SLALOM

The First Scalable Supercomputer Benchmark

John Gustafson, Diane Rover,
Stephen Elbert, and Michael Carter
Ames Laboratory-USDOE
Ames, IA 50011

SLALOM stands for Scalable, Language-independent, Ames Laboratory, One-minute Measurement. We didn't force-fit that acronym. It was pure luck that it turned out to mean "a timed race through obstacles."

"How long will my computer take to run **SLALOM**?" One minute. That's the wrong question. It *always* takes one minute! The right question is, "How big a **SLALOM** problem will my computer solve?"

For years, we've said that the right way to measure computers is to fix *time*, not problem size. The trouble is, all the popular benchmarks are of *fixed size*, which is one reason they have a way of going out-of-date as computing power increases. By using the principle of *fixed-time* benchmarking, it's possible to compare a very wide range of computers, from the smallest personal computer to the most powerful supercomputer, on a *single scale*. **SLALOM** is designed to remain useful for many computer generations. **SLALOM** will be able to benchmark the first 100 TFLOPS computer.

Suppose you wanted to compare a tricycle to a jet. The worst way to do it would be to pick a distance and clock them. If you make the distance 100 miles, the poor tricyclist probably won't even make it. If you make the distance 100 feet, the tricyclist will probably win! Instead, why not ask, "How far can each go in *one hour*?" The ratio of the distances traveled is a fair measure of their relative performance.

Over a year ago, we set out to create a benchmark for "grand challenge" supercomputing, one that would span the wide spectrum of computing approaches in use today, and one that shouldn't go out-of-date for several decades. Most "benchmarks" are simply excerpts from programs that have been run on multiple machines, or "synthetic" combinations of sample operations that correlate poorly with entire problems. Building a good benchmark is a challenge, since it has to be unprejudiced toward any machine or programming environment, free from undiscovered "shortcuts," and capable of self-verification. Most challenging of all is to create a single program that captures salient features of scientific computing generally.

SLALOM corrects several deficiencies in various existing benchmarks:

- SLALOM is scalable. By using fixed-time principles, it can compare a Macintosh to a CRAY, and do justice to both.
- SLALOM does not have a preferred computer language.
- SLALOM is maintained by a vendor-independent organization.
- SLALOM solves a representative real problem, in its entirety. It includes problem set up, input, and output times.
- SLALOM can be run on scalar, vector, and parallel machines of all kinds.
- The SLALOM report tells you who ran the benchmark (no anonymous submissions).
- SLALOM ranks performance by *problem size solved*, not MFLOPS or other esoterica.
- SLALOM states precision requirement in terms of answer accuracy, not hardware "word size."
- The SLALOM ranking is a *single number*, not a multidimensional enigma.
- Program "tuning" is encouraged in SLALOM, without artificial restrictions.

The benchmark has been given shouts of approval from vendors spanning the supercomputing industry. The rules of the benchmark are patently fair, and both vector and parallel approaches get shown in their best light.

Fortran, C, and Pascal definitions of the revised benchmark are available, with variants for SIMD, shared-memory MIMD, distributed-memory MIMD, and vector computers, through "`anonymous ftp`" to a workstation at Ames, `tantalus.al.iastate.edu`, and we encourage submission of the results of any experiments run using this benchmark. Mail results to `slalom@tantalus.al.iastate.edu`.

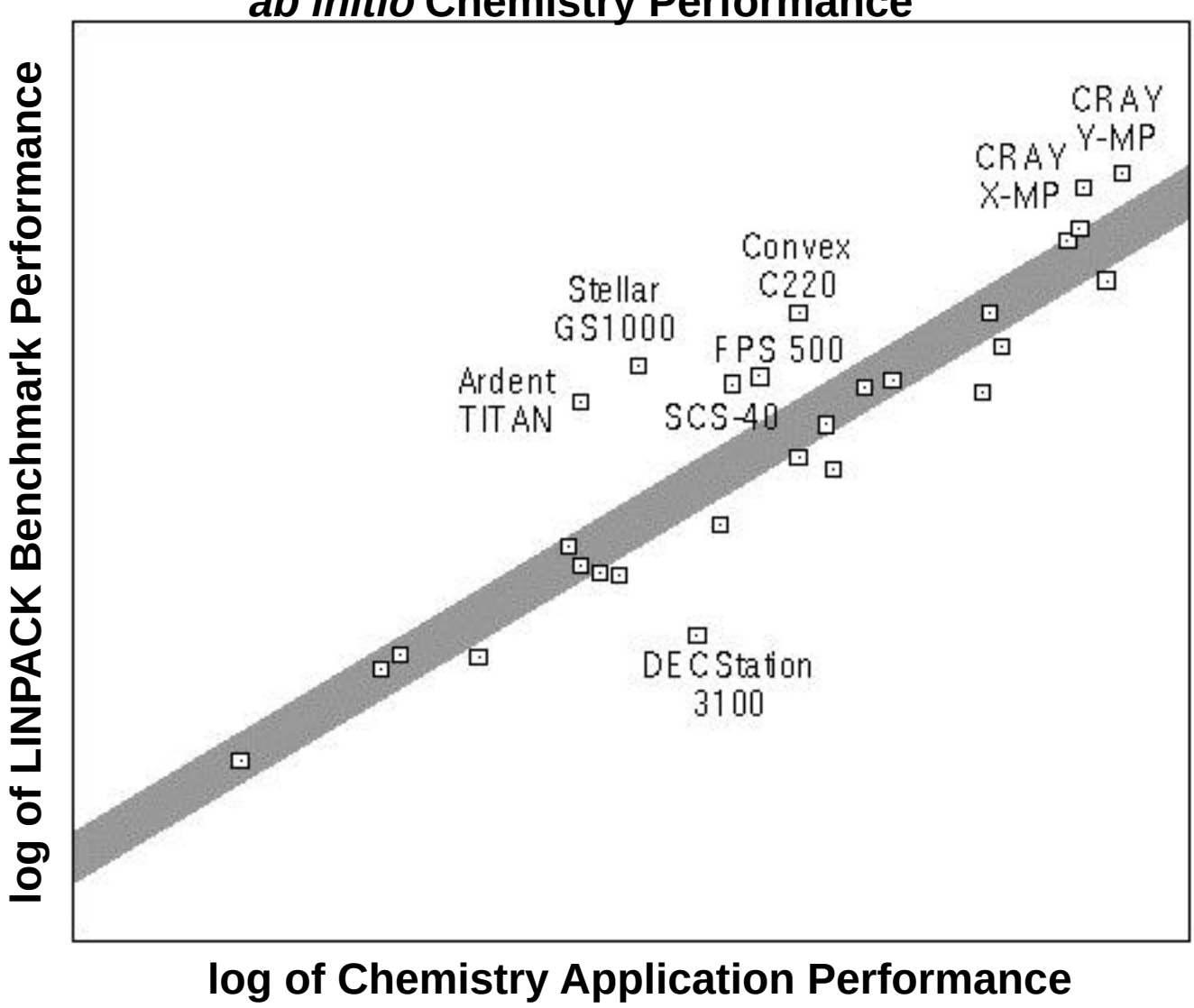
Building on the Experience of Past Benchmarks

It's natural in measuring computer performance that the problem being solved be fixed as the computing power varies. Unfortunately, this natural assumption is highly questionable since it doesn't reflect the way people actually use computers. Generally, problems *scale* to use the available resources, both memory and speed, so that the execution time remains roughly constant. The need for problem scaling has been mentioned in [4], and the more specific goal of scaling the problem to keep execution time constant described in [1].

This is certainly true for the problem of solving systems of linear equations. The popular LINPACK benchmark [2] has undergone at least three size changes in an attempt to keep up with performance increases, but already the latest version is looking too small for the most powerful computers. When the LINPACK software was first introduced, timings for solving a 100 by 100 system were gathered from a number of institutions and published.

As computers have increased in size and speed, the 100 by 100 problem is increasingly inappropriate for measuring high-speed computers. The fastest CRAY does that problem in less than 1/200 of a second, faster than almost any display can refresh to inform the user that the job is done. Even in 64-bit precision, the CRAY uses only 1/400 of its main memory for that problem. As a result, a new version of the benchmark was introduced for a 300 by 300 problem. When this also began to look small, a 1000 by 1000 version was added. To add to the confusion, variations exist for 32-bit versus 64-bit precision, "rolled BLAS" versus "unrolled BLAS," pure Fortran versus various degrees of hand-coded tuning and compiler directives. This proliferation results from the erosion of restrictions on how to achieve the answer.

There's clearly a need for a "gold standard" in benchmarking scientific computers. Increasingly, however, the LINPACK currency has become a bit inflated. The figure below shows an example of what the LINPACK report itself warns [2]: it "…should in no way be used to judge the overall performance of a computer system." The horizontal axis shows performance on a chemistry program called GAMESS [6], and the vertical axis shows performance on the 100 by 100 LINPACK benchmark.



The PERFECT™ Club makes the improvement of starting with real applications, but has had limited success (7%) in putting parallel versions in its suite. It also lacks input and output, and is aimed at "dusty deck Fortran" environments.

We hope that the SLALOM benchmark will last several decades without fundamental change. It may be the first benchmark with such longevity, and will permit the tracking of technology trends over a wide baseline.

A Complete, Real Problem: Radiosity

We started with a scientific paper from 1984 describing the use of radiation transfer to generate realistic images with diffuse surfaces [3]. The buzzword for this method of rendering is "radiosity," and it has passed ray-tracing as a computational challenge for computer-generated scenes. Stated simply the problem is to find the equilibrium radiation inside a box made of diffuse colored surfaces. The faces are divided into regions called "patches," the equations that determine their coupling are set up, and the equations are solved for red, green, and blue spectral components.

The reason we picked this problem is that it's scalable, involves solving a nearly-dense system of equations, has I/O and set up costs, and appears to lack hidden "shortcuts" that might be unevenly exploited. If you want an answer that's correct to 8 digits, the only way to do the job in general is to set up and solve the equations directly. If there's an unanticipated algorithmic breakthrough, we will simply make it available and watch the performance figures improve.

Instead of obtaining a program from the authors of the radiosity paper, we built one from their description. The only changes made were to calculate the matrix entries from an exact expression for the integrals, instead of approximating the entries by quadrature. We even use the same radiative properties as their example. We also made the problem decomposable into an arbitrary number of patches, from 6 on up, so that the scaling would be as smooth as possible.

What to Time

The LINPACK benchmark times only the solution of the set of equations [2]. The PERFECT Club benchmarks [5] have had all the input and output operations carefully deleted. For a fixed-size problem, these omissions are understandable. For fixed time, one can fairly time everything from when the RETURN key is pressed to when the answer becomes viewable.

Given that the problem is a real one, we can now time the setup of the matrix as well as its solution. Constructing an N by N dense matrix takes order N^2 time, whereas solving that matrix requires order N^3 time (or $N^{2.?}$ for clever block methods with Strassen-type multiplications). For large N, the solving will dominate the benchmark. Yet, the operations needed to set up each element is in the hundreds, so the operation counts are roughly in balance when $N = 200$. The slower machines in our list are spending much of the time setting up the problem instead of solving it.

We go further than this, however. We also time *input* and *output*. The input is in the form of reading a geometry file from mass storage. The output consists of writing the answer (position, size, and color of every patch) to mass storage. While this should pose no burden to most computers now in use, including input and output time will screen out those machines so immature as to lack usable mass storage. These regions are easily projected onto a two-dimensional plane and drawn on a graphics display, and we provide software to do that for anyone interested.

Also, the performance is fed back to the user for each problem size, in a form that summarizes and profiles the run. A typical iteration (this is from the single processor Silicon Graphics 4D/380S) produces something like this:

```
  531 patches:
   Task   Seconds        Operations        MFLOPS     % of Time
Reader     0.001               258.      0.025800        0.0 %
Region     0.001              1148.      1.148000        0.0 %
SetUp1    10.520          20532123.      1.951723       17.8 %
SetUp2    23.130          39372520.      1.702227       39.1 %
SetUp3     0.130            236856.      1.821969        0.2 %
Solver    24.890         135282624.      5.435220       42.1 %
Storer     0.480             25440.      0.053000        0.8 %
 TOTAL    59.160         195425529.      3.303339      100.0 %
residual is 2.2566160051696E-12
Approximate data memory use:  2311600 bytes
```

Verification

We have three ways of verifying answers. The setup of the matrix is cross-checked by seeing that the rows sum to unity. The residual of the matrix solution is found after the job is done. Lastly, answer files can be compared for small and large problem sizes, against examples we maintain with the program versions. Also, displaying the result graphically can quickly show errors.

The Rules

In this first SLALOM benchmark, we fix problem execution time at 60 seconds. The benchmark has logic to time itself and adjust automatically to find that problem size, or the user can do the trial-and-error manually. We can supply versions in C, Fortran 77, Pascal, and eventually other languages. We can also supply variations with compiler directives for shared-memory parallel machines, message-passing versions for distributed memory MIMD, and a MasPar version for massively-parallel SIMD computers.

These are just starting points. You may optimize the program for a given machine so long as you *don't specialize your program to the input data*. These are "grand challenge" rules: if you would undertake an optimization to push a scientific frontier, then you can do it to SLALOM.

If we can't verify the reported performance ourselves, the list will so indicate. The usual system of scientific accountability holds for our list: you are responsible for the numbers you report. You must disclose any special technique that you use.

You Must Be This Tall To Go On This Ride

The smallest SLALOM run possible is one with only 6 patches: one for each face of the box. We weight that job as 8812 floating-point operations, so a computer must be capable of at least 148 FLOPS (not MFLOPS, but *FLOPS*) to run SLALOM in under a minute. The Radio Shack Model 100, a 6-year old lap computer running interpretive BASIC, was too slow to make it. We haven't found any current machines unable to run the benchmark, but if you can't compete for the largest problem, we'd love to have a new record *smallest* problem to show SLALOM's scalability! Even running BASIC, our Macintosh IIcx was well above the entry threshold.

Right now, the list has a dynamic range of about a million to one between the fastest and slowest machines, in terms of MFLOPS. Using "patches" or "kilopatches" compresses the range nicely, and gives a better feel than MFLOPS for the size of a physical problem that can reasonably be solved using that machine.

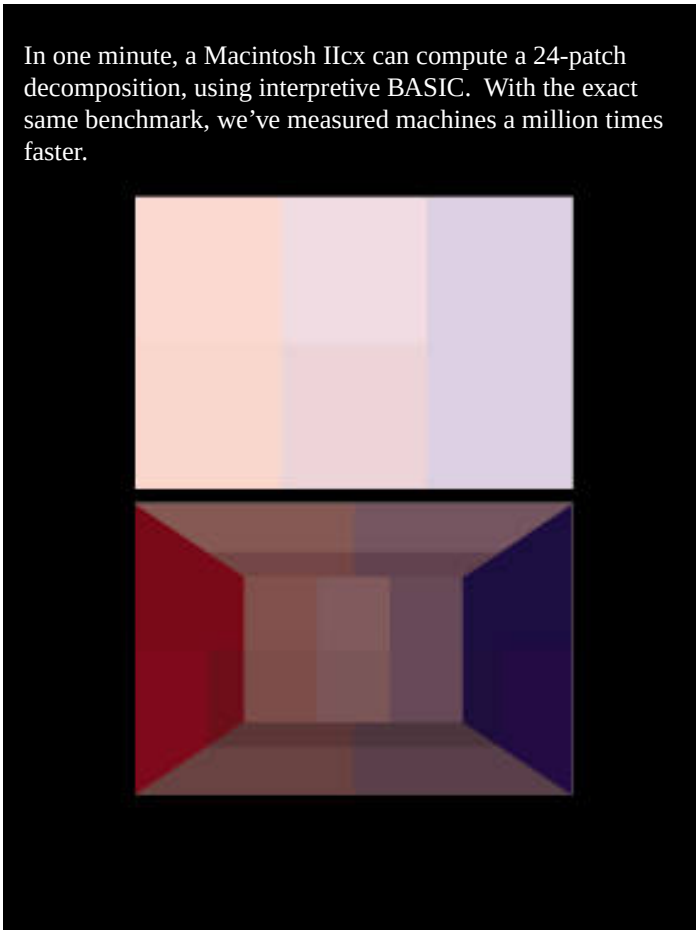
The Report

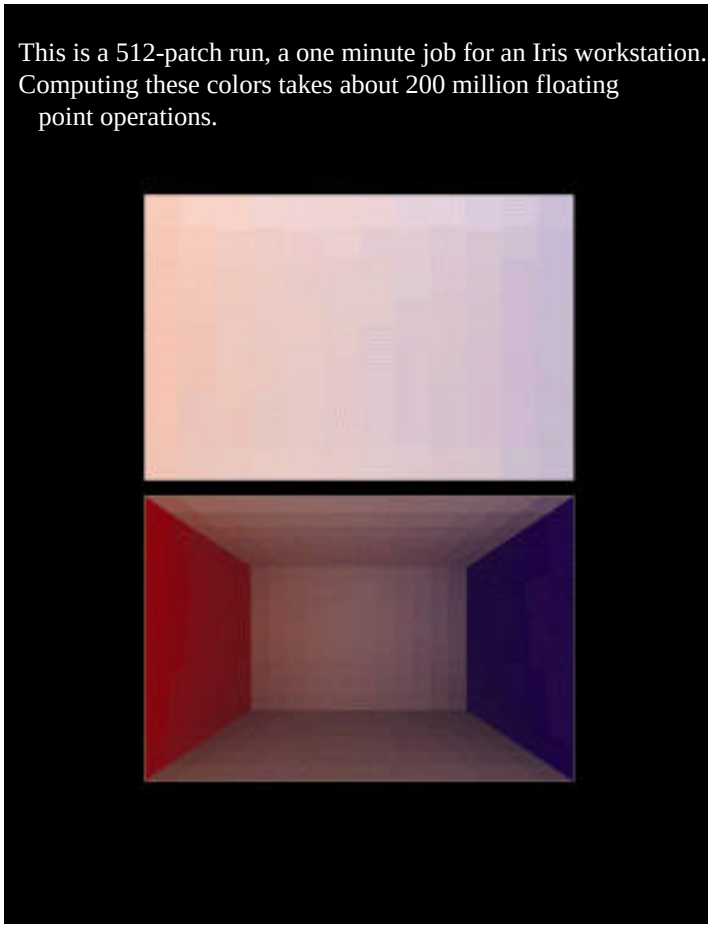
Remember, this is the *first* installment of a rapidly changing list. We don't have Connection Machine or IBM 390 times yet. The NCUBE times only go up to 64 processors. The Intel iPSC/860 and Cogent figures are for only one processor. Our run on the world's only CRAY-2/8 was not tuned nearly as much as was the CRAY Y-MP, so that gap will soon close. But with the publication of this article, the race is now on!

Incidentally, you won't find older machines cluttering up our list. We will keep, and occasionally publish, data interesting from the historical viewpoint, but what we mainly intend to publish are figures for machines that are *actively marketed*.

Machine, environment	Processors	Patches	Operations	Seconds	MFLOPS	Measurer	Date
Cray Y-MP/8, 167 MHz Fortran+tuned LAPACK solver	8	5120	126 G	59.03	2130.	J. Brooks (v) Cray	9/21/90
Cray Y-MP/8, 167 MHz Fortran+tuned LAPACK solver	4	4096	65.2 G	54.81	1190.	J. Brooks (v) Cray	9/21/90
Cray Y-MP/8, 167 MHz Fortran+tuned LAPACK solver	2	3200	31.6 G	56.7	556.	J. Brooks (v) Cray	9/21/90
Cray Y-MP/8, 167 MHz Fortran+tuned LAPACK solver	1	2560	16.5 G	58.27	293.	J. Brooks (v) Cray	9/21/90
Cray 2S/8-128, 244 MHz Fortran+directives, FPP 3.00Z25	8	2443	14.4 G	59.83	240.	S. Elbert Ames Lab	9/8/90
NCUBE 6400, 20 MHz Fortran+assembler	64	1438	2.83 G	59.95	47.2	J. Gustafson Ames Lab	9/17/90
MasPar MP-1, 12.5 MHz C with plural variables (mpl)	8192	1407	2.93 G	57.65	50.8	M. Carter Ames Lab	8/11/90
Silicon Graphics 4D/380S, 33 MHz Fortran (-O2 -mp -lparalin)	8	1010	1.15 G	59.85	19.2	S. Elbert Ames Lab	6/15/90
Silicon Graphics 4D/380S, 33 MHz Fortran (-O2 -mp -lparalin)	4	853	716. M	59.87	11.96	S. Elbert Ames Lab	6/15/90
Silicon Graphics 4D/380S, 33 MHz Fortran (-O2 -mp -lparalin)	2	676	378. M	59.15	6.39	S. Elbert Ames Lab	6/15/90
IBM RS/6000 POWERstation 320 Fortran (xlf -O -Q)	1	642	328. M	59.+	5.6	S. Elbert Ames Lab	5/14/90
Silicon Graphics 4D/380S, 33 MHz Fortran (-O2 -mp -lparalin)	1	531	195. M	59.16	3.30	S. Elbert Ames Lab	6/15/90
iPSC/860, 40 MHz Fortran (-OLM -i860)	1	419	105. M	59.91	1.75	J. Gustafson Ames Lab	5/17/90
Myrias SPS2 (mc68020, 16.7 MHz) Fortran (mpfc -Ofr)	64	399	92.2. M	59.26	1.56	J. Roche (v) Myrias	6/21/90
SUN 4/370, 25 MHz, C (ucc -O4 -dalign etc.)	1	380	81.1 M	59.85	1.35	M. Carter Ames Lab	7/17/90
NCUBE 2, 20 MHz Fortran + assembler subroutines (-O2)	1	354	67.5 M	59.73	1.13	J. Gustafson Ames Lab	8/13/90
Silicon Graphics 4D/20, 12.5 MHz, Fortran (f77 -O2)	1	290	40.5 M	59.70	0.679	S. Elbert Ames Lab	5/15/90
DECStation 2100, 12.5 MHz, Fortran (f77 -O2)	1	285	38.8 M	59.72	0.649	J. Gustafson Ames Lab	5/4/90
Cogent XTM (T800 Transputer) Fortran 77 (-O -u)	1	149	7.89 M	59.37	0.133	C. Vollum (v) Cogent	6/11/90
Mac IIcx, 68030, Interpreted QuickBASIC	1	24	0.142 M	59.+	0.00239	J. Gustafson Ames Lab	5/1/90

NOTE: a "(*v*)" after a name means the benchmark was run by the vendor. Vendors often have access to special tools, early compiler releases, and proprietary libraries, so remember the source. We quote MFLOPS for continuity with earlier benchmarks, but it's *patches* that determines rank. See the MasPar versus the NCUBE 6400 ranking, for example.





Acknowledgments

We thank everyone who has participated in this effort. This work was supported by the U. S. Department of Energy Applied Mathematical Sciences subprogram of the Office of Energy Research, under contract W-7405-ENG-82.

Summary

We view SLALOM as a significant step toward providing a level playing field for advanced architectures. Right now, Cray Research is the clear winner, but the highly parallel computers have yet to compete in the same price range. The largest NCUBE, Intel, and Thinking Machines systems should be worthy opponents for Cray on SLALOM.. We're committed to maintaining the scientific integrity of this benchmark, and, with your help, we look forward to measuring and publishing even more wide-ranging SLALOM numbers in the future.

References

[1] R. E. Benner, G. R. Montry, and J. L. Gustafson, “A Structural Analysis Algorithm for Massively Parallel Computers,” *Parallel Supercomputing: Methods, Algorithms, and Applications*, edited by G. F. Carey, Wiley series in parallel computing, 1989.

[2] J. J. Dongarra, “Performance of Various Computers Using Standard Linear Equations Software in a Fortran Environment,” Argonne National Laboratory, Technical Memorandum No. 23, Feb. 2, 1988.

[3] C. M. Goral, K. E. Torrance, D. P. Greenberg, and B. Battaile, “Modeling the Interaction of Light Between Diffuse Surfaces,” *Computer Graphics*, Volume 18, Number 3, July 1984.

[4] J. L. Gustafson, “Reevaluating Amdahl’s Law,” *Communications of the ACM*, Volume 31, Number 5, May 1988.

[5] L. Pointer, “PERFECT: Performance Evaluation for Cost-Effective Transformations, Report 2,” CSRD Report No. 964, March, 1990.

[6] M.W. Schmidt, J.A. Boatz, K.K. Baldridge, S. Koseki, M.S. Gordon, S.T. Elbert, and B. Lam, “General Atomic and Molecular Electronic Structure System (GAMESS),” *Quantum Chemistry Program Exchange Bulletin*, Vol. 7, No. 115, 1987.