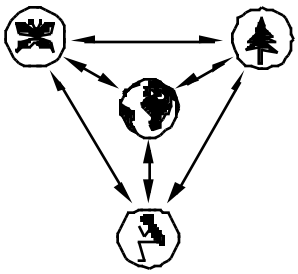

SYSTEMÖKOLOGIE ETHZ

SYSTEMS ECOLOGY ETHZ

Bericht / Report Nr. 24

# Benchmark Experiments on Workstations

T.J. Löffler, A. Fischlin, M. Ulrich

November 1996

**Eidgenössische Technische Hochschule Zürich ETHZ**
**Swiss Federal Institute of Technology Zurich**

Departement für Umweltnaturwissenschaften / Department of Environmental Sciences
Institut für Terrestrische Ökologie / Institute of Terrestrial Ecology

The System Ecology Reports consist of preprints and technical reports. Preprints are articles, which have been submitted to scientific journals and are hereby made available to interested readers before actual publication. The technical reports allow for an exhaustive documentation of important research and development results.

Die Berichte der Systemökologie sind entweder Vorabdrucke oder technische Berichte. Die Vorabdrucke sind Artikel, welche bei einer wissenschaftlichen Zeitschrift zur Publikation eingereicht worden sind; zu einem möglichst frühen Zeitpunkt sollen damit diese Arbeiten interessierten LeserInnen besser zugänglich gemacht werden. Die technischen Berichte dokumentieren erschöpfend Forschungs- und Entwicklungsresultate von allgemeinem Interesse.

Adresse des Autors / Addresse of the author:

T.J. Löffler/ A. Fischlin/ M. Ulrich
Systems Ecology
Institute of Terrestrial Ecology
Department of Environmental Sciences
Swiss Federal Institute of Technology ETHZ
Grabenstrasse 3
CH-8952 Schlieren/Zürich
Switzerland

e-mail: loeffler@ito.umnw.ethz.ch

# Benchmark Experiments on Workstations

T. J. Löffler‡, A. Fischlin‡, M. Ulrich†

Systems Ecology, Institute of Terrestrial Ecology, Swiss Federal Institute of Technology

**CONTENTS**



## 1. Introduction

This paper deals with classical benchmark tests, which are one important part for the performance evaluation of computers as well as of complex software.

Benchmark results can be used to test either specific functionality of compilers, e.g. costs of integer and real multiplication, the general performance of the combination machine and operating system as well as the performance of applications within a simulation environment as e.g. RAMSES[1] (Fischlin *et al.*, 1988; Keller, 1989; Fischlin, 1991; Fischlin *et al.*, 1994).

Here, we focus on testing the overall performance of the machines with benchmark programs. Our aim is to get information about the run-time behaviour of different platforms (computers, operating systems, and emulations) and to compare the results with the per-

‡ Systems Ecology, Institute of Terrestrial Ecology, Department of Environmental Sciences, Swiss Federal Institute of Technology ETHZ, Grabenstr. 3, CH-8952 *Schlieren / Zürich*, SWITZERLAND.

† Swiss Federal Institute for Environmental Science and Technology (EAWAG), CH-8600 *Duebendorf*, Switzerland

[1] RAMSES is an acronym for Research Aids for the Modelling and Simulation of Environmental Systems.

formance examinations of the simulation software RAMSES and its batch version RASS[2] (Thöny *et al.*, 1995), which is treated in a separate report (Löffler & Fischlin, 1997). The results will give essential information for the latter assessment, even though they may neither be accurate nor optimal.

For instance, to such aims these data about computers and software are necessary for

- the decisions which platforms should be preferred for the future development of RAMSES and RASS;
- potential performance improvements of RAMSES and RASS;

The presented benchmark experiments tie on a long tradition of established benchmark tests (Wirth, 1981) with the advantage that we are able to compare the whole collected material and get results out of a long expertise in these benchmark tests.
Nevertheless, we concentrate mainly to the new and thus for our aims relevant benchmark experiments. Therefore the results of the older benchmark experiments are concentrated shortly in the appendix A.

# 2. Materials and Methods

## 2.1 Machines, Compilers, and Compiler Options

The computers tested in the past are mainly those which are available at the ETH Zürich at that time and the computers which we used for the actual benchmark tests are today often used machines.

The late benchmark tests were performed on six Macintosh and three SUN workstations as well as on two IBM workstations. The characteristics of the used machines are summarized in the following three tables 2.1 to 2.3.

| Macintosh | IIfx | Quadra 700 | Quadra 950 | PowerPC 8100/80 | PowerBook 170 | PowerBook 520 and 540c |
|---|---|---|---|---|---|---|
| CPU | MC68030 (CISC) | MC68040 (CISC) | MC68040 (CISC) | PPC601 (RISC) | MC68030 (CISC) | 68L040 (CISC) |
| FPU | MC68882 | Integrated | Integrated | Integrated | MC68882 | no |
| Rate | 40 MHz | 25 MHz | 40 MHz | 80 MHz | *no information* | 66/33 MHz |
| System | 7.01 | 7.01 | 7.01 | 7.12 | 7.01 | 7.11 |

Table 2.1: Characteristic data of the tested Macintosh machines.

[2] RASS is an acronym for RAMSES Simulation Server.

| SUN workstations | SPARCstation 10 | SPARCserver 630 MP | SPARCstation ipx |
|---|---|---|---|
| CPU | SuperSPARC | 2 x SuperSPARC | FJMB86903 |
| FPU | Integrated | Integrated | Integrated |
| Rate | 40 MHz | 40 MHz | 40 MHz |
| System | SunOS 4.1.4 | SunOS 4.1.2 | SunOS 4.1.2 |

Table 2.2: Characteristic data of the tested SUN workstations.

| IBM wokstations | Intel 486 | Intel Pentium |
|---|---|---|
| CPU | 80486 | Pentium |
| FPU | 80487 (Integrated) | Integrated |
| Rate | 66 MHz | 90 MHz |
| System | DOS 6.21/Windows 3.1 | DOS 6.21/WFW 3.11 |

Table 2.3: Characteristic data of the tested IBM workstations. "WFW" is an abbreviation for "Windows for Workgroups".

We used the programming languages Modula-2, Pascal, and C with different compilers (cf. tab. 2.4), depending on the used platform.

| | Modula-2 | C | Pascal |
|---|---|---|---|
| Macintosh | MacMETH V3.2.2 | MPW C V3.3 | |
| SUN Workstation | em2 V2.0.8 | gcc/g++ V2.4 | |
| IBM Workstation | | | Borland Pascal V7.0 |

Table 2.4: The used combinations of compilers and platforms.

Benchmark results are dependent on the hard- and software configuration of the tested machines. For example an e-mail system can strongly influence the execution of programs, dependent e.g. on how often the mail server is contacted for new mail. This can strongly distort the benchmark results. Hence, to get comparable information we configured the tested computers similarly with respect to such applications, but besides left the machines in their normal working state because we were interested in testing the computers as they mainly were in everyday use. Therefore, we did not test optimized machines but standardized ones which are in practical use. That means, the end user of the machines is the main perspective of the benchmark tests.

The used Modula-2 compiler MacMETH V3.2.2 (Wirth *et al.*, 1992) is able to generate native code for the Motorola CISC processor family MC68. The MacMETH compiler **Compile** generate native code for this CPU type (cf. tab. 2.1) and the MacMETH compiler **Compile20** additionally supports the FPU (cf. tab. 2.1) of these processor family, i.e. **Compile20** generate code for the MC68020 processor and run on hardware platforms equiped with the MC68020/MC68881, the MC68030/MC68882, or the MC68040 processor. The compiler **Compile40** of MacMETH which generate native code for the MC68040 processor was not tested. Therefore, on the machines with such a FPU only native code for the FPU MC68020 was executed, i.e. the FPU of e.g. the Quadra 950 was not tested in its fastest mode.
MacMETH V3.2.2 is not able to generate native code for the RISC processors used by the

PowerPC family. Hence, all results produced on the tested PowerPC 8100/80 do not reflect the true abilities neither of this machine nor of the PowerPC family in general.
The integer overflow and, except for the subtest h and j, the range checks were disabled. All traps were enabled and for SANE all checks were enabled except the two options 'underflowHalt' and 'inexactHalt' (Wirth *et al.*, 1992).

On the SUN workstation the EPC Modula-2 compiler em2 V2.0.8 was used with the compiler options which enable the detection of range errors.
The information which are necessary for working with the debugger and the profiler was not produced. During the compilation and linking (**1**) the production of all post mortem diagnostics were suppressed and (**2**) all possible range checks were enabled (EPC, 1991).

For the MPW C V3.3 compiler the Apple's language extensions were enabled. The generated code was optimized for speed, but with no loop unrolling and without repeating global propagation and redundant store elimination. (Apple Computer, 1993b; Apple Computer, 1993a). No native code for the PowerPC 8100/80 was generated.

For the GNU compiler gcc/g++ V2.4 the information which are necessary for working with the debugger and the profiler was not produced. The benchmark program was compiled and linked with an optimization level 2. For a description refer to the GNU manual (foundation, 1993) and to the man pages. The generated code was not optimized for the processors of SUN workstations.

With Borland Pascal V7.0 native code for the 80286/80278 CISC processor of Intel was generated from the benchmark programs. Information for the debugger and the profiler was not produced. Range, stack, pointer, and overflow checks were disabled, and I/O checks were enabled. For the subtests h and j the range checks were also enabled (Borland International, 1992).

## 2.2 Standard Benchmark Tests

We define the benchmark test in a traditional way as a set of instructions which measures how well the hardware and software of a computer system perform together. Generally, benchmarks can either test individual, specific functions of a compiler or operating system (e.g. function-call overhead) or they can test the general performance of the machine by executing a number of operations (e.g. looping, searching etc.).

Such measurements can be executed in two ways, by

- counting the number of operations executed within a fixed time;
- fixing the number of operations and stopping the time which the program needs for the execution;

We preferred the first version whereby in the earlier tests a human monitored and interrupted the computation after a fix defined run-time, for example after 100 seconds. For the new tests we gave this control to the benchmark program and were therefore able to run the benchmark application at times where the machines usually are not used and hereby to run more tests.

The earlier tests used the traditional benchmark set (Wirth, 1981) which is shortly described in table 2.5. By this test set the basic operations which are often used in a program is covered.

| Test | (also used) | Short description of the benchmark tests |
|---|---|---|
| a | (1) | Empty REPEAT |
| b | (2) | Empty WHILE |
| c | (3) | Empty FOR |
| d | (4) | Integer arithmetic: (x * y) DIV (z + u) |
| e | (5) | Real arithmetic: (x * y) / (z + u) |
| f | (6) | SIN, EXP, LN, SQRT |
| g | (7) | Array access a[ ]=b[ ], b[ ]=a[ ] |
| h | (8) | Test g with bounds tests |
| i | (9) | Matrix access a[ ][ ]=b[ ][ ] |
| j | (10) | Test i with bound tests |
| k | (11) | call of an empty procedure |
| l | (12) | call of an empty procedure with 4 parameters |
| m | (13) | Array copying |
| n | (14) | Access via pointers |
| o | (15) | Reading a disk stream of integer values |

Table 2.5: Short description of the traditional benchmark tests which includes often used compiler instructions and integrated functions. These tests cover the basic operations which are often used in a program. Note, in figures 3.1 to 3.4 the numbers 1 to 15 are used instead of a to o.

Beside the tests listed in table 2.5 we performed an additional class of tests which was chosen to get more detailed information about widely used operations. This additional test set is shortly described in table 2.6.

This additional test class was introduced because the results of first benchmark experiments with RAMSES model-definition-programs (MPDs) as the stochastic forest model FORCLIM (Bugmann, 1994; Bugmann & Fischlin, 1994) or the deterministic forest model DISCFORM (Lischke *et al.*, 1996) revealed that beside the traditional benchmark test set additional tests are advantageous to assess the run-time behaviour of these programs. Therefore, we inspect the run-time behaviour of some few procedures as e.g. sinus or square root computation which seems essential to our aims. The data from this additional benchmark test set can help to interpret the run-time measurements or to improve the performance of model-definition-programs (MPDs). Particularly, this selection is mainly restricted to our special aims and hence not representatively chosen.

| Test | (also used) | Short description of the benchmark tests |
|---|---|---|
| N1 | (16) | Empty loop with fast computation of i := i - 1 |
| N2 | (17) | Sinus function |
| N3 | (18) | Arcus tangens function |
| N4 | (19) | Exponential function function |
| N5 | (20) | Natural logarithm function |
| N6 | (21) | Square root function |
| N7 | (22) | Type conversion: real to longint |
| N8 | (23) | Random (integer) number generation |
| N9 | (24) | Random (real) number generation |
| N10 | (25) | Absolute function |
| N11 | (26) | Type conversion: real to integer |
| N12 | (27) | Type conversion: integer to real |
| N13 | (28) | Power (x^r, x and r real) |
| N14 | (29) | Power (x^i, x real, i integer) |
| N15 | (30) | Maximum of two real values |
| N16 | (31) | Maximum of two integer values |
| N17 | (32) | Access of real values |
| N18 | (33) | Reading a disk stream of real values |

Table 2.6: Short description of the additional benchmark set which includes often used routines. These additional test set is arranged for special purposes of RAMSES model-definition-programs (MDPs) and therefore not representatively chosen.
Note, in figures 3.1 to 3.4 the numbers 16 to 33 are used instead of N1 to N18.

Because of incompatibilities of different programming languages some language features had to be replaced by equivalent constructions. For example, type conversion from integer to real variables have to be made explicitly in Modula-2 by the procedure *FLOAT* whereas in C this conversion can be made explicitly by the cast operator *(int)* or implicitly by the compiler. Such differences as explicit or implicit type conversion are summarized in the tables 2.7 and 2.8.

For the programming languages Modula-2 and Pascal we followed the implementation of Wirth (Wirth, 1981) because of possible comparisons of the long series in these benchmark tests, whereas for the language C there is no such restriction. Hence, for the comparisons of the results gained by the C and Modula-2 benchmark test, the differences of the implementations which are described in the following has to be in mind.

For the traditional benchmarks the differences between the Modula-2, Pascal, and C version are minimal. They consist in the use of the

- operators *--i* and *++i* in C vs. *i:=i-1* and *i:=i+1* in Modula-2 and Pascal for counting the loops in the subtests a to o. The measurements for the C benchmark version showed that it doesn't matter whether the operator *++i* or *i=i+1* respective *--i* or *i=i-1* is used (cf. tab. 3.18).

- system procedure *read* in Pascal vs. procedure *GetInt* from the Module DMFiles of the Dialog Machine V2.2 (Fischlin, 1986; Vancso-Polacsek *et al.*, 1987; Fischlin *et al.*, 1988; Keller, 1989; Fischlin, 1991; Fischlin *et al.*, 1994) in Modula-2 vs. the library procedure *fscanf* in C for the subtest o (reading disk streams);

- element wise array copying in C vs. direct array copying (by the construction *array1[ ] := array2[ ]*) in Modula-2 and Pascal for subtest m;

- do while loop in C vs. repeat until loop in Modula-2 and Pascal for all subtests except the subtest c;

| Test no. | Modula-2 | Pascal | C |
|---|---|---|---|
| a to o | i := i - 1; | *such as Modula-2* | --i; |
| c and d | i := i + 1; | *such as Modula-2* | ++i; |
| a, b, d to o | repeat ... until ... | *such as Modula-2* | do { ... } while (not ...) |
| m | array1 := array2; | *such as Modula-2* | for (...)<br>{array1[] := array2[]} |
| o | GetInt ( ... ); | read ( ... ); | fscanf ( ... ); |

Table 2.7: Summarization of the differences between the programming languages Modula-2, Pascal, and C in the additional benchmark test set.
The measurements for the C benchmark version showed that it doesn't matter whether the operator *++i* or *i=i+1* respective *--i* or *i=i-1* is used (cf. tab. 3.18).

Another difference are the compiler options which are different for the used compilers. We used the available compiler options for avoiding array index-, subrange-, pointer- etc. checks to perform the tests h and j.

For the additional test class, the differences between the benchmark versions for the languages (cf. tab. 2.8) Modula-2, Pascal, and C stem from not existing built-in functions of the compilers as well as from testing library procedures. For some cases, library procedures were not available and hence had to be defined in the benchmark programs.
For the Modula-2 benchmark version we took the procedures RandomReal, RandomInt, POWER, POWERI, Rmax, Imax, and GetReal from the Dialog Machine V2.2.

The results coming from these benchmark versions are used to get an impression of the difference in the run-time behaviour of Modula-2 vs. C and to compare the IBM with the SUN workstations.

| Test No. | Modula-2 | Pascal | C |
|---|---|---|---|
| N7 | Entier | Round | (int) floor |
| N8 | RandomInt | Trunc (Random) | (int) rand |
| N9 | RandomReal | Random | rand |
| N11 | TRUNC | such as Modula-2 | (int) ceil |
| N12 | FLOAT | implicit type conversion | implicit type conversion |
| N13 | POWER | *RPot ()* | pow (real,real) |
| N14 | POWERI | *RIPot ()* | pow (real,int) |
| N15 | Rmax | *dMax* | *dMax* |
| N16 | Imax | *iMax* | *iMax* |
| N18 | GetReal | read | fscanf |

Table 2.8: Summarization of the differences between the Modula-2, Pascal, and C benchmark versions of the additionally measured tests. The procedures which are set in italic are standardly implemented in the benchmark program.
These additional tests were introduced to obtain more data to interpret the run-time measurements or to improve the performance of RAMSES model-definition-programs (MDPs). Particularly, this selection is mainly restricted to our special aims and hence not representatively chosen.

## 2.3 Test Conditions

For a good evaluation of computer systems, different factors must be taken into consideration which can strongly influence the programmer's productivity. Therefore the aims of the new benchmark examinations are quite numerous. Beside the run-time behaviour of different platforms, programming languages as well as the contrast of constructions such as different loops in Modula-2 vs. in C or one- vs. two dimensional array handling etc. (cf. tab. 3.2 and 3.18), we are interested in features such as measuring the effects of the

- SANE[3] library of Apple (Apple Computer, 1986);
- extended compiler of MacMETH (Wirth *et al.*, 1992) which generates native code for the MC68020, MC68881/2, and MC68040 co-processor;
- optimized library DMMathLibF of the Dialog Machine V2.2 for the co-processors MC68020, MC68881/2, and MC68040;
- system extensions which are included in the system of a Macintosh computer to extend the functionality of a Macintosh. These extensions are executed as background processes which can affect the run-time behaviour of applications.
- network. For example, the effect of programs such as e-mail which periodically contact a server via a (local) network can strongly influence the run-time behaviour of applications.
- FPU software emulation for the PowerPC 8100/80 combined with non-native code applications which needs a co-processor. This FPU software emulates a mathematical co-processor which is not included in some computers. The available FPU software for these tests is the shareware SoftwareFPU V3.02 and V3.03 as well as PowerFPU V1.01;

---

[3] SANE is an acronym for Standard Apple Numeric Environment.

- the SANE library of Apple for the PowerPC 8100/80.

## 2.4 Benchmark Algorithm

For each subtest a to o respective N1 to N18 the benchmark algorithm determines the number of iterations which can be exactly executed in a given (fixed) run-time $t_{fix}$, e.g. 100 seconds. This number of iterations is iteratively calculated by the formula

$$n_\kappa = n_{\kappa-1} * t_{fix} / \Delta t_{\kappa-1},\ \kappa > 1$$

with $\Delta t_{\kappa-1}$ the measured run-time. The finite sequence $(n_\kappa)_{\kappa>0}$ of iteration numbers starts with the chosen value $n_0$ and closes with a number $n\gamma$ for which the criterion $t_{fix} = \Delta t\gamma$ is fulfilled.

## 2.5 Sensitive Tests and Accuracy

The benchmarks are affected by errors of the time measurement. By giving the control of this time measurement to the benchmark program we eliminated the errors coming through the unreliability of stopping the time by hand. As a consequence of the time measurement algorithm described above, the best relative accuracy we can get is given by the formula

$$a_{rel} = a_{abs} / t_{fix}, \tag{2.1}$$

with $a_{abs}$ the absolute accuracy, and $t_{fix}$ is the fixed time for which we want to know how many iterations of each subtest can be executed.
The absolute accuracy $a_{abs}$ of the time measurement is one second, due to the implementation of the Module DMClock of the Dialog Machine V2.2. Hence we have a maximal relative accuracy $a_{rel}$ of

- 1% for $t_{fix} = 100$ seconds;
- 0.2% for $t_{fix} = 500$ seconds;

For most subtests the time was fixed to 100 seconds, but for some sensitive test to 500 seconds. By sensitive tests we mean those whose effects on the number $n_{run}$ of repetitions (of each subtest) for $t_{fix} = 100$ seconds are hardly to recognize, i.e. in the order of magnitude of the relative error. Then we try to minimize the error by increasing the time $t_{fix}$.
Afterwards, the result of these sensitive benchmark tests were normalized to 100 seconds in order to be able to compare these results with tests under other conditions, too.

Mainly, the results of the subtests show a clear trend and can thus directly be interpreted and compared. For sensitive tests in the contrary it is not always possible to compare the results of the different subtests in this way. For such cases we used the criteria described in the following.

The number

$$e_{rel} = n_{run} * (a_{abs} / t_{fix}) = n_{run} * a_{rel} \tag{2.2}$$

is an estimation of the error band width of the benchmark measurement, whereby $n_{run}$ is the measured number of iterations of a subtest which is executed within the time $t_{fix}$. To get an

impression whether the difference of two runs is significant related to the error $e_{rel}$, that means to find out if there is a recognizable effect in the measurement, we use the relation

$$r_{rel} = 100 * (n_{run2} - n_{run1}) / e_{rel1} \quad (2.3a)$$

which is an estimation (in percent) for how many times faster or slower the run was under the different conditions, i.e. whether the difference $\Delta n_{run} = n_{run2} - n_{run1}$ is a significant measurement related to the error band width $e_{rel1}$ of the iteration number $n_{run1}$. Thereby $n_{run1}$ and $n_{run2}$ are the measured numbers of iterations of a subtest under different conditions, which are executed within the time $t_{fix}$. For our purposes we use the following two numbers

$$r_{effect} = r_{rel} / 100 \quad (2.3b)$$

and $$r_{sign} = sign(r_{effect}). \quad (2.4)$$

Thus, if $r_{effect} > 1$ we can assume that we measured an effect and therefore that the run with iteration number $n_{run2}$ is measurably faster than the run with the iteration number $n_{run1}$. Otherwise we assume that the measured difference lies in the error band width of the first run.

The number $r_{sign}$ shows whether the second run was faster ($r_{sign} > 0$) or slower ($r_{sign} < 0$) and defines therefore the border of significance ($|r_{effect}| \geq 1$) for the different runs.

### 2.6 Standard Benchmark Tests with the Application Speedometer

The Application Speedometer V4.02 of Scott Berlfield offers different standard benchmark tests for Macintosh computers such as Quicksort, Queens, Sieve etc.
We included tests with this application for Macintosh computers, too.

Tests with Speedometer V4.02 can not be taken too seriously, because without the source code of this application it is impossible to make a decision about the quality of the resulting benchmarks. Nevertheless, these tests can help to complete or revise the view of the tests with our own benchmark program.

The results from these tests are summarized in chap. 3.2.

## 3. Results

We give a summarization of the results of the benchmark tests completed in the last decade. Thereby, the older benchmark test results are treated into the appendix A in a short and compressed way without graphical representations and statistical analysis of the data.

The new benchmark results are completely described in this chapter. These results are given in absolute speed, measured in number of iterations within the time $t_{fix}$ of 100 seconds, or in percent speed. Because of the large amount of data coming out of the many benchmark runs, we extracted the results into mean values $M$ (of the percentage values) if this is justified by the behaviour, i.e. significant results of all subtests with respect to the error band width. Otherwise we give the additional information of the differences in these subtests, too.

The extraction of the data are accomplished in two ways. First, we use the mean values of all subtests of a tested machine and compute by the equation

$$M = 100 * M_{test} / M_{reference} \quad (3.1)$$

how many times faster or slower a run is. The factor $M$ represents the percentage relation of $M_{test}$ (mean of the test over all subtests) to $M_{reference}$ (mean of the test over all reference subtests). $M_{reference}$ is the reference to which the value $M_{test}$ is related.

Second, the difference of each subtest a to o and N1 to N18 between the benchmark test series on several machines or on the same machine under different conditions are computed by the formula

$$D_{subtest} = 100 * S_{machine} / S_{reference} - 100 \quad (3.2)$$

where $S_{reference}$ is the reference to which the value $S_{machine}$ is related. The number $D_{subtest}$ represents the absolute gain or loss of speed in percent. Note, formula (3.2) can be written as $D_{subtest} = 100 * (S_{machine} / S_{reference} - 1) = 100 * (S_{machine} - S_{reference}) / S_{reference}$.

Then, the presented values of the benchmark test series are the mean, standard deviation, minimum, and maximum of the values $D_{subtest}$ of all subtests.

### 3.1 The new benchmark tests

First, we extract general statements (cf. chap. 2.3) from the two benchmark test sets (cf. tab. 2.5 and 2.6). Therefore we use the data obtained by the benchmark run (Modula-2 version of the benchmark program) on a Quadra 700, whereby this benchmark test is performed by the combination Sane on/Comile20 off/DMMathLib (cf. tab 3.4 and tab. 4.5).

| Test | No. of iterations | Test | No. of iterations |
|---|---|---|---|
| a | 272747475 | N1 | 351160000 |
| b | 271840000 | N2 | 305556 |
| c | 223020000 | N3 | 420792 |
| d | 40623762 | N4 | 228500 |
| e | 1790000 | N5 | 390909 |
| f | 75000 | N6 | 390594 |
| g | 52950495 | N7 | 1445000 |
| h | 30772277 | N8 | 686500 |
| i | 3859 | N9 | 1955941 |
| j | 2605 | N10 | 223030303 |
| k | 82020000 | N11 | 271020000 |
| l | 66100000 | N12 | 204110000 |
| m | 1145000 | N13 | 131111 |
| n | 2192500 | N14 | 1210303 |
| o | 378788 | N15 | 8616162 |
| | | N16 | 52484849 |
| | | N17 | 115930000 |
| | | N18 | 248485 |

Table 3.1: This table shows the absolute number of iterations which could be executed for each subtest (cf. tab.2.5 and 2.6) for $t_{fix} = 100$ seconds on a Quadra 700 running the Modula-2 version of the benchmark program. This benchmark test is performed by the combination Sane on/Comile20 off/DMMathLib (cf. tab. 3.4 and 4.5).

The table 3.2 contains information about various differences of the run-time behaviour of program constructions as e.g. of different loop constructions, using compiler options for checking ranges etc. In each group we used eq. 3.1 and refer to the slowest run as 100%. For the comparison of one dimensional array vs. multi-dimensional array access and array vs. list access the different implementations related to the assignments of variables of these subtests has to be taken into account.

Refer to table 3.18 which contains the same information for the C benchmark test.

| *compiler options* | ***I*** | *type conversion* | ***II*** | *loops* | ***III*** |
|---|---|---|---|---|---|
| g - no options | 172 | N12 | 18756 | N1 | 158 |
| h | 100 | N11 | 14125 | a | 129 |
| i - no options | 148 | N7 | 100 | b | 122 |
| j | 100 | | | c | 100 |
| *integer vs. real disk streams* | ***IV*** | *built-in function Inc(i)* | ***V*** | *integer vs. real arithmetic* | ***VI*** |
| o | 152 | a | 129 | d | 2270 |
| N18 | 100 | N1 | 100 | e | 100 |
| *procedure calls* | ***VII*** | *1-D array vs. 2-D array* | ***VIII*** | *array vs. pointer* | ***IX*** |
| k | 124 | g | 137 | g | 1208 |
| l | 100 | i | 100 | n | 100 |

Table 3.2: This table contains in nine boxes ***I*** to ***IX*** information about various differences of the run-time behaviour of program constructions in the programming language Modula-2 in percent (cf. eq. 3.1) to the slowest run in each group which is the reference, i.e. 100%. These data are extracted from table 3.1.
Refer to table 3.18 which contains the same type of data for the C benchmark test.

Second, we present the data of the tests executed under various conditions (cf. chap. 2.3) as using the SANE library or not etc. We examined the most important combinations of the options SANE on/off, using Compile/Compile20 of MacMETH and the library DMMathLib/DMMathLibF from the Dialog Machine V2.2. The runs were performed in the programming language Modula-2 (by the compiler MacMETH) on a Quadra 950.

Table 3.3 shows the absolute data for the most important combinations of these options, whereas table 3.4 shows the mean value of the subtests in percentage difference (cf. eq. 3.2) related to the option combination SANE on/Compile20 off/DMMathLib which performed the slowest run.

| Test no. | SANE on Compile DMMathlib | SANE off Compile DMMathlib | SANE off Compile20 DMMathlib | SANE off Compile20 DMMathlibF |
|---|---|---|---|---|
| a | 405959616 | 405959616 | 407386144 | 406039616 |
| b | 328282848 | 325000000 | 361346528 | 363757568 |
| c | 362646464 | 364424224 | 327100000 | 327100000 |
| d | 54636364 | 54636364 | 54636364 | 54090000 |
| e | 2370000 | 6010000 | 47590908 | 47590908 |
| f | 95959 | 213636 | 220202 | 1448514 |
| g | 71070712 | 70840000 | 88019800 | 88237624 |
| h | 41780000 | 41780000 | 48222224 | 48222224 |
| i | 5344 | 5315 | 5805 | 5805 |
| j | 3415 | 3429 | 3692 | 3655 |
| k | 112217816 | 112316832 | 112202024 | 112316832 |
| l | 87801976 | 87801976 | 90900000 | 90900000 |
| m | 1516500 | 1521717 | 1521782 | 1521717 |
| n | 2923500 | 2923500 | 2684500 | 2698500 |
| o | 479797 | 485000 | 485000 | 479797 |
| N1 | 465900992 | 465979776 | 409737376 | 408747488 |
| N2 | 389899 | 866666 | 936633 | 4950495 |
| N3 | 563131 | 1472772 | 1472772 | 5136363 |
| N4 | 291584 | 628282 | 692929 | 4590000 |
| N5 | 509595 | 1373762 | 1373762 | 4440000 |
| N6 | 514851 | 1320297 | 1320297 | 21075500 |
| N7 | 1906565 | 2775757 | 2775757 | 45356060 |
| N8 | 801000 | 1240000 | 1268811 | 10787879 |
| N9 | 2600495 | 4427272 | 4672277 | 14903000 |
| N10 | 297420000 | 294475264 | 272606048 | 272606048 |
| N11 | 361188128 | 361683168 | 326217824 | 326404032 |
| N12 | 270722784 | 270722784 | 271128704 | 273430016 |
| N13 | 163000 | 341400 | 334600 | 334600 |
| N14 | 1626666 | 3541800 | 3539400 | 3541800 |
| N15 | 11405941 | 27696970 | 27696970 | 27696970 |
| N16 | 69220000 | 69545456 | 69545456 | 69545456 |
| N17 | 163360000 | 172868688 | 171140000 | 172868688 |
| N18 | 303030 | 296000 | 298989 | 296000 |

Table 3.3: This table shows the absolute number of iterations which could be executed for each subtest for $t_{fix} = 100$ seconds for essential possibilities of mathematical choices. These choices are SANE on/off, using Compile/Compile20 of the Modula-2 compiler MacMETH, and the libraries DMMathLib/ DMMathLibF of the Dialog Machine V2.2.

| | | DMMathLib | | DMMathlibF |
|---|---|---|---|---|
| | | Compile20 | Compile20 | Compile20 |
| | | on | off | on |
| SANE | on | 61 | **0** | 476 |
| SANE | off | 106 | 40 | 498 |

Table 3.4: Results of different options SANE on/off, Compile20 on/off, and DMMathLib or the fast version DMMathLibF for the benchmark tests. These results are the mean value of the subtests in percentage difference (cf. eq. 3.2) given in tab 3.3. The results are related to the option combination SANE on/Compile20 off/DMMathLib (bold faced in the table) which is chosen as reference value because this combination performed the slowest run.
For example, the combination SANE off/Compile20 on/DMMathLib is in mean 106% faster than the reference.

These three options (SANE on/off, using Compile or Compile20, and using DMMathLib or the fast version DMMathLibF) are essential for mathematical operations using MacMETH and RAMSES. However, the mean values of table 3.4 include all subtests, i.e. mathematical and non-mathematical ones. Therefore, in table 3.5 we look only to the behaviour of the mathematical subtests f, N2 to N7, N13 and N14 (cf. tab. 2.5 and 2.6), giving the same view such as in table 3.4.

| | | DMMathLib | DMMathLib | DMMathlibF |
|---|---|---|---|---|
| | | Compile20 | Compile20 | Compile20 |
| Test no. | SANE | on | off | on |
| f: Combination of | on | 1908 | **0** | 1883 |
| mathematical functions | off | 1908 | 154 | 1908 |
| N2: Sinus | on | -12 | **0** | -12 |
| | off | -13 | 0.02 | -12 |
| N3: Arcus Tangens | on | 2 | **0** | 1158 |
| | off | 140 | 122 | 1170 |
| N4: Exponential | on | 0.18 | **0** | 811 |
| function | off | 162 | 162 | 812 |
| N5: Logarithmical | on | 1 | **0** | 1481 |
| function | off | 138 | 116 | 1474 |
| N6: Square root | on | 2.04 | **0** | 769 |
| | off | 170 | 170 | 771 |
| N7: Type | on | 0 | **0** | 3953 |
| conversion | off | 156 | 156 | 3994 |
| N8: Power with | on | 0 | **0** | 1 |
| real values | off | 0.15 | 0 | 1 |
| N9: Power with | on | 1 | **0** | 4 |
| integer values | off | 105 | 110 | 105 |

Table 3.5: Results of different options for the benchmark tests for the mathematical subtests from tab 3.3. These results are given in percentage difference (cf. eq. 3.2) related to the option combination SANE on/Compile20 off/DMMathLib (bold faced in the table) which is the reference of table 3.4, too.
For example, for the subtest N6 (square root) the combination SANE on/Compile20 on/DMMathLibF is 769% faster than the reference.

Third, we examined the influence of extensions (cf. chap. 2.3) used by a Macintosh and of a network connection (cf. chap. 2.3) on the run-time of the benchmark program. The runs were designed with benchmark program written in the programming language Modula-2 (using MacMETH) on a Quadra 700.
We give the mean, standard deviation, minimum, and maximum value over all subtests of the results in table 3.6 and by the graphics 3.1, 3.2 and 3.3[4] we show the values $r_{trend}$ and $r_{sign}$ (cf. chap. 2.5) for all subtests.

The results of the subtests under the different conditions are difficult to interpret even if we run the tests with $t_{fix} = 500$ seconds; they are for some subtests small, for others rather big and for some subtests they are positive and for others negative so that they might be due to the error in the time measurements (cf. eq. 2.2).
This is regognizable in the graphics 3.1, 3.2, and 3.3 which look closer into the test results. We detect that the high gain in run-time is contributed mostly from the mathematical subtests.

| | No extensions | Unconnected to a network | No extensions, unconnected to a network |
|---|---|---|---|
| Mean | 396 | 29 | 1 |
| Std Deviation | 836 | 55 | 6 |
| Min | -18 | -10 | -15 |
| Max | 3894 | 165 | 23 |

Table 3.6: The influence of extensions (cf. chap. 2.3), the network connection, and both: using extensions and being connected with a network, for a Macintosh Quadra 700. The run-time behaviour of the benchmark program is measured as mean run-time of all subtests (cf. chap. 2.2, tab. 2.5, and 2.6) in percent (cf. eq. 3.2) related to using extensions and being connected to a network.
For example, the measurements yield that not using extension increase the run-time in mean of about 396 percent.

[4] In these graphics the subtests are numbered from 1 to 33 which represent the marking a to o and N1 to N18.

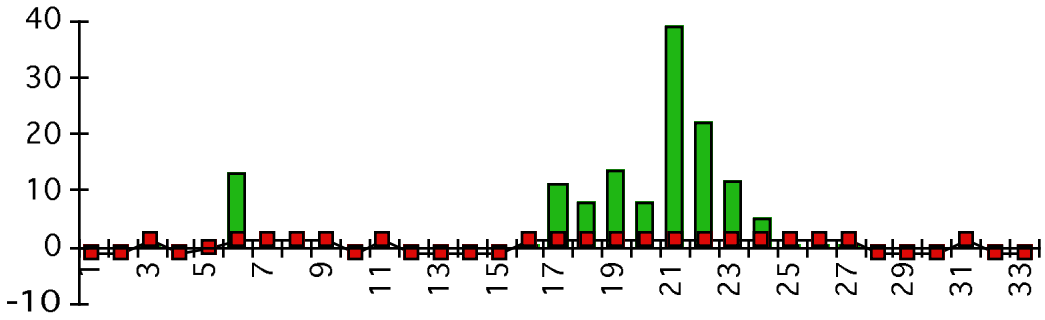


Figure 3.1: The values $r_{effect}$ (cf. eq. 2.3b) represented by bars and $r_{sign}$ (cf. eq. 2.4) represented as line with squares show for each subtest (cf. chap. 2.2, tab. 2.5, and 2.6) the measured differences between using extensions or not (cf. chap. 2.3). The squares lying at 1 or -1 give also the border for which the subtests can be assumed as significantly faster (values > 1) or slower (values < 1).
For example, the mathematical subtests f, N2 to N7, N13, and N14 seem to performe faster for the configuration "not using extensions". Since the results passed the border $r_{sign}$, they are assumed as significant.
Note, the subtests a to o and N1 to N18 are declared as 1 to 33 (cf. tab. 2.5 and 2.6).

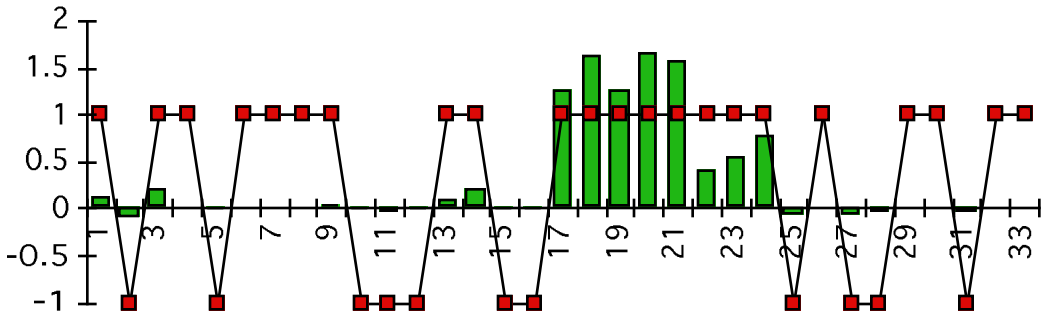


Figure 3.2: The values $r_{effect}$ (cf. eq. 2.3b) represented by bars and $r_{sign}$ (cf. eq. 2.4) represented as line with squares show for each subtest (cf. chap. 2.2, tab. 2.5, and 2.6) the measured differences between being connected to a network or not. The squares lying at 1 or -1 give also the border for which the subtests can be assumed as significantly faster (values > 1) or slower (values < 1).
For example, the mathematical subtests f, N2 to N7, N13, and N14 seem to performe faster for the configuration "not connected to a network".. Since the results passed the border $r_{sign}$, they are assumed as significant.
Note, the subtests a to o and N1 to N18 are declared as 1 to 33 (cf. tab. 2.5 and 2.6).

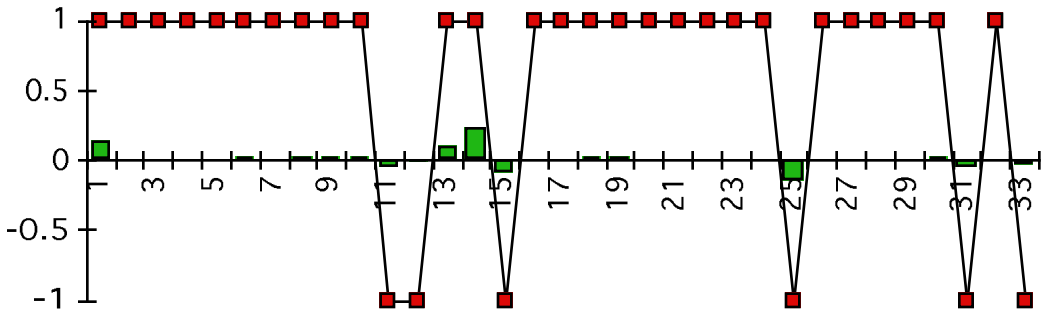


Figure 3.3: The values $r_{effect}$ (cf. eq. 2.3b) represented as bars and $r_{sign}$ (cf. eq. 2.4) represented as line with squares show for each subtest (cf. chap. 2.2, tab. 2.5, and 2.6) the influence of not using extensions (cf. chap. 2.3) combined combined with not being connected to a network. The squares lying at 1 or -1 give also the border for which the subtests can be assumed as significantly faster (values > 1) or slower (values < 1).
For example, the mathematical subtests f, N2 to N7, N13, and N14 seem to performe faster for the configuration "not using extensions and not connected to a network", but since no result passed the border $r_{sign}$, the results can not be assumed as significant.
Note, the subtests a to o and N1 to N18 are declared as 1 to 33 (cf. tab. 2.5 and 2.6).

Fourth, the results of the benchmark runs on the PowerPC 8100/80 using several software emulation programs (cf. chap. 2.3) for the FPU or not is given in table 3.7, in which the mean, standard deviation, minimum, and maximum value over all subtests of the generated results are included. The runs were designed with the benchmark program written in the programming language Modula-2 (MacMETH) on a PowerPC 8100/80. Additionally, figure 3.4 shows the influence of the PowerFPU V1.01 vs. using no such emulation for each subtest.

This results show clearly that the shareware SofwareFPU V3.02 and V3.03 allows to run programs which need a FPU on a PowerPC 8100/80 but without the performance which a co-processor adds to a computer. In contrary, the shareware PowerFPU V1.01 yields additionally a significant increase in the run-time behaviour of applications which needs a FPU.

For details, each subtest for the case using PowerFPU is shown in figure 3.4 in which the values $r_{trend}$ and $r_{sign}$ are given related to the case of using no software emulation. As suspected, figure 3.4 shows that the gain in run-time is realized in the mathematical subtests (cf. chap. 2.2).

| | SofwareFPU V3.02 | SofwareFPU V3.03 | PowerFPU V1.01 |
|---|---|---|---|
| Mean | 0.5 | -0.7 | 96 |
| Std Deviation | 7.7 | 0.9 | 212 |
| Min | -3 | -3 | -100 |
| Max | 43 | 1 | 933 |

Table 3.7: The influence of using software emulation or not with the shareware SoftwareFPU V3.02, SoftwareFPU V3.03, and PowerFPU V1.01 to the run-time behaviour of the benchmark program on a PowerPC 8100/80. These values over all subtests are given in percent (cf. eq. 3.2) related to the run without using such a software emulation.
For example, the measurements yield that using PowerFPU V1.01 increase the speed in mean of about 96 percent.

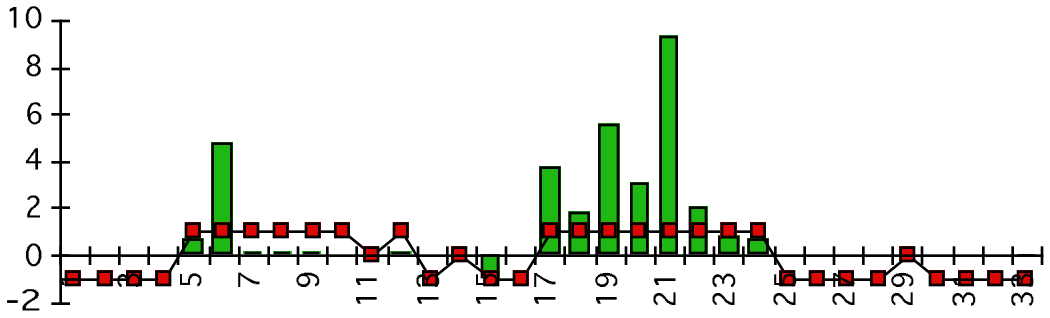


Figure 3.4: The values $r_{effect}$ (cf. eq. 2.3b) represented as bars and $r_{sign}$ (cf. eq. 2.4) represented as line with squares show for each subtest (cf. chap. 2.2, tab. 2.5, and 2.6) the influence of using the FPU emulation PowerFPU V1.01 on a PowerPC 8100/80. The squares lying at 1 or -1 give also the border for which the subtests are significantly faster (values > 1) or slower (values < 1).
For example, the mathematical subtests f, N2 to N7, N13, and N14 seem to performe faster if the software emulation PowerFPU V1.01 is used. Since the results passed the border $r_{sign}$, they are assumed as significant.
Note, the subtests a to o and N1 to N18 are declared as 1 to 33 (cf. tab. 2.5 and 2.6).

Fifth, the results for the SANE library for the PowerPC 8100/80 (cf. chap. 2.3) is given in table 3.8. The runs were designed with the benchmark program written in the programming language Modula-2 (MacMETH) on a PowerPC 8100/80. This test run were performed by the combinations SANE on/Compile/DMMathLib vs. SANE off/Compile/ DMMathLib.

| | all subtests | mathematical subtests e, f, N2 to N7, N13 to N15 |
|---|---|---|
| Mean | 8 | 24 |
| Std Deviation | 17 | 21 |
| Min | 81 | 81 |
| Max | -7 | 10 |

Table 3.8 The run-time behaviour of the SANE library on a PowerPC 8100/80. The values of the subtests are given in percent (cf. eq. 3.2) related to the run using SANE. For example, not using SANE is in mean 24% faster for the mathematical subtests.
In this test the mathematical combinations SANE on/Compile/DMMathLib vs. SANE off/Compile/DMMathLib were used.

Sixth, we come to the performance of the different machines (and therefore of the combination computers, operating systems, and emulations as it is mainly in every day use) which are used for the benchmark test. Here we use three sights of view.

First, we took the means of all subtests of each tested machine and compute by eq. 3.1 the gain and loss of the performance. The two tables 3.9 and 3.10 and figure 3.5 show how many times faster or slower the tested machines are related to the Macintosh IIfx which is referenced as 100%. Note, the Modula-2 benchmark program for the Macintosh computers was configured by the options SANE on/Compile/MathLib (cf. chap. 2.1, tab. 3.4 and 3.5). For the configuration of the used compilers on the SUN and IBM workstations refer to chap. 2.1.
Note, for these tests the software emulation SoftwareFPU V3.02 was used, since the software emulation PowerFPU V1.01 was not available at this time. With the results which are summarized in table 3.7 and figure 3.4, the performance of the PowerPC 8100/80 with PowerFPU V1.01 can be assumed as similar as the Quadra 700.

| Quadra 700 | Quadra 950 | PowerPC 8100/80 | PowerBook 170 | PowerBook 520 and 540c |
|---|---|---|---|---|
| 158 | 217 | 103 | 60 | 215 |

Table 3.9: The mean gain of performance of the Macintosh computers related to the Macintosh IIfx, i.e. 100%. These values are the average of all subtests related to 100% for the Macintosh IIfx. For example, the Quadra 950 is 217% faster than the Macintosh IIfx.
Note, we used the options SANE on/Compile/MathLib (cf. chap. 2.1, tab. 3.4 and 3.5).
The PowerPC 8100/80 had included the software emulation SoftwareFPU V3.02 (cf. tab. 3.7, 3.8, and fig. 3.4). With the results which are summarized in table 3.7 and figure 3.4, the average performance of the PowerPC 8100/80 with PowerFPU V1.01 can be assumed as similar as the Quadra 700.

| SPARCstation 10 | SPARCserver MP 630 | SPARCstation ipx | IBM (486) | IBM (Pentium) |
|---|---|---|---|---|
| 566 | 237 | 212 | 279 | 663 |

Table 3.10: The mean gain of performance of the SUN and IBM workstations related to the Macintosh IIfx, i.e. 100%. These values are the average of all subtests related to 100% for the Macintosh IIfx. For example, the IBM (Pentium) is 663% faster than the Macintosh IIfx.
For the configuration of the used compilers on the SUN and IBM workstations refer to chap. 2.1.

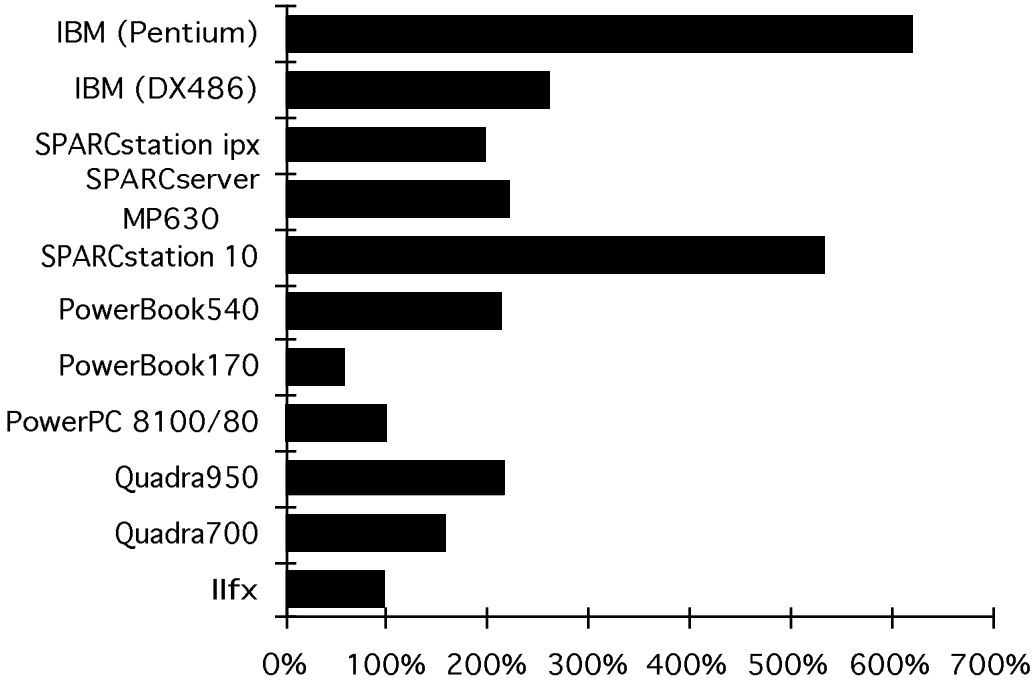


Figure 3.5: The mean gain of performance of all computers related to the Macintosh IIfx, i.e. 100%. These values are the average of all subtests related to 100% for the Macintosh IIfx. For example, the SPARCstation 10 is 566% faster than the Macintosh IIfx.

Note, the benchmark on the Macintosh computers were not performed in the fastest way; we used the options SANE on/Compile/MathLib (cf. chap. 2.1, tab. 3.4 and 3.5). For the configuration of the used compilers on the SUN and IBM workstations refer to chap. 2.1.
The PowerPC 8100/80 had included the software emulation SoftwareFPU V3.02 (cf. tab. 3.7, 3.8, and fig. 3.4). With the results which are summarized in table 3.7 and figure 3.4, the average performance of the PowerPC 8100/80 with PowerFPU V1.01 can be assumed as similar as the Quadra 700.

Second, with eq. 3.2 we looked more detailed at the run-time behaviour of each subtest.

The differences between the several machines in relation to the subtests a to o and N1 to N18 are in parts quite big. The following two tables 3.11 and 3.12 give more detailed information. These data series are computed by eq. 3.2 (with $S_{reference} = S_{IIfx}$) with which the resulting data $D_{subtest}$ for each subtest is computed with $S_{machine}$ and $S_{IIfx}$ the values of a subtest of the special machine and the Macintosh IIfx, respectively.

| Test no. | Quadra 700 | Quadra 950 | PowerPC 8100/80 | PowerBook 520 and 540c |
|---|---|---|---|---|
| a | 58 | 135 | -12 | 111 |
| b | 98 | 139 | -13 | 118 |
| c | 52 | 146 | -15 | 150 |
| d | 71 | 130 | 72 | 130 |
| e | *141* | *219* | *380* | 28 |
| f | *106* | *163* | *1451* | 12 |
| g | 45 | 95 | 0 | 96 |
| h | 16 | 58 | 11 | 60 |
| **i** | 43 | 98 | 29 | 100 |
| **j** | 24 | 62 | 33 | 65 |
| k | 52 | 108 | 25 | 113 |
| **l** | 87 | 149 | 39 | 160 |
| m | 100 | 165 | 37 | 194 |
| n | 65 | 120 | 35 | 173 |
| N1 | 49 | 98 | -4 | 100 |
| N2 | 100 | *155* | *1175* | 14 |
| N3 | *138* | *219* | *694* | 18 |
| N4 | *104* | *160* | *1654* | 13 |
| N5 | *137* | *209* | *1000* | 16 |
| N6 | *135* | *209* | *2799* | 22 |
| N7 | 55 | 104 | 488 | 6 |
| N10 | 52 | 102 | 2 | 104 |
| N11 | 61 | 115 | 1 | 117 |
| N12 | 39 | 85 | 1 | 104 |
| N13 | *108* | *159* | *155* | 13 |
| N14 | *133* | *213* | *170* | 36 |
| N15 | *122* | *194* | *106* | 9 |
| N16 | 91 | 151 | 18 | 146 |
| N17 | 61 | 127 | 24 | 65 |
| N18 | 55 | 89 | -46 | 289 |

Table 3.11: This table shows (in percent) for every subtest which Macintosh computer is faster or slower as the reference machine Macintosh IIfx, i.e. 100%. High values are set in italic.
The used configuration is (cf. chap. 2.1, tab. 3.4 and 3.5) Sane on/Compile/DMMathLib. Note, for these tests, the PowerPC 8100/80 had included the software emulation SoftwareFPU V3.02 (cf. fig. 3.4).

| Test no. | SPARCsta–tion 10 | SPARCser–ver MP 630 | SPARCsta–tion ipx | IBM(Pen–tium) | IBM (486) |
|---|---|---|---|---|---|
| a | 474 | 130 | 130 | 777 | 223 |
| b | 477 | 142 | 141 | 500 | 133 |
| c | 349 | 94 | 93 | 456 | 138 |
| d | 428 | 182 | 181 | 111 | 6 |
| e | *33299* | *13298* | *10614* | *12274* | *5599* |
| f | *25867* | *10133* | *7612* | *10514* | *6586* |
| g | 109 | 15 | 15 | 1270 | 358 |
| h | 187 | 58 | 58 | 163 | *-7* |
| i | 153 | 45 | 44 | 688 | 239 |
| j | *3139* | 86 | *3580* | 183 | 4 |
| k | 466 | 207 | 194 | 1011 | 326 |
| l | 461 | 155 | 65 | 703 | 216 |
| m | 199 | -20 | 2 | 456 | 83 |
| n | 322 | 46 | 127 | 508 | 174 |
| N1 | 320 | 68 | 69 | 325 | 102 |
| N2 | *28447* | *14659* | *12901* | *7437* | *4490* |
| N3 | *28271* | *13271* | *11597* | *8317* | *5002* |
| N4 | *31227* | *15271* | *13318* | *10001* | *6035* |
| N5 | *26312* | *12809* | *10930* | *8683* | *5389* |
| N6 | *17943* | *6893* | *6194* | *58723* | *29903* |
| N7 | *4398* | *1610* | *1390* | *87812* | *37258* |
| N10 | 350 | 81 | 36 | 144 | 22 |
| N11 | 372 | 70 | 41 | 390 | 120 |
| N12 | 352 | 82 | 36 | 209 | 35 |
| N13 | *19474* | *8098* | *7726* | *9346* | *5772* |
| N14 | *7833* | *3795* | *3442* | *958* | 567 |
| N15 | *3685* | *1802* | *1506* | *2103* | *965* |
| N16 | 553 | 209 | 146 | 614 | 267 |
| N17 | 510 | 141 | 65 | 133 | 16 |
| N18 | 982 | 351 | 289 | no data | no data |

Table 3.12: This table shows (in percent) for every subtest which SUN and IBM workstation is faster or slower as the reference machine Macintosh IIfx, i.e. 100%. Extremely high values are set in italic.
For the configuration of the used compilers on the SUN and IBM workstations refer to chap. 2.1.

The data series of tables 3.11 and 3.12 are now condensed to mean values, standard deviations, minima, and maxima. Tables 3.13 and 3.14 show these values for each machine. The information of tables 3.11 and 3.12 are represented as box plots in figures 3.10a and 3.10b, too.

| | Quadra 700 | Quadra 950 | PowerPC 8100/80 | PowerBook 520 and 540c |
|---|---|---|---|---|
| Mean | 81 | 140 | 332 | 77 |
| Std Deviation | 36 | 46 | 627 | 58 |
| Min | 16 | 58 | -46 | 6 |
| Max | 141 | 219 | 2799 | 194 |

Table 3.13: The mean value, standard deviation, minimum and maximum value extracted from table 3.11 of all subtests for the Macintosh computers related to 100% for the Macintosh IIfx.
The used configuration is (cf. chap. 2.1, tab. 3.4 and 3.5) Sane on/Compile/DMMathLib. Also, for these tests, the PowerPC 8100/80 had included the software emulation SoftwareFPU V3.02 (cf. tab. 3.7, 3.8, and fig. 3.4).

| | SPARCsta–tion 10 | SPARCser–ver MP 630 | SPARCsta–tion ipx | IBM(Pen–tium) | IBM (486) |
|---|---|---|---|---|---|
| Mean | 7667 | 3355 | 2991 | 8065 | 3866 |
| Std Deviation | 11606 | 5331 | 4540 | 18419 | 8350 |
| Min | 109 | -20 | 2 | 111 | -7 |
| Max | 33299 | 15271 | 13318 | 87812 | 37258 |

Table 3.14: The mean value, standard deviation, minimum and maximum value (extracted from tab. 3.12) of SUN and IBM workstations related to 100% for the Macintosh IIfx.
For the configuration of the used compilers on the SUN and IBM workstations refer to chap. 2.1.

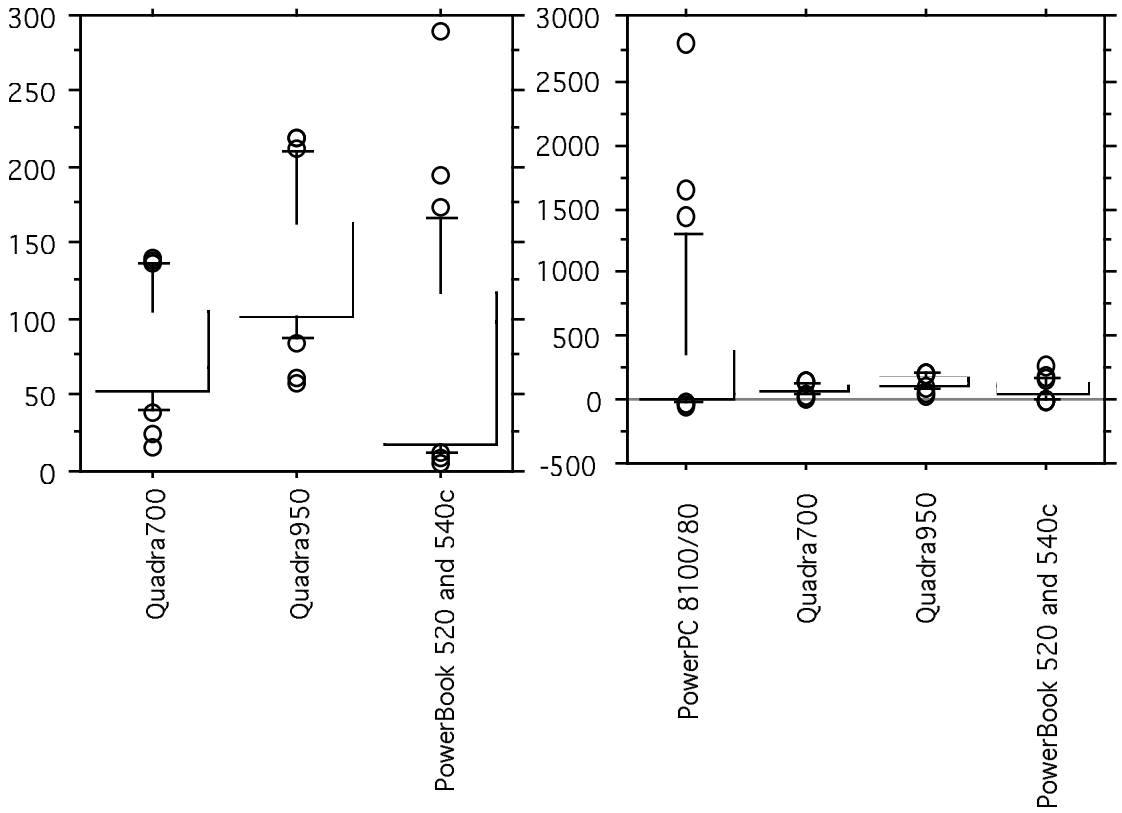


Figure 3.10a: The data series of the tables 3.13 and 3.14, and therefore the extracted data of the tables 3.10 and 3.11, are represented as box plots.
The used configuration is (cf. chap. 2.1, tab. 3.4 and 3.5) Sane on/Compile/DMMathLib. Also, for these tests, the PowerPC 8100/80 had included the software emulation SoftwareFPU V3.02 (cf. tab. 3.7, 3.8, and fig. 3.4).

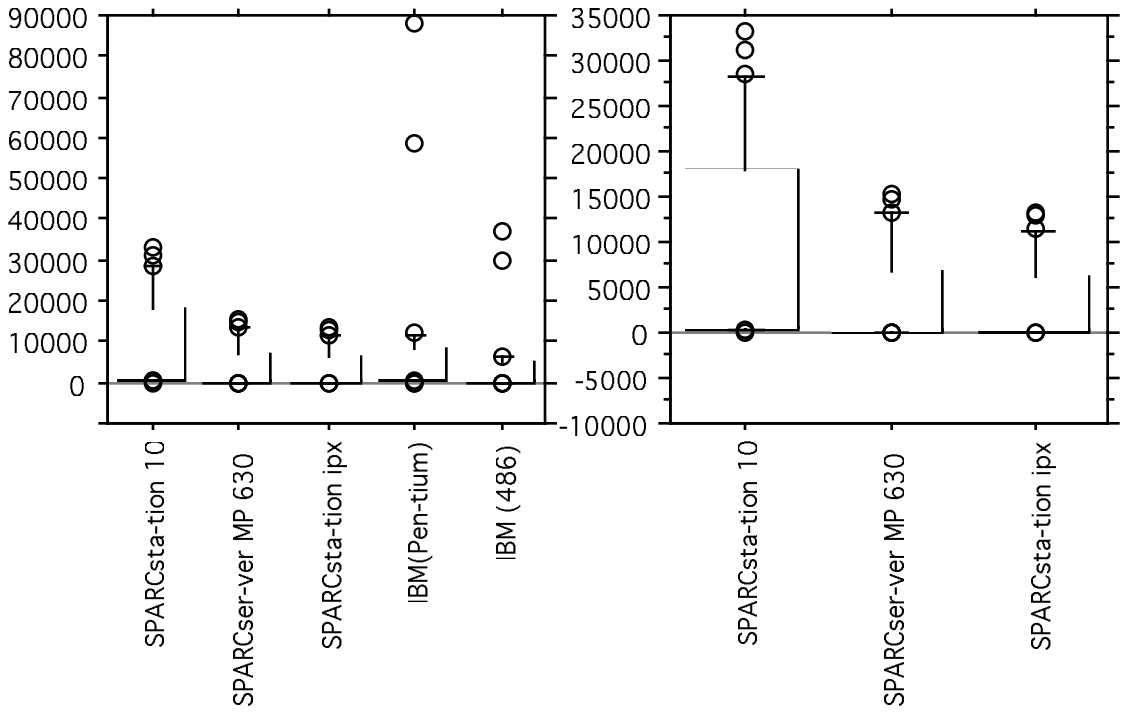


Figure 3.10b: The data series of tables 3.13 and 3.14, and therefore the extracted data of tables 3.10 and 3.11, are represented as box plots.
For the configuration of the used compilers on the SUN and IBM workstations refer to chap. 2.1.

Third, on the base of the values given in tables 3.11 and 3.12 we compute the mean, standard deviation, maximum, and minimum along *all* tested machines. These values are shown in table 3.15. Figures 3.11 to 3.14 show the information contained in this table as box plots for the several machines.

| Test no. | Mean | Std. Deviation | Min | Max |
|---|---|---|---|---|
| a | 225 | 247 | -12 | 777 |
| b | 193 | 175 | -13 | 500 |
| c | 163 | 148 | -14 | 456 |
| d | 146 | 120 | 6 | 428 |
| e | 8406 | 10865 | 28 | 33299 |
| f | 6795 | 8427 | 12 | 25867 |
| g | 221 | 408 | -13 | 1270 |
| h | 66 | 68 | -7 | 187 |
| i | 159 | 210 | 21 | 688 |
| j | 797 | 1458 | 4 | 3580 |
| k | 278 | 308 | 22 | 1011 |
| l | 224 | 219 | 24 | 703 |
| m | 135 | 144 | -20 | 456 |
| n | 174 | 153 | 35 | 508 |
| o | 207 | 233 | 9 | 712 |
| N1 | 125 | 117 | -3 | 325 |
| N2 | 7597 | 9661 | 14 | 28447 |
| N3 | 7445 | 9389 | 18 | 28271 |
| N4 | 8477 | 10461 | 13 | 31227 |
| N5 | 7184 | 8774 | 16 | 26312 |
| N6 | 13356 | 19828 | 22 | 58723 |
| N7 | 14748 | 29928 | 6 | 87812 |
| N8 | 4101 | 7225 | 10 | 17938 |
| N9 | 1872 | 2960 | 20 | 7253 |
| N10 | 99 | 104 | 1 | 350 |
| N11 | 143 | 141 | 1 | 390 |
| N12 | 105 | 111 | 1 | 352 |
| N13 | 5651 | 6498 | 13 | 19474 |
| N14 | 1905 | 2652 | 36 | 7833 |
| N15 | 1165 | 1239 | 9 | 3684 |
| N16 | 244 | 205 | 19 | 614 |
| N17 | 134 | 149 | 16 | 510 |
| N18 | 254 | 349 | -18 | 982 |

Table 3.15: On the base of the values given in tables 3.11 and 3.12 the mean, standard deviation, maximum and minimum of along *all* tested machines are represented in this table.

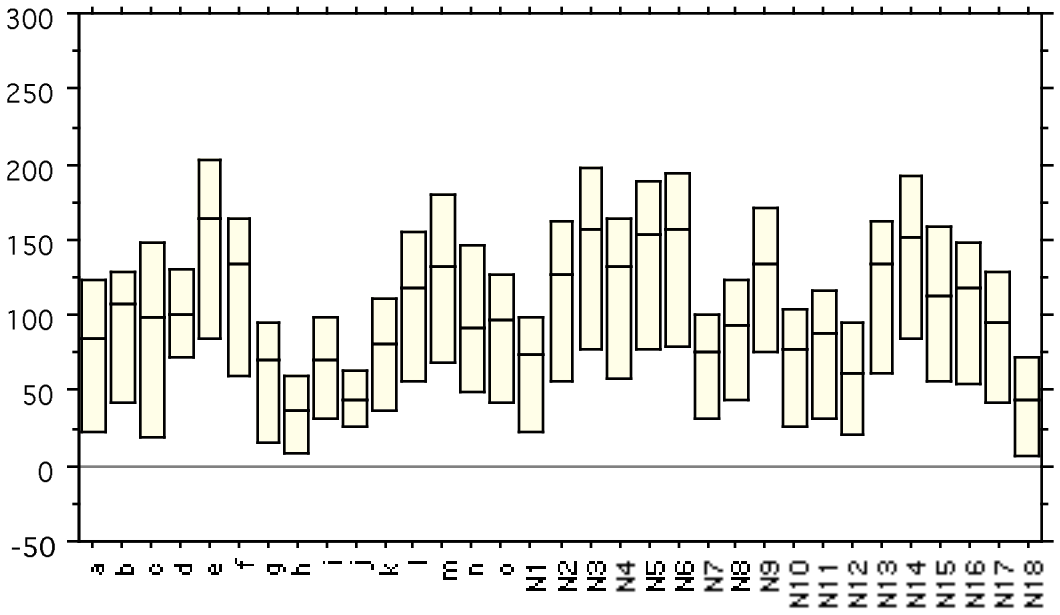


Figure 3.11: This figure shows the mean, standard deviation, maximum, and minimum of the values in table 3.11. The box plots represent only the mean, standard deviation, maximum, and minimum from the tested Macintosh computers. Note, for these tests, the PowerPC 8100/80 had included the software emulation SoftwareFPU V3.02 (cf. fig. 3.4).

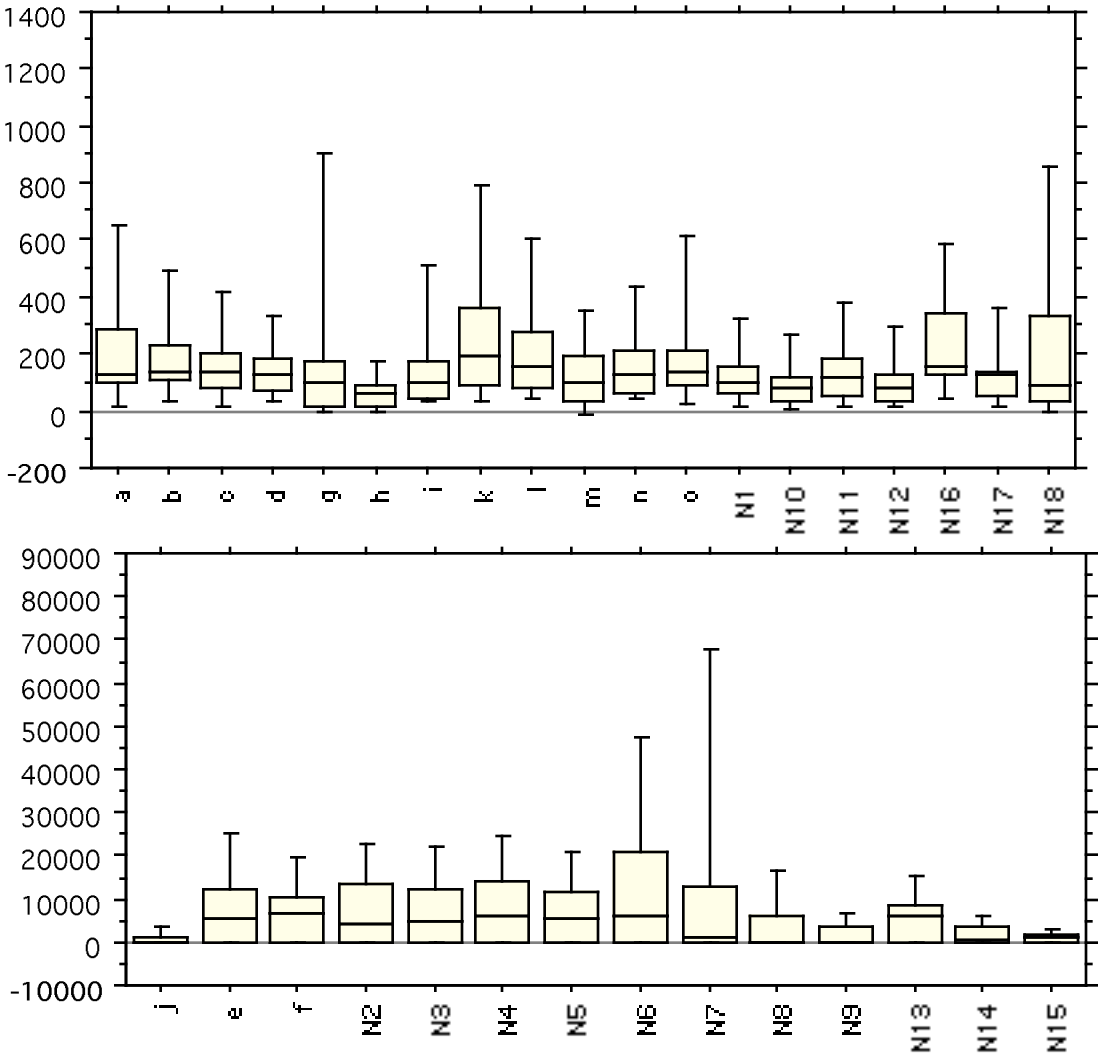


Figure 3.12: On the base of the values given in tables 3.11 and 3.12 the mean, standard deviation, maximum, and minimum of along *all* tested machines (cf. tab. 3.14) are represented as box plots. Note, for these tests, the PowerPC 8100/80 had included the software emulation SoftwareFPU V3.02 (cf. fig. 3.4).

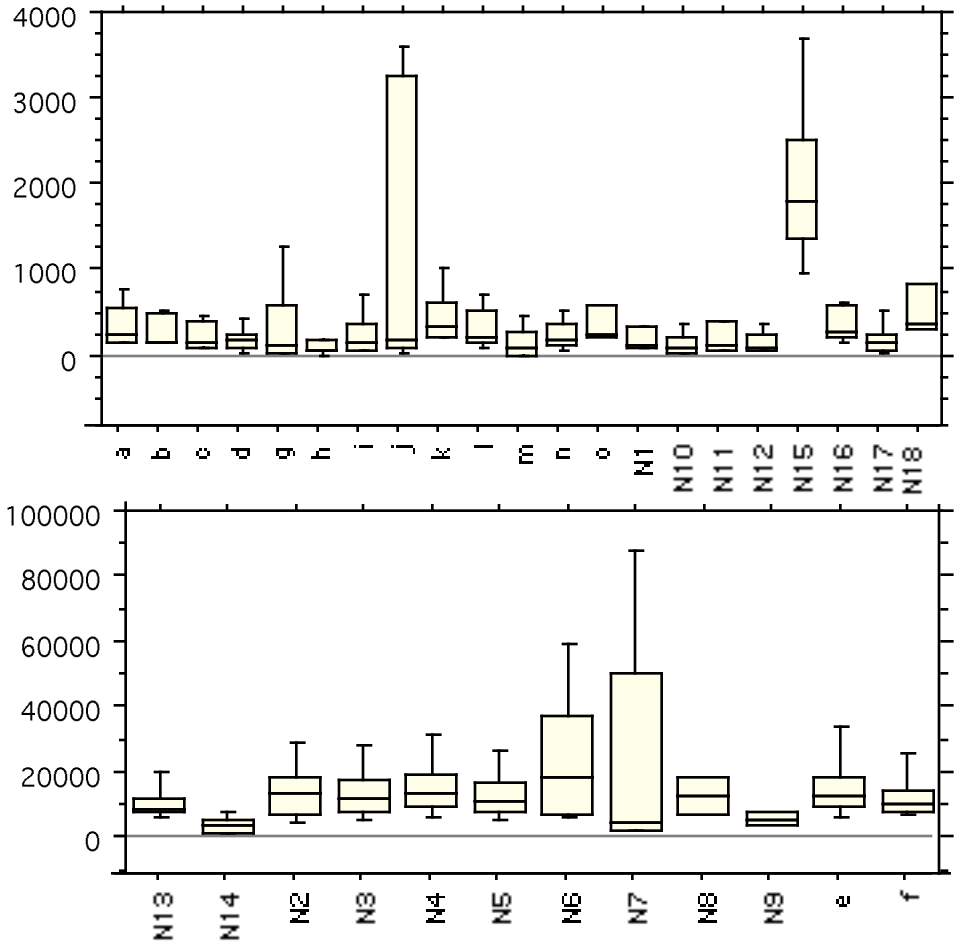


Figure 3.13: This figure shows the mean, standard deviation, maximum, and minimum of the values in table 3.12 as box plots of the tested SUN and IBM workstations.

In order to complete the tests, we examined also the influence of seceral options (SANE on/off, using Compile/Compile20 of MacMETH and the library DMMathLib/DMMathLibF from the Dialog Machine V2.2) on the performance of the PowerPC 8100/80 which used the software emulation PowerFPU V1.01 (cf. tab. 3.15). Table 3.15 shows that for these different combinations the run-time behaviour is nearly the same.
Refer to tables 3.3 to 3.5 for the equivalent examinations on a Quadra 950.

| | | DMMathLib | | DMMathlibF |
|---|---|---|---|---|
| | | Compile20 | Compile20 | Compile20 |
| | | on | off | on |
| SANE | on | 0.6 | **0** | 1.7 |
| SANE | off | 0.9 | 0.1 | 2.3 |

Table 3.16: Results of different options SANE on/off, Compile20 on/off, and DMMathLib or the fast version DMMathLibF for the benchmark tests on the PowerPC 8100/80 with the software emulation PowerFPU V1.01. These results are the mean value of the subtests in percentage difference (cf. eq. 3.2) of the measurements. The differences are small for the different options.
Refer to tables 3.3 to 3.5 for the data of the equivalent examination on a Quadra 950.

Seventh, we compare runs of the Modula-2 version vs. the C version of the benchmark program by using eq. 3.1. Table 3.17 and figure 3.14 show how much faster or slower the tested compilers on the 4 machines are related to the slowest run which is referenced as 100%. Note, first, on the different platforms there were used different Modula-2 and C compilers. Second, for the Modula-2 benchmark program the options SANE off/Compile20/DMMathLibF are used (cf. tab. 3.4 and 3.5). Third, the C and Modula-2 benchmark programs are implemented in different ways (cf. chap. 2.2). For information about the used compiler options refer to chapter 2.1.
Table 3.18 gives a more detailed view to the most subtests of the C benchmark run. The result of the Modula-2 benchmark is included to get an impression of possible differences.

| PowerPC 8100/80 | Quadra 700 | SPARCstation ipx | SPARCstation 10 |
|---|---|---|---|
| 142 | 154 | 227 | 414 |

Table 3.17: Results of the Modula-2 version of the benchmark program vs. the C version. This table shows four (independent) pairs of mean values of all subtests given in percent (cf. eq. 3.1) to the result of the Modula-2 benchmark run which is chosen as the reference value of 100%. For information about the used compiler options refer to chapter 2.1.
For the Macintosh computers the Modula-2 compilation was performed by the combination SANE off/Compile20/MathLibF (cf. tab. 3.4 and 3.5).

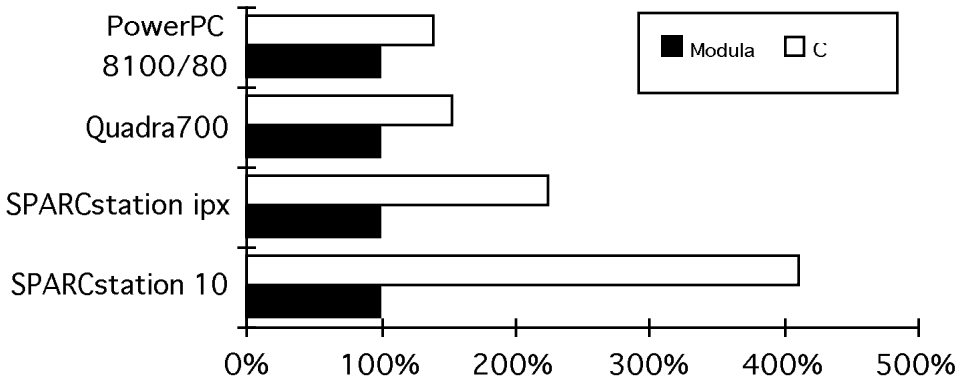


Figure 3.14: Results of the Modula-2 version of the benchmark program vs. the C version. This figure shows four pairs of (independent) mean values of all subtests given in percent (cf. eq. 3.1) to the result of the Modula-2 benchmark run which is chosen as the reference value of 100%. Note, on the different platforms different Modula-2 and C compilers were used. For information about the used compiler options refer to chapter 2.1.
For the Macintosh computers the Modula-2 compilation was performed by the combination SANE off/Compile20/MathLibF (cf. tab. 3.4 and 3.5).

| Test no. | Modula-2 | C | C vs. Modula-2 |
|---|---|---|---|
| | No. of iterations | No. of iterations | Difference in per cent |
| a | 396288000 | 1322166148 | 234 |
| b | 330191264 | 1322084667 | 300 |
| c | 282868512 | 1307158613 | 362 |
| d | 66664000 | 11608000 | -83 |
| e | 79921688 | 1319999987 | 1552 |
| f | 3214200 | 6802400 | 112 |
| g | 41767068 | 237168000 | 468 |
| i | 3873 | 32259 | 733 |
| k | 158080016 | 495807228 | 214 |
| l | 58701196 | 360736000 | 515 |
| m | 584000 | 712350 | 22 |
| n | 3007000 | 7552000 | 151 |
| N1 | 397196800 | 1320070646 | 232 |
| N2 | 19879518 | 26474600 | 33 |
| N3 | 20577290 | 28328400 | 38 |
| N4 | 14991968 | 18693625 | 25 |
| N5 | 18183666 | 23906800 | 32 |
| N6 | 9009601 | 43316800 | 381 |
| N7 | 13825099 | 49800796 | 260 |
| N8 | 200159360 | 41666400 | -79 |
| N9 | 236671984 | 35369721 | -85 |
| N10 | 201360000 | 1322708265 | 557 |
| N11 | 4546613 | 49664000 | 992 |
| N12 | 18512880 | 1322611726 | 7044 |
| N13 | 61999380 | 1711520 | -97 |
| N14 | 67732664 | 1714080 | -98 |
| N15 | 118640000 | 68408000 | -42 |
| N16 | 631872 | 306465863 | 48401 |

Table 3.18: This table shows the results of the benchmark runs C vs. Modula-2 on the SUN SPARCstation ipx for the most subtests. The first two columns contains the absolute measured no. of iteration within 100 seconds. The third column shows whether C is faster (positive values) or slower (negative values) as percentage difference to the C run. The values of the third column are computed by eq. 3.2 with the Modula-2 benchmark run as the reference.
Note, the C and Modula-2 benchmark programs are partly implemented in different kinds (cf. chap. 2.2). For information about the used compiler options refer to chapter 2.1.

Table 3.19 contains information about several differences of the run-time behaviour of program constructions in the programming language C as e.g. of different loop constructions etc. In each group we used eq. 3.1 and refer to the slowest run as 100%. For the comparison of one dimensional array versus multi-dimensional array access and array vs. list access the different implementations related to the assignments of variables of these subtests has to be taken into account. Refer to tab. 3.2 which contains the same information for the Modula-2 benchmark test.

| | | *type conversion* | ***II*** | *loops* | ***III*** |
|---|---|---|---|---|---|
| | | N12 | 407 | a | 101 |
| | | N11 | 100 | b | 101 |
| | | N7 | 304 | c | 100 |
| | | *built-in function* `++i` | ***V*** | *integer vs. real arithmetic* | ***VI*** |
| | | a | 100 | d | 100 |
| | | N1 | 100 | e | 120 |
| *procedure calls* | ***VII*** | *1-D array vs. 2-D array* | ***VIII*** | *array vs. pointer* | ***IX*** |
| k | 269 | g | 108 | g | 1389 |
| l | 100 | i | 100 | n | 100 |

Table 3.19: This table contains in six boxes **II, III,** and **V** to **IX** information about several differences of the run-time behaviour of program constructions (in the programming language C) in percent (cf. eq. 3.1) to the slowest run in each group which is the reference of, i.e. 100%. These data are extracted from tab. 3.18.
We refer to tab. 3.2 which contains the same type of data for the Modula-2 benchmark test.

## 3.2 The Benchmark Tests by the Application Speedometer

Since we do not know how the benchmark tests in Speedometer V4.02 are implemented, the results obtained by these tests can not be taken too seriously. For example, we do not know in which programming language Speedometer V4.02 is written, neither do we know by which compiler and therefore by which compiler options the source code was compiled and linked. Additionally, it is also unknown how the algorithms in Speedometer are implemented. All these points can have a crucial influence of the performance of an application (Löffler, 1995).

The results produced by Speedometer V4.02 can be helpful in order to complete the overview of the performance of the tested Macintosh computers. Although this additional information generated by Speedometer has to be used cautiously, it can give hints and relative information, e.g. which machines show a better performance than in the test by our benchmark program.

The benchmark test by Speedometer V4.02 from Scott Berlfield are summarized in tables 3.20 and 3.21. Note, the reference machine for Speedometer is the Quadra 650 which performance is assumed to be 1.0.

| Benchmark Mix | IIfx | Quadra 700 | Quadra 950 | PowerPC 8100/80 68K code | PowerPC 8100/80 PowerPC code |
|---|---|---|---|---|---|
| KWhetstones | 0.501 | 3.965 | 5.220 | 5.797 | 135.042 |
| Dhrystones | 0.374 | 1.056 | 1.429 | 0.542 | 4.330 |
| Towers | 0.439 | 0.996 | 1.345 | 0.639 | 5.383 |
| Quick Sort | 0.542 | 0.972 | 1.318 | 0.562 | 6.584 |
| Bubble Sort | 0.491 | 0.982 | 1.327 | 0.384 | 4.115 |
| Queens | 0.471 | 0.961 | 1.299 | 0.566 | 4.329 |
| Puzzle | 0.390 | 0.990 | 1.347 | 0.613 | 5.734 |
| Permutations | 0.380 | 0.961 | 1.298 | 0.662 | 5.901 |
| Int. Matrix | 0.408 | 0.966 | 1.319 | 0.757 | 6.534 |
| Sieve | 0.614 | 0.936 | 1.238 | 0.469 | 4.456 |
| Average | 0.461 | 1.278 | 1.714 | 1.099 | 18.241 |

Table 3.20: The benchmark tests "Benchmark Mix" examined with the application Speedometer V4.02. The results are given in relation to the Quadra 650 which performance is assumed as 1.0. For the PowerPC, the application Speedometer V4.02 can test both, the behaviour of the PowerPC for 68K code and for native code.

| FPU Benchmarks | IIfx | Quadra 700 | Quadra 950 | PowerPC 8100/80 PowerPC code |
|---|---|---|---|---|
| KWhetstones | 0.371 | 0.760 | 1.027 | 7.695 |
| Matrix Mult. | 0.218 | 0.722 | 0.971 | 8.347 |
| Fast Fourier | 0.187 | 0.714 | 0.973 | 7.205 |
| Average | 0.259 | 0.732 | 0.990 | 7.749 |

Table 3.21: The benchmark tests "FPU Benchmarks" examined with the application Speedometer V4.02. The results are given in relation to the Quadra 650 which performance is assumed as 1.0. For the PowerPC, the application Speedometer V4.02 can only test the behaviour of the PowerPC for native code.

# 4. Summarize

We remind that the used Modula-2 compiler MacMETH V3.2.2 (Wirth *et al.*, 1992) is not able to generate native code for the RISC processors used by the PowerPC family. Hence, all results produced on the tested PowerPC 8100/80 do neither reflect the true abilities of this machine nor of the PowerPC computer family in general.

In the following we briefly summarize the new benchmark results without any interpretation. First we extract answers for the different test conditions of chap. 2.3. For the description of the used machines, compilers, and compiler options refer to chap. 2.1.

For mathematical operations the results are:

- Not using SANE for Macintosh computers increases the mean speed by approximately 40% (cf. tab. 3.4 and 3.3). More precisely, the range of the increase for *mathematical operations* lies between 1% to 180%, dependent on the particular mathematical operation (cf. tab. 3.4, and 3.5).

- Using Compile20 increases for Macintosh computers the mean speed by approximately 60% (cf. tab. 3.4 and 3.3). More precisely, the range of the increase for *mathematical operations* lies between 12% to 1700%, dependent on the particular mathematical operation (cf. tab. 3.4, and 3.5).

- Using the optimised library DMMathLibF combined with Compile20 increases the mean speed by about 500% (cf. tab. 3.4). More precisely, the range of the increase for *mathematical operations* lies between 0% to 4000%, dependent on the particular mathematical operation (cf. tab. 3.4, and 3.5).

For details refer to tables 3.3, 3.4, and 3.5.

For the influence of a network and of extensions (cf. chap. 2.3) to a Macintosh the results are:

- Using no extensions for Macintosh computers yields a mean increase in speed of approximately 400%. Thereby, the gain in the run-time performance is contributed mainly from the mathematical subtests (cf. tab. 3.6 and fig. 3.1).

- Not to be connected to a network yields for Macintosh computers a mean increase in speed of approximately 30%. Thereby, the gain in the run-time performance is contributed mainly from the mathematical subtests (cf. tab. 3.6 and fig. 3.2).

- Contrary, the benchmark for using no extensions and being unconnected to a network *didn't bring any* measurable effect for Macintosh computers (cf. tab. 3.6 and fig. 3.3).

For details refer to table 3.6 and fig 3.1, 3.2, and 3.3. Note, that these results are interpretations of the measurements based on the equations given in chap. 2.5.

For running *non-native* applications on a PowerPC 8100/80 with the software emulations SoftwareFPU V3.02 and V3.03 and PowerPFU V1.01, the results are:

- For the PowerPC 8100/80, the FPU emulation SoftwareFPU V3.02 and V3.03 (share-ware) allows to run applications which need a FPU but yields no advantage in speed (cf. tab. 3.7 and fig. 3.4).

- The FPU emulation PowerFPU V1.01 yields additionally a mean increase in speed of about 96%. That means PowerFPU V1.01 doubles the speed of a PowerPC 8100/80 for *non-native* applications which needs a FPU (cf. tab. 3.7 and fig. 3.4).
  Thereby, the gain in the run-time performance is caused mainly by the mathematical subtests (cf. fig. 3.4).

- On the PowerPC 8100/80 with the software emulation PowerFPU V1.01, the mathematical options SANE on/off, Compile20 on/off, and using DMMathLib or the fast version DMMathLibF yield mainly the same run-time behaviour.

- In this *special* configuration, i.e. running *non-native* code and using the software emulation PowerFPU V1.01, the PowerPC 8100/80 has an *average* performance such as the tested Quadra 700.

- On a PowerPC 8100/80 with the software emulation PowerFPU V1.01, the options SANE on/off, using Compile/Compile20 of MacMETH and the library DMMathLib/DMMathLibF from the Dialog Machine V2.2 do not differ much (cf. tab. 3.15).

For details refer to tables 3.7, 3.8, 3.16, and figure 3.4.

Again we remember, these results are interpretations of the measurements based on the equations given in chap. 2.5.
The benchmark test by the application Speedometer V4.02 (cf. chap. 3.2, tab. 3.20, and 3.21) show the big difference of running applications which are generated for the Motorola 68K CISC processors vs. running applications in native code on the PowerPC 8100/80.
Also new benchmark examinations (cf. (Meyer *et al.*, 1996)) show that the performance of the PowerPC machines for integer and floating point operations are in the same range as Pentium workstations.

The comparison of the tested different machines yields:

- The speed of Macintosh computers (from IIfx over Quadra 700 to Quadra 950) increases more or less linearly with all tested machines (cf. tab. 3.9, 3.10, and fig. 3.5).

- The SPARCstation 10 is double as fast as the SPARCstation ipx (cf. tab. 3.10 and fig. 3.5) which speed lies in the range of the SPARCserver MP630.

- The tested IBM workstations can compete with the tested SUN workstations (cf. tab. 3.9, 3.10, and fig. 3.5). Note, these tested workstations are not the latest ones. Therefore, we can not make statements about e.g. Intel Pentium computers and SUN UltraSparc machines.

- The SPARCserver MP630 with two processors is about a factor 1.2 faster than the SPARCstation ipx (cf. tab. 3.10 and fig. 3.5).

- The IBM workstations with Pentium processor is about a factor 2 faster than that one with the 80486/80487 chip (cf. tab. 3.10 and fig. 3.5).

For details refer to tables 3.9, 3.10, and figure 3.5.

On the Macintosh computers, the comparison of language elements, compiler options etc. of the Modula-2 compiler MacMETH V 3.2.2 and the C compiler MPW C V3.3 yields:

- Equivalent language elements, such as several loop constructions, can have different run-time behaviour.
  The various loop constructions in Modula-2 (MacMETH) differ in speed within a range of 22% to 58%. The C benchmark tests (MWP C) yield for different loop construction nearly the same run time behaviour (cf. tab. 3.2 and 3.19).

- Built-in functions, such as *Inc()* and *Dec()* in Modula-2, can be faster than the normal constructions.
  The built-in functions *Inc()* and *Dec()* in Modula-2 (MacMETH) are about 30%

faster than the normal constructions *i:=i+1* and *i:=i-1*. In the programming language C (MPW C) no difference between the constructions *i++* and *i--* vs. *i=i+1* and *i=i-1*, respectively, can be recognized (cf. tab. 3.2 and 3.19).

- By increasing the parameter list, calls of subprocedures need significantly more run-time.
  In Modula-2 (MacMETH) the difference between calling a subprocedure with and without a parameter list of four integers (call by value) is about 24%. In the programming language C (MPW C) the difference was measured as 169%.

- An algorithm can be significantly slowed down by checking ranges of arrays, pointers etc.
  For the compiler MacMETH checking ranges etc. decrease the speed by about 48% to 72% depending on the operation.

- The access to one-dimensional vs. two-dimensional arrays of same total size can be different.
  In Modula-2 (MacMETH) the access to one-dimensional arrays was measured as about 37% faster. In the programming language C (MPW C) this difference was 8%.

- The access to one-dimensional arrays vs. lists via pointer can be significantly different. For Modula-2 (MacMETH) the access to arrays was measured as about 1108% faster. For C (MPW C) this difference was about 1289%.

- Reading integer disk streams was in Modula-2 (MacMETH) about 52% faster than reading real disk streams.

For details refer to tables 3.2, 3.18, and 3.19 and figure 3.14, but also table 3.1.

These comparisons between the used compilers of Modula-2 and C and of equivalent language constructions of Modula-2 and C, such as e.g. different loops, can not be taken as very strong results. Also comparisions between the run-time behaviour of e.g. the loop constructions of Modula-2 and C have to be made carefully.
For instance, we do not know the implementation of the compilers and hence the influence of the compiler options to applications can not exactely be taken into consideration (cf. chap. 2.1). Therefore, we did not test standardized compilers but ones in their typical user mode.
This comparison of the different compilers or programming languages yields:

- In average over all subtests (cf. tab. 2.5 and 2.6), the tested C compilers seems to generate code which runs by a factor 1.5 to 4 faster than the comparable code of the tested Modula-2 compilers (cf. tab. 3.18 and fig. 3.14).

# 5. Discussion

We remind that the used Modula-2 compiler MacMETH V3.2.2 (Wirth *et al.*, 1992) is not able to generate native code for the RISC processors used by the PowerPC family. Hence, all results produced on the tested PowerPC 8100/80 do not reflect neither the true abilities of this machine nor of the PowerPC family in general.
This fact is for example demonstrated by the benchmark test with Speedometer V4.02 (cf.

chap. 3.2, tab. 3.20 and 3.21) and also by comparisons between the PowerPC and Pentium machines (Meyer *et al.*, 1996).

Generally, it has to be in mind that the results of benchmark tests depend

- strongly on the used configurations, such as e.g. the used compilers and compiler options, the produced binaries in native code or not, the definition and implementation of the benchmark tests etc.
- on the configuration of the machine, such as the connection to a network etc.
- and on the decade at which the machines were tested, since, e.g. a formerly fast machine can at a later time be one of the slowest tested computers.

Therefore, interpretations of benchmark results has to be made safely and relatively with respect to these facts.
The worth of benchmarks do not lie in strong and absolute results, but in tips and hints for special purposes, in an overview which can be extracted from the produced data, in a summary of data which are collected over years, and in getting an idea of what an user can expect from a machine in every day use.

The benchmark examinations presented in this paper did not yield big surprises, except the results for the PowerPC 8100/80 which therefore needs a special analyse and except the results influence of a connection to a network or using extensions (cf. chap. 2.3) by a Macintosh computer.

For the overview in tables 3.9 and 3.10 and figure 3.5 of the average over the measured performance of the tested machines it has to be in mind that the benchmark programs were compiled and linked with different compilers (cf. chap. 2.1).
Nevertheless, by this overview of the tested platforms we obtain an impression of the performances with which one probably has to deal. Relative to the group of machine they belongs to, the SUN workstations, IBM, and Macintosh computers show an expected performance. So, for example, it was expected that a Pentium, SPARCstation 10, and Quadra 950 are faster than the DX486, SPARCstation ipx, and Macintosh IIfx, respectively.
Also, a comparison of the groups SUN, IBM, and Macintosh computers does not yield a surprise. The Pentium machine can compete with the tested SUN workstations, but these are not the latest available SUN workstation such as e.g. the UltraSparc. The Quadra 950 can compete with the DX486 machine as well with the SUNstation ipx and SPARCserver MP630.
For the PowerPC 8100/80 the results show that this is clearly a fast machine (cf. tab. 3.20 and 3.21). But by executing 68K binaries, this good performance of the PowerPC 8100/80 is not of use, since these binaries are not native code for this RISC machine.
With the software emulation PowerFPU V1.01 the speed of mathematical operations can significantly be increased (cf. tab. 3.7 and fig. 3.4). Therefore, the measured performance of the PowerPC 8100/80 *in this special configuration (executing 68K binaries and using PowerFPU V1.01)* lies in the range of the other tested Macintosh computers.

Hence, for the PowerPC has to be taken into consideration that

- the Modula-2 compiler MacMETH V 3.2.2 can not generate binaries for RISC architecture.
- therefore an software emulation has to be used in order to reduce this lack.

Contrary to other MacIntosh machines, the choice of various options (SANE on/off, Compile/Compile20, DMMathLib/DMMathLibF) did not have an influence on the performance of a PowerPC 8100/80 with the software emulation PowerFPU V1.01. This is no suprise, since a software emulation can not replace a mathematical co-processor. Nevertheless, in average, small differences were recognizable.

As the test with Speedometer V4.02 shows (cf. tab. 3.20 and 3.21), for taking advantage of the RISC processor architecture of the PowerPC 8100/80 native code application are necessary. By tables 3.20 and 3.21 it is *expectable* that for native code applications the overall performance of the PowerPC 8100/80 can at least compete with the SUN and IBM workstations.

Table 3.6 and figure 3.1 show that extensions (cf. chap. 2.3) used by a Macintosh computer influence the performance. Surprisingly, this influence seems to be strong only for the mathematical subtests (cf. tab. 2.5 and 2.6), but not for e.g. matrix access (subtest i).
A similar behaviour, but not such clearly, can be recognized if the tested machine is not connected to the network (tab. 3.6 and fig. 3.2).
Obviously, network *or* extensions use large parts of the processors capacity, as expected.

Astonishing was the result of testing the same Macintosh machine which was unconnected to a network *and* did not use extensions. No difference between using and not using this configuration can be recognized (tab. 3.6 and fig. 3.3). We repeated this experiment several times with the same result.
We have no explanation for this measurement beside the one, that the assumptions of chap. 2.5 for sensitive tests are not sufficient for some tests.

The benchmark results for the Macintosh computers using MacMETH (Wirth *et al.*, 1992) with the three possibilities (**1**) SANE on/off, (**2**) using Compile or Compile20, and (**3**) using DMMathLib or DMMathLibF (cf. chap. 2.1, 2.3, and 3.1) can easily be interpreted.

The SANE software library performs a 64 bit arithmetic, whereas without this library only the available 32 bit arithmetic can be used. Since the 64 bit arithmetic is available by the SANE library, the 64 bit arithmetic is slower than the 32 bit arithmetic but has the better accuracy and vice versa. For details of this subject refer to (Apple Computer, 1986; Apple Computer, 1993a; Löffler, 1995).
In average, not using the SANE library increase the run-time of binaries by about 40%. If a mathematical coprocessor is available in the Macintosh computer, then using the optimized MacMETH compiler Compile20 instead of Compile additionally increase the run-time of applications by an additional 60%. Therefore, not using SANE together with compiling by Compile20, binaries run about 100% faster on Macintosh computers with a mathematical coprocessor than not using these optimizations. By this simple optimization, binaries which are produced by the compiler MacMETH can benefit by a factor of about two.
Additionally, using the optimized mathematical library DMMathLibF instead of DMMathLib (Fischlin *et al.*, 1994) increase the run-time of applications by about 400%.
Therefore, using the three optimzations on a Macintosh machine on which a mathematical coprocessor is available, binaries which are generated with the MacMETH can benefit by a factor of about five.
These results show the importance of using optimized software libraries and of knowing the effects of the used compiler options in order to write and generate fast applications (Löffler, 1995).

Programmers should always have in mind that different elements of programming languages which are used in order to solve a special problem can strongly influence the perfor-

mance of an application. This is demonstrated by the different performances of constructions of programming languages, such as e.g. several loop constructions or reading integer or real disk streams (tab. 3.2 and 3.19). For this subject refer to e.g. (Löffler, 1995).

The comparison between Modula-2 and C can not give strong results. The main perspective was to get answers about the overall performance of different programming languages and impressions of the differences which can occur if people use the compiler in their typical user mode (cf. tab. 3.18 and fig. 3.14). For the used compiler options refer to chap. 2.1. Nevertheless, first, these results give an impression how different equivalent constructions of a programming language such as loops can be and that such differences have not to be in the same way different for several languages or compilers. Second, since the effects of compiler options are not always described in details, differences in the run-time behaviour of e.g. loop constructions in different programming languages can unexpectedly be the result of such compiler options. Third, these results show how essential it can be to carefully choose the constructions of a programming language for the applications which people want to develop (Löffler, 1995).

References

Apple Computer, I., 1986. *APPLE Numerics Manual*. Addison-Wesley, 336 pp.

Apple Computer, I., 1993a. *Building and Managing Programs in MPW, For MPW V 3.3*. Developer Technical Publications.

Apple Computer, I., 1993b. *Macintosh Programmer's Workshop C Reference*. Developer Technical Publications.

Borland International, I., 1992. *Borland Pascal With Objects*.

Bugmann, H., 1994. *On the ecology of mountainous forests in a changing climate: a simulation study*. Diss. ETH No. 10638, Swiss Federal Institute of Technology: Zürich, Switzerland, pp. 258.

Bugmann, H. & Fischlin, A., 1994. *Comparing the behaviour of mountainous forest succession models in a changing climate*. In: Beniston, M. (ed.) *Mountain environments in changing climates*, London: Routledge, pp. 237-255.

EPC, 1991. *EPC Modula-2 User's Reference Manual.* Edinburgh Portable Compilers Limited.

Fischlin, A., 1986. Simplifying the usage and the programming of modern working stations with Modula-2: The Dialog Machine. *Dept. of Automatic Control and Industrial Electronics, Swiss Federal Institute of Technology Zurich (ETHZ)*:

Fischlin, A., 1991. *Interactive modeling and simulation of environmental systems on workstations.* In: Möller, D.P.F. (ed.) *Analysis of Dynamic Systems in Medizine*, Ebernburg, Bad Münster am Stein-Ebernburg, BRD: Springer: Berlin a.o., pp. 131-145.

Fischlin, A. *et al.*, 1994. *ModelWorks 2.2: An interactive simulation environment for personal computers and workstations.* Systems Ecology Report No. 14, Institute of Terrestrial Ecology, Swiss Federal Institute of Technology ETH, Zurich, Switzerland, 324 pp.

Fischlin, A., Vancso-Polacsek, K. & Itten, A., 1988. *The "Dialog Maschine" V1.0*. Projekt-Zentrum IDA/CELTIA, Swiss Federal Institute of Technology ETH, Zurich, Switzerland, 8 pp.

foundation, G., 1993. *GNU project C and C++ Compiler (v2.4), GCC manual, G++ manual.*

Gilbreath, J., 1981. A High-Level Language Benchmark. *BYTE*:

Gilbreath, J. & Gilbreath, G., 1983. Eratosthenes Revisited, Once More through the Sieve. *BYTE*:

Hinnant, D.F., 1984. Benchmarking Unix Systems. *BYTE*:

Keller, D., 1989. *Introduction to the Dialog Machine*. Bericht Nr. 5, Proijektzentrum IDA, ETH Zürich, 37 pp.

Linton, M.A., 1986. Benchmarking Engineering Workstations. *IEEE DESIGN & TEST*:

Lischke, H., Löffler, T.J. & Fischlin, A., 1996. *Aggregation of Individual Trees and Patches in Forest Succesion Models: Capturing Variability with Height Structured Random Dispersions.* 28, Swiss Federal Institute of Technology, Zürich, Switzerland, 17 pp.

Löffler, T.J., 1995. *How to Write Fast Programs - A Practicle Guide for RAMSES Users.* Systems Ecology Report Institute of Terrestrial Ecology, Swiss Federal Institute of Technology ETH, Zurich, Switzerland.

Löffler, T.J. & Fischlin, A., 1997. *Performance of RAMSES.* Systems Ecology Report Institute of Terrestrial Ecology, Swiss Federal Institute of Technology ETH, Zurich, Switzerland.

Meyer, C., Persson, C., Siering, P. & Stiller, A., 1996. Showdown bei Zwo-Null-Null. *c't*, **11/96**: 270-283.

Thöny, J., Fischlin, A. & Gyalistras, D., 1995. *Introducing RASS - The RAMSES Simulation Server.* Systems Ecology Report No. 21,.

Vancso-Polacsek, K., Fischlin, A. & Schaufelberger, W. (eds.), 1987. *Die Entwicklung interaktiver Modellierungs- und Simulationssoftware mit Modula-2*, *Simulationstechnik*, Berlin, Springer Verlag, Vol. **150**, 239-249 pp.

Wirth, N., 1981. *The Personal Computer Lilith.* Departement für Informatik ETH Zürich, Switzerland, 70 pp.

Wirth, N. *et al.*, 1992. *MacMETH. A fast Modula-2 language system for the Apple Macintosh. User Manual. 4th, completely revised ed.* Departement für Informatik ETH Zürich, Switzerland, 116 pp.

# Appendix

## A. The early benchmark tests

In the last decade, traditional benchmark tests (Gilbreath, 1981; Wirth, 1981; Gilbreath & Gilbreath, 1983; Hinnant, 1984; Linton, 1986) described in chap. 2 as well as benchmarks such as *Sieve*, *Ereal*, *Freal*, *Queens* and *MacQueens* were executed. We give a short description of these five test programs in the following.

**(Sieve)** The program *Sieve* is a procedure for finding prime numbers based on the *Eratosthenes sieve*. In the classic sieve procedure, the natural numbers are arranged in order and then crossed out every *n*th number after reaching the number *n*. The numbers that are not crossed out, which pass through the sieve, are prime numbers.
One remarkable feature of the program is that it avoids multiplication and divisions because these operations are usually slow (Gilbreath, 1981; Gilbreath & Gilbreath, 1983)
*Sieve* does many looping and array subscription and is thus biased strongly toward machine-code compilers. *Sieve* tests compiler efficiency and processor throughput. It's an excellent test for looping, testing and incrementing.

**(Ereal)** The program *Ereal* loops five thousand times through a simple multiplication and division calculation of real numbers. This benchmark test is the same as subtest e (cf. tab. 2.5) of the traditional benchmark test set.

**(Freal)** The program *Freal* calculates the standard functions sin, exp, ln and sqrt using real numbers, five hundred times. This benchmark test is the same as subtest f (cf. tab. 2.5) of the traditional benchmark test set.

**(Queens)** The program *Queens* shall find all possible placement of eight queens on a chess board in such a fashion that none is checking any other piece, i.e. each row, column, and diagonal must contain at most one piece. The program outputs a graphical presentation of the queens position on the chess board as the different solutions are being calculated. *Queens* tests the capability for line drawing of a computer.

**(MacQueens)** The program *MacQueens* is similar to the program*Queens* but displays both the individual solutions, which are already found, and the solution which is currently being calculated. That is, this program requires somewhat more drawing capacity than the *Queens*.

We restrict the view for these early benchmark to a summarization of the measured data.
First we give by the tables A.1, A.2, A3, A4, A5, and A6 an overview of the benchmark test described in chap. 2 with a Modula-2 program. The data are given related to the result measured with the machine Lilith by the equation

$$D_{subtest} = S_{machine} / S_{Lilith}. \qquad \text{(a.1)}$$

| Test No. | Litlith | CompuPro (A) | CompuPro (A) only (*$T-*) | CompuPro (B) only (*$T-*) | CompuPro (C) only (*$T-*) |
|---|---|---|---|---|---|
| a | 321 | 0.55 | 0.46 | 1.33 | 1.10 |
| b | 334 | 0.53 | 0.41 | 1.36 | 1.04 |
| c | 422 | 0.55 | 0.56 | 1.27 | 1.42 |
| d | 187 | 0.27 | 0.23 | 1.24 | 0.95 |
| e | 130 | 1.14 | 1.08 | 1.08 | 0.43 |
| f | 87 | 0.92 | 0.9 | 0.94 | 0.39 |
| g | 109 | 0.55 | 0.39 | 1.33 | 1.03 |
| h | 89 | 0.37 | 0.36 | 0.84 | 0.78 |
| i | 197 | 0.3 | 0.25 | 1.03 | 0.89 |
| j | 164 | 0.27 | 0.26 | 0.79 | 0.83 |
| k | 144 | 0.65 | 0.57 | 1.48 | 1.22 |
| l | 94 | 0.56 | 0.5 | 1.27 | 1.26 |
| m | 63 | 0.73 | 0.7 | 2.4 | 2.35 |
| n | 125 | 0.13 | 0.13 | 0.26 | 0.26 |
| o | 207 | 0.07 | 0.1 | 0.25 | 0.24 |

Table A.1: This table shows the relative number of iterations which could be executed for each subtest in 100 seconds related to the test on Lilith which is referenced as 1 (cf. eq. a.1). For the machine Lilith the absolute values in seconds are given. These benchmark tests are done in Modula-2. The symbol (*$T-*) marks tests which has made no index test (arrays etc.)

| Test No. | CompuPro (D) MP/M 8-16 | PdP-11/23 not optimized | PdP-11/23 optimized | PdP-11/40 | Alto 2 floppy-disk |
|---|---|---|---|---|---|
| a | 0.89 | 1.06 | 2.04 | 0.57 | |
| b | 0.84 | 0.92 | 1.51 | 0.55 | 0.35 |
| c | 0.88 | 0.66 | 0.63 | 0.55 | 0.41 |
| d | 0.43 | 0.44 | | 0.45 | 0.29 |
| e | 1.86 | 0.29 | 0.29 | | |
| f | 1.47 | | | | |
| g | 0.88 | 0.69 | 0.77 | 0.5 | 0.29 |
| h | 0.6 | | | 0.12 | 0.29 |
| i | 0.49 | 0.92 | 0.62 | 0.47 | 0.22 |
| j | 0.44 | | | 0.13 | 0.22 |
| k | 1.05 | 0.84 | 1.26 | 0.26 | 0.28 |
| l | 0.9 | 0.82 | 1.02 | 0.31 | 0.34 |
| m | 1.17 | 0.84 | 0.84 | 0.17 | 0.89 |
| n | 0.21 | 1 | 1.3 | 0.53 | 0.43 |
| o | 0.18 | | | | 0.17 |

Table A.2: This table shows the relative number of iterations which could be executed for each subtest in 100 seconds related to the test on Lilith which is referenced as 1 (cf. eq. a.1). These benchmark tests are done in Modula-2.

| Test No. | Macintosh | Macintosh[1] | Macintosh[2] | Macintosh[3] | Mac Plus[2] |
|---|---|---|---|---|---|
| | floppy-disk | floppy-disk | floppy-disk | floppy-disk | floppy-disk |
| a | | 0.83 | 0.93 | 0.93 | 0.92 |
| b | 0.35 | 0.8 | 0.75 | 0.75 | 0.75 |
| c | 0.41 | 0.73 | 0.69 | 0.68 | 0.9 |
| d | 0.29 | 0.56 | 0.6 | 0.6 | 0.6 |
| e | | 0.21 | 0.21 | 0.02 | 0.21 |
| f | | 0.15 | 0.15 | 0.01 | 0.15 |
| g | 0.29 | 0.62 | 0.7 | 0.7 | 0.69 |
| h | 0.29 | 0.61 | 0.85 | 0.85 | 0.84 |
| i | 0.22 | 0.59 | 0.61 | 0.61 | 0.38 |
| j | 0.22 | 0.6 | 0.73 | 0.73 | 0.73 |
| k | 0.28 | 0.88 | 0.79 | 0.79 | 0.78 |
| l | 0.34 | 0.84 | 0.81 | 0.81 | 0.81 |
| m | 0.89 | 0.84 | 0.84 | 0.84 | 0.84 |
| n | 0.43 | 0.52 | 0.69 | 0.69 | 0.69 |
| o | 0.17 | 0.26 | 0.02 | 0.02 | 0.02 |

Table A.3: This table shows the relative number of iterations which could be executed for each subtest in 100 seconds related to the test on Lilith which is referenced as 1 (cf. eq. a.1). These benchmark tests are done in Modula-2.
[1] 5-pass compiler Logitech; [2] 1-pass compiler MacMETH V1.0; [3] 1-pass compiler MacMETH V2.0;

| Test No. | Mac Plus[1] | Mac Plus[2] | Mac Plus[3] | Mac II[1] | Mac II[4] |
|---|---|---|---|---|---|
| | floppy-disk | floppy-disk | only (*-T*) | | |
| a | 0.93 | 0.95 | 0.94 | 4.45 | 4.47 |
| b | 0.75 | 0.75 | 0.76 | 3.75 | 3.75 |
| c | 0.69 | 0.69 | 0.69 | 3.53 | 3.53 |
| d | 0.6 | 0.61 | 0.6 | 2.83 | 2.83 |
| e | 0.02 | 0.21 | 0.22 | 0.11 | 0.77 |
| f | 0.01 | 0.1 | 0.02 | 0.7 | 0.46 |
| g | 0.7 | 0.7 | 0.7 | 3.23 | 3.25 |
| h | 0.85 | 0.85 | 0.85 | 3.89 | 3.89 |
| i | 0.61 | 0.61 | 0.61 | 2.77 | 2.78 |
| j | 0.73 | 0.73 | 0.73 | 3.33 | 3.34 |
| k | 0.79 | 0.79 | 0.79 | 2.39 | 2.4 |
| l | 0.81 | 0.81 | 0.81 | 3.06 | 3.05 |
| m | 0.84 | 0.84 | 0.84 | 3.56 | 3.56 |
| n | 0.69 | 0.69 | 0.69 | 2.96 | 2.97 |
| o | 0.02 | 0.02 | 0.02 | 0.07 | 0.07 |

Table A.4: This table shows the relative number of iterations which could be executed for each subtest in 100 seconds related to the test on Lilith which is referenced as 1 (cf. eq. a.1). These benchmark tests are done in Modula-2. The symbol (*$T-*) marks tests which has made no index test (arrays etc.)
[1] 1-pass compiler MacMETH V2.0 with SANE MathLib; [2] 1-pass compiler MacMETH V2.3;
[3] 1-pass compiler MacMETH V2.2; [4] 1-pass compiler MacMETH V2.2 with DMMathLib V0.4;

| Test No. | KayPro 286 (no 80287) | IBM PC AT (no 80287) | IBM PC AT (with 80287) | VAX 11-780 | VAX 8600 |
|---|---|---|---|---|---|
| a | 1.11 | 1.12 | 1.13 | 1.56 | 6.54 |
| b | 1.17 | 1.15 | 1.18 | 1.80 | 6.29 |
| c | 1.16 | 1.16 | 1.17 | 1.66 | 5.69 |
| d | 1.05 | 1.05 | 1.06 | 1.07 | 3.21 |
| e | 0.08 | 0.08 | 0.42 | 3.46 | 10.77 |
| f | 0.15 | 0.16 | 0.34 | 2.87 | 6.90 |
| g | 1.02 | 0.13 | 1.02 | 0.92 | 3.67 |
| h | 0.66 | 0.66 | 0.66 | 0.56 | 1.69 |
| i | 0.96 | 0.97 | 0.96 | 0.76 | 4.31 |
| j | 0.63 | 0.64 | 0.64 | 0.30 | 3.66 |
| k | 1.20 | 1.22 | 1.21 | 0.69 | 3.13 |
| l | 1.13 | 1.13 | 1.13 | 0.53 | 2.66 |
| m | 2.19 | 2.19 | 2.19 | 1.59 | 9.52 |
| n | 0.67 | 0.67 | 0.67 | 1.60 | 4.40 |
| o | 0.05 | 0.14 | 0.14 | 0.21 | 0.14 |

Table A.5: This table shows the relative number of iterations which could be executed for each subtest in 100 seconds related to the test on Lilith which is referenced as 1 (cf. eq. a.1). These benchmark tests are done in Modula-2.

| Test No. | VAX WS II | VAX 310 | Ceres | SUN 3/50 | SUN 3/160 |
|---|---|---|---|---|---|
| a | 2.18 | 2.05 | 1.7 | 4.08 | 5.06 |
| b | 2.10 | 2.06 | 1.37 | 3.59 | 4.15 |
| c | 1.66 | 1.78 | 1.02 | 2.53 | 3.18 |
| d | 1.07 | 1.07 | 0.89 | 1.94 | 2.5 |
| e | 3.08 | 3.85 | 2.05 | 0.75 | 0.94 |
| f | 3.45 | 4.6 | 2.26 | 0.61 | 0.77 |
| g | 0.92 | 0.92 | 1.41 | 2.87 | 3.43 |
| h | 1.12 | 1.12 | 1.08 | 2.22 | 2.63 |
| i | 1.02 | 1.27 | 0.78 | 2.75 | 3.3 |
| j | 1.22 | 1.22 | 0.74 | 2.13 | 2.46 |
| k | 0.69 | 1.04 | 1.22 | 3.33 | 4.11 |
| l | 1.06 | 0.53 | 1.15 | 3.43 | 4.27 |
| m | 1.59 | 1.59 | 1.86 | 2.43 | 2.83 |
| n | 1.60 | 2.4 | 1.43 | 3.86 | 4.74 |
| o | 0.21 | 0.24 | 0.75 | 0.54 | 0.7 |

Table A.6: This table shows the relative number of iterations which could be executed for each subtest in 100 seconds related to the test on Lilith which is referenced as 1 (cf. eq. a.1). These benchmark tests are done in Modula-2.

Second, we give by the tables A.7 and A.8 an overview of the benchmark test described in chap. 2 with a Pascal program.

| Test No. | CompuPro(B) | CompuPro(C) | CompuPro(D) | Macintosh |
|---|---|---|---|---|
| | only (*$T-*) | only (*$T-*) | (MP/M 8-16) | (floppy-disk) |
| a | 1.07 | 0.3 | 0.84 | 1.21 |
| b | 0.97 | 0.24 | 0.65 | 0.59 |
| c | 1.01 | 0.55 | 0.69 | 0.58 |
| d | 1.01 | 0.11 | 0.38 | 0.54 |
| e | 0.09 | | 0.07 | 0.03 |
| f | 0.08 | | 0.07 | 0.02 |
| g | 0.88 | 0.37 | 0.66 | 0.66 |
| h | 0.56 | 0.34 | 0.47 | 0.81 |
| i | 0.84 | 0.36 | 0.45 | 0.61 |
| j | 0.56 | 0.31 | 0.37 | 0.73 |
| k | 0.85 | 0.63 | 0.66 | 0.58 |
| l | 0.88 | 0.53 | 0.66 | 0.64 |
| m | 1.19 | 0.48 | 0.59 | 0.25 |
| n | 0.32 | 0.4 | 0.256 | 0.66 |
| o | 0.08 | 0.1 | 0.12 | 0.04 |

Table A.7: This table shows the relative number of iterations which could be executed for each subtest in 100 seconds related to the Modula-2 benchmark test on Lilith which is referenced as 1 (cf. eq. a.1). These benchmark tests are done in Pascal. The symbol (*$T-*) marks tests which has made no index test (arrays etc.)

| Test No. | VAX 11-780 | VAX 8600 | VAX WS II | VAX 310 |
|---|---|---|---|---|
| a | 8.41 | 15.89 | 5.92 | 8.41 |
| b | 0.6 | 13.47 | 4.49 | 0.6 |
| c | 48.47 | 47.39 | 146.58 | 48.47 |
| d | 1.87 | 4.546 | 1.07 | 0.83 |
| e | 30.77 | 73.85 | 28.46 | 30.77 |
| f | 146.72 | 235.1 | 159.15 | 146.74 |
| g | 8.26 | 17.89 | 6.41 | 8.26 |
| h | 10.11 | 22.47 | 7.87 | 10.11 |
| i | 2.79 | 6.6 | 2.03 | 2.79 |
| j | 3.35 | 7.32 | 2.44 | 3.35 |
| k | 18.75 | 35.42 | 13.89 | 18.75 |
| l | 27.13 | 63.83 | 21.28 | 27.13 |
| m | 1.27 | 11.11 | 1.59 | 1.27 |
| n | 6 | 19.2 | 4.8 | 6 |
| o | 0.13 | 0.26 | 0.1 | 0.25 |

Table A.8: This table shows the relative number of iterations which could be executed for each subtest in 100 seconds related to the Modula-2 benchmark tests on Lilith which is referenced as 1 (cf. eq. a.1). These benchmark test are done in Pascal.

Third, the tables A.9a, A.9b and, A.10 contain the data coming from the programs *Sieve*, *Ereal*, *Freal* and *Queen* and *MacQueen.*

| | Pascal | | | Modula-2 | | |
|---|---|---|---|---|---|---|
| | Sieve | Ereal | Freal | Sieve | Ereal | Freal |
| Lilith | | | | 4.24 | | |
| Ceres (10 MHz) | | | | 4.98 | 2.26 | 3.07 |
| CompurPro (Z80 8 MHz) | 13.3 | | | | | |
| CompurPro (A) | | | | 8.62 | | |
| CompurPro (A) only (*$T-*) | | | | 9.35 | | |
| CompurPro (B) | | | | 4.04 | | |
| CompurPro (C) | 2.75 | | | 3.6 | | |
| CompurPro (D) (8086 8 MHz) | | | | 6.9 | 72.2 | 107.3 |
| CompurPro (D) (8086/87 8 MHz) | 6.93 | 72 | 107 | 5.32 | 5.6 | 9.7 |
| MacIntosh 512 w/Floppy disk[1] | 7.4 | 21 | 44 | 7.4 | | |
| MacIntosh 512 w/Floppy disk[2] | | | | 7.2 | 22.4 | 47.6 |
| Macintosh 512 w/Floppy disk[3] | | | | 6.4 | 22 | 47 |
| Macintosh 512 w/Floppy disk[4] | | | | 6.4 | 257 | 324 |
| Macintosh Plus w/Floppy disk[3] | | | | 6.5 | 22 | 47 |
| Macintosh Plus w/Floppy disk[4] | | | | 6 | 221 | 360 |
| Macintosh Plus w/Floppy disk[5] | | | | 6.4 | 23 | 75 |
| Mac II[6] | | | | 1.29 | 4.28 | 9.32 |
| Mac II[7] | | | | 1.29 | 5.3 | 16.17 |
| KayPro II (Z80 3 Mhz) | 37.2 | | | | | |
| IBM PC (8088 5 MHz) | | | | 5.4 | 67 | 101 |
| IBM AT (without 80287) | | | | 5.4 | 67 | 101 |
| Cray - FORTRAN | 0.111 | | | | | |

Table A.9a: This table shows the benchmark results (measured time in seconds) for the tests with *Sieve*, *Ereal*, and *Freal*. The needed time is measured in seconds. The sign "†" marks tests which were to fast to be measured by stop-watch means. The sign "‡" means: virtual values - calculated time used. The symbol (*$T-*) marks tests which has made no index test (arrays etc.)

[1] 5-pass compiler; [2] 1-pass compiler; [3] 1-pass compiler MacMETH V1.0; [4] 1-pass compiler MacMETH V2.0;
[5] 1-pass compiler MacMETH V2.0 with SANE MathLib;
[6] 1-pass compiler MacMETH V2.2 with MathLib V0.4;
[7] 1-pass compiler MacMETH V2.2 with MathLib V0.5;

| | Pascal | | | Modula-2 | | |
|---|---|---|---|---|---|---|
| | Sieve | Ereal | Freal | Sieve | Ereal | Freal |
| VAX 11-750 | 4.6 | | | | | |
| VAX 11-780 | 2.23 | † | † | 4.64 | 1.63 | 2 |
| VAX 8600 | 1.2 | † | † | | 0.83 | 0.91 |
| VAX WS II | 1.96 | 0.15 | 0.03 | 2.82 | 1.69 | 2.3 |
| VAX 310 | 2.23 | † | † | 2.6 | 2 | 1.63 |
| SUN 3/50 | | | | 1.45 | 6.2 | 11.5 |
| SUN3/160 | | | | 1.26 | 4.79 | 9.23 |
| Apollo DN320 (68010 12MHz) | 2.3 | | | | | |
| Apollo DN330 (68020 12MHz) | 1.2 | 1200 | 1320 | | | |
| AT&T B2/400 | 1.2 | 1200 | 1320 | | | |
| Olivetti M24 | 6.9 | 72.2 | 107.3 | | | |
| CDC 6400 | ‡ 5.2 | ‡ 1.3 | ‡ 5.9 | | | |
| Cray - FORTRAN | 0.111 | | | | | |

Table A.9b: This table shows the benchmark results (measured time in seconds) for the tests with *Sieve*, *Ereal*, and *Freal*. The needed time is measured in seconds. The sign "†" marks tests which were to fast to be measured by stop-watch means. The sign "‡" means: virtual values - calculated time used. The symbol (*$T-*) marks tests which has made no index test (arrays etc.)
[1] 5-pass compiler; [2] 1-pass compiler; [3] 1-pass compiler MacMETH V1.0; [4] 1-pass compiler MacMETH V2.0;
[5] 1-pass compiler MacMETH V2.0 with SANE MathLib;
[6] 1-pass compiler MacMETH V2.2 with MathLib V0.4;
[7] 1-pass compiler MacMETH V2.2 with MathLib V0.5;

| | Queen | MacQueen |
|---|---|---|
| CompruPro | 185 | |
| MacIntosh | 65 | 1264 |
| Smaky | 100 | |
| MacIntosh+ | | 1140 |

Table A.10: This table shows the benchmark results for the tests with Queen and MacQueen. The needed time is measured in seconds.

## B. Benchmark programs

For the benchmark tests described in this publication, several versions of the benchmark program were written. For Modula-2 exist two versions, one for Macintosh computers under the simulation environment RAMSES, the other for batch runs on SUN workstations under RASS. There exist three C versions, one for SUN workstations (GNU compiler gcc and g++ V 2.4 (EPC, 1991)), and two for Macintosh computers - written for MPW C. A Pascal version for the IBM workstations was also realised.

We do not include the source code of the benchmark programs because these sources are easily available via different publications (Gilbreath, 1981; Wirth, 1981; Gilbreath & Gilbreath, 1983; Hinnant, 1984; Linton, 1986). Our benchmark programs as well as the

simulation environment RAMSES V2.2 and the Modula-2 compiler MacMETH V3.2.2 are/is down-loadable via ftp or WWW from the following addresses:

ftp://ftp.ito.umnw.ethz.ch/pub[5]

http://www.ito.umnw.ethz.ch

The benchmark programs are down-loadable by anonymous ftp from the directory /pub/benchmark and RAMSES, MacMETH from the directory /pub/pc/RAMSES and /pub/mac/RAMSES, respectively.

## ACKNOWLEDGEMENTS

This work has been supported by the Swiss Federal Institute of Technology (ETH) Zurich and by the Swiss National Science Foundation, grants no. 5001-35172 and 31-31142.91. We thank espacially Dr. H. Lischke for her valueable comments and support.

[5] The ip number of the SUNserver "baikal" is 129.132.80.130.

# BERICHTE DER FACHGRUPPE SYSTEMÖKOLOGIE
# SYSTEMS ECOLOGY REPORTS
# ETH ZÜRICH

---

Nr./No.

1 FISCHLIN, A., BLANKE, T., GYALISTRAS, D., BALTENSWEILER, M., NEMECEK, T., ROTH, O. & ULRICH, M. (1991, erw. und korr. Aufl. 1993): Unterrichtsprogramm "Weltmodell2"

2 FISCHLIN, A. & ULRICH, M. (1990): Unterrichtsprogramm "Stabilität"

3 FISCHLIN, A. & ULRICH, M. (1990): Unterrichtsprogramm "Drosophila"

4 ROTH, O. (1990): Maisreife - das Konzept der physiologischen Zeit

5 FISCHLIN, A., ROTH, O., BLANKE, T., BUGMANN, H., GYALISTRAS, D. & THOMMEN, F. (1990): Fallstudie interdisziplinäre Modellierung eines terrestrischen Ökosystems unter Einfluss des Treibhauseffektes

6 FISCHLIN, A. (1990): On Daisyworlds: The Reconstruction of a Model on the Gaia Hypothesis

7 * GYALISTRAS, D. (1990): Implementing a One-Dimensional Energy Balance Climatic Model on a Microcomputer *(out of print)*

8 * FISCHLIN, A., & ROTH, O., GYALISTRAS, D., ULRICH, M. UND NEMECEK, T. (1990): ModelWorks - An Interactive Simulation Environment for Personal Computers and Workstations *(out of print→ for new edition see title 14)*

9 FISCHLIN, A. (1990): Interactive Modeling and Simulation of Environmental Systems on Workstations

10 ROTH, O., DERRON, J., FISCHLIN, A., NEMECEK, T. & ULRICH, M. (1992): Implementation and Parameter Adaptation of a Potato Crop Simulation Model Combined with a Soil Water Subsystem

11 * NEMECEK, T., FISCHLIN, A., ROTH, O. & DERRON, J. (1993): Quantifying Behaviour Sequences of Winged Aphids on Potato Plants for Virus Epidemic Models

12 FISCHLIN, A. (1991): Modellierung und Computersimulationen in den Umweltnaturwissenschaften

13 FISCHLIN, A. & BUGMANN, H. (1992): Think Globally, Act Locally! A Small Country Case Study in Reducing Net $CO_2$ Emissions by Carbon Fixation Policies

14 FISCHLIN, A., GYALISTRAS, D., ROTH, O., ULRICH, M., THÖNY, J., NEMECEK, T., BUGMANN, H. & THOMMEN, F. (1994): ModelWorks 2.2 – An Interactive Simulation Environment for Personal Computers and Workstations

15 FISCHLIN, A., BUGMANN, H. & GYALISTRAS, D. (1992): Sensitivity of a Forest Ecosystem Model to Climate Parametrization Schemes

16 FISCHLIN, A. & BUGMANN, H. (1993): Comparing the Behaviour of Mountainous Forest Succession Models in a Changing Climate

17 GYALISTRAS, D., STORCH, H. v., FISCHLIN, A., BENISTON, M. (1994): Linking GCM-Simulated Climatic Changes to Ecosystem Models: Case Studies of Statistical Downscaling in the Alps

18 NEMECEK, T., FISCHLIN, A., DERRON, J. & ROTH, O. (1993): Distance and Direction of Trivial Flights of Aphids in a Potato Field

19 PERRUCHOUD, D. & FISCHLIN, A. (1994): The Response of the Carbon Cycle in Undisturbed Forest Ecosystems to Climate Change: A Review of Plant–Soil Models

20 THÖNY, J. (1994): Practical considerations on portable Modula 2 code

21 THÖNY, J., FISCHLIN, A. & GYALISTRAS, D. (1994): Introducing RASS - The RAMSES Simulation Server

22 GYALISTRAS, D. & FISCHLIN, A. (1996): Derivation of climate change scenarios for mountainous ecosystems: A GCM-based method and the case study of Valais, Switzerland

23 LÖFFLER, T.J. (1996): How To Write Fast Programs

24 LÖFFLER, T.J., FISCHLIN, A., LISCHKE, H. & ULRICH, M. (1996): Benchmark Experiments on Workstations

---

* Out of print

25 FISCHLIN, A., LISCHKE, H. & BUGMANN, H. (1995): The Fate of Forests In a Changing Climate: Model Validation and Simulation Results From the Alps

26 LISCHKE, H., LÖFFLER, T.J., FISCHLIN, A. (1996): Calculating temperature dependence over long time periods: Derivation of methods

27 LISCHKE, H., LÖFFLER, T.J., FISCHLIN, A. (1996): Calculating temperature dependence over long time periods: A comparison of methods

28 LISCHKE, H., LÖFFLER, T.J., FISCHLIN, A. (1996): Aggregation of Individual Trees and Patches in Forest Succession Models: Capturing Variability with Height Structured Random Dispersions

29 FISCHLIN, A., BUCHTER, B., MATILE, L., AMMON, K., HEPPERLE, E., LEIFELD, J. & FUHRER, J. (2003): Bestandesaufnahme zum Thema Senken in der Schweiz. Verfasst im Auftrag des BUWAL

30 KELLER, D., 2003. *Introduction to the Dialog Machine, 2nd ed*. Price,B (editor of 2nd ed)