

Contact

Publication of the Cleveland Section of the Institute of Electrical and Electronics Engineers.

Cleveland Engineering Society Events

The following upcoming events for CES are open to all. Click on the Website listed for additional information. For reservations, call the Cleveland Engineering Society, (216) 361-3100.

February:

- 6 Leadership Breakfast Series, featuring Thomas Waltermire of PolyOne Corp,

http://www.cesnet.org/weblinker/ces_upc/2001010401.html

- 15 Environmental Division program, "The TSCA Mega-Rule" ...

http://www.cesnet.org/weblinker/ces_upc/2001012900.html

Engineer of the Year Dr. Raymond Muzic, Jr.

By Jerry Lucak

Dr. Raymond Muzic, Jr. has received the Cleveland Section Engineer of the Year award for 2001. Dr. Muzic is Assistant Professor of Radiology and Biomedical Engineering at Case Western Reserve University. He obtained his BS degree in Biomedical Engineering from CWRU in 1987, and his MS degree in 1988. Following a dissertation on scattered radiation in Positron Emission Tomography, CWRU awarded him a PhD in Biomedical Engineering in 1991. After several years of research work, Dr. Muzic was appointed Assistant Director of PET Processing at CWRU & University Hospitals in 1996.

Dr. Muzic has published numerous papers on PET imaging and presented his research results at various technical conferences. He taught biomedical courses at CWRU for a number of years and is an adviser to doctoral candidates in the biomedical engineering program.

Dr. Muzic is a member of three IEEE technical societies and is a reviewer for both the IEEE Transactions on Nuclear Medicine and the IEEE Transactions on Medical Imaging. He recently organized a Medical Imaging chapter within the IEEE Cleveland Section.

Ray was born & raised in the Cleveland area and currently lives in Mentor with his wife Linda and two daughters.

Congratulations to Ray on his selection as IEEE Cleveland Section Engineer of the Year.

CLOSE ONLY COUNTS IN HORSESHOES

by Michael Garvey

The exclusive right protected by a patent is defined by the patent claim. A device only infringes a patent if it includes every element of the claim.

Let's say the patent claim recites "a first flange, a second flange, and a nut and bolt holding the flanges together". A device using glue to hold the flanges together would not infringe the claim.

Since nearly the beginning of the U.S. patent system, courts have recognized that an injustice is done when someone makes minor changes to avoid a patent. Thus, courts have sometimes enforced patents beyond the literal scope of the claims.

In 1950, the Supreme Court named this principle the Doctrine of Equivalents. The Court held that one will not be permitted to "commit a fraud on a patent" by making "insubstantial changes". Courts have varied in their application of this doctrine. During the past decade or so, the Court of Appeals for the Federal Circuit, which hears appeals in patent cases, has restricted the Doctrine of Equivalents to the point where some were proclaiming its imminent demise in the mid 1990s.

In 1997, however, the Supreme Court pronounced the Doctrine of Equivalents to be alive. In doing so, the Court further refined the application of the Doctrine, stating that the equivalence analysis is done on an element by element basis. That is, each element of the claim or its equivalent must be present in the accused device.

In November of 2000, the Federal Circuit again weighed in on the Doctrine of Equivalents. The court held that the Doctrine of Equivalents cannot be applied to any element that was amended during prosecution of the patent application. In other words, if a claim element was changed during the application process, the accused device must include that element literally to infringe the claim.

Applying the Doctrine of Equivalents to the example, a court would determine whether glue is equivalent to a nut and bolt. What is an "equivalent" can be the subject of another column. If, however, the element "nut and bolt" was amended during the application process, there would be no equivalence analysis and there would be no infringement.

The lesson we learn from the recent decisions is that inventors and patent practitioners should not expect to rely on the Doctrine of Equivalents. Thus, a better claim limitation might have been "fastener", which would be literally infringed by the glue. This assumes this broader language would not make the claim so broad that it covers prior art and, therefore, would not be patentable.

The restricted application of the Doctrine of Equivalents places a great burden on the patent practitioner and the inventor to work together to ensure that future variations on the invention are, if possible, captured within the literal scope of the claims.

Michael Garvey is a patent attorney with Pearne & Gordon LLP in Cleveland.



CPU Performance Improvement 2

Vectorization: The Need

By Dr. Steve Belovich

Pipelining has been and continues to be the main technique for creating fast central processing units (CPUs). However, the peak performance of a pipelined machine is limited by two major factors:

- Clock Cycle Time - the clock cycle time can be decreased by increasing the number of pipe stages (making the pipeline “deeper”) but this will increase the number of hazards, increase the number of clock cycles per instruction (CPI) and will increase the amount of hardware required for implementation.

- Instruction Fetch and Decode Rate - This limitation prevents fetching and executing more than a few instructions per clock cycle (called the “Flynn Bottleneck”). For most pipelined machines, the average number of instructions issued per clock cycle is less than one.

Cache memory can be used to decrease the number of memory reference operations in pipelined machines. However, most scientific and engineering applications operate on array variables. Due to aliasing, the compiler must assign these variables to the heap rather than to registers. Main memory holds all heap variables which forces a memory reference operation for all array variables. Since array element access via a program is highly unpredictable, the locality of reference for these variables can be very low. Hence, the performance of cache memory for array variables may also be very low. The bottom line is that for most scientific and engineering problems of practical importance, cache memory may not be of much help.

The Technique

Given these limitations, the preferred way to increase CPU performance is through vectorization. A “vector machine” provides operations that work on linear arrays of numbers which are called “vectors”. One operation may subtract two 128-element integer arrays in a single step. This vector subtract operation is the same as an entire loop in a non-vector CPU. A non-vector CPU would have to subtract two corresponding elements of the 128-element arrays, save the result, increment the array pointers and branch back to do the subtraction again.

Unlike pipelining, where no program changes are required to realize a performance advantage, vectorization expands the instruction set by adding vector instructions. The burden is on the programmer or the compiler to use these vector instructions to its best advantage. This is not always easy to do.

The Advantages

Vector CPUs can have very deep pipelines without causing data hazards since the computation of each result is independent of the computation of previous results. The compiler or the programmer determines when it is safe to use vector instruction, i.e., when no data hazards are present.

Vector instructions essentially replace entire loops in non-vector CPUs. Because the vector instruction’s behavior is predetermined, any control hazards which may

be caused by the loop branch in non-vector CPUs are eliminated.

Each vector instruction performs many operations even though only one instruction is fetched and executed. Fewer instructions are needed to implement a given algorithm, assuming that the algorithm lends itself to vector processing at all. In such cases, the deleterious effects of the “Flynn bottleneck” are greatly reduced.

Vector instructions have deterministic memory access patterns. If the vector operation’s memory-resident operands are logically adjacent in address space, techniques such as memory interleaving are very effective in reducing the memory latency time. Interleaved memory allocates adjacent operands to different physical memory banks which overlaps the time required for a refresh/rewrite of operand n with the access of operand $n+1$. The per-operand memory latency time for the entire vector operation can be much less than latency time for a single operand.

The Architecture

Vector operations are pipelined on the individual elements of the vector. The vector pipeline includes the arithmetic hardware as well as the memory access and effective address calculation hardware. Also, most supercomputers allow more than one simultaneous vector operation which creates parallelism among the operations on different elements.

Vector CPU designs come in two main flavors: memory-memory vector machines and vector-register machines. A memory-memory vector machine lets all vector operations operate on memory operands. The first vector machines were of this type as were the CDC Cyber series of supercomputers.

A vector-register machine forces all vector operations (except load and store) to operate only on vector registers. This architecture is the vector counterpart to the load/store architecture (e.g., the Alpha 21064 family of microprocessors). The Cray-1, Cray-2, Cray X-MP and Cray Y-MP are all vector-register machines. The Convex C-1 and C-2 supercomputers are also vector register machines.

Figure 1 shows a simple vector-register architecture. each vector register has two read ports and one write port. This supports a high degree of overlap among vector operations to different registers

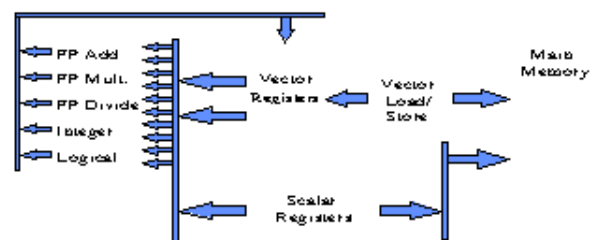


Figure 1: A Simple Vector CPU

The vector functional units are each fully pipelined and can start a new operation every clock cycle. Hazard detection is necessary for structural hazards (contention for a functional



unit) and for data hazards (contention for vector registers). The vector load/store unit loads and stores vectors from memory. This functional unit is also fully pipelined so that operands can be moved between vector registers and memory at the rate of one operand per clock cycle.

The simple vector CPU of figure 1 also has scalar registers. These registers hold additional operands for the functional units. These operands allow hardware support for common high-level language operations such as $y = ax + b$ where x and y are vectors and a and b are scalars.

Potential Problems

Vector CPUs work well if all vectors are small enough to reside comfortably within the vector registers. In the case of figure 1, all vectors would have to have 64 or fewer elements. Unfortunately, the length of a vector may not be known until run-time, especially for loops whose index limits are variables.

“Strip mining” is a technique whereby the compiler generates code in such a manner so as to guarantee that each vector operation requires a vector of length equal to or less than the maximum vector length supported by the machine. This generally involves chopping up loops into sequences of loops, each of which can be implemented via one vector instruction. Doing this adds vector startup overhead and decreases the efficiency from its optimum value.

Another difficulty with vectorization is that operands may not be in adjacent memory locations. A matrix multiply, for example, will require sequential access to operands that are located a fixed distance from each other in memory. Column entries in a matrix will not be adjacent in C-language programs because C uses “row-major” ordering for arrays. FORTRAN uses “column-major” ordering and so row entries will not be adjacent in memory in FORTRAN programs.

This problem is solved by letting the vector load/store unit access successive locations in memory that are separated by a known, fixed amount, called the “stride”. These operands are loaded into a vector register where they are then treated as if they were logically adjacent for the purposes of some vector computation. The stride may not be known until run time and a special register may have to be provided to hold the stride value. Note that the compiler has to compute the stride, or at least the pointer to the location that will hold the value after some computation.

A non-unity stride may cause memory contention problems, if memory requests are made to the same memory bank faster than the latency time. This condition will force a pipeline stall which may negate some or even all of the performance improvement that vectorization provides. Great care must be taken in the code generation section of the compiler to prevent this from occurring.

Performance Issues

Startup overhead negates some of the performance advantages of vectorization. If vector start-up times are long and vectors are short, little is gained in terms of execution time reduction. Many Japanese supercomputers have higher “sustained execution rates” than comparable models within the Cray family of supercomputers but their startup times are between 50% and 100% longer. Consequently, the actual

performance of the Japanese supercomputers is very often less than that of the comparable Cray model. Table 1 below gives an example of the phenomenon.

Algorithm	Cray X-MP	Hitachi S810/20
$a(i) = b(i)*c(i) + d(i)*e(i)$ with vector length of 1000 done 100,000 times	2.6 seconds	1.3 seconds
Vectorized FFT with vector lengths of 64, 32, 16, 8, 4 and 2	3.9 seconds	7.7 seconds

Table 1: Execution Times for Two Algorithms on Two Supercomputers

Increasing vector performance without increasing scalar performance is generally a waste of effort. Good scalar performance reduces “strip mining” and other overhead costs. Table 2 below shows performance measurements for the Livermore FORTRAN kernels on two different machines. The Stardent-1500 has a maximum computation rate of 10.08 MFLOPs for a highly vectorized loop but its overall performance is worse than a fast scalar machine because of its poor scalar performance.

Machine	Minimum Performance for any loop	Maximum Performance for any loop	Harmonic mean for all 24 Livermore loops
Stardent-1500	0.41 MFLOPs	10.08 MFLOPs	1.72 MFLOPs
MIPS M/120-5	0.80 MFLOPs	3.89 MFLOPs	1.85 MFLOPs

Table 2: Performance Measurements for Livermore Loops

Software Implications

Vectorization techniques are only effective if some or all of the tasks to be run are vectorizable. If no data dependencies exist within loops or if they can be removed, vectorization will improve performance. This is a strong function of the algorithms selected and how they are coded.

The compiler must also be capable of detecting when a loop can be vectorized. Although no compiler can vectorize a loop where there is no parallelism, compilers vary widely in their ability to detect vectorizable loops. Most real programs demand careful coding on the part of the programmer to allow the compiler to vectorize the loops. The compilers which have received most attention and vectorization effort have been FORTRAN (F77, F90 and sundry variations) since most scientific and engineering applications are coded in these languages.

Real-World Fun-Facts

One of the most well-known examples of a vectored CPU was the Cray-1, introduced in 1976. The Cray-1 had 8 vector registers. Each vector register held sixty-four 64-bit operands, which was Cray-1’s word size. The Cray-1 had six vector functional units: an add unit, A multiply unit, a reciprocal unit, and integer add unit, a logical unit and a shift unit. The Cray-1 has a single vector load/store functional unit.

The Cray-2 was introduced in 1985. It has 8 vector registers, each of which holds sixty-four 64-bit operands. The Cray-2 has five functional units: a floating-point adder, a floating-point multiplier, a floating-point reciprocal and square-root unit and a logical unit. The fifth functional unit is an integer unit which implements add, shift and count operations.

The Convex C-1 was also introduced in 1985. It has 8 vector registers, each of which hold 128 64-bit operands. The Convex C-1 has four functional units: multiply, add, divide and integer/logical. It has one vector load/store unit.

◆ **JOB LISTING SERVICE**

◆ Send blank e-mail to:
◆ info.ieeeusa.jobs.r02@ieee.org

◆ For member address changes, please call
◆ the IEEE Hotline, (800) 678-IEEE.
◆ Do not call the newsletter editor. For
◆ advertising rates, contact Lee Bernasek.

◆ *MIREA: Engineering in
◆ Medicine & Biology/
◆ Instrumentation & Measure-
◆ ment/ Reliability/ Industrial
◆ Electronics/Aerospace &
◆ Electronic Systems

◆ **MLE: Microwave Theory &
◆ Techniques/Lasers &
◆ Optoelectronics/Electron
◆ Devices

◆ ***Medical Imaging Chapter:
◆ Signal Processing/Nuclear and
◆ Plasma Sciences/Computer/
◆ Engineering in Medicine
◆ Biology Society

◆ **Please note: if you are**
◆ **trying to contact the IEEE**
◆ **Cleveland Section by FAX,**
◆ **dial 440-473-6557.**

◆ **The line is open twenty-**
◆ **four hours per day.**

IEEE Cleveland Section 2001 Officers

ELECTED SECTION OFFICERS

Section Chairman	Carl Dister	440-255-2977
Vice Chair	Ray Heintel	
Secretary	William Stempowski	440-967-2543
Treasurer	Ted Takacs	216-228-4408

APPOINTED SECTION OFFICERS

Area Rep/Ad Manager	Lee Bernasek	440-446-1985
Awards Chairman	Jerry Lucak	440-235-2550
Chair, Section Dev.	Bill Schultz	216-831-4466
Consultants Network	Dragan Dugandzic	440-257-2151
CTSC Representative	Ted Takacs	216-228-4408
Database Manager	James Sens	216-642-1230
Education Chairman	open	
Engineer Week Rep.	Dick Carlson	440-951-1595
Newsletter Editor	M. Greene	216-292-2686
Pace Rep.	Ben Malakooti	216-368-4462

TECHNICAL SOCIETY CHAPTER CHAIR

Chair	Ray Muzic	216-844-3543
Computer	Steve Belovich	216-267-1881
Control Systems	Mike Branicky	216-368-6430
Industrial Applic.	Carl Dister	440-255-2977
**Medical Imaging Chapter	Ray Muzic	216-844-3543
*MIREA	Vincent Lalli	216-433-2354
**MLE	Masud Tabib-Azar	216-368-6431
Power Engineering	Allen Morinec	330-255-1624
Systems/Men/Cybernetics	Ben Malakooti	216-368-4462
Vehicular Technology	open	

STUDENT BRANCH OFFICERS

CSU Advisor	George Kramerich	216-687-2405
CSU Chair	William Stempowski	440-967-2543
CWRU Advisor	Robert Edwards	216-368-2833
CWRU Chair	Mike Suster	mas20@po.cwru.edu

IEEE Internet

◆ Webmaster
◆ Website: <http://www.ewh.ieee.org/r2/cleveland/>
◆ News Group Mgr. Richard Bloss 216-464-0405
◆ FAX 216-464-0490
◆ e-mail: aa974@cleveland.freenet.edu



Cleveland Section

Editor:
M. Greene

Art Director:
Jeff Greene

Advertising Manager:
Lee Bernasek

IEEE Chairman:
Carl Dister

The "Contact"
is published
eight times per year:
September, October,
November, January,
February, March,
April and May.
The deadline for
electronic Newsletter
articles
is the 20th of the
previous month.