What's New in Computers

MMX Technology for Multimedia PCs

S Balakrishnan

S Balakrishnan is a research scholar at the Supercomputer Education and Research Centre, Indian Institute of Science Bangalore. His interests include the design of special purpose computer architectures for digital signal processing and operating systems.

In this article we discuss Intel's MMX technology and its integration as part of multimedia PCs.

In late 1996 Intel announced an enhancement of the Pentium Processor architecture and christened it *MMX technology* or *multimedia extensions.* The Intel MMX technology is purportedly the most significant enhancement to the Intel architecture since the extension of the x86 architecture to 32 bits in 1985 when Intel first introduced the 80386 processor. The primary motivation behind this extension is the realization that a large number of computer intensive applications such as 3D graphics, interactive video games, speech processing, animations and virtual reality demands much more of the current day processors than what could be obtained by increased clock speed and complexity of processors. Hence, a new and simple approach incorporating some of the oft-used techniques in digital signal processing chips had to be adopted. A constraint faced by the designers of MMX technology was that any processor incorporating this technology had to be fully compatible with existing Intel architecture PC designs and all x86 based operating systems and application programs. This meant that an MMX technology extension could not create new modes or states.

Intel's microprocessor architects and software developers ran a wide range of applications that included video and audio processing and reported a performance gain of about 50%-100% over the same applications run on the same processor without using MMX technology. The new technology has been crafted to ensure that increased clock speed will give proportionate improvement in speed of

processing. All future generation Intel processors (P6 and it's successors) will have MMX technology integrated in their design.

Characteristics of Multimedia Applications

Multimedia[1] applications place a high demand on the computing resources of a media processing computer. For example, let us suppose that we want to display compressed video data on a screen consisting of 640×360 pixels[2]. The screen has to be rendered at a rate of 30 frames per second. Also, the amount of processing that has to be done on a pixel before it can be displayed is of the order of 1000 operations. Thus the number of operations that have to be performed per second to display a screen is of the order of a billion. Displaying video or audio is but one of the many applications that have to be done in a media processing application. Media processing[3] often involves coding and decoding frames of image or audio data, compressing them or enhancing them.

While the previous paragraph might suggest that drastic measures have to be taken to enhance the performance of current day processors, a more pragmatic approach would be to borrow the wealth of ideas from the experience gained in designing special purpose Digital Signal Processors (DSP). Most multimedia applications operate on small data types *i.e* 8-bit pixels, 8-bit color component of a color frame and 16-bit audio data. Also many of these applications perform certain operations repeatedly on these data types and have a lot of data parallelism (see *Box 1*). These have been the main features that have been exploited in designing special purpose hardware for DSP. In a conventional DSP one can add small pieces of hardware to meet the demands of the application. In a general purpose processor such an approach is not possible since the application is not known a priori. To remedy the situation the architects of media processing enhanced processors support operations on what are called *packed data types* in an SIMD fashion.

[1] Multimedia: Includes information media such as images, video, audio, text, 2D and 3D graphics and numbers.

[2] Pixel (Picture element): The smallest element on a picture that can be manipulated in a graphics operation.

[3] Media Processing: A term used to describe processing of digital multimedia information. It usually consists of encoding, decoding, dithering, enhancing and rendering multimedia information.

Box 1. Single Instruction Multiple Data (SIMD) Processing

When computations can be performed on different segments of data simultaneously and there is no dependence of one computation on the other, we say that the application has *data parallelism*. To illustrate this further let us take the example of four persons trying to fix four wheels to a car. If all of them have separate spanners then they can work independently — each of them can fix a wheel. This kind of parallelism is exploited in SIMD architectures. In an SIMD architecture a single instruction issued by the processor works on different segments of data to speed up computation.

Another feature of media processing applications is the extensive use of matrix operations which in turn uses the *multiply-accumulate* operation frequently. To explain this let us take the example of the multiplication of two vectors to produce a scalar. Here, each element of a vector X has to be multiplied by a corresponding element in vector Y and the results of all such products 'accumulated' (added). To speed up this operation media processors support an instruction commonly called the *multiply-accumulate instruction* on packed data types. Having seen some of the basic characteristics of media processing applications let us study how these features reflect on MMX technology.

Inside an MMX Technology Enhanced x86 Processor

MMX defines packed data types in which data elements smaller than 64 bits are packed into a 64-bit register. A new set of instructions, 57 in number, have been defined that operate on these packed data types in an SIMD mode (see *Table 1*). The data types defined on MMX are (see *Figure 1*):

- Packed byte, wherein eight bytes are packed into a 64 bit register.
- Packed word, where four 16-bit data elements are packed into a single 64-bit register.
- Packed double word, in which two doublewords are packed into a 64-bit register.

Table 1 The MMX instruction set.

Opcode mnemonic	Options	Description	Throughput/ Latency
padd[b|w|d], psub[b|w|d]	Wraparound, Saturate	Adds or subtracts packed bytes, words or double words of 64-bit operands.	1/1
pmullw	-	multiplies four packed words from 64-bit operands in parallel and selects the lower-order 16 bits	1/3
pmulhw	-	multiplies four packed words from 64-bit operands in parallel and selects the higher-order 16 bits	1/3
pmaddwd	-	Parallel Multiply-Add of four signed 16-bit words. Adjacent pairs of 32-bit results so produced are added yielding two double word results.	1/3
pcmpeq[b|w|d]	-	Compare eight bytes, four 16-bit words or two 32-bit doublewords for equality	1/1
pcmpgt[b|w|d]	-	Compare eight bytes, four 16-bit words or two 32-bit doublewords for a "greater than" condition	1/1
pand pandn por pxor		64-bit bitwise logical operations	1/1
psll[w|d|q] pslr[w|d|q]		Logical shift of four 16-bit words, two 32-bit double words or one 64-bit quadword left/right in parallel	1/1
psra[w|d]		Arithmetic shift of four 16-bit words or two 32-bit double words right in parallel	1/1
punpckl[bw|wd|dq]	-	Unpack eight bytes, four 16-bit words or two 32-bit double words from lower 32-bits.	1/1
punpckh[bw|wd|dq]	-	Unpack eight bytes, four 16-bit words or two 32-bit double words from higher 32-bits.	1/1
packss[wb|dw]	Always Saturate	Pack doublewords to words or words to bytes	1/1
mov[d|q]		Move 32 or 64 bits to or from memory to MMX registers or between MMX registers. 32-bit moves can also be done between the integer and MMX register.	1 (if cache hit)
emms		Empty FP registers tag bits	varies by implementation

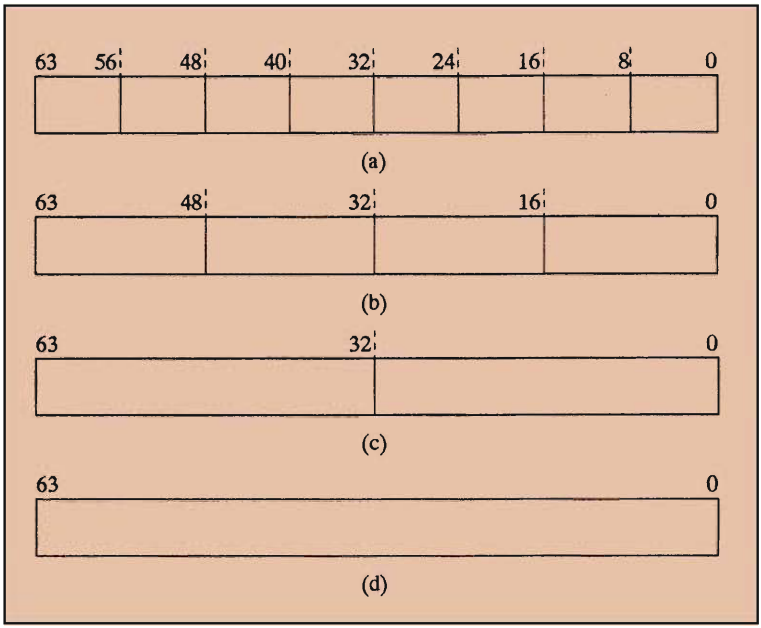
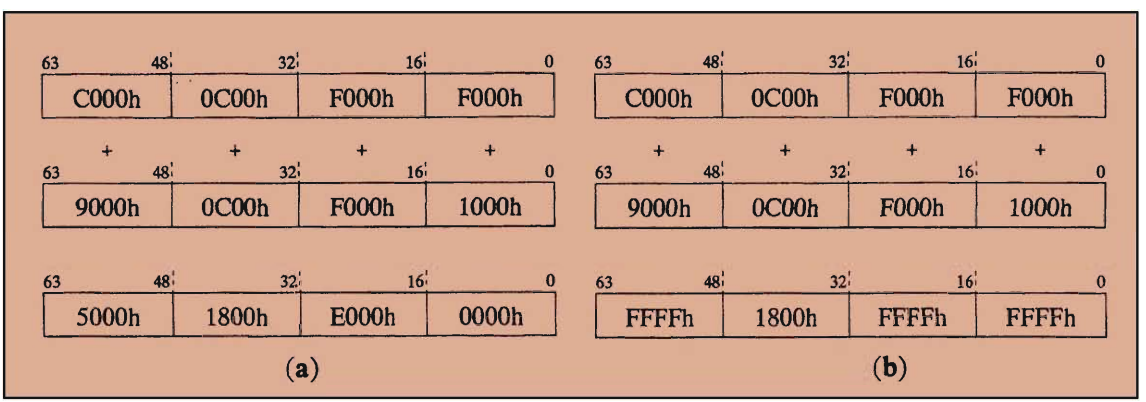


Figure 1 Data types defined on MMX: (a) packed byte (b) packed word (c) packed doubleword and (d) quadword.

- Quadword, here the whole register represents a single logical data element.

Instructions that extend the basic integer operations to operate on packed data types have been defined in MMX (see *Box 2*). These include the packed addition and subtraction operations (see *Figure 2*), packed multiplication, packed

Figures 2 (a) Wrap-around addition on packed word datatype. The h at the end of a number indicates that the number is in hexadecimal. (b) Saturating addition on packed word data type.



Box 2. Wrap-Around and Saturating Arithmetic

In a conventional processor, when an arithmetic operation results in an overflow or underflow the most significant bit is truncated. This leads to an effect called *wrap-around* in which the sum of two large numbers can result in a number which is smaller than either of them. This sort ofarithmetic is called *wrap-around arithmetic*. In saturating arithmetic this effect is avoided and the result of an operation yielding an overflow or underflow becomes the largest or smallest possible representable number in the data type of the operation. While conventional processors support only wrap-around arithmetic, media processors also support saturating arithmetic. This type of arithmetic is particularly useful for processing images. For example, while shading a 3D image using a technique called *Gouraud shading* so that pictures look realistic, polygons are shaded by interpolating color pixel values across scan lines during rendering. It might so happen that somewhere along a scan line calculations start overflowing. If saturating arithmetic is not used, a dark polygon shaded towards black may suddenly start having white pixels. Saturation makes sure that the result is clamped to the maximum dark value and does not overflow to white.

multiply-accumulate (see *Figure 3*), packed compares (see *Figure 4*), packed shift, logical operations such as AND, ANDNOT, OR and XOR, memory transfer and the conversion operations that promote/demote an operand in the registers to a data type of higher/lower precision (see *Figure 5*). In addition to the above mentioned classes of instructions an empty MMX state operation has been defined to provide compatibility with existing software and operating systems.

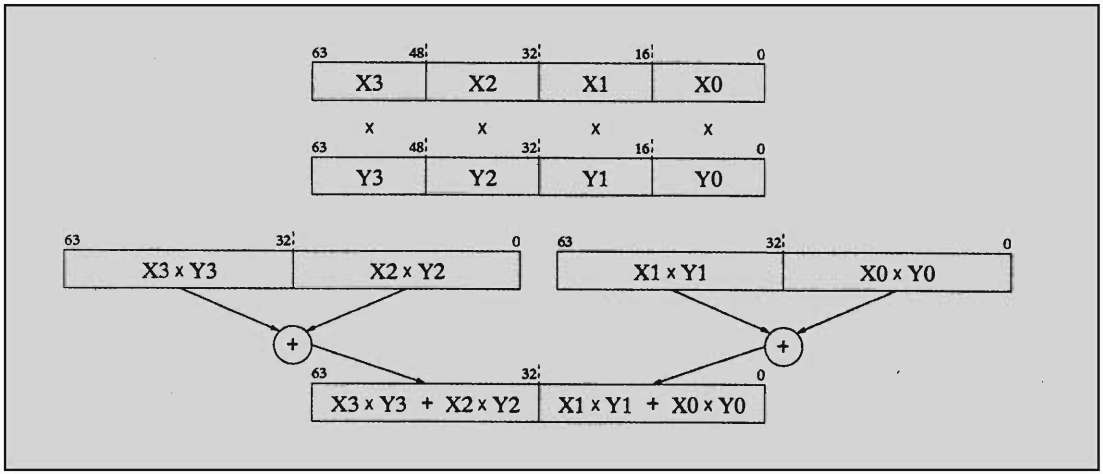
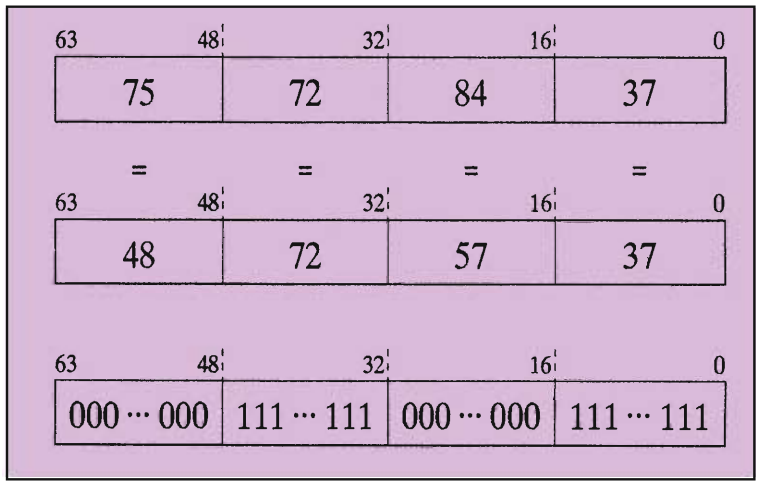


Figure 3 The multiply-accumulate instruction in MMX.

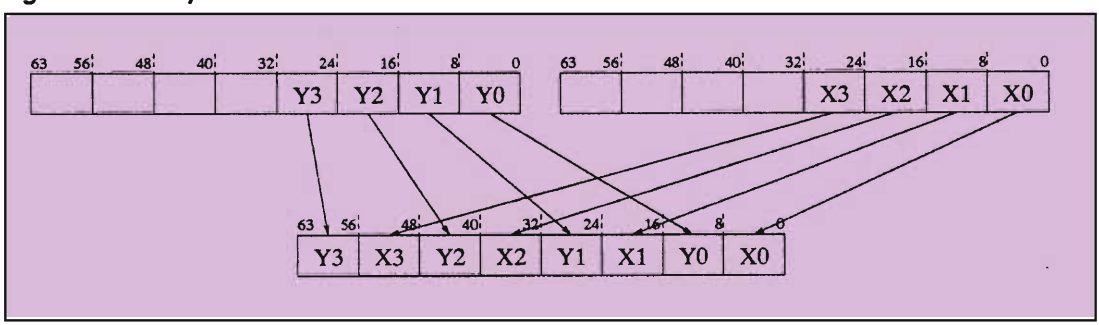
Figure 4 The compare instruction.



MMX Techniques Explained

We will now look at a few examples to gain a first hand knowledge of how MMX technology is exploited to enhance media applications. While writing MMX code for an MMX enhanced Pentium Processor, one should keep in mind the fact that an Intel Pentium Processor can execute only up to two instructions in parallel. To extract maximum performance out of the processor one should therefore order the instructions so that pairs of instructions are independent of each other. A pair of instructions that can be executed in parallel can be either two integer instructions (as on a regular Pentium processor) or two MMX instructions or one of each.

Figure 5 The unpack instruction.



Absolute Differences of Pixels with MMX

The core of motion estimation — a technique used to compress video data, essentially involves finding the sum of the absolute differences (or differences of the squares) of two 16×16 blocks of pixel values to find a block with the least such sum. A good idea would be to find the pixel differences in parallel. The only problem is the fact that a subtraction of two 8-bit unsigned pixel values can yield a 9-bit negative number. It is here that the saturating arithmetic of MMX can be used effectively to prevent any negative numbers from being produced. The main idea is to subtract pixel B from pixel A and then subtract pixel A from B. Since the operations are being performed using saturating arithmetic, one of the differences will be positive while the other will be clamped to 0. However, we do not know which of the two results is the positive number. Hence, we just do a logical OR of the two results to get the final result. Using the instruction psubusb we can actually perform eight such operations in parallel, reaping great performance benefits. *Figure 6* gives the flow diagram to calculate the absolute differences between two

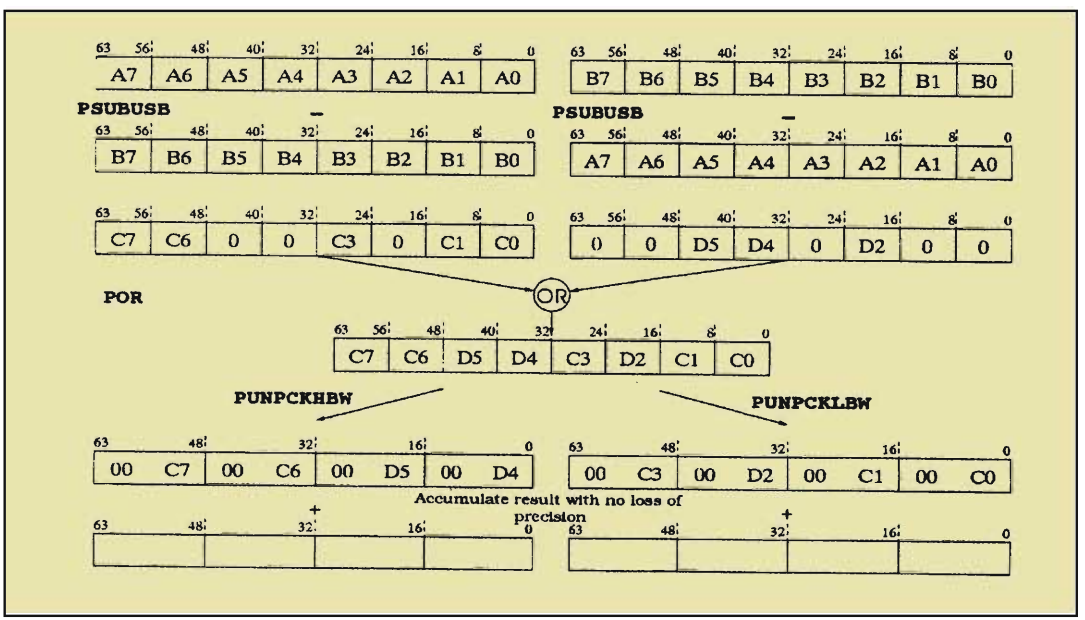


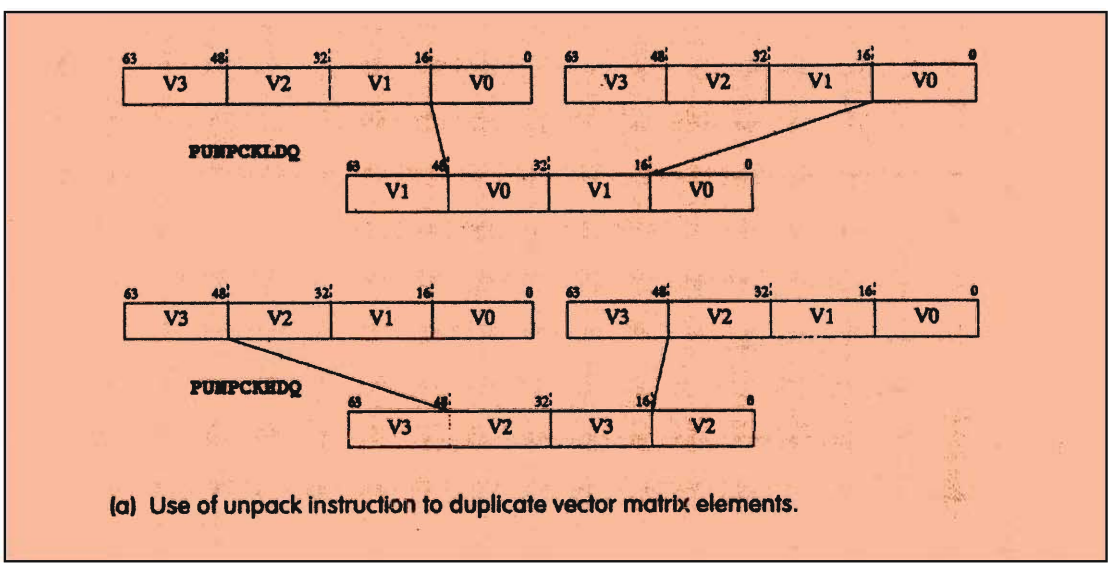
Figure 6 Absolute differences of pixel values using packed byte datatypes.

8×8 sub-blocks of pixel values using the packed data types of MMX. One can also write an optimized scalar code without using the MMX capabilities and operating on one data element at a time. It has been found that such an optimized code for a non-MMX Pentium processor is only half as fast as a code for an MMX enhanced Intel Pentium processor.

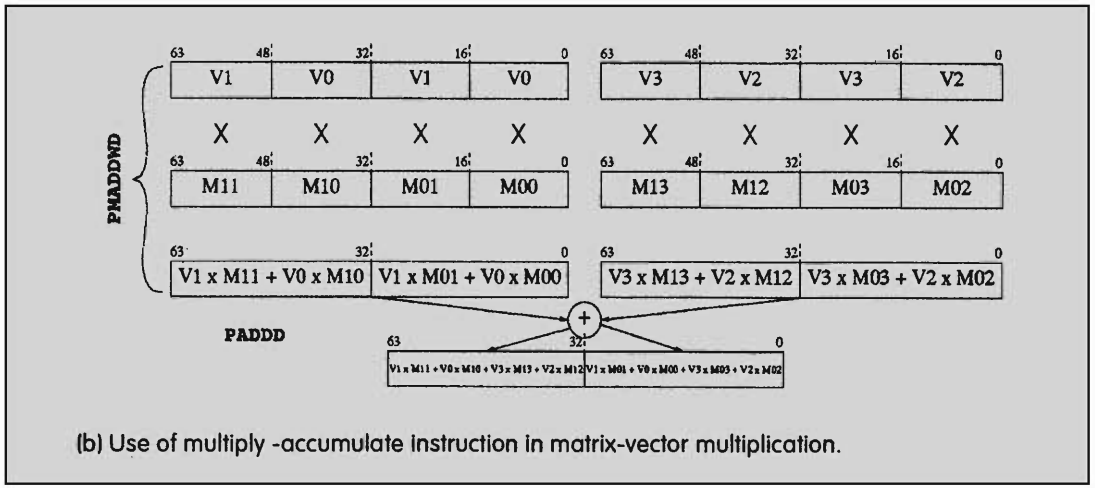
Matrix-Vector Multiplication with MMX

We will now look into an efficient matrix-vector multiplication routine using MMX technology. Matrix-vector multiplication forms the core of many programming problems and hence an efficient code using MMX can speed up an application dramatically. We will deal with packed data elements that are 16-bit signed values as it has sufficient precision for most image proccessing applications. For example, we have shown the flow diagram for multiplying a 4×2 matrix by a four-element vector (*Figure 7a*). It shows how duplicating a pair of data elements of a matrix/vector enables the pmwaddwd instruction to simultaneously multiply two rows of a matrix with a vector as shown in

Figures 7a & b Matrix-vector multiplication using packed word datatype.



(a) Use of unpack instruction to duplicate vector matrix elements.



(b) Use of multiply -accumulate instruction in matrix-vector multiplication.

Figure 7b. The pmwaddwd instruction accumulates results at 32-bits of precision and hence prevents any loss of precision.

Conclusions

MMX technology is not one of its kind in media processing. There are other processors like the Ultra Sparc of SUN which uses a technology called VIS and the PA-RISC 2.0 which uses a technology called MAX-2. These processors exploit the special properties of media applications using what is in general called *subword parallelism* in a way very similar to MMX technology. The designs of these processors have been driven by the needs of the target markets which range from lower end multimedia (for example, games) to high fidelity multimedia (for example, work stations and medical imaging). This range being very broad, it will not be surprising if in future all processors are enhanced with MMX type instructions.

Suggested Reading

- **Marc Trembley, J Michael O'Connor, Venkatesh Narayanan and Liang He. VIS Speeds New Media Processing.** *IEEE Micro.* **pp 10-20, August 1996.**
- **Alex Peleg, Sam Wilkie and Uri Weiser. Intel MMX for Multimedia PCs.** *Communications of the ACM.* **Vol.40. No.1, January 1997.**

Address for Correspondence
S Balakrishnan
Supercomputer Education and Research Centre
Indian Institute of Science
Bangalore 560 012, India.
email:sbalki@serc.iisc.ernet.in