SIMULATION ENVIRONMENTS AND SYMBOL AND NUMBER PROCESSING ON MULTI AND ARRAY PROCESSORS

PROCEEDINGS
OF THE
EUROPEAN SIMULATION MULTICONFERENCE
JUNE 1-3, 1988
PLAZA CONCORDE HOTEL
NICE, FRANCE

EDITED BY

R.C. HUNTSINGER
W.J. KARPLUS
E.J. KERCKHOFFS
G.C. VANSTEENKISTE

ORGANIZED BY SCS INTERNATIONAL
IN COOPERATION WITH ASIM, DBSS, ISCS, SIMS, UKSC, JSST.
AND CSSC

Printed in Belgium

ISBN 0-911801-39-1
Printed in Belgium

Benchmark on Supercomputers for an Industrial Environment

Emmanuel Vergison
Solvay Scientific Computer Centre
rue de Ransbeek 310
B-1120 Bruxelles

ABSTRACT

The aim of this study is to provide an industrial scientific and technical computer center with a reliable and exhaustive tool for making comparisons of performances between different computers including those of the new generation (parallel and vector machines).

INTRODUCTION

The methodology adopted for providing an industrial technical and scientific computer center with a reliable and exhaustive tool for making comparisons of performances between scalar and vector or parallel computers, is based on three main approaches : .

- general purpose considerations covering scalar and vector compilation, computer architecture, programming language, debugging, development tools and training & education
- practical test cases, themselves subdivided into three subgroups :
  - . basic operations and manipulations such as floating point operations, elementary mathematical functions, do-loops and branching
  - . linear algebra, this chapter being of major use for the practical applications and allowing precise estimates of the computational work involved.
    This chapter covers vector & matrix manipulations and direct algebraic linear system solvers
  - . industrial applications covering fluid mechanics, chemical engineering, heat transfer and management
- complementary studies in order to measure the manpower investment needed to make an intelligent use of the vector facility, either in vectorizing existing code, or by developing automatic tools capable of overcoming vector inhibitors.

The test cases under consideration were programmed in different ways in order to show what working principles the vectorizing processors are based on, and to highlight the compiler capabilities as well as the programming effort that is to be expected.

The language used is full FORTRAN 77/ANSI X3.9 - 1978 (only a few tests were carried out in FORTRAN 66).

The computer environments on which the tests were carried out, were composed of two subgroups of machines : mini-computers and super-computers, keeping in mind both departmental and centralized solutions.

A summary of the machines' overall performances is presented in the appendix.

THE "CLASSICAL" BENCHMARK

Since we aimed to check both scalar and vector capabilities of the machines in order to obtain an estimate of the absolute performances of computer codes, the benchmark is CPU oriented and all the tests were carried out in a single user environment.

Although we focused especially on computing performances, the benchmark was also designed to study compiled Fortran code efficiency and the ability of compilers to vectorize, that is, to render programs automatically suitable for super performance.

General Purpose Information

To begin with, we need a comprehensive view of what parallelism or vectorization means for each manufacturer.
So before starting any machine test, we have to have clean and comprehensive documentation on the following subjects :

Architecture

- . to which category does the computer belong : pipelined, array processor, M.I.M.D. [1] ?
- . how does vector processing work, i.e. how does it process array elements ?
- . how does the system handle multitasking, if any ?

Compiler

- . what about compiler capabilities, what does it vectorize and how ?
- . are there levels of vectorization; for instance does the compiler handle individual statements inside do-loops ? If any, which ones ?
- . what about compiler versatility ? Can we compile both in scalar and vectorized mode the programs written in scalar Fortran 77 ?
- . what is its compatibility with Fortran 66 ? Does the program have to conform to Fortran 77 to execute correctly when vectorized ?
- . what are the compiler vectorizing optimization levels, if any ?

Language

. to what extent are there vector language extensions ?
. if any, do they conform to the Fortran 8x draft ?
. which statements are vectorizable ?
. which data types can be handled by the vectorizing compiler ?

Performance Analyzers (Profilers)

. to what extent does the system provide the user with runtime information like the time spent in specified parts of the code or in subroutines ?

Reporting

. to what extent does the system produce a cross reference map or an output that shows the way vectorization was performed and to what extent does it give directives to improve the coding ?

Debugging

. can programs be debugged at any optimization or vectorization level ?

Mathematical Support

. does the manufacturer provide the user with any optimized mathematical software ?
. to what extent does it conform to the Basic Linear Algebra Subroutines concepts ?

Third Party Software

. does the machine allow for widely used scientific and engineering software ?

Practical Testcases

All these tests are programmed in standard Fortran 77 and the typical vector sizes used are 50, 64, 100, 300, 500 and 10,000.

Basic Arithmetic & Programming Operations : these include floating point operations and elementary mathematical functions evaluations on vectors, do-loops and branching.

Floating Point Operations : sixteen cases, taken from a benchmark by Karaki [2], are considered; they reflect common manipulations of scientific programming.

Elementary Mathematical Functions : frequently used mathematical functions have been tested. The only point to be stressed is the choice of different domains of definition for the sine, cosine and tangent function in order to take into account the fact that computing algorithms may depend on it.

Do-loops : several scenarios involving this typically well suited structure for vectorization are considered [3] : flow dependence, anti dependence, vectorization by loop distribution, vectorization by loop distribution and reordering, partial vectorization by loop distribution, as well as reducing scalar dependence by scalar manipulations.

Branching : this Achilles' heel of vectorization is, a priori, a hard to vectorize structure.
None the less, we tried to detect possible parallelism by using boolean structures and a masking technique [3].

Linear Algebra

Four groups of problems have been tackled :
- vector manipulations including the dot product, the DAXPY form and vector components summation
- matrix by vector products based either on the dot product or on the DAXPY form
- matrix by matrix multiplication based on the same techniques and
- direct linear algebraic solvers.

Vector Manipulations

The dot product $\Sigma X_i Y_i$, i = 1,..,n and the DAXPY form $Z_i = Y_i + aX_i$, i = 1,..,n are the basic structures of many algorithms in linear algebra. Their treatment depends strongly on the hardware architecture, so we found it interesting to check their respective performances as BLAS level 1 operations.

Matrix by Vector Product

The performances of both the dot product based multiplication :

```
      dø 10 i = 1,n
      dø 10 j = 1,n
   10 c(i) = c(i) + a (i,j) * b(j)
```

or the DAXPY form one :

```
      dø 10 i = 1,n
      dø 10 j = 1,n
   10 c(j) = c(j) + a(j,i) * b(i)
```

have been coded.

Matrix by Matrix Product

Here again, dot product and DAXPY forms were implemented as well as the "outer product" algorithm

```
      dø 10 k = 1,n
      dø 10 j = 1,n
      dø 10 i = 1,n
   10 c(i,j)  = c(i,j)+a(i,k) * b(k,j)
```

A point of supplementary interest concerning these matrix by matrix routines is the way the vectorizing compiler handles nested do-loops.

Algebraic Linear Systems

The purpose of this chapter is to solve linear algebraic systems using direct methods : the Gaussian and the Crout variant for non symmetric matrices and the Cholesky for symmetric ones.

Both the Crout and Gauss methods require the same computational effort, $(2/3)n^3$ operations, n being the matrix dimension) while Cholesky's needs only half of this. For example, a 100 X 100 non-symmetric system would require about 680,000 floating operations (flops).

The matrix under study is the Frank one that takes the form

$$(a(i,j))_{nxn} = (n+1 - \max(i,j))_{nxn}$$

It has an interesting feature being the inverse of a matrix which can be considered as a discretized form of a two point boundary value problem.

Two kinds of coding were implemented : a non sophisticated one (diagonally dominant pivoting) for both the Gauss and the Crout methods, the overheads being minimized and a more robust form of both the Gauss and Cholesky methods coded in a BLAS form by Dongarra and Eisenstat [4].

Application Programs

The applications described in this chapter are divided into four groups. For the first one, we can reasonably expect improvement simply by compiling; programs in the second need changes at the algorithmic level.
The third group is made of programs which potentially contain parallelism, whilst the last one contains those where parallelism can hardly be expected a priori.
To be closer to reality, some programs are written in single precision.

Applications With Compiler Detectable Parallelism

Two applications have been selected, one in fluid mechanics and one in heat transfer.

Fluid Mechanics : a steady two dimensional turbulent fluid flow is calculated, using an iterative "time marching" technique based on the finite difference Marker & Cell (MAC) method.
This program, implemented in single precision, was dealt with in further complementary studies [6].

Potential Equation : a potential equation solver, based on a multigrid algorithm and written in double precision, was added to the benchmark because the multigrid technique offers three interesting features :

- it is one of the most powerful PDE solvers
- it is naturally vectorizable
- it is easy to make operation count estimations.

Application Programs Needing Algorithmic Changes For Vectorization

A one dimensional nonlinear heat equation, using an unconditionnally stable algorithm requiring a tridiagonal linear system solver, has been coded in two ways (both in double precision).
In the first version, the tridiagonal system is solved by the classical Gaussian elimination algorithm using forward elimination and backward substitution.
A total of 9n scalar arithmetic operations is required, but we have to keep in mind that there is one division and some overhead due to integer manipulations as well as routing.

A variant, based on a cyclic reduction algorithm, better suited for parallel computing, has been implemented.
It requires 17 ($n + n_{\frac{1}{2}} * \log_2 n$) operations where $n_{\frac{1}{2}}$ is the machine half performance length [1], typically 10 for a CRAY 1, 2 for a FPS/164 and 0 for a scalar machine.

Potentially Vectorizable Applications

Many applications in chemical engineering use numerical algorithms which can be viewed as fixed point iteration techniques (Picard's method); these algorithms are "vector minded" and should give good results on parallel machines.

First a non linear regression analysis program designed to check the statistical consistency of productions and consumptions by finding the maximum likelihood estimates of the model parameters was coded in Fortran 77, in double precision. The constrained nature of the problem introduces non-linearities that are handled using Picard's method.

Secondly, a set of kinetic equations, solved using the classical Runge-Kutta method was implemented. Due to problem complexity, the formulation of the second members of the differential equations makes an intensive use of function calls which can actually prevent any benefit of vectorization.
This test case bridges the gap with the next paragraph.

Applications Where Parallelism Is Not To Be Expected

As far as we know from our experience, the programs which fall into this category are the operations research ones, those using complex indirect addressing in their numerical algorithms and those making an intensive use of subroutines and function calls. Two examples were used.

Operations Research

A branch and bound algorithm with binary variables (0-1) and pure integer data, for solving a large linear system with linear constraints was implemented in single precision.

Linear Mean Squares

The minimal norm problem is to be solved : minimize

$$||Ax - b||_2,$$

where A belongs to $R^{m \times n}$, $m \geq n$
and is sparse with random structure.
Sparsity, here means that the matrix density d(a), defined as

$$d(a) = \frac{\text{number of nonzeroes of A}}{m \times n},$$

should be less than or equal to 0.05.
QR factorization exploring sparsity at symbolic level, was the algorithm adopted for solving this problem. It was implemented in Fortran 77, in double precision, for the floating point aspects, otherwise in integers.

THE COMPUTER ENVIRONMENTS

We first recall that the tests were carried out in a single user environment and that the set of tested machines has been divided into two subgroups : the mini-supercomputers or "crayettes", and the super-computers as shown in the table below (the scalar DEC 8700 was chosen as a reference)

	FPS M64/60	Attached Processor & Mono-user Wide Instruction Word Machine SJE operating system
CRAYETTES	CONVEX C1/XP	Stand alone & Multi-user Vector Machine UNIX derived operating system
	CONVEX C210	Stand-alone & Multi-user Vector Machine UNIX derived operating system
	ALLIANT FX/4	Stand-alone & Multi-user Parallel Machine with 4 computer elements UNIX derived operating system
	ETA10 MOD P	Stand-alone & Multi-user Vector Machine EOS operating system
SUPERCOMPUTERS	CYBER 990	Stand alone & Multi-user Vector Machine NOS/VE operating system
	CYBER 205	Stand alone & Multi-user Vector Machine NOS/VE operating system
	IBM 3090/150	Stand-alone & Multi-user Scalar Machine with Vector Facility MVS or VM/CMS operating system
	IBM 3090/180	Stand alone & Multi-user Scalar Machine with Vector Facility MVS or VM/CMS operating system
	CRAY/XMP 145E (*)	Back End Processor Vector Machine COS operating system

The computer environments of the benchmark

(*) by courtesy of Professor R. DEVILLERS from Brussels Free University

The tables in the Appendix summarize the results obtained on all the machines except the IBM 3090/180 and the CYBER 205, the reason being that :

- the overall performances of the IBM were estimated at 1.2 times those obtained from the IBM 3090/150. We also note that we were interested in the IBM 3090/180 machine because it is the basic module of the IBM 3090 field-upgradable family (Models 200 to 600)
- the results we obtained from the CYBER 205 are estimated at 2.5 times those we got from the CYBER 990.

We will now comment on these results.

Results From The Basic Operations

The results, presented in Table 1 of the Appendix, are average values with typical vector lengths of n = 50 and n = 500, which were considered as the most representative of our work. Detailed results are shown in [4].

Results From Linear Algebra

In Table 2 of the Appendix, the dot product and DAXPY form results are presented for typical vector sizes of 64 and 512.

Tables 3 and 4 show how the dot product and the DAXPY form reflect when implemented in both the matrix by vector and the matrix by matrix multiplication. Of great interest is the do-loop unrolling technique (Table 4) which reduces memory references by using the vector registers more efficiently [5].

The benchmark results of linear algebraic solvers in Table 5 of the appendix, were completed by some machine specific equivalents. The use of these specially tuned routines generally improves performances substantially.

Results From The Applications

One should be very careful when interpreting the results summarized in Table 6 of the appendix. We must remember that :

- the application part of the benchmark was set up without any special tuning, even if some applications were, a priori, vectorizable and if some additional work was carried out afterwards either by the manufacturer or by ourselves
- though most applications were programmed using double precision, three important exceptions were included in the benchmark : a fluid dynamic code written in single precision, an operations research one dealing with integer programming and a linear mean square solver mixing double precision and integer programming
- a tuning effort was carried out on two programs : the non-stationary heat solver and the fluid dynamic code [6].

One general observation is that, without any changes in the codes, only the supercomputers actually improve the performances of the whole set of programs. However, all of them do it significantly after some programming or algorithmic work.

Particularly impressive is the speedup already obtained by the replacement of old Fortran 66 structures by Fortran 77 ones, and by inserting frequently called routines inline [6].

Overall Performances

Although overall performances naturally raise objections and criticisms, we found it appealing :

- to summarize in one table the average ratios we got from the computations, the DEC 8700 being taken as a unit
- to give, to some extent, a realistic although somewhat conservative idea of the machine's efficiency.

Table 7 in the appendix summarizes the characteristics of the machines tested so far. By "presently expected ratios" we mean that weights were introduced to take into account the fact that industrial libraries contain programs that are, a priori, not vectorizable or that are not written in a modern structured language.

COMPLEMENTARY STUDIES

Two studies, complementary to the benchmark described earlier, were conducted in order to quantify the manpower needed to vectorize a fluid dynamics code and to develop an inline routine inserting pre-compiler in order to get round subroutine calls which, used in do-loops, seriously inhibit vectorization.

Adaptation of a Fluid Dynamic Code

The objective of this work, conducted in collaboration with Brussels Free University [6], was to vectorize the fluid dynamic code we mentioned in "Applications with Compiler Detectable Parallelism".

The procedure, that was followed, can be divided into four steps :

- performance analysis
- scalar re-programming
- algorithmic changes
- vector programming

Performance Analysis

A profiler (performance analyser) was used to detect those parts of the code that are the most time consuming. This information coupled with relevant directives provided by efficient compilers showed where to put one's effort.
In our example, one routine consumed 60 % of the total CPU time and another one 20 %.

Scalar re-programming

We observed, when re-coding those parts of the code detected by the profiler, that inline insertion of routines, for example, already improved scalar performances. Also replacing branching structures by masking led to the same conclusion. These changes are to be considered as a pre-conditionning of the code.

Algorithmic Changes

These may either affect the whole algorithmic strategy of the problem or only one part of it. In the fluid dynamic code under consideration, the time marching procedure chosen to reach the steady state reflects an explicit iteration technique which does not need special adaptation.

It was not the case for the innermost pressure solver where the line by line mesh sweeping had to be replaced by red-black ordering.

Vectorization

The first transformation that was made, was to re-write matrices as long vectors applying gathering techniques in order to avoid space wasting and to allow fast access to equally spaced vector components.

The second one consisted in inserting parentheses to force execution of the arithmetic operations in the most efficient order (linked triads for example).

Summary of Results of Code Vectorization

The following speedup ratios are calculated, the DEC 8700 being taken as a unit.

	BEFORE TUNING	AFTER TUNING
CONVEX C1 XP	0.7	6.1
CONVEX C210	-	16.8
FPS 64/60	3.1	7.9
ETA 10 MOD P	-	35.6
ALLIANT FX/4	0.7	6.5
IBM 3090/150	4.8	9.4
CYBER 990	3.2	5.5
CRAY XMP/14SE	17.3	19.0

In Line Inserting Pre-Processor

The procedure followed, in a study made in collaboration with Brussels Free University, is made up of two steps preceding the compile-link-go : code standardization and source code pre-processing.

Code Standardization

In order not to make the pre-processing too complex, the source code is standardized using TOOLPACK, a set of programming development tools developed by the Numerical Algorithms Group [7].

Source Code Pre-Processing

This tool inserts routines in line, that means, replaces subroutine calls by their full coding. The pre-processor must be told which routines to handle. This decision follows from the profiler's analysis. The compile-link-go acts on the processed source code, while maintenance is done only on the original one.

CONCLUSIONS

In spite of the inherent limitations imposed on benchmarking work, amongst which :

- the necessity to make choices;
- the limited time available for benchmarking.
- the fact that the units used to quantify performances (seconds and MFLOPS) can be critised, to some extent;
- the continuous improvements in hardware and software;
- the fact that machine workload and I/O's have not been taken into account,

we can conclude that we now dispose of a tool capable of :

- giving a very good idea of the vector and vectorizing capabilities and maturity of machines, both from a hardware and a software point of view, so reducing industrial choices to a few clear strategies
- measuring precisely the effort needed to adapt old programs or to write new ones
- checking how manufacturers cope with both third party software and the present standardization tendencies in scientific languages (Fortran 8X) and operating systems (UNIX)
- adapting easily to new approaches in computing science.

APPENDIX : SUMMARY OF THE BENCHMARK'S RESULTS.

	DEC 8700	CONVEX C1 XP	CONVEX C210	FPS 64/60	ETA 10 MOD P	ALLIANT FX/4	IBM 3090/150	CYBER 990	CRAY XMP 14SE
FLOATING OPERATIONS (MFLOPS)	1.2	2.8	10.0	6.7	54.2	7.2	19.1	31.7	62.3
DO LOOPS (MFLOPS)	1.0	3.1	9.5	5.4	38.5	2.9	12.2	19.1	30.4
MATH. FUNCTIONS (MICRO SEC. PER OPERATION)	35.4	4.9	1.5	3.2	0.6	4.7	1.5	2.9	0.3
BRANCHING (MICRO SEC.)	580.	770.	286.	302.	638.	601.	127.	248.	393.

TABLE 1 : BASIC PROGRAMMING OPERATIONS

		DEC 8700	CONVEX C1 XP	CONVEX C210	FPS 64/60	ETA 10 MOD P	ALLIANT FX/4	IBM 3090/150	CYBER 990	CRAY XMP 14SE
DOT PRODUCT	64	1.3	9.1	12.7	11.5 14.6	25.1	4.3	15.6	25.6	22.9
	512	1.2	9.1	19.3	12.5 18.2	64.6	3.8	24.4	48.8	93.8
DAXPY FORM	64	1.2	9.0	12.2	8.9 10.1	106.1	2.8	8.4 13.8	32.0	75.3
	512	1.2	9.0	15.3	9.4 12.3	157.9	3.1	12.6 25.	41.0	82.0

TABLE 2 : VECTOR OPERATIONS

REM. : when there are two figures in one cell, the second refers to tuned versions of the code.

		DEC 8700	CONVEX C1 XP	CONVEX C210	FPS 64/60	ETA 10 MOD P	ALLIANT FX/4	IBM 3090/150	CYBER 990	CRAY XMP 14SE
DOT PRODUCT BASED	50	0.9	2.0	6.0	9.2	26.7	10.0	10.9	8.3	16.8
	500	0.5	2.9	8.4	11.9	116.4	1.4	3.6	5.3	52.6
DAXPY BASED	50	1.1	3.6	10.7	7.7	22.5	2.6	19.5	20.8	45.5
	500	1.0	5.5	15.2	9.4	104.9	2.7	22.7	38.8	79.6
UNROLLED FORM	50	1.1	5.2	-	13.4	21.2	5.0	7.5	8.8	60.7
	500	1.0	9.3	-	18.6	59.3	6.0	7.6	8.9	121.4
USING SPECIFIC LIBRARIES	50	-	12.6	38.1	29.7	-	-	48.4	-	-
	500	-	16.5	46.6	32.5	-	-	46.5	-	-

TABLE 3 : MATRIX BY VECTOR PRODUCT

		DEC 8700	CONVEX C1 XP	CONVEX C210	FPS 64/60	ETA 10 MOD P	ALLIANT FX/4	IBM 3090/150	CYBER 990	CRAY XMP 14SE
BENCHMARK BEST FORMULATION	50	1.1	11.7	34.1	10.1	19.5	9.4	22.6	22.9	48.3
	300	1.0	15.5	42.6	11.8	71.5	7.7	21.0	36.8	75.4
USING SPECIFIC LIBRARIES	50	-	-	40.4	32.1	-	-	52.5	-	-
	300	-	-	44.7	33.4	-	-	65.4	-	-

TABLE 4 : MATRIX BY MATRIX PRODUCT

		DEC 8700	CONVEX C1 XP	CONVEX C210	FPS 64/60	ETA 10 MOD P	ALLIANT FX/4	IBM 3090/150	CYBER 990	CRAY XMP 14SE
CROUT METHOD	50	0.9	2.0	9.4	7.2	15.1	7.9	6.2	5.0	8.5
	300	0.8	3.7	26.9	11.0	52.1	7.6	12.0	9.8	26.5
GAUSS BLAS	50	0.9	2.0	-	5.4	5.0	1.0	8.4	4.9	16.7
	300	0.7	3.6	-	7.1	10.2	3.5	16.0	10.3	42.1
CHOLESKY BLAS	50	0.9	1.9	8.7	5.6	12.7	1.1	7.6	7.3	18.0
	300	1.0	5.0	17.3	6.1	61.8	4.5	14.5	23.2	58.3
USING SPECIFIC LIBRARIES	50	-	-	9.0	18.3	-	2.4	16.8	-	-
	300	-	-	34.3	31.8	-	16.7	38.0	-	-

TABLE 5 : LINEAR ALGEBRAIC SOLVERS

	DEC 8700	CONVEX C1 XP	CONVEX C210	FPS 64/60	ETA 10 MOD P	ALLIANT FX/4	IBM 3090/150	CYBER 990	CRAY XMP 14SE
HEAT EQUATION	77.7	148.5	-	54.2	29.5	-	33.9	43.	18.
		45.	14.8	32.0	69.6	30.3	9.9	-	-
FLUID DYNAMICS	2475.	3656	-	790.	-	3385.	514.	776.	143.
		408.	147.2	315.	69.6	381.	262.	446.	130.
POTENT. EQUAT.	32.6	24.8	6.9	9.7	20.7	23.0	7.1	18.0	4.7
OPERAT. RES.	118.0	209.4	78.7	570.0	-	375.9	63.1	55.0	30.
LIN. MEAN SQ.	318.7	347.5	74.9	79.1	125.5	476.5	68.6	52.0	18.7
REGR. ANALYSIS	18.0	9.0	5.6	2.5	2.3	-	-	3.6	3.0
CHEM. KINETICS	14.0	-	-	5.1	7.4	-	-	3.5	2.3

TABLE 6 : APPLICATION PROGRAMS

REM : when there are two figures in one cell, the second refers to tuned versions of the code.

	DEC 8700	CONVEX C1 XP	CONVEX C210	FPS 64/60	ETA 10 MOD P	ALLIANT FX/4	IBM 3090/150	CYBER 990	CRAY XMP 14SE
BASIC OPERAT.	1	3.0	11.1	6.0	36.9	4.1	13.9	15.0	56.4
LIN. ALGEBRA	1	5.4	27.6	9.7	59.4	5.1	16.0	20.6	52.6
APPLICATIONS (*)	1	1.1	-	3.6	-	0.9	4.6	3.9	10.0
	(**)	2.2	6.9	4.6	8.7	2.8	6.6		
AVERAGES	1	3.2	-	6.4	-	3.4	11.5	13.2	39.7
		3.5	15.2	6.8	3.5	4.0	12.2		
WEIGHTED RATIOS	1	1.0	-	3.1	-	0.8	4.1	3.6	9.1
		1.9	6.3	4.0	7.6	1.8	5.8		

TABLE 7 : PERFORMANCE RATIOS, THE DEC8700 BEING TAKEN AS A UNIT

(*) without the "spaghetti" operations research program
(**) second figures refers to tuned versions of the codes.

Bibliography

1. R.W. Hockney and C.R. Jesshope, Parallel Computers, Adam Hilgher Ltd, 1986.

2. Y. Karaki : Performance of mathematical libraries on the HITAC S-810 supercomputer.
Supercomputer, May 1985, Pages 37-46.

3. E. Vergison : Benchmark for Scientific & Technical Computers in an industrial environment
SOLVAY Report, July 1987.

4. E. Vergison and T. Engels : Synthesis of the results from the Solvay Scientific & Technical Benchmark on supercomputers
SOLVAY Report, July 1987.

5. J.J. Dongarra and S.C. Eisenstat :
Squeezing the Most out of an Algorithm in CRAY FORTRAN, ACM Transactions on Mathematical Software, vol. 10, No. 3, September 1984, Pages 219-230.

6. P. Geleyn : Superordinateurs : test de performance et méthodologie de programmation.
Student's work conducted at Brussels Free University, December 1987.

7. R. Iles : Toolpack/1, version : 2.1
Quick reference guide
Nag Publication : NP1284 - Oxford, October 1986.

8. I.N.R.I.A. : Parallel Scientific Computation
Cours et Séminaires - Saint Cyprien (France), Octobre 1985.

9. J.J. Dongarra, J. Du Croz, S. Hammerling and R. Hanson :
A proposal for an extended Set of Fortran Basic Linear Algebra Subprograms, Argonne National Laboratory, December 1984.

10. A. Lichnevsky : Parallelism and its use in vector computers Course at ISPRA (Italy), June 1985.

11. J.J. Dongarra : Performance of various computers using standard linear equations software in a FORTRAN environment.
Argonne National Laboratory, February 1986.

12. H. Foerster and K. Witsch :
Multigrid Software for the Solution of Elliptic Problems on Rectangular Domains : MGOO
Gesellschaft fur Mathematik und Datenverarbeitung Mbh, Bonn, September 1982.

SIMULATION ENVIRONMENTS

GENERAL SIMULATION SOFTWARE ENVIRONMENTS I

Introduction to APROS Simulation Environment 5
Eero Silvenoinen, Ilkka Raiskinen and Kaj Juslin

SENDA : Languages and Tools for a Simulation Environment 10
A. Santos, P. Rodrigues and J. Gutierrez

OPTISIM : An Optimisation – Simulation Environment for Model Buildins 16
D.R. De Buyser and J.A. Spriet

MAISTRO : Modular Advanced Interactive Simulation System to Real Time Operation 22
Lars Peter Jensen

Simul-R – A Simulation Language with Special Features for Model-Switching and -Analysis (The Preprocessors BAPS, CAPS and Goma), ... 28
Ronald Ruzicka

GENERAL SIMULATION SOFTWARE ENVIRONMENTS II

The Simulation System Hybsys 35
Felix Breitenecker and Dietmar Solar

Systems Analysis, Model Construction, Simulation : Methodological Basis of the Simulation System SIMPLEX II 39
Bernd Schmidt

Systems Analysis, Model Construction, Simulation. The Structure of the Simulation System SIMPLEX-II 47
Bernd Schmidt

Higher Level Modeling in RESQME 52
Robert F. Gordon, Edward A. Macnair, Kurtiss J. Gordon and James F. Kurose

Graphic Input Procedures for Simulation Programs in Pascal Environments 58
Berta Buttarazzi

DEDICATED SIMULATION SOFTWARE ENVIRONMENTS

A User-Friendly Software Environment for Design and Simulation of DSP Algorithms and Processes on a SUN Workstation 65
Richard N. Zobel and Adrian M. Cummings

Sequential Function Chart Modelling with RESQ 71
Pierre Massotte

REMIS : Rise Evaluation and Management Information System 75
K. Harald Drager, Roy Gulliksen and William J. Stav

The Modelling and Simulation of the Nonlinear Loads and Harmonics in Electric Power Networks 81
Nejat R. Tuncay and Peter J. Brown

ENVIRONMENTS FOR COMPLEX SYSTEM SIMULATION IN REAL TIME CONTEXT

Organizers R. Huber and A. Guasch

Hardware Simulation of Spacecraft Motion with the European Proximity Operations Simulator (EPOS) 87
G. Heimbold and N. Stewart

Current Trends in Parallel Simulation Techniques 93
L.M. Patnaik and K.J. Mohan

Interactive Simulation in a Computer Based Training Environment 99
H. Kuiper and J.G.M. van der Arend

ASPECTS OF PARALLEL SIMULATION ENVIRONMENTS

Organizers M.A. de Bruyn / A. van der Horst

Neural Networks : A Feasible Approach for Process Planning Automation 107
Dia L. Ali, Adel L. Ali and Kamal S. Ali

Some Ideas about Parallelization of Knowledge Processing in Research and Development 112
L. Dekker and J.C. Zuidervaart

A C-Like Implementation of the « Coffee-Pot » Benchmark in DPP84 Parallel Processing Environment 117
M.A. de Bruijn and S.W. Brok

Simulation with Hybsys-Ptran at the EAI Simstar 123
Dietmar Solar, Wolfgang Kleinert, Martin Graff and Felix Breitenecker

METHODOLOGICAL CONCEPTS FOR SIMULATION ENVIRONMENTS

The Systematic Ranking Methodology Model for the Assessment of Environmental Impact 129
Errol E. Kalmaz

Algebraic Specification of a Continuous Simulation Environment 136
M. Grana, L.M. Alonso, F.J. Torrealdea and F. Ferreres

Bionics and Artificial Intelligence : On Some Problems of Mnemology 141
Simeon Jordanov Mrchev

A Methodology for Performance Evaluation of Enhanced Operations on a RISC Design Using Simulation 149
Nashat E.Al-Ghitany, J.M. Jagadeesh and P. Sadayappan

KNOWLEDGE-BASED (EXPERT) SYSTEMS AND THEIR ENVIRONMENTS

Organizer H.J. van den Herik

Modelling Uncertainty in ESATS : An Expert System for the Analysis of Thallium-201 Scintigrams 157
E. Backer, J.J. Gerbrands, J.H.C. Reiber, A.E.M. Reijs, W. Krijgsman and H.J. van den Herik

The Probability of Erroneous Firing of a Binary Rule in a Rule-Based Expert System 163
Henk Koppelaar

An Intelligent Environment for a Simulation Library 169
Patrick Laug and Alain Perronnet

ALLEX, an Expert System Shell to support Simulation 173
Ivan Futo

Simulation in Administrative Law 179
A.H. J. Schmidt

KNOWLEDGE-BASED SIMULATION ENVIRONMENTS

A Knowledge Based Simulation Environment for Fast Prototyping 187
Giorgio Guariso, Martin Hitz and Hannes Werthner

Object-Based Simulation : Foundations and Implementation in Modula 2 193
Thomas Witte and Ronald Grzybowski

An Interactive Simulation Environment based on Systems Theory Concepts and Object Oriented Programming 199
Herbert Praehofer and Albert Spalt

A Simulation Environment for the Development of Information Systems 204
Martijn A.H. Cohen and Henk G. Sol

KNOWLEDGE-BASED SIMULATION ENVIRONMENTS IN CONTROL

Organizer A. Kornecki

Simulation Based Expert System for ATC Training 211
A. Ransom, A. Kornecki, A. Gonzales, P. Bauert and R. Phinney

Prolog Based Air Traffic Control Expert / Simulator 217
Andrzej Kornecki

Knowledge Representation in Computer-Aided Control Systems Design Packages 223
Gerard Hoffmann and Magnus Rimvall

INTERACTIVE AND KNOWLEDGE-BASED MODELLING ENVIRONMENTS

Organizer A. Lehmann

The Integrated Modelling Package (IMP) : An Object-Oriented Module for Manufacturing Simulation 231
Bruce McLaren and Peter M. Neuss and Onno De Groote

Computer Aided Construction of High Level Conceptual Q.N Models 237
Axel Lehmann, Gerald Recktenwald and Helena Sczerbicka

Support Environments for Discrete Event Simulation Experimentation 242
R.P. Taylor and R.D. Hurrion

Communication Means between PDP-11 and Prolog in the Poplog Environment 249
I. Bruha

KNOWLEDGE-BASED SIMULATION TOOLS AND TECHNIQUES

Organizer L. Birta

An Expert Modelling and Simulation System on Sun Workstation 255
Tuncer I. Oren and J.C.M. Tam

On Constraint Oriented Environments for Continuous System Simulation 261
Richard A. Huntsinger

An Operational Semantics for a Discrete Event Simulation Suite in Concurrent Prolog 265
V.L.B. Freitas, A.J.T. Santos, A.J.M.G. Rodrigues and J.C.F.M. Neves

SYMBOL AND NUMBER PROCESSING ON MULTI AND ARRAY PROCESSORS

PLENARY SESSION

Innovative Computational Architecture for Parallel Applications 275
David J. Evans

PROGRAMMING SUPPORT FOR PARALLELISM

Organizer J. Van Campenhout

Dynamic Processor Clustering in a Parallel Computer with Fully Interconnected Processors 283
L. Dekker

Development of an Attached Distributed Processing System 289
Adrian D. Lincoln

The Extension of the Coprocessor-Concept to the Instruction Path 294
E.H. Debaere

HIDE : Decomposition System for Parallel Processors 300
Henk J. Sips and Robert de Vries

An Efficient Multiprocessor System 304
Rudi Cuyvers, Rudy Lauwereins, and J.A. Peperstraete

PERFORMANCE OF PARALLEL COMPUTING

Performance Measurements and Analysis of a Shared-Memory Multi Processor 313
E.H. D'Hollander and J.M. Opsommer

The Conjugate Gradient Method on the Delft Parallel Processor 318
J.H.M. Andriessen

Solving the Equations of Laplace on a Network of Transputers, Some General Conclusions 324
Eric Verhulst