The VIS™ Instruction Set

Version 1.0

June 2002

A White Paper

This document provides an overview of the Visual Instruction Set.

4150 Network Circle
Santa Clara, CA 95054 USA
www.sun.com

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

With the majority of hardware manufacturers now including Single Instruction Multiple Data (SIMD) capabilities in their processors, SIMD processing has become a hot topic. However, SIMD, which refers to the exploitation of data parallelism and the application of a single instruction to multiple data entities in parallel, is not a new idea. Historically, this technique was used by large vector machines, typically containing multiple processor elements which were slaves to a single control unit.

The potential benefits of employing this style of processing are obvious: by processing X elements in parallel, the time taken to process a total of N elements can theoretically be reduced by a factor of X, in comparison to that achievable with a standard Single Instruction Single Data (SISD) implementation. Despite these benefits, this style of processing failed to gain widespread acceptance and the majority of general purpose machines created to-date have been SISD in nature. It is only over recent years, as an alternative form of SIMD processing became practical – and driven to a large extent by interest in multimedia applications, which greatly benefit from this technique – that this concept has finally become accepted into mainstream computing. This new technique does not require the use of multiple processors, but rather focusses on employing SIMD within a single processor.

Since the 1980s ALUs, registers and datapath widths have increased from 8 → 16-bits to 64-bits and beyond. Yet, in many applications, the majority of the variables remain much smaller than 64-bits. One class of applications which typifies this is multimedia applications, which primarily operate on 8- and 16-bit variables. The functionality to handle much larger variables than are required in practice raises the possibility of attempting to pack multiple data elements into one 64-bit register and then process all of these smaller elements of data in parallel – SIMD within a register.

This concept was championed by Sun Microsystems, which in 1995 introduced into their flagship the UltraSPARC® processor, dedicated hardware to facilitate this form of SIMD processing. Named the Visual Instruction Set (VIS™ instruction set), this enhancement represented the first comprehensive SIMD instruction set extension on a general purpose processor.

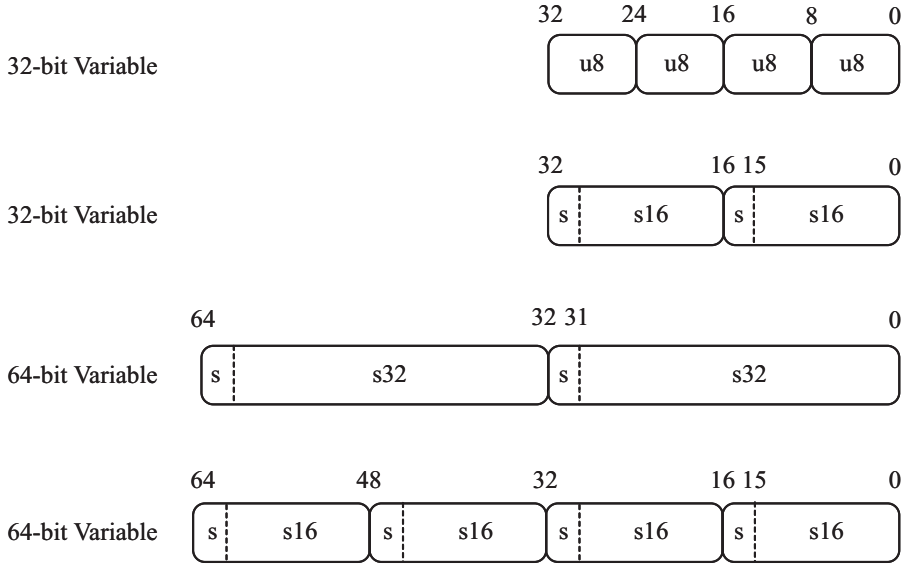
CHAPTER 2

The VIS Instruction Set

The VIS instruction set is a set of "RISC-style" SIMD instructions which are extensions to the standard SPARC® V9 instruction set. Initially introduced with the UltraSPARC I processor, these integer SIMD instructions were primarily focussed on boosting the performance of multimedia applications. Capitalizing on the realization that variables are frequently smaller than the width of the processor's ALUs, registers and datapaths, the VIS instruction set breaks with the traditional view of one-register-one-variable. Instead, the VIS instruction set views the UltraSPARC processor's registers as being virtually partitioned to contain a number of smaller variables, frequently referred to as packed data types. Providing support for integer SIMD operations, the VIS instruction set utilizes several different register partitioning formats and operates on 8-, 16- and 32-bit packed integers, as is illustrated in *Figure 2-1*. With the UltraSPARC processor's 64-bit registers, these formats facilitate both two- and four-way parallelism.

In *Figure 2-1*, the VIS instruction set partitions 32-bit and 64-bit floating point registers to hold multiple short (8-, 16-, and 32-bit) integer variables.

Figure 2-1 The VIS Instruction Set Data Types



While targeted at integer data types, the VIS instruction set was incorporated into the UltraSPARC processor's Floating Point Unit (FPU). The decision to segregate the integer and integer SIMD units has a couple of important performance benefits. Firstly, by integrating the VIS instruction set into a different set of pipelines, both the VIS instruction set and the integer units can be utilized in parallel, maximizing instruction level parallelism. Secondly, this segregation has the additional benefit of maximizing the number of usable registers, with the VIS instruction set utilizing a different register set to the standard integer units.

The VIS instruction set, which has recently been expanded with the debut of VIS 2.0 in Sun's UltraSPARC III processor, provides the entire suite of instructions required to facilitate integer SIMD operations in the FPU. The VIS instruction set is comprised of over 80 instructions which can be loosely categorized as follows *[1][2]*:

- VIS 1.0 (available on all UltraSPARC processors):
  - Arithmetic and logical instructions
  - Comparison instructions
  - Format conversion instructions, to provide an easy mechanism to switch between the different supported data formats
  - Instructions to handle misaligned data
  - Memory access instructions suited to high bandwidth data movement and the retrieval of the supported data formats
  - Array instructions to provide efficient access to large three dimensional data sets
  - An instruction specifically targeted at the motion estimation required in MPEG
- VIS 2.0 (available starting with the UltraSPARC III processor) also includes:
  - A highly flexible data manipulation instruction

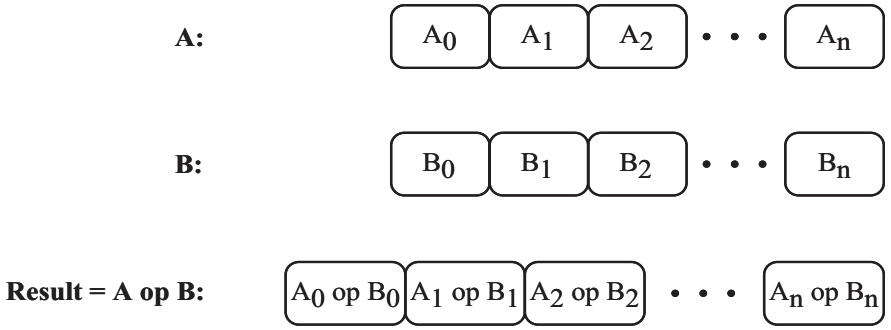
These instruction classes are now discussed in more detail.

2.1. Arithmetic and Logical Instructions

The VIS instruction set provides the ability to add, subtract, and multiply up to four different variables in one operation. In common with standard SISD arithmetic operations, the operands continue to reside in separate registers, with the packed elements in each source operand interacting with identically positioned elements in the other source operand to produce the result, as is illustrated in *Figure 2-2*. Functionality is provided to perform 16- and 32-bit SIMD additions and subtractions, as illustrated in *Figure 2-3* and *Figure 2-4*. Multiplication operations center on 8-bit by 16-bit SIMD multiplications, with the results either truncated to 16-bits (as illustrated in *Figure 2-5*) or expanded to 32-bits. Multiplications can be combined to facilitate 16-bit by 16-bit operations.

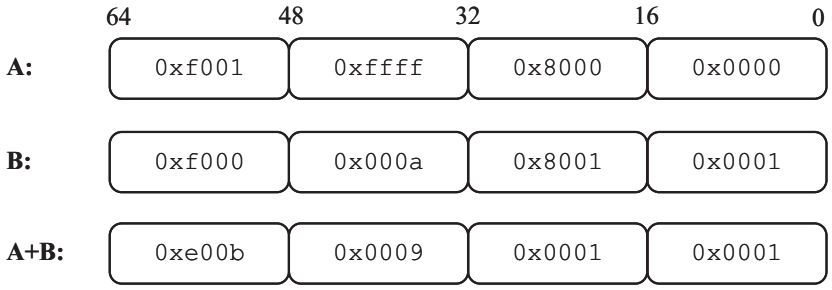
In *Figure 2-2*, the VIS instruction set SIMD instructions exploit the SPARC processor's three operand instruction set architecture. The packed variables contained in source operands A and B interact in a pair-wise fashion to generate the result.

Figure 2-2 VIS Instruction Set SIMD Instructions Exploit the SPARC Processor's Three Operand Instruction Set Architecture



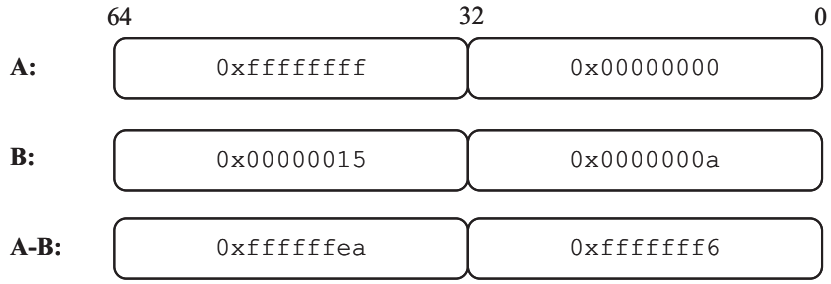
In *Figure 2-3*, the signed 16-bit packed variables in A and B are added together in a pair-wise fashion.

Figure 2-3 The 16-bit SIMD VIS Instruction Set Addition Instruction



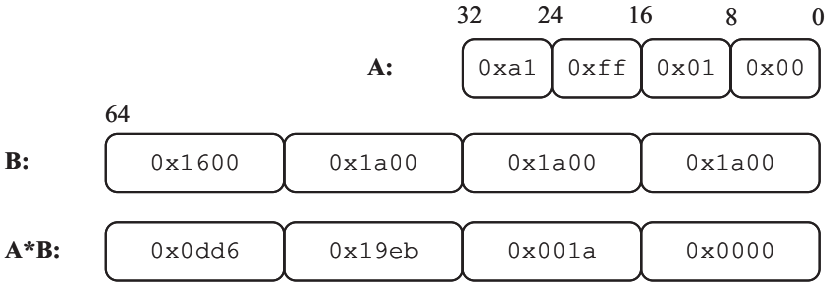
In *Figure 2-4*, the signed 32-bit packed variables in B are subtracted from those in A in a pair-wise fashion.

Figure 2-4 The 32-bit SIMD VIS Instruction Set Subtraction Instruction



In *Figure 2-5*, the packed variables contained in A (unsigned 8-bit variables) and B (signed 16-bit variables) are multiplied together in a pair-wise fashion. For this particular variant of the VIS instruction set multiplication instruction, only the upper 16-bits of each of the 24-bit products are retained and appear in the result, although the VIS instruction set provides other multiplication instructions which retain the entire product.

Figure 2-5 VIS Instruction Set 8x16-bit –> 16-bit SIMD VIS Instruction Set Multiplication Instruction



The VIS instruction set also provides a full suite of logical instructions. While SIMD logical operations can be trivially performed using standard integer logical instructions[1], the VIS instruction set provides its own logical instructions in order to avoid the requirement to move the data back to the integer registers every time a logical operation is required.

2.2. Comparison Instructions

The VIS instruction set contains a number of variants of the SIMD compare instruction, with functionality being provided to undertake the following operations on 64-bit variables, with either 16- or 32-bit partitioning:

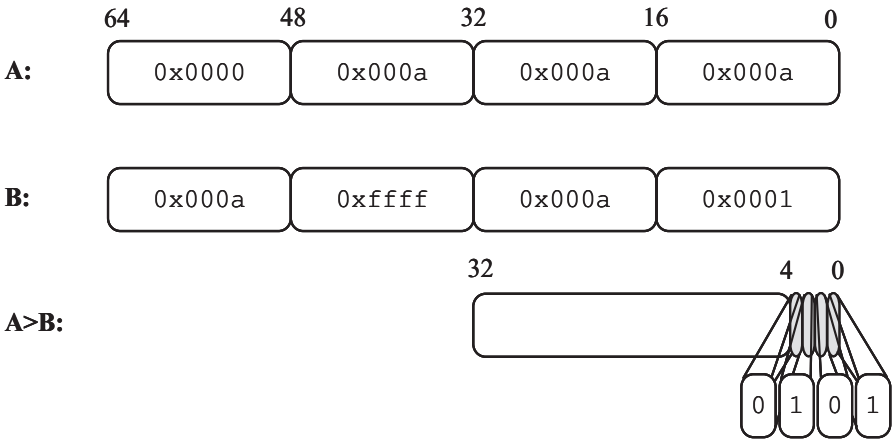
- less-than
- less than or equal to
- greater-than
- greater than or equal to
- equal to
- not equal to

The instructions return a 2- or 4-bit bit field contained in the low order bits of a 32-bit integer, with each bit representing the outcome of a comparison, as is illustrated in *Figure 2-6* and *Figure 2-7*.

In *Figure 2-6*, the signed 16-bit packed variables in A and B are compared with each other in a pair-wise fashion. This figure illustrates a greater-than operation, with the 4-bit bit-field contained in the result indicating, for each of the four pairs of packed variables, whether A is greater-than B (bit set on true).

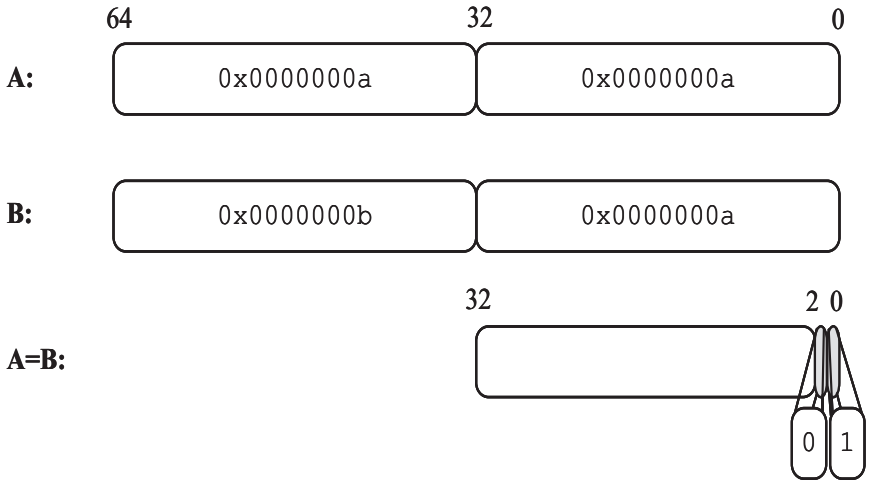
1. The VIS instruction set operates on the floating point (FP) registers.

Figure 2-6 The 16-bit SIMD VIS Instruction Set Compare Instruction



In *Figure 2-7*, the signed 32-bit packed variables in A and B are compared with each other in a pair-wise fashion. This figure illustrates an equal-to operation, with the 2-bit bit-field contained in the result indicating, for each of the two pairs of packed variables, whether A is equal-to B (bit set on true).

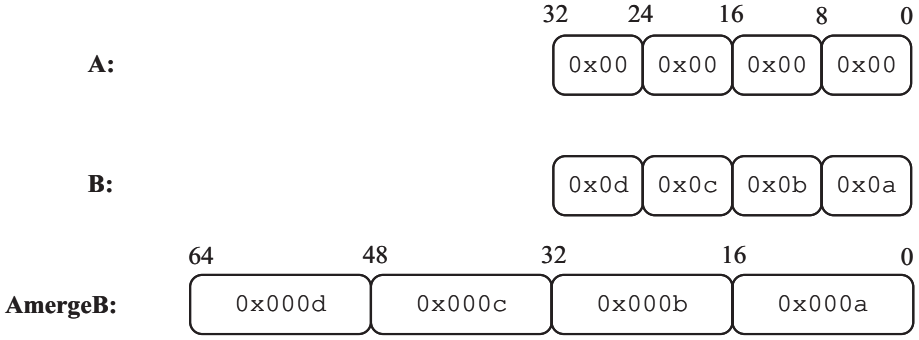
Figure 2-7 The 32-bit SIMD VIS Instruction Set Compare Instruction



2.3. Format Conversion Instructions

These conversion instructions convert SIMD data of one type into another, facilitating changes in both the level of parallelism and the degree of precision achievable. The VIS instruction set provides 32 ↔ 8, 32 ↔ 16 and 16 ↔ 8-bit conversion operations. *Figure 2-8* illustrates one of the most frequently used instructions of this class, `fpmerge`, which generates a 64-bit result by combining alternate bytes from A and B.

Figure 2-8 One of the Many Format Conversion Instructions Provided in the VIS Instruction Set



2.4. Data Misalignment Handling Instructions

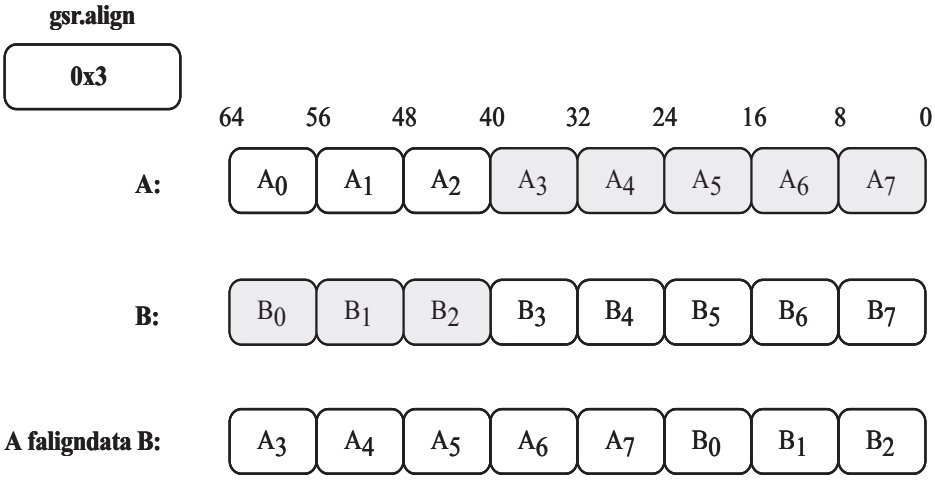
When developing SIMD code, in order to achieve maximum performance, it is beneficial if the packed data elements that are the target of the SIMD instructions are loaded using as few operations as possible. This normally requires that the two, four or eight packed data elements that constitute an operand be considered as a 64-bit entity and retrieved using a single 64-bit load instruction (and similarly for store). Consequently, in addition to the requirement to ensure that these elements occupy contiguous locations in memory, the memory model mandates that this 64-bit variable be correctly aligned in memory, i.e., the first packed data element must be aligned on an 8-byte boundary.

This alignment criteria is not always easy to satisfy – especially on legacy applications – and data misalignment can be a serious problem when developing SIMD code. However, the VIS instruction set readily overcomes this problem by providing the `faligndata` instruction which can be used to resolve arbitrary byte misalignment.

In *Figure 2-9*, the VIS instruction set `faligndata` instruction, which is utilized for resolving misalignment problems, concatenates A and B and then extracts eight contiguous bytes. The start byte is stipulated by the align field in the `gsr` register[2]. The shaded bytes in this figure illustrate the bytes which are extracted from A and B when the align field is set to 0x3.

2. The correct value for the align field in the `gsr` register can be trivially computed using the VIS instruction set `alignaddr` instruction: provided with the address of the first required byte, this instruction sets the align field appropriately and returns the correct 8-byte aligned address for A.

Figure 2-9 The VIS Instruction Set `faligndata` Instruction



2.5. Data Access Instructions

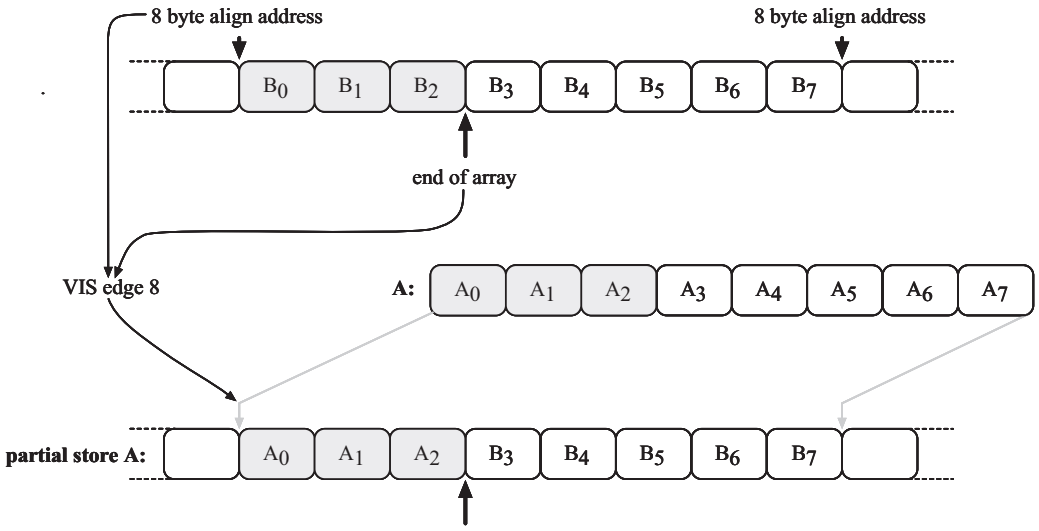
The SPARC processor's FPU was originally designed to process 32-bit single precision or 64-bit double precision FP numbers and provided no facility to load and store the 8- and 16-bit variables frequently utilized by the VIS instruction set. Consequently, the VIS instruction set provides additional instructions to load and store these variables to and from the FP registers.

Additionally, the VIS instruction set provides a partial store instruction. This requirement is a reflection of the SIMD nature of the VIS instruction set and the realization that it may be necessary to alter any of the packed variables contained in an 8-byte partitioned variable, while leaving the remainder unchanged. The packed variables to be updated can be readily stipulated using the VIS instruction set edge or compare instructions, as is illustrated in *Figure 2-10*. In *Figure 2-10*, the VIS instruction set partial store instruction overwrites only select bytes in an 8-byte variable. This instruction is usually used in conjunction with the VIS instruction set `edge` instruction, which can be utilized to quickly generate, from the user supplied stop address, the mask required by the partial store.

Finally, the VIS instruction set also facilitates the rapid transfer of large blocks of data. This is accomplished with the block load and store instructions which facilitate, via a single instruction, the transfer of 64-bytes of data between the FP registers and memory. These operations bypass the data caches, allowing large quantities of data to be processed without cache thrashing.

Block load and store instructions have been used with great success to accelerate functions such as `memcpy`, frequently more than doubling performance.

Figure 2-10 The VIS Instruction Set Partial Store Instruction



2.6. Fast 3-D Array Access Instructions

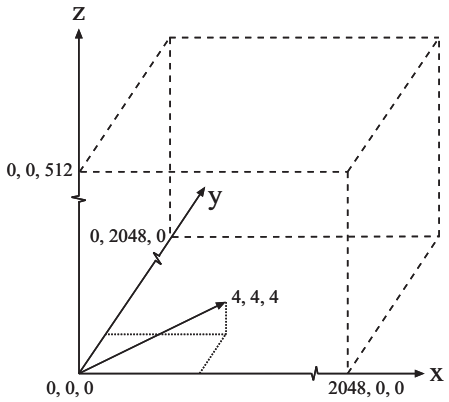
With the ever increasing discrepancy between processor speeds and the speed of the main memory, ensuring that the required data is located in cache memory is increasingly important. When performing operations such as 3-D texture mapping and volume rendering, the standard data ordering ensures that, when performing a data lookup based on x, y and z coordinates, even a small change in the z coordinate will frequently reference distant memory locations, resulting in frequent stalls due to TLB and cache misses.

These stalls can be minimized by arranging the data in a blocked fashion, such that points with spacial locality exhibit locality in memory, as illustrated in *Figure 2-11*. However, the benefits of this blocked data organization are normally counterbalanced by difficulty associated with transforming, under this new mapping, x-y-z coordinates into memory addresses.

The VIS instruction set addresses this problem by providing an instruction to rapidly compute the memory address of the required data element when supplied with its x, y, and z coordinates, allowing this blocked data layout to be more easily utilized.

Figure 2-11 illustrates the locality benefits of employing the block data formatting structure when tracing a ray through 3-D space; the addresses of successive voxels are normally widely separated in memory. For example, with the illustrated ray, the separation in memory of the first voxel (1,1,1) from the origin (0,0,0) is 8 MB (16-bit data elements). Using the VIS instruction set blocked data formatting structure, successive elements are much closer, minimizing TLB and cache misses, i.e., the 1,1,1 voxel is now only 42-bytes from the origin.

Figure 2-11 The Locality Benefits of Employing the Block Data Formatting Structure

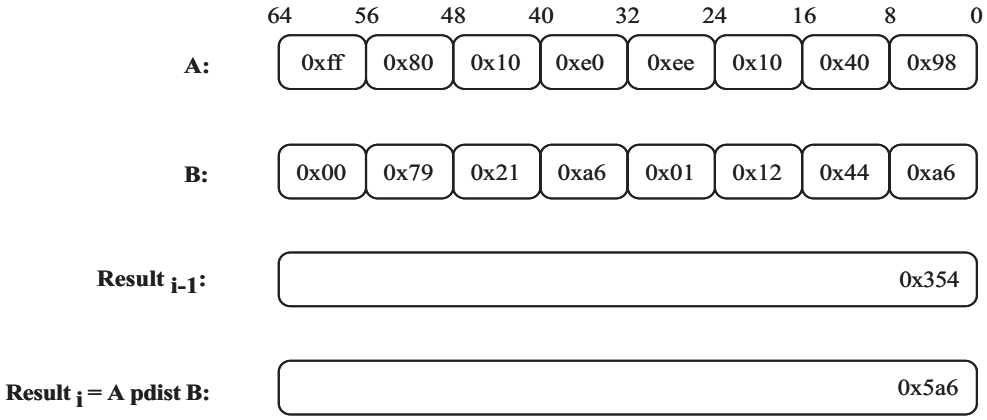


Voxel address	Standard offset	VIS offset
1, 1, 1	8MB	42B
2, 2, 2	16MB	16.01KB
3, 3, 3	24MB	16.06KB
4, 4, 4	32MB	33.06KB

2.7. PDIST Instruction

The VIS instruction set `pdist` instruction computes the sum of the absolute values of the differences between the eight 8-bit pixel pairs contained in two 64-bit variables. Additionally, `pdist` combines this new accumulation total with the result already present in the result register, as is illustrated in *Figure 2-12*. In *Figure 2-12*, the VIS instruction set `pdist` instruction computes the absolute difference between the eights pairs of byte variables contained in A and B, sums these results and then finally combines this new total with the value already present in the result register to generate the final result. This instruction is extremely useful for accelerating video encoder applications and its operation is discussed in more detail in Section 4.1.

Figure 2-12 The VIS Instruction Set `pdist` Instruction



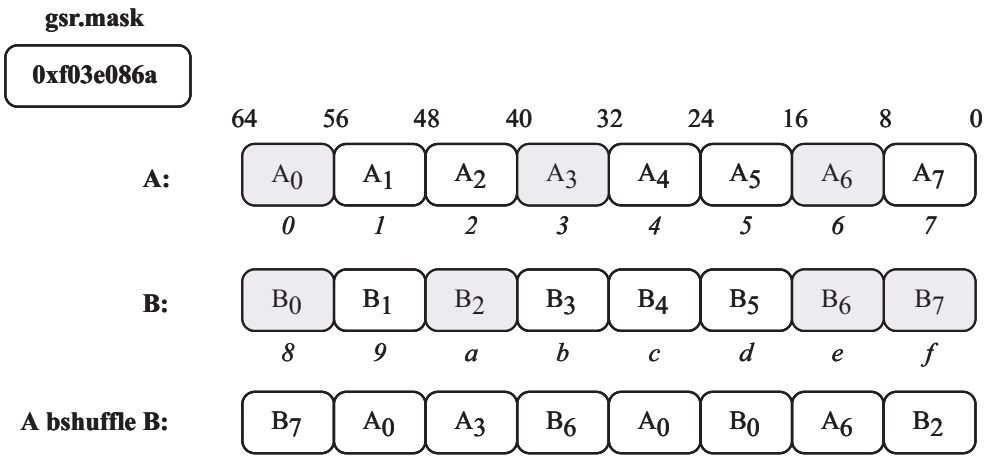
2.8. Data Manipulation Instructions

The introduction of VIS 2.0 with the UltraSPARC III processor saw the first expansion in the functionality provided by the VIS instruction set. The primary new feature introduced in VIS 2.0 was the byte shuffle instruction (`bshuffle`), which facilitates the arbitrary mixing of the bytes that comprise two 64-bit variables, as is

illustrated in *Figure 2-13*. The mix of bytes in the result is controlled by a 32-bit field (termed the `mask` field) in a dedicated global register (the `gsr` register). This instruction represents a much more efficient and powerful tool for data manipulation than was available under VIS 1.0 (until this point it had been necessary to use the format conversion instructions to manipulate the ordering of packed data elements within a 64-bit variable).

In *Figure 2-13*, the mask field of the `gsr` register controls the mapping of the bytes contained in A and B into the result. For instance, in this example, the first digit (f) in the mask field stipulates that the 15th byte of the input operands (B[7]) will appear as the first byte in the result.

Figure 2-13 The VIS Instruction Set Data Manipulation Instruction



CHAPTER 3

mediaLib™ Software

Developing SIMD code can frequently be problematic and normally requires that the developer have a detailed knowledge of the target processor and code the algorithms of interest directly in assembler. To make the VIS instruction set more easily accessible to general applications, Sun provides two methods via which the VIS instruction set can be used with ease.

Firstly, the VIS instruction set can be utilized by employing inline macros. These macros, which are provided as part of Sun's VIS Software Developers Kit *[3]*, allow the VIS instruction set instructions to be accessed from a high level language like C, significantly simplifying and accelerating the development process.

Secondly, Sun provides one further level of abstraction – mediaLib™ software. mediaLib software is a vast library of routines that is freely available for download from Sun Microsystems *[4]* and provides high performance VIS instruction set implementations of core algorithms from six primary functional areas, as is illustrated in *Table 3-1 [5]*. By utilizing these libraries, developers gain instant access to over 3,000 functions, all of which have been tuned for the UltraSPARC processor and provide speedups of between 1.3 times to 10 times, with respect to standard SISD implementations.

Additionally, by providing a simple, platform independent interface and making minimal assumptions about the way the user data is presented, minimal code modifications are normally required in order to use mediaLib software. Accordingly, mediaLib software, which can be used from C, C++ and Java™ programming languages, provides the performance benefits of the VIS instruction set, but at a minute fraction of the normal development time.

To-date, mediaLib software is the most complete library of its type and is significantly more comprehensive than similar offerings from other processor vendors. Nevertheless, the mediaLib software development remains an ongoing process that is being constantly expanded and reworked to ensure peak performance on the latest UltraSPARC processors. This has the advantage of removing the requirement that developers reoptimize the core components of their codes to reflect any microachitectural changes in next generation processors, greatly simplifying the migration process.

Table 3-1 An Overview of the Functionality Provided by mediaLib Software

mediaLib Software Components					
Algebra	**Graphics**	**Image**	**Signal**	**Video**	**Volume**
Matrix operations	Draw and fill circles and ellipses	Basic Arithmetic: add, ave., blend	Basic DSP: FFT, FIR, IIR	DCT/IDCT	Maximum intensity searching
Vector operations	Draw and fill polygons	Convolution and auto correlation	ADPCM	Motion estimation	Ray casting functions
Data manipulation operations	Draw lines and arcs	Max, min, thresholding	Windowing	Color conversion	
		Rotate and zoom	Convolution and correlation	Interpolation	

CHAPTER 4

Applications

While the VIS instruction set, and indeed all of the other SIMD instruction set extensions, were originally targeted at multimedia applications and provide significant performance gains in this type of application, these instructions have proved very beneficial in a much wider sphere of applications:

- Bioinformatics
- Cryptography
- Database systems
- Digital Signal Processing
- 3-D Visualization
- Graphics and Imaging
- Multimedia
- Networking
- Telecommunications

Several of these important application areas are now discussed in more detail.

4.1. Multimedia

Video encoding, a key component of multimedia applications, frequently involves a computationally intensive technique referred to as *motion estimation*. This technique involves computing the difference between two groups of pixels to determine if elements in the current picture exist at some location in another image. If a close enough match can be found, then these pixels need not be retransmitted. Instead, they can be replicated merely by relaying their displacement in the subsequent images, facilitating substantial reductions in bandwidth. In the majority of implementations of the current video compression standards, this measure of difference is computed by determining the absolute difference between pairs of pixels and then summing this error over a block of pixels – referred to as the Sum of the Absolute Difference (SAD) computation – and frequently accounts for the large majority of the computational overhead associated with encoding.

The VIS instruction set `pdist` instruction is specifically targeted at accelerating this computationally intensive encoding technique. By facilitating the accumulation of the absolute difference between eight pairs of pixels in a single cycle, it reduces the

number of instructions required to perform the SAD computation by over a factor of ten. By providing support for these computations, motion estimation calculations can be accelerated by over 500% (the VIS instruction set also accelerates a wide variety of other core video compression algorithms, as illustrated in *Table 4-1*), facilitating significant streamlining of video encoding and the other applications – such as pattern matching and character recognition applications – which rely heavily upon this technique.

Table 4-1 illustrates the VIS instruction set based performance gains for core components of video compression applications. Performance gains are in comparison to an optimized SISD implementation.

Table 4-1 VIS Instruction Set Based Performance Gains for Core Components of Video Compression Applications

Algorithm	Performance Gain, %
Motion estimation	500
Quantization	393
Interpolation	326

4.2. DSP Algorithms

Arithmetic instructions form the core of the VIS instruction set, providing the functionality to perform multiple arithmetic operations in parallel. The variety of algorithms which can benefit from this functionality is virtually limitless and encompasses such crucial algorithms as DSP (including FFTs, DCTs and filters), checksums and a wide variety of algebraic functionality for vectors and matrices. By providing 4-way parallel arithmetic operations and allowing the dispatch of up to two of these SIMD instructions every cycle, the VIS instruction set provides four times or greater speed-up for many algorithms.

For example, consider the two dimensional Inverse Discrete Cosine Transform (IDCT). This algorithm is a central component of today's video compression standards *[6]*, and, in MPEG, operates on 8x8 blocks of sub 16-bit data, frequently accounting for over 30% of the processing overhead in an SISD decoder implementation.

For standard SISD implementations of IDCTs, minimizing the number of multiplications is frequently of critical importance, favoring implementation using true 2-D techniques. However, the VIS instruction set provides a low-latency, pipelined, SIMD multiplication unit, moving the emphasis toward reducing the number of the predominant addition operations. This favors the exploitation of the IDCT's separable nature and implementation as a series of row and column based one dimensional computations. For an 8x8 data block, the 2-D operation can be computed by eight row based 1-D IDCTs followed by eight column based 1-D IDCTs.

Considering the likely memory organization of the 2-D data blocks, it is readily apparent that undertaking row based IDCTs using the VIS instruction set cannot be

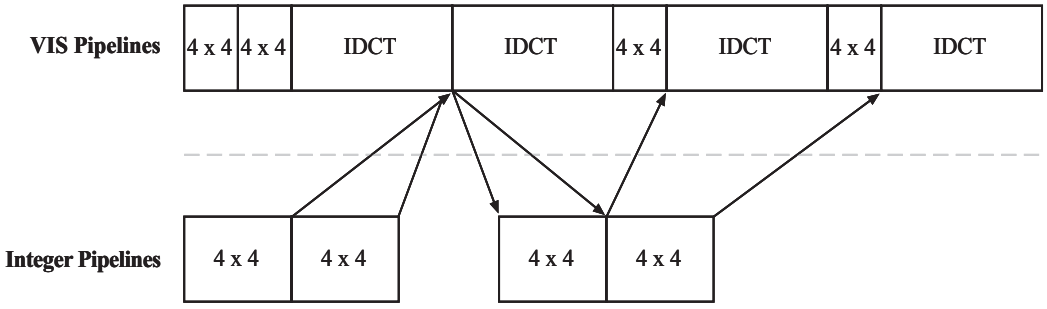
achieved without significant data reorganization. Consequently, it transpires that it is optimal to undertake the 2-D IDCT as follows: undertake a series of 1-D column based IDCTs, transpose the 8x8 block of semi-transformed data and then perform a further set of column based IDCTs.

For 16-bit data elements, the VIS instruction set provides the functionality to perform four 1-D column based IDCTs in parallel. As discussed in detail in Chapter 6, *Competitive Analysis*, the UltraSPARC processor facilitates the dispatch of one VIS instruction set addition / subtraction and one VIS instruction set multiplication per cycle, allowing the four 1-D IDCTs to be performed in just 26-cycles (an eight point scaled 1-D Chen IDCT requires eight multiplications and twenty-six additions *[6]*).

The data must then be transposed. For VIS 1.0, this was achieved using `fpmerge` operations (illustrated in *Figure 2-8*), but with VIS 2.0, the `bshuffle` instruction (illustrated in *Figure 2-13*) significantly allows this to be undertaken more elegantly – reducing the overhead associated with the transposition by 50%. After the data is transposed, a further set of 1-D IDCTs are then performed on the data and the transform is complete.

In comparison to a SISD FP implementation of the IDCT, utilizing the VIS instruction set boosts performance by over three times. Yet, performance can be further increased. During this computation, the integer pipelines are essentially idle, with the vast majority of the processing performed in the FP pipelines using the VIS instruction set. However, the computation can be readily partitioned to capitalize on this unused resource, such that the integer pipelines are utilized to perform operations in parallel with the VIS instruction set based processing, as is illustrated in *Figure 4-1* (the boxes labelled 4x4 refer to 4x4 transpose operations, while the IDCT boxes refer to sets of four 1-D IDCTs). Utilizing this approach makes full use of the UltraSPARC processor's superscalar capabilities, dispatching four instructions during most cycles and performing up to ten arithmetic operations every cycle.

Figure 4-1 One Possible Partitioning of the IDCT Computation Between the VIS Instruction Set and the Integer Pipelines



Adopting this hybrid approach boosts performance by an additional 40%, yielding a total speedup of over four times. Similar performance gains can also be achieved for a wide range of other arithmetic algorithms, as is illustrated in *Table 4-2* (data are 16-bits). Performance gains are in comparison to an optimized SISD implementation.

Table 4-2 VIS Instruction Set Based Performance Gains for a Variety of Arithmetic Algorithms

Algorithm	Performance Gain, %
Fletcher checksum, 8-bit	600
256-point complex FFT	136
RAID computations	274
5x5 convolution, 8-bit	243
Vector dot product	200
Matrix addition (32x32)	400

4.3. Database Systems

The VIS instruction set also provides instructions to perform compares on multiple data elements in parallel (as discussed in Section 2.2., *Comparison Instructions* on page 12). In addition to accelerating graphics operations, such as the edge detection algorithm, image enhancement operation and 3-D rendering, these instructions have the potential to streamline a wide variety of search algorithms, including sequential searches, hash table lookups, message parsing and multi-way tree searches. The optimization of these algorithms, which represent core operations in a wide variety of applications – including database systems – is not as easily achieved with the other SIMD instruction set extensions because of the way in which the results of these comparisons are returned: the VIS instruction set returns the result of the SIMD compare in the 1-bit-1-result format (illustrated in *Figure 2-6* and *Figure 2-7*) and places this result into the integer registers, significantly simplifying the process of utilizing the information to direct the next iteration of the search.

The simplest search technique is the sequential search and it is readily apparent how the VIS instruction set can be utilized to speedup this application: by comparing the key with multiple data elements in parallel, the number of iterations required to locate the desired element is significantly reduced. This process is illustrated in *Figure 4-2* (coded in C using Sun's inline templates) and a basic VIS instruction set implementation of this nature can provide a speedup in excess of 300%, with respect to an optimized SISD implementation.

Figure 4-2 A Basic VIS Instruction Set Implementation of the Sequential Search Algorithm

```
int offset[16] ={0,3,2,2,1,1,1,1,0,0,0,0,0,0,0,0};

int
searchArray(unsigned short *key, unsigned short *dataArray, int size)
{
  int     i, j, sr0;
  vis_d64 *dataArray64, key64;

  dataArray64 = (vis_d64 *) dataArray;

  key64 = vis_bshuffle(vis_ld_u16(key), vis_fzero());

  for (i = 0; i < size; i+=4)
      if ((sr0 = vis_fcmpeq16(key64, dataArray64[i]) != 0) break;

  return(i + offset[sr0]);
}
```

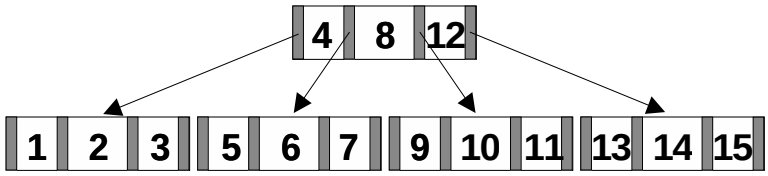
The implementation presented in *Figure 4-2* operates on variables which are of equal size to the VIS instruction set partitioned data size, i.e., 16-bits. However, the code

can be readily modified to handle arbitrary length keys: by comparing the search key and the data elements in an iterative fashion, there is now no restriction on the length of search key, i.e., process the first 16-bits, then the second 16-bits and so on.

But why does processing four 16-bit sub-elements per operation facilitate a faster implementation than processing one entire 64-bit key? In the worst case, it will not. That is, when all of the data elements differ from the search key only in the last 16-bits, i.e., the upper 48-bits are identical, the performance of the SISD and the VIS instruction set implementations will be similar. However, this is unlikely to be true all of the time, or even, in many cases, for much of the time. Under these conditions, it is not necessary to process all four sub-components of each data element. Rather, it is only necessary to compare sub-elements until no more matches are detected. In many cases, this will be after the first compare, facilitating, for arbitrary length keys, a four times reduction in the number of compare operations required and a corresponding increase in performance.

This technique can be extended to facilitate accelerated data retrieval from search trees. While the single comparison that is necessitated per iteration in the binary tree search tends to hamper the utilization of SIMD instructions, the binary tree is by no means the only tree search method. Rather, many multi-way tree types exist and can be readily accelerated using SIMD instructions. For example, nodes in an M-order B-tree have up to M children and contain between M / 2 and M – 1 separate keys, as is illustrated in *Figure 4-3* (in this tree each node can contain up to 3 keys and pointers to up to 4 children), providing ample opportunity for parallelism.

Figure 4-3 A Fourth Order B-tree



From *Figure 2-6* and *Figure 2-7*, it is apparent that the VIS instruction set excels at implementing third and fifth order B-trees (and multiples thereof) and is capable of providing a significant performance increase over that achievable with a standard binary tree. Significant performance gains can also be achieved for a wide range of other search techniques, as is illustrated in *Table 4-3* (data are 16-bits, performance gains are in comparison to an optimized SISD implementation).

Table 4-3 VIS Instruction Set Based Performance Gains for a Variety of Search Algorithms

Algorithm	Performance Gain, %
Sequential search	300
B-tree search	200
Message parsing	600

4.4. Codecs

In the previous paragraphs, the ability of the VIS instruction set to accelerate the core components of important applications has been highlighted. These optimizations in turn provide a significant performance boost to the application overall. Several notable examples are presented in *Table 4-4* (performance gains are in comparison to an optimized SISD implementation), with the VIS instruction set able to double the performance of many applications.

Table 4-4 VIS 1.0 Based Acceleration for Several Important Applications

Application	Performance Gain, %
MPEG-2 encoder	72
MPEG-2 decoder	100
H.263 encoder	111
H.263 decoder	100
H.261 encoder	66
H.261 decoder	79
JPEG encoder	139
JPEG decoder	126
G.711	47

CHAPTER 5

Architectural Features

Since the introduction of the UltraSPARC I processor in 1995, Sun has released two further generations of the UltraSPARC processor. The VIS instruction set, which debuted with the UltraSPARC I processor, has been a prominent feature in each subsequent generation, with 2001 seeing the introduction of the first revision of the VIS instruction set included in Sun's third generation UltraSPARC processor. Spanning multiple generations of processors, the VIS instruction set is influenced by microachitectural changes from one generation to the next.

5.1. VIS Instruction Set Latencies

In addition to the increased functionality discussed in Section 2.8., *Data Manipulation Instructions* on page 17, the latencies of the VIS instruction set instructions on the UltraSPARC III processor have changed from that seen in the previous generations of UltraSPARC processors, as is illustrated in *Table 5-1*. These increases are similar to those observed when moving from Intel's Pentium III *[7]* to Intel's Pentium 4 *[8]*, but with the wealth of registers available in the UltraSPARC processor, coupled with Sun's aggressive optimizing compiler and the high level of instruction level parallelism normally achievable, it is significantly easier to continue to keep the pipelines fully utilized with these longer latency instructions.

Table 5-1 The VIS Instruction Set Latencies

VIS Instruction Set Type	UltraSPARC I/II Latency (cycles)	UltraSPARC III Latency (cycles)
Addition/subtraction	1	3
Multiplication	3	4
Comparison	3	4
Format conversion	1 & 3	3 & 4
`pdist` - motion estimate	1	1

5.2. Prefetch Instructions

While prefetch instructions have been available in previous generations of the UltraSPARC processor (facilitating data prefetching to the level two cache), with the UltraSPARC III processor prefetching becomes even more beneficial. With this processor, there is now the option to prefetch data to a special prefetch cache which has similar retrieval times to the level one data cache.

Additionally, by utilizing prefetch it is now also possible to achieve two loads per cycle – one from the prefetch cache and one from the standard data cache – which can have large benefits for many types of application. This prefetching can be achieved in several different ways:

1. By stipulating the *-xprefetch* flag during compilation, the Sun compiler can automatically determine where it would be beneficial to place these prefetch instructions.
2. Explicitly indicate what should be prefetched by inserting the appropriate pragmas.
3. The UltraSPARC III processor also provides an automatic hardware initiated prefetch, such that, as long as the data access patterns are predictable, the required data should automatically be fetched into the prefetch cache.

Using these techniques, it is possible to ensure that VIS code performs at maximum efficiency on the UltraSPARC III processor.

CHAPTER 6

Competitive Analysis

While the VIS instruction set was the first comprehensive SIMD instruction set extension for a general purpose processor, it is now by no means the only one. Intel offers MMX and SSE, Motorola offers AltiVec and HP offers MAX, to name but a few. Additionally, many DSPs now also provide SIMD functionality, e.g., Analog Device's TigerSHARC. Nevertheless, the VIS instruction set continues to represent a powerful SIMD solution, primarily because of the following:

- **Access to a large number of registers:** The VIS instruction set has been integrated into the floating point (FP) pipelines on the UltraSPARC processor. This allows the VIS instruction set to utilize the thirty-two 64-bit registers that comprise the FP register file, providing ample space for temporary variables and hence facilitating the aggressive optimization techniques that are frequently required to maximize performance. This compares favorably with the eight registers available on the Intel Pentium 4 for use with SSE *[8]*.
- **Multiple pipelines:** The UltraSPARC processor sports two FP pipelines, allowing the dispatch of up to two FP operations per cycle. The VIS instruction set functionality has been split between these two pipelines, facilitating the dispatch of two non-identical VIS instruction set instructions per cycle, subject to a number of grouping caveats. This compares favorably with the one pipeline on the Intel Pentium 4 that handles all FP, MMX and SSE instructions *[8]*.
- **High throughput:** For each of the FP pipelines, the throughput for the VIS instruction set instructions is one. That is, during each and every cycle, an independent VIS instruction set instruction can potentially be dispatched into each FP pipeline. This is not achievable on all architectures. For example, the Intel Pentium 4 processor SSE instructions frequently only have a throughput of two: two cycles must elapse before the pipeline can accept the same instruction again *[8]*.
- **Three operand instructions:** The SPARC Instruction Set Architecture (ISA) is centered on an instruction format that stipulates two source and one destination operand, removing the requirement of two operand ISAs (such as the Intel Pentium 4) to regularly duplicate one of the source operands in order to preserve its value. This significantly reduces the number of move instructions required, minimizing redundancy and frequently leading to a more efficient implementation, as is illustrated in *Figure 6-1*.
- **Mixed mode operation:** While the VIS instruction set utilizes the FPU, it is possible to use both the VIS instruction set and FP instructions in the same cycle, i.e., the VIS instruction set can be used without any performance penalty in mixed mode applications which perform both FP and integer computations: each FP register holds 64-bits of data. How that data is interpreted is left to the individual functional units, with the VIS instruction set units interpreting the data as a 64-bit packed integer and the FP units interpreting the data as a FP number. This is not true for MMX, which cannot be used in conjunction with FP operations and requires the dispatch of a long latency state change

instruction every time it is necessary to switch between FP and MMX computations *[8]*. While FP registers are stack based on Intel Pentium systems, for performance reasons, it was decided that MMX instructions should operate with a flat register access model. These two views of the FP register file are mutually incompatible, preventing MMX and FP instructions from operating in parallel and necessitating the state change instruction to move between these different views of the register file *[9]*.

These architectural advantages, which are summarized in *Table 6-1*, clearly illustrate that the VIS instruction set is a well balanced addition to the UltraSPARC processors, seamlessly integrated and provided with sufficient CPU resources to allow high performance VIS instruction set implementations to be developed.

Figure 6-1 Coding Efficiency Achieved With 3 Operand Instruction Formats for a Butterfly Operation

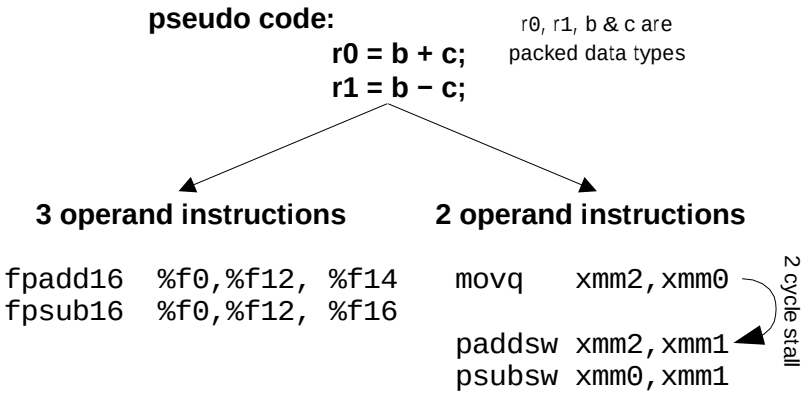


Table 6-1 An Overview of the VIS Instruction Set Architectural Advantages

Architectural Features	VIS and the UltraSPARC	SSE and the Intel Pentium 4 *[8]*
Number of Registers	32	8
SIMD instructions/cycle	2	1
Throughput[1]	1	2
Three operand ISA	Yes	No
Mixed mode operation (FP and Integer)	Yes	SSE – yes MMX – no
Floating Point SIMD	Planned for a later version	Yes

1. Smaller numbers are preferable.

CHAPTER 7

Conclusions

The previous sections of this paper provide a brief overview of Sun's VIS instruction set, beginning with the rationale for such an enhancement and concluding with a comparison of the VIS instruction set and SSE. Discussion focused on the functionality provided by the VIS instruction set and the various methods by which customers can integrate the VIS instruction set into their applications. The most notable of these is Sun's free mediaLib software, an extensive collection of the VIS instruction set optimized algorithms that can be called from customer code, representing a rapid and painless way to accelerate applications using the VIS instruction set.

The wide range of applications which can benefit from this SIMD instruction set extension, that is provided on all generations of the UltraSPARC processor, is then examined and it is illustrated that the VIS instruction set represents an extremely powerful tool for accelerating a wide variety of customer applications.

Finally, a comparison of the VIS instruction set and the significantly more heavy-weight SSE was provided to illustrate that, while manufacturers may expound the extensive nature of their SIMD instructions, this is not the whole story. In many situations, the VIS instruction set performs better because of the overall benefits of the UltraSPARC processor architecture and because the intelligent way that the VIS instruction set has been integrated into the processor, minimizes register pressure, maximizes the potential for instruction level parallelism and allows the UltraSPARC processor to perform up to ten operations per cycle.

References

[1] Sun Microsystems. (1996). VIS User's Manual.

[2] Sun Microsystems. (1999). UltraSPARC IIi User's Manual.

[3] Sun Microsystems. (current, December 2001). VIS Software Developers Kit. Retrieved from the World Wide Web: http://www.sun.com/sparc/vis

[4] Sun Microsystems. (current, December 2001). mediaLib. Retrieved from the World Wide Web: http://www.sun.com/sparc/vis/mediaLib.html

[5] Sun Microsystems. (2001). mediaLib User's Guide.

[6] Bhaskaran, V. and Konstantinides, K. (1997). Image and Video Compression Standards. Kluwer Academic Publishers.

[7]Intel Corporation. (1999). Intel Architecture Optimization Reference Manual.

[8] Intel Corporation. (1999). Intel Pentium 4 and Intel Xeon Processor Optimization Reference Manual.

[9] Intel Corporation. (1997). *MMX Technology Architecture Overview.* Intel Technology Journal.