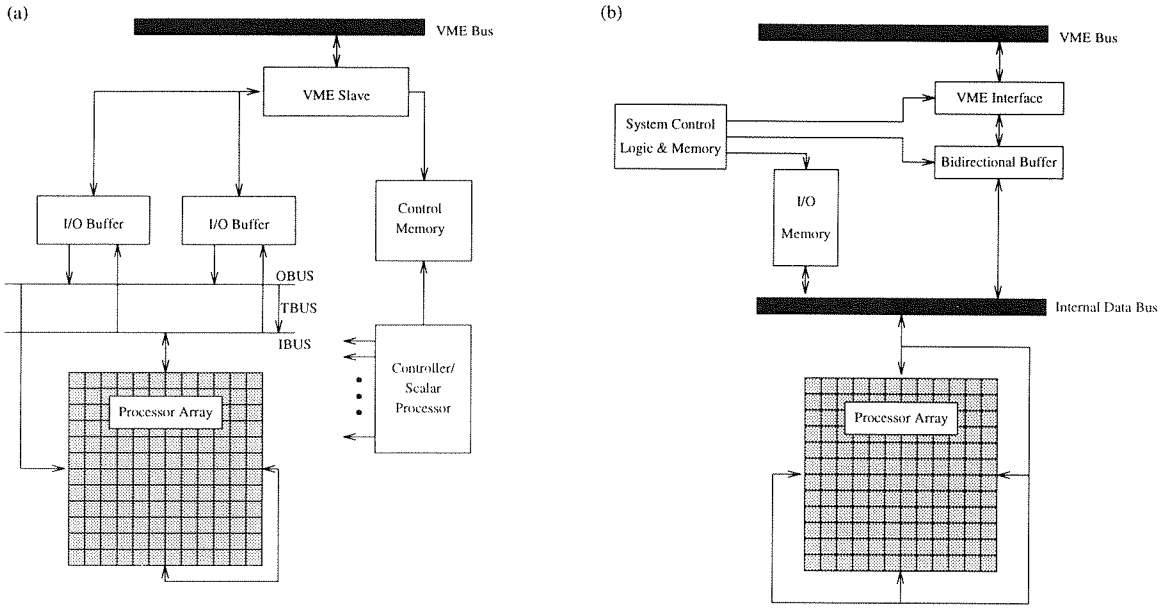

Figure 4. System architecture for (a) the MGAP-1 (b) the MGAP-2.

would significantly reduce the number of cycles required for moving data between the array and the external memories, and would reduce the need of reserving PEs for storage purposes only. As expected, many applications required the use of many more PEs that were available, thus forcing applications to work on only portions of the data and waste time swapping data into and out of the array. This was especially true for image processing applications where only small portions of images could be loaded onto the array. Another problem with the MGAP-1 design was that the extensive use of PLDs in the control logic resulted in control pipelines being of different lengths. This resulted in lost cycles due to bubbles in the pipeline at the start and end of longer computations. The control logic was also designed with the anticipation that most communication between the I/O memory and the processor array would involve simultaneous input and output for systolic algorithms. We also conjectured that a corner-turning/transpose movement would be important. These predictions led to a design with an extremely complex I/O topology. We found that the transpose movement was not needed for most applications, and that for several of our applications, especially those involving filtering operations, there was a requirement for constant I/O. These applications either do not use the I/O memories, instead buffering data through the FIFOs, or only briefly use the I/O memories, resulting in heavy I/O across the host system's VME bus. The MGAP-1 board used a simple slave interface which quickly became a bottleneck for many of these applications.

3.3. The MGAP-II Architecture

The MGAP-2 project started in the summer of 1994, and in October of 1996 it became fully operational. Though the MGAP-2 is architecturally similar to the MGAP-1, there have been several major enhancements to the MGAP-1 design to overcome some of its noted limitations. First, the chips were fabricated in a more aggressive 0.8μm single-poly, double-metal CMOS technology. This allowed us to triple the number of PEs to 49,152 while still maintaining a 32 chip array on a 9U$\times$400 mm board. The board speed was also doubled to 50 MHZ. To help alleviate the PE memory limitation, the local memory of each PE was also doubled to 32 bits. The PE's memory cells were reduced from eight to six transistor by separating the read and write operations within each cycle, which eliminating the ability of concurrent reads and writes, but allowing us to double the memory with only a minimal area penalty for each PE.

The control logic was also dramatically altered to enhance the design. Most of the decoding PLDs have be incorporated into two FPGAs, allowing for a much more flexible and programmable environment. This has also allowed the equalization of the control pipelines, thus both reducing lost pipeline cycles and making programming much easier. The MGAP-2 also transfers data between the host and itself more efficiently by utilizing a VME bus master requester with block transfer capability. This essentially gives the system DMA capability, greatly reducing the overhead associated with single word transfers. The I/O memory communication structure has also been significantly improved by using a much simpler bi-directional FIFO approach. The bus orga-

nization was also altered to better fit the needs demonstrated by the target applications, together with the operational characteristics of the processor array. The processor array must be capable of performing I/O to/from any of its four compass directions. There is no need for a having an I/O memory capable of simultaneous reads and writes, because the array doesn't perform input and output concurrently. The FIFO has been aided for providing buffering for filter-like applications.

4. Applications for the MGAP Family

The MGAP is a prime architecture for applications that are highly parallel and require fast data movement, such as in signal and image processing. The MGAP-2 contains three times more processors than the MGAP-1, and runs at twice the clock frequency, thus a six-fold time improvement can immediated be expected in the MGAP-2. But for many applications, a much better improvement is attained due to the fact that larger portions of the data can be worked on at a time, saving significant data swap time. In this section, we compare the performance of the MGAP-1 and MGAP-2 for three popular image/video compression algorithms: Discrete Cosine Transform, Motion Estimation, and Fractal Compression.

4.1. Discrete Cosine Transform

The 2-D Discrete Cosine Transform is a very popular image compression scheme used in both JPEG and MPEG compression standards. The small-n algorithms, with slight modification, can be used to compute the discrete Hartley transform (DHT), which in turn can be used to compute the 1-D and 2-D DCT [2]. The following equations show this process of calculating the 1-D and 2-D DCT:

$$DCT_{1D}(x) = ASC'TPx$$

$$DCT_{2D}(X) = (ASC'TP(ASC'TPX)^T)^T$$

where x is a sequence of length N, P is a permutation matrix, T and S contain elements from -1,0,1, C' is diagonal, and A embodies the relationship between the DHT and DCT. This algorithm was mapped to the MGAP architecture in [6] and figure 6 summarizes the eight computation stages required for each DCT block along with the data flow within the MGAP subarray. Table 1 shows the timing results for image sizes 256x256 and 512x512 using both 8x8 and 16x16 sized block. The results show that the MGAP-2 is 9 times faster for the 8x8 block size and the MGAP-2 is 31 times faster for the 16x16 block size. The results also shows that the MGAP-2 can compute the DCT of reasonably sized images in real-time.

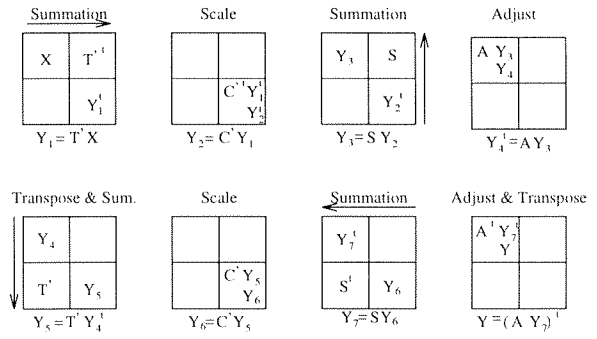

Figure 6. 2-D DCT data flow on the MGAP.

Table 1. 2-D DCT on MGAP-1 and MGAP-2.

	DCT size	256x256 image	512x512 image
MGAP-1	8x8	34.55 msec	138.19 msec
	16x16	202.22 msec	808.88 msec
MGAP-2	8x8	3.52 msec	14.08 msec
	16x16	6.50 msec	26.0 msec
MGAP-2 Speedup	8x8	9.81	9.81
	16x16	31.1	31.1

4.2. Motion Estimation

Motion estimate is a very powerful video compression technique used in many compression schemes include the MPEG standard. Motion estimate is based on Block Matching Algorithms (BMA), such as full-search BMA, where a template pixel block from the current video frame is compared to candidate pixel blocks within some predefined distance d of the template block in the previous video frame. The idea is to determine pixel block motion by finding a candidate block that best matches the template block. The goodness of the match is determined by a distance metric. The most commonly used distance metric is the Mean Absolute Difference (MAD), given in the following equation, where pixel blocks x and y of size nxn are compared.

$$MAD(x,y) = \sum_1^n \sum_1^n |x(i,j) - y(i,j)|$$

The MAD is often used because of its relative accuracy and inexpensive calculation. This algorithm was mapped to the MGAP architecture in [7] using both a broadcasting and a non-broadcasting algorithm. In the broadcasting approach, the PE's contain the search area and the template block is broadcast to all PEs within the search area. The distance between the template block and the search area blocks are calculated in parallel. In the non-broadcast scheme, the search area and the template blocks are stored in separate bitplanes on the array. The minimum distance is systolically calculated by shifting the template plane over the search area plane. Table 2 gives the relative results for a block matching algorithm for the MPEG SIF video format (352x240 pixels)

with the template block size of 16x16. There is little time gain in the broadcasting method because broadcasting reduces the data-parallelism and therefore inefficiently uses the array.

Table 2. Motion Estimation (16x16 template)

	d	broadcasting	non-broadcasting
MGAP-1	16	117.2 msec	374.7
	32	325.4 msec	740.52
MGAP-2	16	44.4 msec	22.72
	32	65.21 msec	99.6
MGAP-2	16	2.63	16.5
Speedup	32	5.0	7.43

4.3. Fractal Image Compression

Fractal image compression is not as popular as DCT or motion estimation due to its expensive compression algorithm, but recent algorithmic improvements along with advancing hardware are making fractal compression a viable compression technique [4]. Fractal image compression uses redundancy of scale and the fractal behavior of most images to create highly compressed images with very good fidelity. Fractal compression also has a very high degree of data-parallelism, which will map very well to the MGAP architecture.

The basic fractal compression algorithm consists of first dividing the image into non-overlapping blocks, called *Range* blocks, and also dividing the image into possibly overlapping blocks called *Domain* blocks. The *Domain* blocks need to be larger in area than the *Range* blocks, and the *Domain* blocks will be contracted down through pixel averaging to the size of the *Range* blocks. For each *Range* block, a "good" *Domain* match must be found. This is accomplished by compare each *Range* with a subset of the *Domains*, using a distant metric such as Mean Square Error (MSE), and allowing for pixel parameters such as a brightness scale factor and contrast offset factor. The following equations calculate the MSE between a *Domain* block d, and a *Range* block r, each containing n pixels. s is the brightness scale factor, o is the contrast offset factor, and MSE is the distance between the blocks.

$$s = \frac{\left[n \sum d_i r_i - \sum d_i \sum r_i\right]}{\left[n \sum d_i^2 - \left(\sum d_i\right)^2\right]}$$

$$o = \frac{\left[\sum d_i - s \sum r_i\right]}{n}$$

$$MSE = \sum (s\, d_i + o - r_i)^2$$

In the algorithm we mapped to the MGAP, the *Range* blocks are of a fixed 4x4 pixel size, and *Domain* blocks are 8x8 pixels overlapped with a step size of two. Table 3 show the timing results. The MGAP-2 shows a large improvement over the MGAP-1 due to the excessive swapping of data required by the MGAP-1.

Table 3. Fractal Compression

	256x256	512x512
MGAP-1	37.12 sec	593.92 sec
MGAP-2	0.29 sec	4.64 sec
Speedup	128	128

5. Conclusion

In this paper, we have presented an updated status of the MGAP-2. We compared the designs of the MGAP-1 and the MGAP-2 with emphasis on the design improvements in the MGAP-2 that helped overcome the limitations of the MGAP-1. We have also shown how our experience designing applications on the MGAP-1 motivated the architectural changes realized in the MGAP-2. Finally we presented comparison of image compression algorithms between the MGAP-1 and MGAP-2 and showed that the reduction in data swapping produced large gains in performance.

References

[1] Kenneth E. Batcher. Design of a Massively Parallel Processor. *IEEE Transactions on Computers*, pp.836-840, September 1980.

[2] C. Chakrabarti and J. JaJa. Systolic Architectures for the Computation of the Discrete Hartly and Discrete Cosine Transforms Based on Prime Factor Decomposition. *IEEE Transactions on Computers*, pp.1359-1368, November 1990.

[3] M.J.B. Duff. CLIP 4: A Large Scale Integrated Circuit Array Parallel Processor. *IEEE International Joint Conference on Pattern Recognition*, pp.728-733, 1976.

[4] Y. Fisher. *Fractal Image Compression.* Springer-Verlag, New York, 1995.

[5] M.J. Irwin and R.M. Owens. A Micro-grained VLSI Signal Processor. *1992 Proceedings of ICASSP*, pp.641-644, March 1992.

[6] H.N. Kim, M. Borah, R.M. Owens, and M.J. Irwin. 2-D Discrete Cosine Transforms on a Fine Grain Array processor. *1994 Proceedings of IEEE Workshop on VLSI Signal Processing*, pp.356-367, October 1994

[7] H.N. Kim, M.J. Irwin, and R.M. Owens. Motion Estimation Algorithms on Fine Grain Array Processors. *1995 Proceedings of ASAP*, August 1995.