Improving processing time of large images by instruction level parallelism

Flávio L. C. Pádua[1], Guilherme A. S. Pereira[1], José P. de Queiroz Neto[1,2], Mário F. M. Campos[1], Antônio O. Fernades[1]

[1]DCC/UFMG – Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Av. Antonio Carlos 6627, Belo Horizonte, MG, Brasil

[2]CEULM – Centro Universitário Luterano de Manaus, Av. Solimões 02, Manaus, AM, Brasil

[2]CEFET/AM – Centro Federal de Educao Tecnológica do Amazonas, Av. Danilo Areosa, s/n, Distrito Industrial, Manaus - AM

{cardeal, gpereira, jpq, mario, otavio}@dcc.ufmg.br

Abstract. This paper presents experiments which show how Multimedia Extensions found in the modern computer architectures can be used to speedup common image processing algorithms. This approach can be very useful in the development of efficient algorithms for processing of large images like the ones acquired with satellites. The experiments were performed with satellite images from the Amazonas Forest in Brazil.

Keywords: MMX Technology, Instruction Level Parallelism, Satellite Images, Speedup

1 Introduction

Remote sensing involves the use of instruments or sensors to acquire the spectral and spatial relations of objects and materials observable at a given distance [15]. Multitemporal images collected by digital multispectral imaging systems have been available for more than 25 years and have been used for various Earth-science applications [12] like detection of land cover changes driven by seasonal and interannual climate variations, long-term climate shifts, vegetation succession, and human or natural disturbances [14]. The Landsat series of satellites provides the longest running continuous data set of high spatial-resolution imagery dating back to the launch of Landsat 1 in 1972 , using the TM (Thematic Mapper) Multispectral Scanner [19]. The TM has 8-bit radiometric resolution and seven bands in a 185Km^2 area, with 30m spatial resolution, except band 6 (120m). Each band has an specific spectral range, appropriate to applications, i.e., band 4 (near infrared) to biomass surveys, water-body delineation, and band 5 (shortwave infrared) mainly to vegetation moisture and snow-cloud differentiation [17].

This work presents the use of instruction level parallelism based on the MMX instruction set, available in several modern processors, to speedup the processing of images, in particular the processing of large images like the ones acquired with satellites. An important advantage is that it is not necessary to transmit data among different machines in a computer network. The communication among processors is the greatest bottleneck in performance degradation of parallel programs [18]. In a previous work [13] it is shown that it is possible to implement functions in MMX Assembly and use them inline with C code to improve digital image processing in a single machine. Another approach made by a major microprocessor manufacturer is a free library for image processing using its own technology [10]. In a workshop organized by IEEE in 1999 [9], many authors acknowledged the use of this technology as being very useful in real time image processing.

As the processing of satellite images can be very time consuming, this situation has motivated the introduction of a level of parallelism to improve the processing. This paper presents results of several experiments using satellite images that show how the use of MMX instructions can speed up the algorithms for image processing.

The remaining of this text is organized as follows: Section 2 makes a brief introduction to the satellite image processing; Section 3 focuses on the parallel processing in a single processor, describing the MMX features; Section 4 shows the results of several experiments indicating that the instruction level parallelism can be a good alternative to be used for processing a large amount of data. Finally, the main points of the work are summarized in Section 5.

2 Satellite Image Processing

Prior to data analysis, a pre-processing of the satellite raw data is usually carried out in order to correct any distortion due to the characteristics of the imaging system and imaging conditions. Depending on the user's requirement, some standard correction procedures may be carried out by the ground station operators before the data is delivered to the end-user. Those procedures include radiometric correction for uneven sensor response over the whole image and geometric correction to minimize the geometric distortion due to Earth's rotation and other imaging conditions (such as oblique viewing) [2, 6, 4, 8]. Different landcover types in an image can be discriminated using image classification algorithms using spectral features, i.e. the brightness and "color" information contained in each pixel. The classification procedures needs mathematical manipulations that has influence in the efficiency of the algorithm, due to the large number of pixels in a full scene image (185Km^2) with more than 64 million pixels, depending on the spatial resolution. Figure 1 is an example of one of the satellite images used in this work.

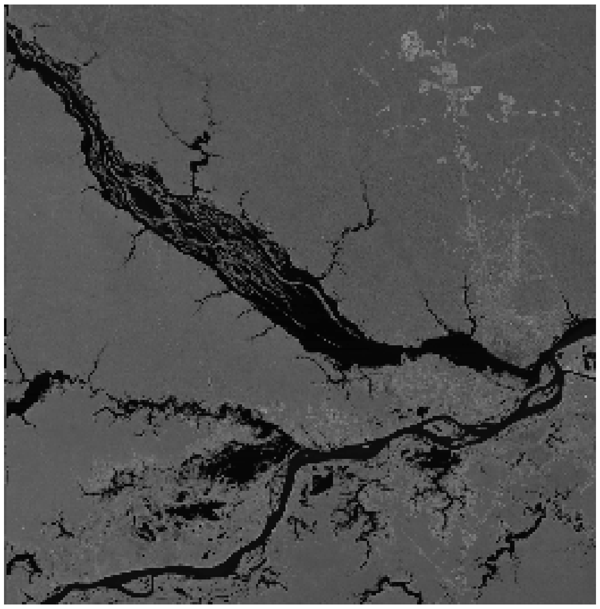


Figure 1: Images of the surroundings of Manaus city in the Amazon region taken by the satellite Landsat 5 in August, 1995.

3 Parallel Programming in a single processor

Parallel processing is a wide concept that can refer to the improvement of the processing time of a program by subdividing it in multiple fragments which can be processed concurrently, each one in its own processor. Under another point of view, parallel processing can be seen as sharing a large amount of data among several processors that run the same program in different parts of the data [18]. This second kind of parallelism, know as SIMD (Single Instruction Multiple Data) [5] consists the base of modern processors to improve processing time of programs that use a large amount of data such as multimedia applications. The basic idea is to use SIMD type of parallelism in a single processor to allow simultaneous processing of single instruction on several small registers, which actually are part of a single large register. This concept has been called SWAR (SIMD Within A Register) [3] and it means that a single instruction of a machine with registers, buses and function units of *k bits* can be used to process *n* parallel operations SIMD on the *n* slots of *k/n bits* that subdivide the registers.

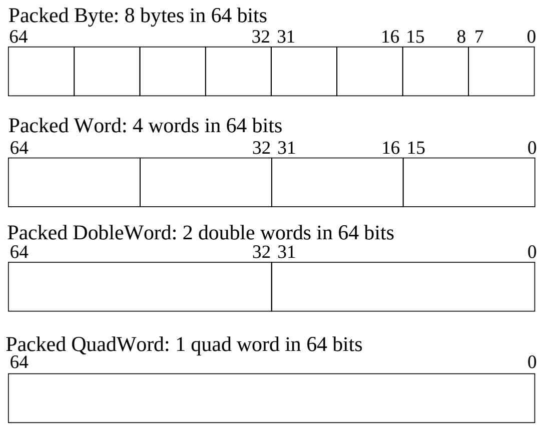
Although SWAR parallelism can be implemented in all machines using integer registers and operations [3], most of modern processors have been designed with special instructions to improve this technique in multimedia tasks. Several technologies have been created for MMX (MultiMedia eXtensions) [11, 1]. A complete list of the processors that have extensions for the SWAR parallelism can be found in [3]. In spite of the fact that this technology is present in commercial processors since 1997 with the introduction of the first Pentium MMX, today it is still difficult to find compilers that automatically use this technology. The small number of commercial compilers that generate MMX code are in most cases expensive and the results are unsatisfactory. On the other hand, some new compilers like the Microsoft Visual C++, Powersoft Watcon and the GNU GCC, that is open-source, provide inline assembly support for MMX instructions which allow programmers to develop applications using this technology. Such technique that consists in using a high level language mixed with MMX assembly code is discussed in the next section.

3.1 SWAR Programming with MMX

MMX technology was developed by the Intel Corp. in 1996 to improve multimedia and telecommunication applications. This technology was implemented in a architecture that includes instructions and data types that allow a high level of performance. It explores the inherent parallelism of multimedia algorithms, including image processing, where processing happens on a single pixel or region independent of its neighborhood.

As mentioned before, the key to MMX technology is the hardware implementation of SWAR. To do this, four new data types were created. These data types that are called packed fixed-point and are implemented in eight 64-bit registers are: packed bytes (8 bits), packed words (16 bits), packed doublewords (32 bits) and packed quadwords (64 bits). Figure 2 shows how these data types are implemented in a 64-bit register.

These types allow to understand how powerful is the SWAR implementation. As an example, consider that a pixel in a gray level image is represented by 8 bits. With the MMX operations it is possible to encapsulate in a single vector 8 pixels



Packed Byte: 8 bytes in 64 bits
64
32 31
16 15
8 7
0
Packed Word: 4 words in 64 bits
64
32 31
16 15
0
Packed DobleWord: 2 double words in 64 bits
64
32 31
0
Packed QuadWord: 1 quad word in 64 bits
64
0

Figure 2: Data types in a single register of MMX architecture.

of the image and then load this vector in a 64-bit MMX register. Afterwards any of the 57 MMX arithmetic or logic operations can be executed on all of the 8 elements concurrently. This operation can also be carried out among two or more registers each one with eight pixels.

Another important characteristic of the MMX instruction set is that some of its instructions are able to deal with saturation, that is, if the result of an operation is a number greater (smaller) than the maximum (minimum) supported by the type, this number will be saturated in this maximum (minimum). This feature is very useful in image processing since in many applications an extra instruction is necessary to provide this saturation level and to avoid data overflow or underflow.

To explain the previous characteristics, a binarization algorithm will be presented. In C language, once the image is in the main memory in a region pointed by the variable `data`, the binarization with a threshold of 127 is accomplished by:

```
for (i=0;i<size;i++)
 data[i] = (data[i] >= 127) ? 255 : 0;
```

Each pixel of the image is loaded from memory to a processor register and then compared to the threshold. This operation is repeated `size` times where `size` is the number of image lines multiplied by the number of image columns. With SWAR programing, the key idea is to reduce the number of load and compare operations by loading and comparing at the same time and with a single instruction or group of instructions 8 pixels of the image. Consequently, the commands in the loop are repeated only `size/8` times. For the binarization, a solution is shown graphically in Figure 3 where the binarization is made only with 4 basic MMX instructions.

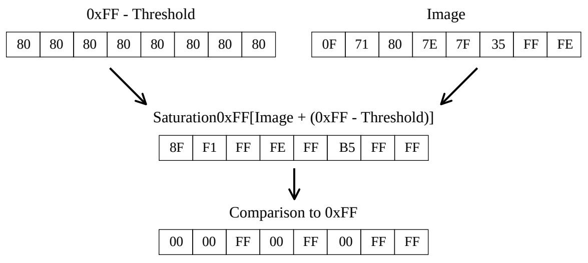


Figure 3: Binarization of 8 pixels using SWAR. The threshold used in this example is 127 or `0x7F`.

The first instruction loads 8 pixels from memory to a single register. The second adds with saturation these 8 pixels to a previous filled register that contains 255 (0×FF) minus the threshold. After this, the resultant register is compared with 255 an finally the binarized vector of 8 pixels is stored in memory. A complete MMX code to make a image binarization can be seen in Figure 4.

Due to the lack of compilers that are able to generate a complete MMX Assembly from a C program, the most efficient way to use the MMX features is writing functions directly in the Assembly language, like the one shown in Figure 4, and include then in-line in a main program written in C. Furthermore, the programmer must follow some “rules” that consider the processor architecture, its pipeline and the dynamic characteristic of program execution. Among these rules, one of the most important is related to the alignment between the data and stack. When the data access produce cache miss, an entire line of the cache is fetched from main memory. In the Pentium architecture, data arrives in 4 or 8-byte blocks to fill the 32 bytes of the cache line. Thus, aligning the data in blocks of 32 bytes at main memory, it is possible to explore the line size to avoid multiple memory transfers. The delay of a cache miss in the Pentium is 8 clock cycles and for some P6 processors can be as large as 10 cycles. If in a MMX routine uses an argument two or more times, it is better that this argument is aligned to avoid cache misses. A simple C code for stack alignment shown in [13] is:

```
void Binarize(unsigned char *Img, int Size,
                               unsigned char Threshold){
asm volatile(

 "pcmpeqb   %%mm1, %%mm1 \n\t" // generate 0xFF in mm1

 // Put Threshold in 8 bytes of MM3
 "pcmpeqb   %%mm2, %%mm2 \n\t" // generate 0xFF in mm2
 "mov           %2, %%al \n\t" // load Threshold into al
 "mov         %%al, %%ah \n\t" // copy al into ah
 "mov         %%ax, %%bx \n\t" // copy ax into bx
 "shl         $16, %%eax \n\t" // shift 2 bytes of eax
 "mov         %%bx, %%ax \n\t" // copy bx into ax
 "movd      %%eax, %%mm3 \n\t" // copy eax into mm3
 "movd      %%eax, %%mm4 \n\t" // copy eax into mm4
 "punpckldq %%mm4, %%mm3 \n\t" // fill higher bytes of
                               // mm3 with Threshold

 "psubusb   %%mm3, %%mm2 \n\t" // mm2=0xFF-Threshold

 "mov          %0, %%edi \n\t" // edi = Img address
 "mov          %1, %%ecx \n\t" // ecx = Size
 "shr          $3, %%ecx \n\t" // Size/8 (MMX loads
                               // 8 bytes  at a time)

 //  Main Loop
 ".Loop:                 \n\t"
 "movq    (%%edi), %%mm0 \n\t" // mm0=8 bytes of Img
 "paddusb   %%mm2, %%mm0 \n\t" // mm0=Img+0xFF-Threshold
 "pcmpeqb   %%mm1, %%mm0 \n\t" // binarize, comp. to 255
 "movq    %%mm0, (%%edi) \n\t" // store result in Img
 "add          $8, %%edi \n\t" // Img pointer += 8
 "dec              %%ecx \n\t" // decrease loop counter
 "jnz             .Loop  \n\t" // check loop termination

 "emms                   \n\t" // exit MMX state

 : "=m"  (Img)                 // %0
 :  "m"  (Size),               // %1
    "m"  (Threshold)           // %2
 );
}
```

Figure 4: A binarization function written in MMX assembly.

```
im=(unsigned char *)malloc(size*sizeof(unsigned char)+31);
data=(unsigned char *)((((unsigned int) im1)+31) & (~31));
```

where `data` is the position used to load the image. This code is very important when comparing the performance of different codes because it put the memory hierarchy in the same initial condition. It is used in the experiments of next section when the execution time of a MMX code is compared with both the code generated by a C compiler with and without optimization.

4 Experiments

Seven operations commonly present in many digital image processing algorithms were implemented in both MMX Assembly and C. Most of the MMX implemented operations were based on the algorithms proposed in [13] to work with small images. Those operations were chosen due to their frequent usage in satellite image processing programs. The seven operations are:

1. **Sub**: performs the subtraction of two images. It uses instructions that are able to deal with saturation. Normally, a subtraction operation among a reference and test multitemporal satellite images is used to find changes in a coarse analysis.
   $I3 = saturation255(I1 - I2)$;

2. **Bin1**: performs binarization on an image when a threshold is supplied.
   $I = I > T?255 : 0$;

3. **And**: performs the logical AND operation between two images. $I3 = I1\&I2$;

4. **Neg**: performs the complement of the original image. $I = !I$;

5. **Bin2**: performs image binarization when two different thresholds are supplied. Using one or two thresholds, the binarization operation can be used as a basic operation to pattern recognition, or to find pseudoinvariants features in order to select control points used in a geometric correction of satellite images.
   $I = (I > T_{min})\&(I < T_{max})?255 : 0$;

6. **Sob**: implements the Sobel's edge detection method. In satellite images this method is an essential algorithm to find specific features like rivers and roads. This routine calculates only the *x* component of the gradient for each pixel in the image. It uses instructions that are able to deal with saturation;

7. **Lin**: performs the function: $I2 = a \cdot I1 + b$ that is used in some algorithms to make relative radiometric normalization using linear regression [4].

The conditions in which the tests were performed are presented below:

- **Hardware**: two machines were used, namely a Pentium II 233 MHz and a AMD K6 233 MHz, both with a main memory of 96MB. The AMD K6 memory is around 40% faster than the one of the Pentium machine;

- **Software**: Linux operating system [16] - version 2.2.14-14cl, GCC compiler [7] - version egcs-2.91.6619990314/Linux (egcs-1.1.2 release);

- **Images**: gray level satellite images with resolution of 6120x6176 pixels;

- **Measurements**: the running times are average values given in seconds. They were measured using the function clock(). This function returns the time a process uses the CPU. Each routine was executed five times and the average value is presented.;

The measured running times are presented in Tables 1 and 2. In these Tables, the numbers in the first column correspond to operations described before. C_{opt} corresponds to the execution time of the C program which had its object code optimized by the compiler using the `-O2` GCC option. $Sp_{\frac{C}{MMX}}$ refers to the speedup of the MMX over C codes and $Sp_{\frac{C_{opt}}{MMX}}$ refers to the speedup of the MMX over the optimized C codes.

In order to better visualize the results of the experiments, two graphics were obtained for the data in the Tables 1 and 2. Figure 5 corresponds to the

Table 1 - Pentium II 233 MHz Processor

	Op.	MMX (s)	C (s)	C_{opt} (s)	$Sp_{\frac{C}{MMX}}$	$Sp_{\frac{C_{opt}}{MMX}}$
1	**Sub**	10.42	17.87	13.38	1.72	1.28
2	**Bin1**	0.54	2.50	0.94	4.63	1.74
3	**And**	11.60	16.97	13.60	1.46	1.17
4	**Neg**	0.53	2.29	1.08	4.32	2.04
5	**Bin2**	0.64	2.48	0.94	3.88	1.47
6	**Sob**	0.71	4.26	1.25	6.00	1.76
7	**Lin**	1.32	10.80	5.01	8.18	3.80

Table 2 - AMD K6 233 MHz Processor						
	Op.	MMX (s)	C (s)	C_{opt} (s)	$Sp_{\frac{C}{MMX}}$	$Sp_{\frac{C_{opt}}{MMX}}$
1	**Sub**	1.10	4.90	2.98	4.45	2.71
2	**Bin1**	0.65	2.48	1.19	3.80	1.82
3	**And**	1.09	3.76	1.92	3.44	1.76
4	**Neg**	0.65	2.34	1.27	3.57	1.94
5	**Bin2**	0.66	2.48	1.34	3.73	2.02
6	**Sob**	1.83	16.03	6.35	8.74	3.46
7	**Lin**	0.67	4.07	1.98	6.11	2.97

speedup data of Table 1 and Figure 6 to the data in Table 2.

The analysis of Figures 5 and 6 and their respective Tables shows that routines in Assembly MMX can bring substantial improvements in the performance of the algorithms for processing of large images. The running times must be analyzed carefully since their values are intimately related to issues like memory hierarchies and clock frequency of the machine. Despite the two processors have the same clock frequency in most of experiments the AMD K6 was faster than the Pentium II. The biggest difference occurs when two images are used like **Sub** and **And**. A reason for this fact is related to the difference between the memory technologies of both machines. As the use of two images causes frequent cache misses and the AMD K6 machine has a newer and faster main memory it have advantage during the transfer from memory to the cache.

The memory access time is also responsible for the variation in the speedup among the operations. The best results in both machines were obtained for the operations 6 (Sobel) and 7 (Linear correction) that is the most complex executed. It occurs because in both operations more instructions are executed with a single vector of data previously load. It turns lower the relation between the number of loads and stores (instructions that effectively causes cache misses) and the number of the other instruction. This lower relation favor the power of the MMX instructions since the number of cache misses are supposed to be the same for the C and MMX codes. To show that this observation is correct, the same operations applied in satellites images were executed for other image sizes. Figure 7 shows the results of the speedup between a C program (without the `-O2` option) and the MMX code as a function of the image size. Those results were obtained in the AMD K6 processor. Notice that the speedups are higher for the small images than for the large ones. It also can be noticed that the speedup variation is higher when the amount of data is about the same magnitude of the cache size, due to the lower number of cache misses.

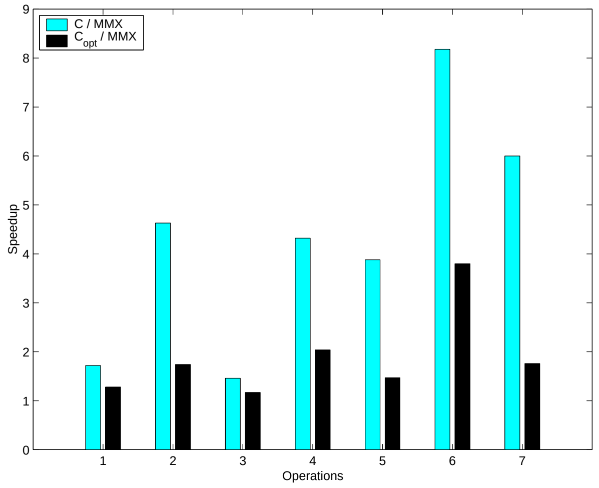


Figure 5: Speedup in the Pentium II 233 MHz processor.

5 Conclusions and Future Work

This paper showed how the multimedia extensions, available in most of the modern computers, can be used to improve the time processing of large satellite images. Despite the large number of cache misses caused by this kind of image the MMX technology, a hardware implementation of the SWAR parallelism, proved to be useful for it time processing. The results suggest that the routines in MMX as being a good alternative in the construction of image processing systems. In spite of the apparently difficulty in construct this routines, once they are written as a library their use can bring a great improvement.

Next steps for this research will be first to use

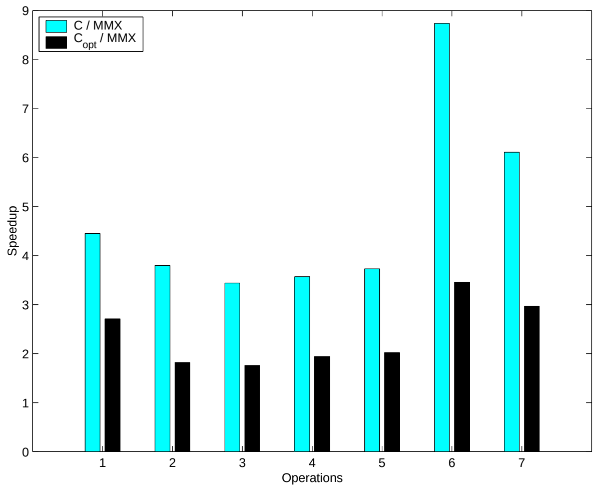


Figure 6: Speedup in the AMD K6 233 MHz processor.

the MMX technology in order to improve the algorithms used to radiometric and geometrical correction of satellite images. The final goal will be to apply the technique to construct a library and an application program to manipulate this kind of image.

Acknowledgements

This work would not have been possible without CNPq financial support. The authors would also like thanks to Dr. Bruce W. Nelson from the National Institute of Amazonia Researches (INPA) for providing the information about the use of satellite images of the Amazon Forest. The TM-Landsat 5 image used in this paper was kindly provided by the 4th Measures Maps Division (4^a DL) of the Brazilian Army.

References

[1] AMD. *AMD Extensions to the 3DNow and MMX Instruction Sets Manual*, 2000.

[2] L. Gottesfeld Brown. A survey of images registration techniques. *Computing Surveys*, 29:325–376, 1992.

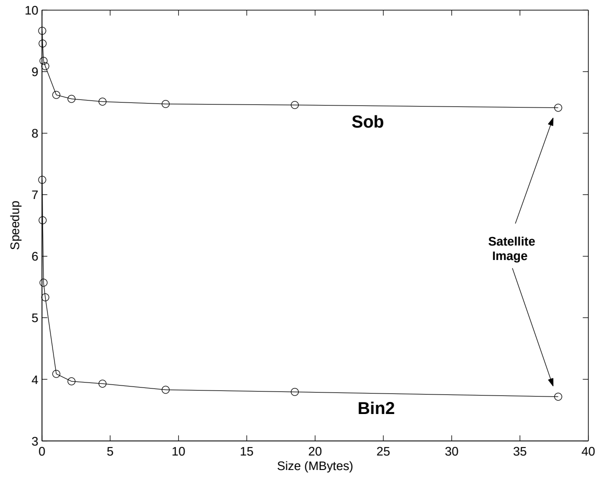


Figure 7: Speedup in the AMD K6 233 MHz processor using images with different sizes.

[3] H. G. Dietz and R. J. Fisher. Compiling for SIMD within a register. In *Proceedings of the Workshop on Languages and Compilers for Parallel Computing*, University of North Carolina at Chapel Hill, North Carolina, EUA, August 1998.

[4] C. D. Elvidge, D. Yuan, R. D. Weerackoon, and R. S. Lunetta. Relative radiometric normalization of landsat multispectral scanner (MSS) data using an automatic scattergram-controlled regression. *Photogrametric Engineering & Remote Sensing*, 61(10):1255–1260, October 1995.

[5] M. J. Flyn. Some computer organisations and their effectiveness. In *in Proc. of IEEE Transactions on Computers*, September 1972.

[6] Leila M.G. Fonseca and B.S. Manjunath. Registration techniques for multisensor remotely sensed images. *Photogrammetric Engineering & Remote Sensing*, 62(9), September 1996.

[7] GCC home page. http://gcc.gnu.org/.

[8] F. G. Hall, E. D. Strebel, J. E. Nickeson, and S. J. Goetz. Radiometric rectification: Toward a common radiometric response among multidate, multisensor images. *Remote Sensing of Enviroment*, 35:11–27, 1991.

[9] IEEE ICCV'99 frame-rate workshop. http://www.eecs.lehigh.edu/FRAME/, 1999.

[10] Intel image processing library. http://developer.intel.com/software/products/.

[11] Intel. *Intel Architecture MMX Technology Programmer's Reference Manual*, 1997.

[12] Pat S. Chavez Jr. Image-based atmospheric corrections-revisited and improved. *Photogrammetric Engineering & Remote Sensing*, 62(9), September 1996.

[13] Vladimir Kravtchenko. Using MMX technology in digital image processing. Technical Report TR-98-13, Departament of Computer Science – University of British Columbia, 1998.

[14] Eric F. Lambin. Change detection at multiple temporal scales: Seasonal and annual variations in landscape variables. *Photogrammetric Engineering & Remote Sensing*, 62(8), August 1996.

[15] T. M. Lillesand and R. W. Kiefer. *Remote Sensing and Image Interpretation*. John Wiley & Sons, Inc., 3rd edition, 1994.

[16] Linux home page. http://www.linux.org/.

[17] Aram M. Mika. Three decades of landsat instruments. *Photogrammetric Engineering & Remote Sensing*, 63(7), July 1997.

[18] Michael J. Quinn. *Parallel computing : theory and practice*. Mcgraw-Hill, 1994.

[19] K. Thome, B. Markham, J. Barker, P. Slater, and S. Bigger. Radiometric calibration of landsat. *Photogrammetric Engineering & Remote Sensing*, 63(7), July 1997.