A Statistical Method for Real-Time Software Estimation

Patrick Facchini

Abstract

This report is the result of an equivalent to a Masters thesis project done at the Linköping University, Sweden, during the period February to July 1996.

The execution time of a program or a block of a program is important in real-time design. Different methods to estimate the performance of a program, taking into account the features of a processor (clock frequency, memory access time), have been formulated. However, those methods cannot be used early in the design. This report introduces a new method of estimation based on statistical analysis and measurements which model the hardware. It uses linear regression to estimate the execution time of a program on different systems. A model, based on the execution time of narrow programs (describing basic operations), determines the characteristics of the different processors. Then, the estimation of the execution time of a program is the product of the values of this model and the regression parameters, which are calculated with linear regression.
The method has been tested with Ada programs on Sparcstations and PCs. We point out the importance of the accuracy of measurements, which may have a non-negligible influence on the results. The error of the estimates is very small for a set of only Sparcstations, but is larger when PCs are added to that set. We discuss plausible causes of these results.

The first part of this report is a description of the subject and of previous research. Then the report deals with the method of evaluation, followed by the description of the experiments. The last part is the analysis and conclusion about this method.

IDA Technical Report 1996
LiTH-IDA-R-96-22
ISSN-0281-4250

Department of Computer
and Information Science,
Linköping University,
S-581 83 Linköping, Sweden

Acknowledgements

I would like to thank my supervisor, Anders Törne, for his support and his encoura ment. Thanks also to Jakob Axelsson, Ph. D. student, for helping me amicably du the whole project. Furthermore, I wish to thank Bernt Nilsson, Leif Finmo, I Viklund, the system managers for their indispensable help. I am grateful to A Eskilsson for her administrative services. Last, but not least, I owe a lot to all graduates students at RTSLAB, for their frienship and their invaluable contributi to the pleasant atmosphere.

Table of contents

1. Motivation: Performance evaluation

Real-time system design needs to make a choice of hardware architecture. Indeed the embedded real-time systems might have a part of their functionality implemented in software, and the rest in hardware, using application-specific circuits (Figure 1).

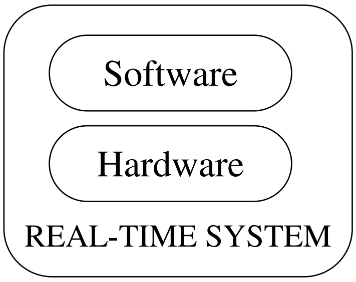


Figure 1: Parts of a real-time system

The execution time of an algorithm varies for different combinations of compilers and hardware architectures. The problem is to know the performance of the software early during the design in order to decide which processor, and other components related to it, will fit the best. The performance of a real-time system does not depend only on the program and the clock frequency of a processor. In that case it would be easy to find out which processor would be the optimal one for a given application.

The compilers and the hardware way of operating are so complicated and their performance is so unexpectable that it is not possible to calculate the precise execution time of a program, it is only possible to estimate it. In real-time it is needed to know the reaction time of a program. This time is usually constrained by an interval. The actual execution time must be in the interval $t_e \pm \Delta t_e$ where t_e is the estimated time and Δt_e is the estimated error.

Different methods and tools of timing analysis already exist (see also next chapter), but the results are sufficiently accurate only if the system is already implemented. Thus precise estimations can only be made very late in the design process. At this stage the software could be unadapted to the hardware and the program should be designed and implemented again. This is not possible in practice because of the time it needs.

New methods must therefore be found for the early design. We will study the timing analysis with statistic tools such as linear regression. The approach will be oriented towards Ada programs and single processors.

2. Previous research

Several methods have been suggested that can be used for software performance estimation. They have different levels of abstraction and of different portability. Some methods transform program into an intermediate format, on which the analysis is done.

- Bonnier *et al.* ([3]) show the possibility of a new method which uses regression analysis. The report is an inventory of methods and tools for timing and performance analysis to be used during software design.

- Saavedra ([14]) presents a method which uses narrow spectrum benchmarking. Each benchmark corresponds to a basic operation of a program. The execution time of all of them is measured on a machine. To know the execution time of a program on this machine, it must be decomposed into a sum of these benchmarks and multiplied by their execution time. This report contains a lot of informations about time measurement, optimizing compilers and previous research in timing analysis. The drawback of this method is that it needs a detailed decomposition of a program into the basic operations, that cannot be done at early design. Moreover, the method does not give correct estimates for all kind of programs.

- Ein-Dor and Feldmesser ([5]) have built a model for a computer, by using statistical analysis. The model is based on CPU characteristics - main memory size, cache memory size, number of channels, and machine cycle time. This analysis indicates that it is possible to model the relative performance of computer systems with reasonable accuracy on the basis of system components. Relative performance is a dimensionless number that can be used for the comparison of computer of any type or size. The main advantage it poses is that it is a single, easily understood, and eminently applicable characteristic. However, data is not available for all computers and the measure cannot be applied to new machines not yet on the market, but for which the technical data may be known.
  The model is linear and uses the CPU characteristics, after transformation and standardization, as independent variables. This method gives good results for the relative performance, but it can only be used for a preliminary selection of machines, since it is not closely related to the execution time of a particular program.

- Park and Shaw ([16]) discuss a source-level analysis which is based on timing schemas for the different constructs of the source language. It can be ported to a different processor by calculating new timing schemas, but since it works directly on the source code, it has some problems taking into account compiler optimization made at lower levels.

- Gong *et al.* ([6]) compile a source program to an intermediate three-address code. The analyser is then provided with a processor description file, which gives the expected execution time of each intermediate code instruction on the particular processor. A total of over 300 values are needed. The analyser can be ported to a different processor by writing a new processor description file, but this requires detailed knowledge about the processor architecture and instruction set.

- Harmon *et al*. ([8]) do analysis on machine code level. The estimator is portable, but only with some effort, since it requires information about the tiniest details of the processor instruction set. The technique can handle advanced processor features, such as cache memories and pipelines, with high accuracy.

- Axelsson ([2]), in a similar approach to Gong's, compile the source code to an intermediate format, on which the analysis is done, but uses a model constituted by the characteristics of the processor (as Ein-Dor) to make a prediction of the execution time. The estimates given by this method seem to have good quality, but it does not take into account all features of the different processors in the market and needs to have a implemented program that can be compiled in the intermediate code.

- Benchmarking
  Certain methods ([4], [9], [12], [13], [17]) use benchmarks which are programs used to measure characteristics of a processor. There are some limitations to the technique:
  1. It is very difficult to explain benchmark results from the characteristics of the machines.
  2. It is not clear how to combine individual measurements to obtain a meaningful evaluation of various systems.
  3. Using benchmark results it is not possible to predict and/or extrapolate the expected performance for arbitrary programs.
  4. The benchmarks may still not be representative, and/or not do the kind of computation expected.
  5. The large variability in the performance of highly optimized computers is difficult to characterize with benchmarks.

Our method takes some ideas from Saavedra (benchmarking of basic operations) and Axelsson (model of processor).

3. Definitions

A few definitions are needed to understand what follows. The execution time of a real-time system is based on the software characteristics, the *Code Profile,* and on the hardware/compiler characteristics, the *Processor Performance Model.*

- **The Code Profile**

  The Code Profile describes the amount of a specific “construct”, or basic operations, that the program contains. It is constituted of numbers (c_i) which represent how much these different constructs are used. The coefficients (c_i) may be deducted from a analysis of the program or with measurements and linear regression. The Code Profile is related to the Processor Performance model in so far as a computation of both of them allows us to estimate the performance of the program on the hardware characterized by the Processor Performance model.

- **The Processor Performance model**

  The hardware is characterized by a processor performance model, P, which in the same way is a function from labels to numbers. Those numbers correspond to the hardware features which influence the execution time. Clock frequency, cache memory size, number of clock cycles per instruction, instruction set are examples of such features. Given these numbers and the corresponding set of numbers for how much the application program “contains” or is influenced by the different factors, the execution time can be calculated as a sum in the same way as above.

  A compiler does not produce the same code for different machines. They actually adapt the compiled code to the hardware architecture. Therefore, the processor performance model is actually a model of a certain compiler and processor combination. The processor perfomance model might be a set of execution times of narrow programs; these times would be obtained by measurements.

- **Narrow programs**

  The narrow programs are the set of small programs which represent the basic operations of a program. Their execution time is used as the characteristics of the processor performance model. They are used in the linear regression as predictor variables.

- **Application programs**

  The application programs are general large programs used to establish the method of estimation. The method is applied to determine their execution times, corresponding to a response variable in the statistical analysis.

4. Statistic analysis

This chapter explains the basics of linear regression which are necessary to understand the next chapters. It deals only with the mathematical points, so this part could be skiped by people, that have a sufficient background in statistics. More information can be found in [7].

4.1. Linear regression

One value is assumed to be a linear function of some others, and linear regression is a method for approximating that function based on a set of observations. The goal of the linear regression is to find the different factors and their coefficient which determine another value, called response variable. Regression analysis is distinguished from other types of statistical analyses in that the goal is to express the response variable as a function of the predictor variables. Once such an expression is obtained the relationship can be utilized to predict values of the response variables, identify which variables affect the response most, or verify hypothesized causal models of the response. We have the following equation:

$$Y = \alpha + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4 + \beta_5 X_5 + \ldots + \beta_p X_p + \varepsilon \qquad (1)$$

where

Y is the response variable,

$(\beta_1, \beta_2, \ldots, \beta_p)$ are the (unknown) regression coefficients,

α is a constant,

$X_1, X_2, X_3, \ldots, X_p$ are the p predictor variables,

ε is the error.

With an amount of data (predictors and response variables) it is possible to determine an estimation of the regression coefficients. These data have to be correct, i.e., they have to satisfy the following assumptions:

* The model is correctly specified (i.e. all relevant predictor variables are included in the model and they appropriately enter and affect the response as linear terms).
* Predictor variables are non-stochastic and are measured without error.
* Model error terms have zero means, are uncorrelated, and have constant variance.
* Model error terms follow a normal probability distribution.

The purpose of linear regression is to establish a quantitative relationship between a group of predictor variables and the response. The method to calculate the estimations of the regression coefficients uses the least squares parameter estimation. It minimizes the average of the squared residuals r_i, which are the difference between the response variables (Y_i) and the estimates of these response variables ($\hat{Y}_i$).

$$\sum_{i=1}^{n} r_i^2 = \sum_{i=1}^{n} \left(Y_i - \hat{Y}_i \right)^2$$

$$= \sum_{i=1}^{n} \left(Y_i - \hat{\alpha} - \hat{\beta}_1 X_{i1} - \hat{\beta}_2 X_{i2} - \ldots - \hat{\beta}_p X_{ip} \right)^2$$

where $\hat{Y}$ is the estimated response,

$\hat{\alpha}, \hat{\beta}_1, \hat{\beta}_2, \ldots, \hat{\beta}_p$ are the estimates of the regression coefficients,

and X_{ij} are the observations corresponding to the reponse Y_i.

Matrix algebra formulation

Because of the large amount of values, matrices are very useful to compute the linear regression. (1) is written

$\underline{Y} = X^* \underline{B} + \underline{\varepsilon}$ with matrix formulation.

$\underline{Y}$ is the vector formed by the responses $Y_1, Y_2, ..., Y_n$,

X^* is the matrix formed by the vectors $\left[\underline{1}\ \underline{X}_1\ \underline{X}_2\ \ldots\ \underline{X}_p\right]$, a column of n ones and n observations on each of the p predictor variables,

$\underline{B}$ is the vector of model parameter $(\alpha, \beta_1, \beta_2, \ldots, \beta_p)$

and $\underline{\varepsilon}$ is a vector of the n random error terms.

Equivalently, the constant term can be partioned from X^* and $\underline{B}$ to give the following representation of the model:

$\underline{Y} = \alpha \underline{1} + X\underline{\beta}' + \underline{\varepsilon}$

where $X = (\underline{X}_1, \underline{X}_2, \ldots, \underline{X}_p)$,

$\underline{\beta}' = (\beta_1, \beta_2, \ldots, \beta_p)$.

The least squares estimators are given by:

$\hat{\underline{B}} = (X^{*\prime} X^*)^{-1} X^{*\prime} \underline{Y}$

where $X^{*\prime}$ is the transpose of the matrix X^*.

4.2. Selection method

If there are many potential variables available in a regression analysis, a selection of a subset may be done. We usually do not know which ones are the most influential on a response but we want to reduce them to a smaller pool of variables. The goal in the variable selection process is to include the most important predictor variables in the final prediction equation, and exclude the unimportant.

The selection method will be more appropriate if the data set reveals no obvious error assumption violation and if the normal probability plot (graphic showing the distribution of the error) is approximately linear and the other residual plots (graphics showing the distribution of the residuals as a function of the predictor variables) show horizontal scatter. In the same way, multicolinearity between the variables must be avoided.

If we add a new predictor variable to the equation of a linear regression, the estimation will always be equal or better, but we have to consider if the gain is really interesting and if the set of predictors without this new one could not offer an adequate fit. The advantage of such a reduction would more than balance a small loss in the predictive ability of the regression equation.

The selection method uses a criterion to determine whether if the improvement of precision is sufficient. The next section describes the F-statistic which is used as a criterion.

4.2.1. F statistic

The regression equation is written

$$\underline{Y} = X\underline{B} + \varepsilon$$

Consider now that we split the set of predictor variables X in two parts:
X_1 which contains the n values of the k predictor variables that are to be retained and
X_2 which contains the n values of the r predictor variables that are to be deleted.
The new regression equation is

$$\underline{Y} = X_1\underline{B_1} + X_2\underline{B_2} + \varepsilon$$

The hypothesis H_0: $\underline{B_2} = \underline{0}$ vs. H_a: $\underline{B_2} \neq \underline{0}$ can be tested with the following statistic:

$$F = \frac{(SSR - SSR_1)/r}{MSE}$$

where SSR is the regression sum of squares due to regression,

$$SSR = \sum_{i=1}^{n} (\hat{Y}_i - \bar{Y})^2$$

SSR_1 is regression sum of squares from the least squares fit to the *k* variables in X_1,
MSE is the residual mean square

MSE=SSE/(n-p-1) (with $SSE = \sum r_i^2$, the residual sum of squares),
r is the degree of freedom.

Large values of F lead one to reject the hypothesis that $\underline{B_2} = \underline{0}$.

4.2.2. Forward selection

The Forward Selection (FS) method adds predictor variables one at a time to the predict equation. This process continues until either all predictors are included or until the selection criterion $F_j < F_{1-\gamma}(1, n-s-2)$ is satisfied. This criterion is derived from the F-statistic described in the previous section. The variable considered is the one that produces the greatest decrease in SSE_1 (the residual sum of squares fit to the *k* variables in X_1).

The F_j statistic tests whether in the reduced model

$$Y = \alpha + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_s X_s + \beta_j X_j + \varepsilon$$

Let's take the example given in [7]:

Table 1: Forward selection method with four variables

stage(s)	Variables in the subset	SSR	SSE_1	MSE_1	F_j
1	X_1	4,993.78	889,886.75	1964.43	2.54
1	X_2	20,890.21	873,990.31	1929.34	10.83
1	X_3	301,351.25	593,529.25	1310.22	230.00
1	X_4	313,342.31	581,538.19	1283.75	244.08
2	X_1, X_4	313,700.56	581,179.94	1285.80	0.28
2	X_2, X_4	326,852.06	568,028.44	1256.70	10.75
2	X_3, X_4	561,096.31	333,784.19	738.46	335.52
3	X_1, X_3, X_4	516,152.50	333,728.00	739.97	0.08
3	X_2, X_3, X_4	574,910.75	319,969.75	709.47	19.47
4	X_1, X_2, X_3, X_4	574,936.31	319,944.19	710.99	0.04

Step1: each of the four predictor variables is evaluated separately in single variable regression analyses, $F_j = SSR_1 / (MSE_1)$, in [7] it is suggested to take $\gamma = 0.25$ and there are 456 values for each predictor variable. So the criterion to continue the process is:

$$F_j > F_{0.75}(1, 453) = 1.33$$

The value 1.33 is given by a table. This criterion may be passed out, it is only used to have an idea of "when the prediction is appropriate".
The predictor variable which has to be chosen is the one which has the largest value for F_j.
The stage one analysis indicates that X_4 should be the first predictor variable selected.

Step2: three regressions must be run. The SSE found for the predictor equation for X_4 in stage one is used to calculate F_j. For step 2 and the following ones, F_j is given by:

$$F_j = \frac{SSE - SSE_1}{MSE_1}$$

The cutoff value is $F_{0.75}(1, 452) = 1.33$. The value of F_j for X_4 is the largest one and respects the criterion, so it can be added to the selection.

Step3: the FS procedure terminates since the F_j statistic for stage 4 is less than 1.33. Thus the forward selection has chosen the best subset of size three and the best overall subset, (X_2, X_3, X_4).

Forward selection does not always lead to optimal subsets of a given size. Another criticism would be that the F_j statistics are not true F random variables. The advantages are that it requires much less work than all possible regressions, it is a simple method to understand and apply, and it generally results in good, although perhaps not the best, subsets.
The order in which predictor variables are selected is not necessarily related to their actual importance.

5. Execution time estimation

This study aims at showing that the execution time may be seen as a vector multiplication between the code profile and the processor performance model. The elements of these vectors are called characterizing factors.

This work does not include the partition of a program in basic operations or the composition of the processor performance model. The method is a way to estimate the execution time of a program which is already implemented, by using time measurements and regression analysis. However, this chapter deals with a more general subject because the studied method should be applied to the early design. The general subject is about a method of estimation of the performance of a non available processor for a not yet implemented real-time software. Then the part of this general goal which has been the center of this study will be explained.

5.1. Method at early design

• Characterizing factors

The characterizing factors are the elements of the Code Profile and the Processor Performance Model (defined in chapter 3). The execution time estimation depends, first, on the choice of the characterizing factors. The determination of the set of factors is based on engineering background. They can be changed or extended to give more complete coverage or more accurate estimates. Different situations during the design also might require different sets of factors. The execution time can be seen like the product of the two vectors of characterizing factors: the Code Profile ($\{c_i\}$) and the Processor Performance model ($\{p_i\}$).

$$Q\left\langle \begin{bmatrix} p1 \\ p2 \\ p3 \\ p4 \\ \end{bmatrix}^T \times \begin{bmatrix} c1 \\ c2 \\ c3 \\ c4 \\ \end{bmatrix} \right\rangle = Texec$$

Q is a normalizing function.

• Regression analysis

The studied method uses the regression analysis. A more complete explanation of this technique are given in chapter 4.

Example: given X_i the independent variables, b_i the estimated factors, and Y the response variable. The linear equation is: $Y = a + b_1X_1 + b_2X_2 + b_3X_3 + ... + b_nX_n$.

The vectors $\{X_i\}$ and $\{b_i\}$ correspond to the vectors $\{p_i\}$ and $\{c_i\}$. We have to decide which of the code profile or the processor performance model will be the independent variables. The choice is liable to the complexity of determining one or the other.

- **Determining the processor performance model**

It can be derived from the data-sheets and from measurements.

- **Determining the code profile**

Given the differences to older programs, the code profile of a new planned program can be estimated. That means that the older programs must have similar characteristics as the new one.

If it is possible to determine by derivation a new processor performance model and a new code profile, then it is possible to estimate the execution time of this combination of hardware and software without any measurements.

The experimental measurements must be representative of "typical" occurrences of the parameters in real programs.

• Approach

The final goal of this method could be summarized with the help of the following table.

Proc id	clock freq	cache mem	I/O speed	Time (prg1)	Time (prg2)	Time (prg3)	Time (prg4)
1	33 MHz	256 K	10 Mbit/s	4.3 ms	6.2 ms	0.26 ms	**?**
2	80 MHz	172 K	15 Mbit/s	2.1 ms	1.2 ms	0.10 ms	**?**
3	15 MHz	256 K	10 Mbit/s	8.9 ms	10.7 ms	10.7 ms	**?**
4	2 MHz	28 K	4 Mbit/s	48 ms	40.3 ms	40.3 ms	**?**
5	125 MHz	256 K	40 Mbit/s	1.2 ms	0.56 ms	0.56 ms	**?**
6	75 MHz	18 K	20 Mbit/s	**?**	**?**	**?**	**??**

Consider we have a processor defined by the following characteristics: *clock freq, cache mem, I/O speed.*

The execution time of prg1, prg2 and prg3 must be determined on the processors 1..5.

With regression analysis, we can derive the code profile of prg1, prg2 and prg3.

If prg4 is a known combination of prg1, prg2 and prg3, then we can derive its code profile from the corresponding code profiles.

We can calculate the processor performance model of proc 6 from datasheets.

Now the execution time of prg4 on proc 6 can be estimated, without knowing the execution time of prg4 on 1 ..5(1) and without knowing the execution time of prg1, prg2 and prg3 on proc 6(2).

The conclusion of the argument above is that:

(1) We can use this method for a not implemented program.

(2) We can use this method for a not available processor.

A subset of Ada has been used in this study because the method should be applicable to a control system design which will be implemented in this language. Only sequential programs have been considered. The method, if its benefits are satisfactory, may be extended subsequently.

5.2. Method overview

This research has been focused on the analysis of the use of linear regression to estimate the execution time of a program on a particular machine. It will be shown that the performance of a processor can be characterized by a set of narrow programs. The execution times of these narrow programs are used as predictor variables in the linear regression. The way to measure the time of these narrow programs and the application program on different machines is one of the most important points in this method.

The method might be split into two parts: the measuring and the statistical analysis. The measuring includes the writing of the narrow programs, of the measurement program, and finally the measurements themselves, with all the needed considerations in order to make correct measurements. The second part is the analysis of the data of these measurements, and the computation of the execution time estimate of the application program on another machine. This analysis tries to find the most suitable coefficients in the linear equation. Different techniques can be used in the statistic analysis.

The different experiments done in order to improve the method of estimation have used different application programs, so the method should be applicable to different kinds of programs and not only for one in particular.

6. Method

This chapter describes the method to estimate the execution time of a program on different machines. The way of choosing and writing the narrow programs, of accomplishing the measurement, and of doing the linear regression efficiently will be detailed in this part. Figure 2 gives an abstract idea of the method. With the execution times of the narrow and application programs on different computers, we can determine the regression parameters of an application program with a statistical analysis. Those parameters multiplied by the execution time of the narrow programs give an estimate of the execution time of this application program.

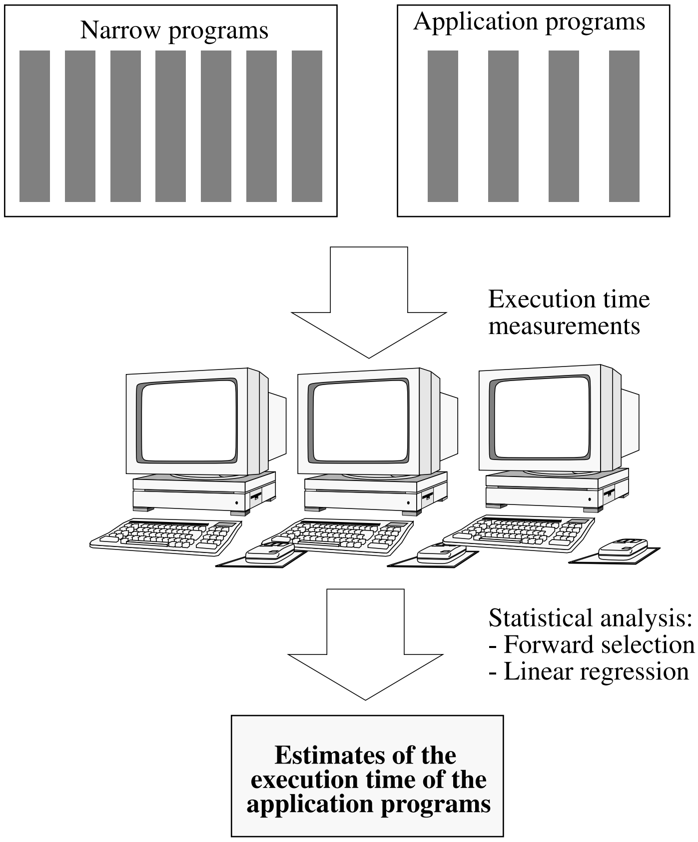


Figure 2: Principle of the method

6.1. Choice of characterizing factors

The narrow programs must be general characteristics that are valid for any program. Thus they have to be basic enough to be applicable to any kind of algorithm and close to the simple hardware operations.

Decomposing a implemented program can be easy, though the choice of the characterizing factors is important. Our problem is to find how to characterize a program that does not yet exist. This study must be associated with the choice of a good design method.

Possible categories of parameters are:

- arithmetic and logical

• procedure calls
• array references
• iterations

This was proposed in different articles of other researches ([2], [14]), but in our case the basic operations must be both representative of all the operations in a program and linear independent. The last condition is one of the assumptions in order to make a regression analysis. This point is developed in chapter 4.

The parameters mentioned above are among the most common operations in a software, but if we look at them carefully we can notice that some of them are indeed a combination of more basic operations, in regard to the hardware execution. Nothing, except experience, can prove that a choice of characterizing factors is better than another one. The first criterion of the selection is that the operations must be one of the most frequently used operations in a program. The second point is to check if the basic operation is not (if we consider the corresponding hardware operations) a combination of more basic operations. An exponential, for example, is a set of multiplications. Then, considering all kinds of machines, we have to check if two operations do not depend on the same hardware operations. That would mean that they are linear dependent.

According to these considerations, a good set of narrow programs should include:
• a program of additions,
• a program of multiplications,
• a program of assignments (memory accesses),
• a program of comparisons (if, else),
• a program of functions calls,
• a program with an important number of loops,
• a program with pointers manipulations.

This set of programs is constructed for the Ada language, but could be constructed for most of the current languages, with eventually some corrections in order to have a better fit.

6.2. Dual-loop

Measurement of related code on similar processors is important as input data to the estimation. The critical assumption in the dual loop execution time measurement method is that textually equivalent source code and input data always have the same execution time given that the compiler and the hardware are identical.

The principle is simple: the program consists of two loops - one is the dummy to measure the overhead of iteration and the other contains the code to be measured. The time before and after the dummy loop is measured and likewise for the loop containing the code. The execution time for the code is then the difference divided by the number of loops.

```
take t1
loop N times do
<code>
```

```
<dummy code>
end loop
take t2
loop N times do
<dummy code>
end loop
take t3
output t4=(2t2-t1-t3)/N
```

The dummy code has to do something, otherwise the compiler will skip the loop which contains it. Thus it is obvious that the compiled code has to be checked. Moreover this dummy code must have no influence on the measured code, it must execute a real operation but this one must have no effect on the execution time of the code to be measured. Care must be taken when the dual loop and the dummy code are implemented.

The dual-loop handles the problem of an imprecise clock by extending the duration of the test to a length that the clock can readily measure. The tested program is run several times inside a loop. But this loop adds time and this time must be factored out, that is why there is this second loop with no test program.

However, the accuracy of a dual-loop benchmark depends upon a highly specific set of circumstances (see [1]) and cannot be controlled by a general technique when the benchmark is written.

6.3. Measurements

To find out the coefficients of regression for the execution time estimation of a program, it is necessary to measure the execution time of this program and of the narrow programs. These measurements must be accomplished on different machines. Since the aim is to estimate the execution time of the program on a particular machine, the measurement must be done on a set of machines which must accumulate all the characteristics of the particular machine. The measurements of the execution time of the narrow programs must also be done on the particular machine.

The data concerning the narrow programs are the features of each machine. The narrow programs are so basic that they characterize the hardware operations of a processor. The data from the measurement of their execution time is then a kind of processor performance model, as it has been defined before in this report (chapter 3).

The more measurements you have, i.e. the more different processors you have, the more reliable results you will get. But the machines have to be really different. If you use, for example, a set of machines which differ only by the clock frequency to make the regression model and if you want to estimate the execution time of the program on a machine which has, in comparison with the other ones, differences other than the clock frequency, the results will be unsatisfactory.

The times of the narrow programs can obviously be conserved and be reused for another program if its coefficients of regression have been calculated previously, with other machines. In the same way, we can conserve the data concerning the application programs: the execution times might be included in a new analysis with other machines and the coefficients of regression are useful for any machines if we are able to measure the execution time of the narrow programs.

Even if the dual-loop avoids certain problems of time measurement, care must be taken at the moment of measurement. Nothing has to disturb it, no input/output during the execution of the tested program, no other process using the processor in the same time. Examples of prevention are given in section 7.2.

6.4. Normalization

It is necessary to normalize the data in order to make the regression analysis, otherwise the figures are not of the same magnitude. The purpose of the normalization is to avoid a too big difference between the data, which contributes to larger error in the linear regression. For our case, the normalizing function is a multiplication of the times for each machine by their clock frequency. In other words, the values of the measurements of all the programs for a processor have to be multiplied by the clock frequency of this machine. This choice is explained by the large importance of the clock frequency in the performance of a processor.

The following diagrams show the results of the linear regression of an application program for 10 Sparcstations. The residuals are the difference between the measured values and the estimates (a confidence interval is given for each residual, but it has no importance for this problem). The first diagram is without the normalization, the second one is with the normalization (multiplication by the clock frequency). The clock frequency of the machines of this sample is between 33 and 85 MHz. The residuals, without normalization, are 10 times larger than the ones with normalization.

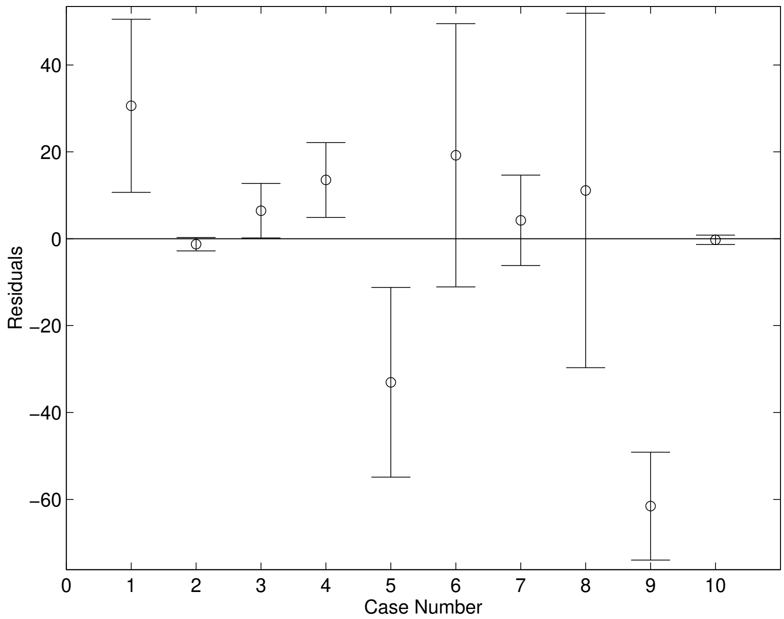


Figure 3: Residuals from the regression for 10 processors, without normalization

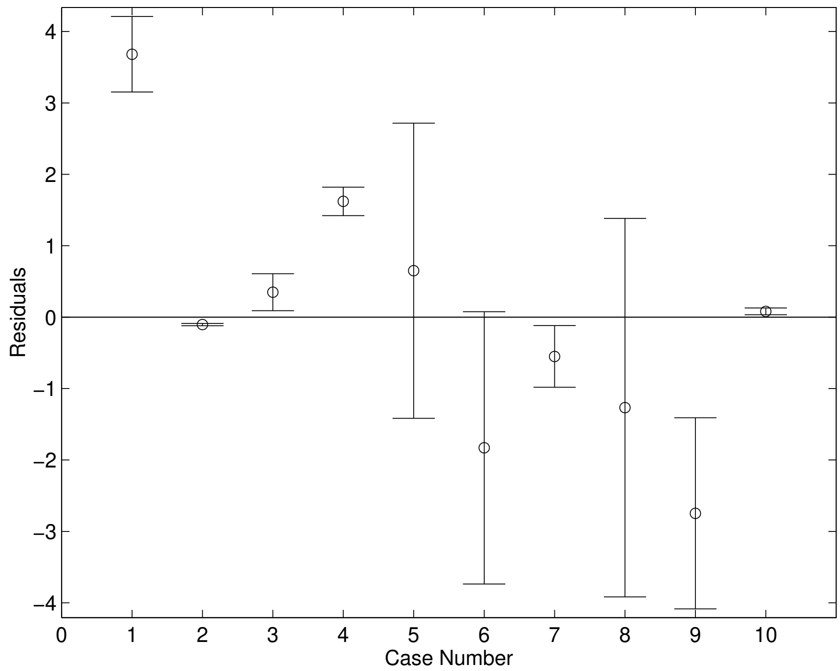


Figure 4: Residuals from the regression for the same 10 processors, with normalizat.

6.5. Linear regression

The data is now ready for the linear regression, but it is nevertheless necessary to check the raw data to find eventual obvious anomalies. For example, one or two incorrectly transcribed data values can severely distort estimates of regression coefficients. Yet because the estimated model attempts to fit all the data points could lead to a good fit to the remaining ones. Care is needed in the initial specification of variables in a regression analysis as well as in later transformations and the insertion and deletion of individual predictor variables.

Remember that the linear equation is:

$\underline{Y} = X^{*}\underline{B} + \underline{\varepsilon}$ with matrix formulation.

$\underline{Y}$ is the vector formed by the responses $Y_1, Y_2, ..., Y_n$,

X^* is the matrix formed by the vectors of predictor variables $\left[\underline{X}_1\ \underline{X}_2\ \cdots\ \underline{X}_p\right]$

The narrow programs, or more precisely the values of the execution time of the narrow programs, multiplied by the clock frequency (see "Normalization" above), are the predictor variables. Each vector $\underline{X}_i$ is the normalized execution time of a narrow program on the different machines.

$\underline{Y}$ is the normalized execution time of an application program for the set of machines.

Obviously a linear regression is needed for each application program, since each one has different regression coefficients.
The result of the linear regression is the coefficients corresponding to the regression model of an application program, which will be used to calculate the estimate of the execution time of this application program for another machine.

• Forward Selection

Each application program does not necessarily need all the narrow programs as predictor variables. Depending on the application program, some narrow programs have little or not importance in the estimation. The addition of one might bring an insignificant improvement of the estimates. The Forward Selection, described in [7], is a way to select the most important predictor variables in a linear regression.

This method is based on the increase of the Sum of Squares and selects one by one the predictor variable that has the most important weight in this increase. An F-statistic checks whether it is sufficient or not. The method is probably not the one which always gives the best set of coefficients, but the estimates calculated with these coefficients are always good (if coefficients are adequate to the regression) and it is a simple and fast method to do a selection.

The result of the Forward Selection is a set of the best predictor variables. A simple calculation gives then the corresponding regression parameter (or coefficient of regression), $\underline{B}$.

• Error of the estimation

Although the method is done rigorously, the estimates calculated with the regression parameters might be really wrong. Therefore, the estimates have to be computed and compared with the response variables.

$\underline{\hat{Y}} = X^{*}\underline{B}$ where $\underline{\hat{Y}}$ is the vector of estimates of the vector of response variables $\underline{Y}$.

The percentage of error is a good way to do this comparison and it will be useful to compute the worst value of the execution time of the program on another machine. If the error is too large, check the computation of the values. With a wide set of data it is easy to make mistakes like omission of a factor or wrong implementation of the computation. However a wrong estimate might be due to bad measurements (bad conditions, too much disturbances) or to a wrong set of machines which does not fit the regression.

• Non negative least squares

There is another way to do the linear regression, the coefficients of regression can be found so that they are not negative. Then they are all positive and Y, the execution time of an application program, is a sum of positive values. The values (X_iB_i) do not compensate each other but add a complementary information or precision to the other ones.

The non-negative independent variables are computed through the use of Lagrangian multipliers. The problem is mathematically stated as

$$\min_{x} ||Ax - b||^2 \qquad \text{such that } x \geq 0$$

where A and the vector b are the coefficients of the objective function. The vector x, of independent variables is restricted to be negative.

7. Experiments

Regression analysis needs a lot of data to be efficient. That is why many measurements have been done on different machines. The next sections deal with the choice of programs used for the measurements and the way of measuring execution time.

7.1. Testing programs

The choice of narrow programs has been explained in chapter 6.. Their implementation is not tricky but the problem is to avoid changes due to compiler optimization. The tested programs are so simple that, after compilation, the running code does not do what is expected and the measurements are totally wrong. The assembler code of the first programs has been checked in order to know how the compiler optimizes the Ada code and to find out how to avoid this optimization. Sparcstations use a pipeline and are able to execute an instruction and have, in the same time, another instruction in the pipeline. For example, the instruction following a branch will be executed in the same time as the branch. If you do not want it you have to use the instruction *NOP* (no operation) after the instruction that makes the branch .

As mentioned in a previous section, the compilers do not always generate the same code. For this reason, all the compilations have been done with the same compiler, Gnat95, version 3.03. Gnat is a freeware available for different operating systems such as Unix, Dos, OS2, and Windows.
The set of Ada programs is composed of a package containing the narrow programs, a package for each benchmark (and all the packages connected to them) and a main program.

7.1.1. Narrow programs

The narrow programs do the following operations: additions, multiplications, memory assignments, comparison, functions calls, loops, and pointer assignments. They are actually procedures to make the measurements easier. The main program (dual-loop) can call them to measure their execution time.

Some of them are very short procedures, since only one operation needed to be characterized. Remember that this operation must be representative of an hardware operation of the processor, so it has to be very basic, but the set of narrow programs must take in account all the characteristics of the processor. To find out the different narrow programs both the software and the hardware must be considered. The experiment has been done with Ada, so all simple operations that could be used in this language have been reviewed. When it was possible to break them up, we excluded them but kept the operations which form them. An access to an array, for example, is a combination of comparisons, additions and assignments.

The first narrow program is the procedure `addition` which is only one addition of two integers (A := A + B). The procedure multiplication is of the same type, it is a multiplication of two integers. Although there is an assignment in the two previous programs, it was necessary to write a program of assignments (or memory call), because a block of instructions might have only assignments without other operations. The fourth one has several comparisons (if, then, else). Then comes the program which calls the addition program. This one takes into account the time needed to do a function call. Then a program containing several loops. This one could have been split in a set of additions and a set of comparisons, but a loop is so important in most programs that we decided to have a coefficient for this. The Forward Selection (see 4.2.) can determine whether this coefficient is useful for the regression or not. Then, the last program is a program which uses a lot of pointers.

7.1.2. Application programs

The application programs are real programs which have been adapted to the measurements. They may not contain any input/output or tasks, because an input/output does not have a regular time of execution and would distort the computations. All programs have some input/output, so those instructions have been removed and, in order to avoid a possible optimization, other instructions have been added instead. We wanted the programs to be quite large, with different kinds of instructions, and they have to be representative of most of software. Three of them are well-known benchmarks. They are parts of ACES, the Ada Compiler Evaluation System. Dhrystone and Whetstone are the most known.

7.1.3. Dual-loop

There is a main program which contains the Dual-loop (see 6.2.) used for the time measurement and a menu allowing the user to choose the procedures he wants to test and to change the number of loops. This program has different parts which measure the time, print the time on the screen and in a file, and choose the program to measure. The reason of this partition is to avoid any optimization inside the dual-loop and any disturbance with the clock.

The number of programs to measure was too large to write a dual-loop for each one and we wanted the user may choose which program he wants, then each procedure is called through a pointer and the measurement is done with the procedure assigned to this pointer.

Some compilers do not compile an instruction that has values that are not reused in the program. All narrow and application program have two integers as parameter and modify their value. A study of these modifications must be done because they sometimes might generate errors. The compiler used for this experiment does not do this kind of optimization, but the program has been written in order to be adaptable for other Ada compilers.

7.1.4. Time

Measuring the execution time of a program is not an easy task. It is essential to have a precise and accurate clock. The values given by the clock of a CPU are not really precise. Ada 95 has a package Real_Time which defines low-level timer operations. This clock model progresses monotonically and the irregularities are strictly bounded. After the system starts, the clock is not allowed to be reset by an operator, the underlying system, or the run-time environment. Real_Time.Clock is used for computations of physical parameters based on time, and scheduling of task execution to satisfy real-time constraints. With this clock and the dual-loop, we had values accurate enough for the analysis.

7.2. Measurements

Many measurements have been done during this study. The first ones were to see the best way to accomplish them and the other ones have been used for the regression analysis and for testing their accuracy.

7.2.1. What to measure, on which processors

The execution time of the narrow and application programs have been measured on 10 Sparc stations (Sun) and 3 PCs. It was exactly the same programs. For all the Sparcstations it was the same code and compiled programs, and for the PC the code was the same. The compiler was Gnat-3.03, the Unix version for the Sparcstations and the Linux version for the PC.

The DOS version for PC could not be used since the package of RealTime was not implemented yet.

It was essential to have the same compiler, because the way to compile is different from one compiler to another one and the measured performance would be distorted. That is why the processor performance model has been defined as a combination of the hardware and the compiler. An instruction is not always translated in the same way by two different compilers, even if they are both ADA95 compilers and the optimization might be different.

The Sparcstations used in the experiment are all different. They have different clock frequency, memory size, cache memory, etc. More details are given in Appendix 9. The redundancy of the characteristics must be avoided.

7.2.2. Number of measurements and loops

Due to the precision of the clock frequency the programs have been run several times. The problem was to know how much time they need to be executed. Saavedra in [15] made some tests in order to find the necessary magnitude of the number of loops. He ran several experiments using different values for the number of loops in several machines. Each test has been run at least 0.1, 0.2, 0.5, 1.0, 2.0 and 4.0 seconds and he obtained measurements for 5, 10, and 20 observations.

For a fixed number of observations the width of the confidence interval of the measurements decreases as the time of the test increases, but for small numbers of observation there is a limit to how much we can decrease the confidence interval by only increasing the time of the test. The reason for this is that by increasing the length of the test we reduce the variability due to short term variations in the concurrent activity of the system. However the probability of a change in the overall concurrent activity of the system increases with a larger test. This change may produce a greater variance if the size of the sample statistic is small. The best results are obtained for 20 observations and 1 to 2 seconds for the duration of the test.

Because the clock of the Sparcstations has a good resolution, it was decided to run the programs for about 1 second with 5 observations. For any number of observations in the study of Saavedra the confidence interval was really smaller for 1 second than for the other times. I did preliminary tests to know how many loops I needed in order to run the programs one second, but it would be probably better to make some tests inside the loop to be sure that the program is run the wanted time. This could have an influence upon the execution time, hence my choice.

From these five measurements, Saavedra advised to take the average to have a better estimate, even if the minimum value should be closer to the ‘effective’ execution time (without concurrent activity). Some experiments prove it. There was another reason for us to take average rather than the minimum: in real-time we are not interested by the shortest time a task can take but by the time in a real application.

7.2.3. Accuracy of measurements

The quality of the estimates depends obviously on the quality of the measurements. That is why they have to be accomplished with an important care. Some assumptions about an eventually wrong way of proceeding have been disputed through experiments. The following paragraphs deal with this problem of accuracy of measurement.

• Constancy

We started the measurements according to the hypothesis of Altman [1] who made experiments about the constancy of the time measurements of a test benchmark. His conclusions are the following ones. Since each of the tested systems have a consistent pattern of variation, the order of execution of the individual loops has no effect on the times. The compiler places the loop statement without regard to word boundaries. As a consequence, certain loops are aligned more advantageously and require fewer memory access to execute.

• Linearity and number of loops

It was important to check if the execution time of an amount of loops including a tested program increases linearly with the number of loops (Figure 5). Logically if you double the number of loops, the execution time is twice longer, but the way of operating of a computer is so complex that such things need to be checked.

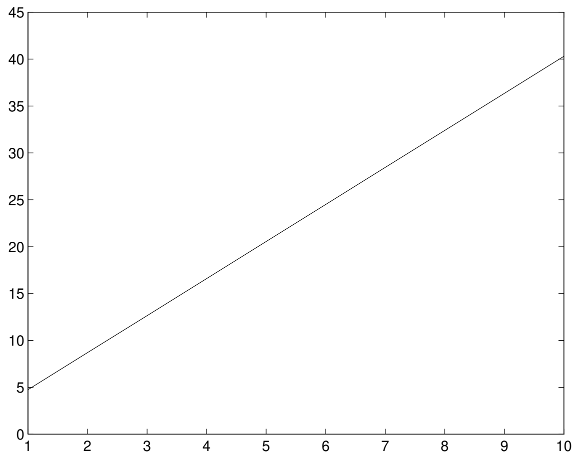


Figure 5: Execution time for all loops = f (number of loops)

We made the measurement of a particular program (narrow program of addition) on a particular machine (Sparcstation) with different numbers of loops, from 1 to 1,000,000. In Figure 6 we can see the results of the measurements.

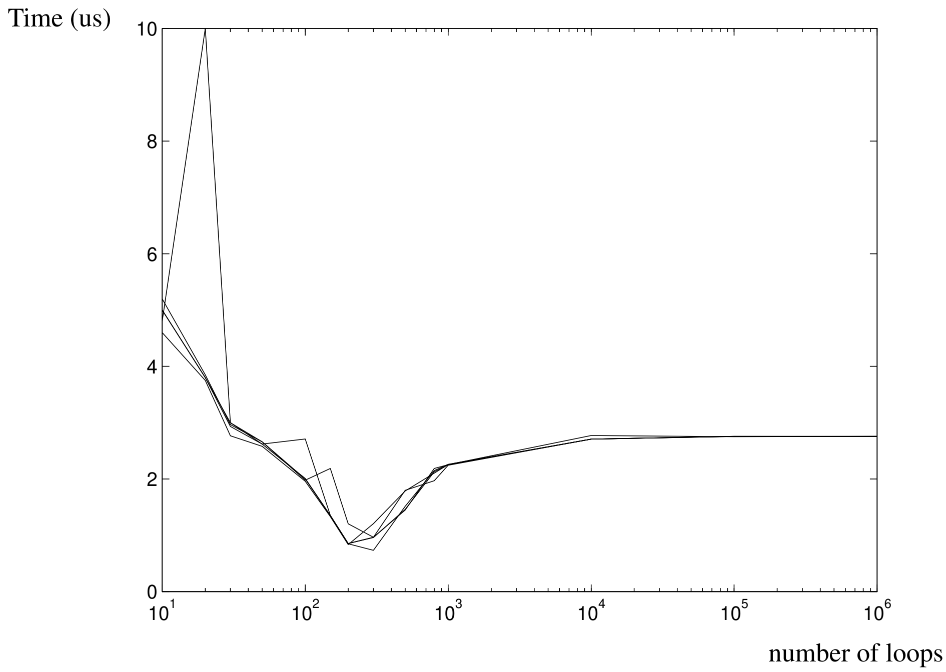


Figure 6: Execution time per loop related to number of loops

In Figure 6, the different times measured have been divided by the number of loops. For a small number of loops, the time is imprecise. Each curve is one of the five measurements for each number of loops. With few loops, there is a important variation within the five measurement. Beyond 10,000 loops the variance is almost null, the execution time is the same for these numbers of loops. The variations before 1000 loops are due to the precision of the clock and the problems generated by reading it in such a short time. However, deeper investigation is needed to explain the minimum values of execution time near 200 loops.

The execution time of the narrow programs have been measured, at least, with 10,000 loops, i.e. when the time is linear with the number of loops.

• Size of measurement program

The narrow programs and the application programs were considered as procedures in the main program of measurement (with the dual loop), thus, after compilation this program was quite huge. It was interesting to know whether the size the whole program could have an influence on the time. To test this two different programs have been measured, both including the dual loop. The first one was the program used for all the measurements, it had then all the packages for all the narrow and application programs (procedures). The second one contained only the package of the narrow programs and only called one of them (addition). The first one was thus 65% larger than the second one.

The means of the execution times had a difference of 0.16%, that is inferior to the precision of the measurement. A more important difference between two programs could have an influence on the time, due to the use of memory cache. Therefore, in our case, adding another program to test in the dual loop has no influence on the execution time of the programs already included in the measurement program.

• File access time

The measurement program writes the measured values in a file and prints on the screen. Unix being a multiprocessing system, the outputs could be done while the program is executing the next measurement and increase the execution time. A second program has been written without outputs between the measurements, the data were stored in an array and copied in a file after the measurements. The comparison between the values of the execution time of the two programs shows there is no significant difference between them; the differences are very small and do not show that a program gives shorter times than the other one. This had been expected, because the file is closed after each measurement, so the data are flushed from the buffer into the file just before.

8. Measures analysis

In this chapter I describe the analysis of the data collected by the measurement, such as the normalization, the use of matrices, the research of best regression parameters and the correctness of the estimates. Matlab has been a good help to accomplish this study, so this chapter starts with a description of its most interesting features for this analysis.

8.1. Matlab

Matlab has been chosen for the facilities it offers for matrix manipulation and for the included statistic tools.

• What is MATLAB ?

Matlab is a technical computing environment for high-performance numeric computation and visualization. Matlab integrates numerical analysis, matrix computation, signal processing, and graphics in an easy-to-use environment where problems and solutions are expressed just as they are written mathematically - without traditional programming.

Matlab is an interactive system whose basic data element is a matrix that does not require dimensioning. It enables you to solve many numerical problems in a fraction of the time that it would take to write a program in an other current language. (For further information please refer to [9]).

• Statistic functions

Matlab features a family of application-specific solutions that are called toolboxes. Those are comprehensive collections of Matlab functions (M-files) that extend the Matlab environment in order to solve particular classes of problems. Among them there is the Statistics toolbox. This one provides statistics functions which are building blocks suitable for use inside other analytical tools.

Two functions are very useful for this study, `regress` and `corrcoef`.
regress:
This function executes a multiple linear regression using least square. Using the same principle as explained in section 4.1., it returns an estimate of the coefficients (B), their corresponding confidence interval of a given confidence, the residuals (r), a confidence interval for each residual and the value of the R-square statistic along with the F and p values for the regression.

corrcoef:
This function returns the correlation coefficient matrix of a matrix. It was necessary to use it to check if there was not to much pairwise correlation between the variables.

nnls:
This function returns the non negative coefficients in a least square regression.

Different functions, such as calculation of the error of the estimates, application of the forward selection, has been written in addition to these.

8.2. First analysis

First, the analysis of only the measurements of the 10 Sparcstations has been done. Its aim was to understand the values, how to manipulate them, so that the regression analysis is correct. It was necessary to normalize the data, because the range of the values was to large and therefore distorted the regression. The most important factor of normalization we found was the clock frequency of each machine. Then the data for each machine has been multiplied by its clock frequency. All the clock frequencies of the Sparcstations are in the range 10 to 100 MHz, it would not have been clever to multiply by 10^6 and have a standard unit, it would have only added a lot of useless digits to the numbers. The data must be of the same order. However, the programs do not have the same execution time and, in order to have the smallest range as possible, it is important to choose a unit responding to the previous requirement.

For the statistical analysis the data must be as following:

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} & x_{16} & x_{17} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} & x_{26} & x_{27} \\ x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & x_{36} & x_{37} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ x_{101} & x_{102} & x_{103} & x_{104} & x_{105} & x_{106} & x_{107} \end{bmatrix}$$

x_{ij}: execution time[1] of narrow program j on machine i.

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \dots \\ y_{10} \end{bmatrix}$$

y_i: execution time of an application program on machine i.

Then we have:
$\{x_{ij}\}$: predictor variables
$\{y_i\}$: response variable.

We do not use a constant (α) in the regression because the value of the execution time of an application program is zero if all the execution times of the narrow programs are zero, which is implicite. Thus the constant must be zero.

1. execution time multiplied by the clock frequency. In the next paragraphs of this chapter the same reduction of words will be used.

A first look at the values of the matrices did not show any irregularity. The data was ready for the linear regression. We first tried to use the data of all the narrow programs, the determination of the coefficients of regression was done for nine machines (for example, machines 1 to 9) and used them to calculate the estimate of the application program on the 10th machine, with the times of the narrow programs on this machine. We noticed then that some narrow programs did not bring interesting precision. It was necessary to factor out which ones must be used in the regression. A method was essential to carry out this.

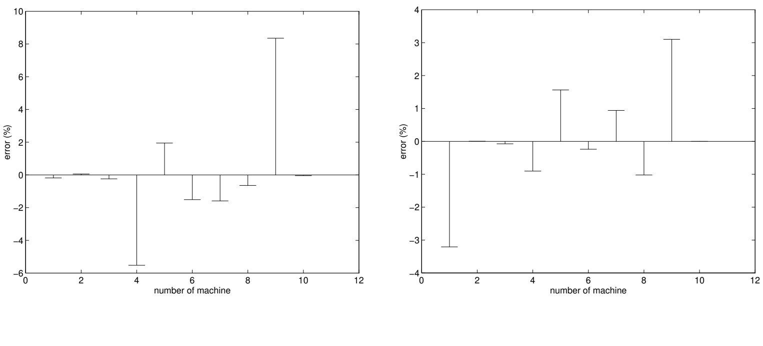
8.3. Regression analysis with only Sparcstations

The method to find which predictor variables fit the best had to be simple but efficient. We chose a step by step method , the Forward Selection. This method, expounded in chapter 4, is based on the increase of the sum of square of residuals (SSE). I wrote a program with Matlab to perform this method with all the four application programs. There was a problem with the F-statistic which defines the criterion for stoping. Indeed, the description in [7] says that the precision introduced by another coefficient is negligible if the F-statistic is inferior to a value from a given table. In our case, the addition of one coefficient might have given a too small improvement of the accurance of the estimate, but two more could give a non negligible contribution to it. For that reason, we decided not to respect this criterion except if the value was really small.

• Results

A table in the appendix shows the different results of the forward selection and of the error of the estimates computed with the predictor variables found thanks to it.

The method selected five or six coefficients for the application programs. With these ones I computed the regression parameters and then the estimates of the execution time of the application programs. The error between each response variable and the corresponding estimate is very small. The average of the error is about 2% for the first applicatiom program, with a maximum of 8.3%. The two other ones have better results, with a maximum of 3.2% and 3.1%. The average for the fourth application program is about 5% with a maximum of 18.5%. Even if a few values of the estimation are over 10%, the other ones are very satisfactory (Figure 7 shows the results for the four application programs).



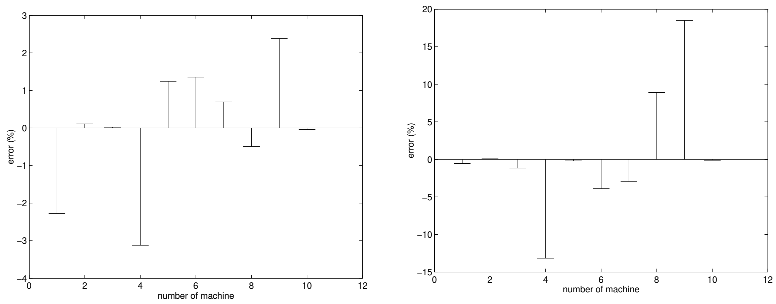


Figure 7: Percentage of error (only Sparcstations)

8.4. Regression analysis with PC and Sparcstations

The method had been applied on quite similar machines, the Sparcstations. It was interesting to see if it was still valid for set of machines which have really different architectures. We added to the set of values the execution time of the same programs on PC.

• First coefficients

A first attempt was to take the coefficients of regression found in the previous analysis ($\underline{B}_{(Sparcstations)}$) and to calculate an estimation of the execution time of the application program on the PC ($\hat{Y}_{i\,(PC)}$).

$$\hat{Y}_{i\,(PC)} = X_{i\,(PC)} \times \underline{B}_{(Sparcstations)}$$

The regression parameters should be the representation of an application program, as a combination of the narrow programs. This representation should be available for any processor, however, the results do not correspond to this logical reasoning. The values of the estimates are sometimes 3 to 4 times larger than the measured values of the execution times of the application programs.

• General regression

A new regression with all the values of PC and Sparcstations was made. The Forward Selection has been used to find the best coefficients of regression. The narrow programs selected are a bit different from the selection with only Sparcstations, but it is difficult to conclude about this because the selection includes always 5 or 6 predictor variables and so only one or two might differ. As expected the error of the estimates is larger than with only Sparcstations.

Table 2: Error for general estimation

Applic. prg	Average	Maximum
1 (Sparcs.)	17%	59%
1 (all)	21%	26%
2 (Sparcs.)	9%	34%
2 (all)	14%	26%

Table 2: Error for general estimation

Applic. prg	Average	Maximum
3 (Sparcs.)	11%	22%
3 (all)	28%	179%
4 (Sparcs.)	22%	79%
4 (all)	29%	105%

In the table 2 the results of the PC have been separated to show their values are often worse than for the Sparcstations (Figure 8 shows the results for the four application programs). The estimation is not as good as for the regression with only the Sparcstations, that means that the model does not correspond sufficiently to the programs, or the machines.
A first conclusion of this is that method is valid for a kind of machine, and a larger error is generated with different types of machines.

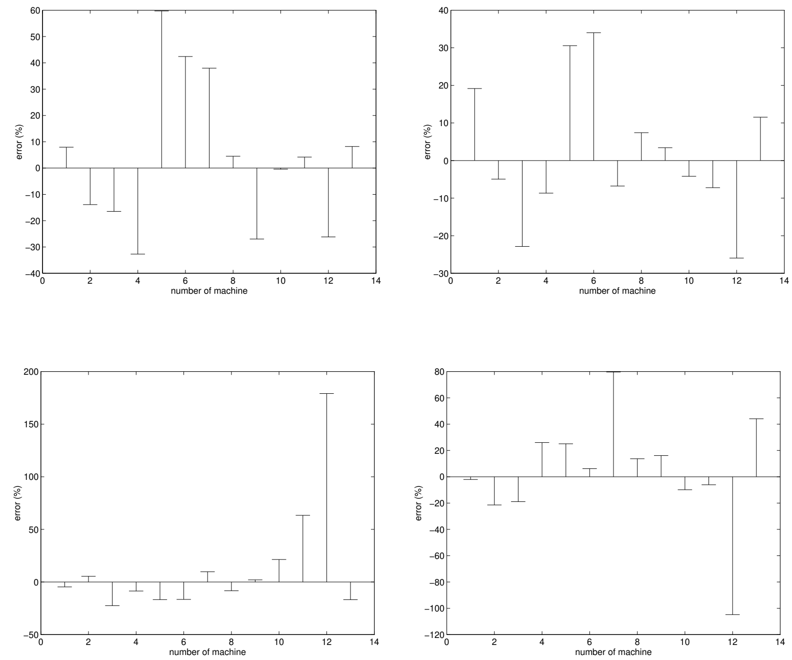


Figure 8: Percentage of error (Sparcstations and PCs)

• Non negative least squares

Because of the relatively poor results, we made a regression with non negative least squares according to the idea that the predictor variables should not compensate each other but be added to each other. I made the same experiments as before with a non negative least square regression.
For the regression with only Sparcstations there are only two regression parameters (i.e. different to zero) for each application programs. The estimates of the execution time of the application programs are not acceptable (averages: 21%, 13%, 5%, 22%; maxima: 36%, 22%, 19%, 51%). The values for the PC with these coefficients are a bit

better than before, they are however still wrong (the estimates are often 2 times larger than the measured execution time).

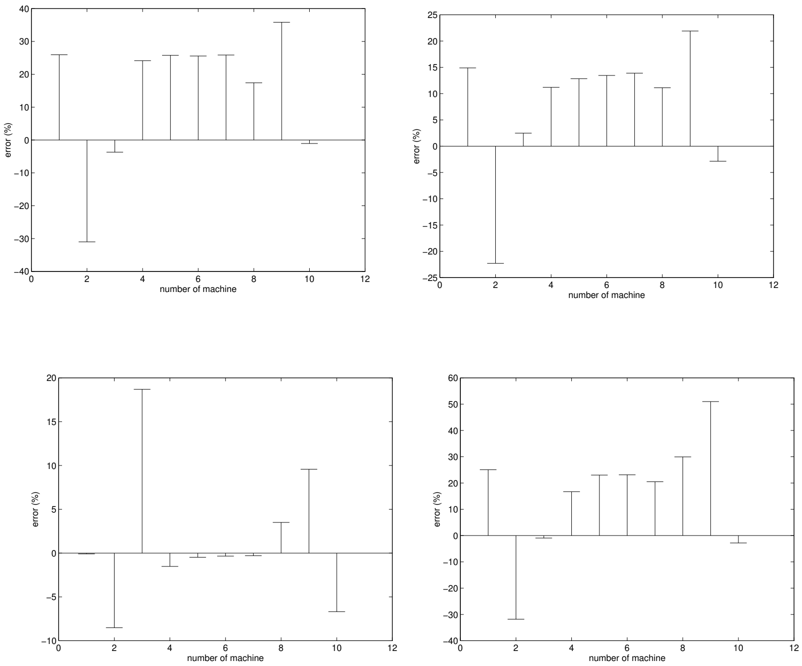


Figure 9: Percentage of error with non-neg. (Sparcstations)

There is only one coefficient of regression for the regression with all data (Sparcstations and PC). The estimates are not so wrong, at least for the two first application programs; the two other ones have a large error for the estimates on the PC (Figure 9 shows the results for the four application programs).

Table 3: Error for general estimation

Applic. prg	Average	Maximum
1 (Sparcs.)	36%	45%
1 (all)	34%	45%
2 (Sparcs.)	26%	33%
2 (all)	22%	33%
3 (Sparcs.)	10%	30%
3 (all)	31%	165%
4 (Sparcs.)	32%	48%
4 (all)	38%	109%

The two first application programs do not have a important difference between Sparcstations and PC, however the error remains large. Some estimates of the other application programs for PC are totaly wrong (maximum of 165% for third one).

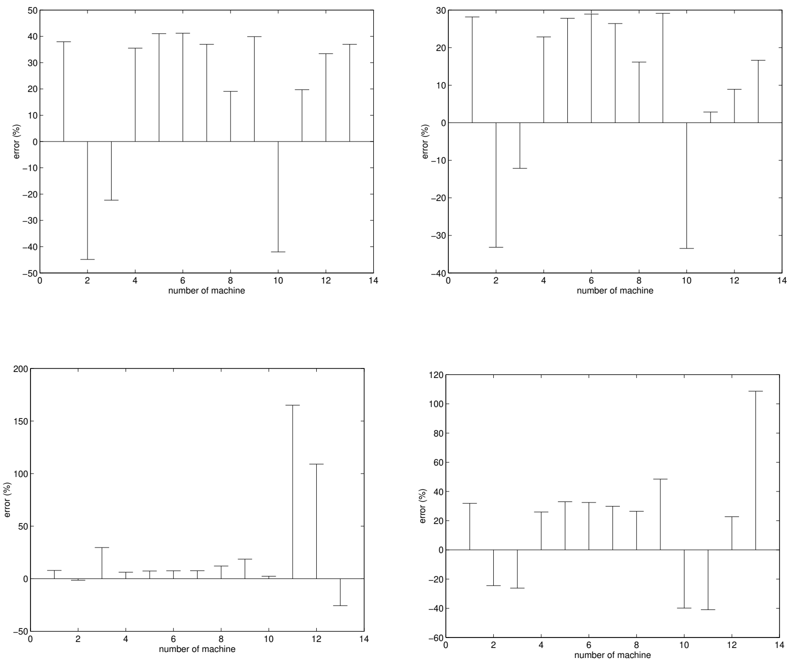


Figure 10: Percentage of error with non neg. (Sparcstations and PCs)

• Attempt to orthogonalize the data.

A mathematical explanation of the error could be the tendency of the vectors of the model (narrow programs) to be almost colinear. If, for example, {a, b, c, d} is a base and v a vector built with base (Figure 11)

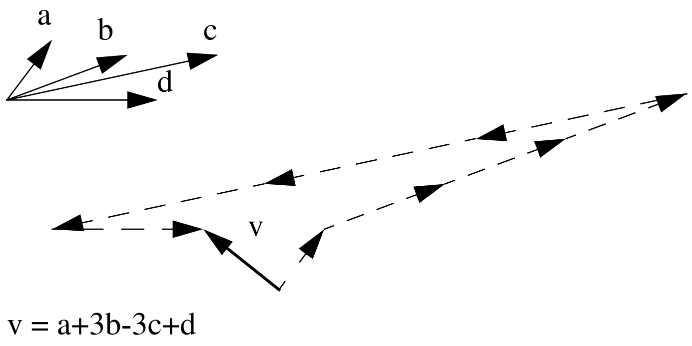


Figure 11: Sum of vectors.

When the vectors of the base have almost the same direction and are quite small, and when the resulting vector is “orthogonal” to them, then a high coefficient of multiplication (k_i) is necessary: v = 200a + 98b - 378c - 45d. If the vectors of the base have a small error (< 10%), this error is multiplied by k_i, that means that the absolute value of the error is k_i times more important.

The vectors built with the execution time of the narrow programs are probably almost colinear, the matrix of covariance shows it. So we have tried to orthogonalize the matrix of data. We chose the vector (U) which was less colinear to the other ones (V_i) and calculated the orthogonal vector(Vo_i) which forms V_i with the projection of V_i on U (Figure 12).

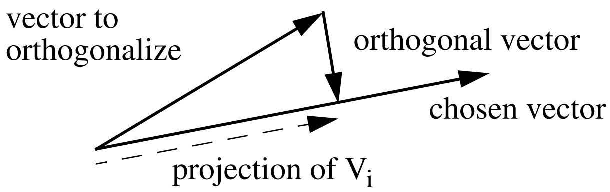


Figure 12: Orthogonalization of vectors.

Then we made the regression with the orthogonal vectors, but the results were not better.

8.5. Discussion

The method of estimation of the software performance on a particular machine is efficient for a set of machines of the same type. The difference between the measured value and the estimation is very small, if we compare with the other methods of estimation. However, the method does not seems to be applicable for a set of different types of machines. The set of Sparcstations is a set of different machines, but still have a similar architecture and some similar components. We are indeed not sure that the imprecision of the estimation with the PC is due to the difference between PC and Sparcstations. The results of the non-negative least squares regression are more regular, at least for the PC, but the estimates with only Sparcstations are worse. The next paragraph proposes different hypothesis which could explain the error.

First, we could have forgotten something when we did the measurements. A improvement of the way of doing them could bring better results. All the disturbances have certainly been avoided, but one might be still present. The programs could also be modified, the dual-loop could be improved and the narrow programs optimized. These last ones seem to characterize a computer, but a characteristic could have been left out of the model. The model needs more different machines in order to characterize a software, then it could be available for any machine. These points concern the measurements, but the statistical analysis could probably be improved. For example, a standardization or transformations avoiding correlation of the data could be useful.

The disparity of the results make it difficult to establish a first relationship between the execution time of narrow programs and processor performance model or code profile. We could only say that for a given operation (addition, multiplication,...) a computer is faster than another one. The narrow error of the estimates with only Sparcstations shows that the narrow programs characterize a processor well.

9. Conclusions

9.1. Summary

The codesign hardware and software in real-time needs a method estimating the performance of a software on a particular hardware. The method must take in account the characteristics of both the program and the processor. This research tended to respond to this requirement with the establishment of a method using statistical analysis. This method is based on the measurement of the execution time of narrow programs. These narrow programs are basic operations of any program and represent the characteristics of a processor (and of the compiler associated to it). A linear regression gives the coefficients needed to calculate the estimation.

This report has shown the importance of having accurate measurements and different precautions to take for having values which be the most exact as possible. The error of the estimates is quite small for a set of different Sparkstations, the results are very satisfying. However, with other machines (PC) added to the set the estimation is not so good. Moreover, it must be taken into account that the Sparcstations are not used for real-time systems. The method must be improved for a set of different types of machines.

9.2. Future work

The results of the research only represent small steps towards a better estimation of real-time softwares performance. There are several areas where we think that important work remains to be done, and this section mentions a few of them.

Some results could be explained by wrong measurements, a method must be found in order to have more trustworthy measurements. It could be interesting to check if there is no context switch during the measurements with the Sparcstations. If there is one or more, it is necessary to find a way to avoid this.
The model must be expanded to different kinds of machines. The method is valid for a set of machines which have several things, such as the main architecture, in common, with small improvements it could be adapted to larger sample of machines.
The narrow programs do probably not take in account the effects of the cache memory. It is interesting to determine a way of characterizing it.
This study was a starting point for the performance estimation. Now it remains another important part, that is to say determining of the processor performance model or the code profile by using the values of the execution time of the narrow programs.

References

[1] Altman N., Weiderman N., *Timing Variation in Dual Loop Benchmarks*, Tech. Rep. CMU/SEI-87-TR-21, Carnegie Mellon University

[2] Axelsson Jakob, *Schedulability-Driven Partitioning of Heterogeneous Real-Time Systems*, Licentiate Thesis no. 517, Linköping University, 1995.

[3] Bonnier S., Kalmelid S., Törne A., *Control System Design - An Overview of Timing Analysis Methods and Tools*, Department of Computer and Information Science, Linköping University, December 1995.

[4] Cybenko G., Kipp L., Pointer L., and Kuck D., Supercomputer Performance Evaluation and the Perfect Benchmarks, University of Illinois Center for Supercomputing R&D Technical Report 965, March 1990.

[5] Ein-Dor P. and Feldmesser J., Attributes of the Performance of Central Processing Units: A Relative Performance Prediction Model, *Comm. of ACM*, 30 (4); p308-317, 1987.

[6] Gong Jie, Gajski Daniel D., and Narayan Sanjiv, Software estimations for executable specifications. *Journal of Computer & Software Engineering*, 2(3):239-258, 1994

[7] Gunst Richard F., *Regression Analysis and its application*, Marcel DEKKER, INC, 1980.

[8] Harmon Marion G., T.P. Baker, and David B. Whalley, A retargetable technique for predicting execution time of code segments. *Real-Time Systems*, 7(2):159-182, September 1994.

[9] Hinnant David, Accurate Unix Benchmarking: Art, Science or Black Magic?, IEEE MICRO, October, 1988, p. 64-75.

[10] *Matlab, User's Guide*, MATH WORKS Inc., August 1992.

[11] *Matlab, Reference Guide*, MATH WORKS Inc., October 1992.

[12] Price Walter, A Benchmark Tutorial, IEEE MICRO, October, 1989, p. 28-43.

[13] SPEC, SPEC Newsletter: Benchmark Results, Vol. 1, Issue 1, Fall 1989.

[14] Saavedra-Barrera R., *CPU Performance Evaluation and Execution Time Prediction Using Narrow Spectrum Benchmarking*, Ph.D. Thesis, UC Berkeley, Tech. Rep. No. UCB/CSD 92/684, Feb. 92.

[15] Saavedra R., *Performance Characterization of Optimizing Compilers*, Computer Science Division, University of California Berkeley, August 1992.

[16] Shaw Alan C., Reasoning about time in higher level language software. IEEE

Transaction on Software Engineering, 15(7):875-889, July 1989.

[17] Weicker R., Dhrystone: A Synthetic Systems Programming Benchmark, CACM, 27, 10, October, 1984, p. 1013-1030.

APPENDIX

APPENDIX 1: Table of measurements

Narrows programs (µs):

Sparcstations[1]:

	add	mul	assign	comp	fcall	loops	pointers
(1)	2.0786	3.9443	2.2657	2.2684	3.3865	13.1178	105.6450
(2)	1.5450	2.2519	1.7973	1.6397	2.7837	9.1496	81.9264
(3)	1.8153	3.3816	1.9518	1.9844	2.7895	11.0795	129.2722
(4)	2.0852	3.8617	2.2458	2.1937	3.4044	12.7180	103.3316
(5)	2.5395	4.8863	2.8061	2.8207	4.2003	15.9985	127.2404
(6)	2.3090	4.4747	2.5568	2.5445	3.8171	14.6271	116.2845
(7)	1.6617	3.1196	1.7741	1.8343	2.7553	10.5390	83.7576
(8)	1.6533	3.1442	1.8008	1.8020	2.6923	10.4523	92.1639
(9)	1.4111	2.6539	1.5158	1.5053	2.2607	8.7982	76.4697
(10)	2.1136	4.5036	2.4186	2.3370	4.2451	18.7262	241.7245

PCs

	add	mul	assign	comp	fcall	loops	pointers
(11)	3.0492	3.1206	3.4157	2.8681	4.9121	14.0633	246.0310
(12)	1.0997	1.2394	1.1002	1.2355	1.5201	6.7565	103.0288
(13)	4.2762	4.6825	4.5513	4.4570	6.5838	25.0554	410.1983

Application programs:

Sparcstations:

1.0e+03 *

	(ms)	(µs)	(ms)	(µs)
(1)	0.0240	0.4745	0.0223	0.3128
(2)	0.0343	0.5533	0.0189	0.4488
(3)	0.0366	0.5909	0.0227	0.4600
(4)	0.0239	0.4833	0.0221	0.3292
(5)	0.0291	0.5868	0.0270	0.3847
(6)	0.0266	0.5325	0.0246	0.3509
(7)	0.0191	0.3823	0.0177	0.2584
(8)	0.0222	0.4173	0.0187	0.2592
(9)	0.0159	0.3166	0.0147	0.1854

1. Description of the machines in appendix

```
(10)    0.0653    1.1204    0.0537    0.8596
```

PCs:

```
  1.0e+03 *
(11)    0.0219    0.5166    0.0211    1.0135
(12)    0.0078    0.2084    0.0112    0.1509
(13)    0.0287    0.7297    0.1255    0.3842
```

APPENDIX 2: Values of measurements after multiplication by the clock frequency

Narrow programs:

Sparcstations:
1.0e+04 *

0.0083	0.0158	0.0091	0.0091	0.0135	0.0525	0.4226
0.0131	0.0191	0.0153	0.0139	0.0237	0.0778	0.6964
0.0127	0.0237	0.0137	0.0139	0.0195	0.0776	0.9049
0.0083	0.0154	0.0090	0.0088	0.0136	0.0509	0.4133
0.0084	0.0161	0.0093	0.0093	0.0139	0.0528	0.4199
0.0083	0.0161	0.0092	0.0092	0.0137	0.0527	0.4186
0.0083	0.0156	0.0089	0.0092	0.0138	0.0527	0.4188
0.0083	0.0157	0.0090	0.0090	0.0135	0.0523	0.4608
0.0085	0.0159	0.0091	0.0090	0.0136	0.0528	0.4588
0.0106	0.0225	0.0121	0.0117	0.0212	0.0936	1.2086

PCs:
1.0e+04 *

0.0201	0.0206	0.0225	0.0189	0.0324	0.0928	1.6238
0.0110	0.0124	0.0110	0.0124	0.0152	0.0676	1.0303
0.0282	0.0309	0.0300	0.0294	0.0435	0.1654	2.7073

Application programs:

Sparcstations:
1.0e+04 *

0.0961	1.8980	0.0891	1.2511
0.2918	4.7028	0.1607	3.8151
0.2560	4.1364	0.1588	3.2200
0.0958	1.9333	0.0885	1.3168
0.0961	1.9363	0.0890	1.2695
0.0959	1.9168	0.0886	1.2634
0.0957	1.9114	0.0886	1.2920
0.1109	2.0867	0.0935	1.2962
0.0956	1.8995	0.0880	1.1124
0.3263	5.6021	0.2686	4.2982

PCs:
1.0e+04 *

0.1446	3.4093	0.1393	6.6892
0.0781	2.0842	0.1121	1.5087
0.1896	4.8159	0.8280	2.5355

APPENDIX 3: Values of measurements (difference with mean)

The mean of each column has been calculated and the following values are the difference between the mean and the normalized measurements. It is a way to compare the values with each other.

Narrow programs:

Sparcstations:

```
  -35.4247  -26.7846  -38.6779  -35.2211  -57.6607 -199.3249   -
0.4377e+04
   12.7519    6.8561   23.4682   13.4202   43.4967   53.6769   -
0.1639e+04
    8.5035   52.1552    7.3237   12.9537    2.1471   51.5301
0.0446e+04
  -35.1623  -30.0896  -39.4724  -38.2084  -56.9427 -215.3190   -
0.4470e+04
  -34.7662  -23.3103  -36.7055  -32.8746  -54.5102 -196.0867   -
0.4404e+04
  -35.4478  -23.4685  -37.2608  -34.3566  -55.7046 -197.4637   -
0.4417e+04
  -35.4863  -28.5782  -40.6011  -34.2431  -55.3567 -197.0877   -
0.4415e+04
  -35.9033  -27.3502  -39.2661  -35.8571  -58.5077 -201.4237   -
0.3995e+04
  -33.9067  -25.3242  -38.3583  -35.6367  -57.4799 -196.1433   -
0.4015e+04
  -12.8883   40.6238   -8.3731   -9.1091   19.1363  212.2743
0.3483e+04
PCs
   82.6769   21.4014   96.1278   63.3358  131.0812  204.1384
0.7635e+04
   -8.6044  -60.6191  -19.2818   -2.4063  -41.1090  -48.3894
0.1700e+04
  163.6573  124.4884  171.0774  168.2032  241.4101  929.6187
1.8470e+04
```

Application programs:

Sparcstations:

1.0e+04 *

```
   -0.0556   -1.0506   -0.0873   -1.1233
    0.1400    1.7541   -0.0157    1.4406
    0.1043    1.1877   -0.0176    0.8456
   -0.0559   -1.0154   -0.0879   -1.0577
   -0.0557   -1.0124   -0.0874   -1.1050
   -0.0559   -1.0319   -0.0878   -1.1111
   -0.0560   -1.0373   -0.0878   -1.0825
   -0.0408   -0.8620   -0.0829   -1.0783
   -0.0561   -1.0491   -0.0884   -1.2621
    0.1745    2.6534    0.0922    1.9237
```

```
-0.0071    0.4606   -0.0370    4.3147
-0.0737   -0.8644   -0.0642   -0.8658
 0.0379    1.8672    0.6517    0.1610
```

APPENDIX 4: Evaluation of execution time of application programs: with the coefficients found with FS, with the values of the PC (10 predictor variables)

Application program 1:

Coef:

```
1 =   -23.2142
2 =   -17.8449
3 =    29.1102
4 =    23.3703
6 =    -1.3733
7 =     0.3945
```

1.0e+03 *

Y	$\hat{Y}$	%error
Sparcstations		
0.9612	0.9594	-0.1805
2.9175	2.9191	0.0553
2.5601	2.5540	-0.2374
0.9577	0.9048	-5.5230
0.9606	0.9793	1.9463
0.9585	0.9441	-1.5102
0.9569	0.9417	-1.5869
1.1095	1.1024	-0.6439
0.9564	1.0363	8.3552
3.2627	3.2611	-0.0464
PCs		
1.0e+04 *		
0.1446	0.7770	437.4011
0.0781	0.4462	471.6500
0.1896	1.1960	530.8325

Application program 2:

Coef:

```
1 =  -465.5525
2 =     3.2261
4 =   221.0902
5 =   508.6058
6 =  -112.2042
7 =     6.2686
```

1.0e+04 *

Y	$\hat{Y}$	%error
Sparcstations		
1.8980	1.8372	-3.2064
4.7028	4.7030	0.0042
4.1364	4.1333	-0.0742
1.9333	1.9158	-0.9026
1.9363	1.9666	1.5625
1.9168	1.9123	-0.2384
1.9114	1.9294	0.9413
2.0867	2.0654	-1.0214
1.8995	1.9585	3.1016
5.6021	5.6021	0.0005
PCs		
1.0e+05 *		
0.3409	1.1136	226.6358
0.2084	0.4261	104.4348
0.4816	1.3981	190.3074

Application program 3:

Coef:

```
1 =   -21.5392
2 =     2.2368
5 =    13.2503
6 =    -0.3110
7 =     0.1602
```

1.0e+03 *

Y	$\hat{Y}$	%error
Sparcstations		
0.8909	0.8706	-2.2787
1.6067	1.6084	0.1090
1.5878	1.5881	0.0197
0.8849	0.8572	-3.1227
0.8896	0.9006	1.2438
0.8855	0.8975	1.3561
0.8856	0.8917	0.6938
0.9349	0.9303	-0.4910
0.8797	0.9006	2.3860
2.6857	2.6847	-0.0389
PCs		
1.0e+03 *		
1.3934	2.7342	96.2217
1.1211	1.3631	21.5786
8.2801	4.1923	-49.3686

Application program 4:

Coef:

```
1 =  -331.3304
2 =  -271.0261
3 =   315.1788
4 =   379.3539
7 =     4.6778
```

1.0e+04 *

Y	$\hat{Y}$	%error
Sparcstations		
1.2511	1.2443	-0.5478
3.8151	3.8210	0.1550
3.2200	3.1829	-1.1517
1.3168	1.1435	-13.1587
1.2695	1.2669	-0.2036
1.2634	1.2141	-3.8994
1.2920	1.2537	-2.9653
1.2962	1.4117	8.9087
1.1124	1.3182	18.5036
4.2982	4.2933	-0.1145
PCs		
1.0e+05 *		
0.6689	0.9632	43.9926
0.1509	0.5972	295.8026
0.2536	1.5564	513.8300

APPENDIX 5: Evaluation of execution time of benchmarks: with the coefficients found with FS, with the values of the Sparcstations and the PC (13 predictor variables)

Application program 1:

Coef:

```
1 =  -260.7173
2 =    18.8134
4 =   183.4787
5 =    75.4212
6 =   -16.7893
7 =     0.4003
```

1.0e+03 *

	Y	$\hat{Y}$	%error
Sparcstations			
	0.9612	1.0375	7.9469
	2.9175	2.5123	-13.8877
	2.5601	2.1388	-16.4573
	0.9577	0.6445	-32.7074
	0.9606	1.5343	59.7106
	0.9585	1.3650	42.4040
	0.9569	1.3203	37.9821
	1.1095	1.1594	4.4990
	0.9564	0.6983	-26.9875
	3.2627	3.2494	-0.4065
PCs			
	1.4458	1.5060	4.1656
	0.7805	0.5763	-26.1647
	1.8960	2.0517	8.2108

Application program 2:

Coef: 1.0e+03 *

```
1 =    -2.5524
3 =     2.3663
4 =     0.2156
5 =    -0.3596
6 =     0.0944
```

1.0e+04 *

	Y	$\hat{Y}$	%error
Sparcstations			
	1.8980	2.2618	19.1635
	4.7028	4.4703	-4.9447
	4.1364	3.1911	-22.8525
	1.9333	1.7656	-8.6752
	1.9363	2.5283	30.5727
	1.9168	2.5689	34.0169
	1.9114	1.7818	-6.7825
	2.0867	2.2417	7.4263
	1.8995	1.9645	3.4212
	5.6021	5.3676	-4.1849
PCs			
	3.4093	3.1631	-7.2210
	2.0842	1.5432	-25.9567
	4.8159	5.3717	11.5409

Application program 3:

Coef:

```
1 =    31.6053
2 =   -20.8965
3 =   -37.1767
4 =    -7.0684
6 =    10.3915
7 =     0.0180
```

1.0e+03 *

	Y	$\hat{Y}$	%error
Sparcstations			
	0.8909	0.8491	-4.6903
	1.6067	1.6931	5.3756
	1.5878	1.2310	-22.4687
	0.8849	0.8093	-8.5440
	0.8896	0.7406	-16.7478
	0.8855	0.7389	-16.5539
	0.8856	0.9718	9.7415
	0.9349	0.8573	-8.3061
	0.8797	0.8973	2.0002
	2.6857	3.2606	21.4043
PCs			
	1.3934	2.2759	63.3348
	1.1211	3.1289	179.0805
	8.2801	6.8879	-16.8145

Application program 4:

Coef: 1.0e+03 *

```
2 =      0.4139
3 =     -3.3292
4 =      1.3883
5 =      2.2063
6 =     -0.3924
7 =      0.0070
```

1.0e+04 *

	Y	$\hat{Y}$	%error
Sparcstations			
	1.2511	1.2256	-2.0418
	3.8151	2.9979	-21.4192
	3.2200	2.6112	-18.9072
	1.3168	1.6595	26.0257
	1.2695	1.5876	25.0575
	1.2634	1.3418	6.2059
	1.2920	2.3213	79.6651
	1.2962	1.4742	13.7303
	1.1124	1.2919	16.1367
	4.2982	3.8740	-9.8684
PCs			
	6.6892	6.2836	-6.0631
	1.5087	-0.0723	-104.7926
	2.5355	3.6538	44.1035

APPENDIX 6: Regression with non-negative least squares: 10 predictor variables: Sparcstations

Application program 1:

Coef:

```
5 =     1.3391
7 =     0.2436
```

1.0e+03 *

	Y	$\hat{Y}$	%error
Sparcstations			
	0.9612	1.2106	25.9554
	2.9175	2.0129	-31.0053
	2.5601	2.4655	-3.6975
	0.9577	1.1890	24.1545
	0.9606	1.2083	25.7794
	0.9585	1.2036	25.5655
	0.9569	1.2045	25.8750
	1.1095	1.3026	17.4066
	0.9564	1.2991	35.8385
	3.2627	3.2279	-1.0642
PCs			
1.0e+03 *			
	1.4458	4.3891	203.5748
	0.7805	2.7129	247.5679
	1.8960	7.1758	278.4718

Application program 2:

Coef:

```
5 =    45.3844
7 =     3.7056
```

1.0e+04 *

	Y	$\hat{Y}$	%error
Sparcstations			
	1.8980	2.1807	14.8910
	4.7028	3.6543	-22.2947
	4.1364	4.2394	2.4903
	1.9333	2.1497	11.1921
	1.9363	2.1850	12.8434
	1.9168	2.1749	13.4639
	1.9114	2.1771	13.8993
	2.0867	2.3185	11.1094
	1.8995	2.3158	21.9129
	5.6021	5.4420	-2.8576
PCs			
1.0e+05 *			
	0.3409	0.7489	119.6497
	0.2084	0.4508	116.2764
	0.4816	1.2004	149.2619

Application program 3:

Coef:

```
5 =      0.2263
7 =      0.2034
```

1.0e+03 *

	Y	$\hat{Y}$	%error
Sparcstations			
	0.8909	0.8901	-0.0916
	1.6067	1.4698	-8.5169
	1.5878	1.8846	18.6959
	0.8849	0.8714	-1.5173
	0.8896	0.8854	-0.4749
	0.8855	0.8825	-0.3421
	0.8856	0.8829	-0.2976
	0.9349	0.9677	3.5028
	0.8797	0.9638	9.5712
	2.6857	2.5062	-6.6854
PCs			
1.0e+03 *			
	1.3934	3.3759	142.2775
	1.1211	2.1298	89.9695
	8.2801	5.6045	-32.3134

Application program 4:

Coef:

```
5 =    16.9843
7 =     3.1584
```

1.0e+04 *

	Y	$\hat{Y}$	%error
Sparcstations			
	1.2511	1.5647	25.0661
	3.8151	2.6013	-31.8151
	3.2200	3.1897	-0.9421
	1.3168	1.5367	16.7047
	1.2695	1.5616	23.0096
	1.2634	1.5556	23.1270
	1.2920	1.5567	20.4869
	1.2962	1.6841	29.9242
	1.1124	1.6795	50.9815
	4.2982	4.1778	-2.8011
PCs			
1.0e+04 *			
	6.6892	5.6793	-15.0978
	1.5087	3.5122	132.7963
	2.5355	9.2888	266.3478

APPENDIX 7: Regression with non-negative least squares 13 predictors variables: Sparcstations + PCs

Application program 1:

Coef:
```
2 =     8.4034
```

1.0e+03 *

	Y	$\hat{Y}$	%error
Sparcstations			
	0.9612	1.3258	37.9415
	2.9175	1.6085	-44.8667
	2.5601	1.9892	-22.3008
	0.9577	1.2981	35.5368
	0.9606	1.3550	41.0531
	0.9585	1.3537	41.2232
	0.9569	1.3108	36.9824
	1.1095	1.3211	19.0697
	0.9564	1.3381	39.9138
	3.2627	1.8923	-42.0017
PCs			
	1.4458	1.7308	19.7100
	0.7805	1.0415	33.4337
	1.8960	2.5970	36.9756

Application program 2:

Coef:
```
2 =   108.9284
6 =    13.6117
```

1.0e+04 *

	Y	$\hat{Y}$	%error
Sparcstations			
	1.8980	2.4328	28.1754
	4.7028	3.1437	-33.1541
	4.1364	3.6342	-12.1420
	1.9333	2.3751	22.8513
	1.9363	2.4751	27.8230
	1.9168	2.4715	28.9364
	1.9114	2.4163	26.4161
	2.0867	2.4238	16.1542
	1.8995	2.4531	29.1399
	5.6021	3.7274	-33.4645
PCs			
	3.4093	3.5069	2.8626
	2.0842	2.2697	8.8994
	4.8159	5.6173	16.6404

Application program 3:

Coef:

```
7 =     0.2275
```

1.0e+03 *

	Y	$\hat{Y}$	%error
Sparcstations			
	0.8909	0.9612	7.8906
	1.6067	1.5840	-1.4116
	1.5878	2.0583	29.6380
	0.8849	0.9402	6.2492
	0.8896	0.9551	7.3664
	0.8855	0.9522	7.5311
	0.8856	0.9526	7.5709
	0.9349	1.0482	12.1146
	0.8797	1.0437	18.6430
	2.6857	2.7492	2.3637
PCs			
	1.3934	3.6936	165.0768
	1.1211	2.3435	109.0319
	8.2801	6.1582	-25.6270

Application program 4:

Coef:

```
5 =   121.7897
```

1.0e+04 *

	Y	$\hat{Y}$	%error
Sparcstations			
	1.2511	1.6498	31.8615
	3.8151	2.8818	-24.4639
	3.2200	2.3782	-26.1449
	1.3168	1.6585	25.9524
	1.2695	1.6881	32.9761
	1.2634	1.6736	32.4682
	1.2920	1.6778	29.8634
	1.2962	1.6394	26.4809
	1.1124	1.6520	48.5058
	4.2982	2.5851	-39.8570
PCs			
	6.6892	3.9484	-40.9726
	1.5087	1.8513	22.7099
	2.5355	5.2921	108.7213

APPENDIX 8: Listing of the programs: dual-loop and narrow programs

```
-------------------------------------------------------------------
-- MODULE DUAL LOOP                                              --
-- This module contains the loops used to do time measurements--
-- The narrow programs are in the package operations, the        --
-- packages BuildFlight2, bench1, whetstone, d1_pack_1           --
-- contain the 4 application programs                            --
--                                                               --
-- Patrick FACCHINI <patfa@ida.liu.se>                           --
-- Spring 1996                                                   --
-------------------------------------------------------------------

----------------------------------
-- Included packages--
----------------------------------
with Ada.Real_Time; use Ada.Real_Time;
-- used for the type Time

with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Text_IO.Fixed_IO;
with operations; use operations;
with BuildFlight2;
with bench1; with whetstone; with d1_pack_1;

----------------------------------
-- Main procedure--
----------------------------------
procedure dualoop is

------------------------------------------
-- Variables and types declaration--
------------------------------------------

datafile: File_Type;          -- File
machine_id: String(1..20);    -- Name of file
length: Natural;              -- Lenght of the name
nb1: Integer:=3;              -- Values used to
nb2: Integer:=31;             -- avoid optimization
d4: Integer:=71;              -- Values of dummy
d5: Integer:=69;              -- code
choice: Integer:=4;           -- Choice in the main menu
nprg: Integer:=1;             -- Number of executed program

nloop : Integer:=10000;       -- Number of loops
Time_first, Time_second,
            Time_last : Time;  -- Times

----------------------------------
-- Procedures and functions--
----------------------------------
----  Function ReadTime  ----
function ReadTime return Time is
begin
  return Clock;
```

```
end ReadTime;

---- Procedure PrintTime ----
-- This procedure calculates the execution time and print it
-- in a file and on the screen
procedure PrintTime (Time_First,Time_second, Time_Last: in Time; nb:
in Integer) is

a: Duration;
TS: Time_Span;
dataf: File_Type;

package Myduration_Text_IO is
 new Ada.Text_IO.Fixed_IO (Duration);
use Myduration_Text_IO;

begin
     Open(dataf,Append_File,machine_id(1..length));
     TS:=(Time_second - Time_last) + (Time_second - Time_first);
     Put("Difference: "); Put("ms");
     a:=To_Duration(TS);
     Myduration_Text_IO.Put(a*1000);
     Myduration_Text_IO.Put(dataf,a*1000);
     New_line;
     close(dataf);
end PrintTime;

----  Procedure Measure ----
-- That is the dual loop
-- The different programs are called through a pointer
procedure Measure(nfct: IN Integer) is

type Proc is access procedure(a,b: IN OUT Integer);
ptr_fct: Proc;

begin
  case nfct is
    when 1 => ptr_fct:= addition'Access;
    when 2 => ptr_fct:= multip'Access;
    when 3 => ptr_fct:= transfer'Access;
    when 4 => ptr_fct:= compar'Access;
    when 5 => ptr_fct:= calladd'Access;
    when 6 => ptr_fct:= loopadd'Access;
    when 7 => ptr_fct:= pointer'Access;
    when 8 => ptr_fct:= BuildFlight2.BuildFlightList'Access;
    when 9 => ptr_fct:= bench1.ap_ai_a_star'Access;
    when 10 => ptr_fct:=whetstone.cl_wh_whet_04'Access;
    when 11 => ptr_fct:=d1_pack_1.proc_0'Access;
    when others => Put ("Error");
  end case;

  if nfct<12 then
    Time_first:=ReadTime;
    For i in 1..nloop loop
ptr_fct(nb1,nb2);
ptr_fct(nb1,nb2);
   --dummy code
nb1:=nb1*d4;
```

```
nb1:=nb1/d5;
    end loop;

    Time_second:=ReadTime;

    For i in 1..nloop loop
    --dummy code
nb1:=nb1*d4;
nb1:=nb1/d5;

    end loop;

    Time_last:=ReadTime;

    PrintTime(Time_first,Time_second,Time_last,nloop);
    Put(Nb1);
    New_Line;
  end if; -- nfct > number of function
end Measure;


----------------------------------
-- Body of main procedure--
----------------------------------

begin
---- Creation of a file for data -----
Put_Line("Enter Machine id:");
Get_Line(machine_id,length);
Create(datafile,Out_File,machine_id(1..length));
Put_Line(datafile, "*** Measurements ***");
close(datafile);

while choice/=0 loop
choice:=4;
----- Menu -----
New_Line;
Put_Line("***** Measurement of time ******");
Put_Line("     (1) Measure all the narrow programs");
Put_Line("     (2) Measure one program");
Put_Line("     (3) Change the number of loops");
Put_Line("     (0) Quit");
New_Line;
while (choice<0 OR choice>3) loop
Put("Enter your choice: ");
Get(choice);
end loop;

----- Choice 1: all the small narrow programs -----
if choice=1 then
  for j in 1..7 loop
     Put(j);Put(" : ");
     for I in 1..5 loop
       Measure(j);
       nb1:=3;
       nb2:=31;
     end loop;
  end loop;
```

```
----- Choice 1: one program -----
elsif choice=2 then
  nprg:=12;
  while nprg>11 loop
    Put("Enter the number of the program (1 .. 11): ");
    Get(nprg);
  end loop;
  for I in 1..5 loop
    Measure(nprg);
    nb1:=3;
    nb2:=31;
  end loop;

----- Choice 3: change the number of loops -----
elsif choice=3 then
    Put("Enter the number of loops: ");
    Get(nloop);

end if; -- if choice
end loop;

end;
```

```
----------------------------------------------------------
-- Package Operations.                                  --
-- This package contains the narrow programs            --
-- Their execution time will be used for the            --
-- linear regression                                    --
----------------------------------------------------------

----------------------------------
--  Included packages          --
----------------------------------
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
with Ada.Text_IO; use Ada.Text_IO;
with Unchecked_Deallocation;

----------------------------------
--   Body of package           --
----------------------------------
package body operations is

-----  Program 1  -----
procedure addition(n1, n2: IN OUT Integer) is
begin
     n1:=n1+n2;
end;

-----  Program 2  -----
procedure multip(n1, n2: IN OUT Integer) is
begin
     n1:=n1*n2;
end;

-----  Program 3  -----
procedure transfer(n1, n2: IN OUT Integer) is
tmp:Integer;
begin
     tmp:=n1;
     n1:=n2;
     n2:=tmp;
end;

-----  Program 4  -----
procedure compar(n1, n2: IN OUT Integer) is
begin
     if n1>n2 then n1:=n1-1;
     else n1:=n1+1;
     end if;
end;

-----  Program 5  -----
procedure calladd(n1, n2: IN OUT Integer) is
begin
     addition(n1,n2);
end;
```

```
-----  Program 6  -----
procedure loopadd(n1,n2: IN OUT Integer) is
n3: Integer:=1;
begin
    for i in 1..(n2-28) loop
      for j in 1..5 loop
          n1 := n3+n1;
      end loop;
      n3:=n3+2;
    end loop;
end loopadd;

-----  Program 7  -----
procedure pointer(n1,n2: IN OUT Integer) is

     type Cellule;
     type Intlist is Access Cellule;
     type Cellule is
       record
          val1: Integer;
          val2: Integer;
          next: Intlist;
      end record;
     procedure Free is new Unchecked_Deallocation
          (Object => Cellule,
           name => Intlist);
     list,ptr,head: Intlist;

begin
     head:=new Cellule'(val1=>3,
                        val2=>3+1,
                        next=>null);
     list:=head;
     for i in 2..5 loop
          ptr:=new Cellule'(val1=>3,
                            val2=>6*i+1,
                            next=>null);
          list.next:=ptr;
          list:=ptr;
     end loop;
     list:=head;
     while list /= null loop
       if list.val1=n1 and list.val2=n2 then
          n1:=n2;
       end if;
       ptr:=list;
       list:=list.next;
       Free(ptr);
     end loop;

     Free(list); Free(head);
end pointer;


end operations;
```

APPENDIX 9: Computers characteristics

PCs:

Table 10:

Architecture identification	Implementation	Characteristics
PC11	486DX2 66	Clock frequency: 66 MHz
PC12	Pentium 100	Clock frequency: 100 MHz Mem. access time: 70 ns (32MB) Cache internal memory: 8+8 KB Cache external memory: 256 KB Pipeline system Coprocessor
PC13	IBM 486BL2	Clock frequency: 66 MHz (33*2) Mem. access time: 70 ns (12MB) Cache internal memory: 8+8 KB Cache external memory: 256 KB Coprocessor

Sparcstations:

The characteristics of the ten Sparcstations are in the table on the next page.

Table 1: Hardware architectures

:ation	Memory access time	Cache internal mem: size	Cache second-level	Addition exec. time (clock cycles)	Multiplic. exec. time	Clock frequency (MHz)	coprocess. (y/n)
50 RC	80	20 Kb Ins 16 Kb data	0 Mb	1	?	40	y
10 RC	60	16 Kb Ins 8 Kb data	0 Mb	1	?	85	y
10 RC	60	16 Kb Ins 8 Kb data	0 Mb	1	?	70	y
50 RC	80	20 Kb Ins 16 Kb data	1 Mb	1	?	40	y
50 RC	80	20 Kb Ins 16 Kb data	0 Mb	1	?	33	y
50 RC	80	20 Kb Ins 16 Kb data	0 Mb	1	?	36	y
50 RC	70	20 Kb Ins 16 Kb data	0 Mb	1	?	50	y
50 RC	70	20 Kb Ins 16 Kb data	2 Mb	1	?	50	y
50 RC	70	20 Kb Ins 16 Kb data	1 Mb	1	?	60	y
10 RC	60	16 Kb Ins 8 Kb data	0 Mb	1	?	50	y

:quency for the memory bus (MBUS), respectively 40 and 50 MHz for any clock frequ