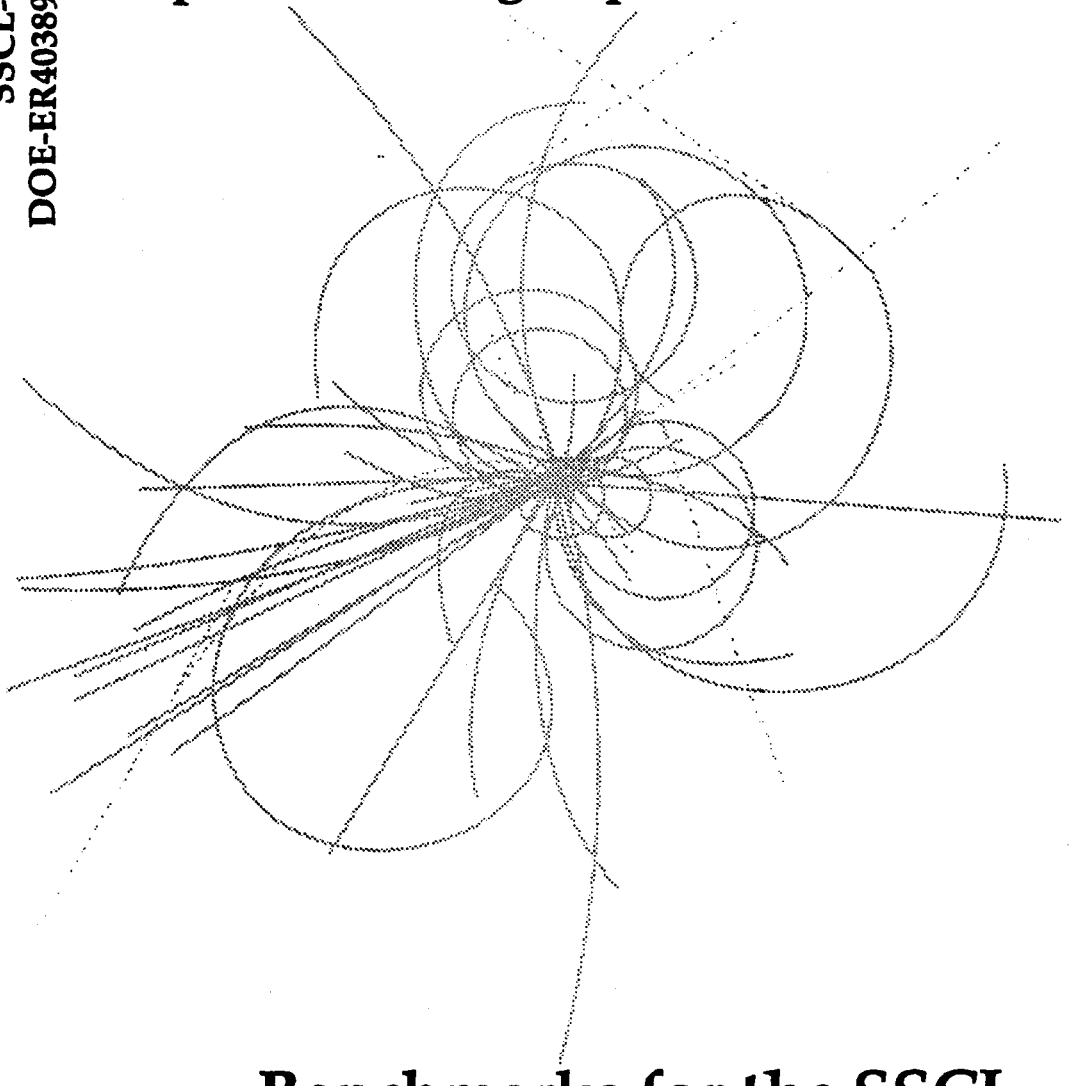
SSCL-375
DOE-ER40389-70

SSCL-375
DOE-ER40389-70

Superconducting Super Collider Laboratory



Benchmarks for the SSCL Physics Detector Simulation Computing Facility

K. McFarlane

September 1991

SSCL-375
DOE-ER40389-70

Benchmarks for the SSCL Physics Detector Simulation Computing Facility

K. McFarlane

Physics Research Division
Superconducting Super Collider Laboratory*
2550 Beckleymeade Avenue
Dallas, TX 75237

and

Physics Department
Temple University†
Philadelphia, PA 19122

September 1991

*Operated by the Universities Research Association, Inc., for the U.S. Department of Energy under Contract No. DE-AC35-89ER40486.
†Work supported in part under U.S. DOE grant DE-FG02-88ER-40389.

SSCL-375
DOE-ER40389-70

Benchmarks for the SSCL Physics Detector Simulation Computing Facility

K. McFarlane

Abstract

The benchmark suite used for initial technical evaluation of bids for the Superconducting Super Collider Laboratory (SSCL) Physics Detector Simulation Facility is described, and its results compared with standard industry benchmarks and the CERN benchmark. It is concluded that the industry benchmarks do not give an accurate guide to performance on HEP simulation programs across all architectures. The CERN and SSC benchmarks do agree. For the *RISC* architectures surveyed, the rating in SSCL units of processing power (SSCUPs) was approximately half the rating in industry-standard MIPS. For all systems, the ratio between SSCUPs and CERN units was four, with optimized programs. It is found to be clearly advantageous to allow optimization of code. The goal of 500 SSCUPs is achieved for the first phase of the facility: the facility has 678 without optimization, and with optimization of programs, 1010 units. The rating in CERN units is 232.

1.0 INTRODUCTION

This note describes an initial benchmark suite developed to assist in evaluating systems for the SSCL Physics Detector Simulation Computing Facility (PDSF), and gives results from this suite. Also, comparisons with other benchmarks and comments on different compilation options are included. The Facility is being developed along the lines recommended by the SSC Computer Planning Committee,[1] which opened the door to taking advantage of new technology by recommending an "open systems" approach.

RISC technology, combined with Unix, has dramatically reduced the cost of accessible computing power. However, to take full advantage of this power, it is helpful to be aware of two aspects:

- Neither MIPS, Mflops, nor SPECmarks are accurate guides to power on HEP programs;
- On RISC machines, optimization can produce substantial (factors of 2) gains; but HEP programs may have to be edited in detail to allow these gains.

A second purpose of the suite was to provide a comparison with the goal set for the initial phase of the Facility of 500 "MIPS" (actually DEC VAX-11/780[2] equivalents, or VUPs). To avoid confusion with the standard meaning of the term MIPS and because the DEC VAX-11/780 is not a current machine, we adopted as a unit the SSCUP (SSCL Unit of Processing power), which is intended to be close to a VUP for the types of programs being considered (the definition is given below). Therefore, we can translate the goal for the Facility to be 500 SSCUPs.

2.0 RISC AND UNIX

The existence of the Unix operating system has been instrumental in bringing new computer technologies to the marketplace quickly. All but a small part of the system (the kernel and compiler) is written in a high-level language (C), resulting in fast and straightforward ports to new hardware.

The basic concept of RISC (Reduced Instruction Set Computer) is to optimize simultaneously the VLSI chip set, the instruction set, and the high-level language

compiler, bearing in mind that off-chip accesses are costly in time. Since this results in new CPU designs, the portability of Unix has been essential to bringing RISC technology to the marketplace quickly. RISC and Unix are therefore linked in practice.

The RISC tradeoffs have led to a smaller instruction set and to the use of the chip area for a pipelined CPU, an instruction cache, and a register file. Further, all instructions are to be executed in one machine cycle, if possible. Memory accesses are restricted to LOAD/STORE and may take longer than one cycle; branches take two or three cycles. Floating-point arithmetic is usually done on a separate, pipelined coprocessor chip (FPU). Features of high-level language that are important for optimization are: branches are frequent, the CALL/RETURN mechanism is heavily used, and variables are often local to a particular module.

RISC compilers thus pay special attention to keeping the CPU and FPU pipelines full, maximizing instruction cache hits, and putting temporary variables in the register file to reduce memory LOAD/STORE cycles. Whether HEP codes can take advantage of these features can affect performance.

Obviously, the tradeoffs will change as VLSI technology changes.

3.0 BENCHMARKS

For the purchase of the SSC Physics Detector Simulation Facility, it was helpful to have a set of benchmarks that reflected existing HEP simulation codes, in addition to the standard industry benchmarks. To motivate this, it is worth looking at the commonly used standard benchmarks:

MIPS An *integer* performance benchmark based on the Dhrystone synthetic benchmark.

Mflops A floating-point benchmark based on LINPACK matrix-manipulation routines, usually *double precision* using the 100×100 version.

SPECmarks A benchmark using ten application programs from a variety of sources. Four are in C, the other six in *double precision* FORTRAN.

These benchmarks are usually optimized to attain the highest possible rating.

Because the standard HEP code is single-precision (32-bit on most systems), may not be optimized in practice, and is "large," it was felt that the above benchmarks would not be a precise-enough guide in case of close competition. The decision was made to use programs which would reflect the initial use of the Simulation Facility. One major restriction was put on the suite: the CERN library was not to be used, since this would put a large burden on the vendors. Furthermore, it was decided to use only single-CPU times.

The programs and compilation options that were selected to give reference execution times were:

Isajet The Isajet (V6.31) test job, with most printing suppressed, compiled with all variables static, and the highest level of optimization compatible with using libraries. A 10 MB unformatted output file is written.

Jetset The Jetset73 test job, compiled allowing automatic variables, both with optimization turned on and with optimization turned off.

Pythia The Pythia54 test job, compiled with automatic variables, also with and without optimization.

The choice of these options reflects standard use for these programs: Isajet V6.31 has not been edited to allow automatic variables, and the authors of Pythia and Jetset recommend turning off optimization on the basis of previous experience. In practice, on most RISC systems tested, Isajet does not give "correct" results unless the static option is used, while Pythia and Jetset do give correct results on the test job when optimized. Optimization and debugging are mutually exclusive in some systems, and initially it is expected that users will use debugging. Therefore, the above combination, with no optimization of Pythia and Jetset, was felt to be a fair basis for evaluating bids, though higher performance can in fact be obtained.

Timing was done in two ways: with the Unix `/bin/time` command, and with a C routine `getime`, which could give the same information. The two times were required to agree where appropriate. The time used for comparisons was the sum of the user CPU time and the system CPU time (on behalf of the user).

Besides the above options, other combinations were used to assess the effect of optimization and debugging on program compilation time, size, and execution time. We also measured performance with other programs: the 300×300 LINPACK benchmark, a program to measure FORTRAN routine-calling performance, one to check byte-ordering compatibility, and one to assess FORTRAN/C mixed language programming issues.

Validation of the results was simple: if the programs themselves reported no errors, and the results seemed "reasonable," the run was accepted. Due to variations in compilers, floating-point units, and optimization strategies, there was variation in results. So even though all RISC systems tested adhered to the IEEE floating-point standard, the results were not the same on different machines, and sometimes not the same for optimized and non-optimized runs.

4.0 BENCHMARK RESULTS

4.1 Units (SSCUPs)

For a standard of computing power, we chose the DEC VAX 6420 at SSCL (SSCVX1). As noted above, the single-process CPU times (i.e., the execution times of a single process running on a single CPU) were used. We took the power of a single 6420 CPU to be 7.0 SSCUPs for the particular situation. Note that the execution times varied by as much as 25% depending on the loading of SSCVX1, so that the *shortest* time was taken. The times are listed in Table 1.

Table 1. Reference Execution Times on SSCVX1.

Program	Compilation option	CPU Time (s)	SSCUPs
Isabench	/NOOPT	246.	–
Isabench	/OPT	192.	7.0
Jetbench	/NOOPT	68.1	7.0
Jetbench	/OPT	56.7	–
Pybench	/NOOPT	88.3	7.0
Pybench	/OPT	65.7	–

To compare machines, we used a combined rating, being the geometric mean of the rating in SSCUPs, as normalized to SSCVX1, of the three selected programs,

compiled using the standard options listed above. This rating we called the overall processing power in *unoptimized* SSCUPs. If improvements can be obtained by turning on optimization for Pythia and Jetset while retaining valid results, that will give a rating in *optimized* SSCUPs.

4.2 Comparisons

In this section we compare the processing power of various systems that either were considered for the SSCL simulation facility or are of interest. The systems discussed here are from DEC, SGI,[3] IBM,[4] HP-Apollo,[5] and Sun.[6] The details of the systems and compilation options used are listed in Appendix A.

The ratings of the systems on various benchmarks are listed in Table 2.

Table 2. System Ratings on Various Benchmarks.

System	MIPS	DP Mflops	SPECmarks	un-opt. SSCUPs	opt. SSCUPs
DEC VAX-11/780	1	0.14	1.0	1.1	1.33
DEC VAX 6420 (1 CPU)	7.5	1.18	7	7.0	8.2
SGI 4D/380 (1 CPU)	30	5	18.5	13.5	21.6
SGI 4D/35S	33	6	23.0	15.3	–
IBM RS6000/530	33	11.3	28.6	14.3	21.3
IBM RS6000/520	29.5	8.5	22.0	11.9	–
Sun Sparc 2	28.5	4.2	21.0	11.8	16.3
DECstation 3100	14	1.6	10.8	6.4	10.0
DECstation 5000	24	3.7	18.5	10.5	–
Apollo DN10000 (1 CPU)	22	6	18.6	11.6	17.4
HP Apollo 9000/720	57	17	55.5	22.7	29.4
HP Apollo 9000/730	76	22	72.2	–	–

These ratings indicate that the performance on physics codes, given the restrictions, is not accurately predicted by the other quantities across the architectures, as discussed in Section 4.2.2.

4.2.1 The CERN Benchmarks. We have run the CERN benchmark suite[7] on two target systems to compare the rating in SSCUPs with that in CERN units. The CERN unit is based on the geometric mean of four programs, using the performance on a DEC VAX 8600 as one unit of performance. This machine is rated at about the

equivalent of 4 VAX-11/780's. Thus we should be able to make a direct comparison between SSCUPs and CERN units, expecting a ratio of about 4:1. The four programs are CRN3, an old FOWL Monte Carlo program; CRN4, a program based on the LUND Monte Carlo code; and CRN5 and CRN12, old event-processing programs for magnetic spectrometers using input data files.

In comparing with the CERN benchmarks, we note that they are edited to allow full optimization, and they are smaller than the SSCL benchmarks: on one machine the executables range from 0.3 MB to 0.5 MB, compared with 0.8 MB to 1.3 MB for the SSCL benchmarks.

The ratings of various systems on the two benchmarks are listed in Table 3.

Table 3. System Ratings on the CERN and SSC Benchmarks.

System	std. SSCUPs	opt. SSCUPs	CERN units	Ratio
DEC VAX6420 (1 CPU)	7.0	8.2	1.9 est.	4.3 est.
Apollo DN10000 (1 CPU)	11.6	17.4	(4.9)	3.6
DECstation 3100	6.4	10.0	(2.8)	3.6
DECstation 5000	10.5	–	(4.4)	–
SGI 4D/380 (1 CPU)	13.5	21.6	5.3	4.1
HP Apollo 9000/720	22.7	29.4	(10.5)	2.8
IBM RS6000/530	14.3	21.3	(5.9/4.9)	3.6/4.3
Sun Sparc 2	11.8	16.3	3.5	4.7

The numbers in parentheses come from the listing in Reference 7, while the others were measured at the SSCL. The two listings for the IBM RS6000/530 reflect different compilation options for the CERN benchmark (not optimization options). From the above table, it is clear that there is reasonable agreement between the two ratings.

4.2.2 Conclusions. Figures 1 through 4 compare the various ratings, showing that the standard industry ratings do not give an accurate guide to performance for HEP simulation code. The correspondence between MIPS and unoptimized SSCUPs is reasonably good, with a mean ratio of 0.45 SSCUPs/MIPS for RISC architectures (0.9 for the VAX CISC system). As expected, there is also good correspondence between optimized SSCUPs and CERN units, with a mean ratio of 4.1 optimized SSCUPs/CERN unit.

4.3 Optimization and Debugging

Optimization can have large effects, but debugging does not much affect times of execution. We give details first for Pythia, Jetset, and Isajet on the SGI 4D/380, which are representative for RISC machines are listed in Table 4.

Table 4. Effect of Compilation Options on SGI 4D 380.

Program	options	Compile time (s)	file size (kBytes)	time (s)	SSCUPs
Pythia	-O0	42	1508	57.4	10.8
	-O0 -g	43	1604	57.4	10.8
	-O1	56	1308	34.2	18.1
	-O2 -Olimit 2000	826	1185	24.1	25.7
	-O2 -Olimit 2000 -g3	828	1281	24.3	25.4
	-O3 -Olimit 2000	1243	1125	24.8	24.9
	-O2 -Olimit 2000 -static	837	1281	24.8	24.9
Jetset	-O0	50	973	32.2	14.8
	-O0 -g	51	1030	32.2	14.8
	-O1	69	860	26.8	17.8
	-O2 -Olimit 2000	511	810	18.8	25.4
	-O2 -Olimit 2000 -g3	512	867	18.9	25.3
	-O3 -Olimit 2000	466	794	18.7	25.1
	-O2 -Olimit 2000 -static	520	867	19.1	24.9
Isajet	-O0 -static	104	1245	120.8	14.3
	-O2 -Olimit 2000 -static	524	1011	88.1	15.3
	-O2 -Olimit 2000 -static -g3	516	1138	88.9	15.2

The option -O2 is the default optimization for the MIPS compiler, and -O3 is a global optimization which can be done only on a complete program. Since the benchmark version of Isajet is built from a library, -O3 cannot be applied to it. The -g switch turns on debugging; some limited debugging is possible with optimization using the -g3 switch. (For example, dummy arguments of subroutines may not be accessible.) As can be seen, going to the "highest" level of optimization (O3) does not improve speed for Pythia or Jetset, and optimization increases compilation time substantially. Using the -static option does not change the running time for Pythia or Jetset, likely because many variables are in common blocks.

The IBM listing is shown in Table 5.

Table 5. Effect of Compilation Options on IBM RS 6000.

Program	options	Compile time (s)	file size (kBytes)	time (s)	SSCUPs
Pythia	–	64.1	885	50.9	12.1
	-g	64.1	1120	50.2	12.3
	-O	435.	651	25.3	24.4
Jetset	–	65.3	515	33.6	14.2
	-g	65.6	632	33.4	14.3
	-O	413.	398	20.7	23.0
Isajet	–	108.	811	119.4	14.4
	-g	107.	1360	119.2	14.5
	-O	392.	633	78.6	17.1

Note that the IBM compiler xlf has one level of optimization and does not permit debugging with optimizing.

4.4 Other Tests

Other tests were run to evaluate special aspects of the systems, as described below.

4.4.1 Double vs. Single Precision. In the computer industry, there is a tendency to equate scientific FORTRAN programming with 64-bit precision; the default floating-point precision for C is 64-bit. This is implemented in most current systems with 64-bit floating-point, and 32-bit integers. However, HEP code currently emphasizes 32-bit reals and integers; the ZEBRA package requires that integers and reals be the same length. One may expect some evolution in this area, as demand for accuracy increases in the HEP community.

For most RISC-Unix systems, FORTRAN single precision means 32-bits reals and integers, and double precision means 64-bit reals. Most systems also support 64-bit floating-point hardware, but there are differences in detail. The most striking one is that the IBM RS6000 series is optimized for 64-bit precision, while allowing 32-bit operations in the 64-bit FPU. The IBM RS6000 FPU supports the operation $A \times B + C$, so matrix operations optimize very well.

To give an independent evaluation of floating point operations for various options, we used two pairs of programs:

LINPACK This pair consisted of 32-bit and 64-bit versions of the 300 × 300 LINPACK benchmarks. It is intensive in arithmetic operations and indexing.

foura This pair is a synthetic benchmark which does a Fourier transform (of a Gaussian wavepacket), using FORTRAN sine and cosine functions. It is a 1024-point discrete transform, but is *not* an FFT. It is intentionally written to consume CPU power.

For both these benchmarks, the rating is taken relative to the optimized VAX times.

Tables 6 and 7 provide the LINPACK performances for various systems.

Table 6. LINPACK Performances.

System	compiler option	Mflops SP	Mflops DP	double/single time ratio	SP SSCUPs
IBM RS6000/530	–	2.2	2.3	0.97	10.1
IBM RS6000/530	-O	7.3	11.3	0.65	33.6
SGI 4D/380	-O0	2.6	1.8	1.46	12.1
SGI 4D/380	-O2	5.3	2.9	1.83	24.6
DEC3100	-O0	1.1	0.73	1.53	5.13
DEC3100	-O2	2.4	1.2	2.02	10.9

The IBM system has an impressive performance in Mflops; however, it does not translate proportionately into performance on HEP programs.

The `foura` performance is listed in Table 7.

Table 7. FOURA Performance.

System	compiler option	SP time (s)	DP time (s)	double/single time ratio	SP SSCUPs
IBM RS6000/530	–	36.6	32.3	0.89	20.3
IBM RS6000/530	-O	33.2	26.8	0.81	22.32
SGI 4D/380	-O0	31.0	41.9	1.35	23.9
SGI 4D/380	-O2	26.3	35.8	1.37	28.3
DEC3100	-O0	63.0	85.1	1.35	11.7
DEC3100	-O2	56.1	73.2	1.31	13.2

Because of the intensive use in `foura` of FORTRAN functions (it spends 25% of the CPU time in the sine routine), the times do not improve much with optimization. The speed for single precision is faster than for double, with the exception of the IBM system, where it is slower.

4.4.2 File and Mixed-language Compatibility. The Facility specifications included the requirement that FORTRAN unformatted sequential and direct access files be Big-Endian in structure (for character, 32-bit and 64-bit data items), so the "fio" test was included. The "fio" group of tests also included an evaluation of FORTRAN/C compatibility issues, since many routines interfacing with the system need to be written in C.

5.0 FORTRAN FOR RISC-UNIX

Early RISC systems were optimized on C and Pascal programs, but F77 does have features that can be used by RISC optimizers. The major example is the ability to support *automatic* variables. These are variables which are used only as temporary working variables, in subprograms for example, in contrast to *static* variables which are assigned permanent memory locations. Automatic variables are allocated storage, in registers or stacks, when the subprogram is invoked, and the the storage is de-allocated when leaving the routine. This is a *permissible* interpretation of the F77 standard, but it is not required by the standard.

Some Unix F77 compilers will make variables automatic by default (this is true also of non-RISC Unix), and will assign them to registers or stacks. This default can be overridden (for a variable, a common block, or all variables in the module) by using the F77 SAVE statement, a DATA statement in the subprogram, or a compiler switch which makes every variable in the routine static (i.e., having a permanent memory location). Note that, unlike C and Pascal, F77 may not make COMMON blocks static by default; an explicit SAVE statement may be needed in every subprogram where that common block is declared. If a program mostly manipulates COMMON blocks, there may not be much improvement in allowing automatic assignment.

Other Unix FORTRAN compilers (e.g., Sun and IBM) have a different mechanism. All variables are static by default, but, as an extension to FORTRAN, AUTOMATIC

and STATIC declarations are provided to allow the programmer to set the treatment of variables. These compilers basically ignore the SAVE statements and have no compiler switch. So programmers wishing to take advantage of automatic variables would need to edit each routine.

Much HEP code depends on assumptions about storage mechanisms. For example, subroutines frequently are written assuming that they have a history: they "know" if they have been called previously. Also, some code may be sufficiently obscure that an optimizing compiler may move statements outside a loop that should not be moved. Taking full advantage of RISC technology will require careful rewriting of code. Examples of changes that will be helpful are:

Automatic variables Write the program to allow as many as possible variables to be "automatic"; make explicit use of the SAVE statement (or the AUTOMATIC statement); do not use array or COMMON block quantities in intermediate calculations.

Optimization Do not use hardware- or compiler-dependent tricks; write loops in such a way that standard optimizing steps leave the code correct; use arguments, rather than common blocks, to pass data

6.0 CONCLUSION

The Simulation Facility is divided into two parts: a back-end compute server, called the Batch Ranch, consisting of three SGI 4D/380 systems; and an front-end interactive compute server, consisting of 30 Sun Sparcstation 2 systems. The Suns are located in a computer room, without displays and keyboards; it is intended that they be accessed over the network. All systems share a file server.

Using standard SSCUPs, the Batch Ranch will have a performance of 24 times 13.5, or 324, with the front-end group having 354, for a total of 678. Using optimized SSCUPs, the total is 1010, in either case exceeding the goal. Multiprocessing and network loading are expected to reduce the overall numbers, but likely less than 20%. The rating in CERN units is a total of 232.

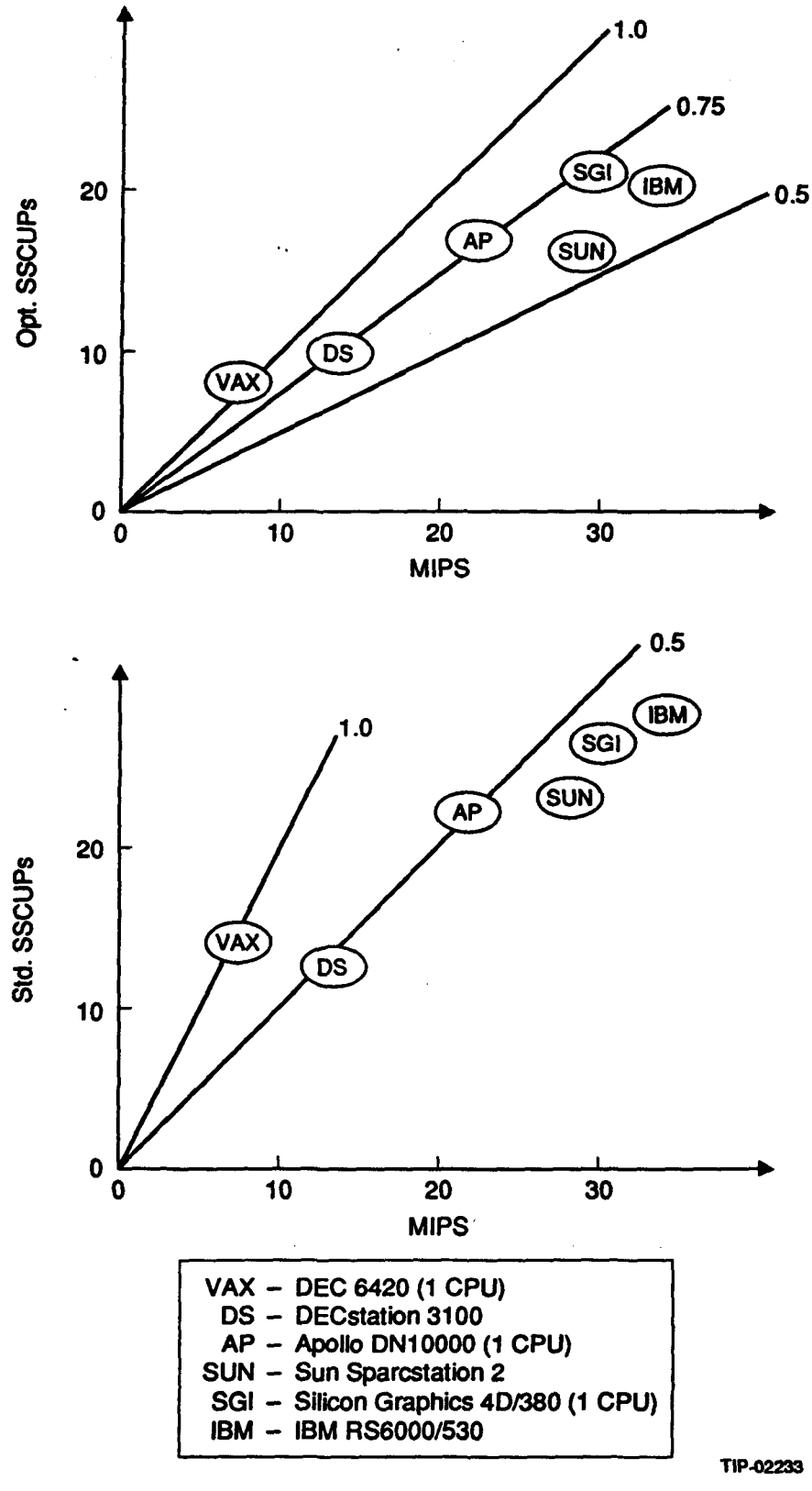


Figure 1. Rating in std. (a) and opt. (b) SSCUPs for several systems, as a function of MIPS. Systems are identified in the key. Lines of labelled slope are to guide the eye.

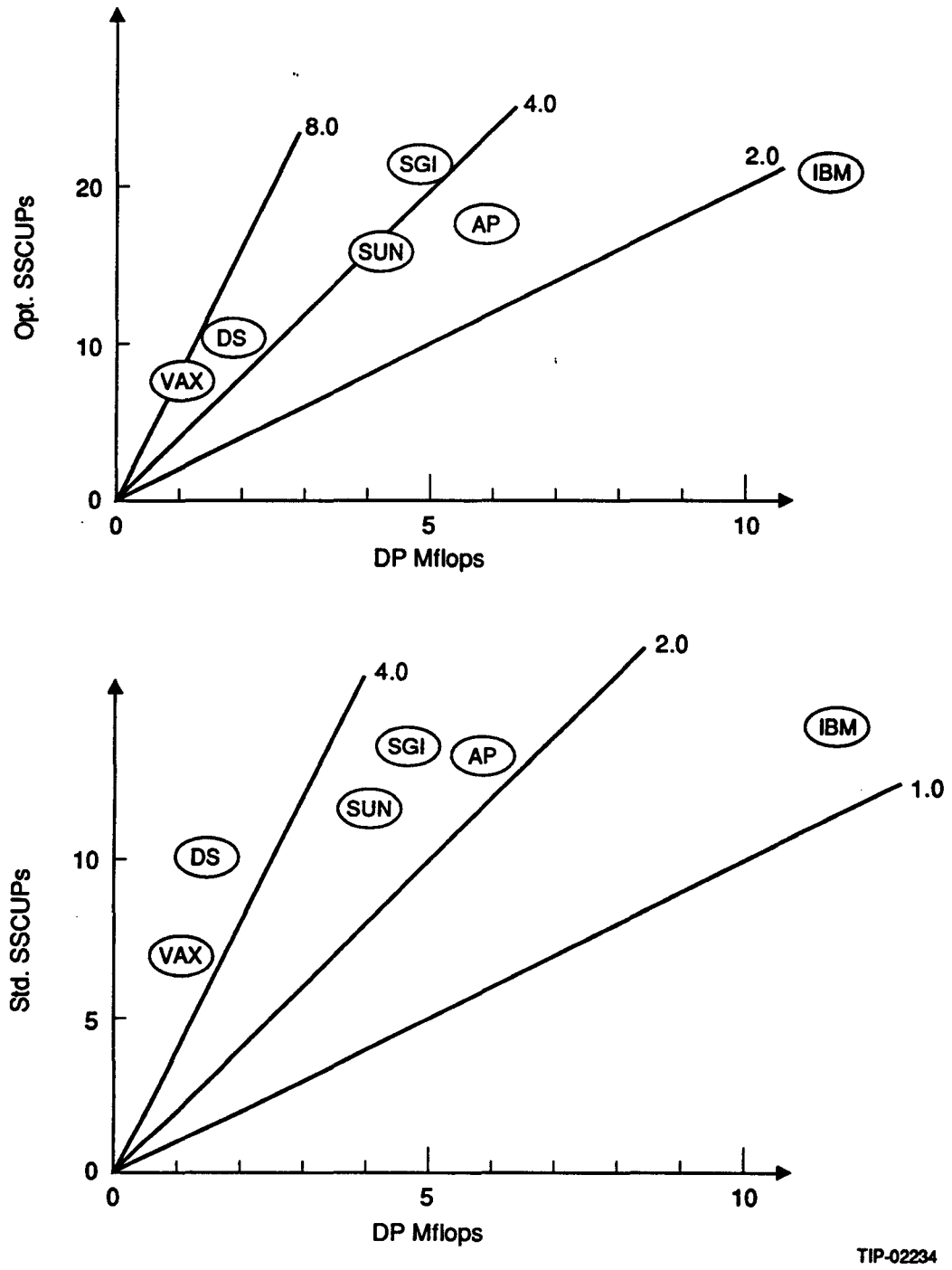


Figure 2. Rating in std. (a) and opt. (b) SSCUPs for several systems, as a function of Mflops.

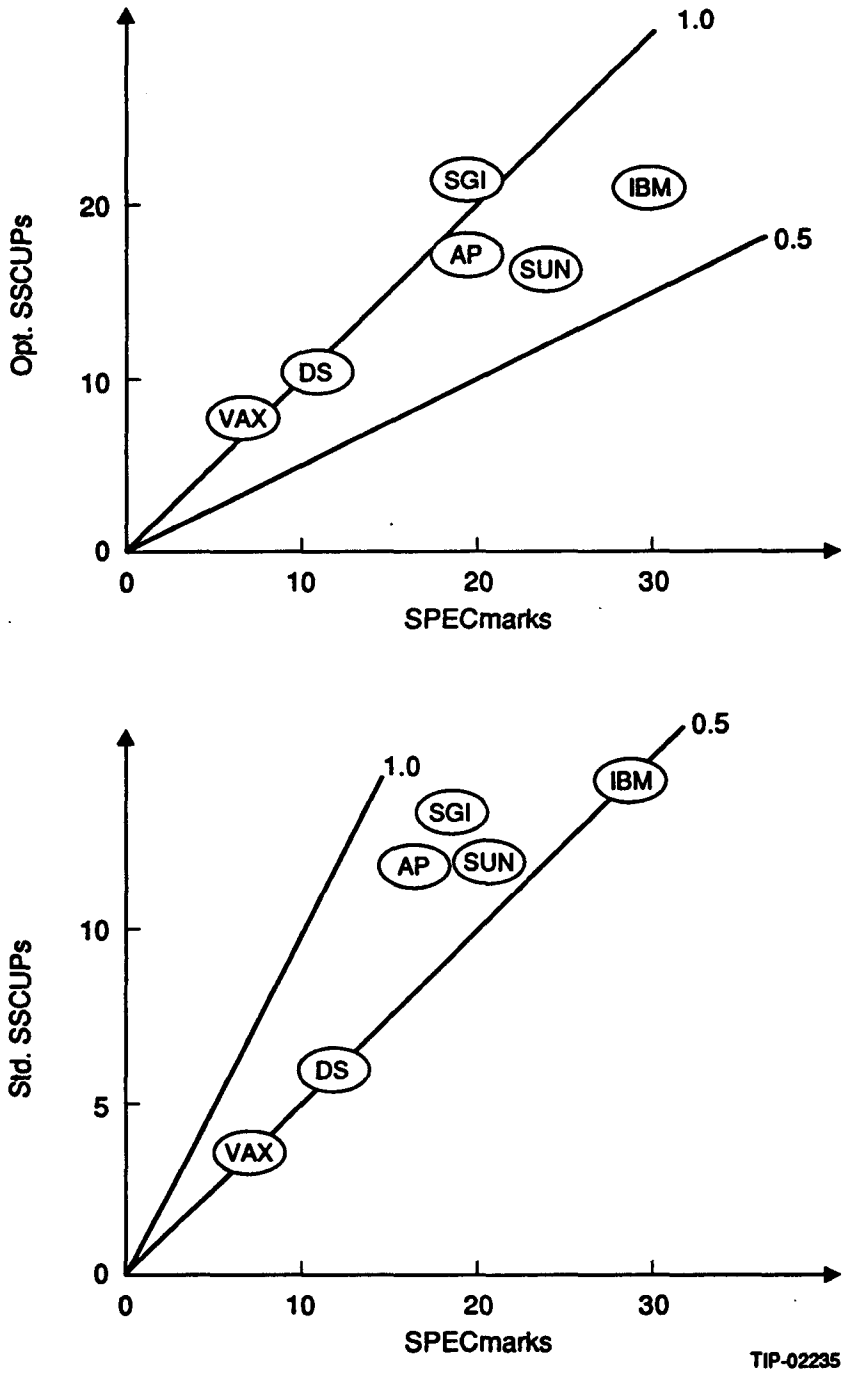


Figure 3. Rating in std. (a) and opt. (b) SSCUPs for several systems, as a function of SPECmarks.

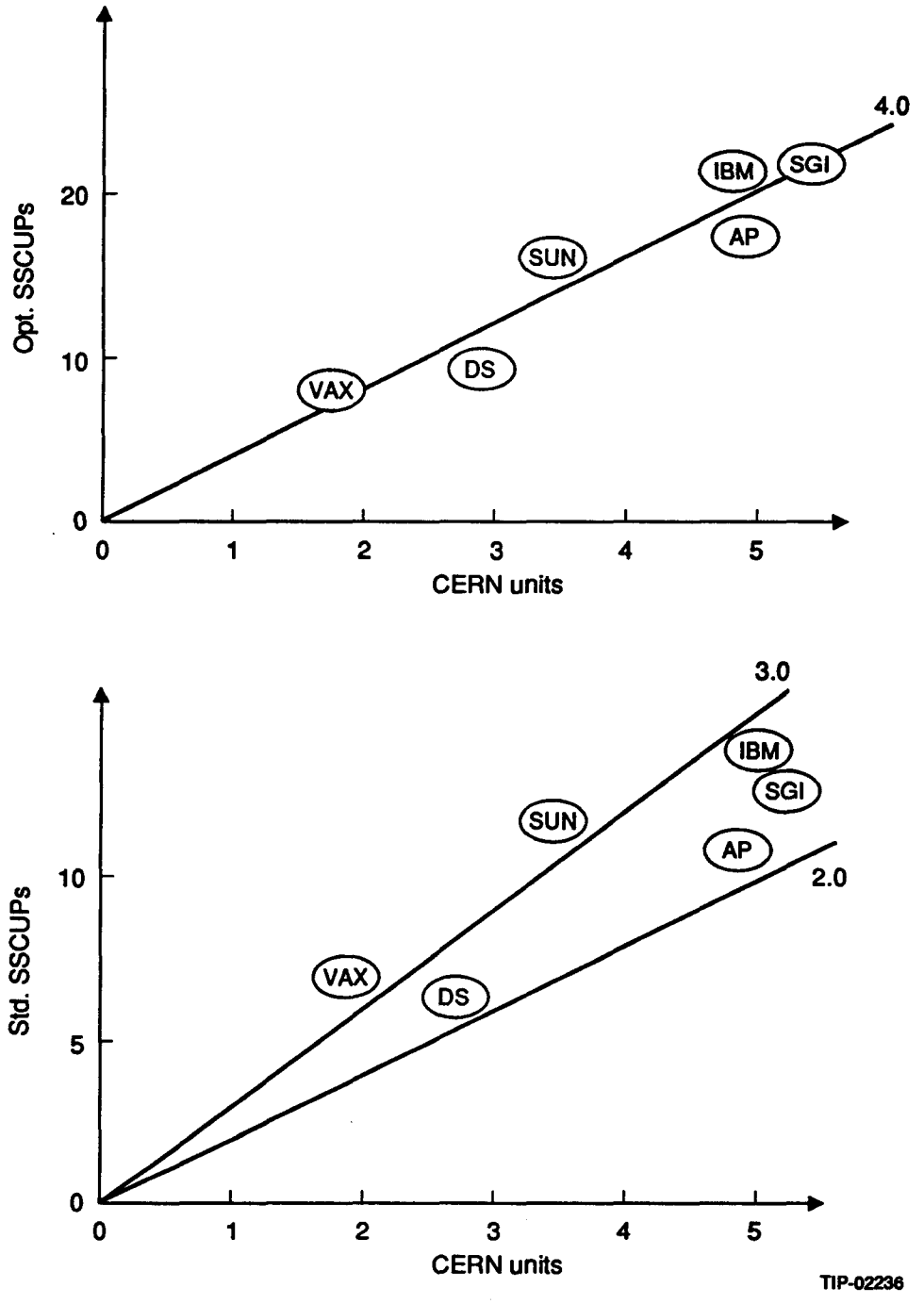


Figure 4. Rating in std. (a) and opt. (b) SSCUPs for several systems, as a function of CERN units.

ACKNOWLEDGEMENTS

Setting up and running these benchmarks involved the cooperation of many people. Particularly, thanks are due to Lee Roberts (SSCL) for much advice and to Les Robertson and Eric McIntosh (CERN) for providing the CERN benchmark. Brian Scipioni, Laird Cormell, and George Yost of the Computer Acquisition Working Group provided valuable support.

REFERENCES

1. Report of SSC Computer Planning Committee, L. Price, ed., SSCL-N-691, Jan. 1990.
2. Digital Equipment Corporation
3. Silicon Graphics Inc.
4. International Business Machines
5. HP-Apollo Inc.
6. Sun Microsystems Inc.
7. Eric McIntosh, CERN, private communication, Feb. 1990.

APPENDIX A. NOTES AND DETAILS OF SYSTEMS EVALUATED

The hardware, operating system, and compiler are listed for the various systems in this section, as are the compiler options used. All systems had at least 16 MB of memory.

Generally, the use of 64-bit hardware with 32-bit reals may mean that floating-point exceptions may be treated in non-standard ways, especially if optimization is used. If it is important to know about such exceptions, some compiler option may be needed. That issue has not been addressed for the benchmarks.

A.1 DEC VAX-6420

The DEC VAX-6420 is a symmetric multiprocessor machine with 2 CPUs; only single-CPU performance was evaluated.

System	DEC VAX-6420 running VMS V5.3-1
Compiler	DEC FORTRAN V5.4
Non-optimizing options	/NOOPT/NOCHECK/NODEBUG
Static option	Not an option
Optimizing options	/OPT/NOCHECK/NODEBUG

The DEC VAX was used only to normalize the benchmarks.

A.2 SGI 4D/380

The SGI 4D/380 is a symmetric multiprocessor machine with 8 CPUs.

System	SGI 4D/380 running IRIX release 3.3.2.
Compiler	f77, MIPS release 2.0
Non-optimizing options	-O0
No optimization with debugging	-O0 -g
Static option	-static
Optimizing options	-O2 -Olimit 2000
Optimization with debugging	-O2 -Olimit 2000 -g3

On the Silicon Graphics systems with the MIPS compiler, variables are automatic by default. The -Olimit option increases compiler table sizes for optimization. The -g3 option allows limited debugging with optimization.

A.3 SGI 4D/35S

The SGI 4D/35S is a single-CPU workstation (Personal Iris).

System	SGI 4D/35S running IRIX release 3.3.2.
Compiler	f77, MIPS release 2.0
Non-optimizing options	-O0
No optimization with debugging	-O0 -g
Static option	-static
Optimizing options	-O (same as -O2)
Optimization with debugging	-O -g3

On the Silicon Graphics systems with the MIPS compiler, variables are automatic by default. The -g3 option allows limited debugging with optimization.

A.4 IBM RS6000/530

System	IBM RS6000/530 running AIX release 3.1
Compiler	xlf
Non-optimizing options	-qextname -qrndsngl -qxflag=dd24 -NQ40000
No optimization with debugging	-g
Static option	no option
Optimizing options	-O -qextname -qrndsngl -qxflag=dd24 -NQ40000
Optimization with debugging	not recommended

The various options are described in Table A-1.

Table A-1. Options.

Option	Description
-qextname	appends underscore to the names of FORTRAN routines; needed for compatibility
-qrndsngl	forces all 32-bit real results to be rounded to single precision; needed to give consistent results
-qxflag=dd24	required to fix a bug when using the above option
-NQ40000	increases compiler table size; -NT40000 may also be useful
-g	generates debugging information
-O	turns on optimization

On the IBM, xlf makes all FORTRAN variables static. There is no compiler option to allow automatic variables; the AUTOMATIC and STATIC (and IMPLICIT AUTOMATIC) declarations are available to allow programmers to tailor code.

A.5 Sun Sparc 2

System	Sun Sparcstation 2 running SunOS 4.1.1
Compiler	F77 1.3.1
Non-optimizing options	default
No optimization with debugging	-g
Static option	no option
Optimizing options	-cg89 -dalign -fnonstd -O2
Optimization with debugging	not allowed

On the Sun, F77 V1.3.1 makes all FORTRAN variables static. There is no compiler option to allow automatic variables; the AUTOMATIC and STATIC (and IMPLICIT AUTOMATIC) declarations are available to allow programmers to tailor code. The optimization options tailor code to specific hardware, and also result in non-standard handling of floating-point exceptions.

A.6 DECstation 3100

System	DECstation 3100 running Ultrix 4.1
Compiler	f77, MIPS release 2.1
Non-optimizing options	-O0
No optimization with debugging	-O0 -g
Static option	-static
Optimizing options	-O2 -Olimit 2000
Optimization with debugging	-O2 -Olimit 2000 -g3

On the DECstation with the MIPS compiler, variables are automatic by default. The -Olimit option increases compiler table sizes for optimization.

A.7 DECstation 5000

This was a DECstation 5000-200 run with options similar to the DECstation 3100, above.

A.8 HP/Apollo DN10000

The Apollo used had two CPUs (four are possible); only single-CPU programs were run for evaluations.

System	HP/Apollo DN10k running DOMAIN/os 10.3 (BSD)
Compiler	f77, PRISM release 10.7
Non-optimizing options	-W0,-opt,0
No optimization with debugging	-W0,-opt,0 -g
Static option	-W0,-save
Optimizing options	-W0,-opt,4
Optimization with debugging	

On the Apollo, variables are automatic by default.

APPENDIX B. WHY DON'T THE BENCHMARKS AGREE?

Even though all but one of the benchmarks discussed here are explicitly standardized to the VAX architecture (and the two industry ones, MIPS and SPECmarks, to the venerable VAX-11/780), there is no precise scaling between one benchmark and another. Reasons for differences in relative performance include program type, choice of compilation options, program size, floating point unit, cache size, etc. For example, one of the SPECmark programs (*matrix300*) fits entirely in the HP Apollo 700 series cache and gives a rating of about 250 on that program, pulling the mean rating up about 20%. One or more of the CERN benchmarks may also fit entirely in the cache.

Other reasons are less "physical": the vendor may not use standard benchmarks. For MIPS, this can include the quoting of "native" MIPS (million instructions per second)—RISC machines can often do up to *five* instructions per clock cycle—instead of Dhrystone MIPS. For the LINPACK benchmarks, there are a variety of special optimization techniques. In the case of SPECmarks, it is permissible to use unreleased compilers and undocumented compiler options. In fact, at least two of the vendors for the SSCL PDSF used one or more of these techniques on the Initial Physics Benchmark Suite. Thus the best defense is to use one's own benchmarks, run under controlled conditions.

Here we describe the various benchmarks in a little more detail, which shows obvious differences, then we list some hardware information. First we note that the SSCL initial benchmark programs are moderate-sized (0.8–1.3 Megabyte executables) programs, in single (32-bit) precision FORTRAN, which use a lot of floating-point operations and branches. They are run with a mix of optimization and no optimization. (VAX6420 = 7.0)

MIPS This is an *integer* performance benchmark based on the Dhrystone synthetic benchmark. The ratio between integer and floating-point performance is architecture-dependent. (VAX-11/780 = 1.0)

Mflops This is a floating-point benchmark based on LINPACK matrix-manipulation routines, usually *double precision* using 100×100 version. The executable is

small and is subject to special optimization techniques. Further, it is double precision (64-bit) compared with our mostly single-precision programs.

SPECmarks A benchmark using ten application programs from a variety of sources. Four are in C, the other six in *double precision* FORTRAN. Hence the manufacturer's decision on how to implement single versus double precision will be important. (VAX-11/780 = 1.0)

CERN units The CERN benchmarks are edited to allow full optimization, and they are smaller than the SSCL benchmarks: on one machine the executables range from 0.3 MB to 0.5 MB. (VAX8600 = 1.0)

These benchmarks are usually optimized to attain the highest possible rating.

To assess the effect of choosing double over single precision, we used two pairs of programs: the 300 × 300 LINPACK benchmark, and "foura," a synthetic benchmark which does a Fourier transform (of a Gaussian wavepacket), using FORTRAN sine and cosine functions. It is *not* an FFT. It is intentionally written to consume CPU power. Table B-1 gives the double to single precision execution time ratios for the two programs for various systems, and the double optimized to single non-optimized ratios. The relevance of this last ratio is that the SPEC and Mflops benchmarks are DP optimized, while the standard SSCUP benchmark is closer to SP non-optimized.

Table B-1. Ratios of Execution Times for Linpack and Foura.

System	compiler option	Linpack DP/SP	foura DP/SP	Linpack DPopt/SPnon	foura DPopt/SPnon
DEC3100	-O0	1.53	1.35	0.92	1.16
DEC3100	-O2	2.02	1.37	–	–
HP Apollo 720	–	1.14	1.14	0.35	1.22
HP Apollo 720	-O	1.61	1.10	–	–
IBM RS6000/530	–	0.97	0.89	0.19	0.73
IBM RS6000/530	-O	0.65	0.81	–	–
SGI 4D/380	-O0	1.46	1.35	0.90	1.36
SGI 4D/380	-O2	1.83	1.37	–	–

As can be seen, the relative times are quite architecture- and optimization-dependent.

APPENDIX C. RATING OF DEC VAX-11/780

Following a query about the relation between SSCUPs and VAX-11/780 units of performance, the SSCL benchmarks were run on such a machine, courtesy of the Computer Science Department at Temple University.

The specific configuration was a VAX-11/780 with "F" floating-point unit, 32 MB of main memory, and RM05 disks (with plenty of free space), running VAX VMS 5.2 and FORTRAN version 5.1. The "Initial Physics Benchmark" programs on this machine were run at a lightly-loaded time, using FORTRAN options /NOCHECK /NODEBUG (besides the optimization options shown), Table C-1.

Table C-1. Options.

Program	FORTRAN option	Reference time (s)	Reported time (s)	Rating SSCUPs
Isajet	OPT	7.*192.	1232.4	1.09
Pythia	NOOPT	7.*88.3	555.5	1.11
Pythia	OPT	-	385.1	1.61
Jetset	NOOPT	7.*68.1	437.6	1.09
Jetset	OPT	-	358.3	1.33

The performance on the LINPACK 300 $\times$ 300 double precision (64-bit) benchmark was 0.145 Mflops.

The final rating of a VAX-11/780 is 1.10 unoptimized SSCUPs (or alternatively a SSCUP is 0.91 VUPs) and 1.33 optimized SSCUPs. (Remember that benchmarking is not a precise science—the specific configuration and loading can affect performance at the 10% level, though I have made an effort to run the benchmarks on all machines under lightly-loaded conditions so that the ratings in SSCUPs are internally comparable at a more precise level.) From the above data, the raw rating of the PDSF, as it stands and using the (standard) options above, is 620 VUPs.

APPENDIX D. CACHE SIZES, ETC.

Table D-1 lists clock frequencies and cache sizes for various systems. The cache to main memory data path width is also listed, if known; some machines fetch data in "lines" of up to 128 Bytes. Note that caches can be for instructions, data, or both. The data come from various sources.

Table D-1. Clock Frequencies, Cache Sizes, and Cache-Memory Path Width for Various Machines.

System(s)	Clock frequency (MHz)	Common cache (kB)	I Cache (kB)	D Cache (kB)	Path width (bits)
DEC 5000/200	25	64	–	–	?
IBM RS6000/320,520	20	–	8	32	64
IBM RS6000/530,730,930	25	–	8	64	128
IBM RS6000/540	30	–	8	64	128
HP Apollo 720	50	–	128	256	?
HP Apollo 730	66	–	128	256	?
HP Apollo 750	66	–	256	256	?
SGI 4D/380	33	–	64/cpu	320/cpu	?
SGI 4D/25S	20	–	64	32	?
SGI 4D/35	35	–	64	64	?
Sun Sparc 2	40	64	–	–	32

APPENDIX E

CERN RESULTS FOR VARIOUS MACHINES (FROM CERN)

Table E-1.

Computer	CERN units
Cray YMP 6ns	11
HP9000/720	10.5
DEC9000	9.1
IBM3090/400E	6.6
SGI 4D/310	6.2
IBM RS6000/530	5.9
HP/Apollo 10000	4.9
IBM RS6000/320	4.7
DEC 5000/320	4.4
SGI 4D/25	3.5
DEC 3100	2.8