Accelerating Core Networking Functions Using the The UltraSPARC™ VIS Instruction Set

As networking technologies move from the Megabits/second to the Gigabits/second realm, designers of bridges, routers, switching hubs and telecommunications equipment are building increasingly sophisticated embedded systems. Sun Microelectronics is addressing the performance shortcoming of existing embedded processors by improving computational efficiency and data transmission performance of the UltraSPARC processor with the Visual Instruction Set (VIS) extensions.

On average, a 1.9X speed-up in I/O performance can be attained by employing VIS instructions in the bcopy() routine to accelerate data movement by the processor. On average, a 4.8X speed-up in the execution speed of system-level routines such as the TCP/IP checksum can be attained using the VIS parallel math instructions. VIS coding is recommended for commonly-used networking routines to boost performance.

1. The UltraSPARC and VIS Instruction Set

As network systems continue to demand faster real-time response and data throughput, designers need faster processor-memory subsystems. VIS is a set of RISC-like extensions to the SPARC V9 instruction set that are executed on UltraSPARC processors to boost performance. With the VIS
Instruction Set, UltraSPARC processors can move data and execute the computations required by networking routines much faster.

Designers of network systems can speed general computation and the processor-to-memory bus bandwidth cycles by coding time-critical memory operations and kernel functions with VIS instructions. The VIS Instruction Set has two categories of instructions that can streamline network functions:

· Parallel Arithmetic: VIS includes single instruction, multiple data (SIMD) mathematical operations (e.g. partitioned or parallel add and multiply operations). Thus, VIS instructions can be used to accelerate data-intensive I/O functions and compute-intensive networking code.

The UltraSPARC processor is based on a four-way superscaler architecture with four pipelines that can execute instructions in parallel. There are two pipelines in the Integer Unit and two pipelines in the special Floating Graphics Unit (FGU) that handle the execution of VIS instructions. With a VIS parallel add instruction, four 16-bit operands are loaded into one 64-bit register, and four 16-bit operands are loaded into another register. Then, four 16-bit add operations are executed in a single cycle. Eight adds can be executed per cycle, as well as two integer operations for a total of ten operations every clock cycle.

· Moving Data: VIS also has instructions which load and store data in 64-byte blocks to accelerate network performance. The use of the VIS block load and store primitives to optimize memory subsystem throughput accelerates I/O operations that occur within the TCP/IP protocol stack. This speed-up in I/O results in a boost in performance for networking operations.

2. VIS In Network Applications

Sun Microsystems and SunSoft are focusing on applications of VIS to accelerate core networking functions. To demonstrate the performance gains possible by using VIS, two common networking functions were optimized (Figure 1). Benchmarks were run to measure the performance of these functions with and without the VIS instructions.
Figure 1. UltraSPARC VIS instructions can be used to accelerate commonly used networking functions such as moving data and performing system-level TCP/IP checksums.

- Fast Data Movement: VIS was used to accelerate the process of moving blocks of data between memory and the processor with the block load and store instructions. Moving data between memory buffers was accelerated an average of 1.9X. In a multi-threaded, connection-oriented client/server benchmark, the VIS-assisted kernel bcopy() routine yielded an average 23.8% increase in total TCP/IP throughput for transfers in the 1-8 Kbyte range.

- Fast Checksums: When data is transmitted between nodes, checksum routines are used to verify correct transmission. The speed of execution of Fletcher's checksum routine (defined in RFC 1145) was compared to the execution speed when coded with calls to VIS functions that take advantage of instruction-level parallelism. The benchmark showed a 4.9X speed-up in computation speed when using VIS-optimized C code.

2.1. Data Copy Operations

The routines that are used to move data segments between the various layers of a network protocol stack are time-critical functions that impact networking performance. In particular, the processor-to-memory bandwidth has a significant impact on networking throughput.

When a typical network application reads or writes data between nodes, a read/write system call initiates the movement of data from a user address space, through a network I/O driver that transfers data from protected memory space in the kernel to data buffers or a networks card.

In the Solaris 2.5 operating environment, copying data is handled
by the low level kernel function bcopy(), which is used for kernel-to-kernel data movement. Within the layers of a network protocol stack, data can be moved as much as eight to nine times between memory locations before it reaches its final destination. Therefore, accelerating the speed of the often-used bcopy() routine improves throughput.

The VIS block load and block store instructions can be used to keep data copying overhead to a minimum. With these instructions, multiple words can be transferred using a single instruction, bypassing the cache memory. This greatly increases the aggregate data throughput compared with copying the same regions of memory using a series of traditional RISC load/store instructions.

2.1.1 Data Copy Using Load/Store

Without VIS, the UltraSPARC processor is a traditional RISC load/store machine. When bcopy() transfers data from one buffer to another, data must first be moved from memory, through the cache, into the processor. Then it is moved back through cache to the target memory location.

This copying process using bcopy() involves a series of single-cycle load and store instructions, as follows:

1. Fetch a cache line of data from memory into the cache.

2. Once the cache line has been fetched, move the data into the register file (four bytes at a time) using loads.

3. Move the data from the register file (four bytes at a time) back out to the cache at a different cache line, using stores.

4. Move the modified cache line out to the new memory location.

Moving 64 bytes of data from main memory, into the processor, and back out to a new location in memory takes at least 32 instructions (16 loads and 16 stores). A cache miss kicks off the process of bringing new data into the cache from memory. The typical cache line size is 32 bytes. Load instructions transfer data from the cache line into the register file (Figure 2). Then store instructions write the data out to a new cache line. Finally the stored cache line is written out to memory.
2.1.2 Data Copy With VIS Block Load/Store Instructions

To demonstrate the potential for network performance improvement, the bcopy() routine in the Solaris 2.5 kernel was rewritten using VISp instructions. UltraSPARCp users with Solaris 2.5 receive the benefit of VIS without having to recompile the kernel. The kernel bcopy() function is accelerated, while the user-callable bcopy() function is not. The user callable memcopy() function is VIS accelerated.

With the bcopy() kernel function implemented using VIS block load and store instructions, 64 bytes of data are loaded from memory directly into the on-chip floating point register file (Figure 3). A second instruction moves the data from the floating point register file to its new location in main memory, bypassing the cache. Moving the same 64 bytes takes two instructions versus 32 instructions without VIS.

Figure 3. A VIS block load instruction brings a 64 byte block on chip. With a block store instruction, the whole block is written back out to memory. Comparing overall latency, there are two memory accesses in the VIS bcopy() case versus up to eight memory accesses when using conventional load/store instructions.
2.1.2 Load/Store Bcopy( ) Cache Impact

With VIS, cache memories are left undisturbed while 64 byte loads and stores are performed directly to main memory. The operating system can directly write a page in memory rather than overwriting the cache to swap pages in and out. Bypassing the cache eliminates the detrimental side effect of cache thrashing, in which excess processor effort is expended bringing data in and out of memory. With an undisturbed cache, other pieces of code can execute without incurring data cache miss latencies, which has a positive impact on overall system performance.

2.1.3 VIS Bcopy( ) Benchmark

The speed of copying data within user-level application space was measured for the bcopy( ) function with and without the assistance of the VIS block load/store instructions on a 167 MHz UltraSPARC machine. Each completion time shown is an average of 1000 data transfers. For this I/O benchmark, data copying runs, on average were 1.9X faster.

<table>
<thead>
<tr>
<th>Data Size</th>
<th>Completion Time With VIS (msec)</th>
<th>Completion Time Without VIS (msec)</th>
<th>Speed-Up Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Kbyte</td>
<td>0.004126</td>
<td>0.005372</td>
<td>1.30X</td>
</tr>
<tr>
<td>10 Kbyte</td>
<td>0.031477</td>
<td>0.0479199</td>
<td>1.52X</td>
</tr>
<tr>
<td>100 Kbyte</td>
<td>0.318588</td>
<td>0.700303</td>
<td>2.20X</td>
</tr>
<tr>
<td>1 Mbyte</td>
<td>5.342989</td>
<td>11.816248</td>
<td>2.21X</td>
</tr>
<tr>
<td>10 Mbyte</td>
<td>57.571815</td>
<td>123.650246</td>
<td>2.15X</td>
</tr>
</tbody>
</table>

When using the VIS instructions, the Solaris protocol stack is capable of sustaining 600 Mbit/s block data transfers across the processor/memory bus. The VIS block load and store instructions support the data throughput necessary for emerging gigabit networking technologies such as ATM and Gigabit Ethernet.

2.1.4 VIS Assisted Client/Server Benchmark

A multi-threaded connection-oriented client/server throughput benchmark also measured the effect of VIS on kernel bcopy( ) in data transfers between 100 Mbit/s Ethernet (100BaseT) nodes. Two UltraSPARC workstations were connected in peer-to-peer mode, and data of different blocks sizes was transferred between
the machines. For transfers in the 1-8 Kbyte range, the VIS bcopy() routine improved throughput an average of 23.8% for total TCP/IP throughput. The data for this client/server throughput benchmark shows the following improvement:

<table>
<thead>
<tr>
<th>Transfer Size</th>
<th>Average Throughput Without VIS</th>
<th>Average Throughput With VIS</th>
<th>TCP/IP Speed-up With VIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 KByte</td>
<td>2700.31 Kbps (2.70 Mbps)</td>
<td>3372.04 Kbps (3.373 Mbps)</td>
<td>24.87%</td>
</tr>
<tr>
<td>2 KByte</td>
<td>5478.69 Kbps (5.47 Mbps)</td>
<td>6496.30 Kbps (6.496 Mbps)</td>
<td>18.57%</td>
</tr>
<tr>
<td>4 KByte</td>
<td>10468.20 Kbps (10.46 Mbps)</td>
<td>13096.90 Kbps (13.09 Mbps)</td>
<td>25.11%</td>
</tr>
<tr>
<td>8 KByte</td>
<td>19621.20 Kbps (19.62 Mbps)</td>
<td>24900.10 Kbps (24.90 Mbps)</td>
<td>26.90%</td>
</tr>
</tbody>
</table>

2.3 Fletcher’s Checksum

Fletcher’s Checksum is a commonly used error detection algorithm that is used extensively in communications error detection due to its easy implementation and rapid computation. Fletcher’s Checksum is used to validate the correct transmission of address headers and messages in TCP/IP stacks and on the Internet and is documented in the Internet Engineering Task Force’s RFC 1146.

When data is transferred over TCP/IP, the messages are first broken into smaller pieces which are labeled so their receipt can be verified on the receiving end. During the process, TCP also calculates a checksum for each smaller piece and sends it along with the data, giving the receiving computer the means of verifying the authenticity of the transmitted data. The receiving end calculates the checksum for each piece received and compares it to the checksum that was sent along. If they are not identical, the receiving end uses the TCP protocol to request retransmission of the data.

The VIS partitioned add and multiply instructions can be used to speed up data checksumming routines used by the TCP/IP protocol. System-level checksums are also used to verify data integrity.

2.3.1 Checksum Without the VIS Instruction Set

One of the most common implementations of the checksum
algorithm processes 8-bit data elements. When the transmitted data is a sequence of \( N \) octets (\( d(0) \) to \( d(N-1) \)), the checksum value is contained in two octets and its generation is governed by the following algorithms. The lower 8 bits of the two computation results are stored as the check octets:

\[
C_0 = d(0) + d(1) + d(2) + \ldots + d(N-1)
\]

\[
C_1 = (N) \cdot d(0) + (N-1) \cdot d(1) + \ldots + (1) \cdot d(N-1)
\]

### 2.3.2 Checksum With the VIS Instruction Set

The VIS implementation of the checksum routine utilizes the FPADD16 and FMUL8x16 instructions, which provide parallelism for single cycle execution of the series of additions and multiplications required for the computation for \( C_0 \) and \( C_1 \). Both instructions facilitate the parallel handling of four data elements. Using VIS reduces the number of separate arithmetic operations required for calculation of checksum by 75%. This reduction enabled a speed-up approaching 4.8X for all data packet sizes.

### 2.3.3 VISp Checksum Benchmark Summary

The benchmark measured the performance of a 2’s complement Fletcher’s checksum routine executed as a C routine compared with the execution time of the C routine optimized using VIS for a number of different data packet lengths. On average, a 4.8X speed-up was attained using VIS parallel math instructions. The benchmark numbers are as follows:

<table>
<thead>
<tr>
<th>Data Packet Size (Kbytes)</th>
<th>VIS Routine Time (msec)</th>
<th>C Routine Time (msec)</th>
<th>Speed up With VIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.0041</td>
<td>0.0181</td>
<td>4.34</td>
</tr>
<tr>
<td>1.0</td>
<td>0.0078</td>
<td>0.0361</td>
<td>4.63</td>
</tr>
<tr>
<td>2.0</td>
<td>0.0149</td>
<td>0.0721</td>
<td>4.86</td>
</tr>
<tr>
<td>4.0</td>
<td>0.0295</td>
<td>0.1442</td>
<td>4.92</td>
</tr>
<tr>
<td>8.0</td>
<td>0.0586</td>
<td>0.2899</td>
<td>4.95</td>
</tr>
</tbody>
</table>

### 3. Summary

To demonstrate the benefits of VIS in network applications Sun Microsystems targeted two key networking operations for optimization with VIS. By re-coding the data copying portion of the Solaris kernel, a 1.9X speed-up in bcopy( ) performance was
attained. VIS is beneficial in optimizing networking operations that requires the processor to move large amounts of data to and from memory such as network protocols and I/O drivers.

Embedded system designers working with real-time operating systems that run on the UltraSPARC can take advantage of similar performance gains by rewriting key kernel functions and networking routines with VIS functions. By optimizing an often-used system-level checksum routine into a VIS implementation, the execution speed increased 4.8X. The VIS partitioned add, subtract, and multiply instructions can add parallelism to reduce a large portion of CPU consumption in core networking routines.

The performance advantages obtained by the use of VIS are not limited to the examples detailed in this paper. VIS has additional instructions not addressed in this paper that can potentially improve the real-time performance of embedded systems such as bridges, routers and switches. VIS has instructions that accelerate the process of reading data that is not 64-bit aligned (vis_falign-address instruction provides an offset that indicates where the data resides; the vis_faligndata instruction uses that offset to extract the data from that location). These instructions can decrease the number of load/store operations needed to access data from memory to accelerate lookup table operations in networking systems.

4. References
