An Evaluation of PowerMac G4 Systems for FORTRAN-based Scientific Computing with Application to Computational Fluid Dynamics Simulation

Craig A. Hunter
NASA Langley Research Center
Configuration Aerodynamics Branch
Hampton, Virginia

– July 2000 –

Evaluations and comparisons within this paper are for technical government purposes only and do not constitute an endorsement of any system by NASA or the U.S. Government.

Abstract

This paper describes work conducted at NASA Langley Research Center during an evaluation of PowerMac G4 systems for FORTRAN-based scientific computing and computational fluid dynamics simulation. A PowerMac G4/500 was configured for dual booting into Mac OS and Linux operating systems. Various developer tools were used to compile and run test codes on the G4 for comparison benchmarking with platforms including Cray C-90, Compaq Alpha, Pentium III, and Silicon Graphics (SGI) systems. Following general benchmarking, more specific AltiVec testing was conducted on the G4 using FORTRAN and C, and approaches for implementing AltiVec in generic FORTRAN computations were developed.

Results indicate that the PowerMac G4 system has the potential to be an inexpensive high performance scientific computing platform. Much of that potential is currently unrealized, however, due to the limited amount of AltiVec support in FORTRAN. Without the parallel vector processing capabilities of AltiVec, the G4 places near the end of the pack in performance tests using standard FORTRAN scientific codes. In limited cases where AltiVec acceleration was available and tested under FORTRAN, the G4 showed a clear advantage with 4-7X greater performance and a 5-8X greater cost effectiveness than all other workstation systems evaluated. Examples presented in this report show that only minor re-coding would be necessary to implement AltiVec instructions if they were accessible to standard FORTRAN programming. Because of this, there appear to be many opportunities to advance scientific computing on the PowerMac G4 platform.

Introduction

In recent years, advances in computer technology have resulted in workstation-class personal computer (PC) systems that offer high performance at low cost, making these systems an attractive option for scientific computing. As a result, there is a growing movement in the scientific community to transition numerical computation and simulation from traditional large scale mainframes and supercomputers to distributed parallel computing on inexpensive PC "clusters". The Configuration Aerodynamics Branch (CAB) at NASA Langley Research Center has several of these so-called "Beowulf" clusters in operation for use in Computational Fluid Dynamics (CFD) simulation, and the systems provide supercomputer performance at a fraction of the cost. Instead of paying $50,000–60,000 for several thousand hours of time on a supercomputer system (the typical amount required to address a detailed aerodynamic design problem), a researcher can now purchase an entire PC cluster with equivalent computational power for about the same price or less. Unlike supercomputer hours that meter away with use, a PC cluster can be used over and over, with a projected service life of 3-5 years or more.

Most Beowulf clusters are built around Intel Pentium- or Compaq Alpha-based computers running a variant of the open source Linux operating system [Reference 1], but the Beowulf concept really applies to any computer system capable of exchanging information and communicating through a "message passing interface" (MPI) standard. In 1998, researchers at UCLA developed a distributed parallel computing system based on Apple PowerMac G3 computers by implementing MPI on the Mac OS. This new class of "AppleSeed" [Reference 2] cluster systems delivers impressive performance at low cost, while retaining all the benefits of traditional Mac OS computers such as high productivity and ease of use. In addition to the AppleSeed system, Terra Soft Solutions has recently developed and demonstrated PowerMac clusters running the company's Black Lab Linux operating system [Reference 3] with MPI. The Black Lab systems combine AppleSeed levels of performance with the added "universality" of the Linux operating system, simplifying ports of software from other Linux-based Beowulf systems.

In the fall of 1999, Apple introduced PowerMac G4 computer systems [Reference 4] using the Motorola MPC7400 processor with "AltiVec" – a high performance vector parallel processing expansion to the PowerPC RISC processor architecture [Reference 5]. In computations involving single precision floating point numbers, AltiVec can offer 4-way parallelism by simultaneously operating on 4 elements of a vector with every instruction. As part of AltiVec, 162 new vector operations have been added to the PowerPC instruction set, all of which are fully pipelined with single cycle throughput [Reference 5]. In June 2000, UCLA benchmarked an AltiVec accelerated 16 node AppleSeed system at performance levels approaching 23 GFLOPS [Reference 2], putting the G4 system near the top end of the Beowulf performance spectrum.

Because of the potential gains in computational performance and low cost offered by PowerMac G4 systems, the Langley CAB decided to evaluate a single-node G4 system for FORTRAN-based scientific computing and computational fluid dynamics simulation. Apple computer loaned CAB a PowerMac G4/500 system which was configured for dual booting into Mac OS 9 and Black Lab Linux. Black Lab Linux comes with GNU compiler tools including gcc and g77, and a special AltiVec enabled version of the gcc compiler (called gcc-vec) can be installed. Version 6.2 of the Absoft FORTRAN compiler [Reference 6] was also installed under Mac OS and Linux (the Mac OS compiler offers limited AltiVec support). Evaluation testing was conducted by running FORTRAN benchmark codes on the G4 and other computer platforms including Cray C-90, Compaq Alpha, Pentium III, and Silicon Graphics (SGI) systems. Following general benchmarking, more specific AltiVec testing was conducted on the G4 using FORTRAN and C, and approaches for implementing AltiVec in generic FORTRAN computations were developed.

The evaluation process and results are presented in this paper, which is organized by section as follows:

Performance Testing

Benchmarks were obtained by running single precision (32-bit) floating point FORTRAN 77 (F77) computations on various computer systems. Test programs were chosen to run through a variety of standard F77 computations and operations found in typical scientific and real world applications. While programs can be written to test specific capabilities on specific machines, the focus of this particular study was on taking legacy F77 programs and testing them on multiple platforms with a minimum of code porting or machine-specific tweaking. This approach is somewhat restrictive, but scientific code developers will only be interested in adopting a new computing platform if it represents an easy port of their existing (and already universally adopted) work. Thus, every attempt was made to stick with standard FORTRAN coding in each of the test programs, using external libraries, subroutines, and compiler options to adapt the codes to each machine as needed. In each case, efforts were made to maximize execution speed by using compiler optimization.

Results for the PowerMac G4 were obtained using the GNU g77 and Absoft v6.2 FORTRAN compilers running under Black Lab Linux, and the Absoft v6.2 FORTRAN compiler running under Mac OS 9. It is important to note that PowerMac G4 executables were compiled without AltiVec acceleration because it was not accessible to the standard F77 computations in the test codes. This is a major shortcoming in the current state of FORTRAN on the PowerMac G4, and it is hoped that future compiler development will address this issue. Implementation of AltiVec FORTRAN will be discussed in more detail later on.

The first results shown are from the single precision "Whetstone" benchmarking program, which runs 10 mathematical test loops described in Reference 7. To obtain consistent timing results, each test loop was run over 1000 cycles, and the entire sequence was repeated 100 times. Results from various computer platforms are summarized in Table 1 and Figure 1.

Computer / OS **Compiler**	**Compiler/Optimization Flags**	**Whetstones**
Cray C90 Single Node / Cray UNICOS Cray Fortran 90 Compiler	f90 -O3,aggress,allfastint,inline2	356773
Compaq Alpha 21264 500MHz / Red Hat Linux Compaq Fortran Compiler	fort -O -tune ev6 -arch ev6 -fast	1348433
Compaq Alpha 21164 533MHz / Red Hat Linux Compaq Fortran Compiler	fort -O -tune ev5 -arch ev5 -fast	1036227
DEC Alpha 564 333MHz / DEC UNIX V4.0B Digital Fortran Compiler	f77 -O4	641283
Pentium III 800MHz / Red Hat Linux Portland Group Compiler	pgf77 -O2 -Mbackslash -Minline=2 -Minfo=inline	625000
SGI Octane R-10K 195MHz / IRIX 6.4 MipsPro Fortran 77 Compiler	f77 -O3	588235
PowerMac G4 500MHz / Black Lab Linux Absoft v6.2 PPC Linux Compiler	f77 -O -h32 -H32	384615
PowerMac G4 500MHz / Black Lab Linux GNU g77 Compiler	g77 -O2	338983
PowerMac G4 500MHz / Mac OS 9 Absoft v6.2 Mac OS Compiler	f77 -O -h16 -H16	338602

Table 1: Summary of Whetstone Benchmark Results

Whetstone performance was dominated by the Compaq Alpha 21X64 machines, with benchmarks over 1 million “whetstones”. These “super-scalar” computers make extensive use of pre-fetching and pipelining during execution, which provide a significant speed boost. A second grouping of machines including an older DEC Alpha 564, a Pentium III, and an SGI Octane scored in the 500,000-700,000 whetstone range. This was followed by the PowerMac G4 systems and the Cray C-90 which all scored in the 300,000-400,000 whetstone range. The poor Cray results may seem surprising, but they make sense because the Whetstone benchmark code has not been structured for efficient vectorization, and the C-90 fares quite poorly in heavy scalar computations. The PowerMac G4 systems suffer a similar drawback without the parallel vector processing capabilities of AltiVec.

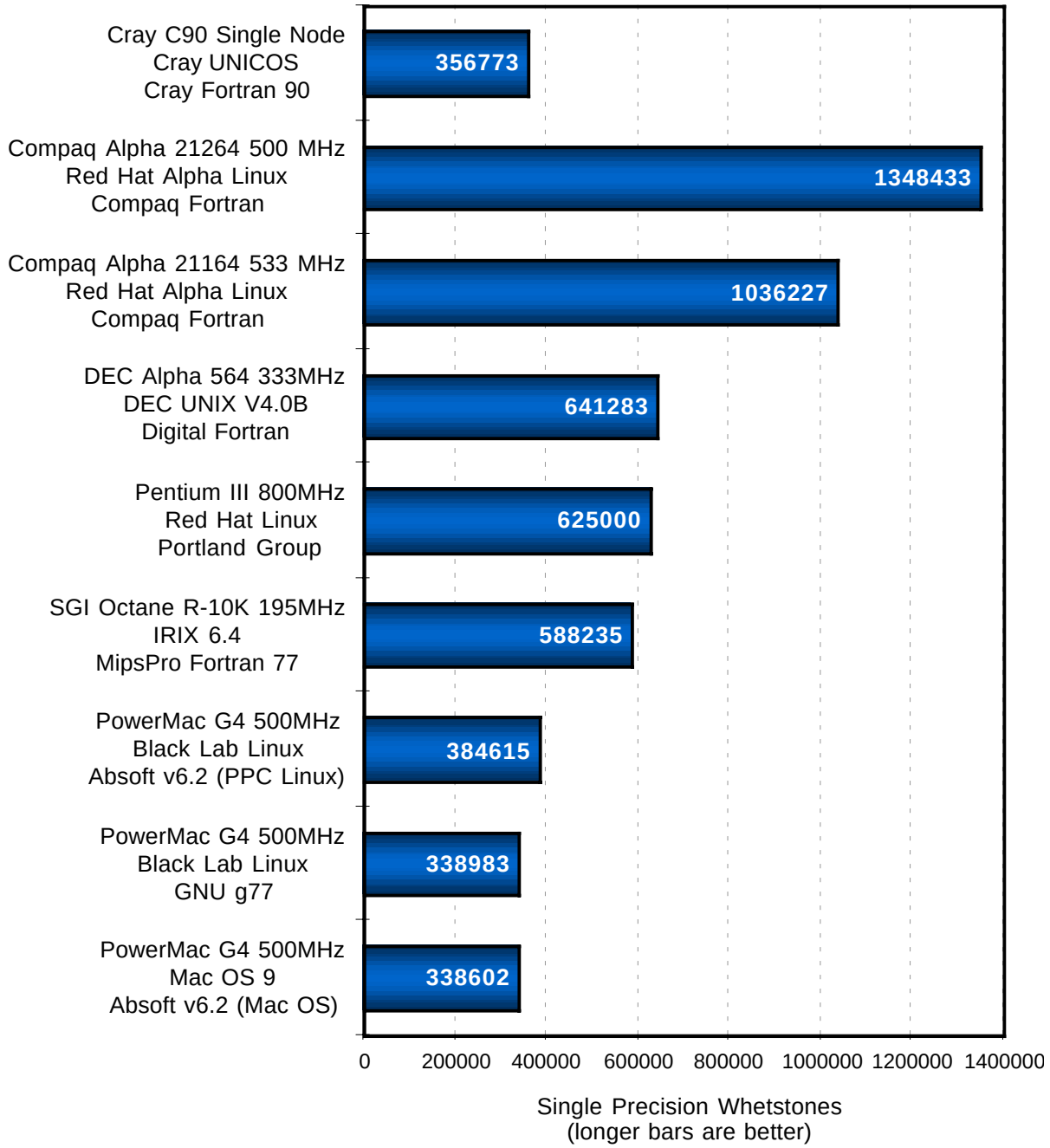

Figure 1: Whetstone Benchmarks

While the Whetstone benchmark tested general F77 performance, the second benchmark was specifically targeted towards CFD and was based on the two-dimensional Euler and Navier-Stokes code "ARC2D" [Reference 8]. For benchmarking, ARC2D was run in Euler (inviscid flow) mode, and used to simulate flow over a 10% thick bi-convex airfoil at a Mach number of 0.8 and 0° angle of attack. ARC2D solves the Euler equations in generalized curvilinear coordinates using an implicit finite difference algorithm with approximate factorization and diagonalization of the implicit operators. The numerical algorithm is a two time level, first or second order accurate time implicit scheme, with second order central differences in computational space. ARC2D was chosen for benchmarking because it is a well tested and documented code and its performance is known to parallel that of other CFD codes on various computer platforms. Thus, ARC2D represents a good test case for the current study.

Benchmarks from ARC2D are shown in Table 2 and Figure 2. Most of the PC and workstation-class machines produced performance in the range of 130-145 MFLOPS. Performance of the PowerMac G4 ranged from 131-136 MFLOPS with the various OS/compiler combinations tested, placing the G4 system at the lower end of the performance spectrum. The two standouts in the ARC2D benchmark test were the Cray C-90 (single node) and the Compaq Alpha 21264/500MHz, giving performance levels of 324 MFLOPS and 332 MFLOPS, respectively. Like most other CFD codes, ARC2D has been structured and coded to allow efficient vectorization, which is implemented automatically by the Cray FORTRAN compiler. This allows the Cray to process data in parallel, providing a significant performance boost (parallel processing and vectorization will be discussed in more detail later on). If AltiVec compiler support had been available, the G4 would likely have attained similar performance.

Computer – OS – Compiler –	**Compiler Command and Optimization Flags**	**MFLOPS**
Cray C90 Single Node Cray UNICOS Cray Fortran	f90 -O3,aggress,allfastint,inline2	324
Compaq Alpha 21264 500MHz Red Hat Alpha Linux Compaq Fortran	fort -O -tune ev6 -arch ev6 -fast	332
Compaq Alpha 21164 533MHz Red Hat Alpha Linux Compaq Fortran	fort -O -tune ev5 -arch ev5 -fast	145
DEC Alpha 564 333MHZ DEC UNIX V4.0B Digital Fortran	f77 -O3	131
Pentium III 800MHz Red Hat Linux Portland Group	pgf77 -O2 -Mbackslash -Minline=2 -Minfo=inline	143
SGI Octane R-10K 195MHz IRIX 6.4 SGI Fortran	f77 -O3	139
PowerMac G4 500MHz Black Lab Linux Absoft PPC/Linux v6.2	f77 -O	133
PowerMac G4 500MHz Black Lab Linux GNU g77	g77 -O2	136
PowerMac G4 500MHz Mac OS 9 Absoft Mac OS v6.2	f77 -O	131

Table 2: Summary of ARC2D Benchmark Results

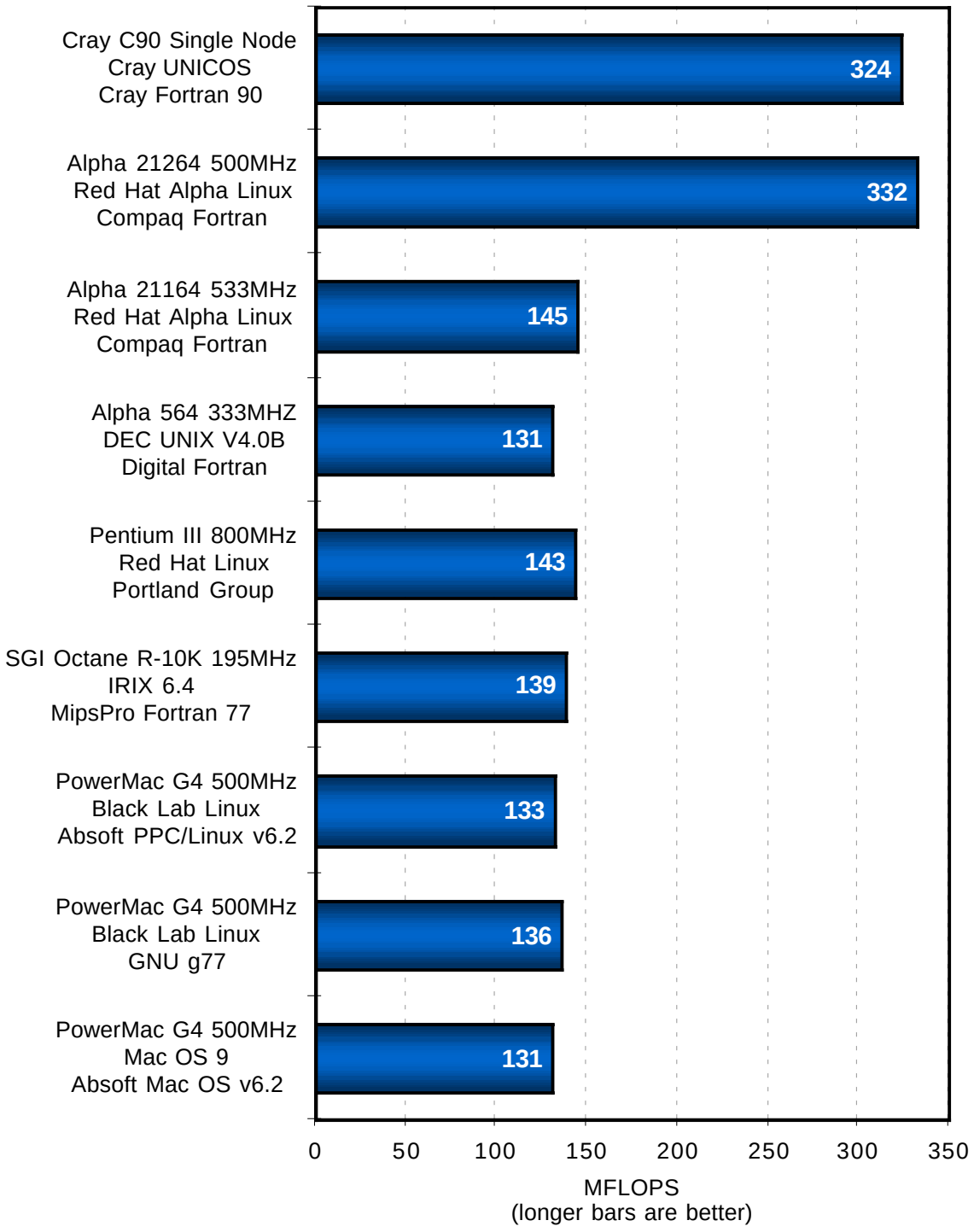

Figure 2: ARC2D Benchmarks

Cost and Price/Performance

For comparison, three similarly equipped systems were priced up – a Compaq Alpha 21264 system, a Micron Millenia Pentium III system, and a PowerMac G4 system. All prices shown are at government rates, with the exception of memory which was quoted at market price. Prices are current as of 6/15/00.

Computer System	Cost ($)	ARC2D MFLOP	$/MFLOP
Compaq XP1000 ($5308) Alpha 21264 500MHz 9.1GB SCSI HD ($370) 1GB ECC Memory ($1800) PCI Fast Ethernet ($80)	7558	332	23
Micron Millenia Max ($1256) Pentium III 800MHz 10GB ATA-66 HD 1GB PC-133 SDRAM ($1260) PCI Fast Ethernet ($80)	2596	143	18
PowerMac G4/500 ($2175) PowerPC G4 500MHz 10GB Ultra ATA HD 1GB PC-100 SDRAM ($1176) 10/100 Ethernet	3351	136	25

Table 3
Cost Comparison

Using ARC2D performance levels as a basis, we can come up with a $/MFLOP metric for these three systems, which is summarized in the chart below. As shown, the PowerMac G4 system is competitive with the Compaq Alpha system, with metrics of $25/MFLOP and $23/MFLOP respectively, but both are undercut by the Micron system at $18/MFLOP. It should be noted that the Micron system was priced up in order to provide a fair comparison with the "commercially available" PowerMac and Alpha systems; lower priced custom-built Pentium III systems can be put together for as little as $10-$12/MFLOP, further increasing the price/performance advantage of Intel based PC systems.

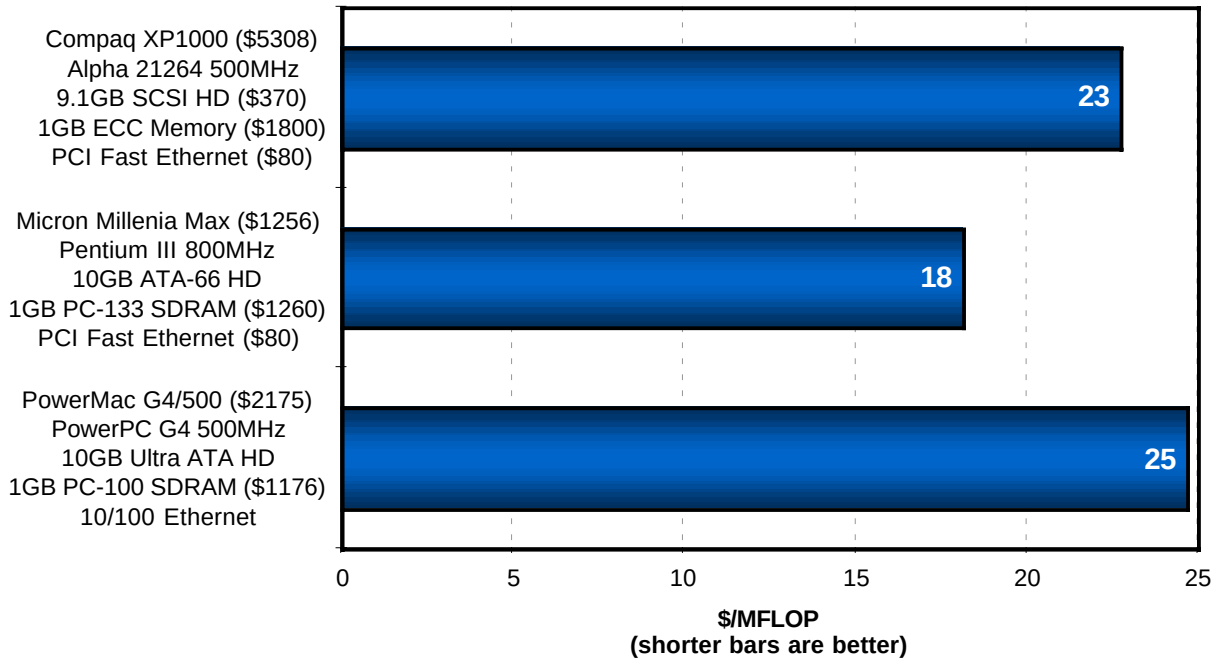

Figure 3: $/MFLOP Price/Performance Summary

AltiVec Performance

While AltiVec compiler support is not available for general F77 computations, Absoft has implemented AltiVec in a limited number of F90 vector functions and BLAS routines in their the Mac OS v6.2 compiler. These operations are accelerated under AltiVec by providing vectorization and 4-way parallel processing of single precision floating point computations [Reference 6]. To test this feature, a benchmark code was developed using the F90 "matmul" function, which multiplies matrices in array form. In this test, 200x200 matrices A and B were multiplied to form the 200x200 matrix C, and the computation was repeated 100 iterations for more accurate timing. The test code is listed below:

```
parameter (n=200,nloops=100)
implicit real (a-z)
dimension tarray(2),a(n,n),b(n,n),c(n,n)
integer i,j
pi=4.*atan(1.)
do i=1,n
  do j=1,n
    a(i,j)=real(j)*real(i)/pi/1024.**2.
    b(i,j)=real(j)/sqrt(pi)*real(i)/1024.
  enddo
enddo
time=DTIME(tarray)
do i=1,nloops
  c=matmul(a,b)
enddo
time=DTIME(tarray)
mflops=real(nloops*2*n**3)/time/1000000.
write (6,*) 'time:',time
write (6,*) 'MFLOPS:',mflops
end
```

The test code was run on the 500MHz PowerMac G4, 800 MHz Pentium III, and 500 MHz Alpha 21264 systems used in the previous price/performance comparison. Results are summarized below:

Computer System	Compile/Optimization Command	Seconds	MFLOPS
Alpha 21264 500MHz Red Hat Alpha Linux Compaq Fortran Compiler	fort -O -tune ev6 -arch ev6 -fast	5.6	286
Pentium III 800MHz Red Hat Linux Portland Group Compiler	pgf90 -O2 -Munroll	10.3	155
PowerMac G4 500MHz Mac OS 9 Absoft Compiler v6.2 **AltiVec OFF**	f90 -O	14.6	110
PowerMac G4 500MHz Mac OS 9 Absoft Compiler v6.2 **AltiVec ON**	f90 -O -altiVec	1.5	1067

Table 4: Summary of F90 "matmul" Benchmarks

The MFLOPS benchmark was obtained by dividing the time benchmark into the number of floating point operations (FLOP) needed to perform the matrix multiplication with traditional scalar computations (involving nested DO loops). Using an operation count of 1 FLOP for each scalar multiply and add, the multiplication of NxN matrices requires approximately $2N^3$ FLOP [Reference 9]. For 200x200 matrices, repeated 100 times, this results in a total of 1596000000 FLOP, or 1.596 GFLOP.

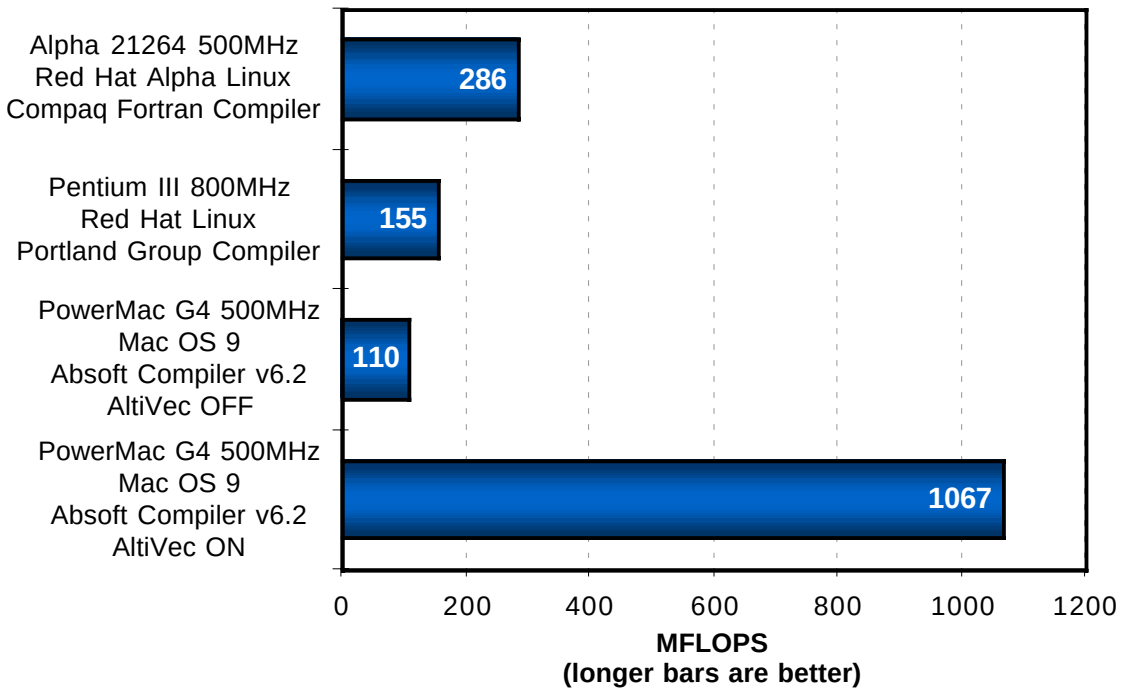

Figure 4: F90 "matmul" Benchmark

As shown in Figure 4, the effects of AltiVec are dramatic. Without AltiVec, the G4 was about 2.6X slower than the Alpha and 1.4X slower than the Pentium II. These results are consistent with earlier benchmarks presented in this study. With AltiVec, however, the G4's performance went up by a factor of nearly 10, making the G4 about 3.7X faster than the Alpha and 6.9X faster than the Pentium III. These results are impressive, and while this single function benchmark is not necessarily an indicator of overall computational performance in real world scientific codes, it does demonstrate the potential of AltiVec.

Using the cost basis developed in the previous comparison and the performance numbers developed from the matmul timing, we can come up with the following price/performance benchmarks:

Computer System	Cost ($)	matmul MFLOP	$/MFLOP
Compaq XP1000 Alpha 21264 500MHz	7558	286	26.42
Micron Millenia Max Pentium III 800MHz	2596	155	16.75
PowerMac G4/500 PowerPC G4 500MHz	3351	1067	3.14

Table 5: Price/Performance based on "matmul" Benchmarks

Here, the G4 is roughly 8X less expensive than an Alpha system, and 5X less expensive than an off the shelf Pentium III system. Looking at this from another direction: with a fixed budget of $30,000, a system of PowerMac G4 computers would provide about 10 GFLOPS of performance, while an Alpha system would provide about 1 GFLOPS and a Pentium III system would provide about 2 GFLOPS. This is an extremely significant result. Again, these numbers are not necessarily an indicator of overall real world price/performance, but they show the potential cost effectiveness of an AltiVec enabled PowerMac G4 computing system.

Implementing AltiVec in General FORTRAN Computations

With AltiVec acceleration, the PowerMac G4 has the potential to become a highly cost-effective option for scientific computing. Unfortunately, AltiVec FORTRAN is in a very early stage, and no means currently exist to call basic AltiVec instructions from FORTRAN. The performance gains in Absoft's AltiVec supported functions and routines are impressive, but this limited set will not be of use in general FORTRAN programming. *To be beneficial, and to gain the widespread acceptance of scientific code developers, AltiVec instructions must be accessible to general FORTRAN computations, and vectorization should be handled by the compiler, similar to the way the Cray compiler implements vectorization.* To facilitate the discussion of implementing AltiVec in FORTRAN computations, we'll start off by looking at parallel processing and vectorization on the Cray C-90.

Parallel Vector Computations on the Cray C-90

On the C-90, parallel processing comes from dual functional units that provide two results per CPU cycle during vector operations [Reference 10]. A vector is a "set" of floating-point data elements that the computer can access as a unit, to perform the same operations on each element in the set. Consider the following example DO loop with a simple floating point add:

```
do i=1,1000
  C(i)=A(i)+B(i)
enddo
```

Since every element of array A is added to the corresponding element in array B, both A and B can be considered vectors in this operation, as can array C. On the Cray, this loop would be executed as follows. To start off, the first 128 elements (words) of arrays A and B would begin loading, simultaneously, into vector registers A and B (see Figure 5 below).

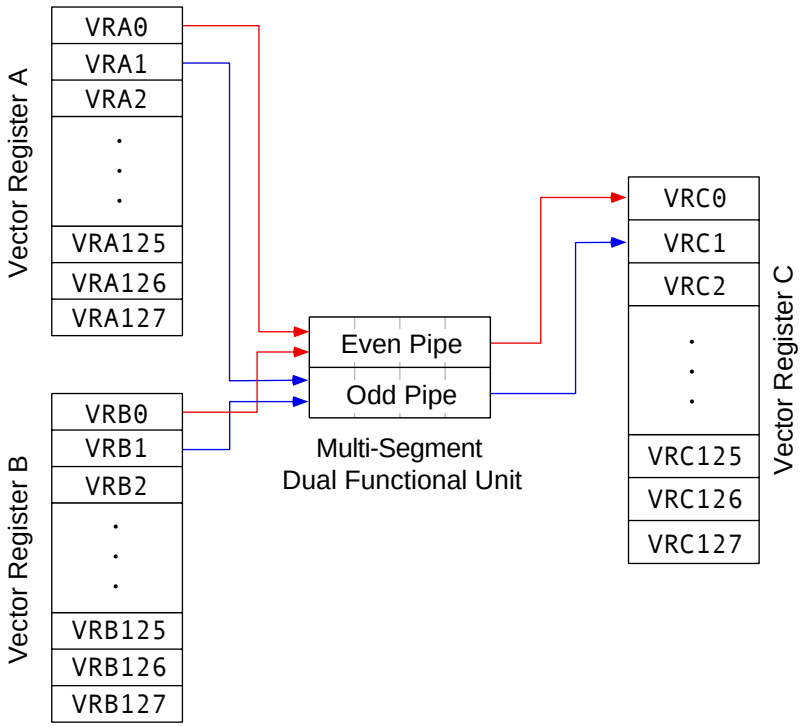

Figure 5: Cray C-90 Vectorization and Pipelining [Reference 10]

Once sufficient data is loaded into the vector registers, element VRA0 and element VRB0 enter the "even" pipe of the multi-segment functional unit, and elements VRA1 and VRB1 enter the "odd" pipe. As elements VRA0 and VRB0 move through the even pipe, they would be followed by elements 2, 4, 6, . . . etc. Similarly, elements VRA1 and VRB1 would be followed by elements 3,5,7, . . . in the odd pipe. When the functional unit is filled, it begins to output data, 2 elements at a time, to vector register C. Once the first 128 elements of A and B have been processed, the next 128 elements of A and B load into the vector registers, and the process continues over and over, until all 1000 elements are processed. In the end, the 1000 element vector operation is carried out on seven 128 element sub-vectors and one 104 element sub-vector from each array. [Reference 10]

The benefit of pipelining in the multi-segment dual functional unit can only be realized when processing vector data, because it gives the functional unit access to an entire "set" of data. During vector operations, the system places new values into the functional unit as soon as the previous values have cleared the first segment [Reference 10]. This is illustrated in the sequence shown below in Figure 6, where each frame represents a clock cycle. Once the dual functional unit fills, it begins to output two elements of data with every clock cycle as discussed above.

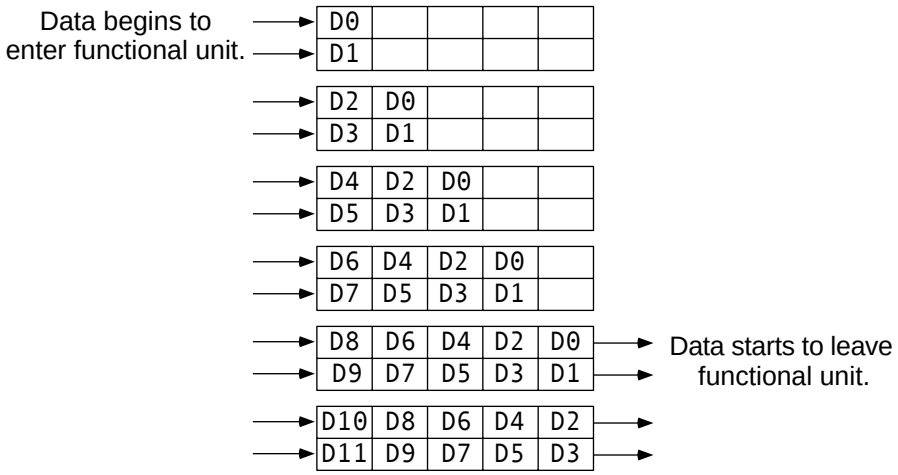

Figure 6: Pipelining Vector Data

During nonvector (scalar) operations, only one segment is performed at any given time because only one element of data is available to the functional unit. In scalar mode, the functional unit outputs one element of data every K cycles, where K is the number of segments in the unit (see figure 7 below). Clearly, the vector processing mode is much more efficient, and in fact, vectorized loops can execute from 10 to 40 times faster than nonvectorized loops [Reference 10].

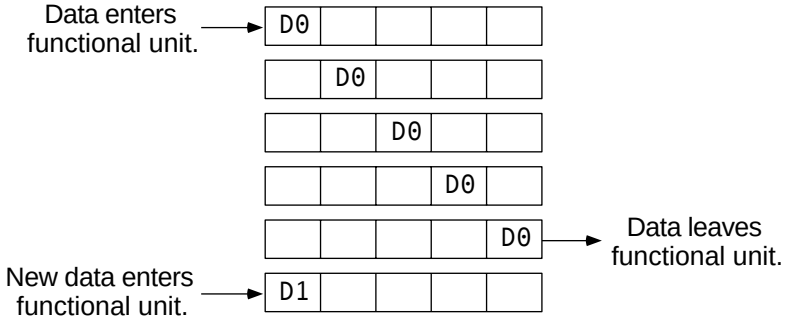

Figure 7: Processing Scalar Data

While vectorization is done automatically by the Cray FORTRAN compiler, it is applied only to innermost DO loops where vectorization is suitable. An inner DO loop is considered suitable for vectorization if the compiler determines that it will produce the same computer arithmetic results in either vector or scalar mode [Reference 10]. Computations with multiple nested DO loops will only realize partial benefit, from vectorization of the innermost loop. As a result, DO loop coding for vector machines like the Cray C-90 takes a little extra planning. For example, the following two nested loops take the same amount of time to process on a scalar computer, but Loop B would execute significantly faster on a vector machine because the large loop is arranged to be the innermost loop.

```
                do i=1,1000
                  do j=1,3
LOOP A:             x(i,j)=y(i)*z(j)
                  enddo
                enddo


                do j=1,3
                  do i=1,1000
LOOP B:             x(i,j)=y(i)*z(j)
                  enddo
                enddo
```

Loop A's inner loop would form a vector of length 3 that fits within one 128 word register. This one vector requires 4 operations: two loads (for y and z), a multiplication, and a store (for x). The inner vector loop is then repeated 1000 times. This would result in 1*4*1000 = 4000 vector operations. Loop B's inner loop, on the other hand, would fit into 8 (the ceiling of 1000/128) 128 word registers, each requiring 4 operations, and is repeated 3 times. This results in 8*4*3 = 96 vector operations, which is clearly much more efficient than the 4000 of loop A.

Often, it is necessary to perform computations over a two- or three-dimensional "grid" of data, such as the two-dimensional MxN grid shown below in Figure 8.

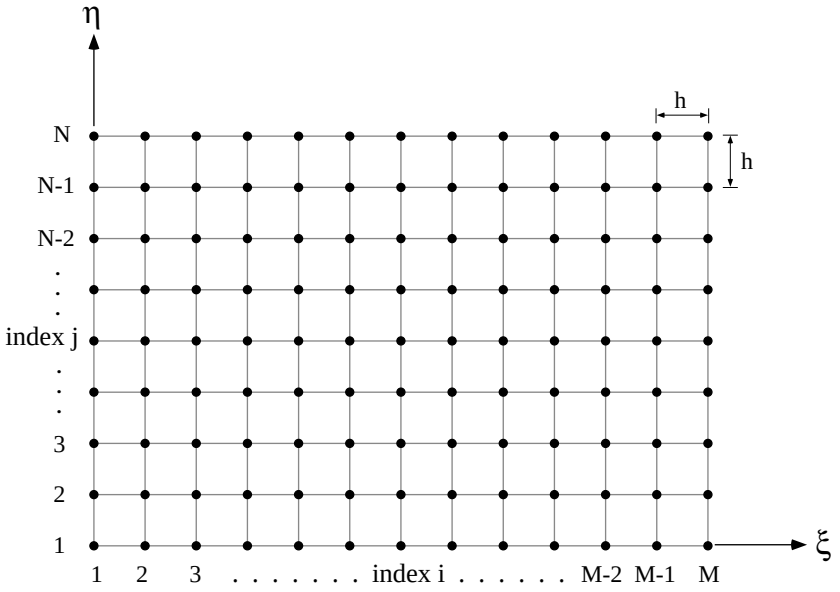

Figure 8: MxN data grid

This example grid consists of equally spaced "node" points, separated a distance h apart. The grid is aligned along the ξ and η directions, and is organized by i and j indices. Consider the case where we have a data variable ϕ(i,j) at each node. If we wanted to calculate some function from ϕ, say $f(\phi) = \phi^2 - \phi$, at each (i,j) point in the grid, we could use the following approach:

```
do i=1,M
  do j=1,N
    f(i,j)=phi(i,j)*phi(i,j)-phi(i,j)
  enddo
enddo
```

In general, there's no best way to arrange this nested loop structure for arbitrary M or N, and either way, this is a situation where vectorization will only apply to the inner loop. Fortunately, the two loops can be combined into a single vectorizable loop by lumping the i and j indices into a single ij index that threads through the data as shown in Figure 9.

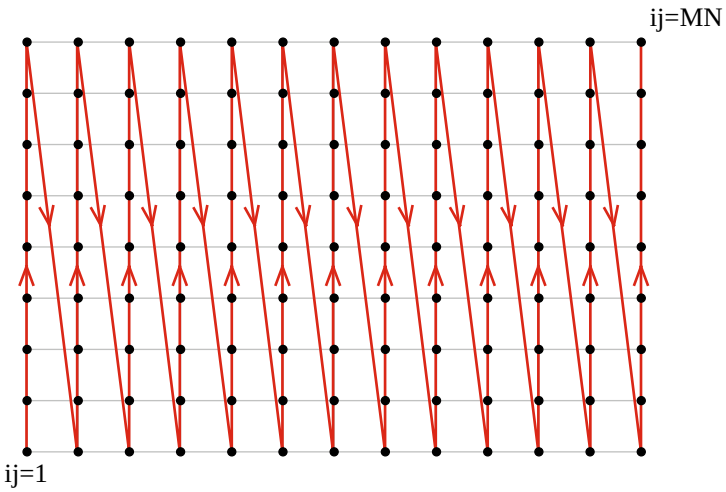

Figure 9: Index Reduction

With this new convention, the loop can be rewritten as:

```
do ij=1,M*N
  f(ij)=phi(ij)*phi(ij)-phi(ij)
enddo
```

which allows full vectorization. Though this was a very basic example, this approach can be applied to complicated cases with multiple nested DO loops and more elaborate equations, as long as no inter-loop dependencies or recursions will be broken. Most high-end CFD codes use this approach for improved performance on vector machines. An important side benefit to this method is that it still runs fine on non-vector (scalar) machines, albeit without the benefits of vectorization.

AltiVec

As shown in Figure 10, AltiVec adds a 128-bit wide vector "execution unit" to the PowerPC architecture. This vector unit operates independently from the integer and floating point units.

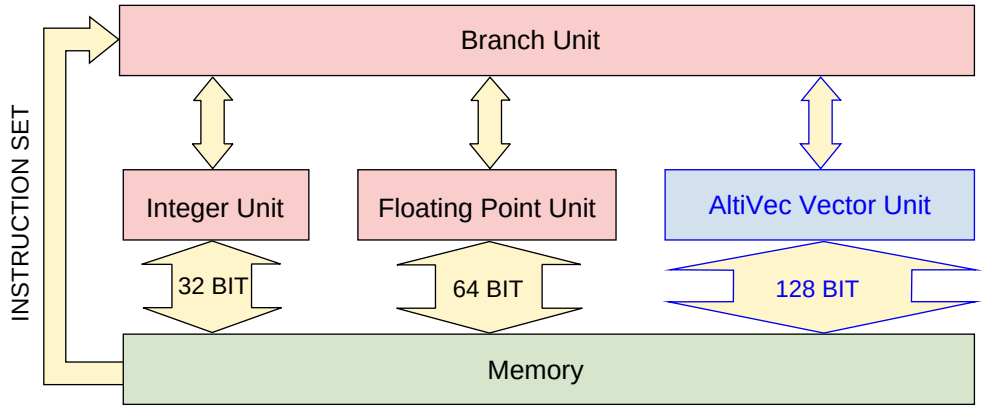

Figure 10: AltiVec Architecture

Within the vector unit, data is held by thirty-two 128-bit "short vector" registers, each capable of holding 4 single precision floating point numbers (i.e., 4 32-bit words), 4 integer words, 8 integer halfwords, or 16 integer bytes (this is in contrast to the "long vector" registers of the Cray C-90 which hold 128 words). See Figure 11. To take advantage of the AltiVec unit, 162 new vector operations have been added to the PowerPC instruction set. Arithmetic operations include add, multiply, multiply-add, subtract, absolute value, truncate, round, sum, multiply-sum, average, and exponent. Simple combinations of these functions allow for the fast and easy computation of operations such as dot product. All AltiVec instructions are fully pipelined, with single cycle throughput. During execution, each AltiVec instruction can address up to three source vector registers and one destination vector register. [References 5, 11]

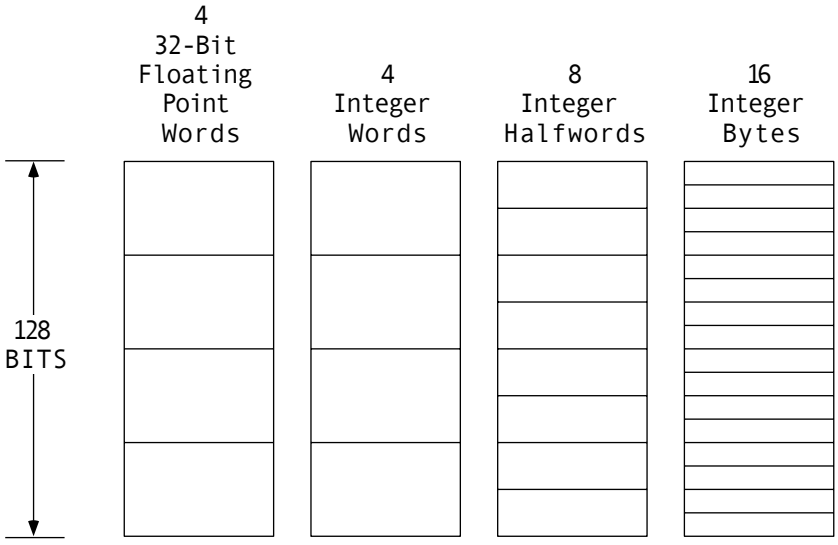

Figure 11: AltiVec Vector Data Types

Unlike the Cray C-90 which operates on two vector elements at a time in the dual functional unit, AltiVec operates on all data elements in a vector register simultaneously, with every instruction. This is known as SIMD – "single instruction, multiple data" – parallel processing. In computations involving single precision floating point numbers, AltiVec can offer 4-way parallelism by simultaneously operating on 4 elements of a vector with every instruction. Consider our earlier example involving a simple DO loop containing a floating point add. The purpose of this DO loop is to add each element in array A to the corresponding element in array B, assigning the result to the corresponding element of array C. It is a vector sum, implemented with scalar computations:

```
do i=1,1000
  C(i)=A(i)+B(i)
enddo
```

As before, each computation involves a total of 4 instructions (2 loads, an add, and a store). On a scalar machine, this would require a total of 1000*4 = 4000 instructions. With AltiVec, the computation can be carried out on 4 elements of data at a time, as shown below:

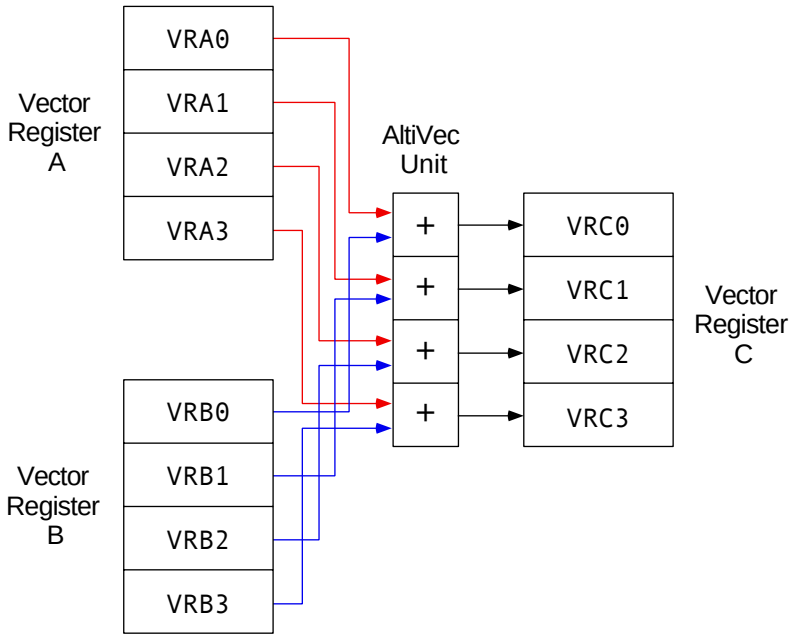

Figure 12: AltiVec Computation Scheme

AltiVec performs the computation on 250 sub-vectors (4 elements each), for a total of 250*4=1000 instructions. Thus, the total number of instructions is reduced by a factor of 4, which should provide a significant reduction in the time required to process the computation. While this is an easy way to improve performance, the unfortunate fact is that no means currently exist to implement this type of AltiVec computation from FORTRAN. We can, however, use the AltiVec enabled "gcc-vec" compiler on Linux to show the potential performance boost this type of vectorization has to offer. In C, the scalar computation loop would look like:

```
n=1000;
for(i=1;i<=n;i++)
{
c[i]=a[i]+b[i];
}
```

where a[ ], b[ ], and c[ ] are defined as "float". To implement this computation, a timing code was written and compiled with gcc on a PowerMac G4/500 running Black Lab Linux. For a benchmark, the loop computation time was divided into the number of scalar instructions in this vector add (4000). For the basic scalar loop shown above, the benchmark was 42 million instructions per second (MIPS).

Using AltiVec's "`vec_add`" operation, the computation loop can be rewritten as:

```
n=1000;
for(i=1;i<=n/4;i++)
{
c[i]=vec_add(a[i],b[i]);
}
```

Here, a[ ], b[ ], and c[ ] are defined as "vector float". Note that the loop count is reduced by a factor of 4 (upper limit is changed from n to n/4); it goes from 1 to 250 because the `vec_add` function will actually operate on 250 4-element sub-vectors. Using this vector approach and the gcc-vec compiler, the benchmark jumps to 146 million instructions per second – an increase of about 3.5X. The benchmarks are summarized in the table and chart below.

Computer	Compile Command	MIPS
PowerMac G4 500MHz Black Lab Linux GNU gcc Compiler **Scalar Computation**	gcc	42
PowerMac G4 500MHz Black Lab Linux gcc-vec Compiler **AltiVec Vector Computation**	gcc-vec -fvec	146

Table 6: Summary of Floating Point Vector-Add Benchmarks

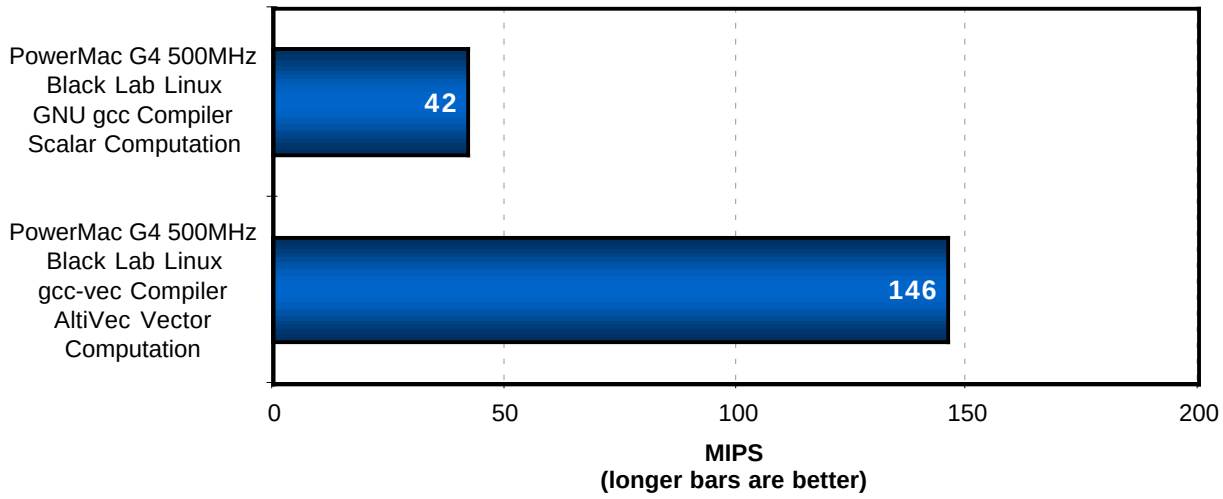

Figure 13: Floating Point Vector Add Benchmarks

Clearly, the vector approach offers a significant performance boost with a minimum of code tweaking. *Conceivably, such minor code changes could be automated by a compiler or preprocessor, as in the case of the Cray compiler.* Some generic examples follow (using FORTRAN structure with C AltiVec operations). In each case, relevant variables are defined as needed. For more details on AltiVec instructions, see Reference 12.

Example 1 - Scalar Multiply-Add: x = ay + z, where x, y, and z are vectors, and a is a scalar.

```
Old  Scalar  Loop:
real x( ),y( ),z( ),a
integer i,N
do i=1,N
  x(i)=a*y(i)+z(i)
enddo

New  AltiVec  Loop:
real x( ),y( ),z( ),a,avec(4)           !> note: treat 1D arrays as vectors
integer i,N
vec_splat_float(avec,a)                 !> create the vector avec=(a,a,a,a)
do i=1,N/4
  x(i)=vec_madd(avec,y(i),z(i))         !> vector multiply-add
enddo
```

In general, DO loops may not be evenly divisible by 4, but this can be handled in several ways – either by "padding" the arrays with dummy elements (rounding the array size up to the next largest multiple of 4), or using a scalar "cleanup" loop as shown in the revised AltiVec code below:

```
real x( ),y( ),z( ),a,avec(4)       !> note: treat 1D arrays as vectors
integer i,N
vec_splat_float(avec,a)             !> create the vector avec=(a,a,a,a)
do i=1,N/4
  x(i)=vec_madd(avec,y(i),z(i))     !> vector multiply-add
enddo
M=mod(N,4)                          !> M = remainder of N/4 = 0,1,2, or 3
do i=N-M+1,N
  x(i)=a*y(i)+z(i)                  !> scalar computation of leftovers
enddo
```

Note that in cases where N is evenly divisible by 4, M=0 and the scalar cleanup loop will be bypassed.

Example 2 – Multiple Vector Operations: x = yz + uv + w, where x, y, z, u, v, and w are vectors.

```
Old  Scalar  Loop:
real x( ),y( ),z( ),u( ),v( ),w( )
integer i,N
do i=1,N
  x(i)=y(i)*z(i)+u(i)*v(i)-w(i)
enddo

New  AltiVec  Loop:
real x( ),y( ),z( ),u( ),v( ),w( )      !> note: treat 1D arrays as vectors
integer i,N
do i=1,N/4
  x(i)=vec_madd(y(i),z(i),vec_madd(u(i),v(i),w(i)))   !> nested multiply-add
enddo
```

Conclusions and Recommendations

The PowerMac G4 system has the potential to be an inexpensive high performance scientific computing platform. Much of that potential is currently unrealized, however, due to the limited amount of AltiVec support in FORTRAN. Without the parallel vector processing capabilities of AltiVec, the G4 places near the end of the pack in performance tests using standard FORTRAN scientific codes. Because of its lower cost, the PowerMac G4 is still price/performance competitive with Alpha-based workstations, but both are undercut by inexpensive Pentium III systems. In limited cases where AltiVec acceleration was available and tested under FORTRAN, the G4 showed a clear advantage with 4-7X greater performance and a 5-8X greater cost effectiveness than all other workstation systems evaluated.

Curently, AltiVec acceleration is only available in the C programming language and a limited number of FORTRAN functions, and these are not comprehensive enough to make the G4 a general purpose scientific computing machine. This is especially true for computational fluid dynamics codes, which are heavily based in FORTRAN. To gain the widespread acceptance of scientific code developers, AltiVec acceleration must be readily accessible to standard FORTRAN programming. The examples presented in this report show that only minor re-coding would be necessary to implement AltiVec instructions if they were made available via FORTRAN. Furthermore, vector restructuring of computation loops is simple enough to be automated by a compiler or preprocessor. It is envisioned that relatively simple "AltiVectorizer" software could be developed to convert standard FORTRAN scalar code into AltiVec-enabled vector code where possible. The automated vectorization and parallel processing capabilities implemented by the Cray FORTRAN compiler are good examples of how standard FORTRAN coding can be adapted to take advantage of machine-specific features.

While the main purpose of this paper was to evaluate the potential of PowerMac G4 systems for FORTRAN-based scientific computing, it is hoped that this paper can serve a more important role as a "catalyst" for further development of AltiVec FORTRAN on the G4. The G4 hardware holds tremendous potential for scientific computing, but we're not quite there yet in terms of programming tools, software, and overall integration. Thus, there appear to be many opportunities for collaboration between Apple Computer, developers, and scientific users. It is recommended that these communities meet in the near future to exchange ideas, share experiences, and discuss the advancement of scientific computing on the PowerMac G4 platform.

Acknowledgements

The author would like to thank Russ Pond and other folks at Apple Computer for the loan of the PowerMac G4 system, Connie Jenkins of GMR for the loan of RAM, Kai Staats and Troy Benjegerdes of Terra Soft Solutions for help and tech support with Black Lab Linux, and Brian Helinski of Absoft for help and support with the Absoft compiler. At Langley, thanks go to Steven Massey of AS&M for his help with benchmarking, code porting, and testing, to Paresh Parikh for taking the time to port and compile USM3D, and to Paul Pao and Darren Pfifer for sharing their experiences with Linux and cluster systems. Finally, the author would like to thank Jeff Flamm, Steve Bauer, Rich Wahls, and Larry Leavitt for supporting this activity within the Configuration Aerodynamics Branch.

References

1. The Beowulf Project, web site. http://www.beowulf.org/

2. Project AppleSeed: A Parallel Macintosh Cluster for Numerically Intensive Computing, web site. http://exodus.physics.ucla.edu/appleseed/appleseed.html

3. Black Lab Linux, web site. http://blacklablinux.com/

4. Apple – Products – PowerMac G4, web site. http://www.apple.com/powermac/

5. Fuller, Sam. "Motorola's AltiVec Technology". Motorola, Inc. white paper ALTIVECWP/D, 1998.

6. Absoft Corporation, web site. http://www.absoft.com/

7. Curnow and Wichmann. "Whetstone Benchmark Code". Computer Journal, vol 19 no 1. February 1976, pg 43-49.

8. Pulliam, T. H. "Euler and Thin Layer Navier Stokes Codes: ARC2D, ARC3D". Proceedings, Computational Fluid Dynamics Workshop held at the University of Tennessee Space Institute. UTSI publ. E02-4005-023-84, 1984.

9. Hockney, R. and Berry, M., eds. "Floating-Point Operation Count", section 2.3.5 of "Public International Benchmarks for Parallel Computers". Report 1, February 1994. Web Address: http://www.epm.ornl.gov/~walker/OLD_ORNL_WEB_PAGE/benrep4/section2_3_5.html

10. NAS Cray Vector Processor User Guide, Section 2.4 "Code Optimization and Vectorization" http://www.nas.nasa.gov/Groups/HSP/UserGuide/ch2.html#2.4.

11. Clarke, Douglas. "AltiVec Technology". Apple Computer 1998 World Wide Developer's Conference Presentation. Available at: http://developer.apple.com/hardware/altivec/summary.html

12. AltiVec Technology Programming Interface Manual. Motorola, Inc. paper ALTIVECPIM/D, 1999.