Time Response Pattern Analysis of C Statements for Performance Evaluation

Iman Kattan
PhD Student
Dept. Computer Engineering
Faculty of Electrical and Electronic Engineering
University of Aleppo

Amer Bouchi, PhD
Assistant Professor
Dept. Computer Engineering
Faculty of Electrical and Electronic Engineering
University of Aleppo

Mouhamad Ayman Naal, PhD
Associate Professor
Dept. Computer Engineering
Faculty of Electrical and Electronic Engineering
University of Aleppo

ABSTRACT

The study and the evaluation of application performance are closely related to the study and the evaluation of time response patterns of the used programming language, including instructions, components, and different programming structures, as well as data types, according to specific criteria.

In this paper, the time response of instructions, programming structures, and basic data types of C language is studied, and the results are analyzed to determine the programming patterns having the best performance for the approved infrastructure.

This paper studies three main points:

- The time response patterns for instructions in C language.

- The best time response programming structures.

- The correlation between the internal architecture of the processor and the resulting time response pattern.

This paper is a step in a research project having as objective to find general features of programming model aiming at enhancing performance of applications. These features are extracted from the analysis of time response patterns of basic instructions and programming structures and data types of the C language. This analysis helps in defining the best time response patterns for the C language. In addition, the correlation between programs and the platform (processor architecture and operating system) is investigated.

General Terms

Application Performance, Programming Patterns

Keywords

Application Performance, Programming Patterns, Data Types, Time Response, and Out-Of-Order Execution

1. INTRODUCTION

Applications deal directly with the infrastructure of the computer system (operating system and hardware) or with a middleware layer. The performance levels of applications vary according to several factors, the most important factors are [1]:

1- Instructions, structures and programming components used to build the application.

2- Data types and their storage location (local or global variables).

3- Data structures and the way to deal with them.

4- The type of application and its purpose (local, distributed, web, mobile applications, or other types of applications).

5- The infrastructure (processor and operating system) on which the application is run.

Evaluation of application performance starts from analyzing basic instructions and data structures of the programming language. Then it continues by studying programming components and methods of dealing with complex data structures and files. It ends with evaluating the performance of programs and applications that use complex data structures and different kinds of database management systems.

In this paper, the proposed programming model is enhanced by a deep study of the basic instructions of the C language. The aim is at defining time-response patterns by studying and analyzing basic instructions and structures of C programming language. Assuming that programs are implemented on local computer (no remote access and no network connection are needed). Data are stored in local or global variables in the program (no need of database).

2. CLASSIFICATION OF PROGRAMMING LANGUAGES

High level compiled programming languages can be classified into the following categories [2]:

1. Procedural programming: it is based on the concept of executive units (procedures or functions) and variables scope.

2. Functional programming: it is based on executive units in the form of mathematical Functions.

3. Logical programming: the program is a set of sentences in logical form, expressing facts and rules about some problem domain.

4. Object-Oriented programming: defines the program in the form of objects that interact with each other to perform the required work.

In this paper, the focus is on procedural programming. We choose the C language due to its high performance compared to the rest of the programming languages, and its ability to directly control the system infrastructure. It is also a language that remains important until now [3, 4].

3. PERFORMANCE EVALUATION OF PROGRAMMING LANGUAGES

There are many studies about performance evaluation of programming languages. These studies have focused on evaluating performance according to two criteria: execution time and size of consumed memory.

Research [4] uses benchmarks to compare 27 programming languages in terms of energy consumption, execution time, and memory consumed. The research compared translated, interpreted and virtual machine languages. It shows that the C language was the best of all the programming languages tested, and that the fastest execution language is the least energy consumption in terms of C, C ++, Java languages. This result cannot be generalized to all programming languages.

Research [5] classifies procedural and declarative programming languages. It studies the memory usage and shows that SML and Python have a greater ability to manage memory for very large numbers, unlike C/C ++, which require the programmer to do memory management. However, this research does not study the performance of programming languages, and does not determine programming patterns for high performance applications.

Research [6] presents a performance comparison between a set of programming languages by executing benchmarks. It shows that languages like Ruby, Python and PHP are 100 times slower than C, Fortran, and Java. However, this research does not provide any criteria or model for high performance applications.

Research [7] uses Benchmarks to evaluate and compare performance and energy consumed for object-oriented (OO) and procedural languages used in embedded systems. It concludes that the use of OOP languages for such systems leads to a significant increase in execution time and energy consumption. However, this research also does not provide any criteria or model for building high-performance applications.

The above mentioned researches showed the importance of the C language in terms of speed, memory usage, and energy consumption. Therefore, C language is adopted in this paper. On the other hand, these researches does not aim at building a programming model that links programs to processor architecture in order to improve performance, but rather they measure performance by executing benchmarks and comparing results.

The new features of modern multi-core processors, such as: Out-Of-Order and Speculative execution [8], are more efficient than all traditional techniques used in parallel programming. It is important now to switch from the use of old parallel techniques and optimization methods to exploit these important features. To achieve this objective, a new programming model based on the intrinsic architectural parallelism of modern processors is under development [9]. The proposed model is based on two concepts: correlation of instructions: the more the instructions are independent, the more the execution of the program is efficient, and the modularity of program: the more the program is modular, the more its execution is efficient. To complete and enhance this model, a deep study and analysis of relation between the instructions execution and processor architecture.

4. BENCHMARK PROGRAMS

The basic measures of system performance are response time and execution rate. Two main classes of benchmarks exist: synthetic and real programs benchmarks.

Synthetic benchmarks are written to compare basic concepts, such as a single operation, or narrow aspects of a larger system. The simplest type of synthetic benchmark programs can perform only basic operations, such as addition and Multiplication. Whetstone and Dhrystone are the most popular synthetic benchmarks [10].

In this paper, the Dhrystone benchmark model is considered in order to evaluate basic instructions time response.

5. RESEARCH OBJECTIVE AND METHODOLOGY

The main objective of this research is to suggest performance criteria for a new programming model that improves the performance of applications written according to it. This is achieved through studying and analyzing time response for instructions and programming structures.

Instructions and programming structures for the C language are classified to the following categories [11]:

- Basic data types.
- Operators.
- Simple and composed statements.
- Storage class specifiers.
- Processor directives.
- Variables, Macros, Functions.

This research focuses on the first three points (basic data types, operators, and simple and composed statements), and it follows the next steps:

1. Obtaining the response times for all instructions shown in table 1.

Table 1: Categories and types of studied instructions

Mathematical and logical instructions, Pointers and Arrays				
C=A+B	A+=B	C=A%B	A%=B	C=A|B
C=A-B	A-=B	C=A^B	A^=B	C=~A
C=A*B	A*=B	A++	++A	C=A<<1
C=A/B	A/=B	A--	--A	C=A>>1
C=A&B	A&=B	A=B	&A	sizeof(A)
D[i]=i : i=50				
Conditional instructions				
if (r):r=A>B	if (A>0)	if (A<0)	if (A==0)	
if (A<=0)	if (A!=0)	if (A>=0)	if (!(A>=0))	
if ((A>=0) || (B>=0))		if ((A>=0)&&(B>=0))		

The study considers all possible forms of the instructions, and mainly two different basic forms of arithmetic instructions: correlated and non-correlated forms. For example, consider the addition instruction, the form (C = A + B) is called non-correlated form because repeating it gives independent instructions. On the other hand, the form (A + = B) is called correlated form, because repeating it gives dependent instructions. Each instruction is repeated /200/ times to be able to calculate the execution time.

2. All basic data types defined in the C language are studied, namely: Integer - Short - Sign - Unsigned - Long - Float-Double – Char.

3. The study is repeated for local and global variables.

4. Comparing time response patterns, and discussing results.
5. This study was done on two different computers (a desktop and a laptop) with the following specifications:

Table 2: Specifications of computers used in the test

	PC1 (Desktop)	L_I (Laptop)
CPU	Intel Core 2 Quad Q9400	Intel Core 2 Duo T5870
Freq.	2.66 GHz	2.00 GHz
RAM	6.00 GB	4.00 GB
System	64 bits	32 bits
Cores	4 Cores	2 Cores

6. ANALYSIS OF TIME RESPONSE PATTERNS FOR INSTRUCTIONS AND BASIC DATA TYPES

6.1 Time response for local and global integer variables

Time response for correlated and non-correlated arithmetic instructions on local and global integer variables executed on Laptop, is shown in Figure 1.

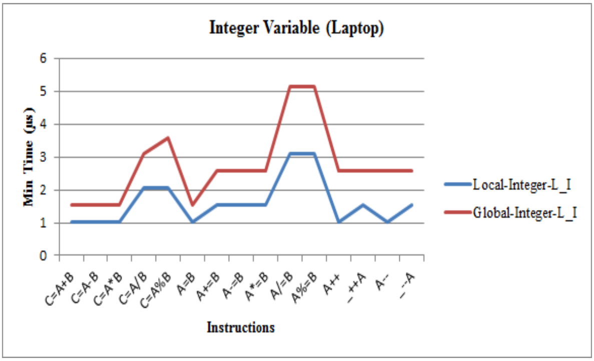

Figure 1: Time response patterns for arithmetic instructions on Integer data type (Laptop)

On Laptop, the execution time of correlated instructions is greater than the execution time of non-correlated instructions, and local variables have better response time than global variables for all instructions.

Figure 2 shows the time response for conditional and logical instructions on local and global integer variables executed on Laptop.

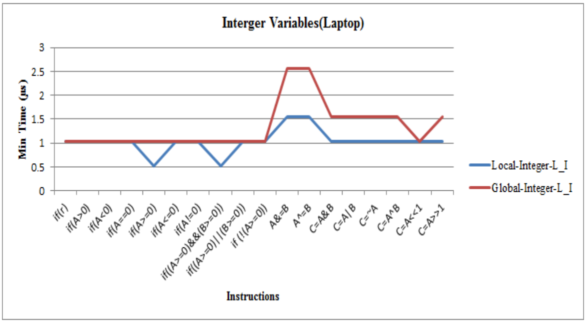

Figure 2: Time response patterns for conditional and logical instructions on Integer data type (Laptop)

The execution time of correlated logical instructions is greater than non-correlated logical instructions. There is no clear difference between local and global variables in conditional instructions. But for logical instructions local variables have better response time.

Figure 3 shows the time response for arithmetic instructions on local and global integer variables executed on desktop.

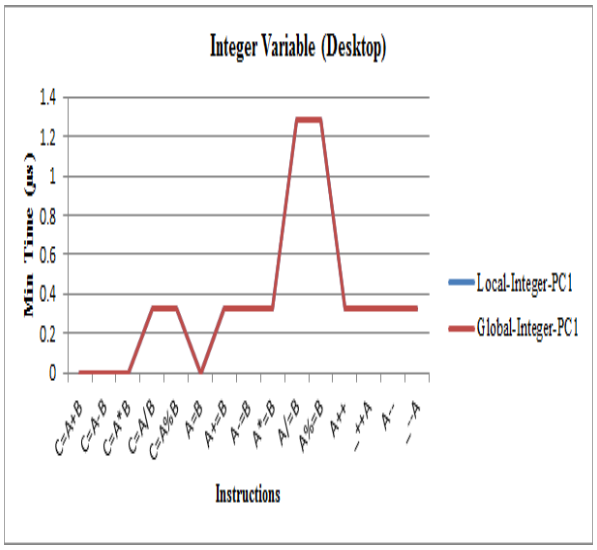

Figure 3: Time response patterns for arithmetic instructions on Integer data type (Desktop)

On Desktop, the execution time of correlated instructions is greater than non-correlated instructions, and division and modulo instructions take more time than other arithmetic instructions.

Figure 4 shows the time response for conditional and logical instructions on local and global integer variables executed on Desktop.

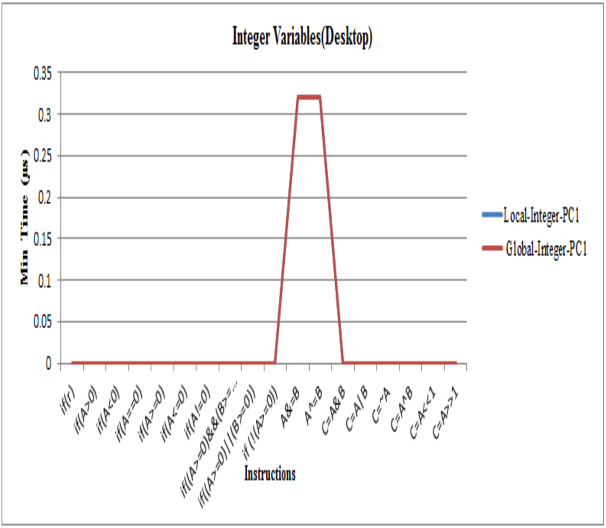

Figure 4: Time response patterns for conditional and logical instructions on Integer data type (Desktop)

The execution time of correlated form of logical instructions is greater than the execution time of non-correlated form. Conditional instructions don't take time as other instructions. In addition, no difference between local and global variables in all instructions on Desktop.

Figure 5 shows the time response for basic instructions on local and global integer variables.

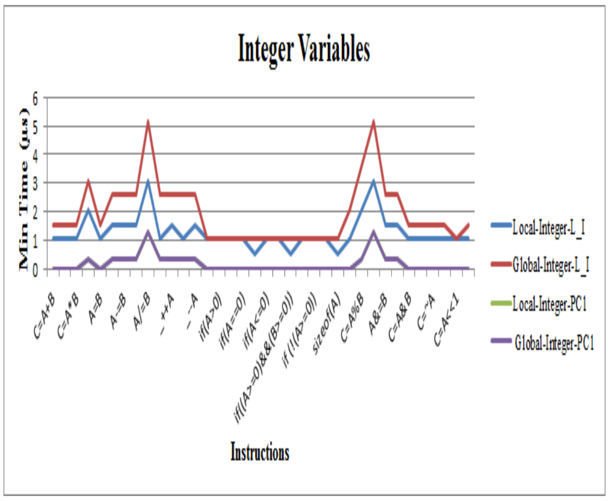

Figure 5: Time response patterns for basic instructions on Integer data type

The integer data type response pattern is the same as the short, long, signed, unsigned, char data types, and for the two types of computers used (Laptop & Desktop).

On Laptop, local variables have better response time than global variables for all except conditional instructions which have equal time response for local and global variables, on Desktop local and global variables have the same response time.

6.2 Time response for local and global float variables

Time response for correlated and non-correlated arithmetic instructions on local and global float variables executed on Laptop, is shown in Figure 6

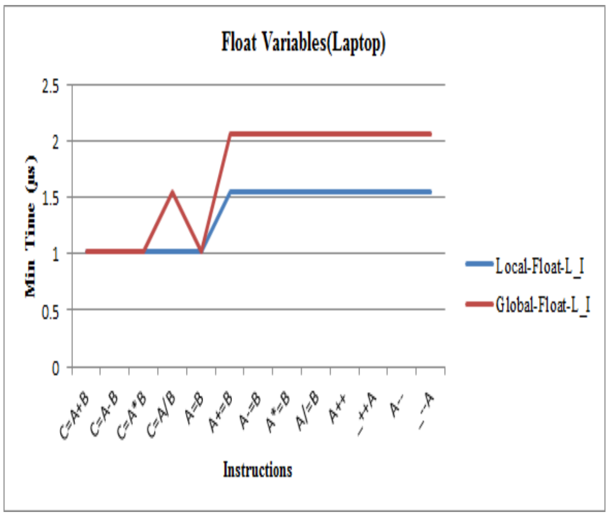

Figure 6: Time response patterns for arithmetic instructions on Float data type (Laptop)

The execution time of correlated instructions is greater than the execution time of non-correlated instructions, and local variables have better response time than global variables.

Figure 7 shows the time response for conditional instructions on local and global float variables executed on Laptop.

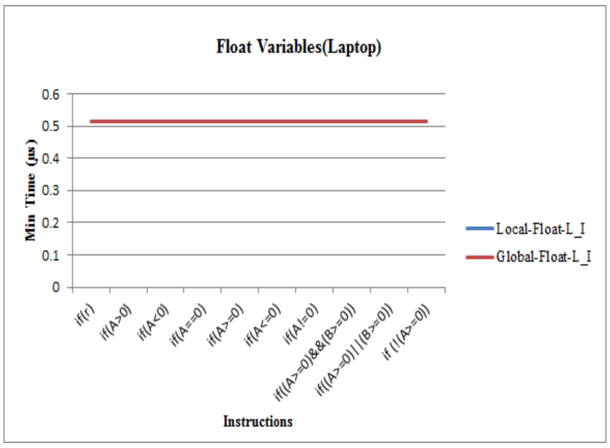

Figure 7: Time response patterns for conditional instructions on Float data type (Laptop)

No difference between local and global variables in all forms of conditional instructions.

Figure 8 shows the time response for arithmetic instructions on local and global float variables executed on Desktop.

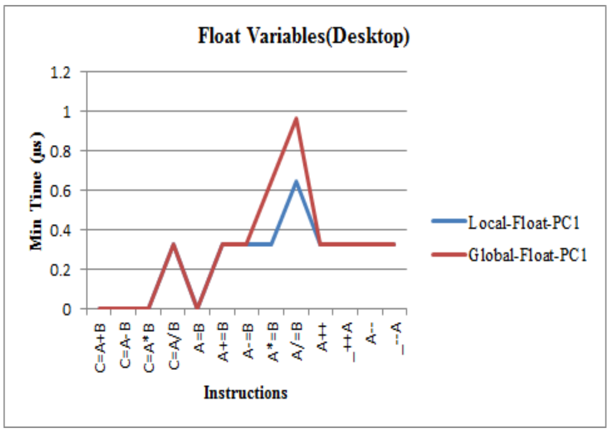

Figure 8: Time response patterns for arithmetic instructions on Float data type (Desktop)

On Desktop, the execution time of correlated instructions is greater than non-correlated instructions, and the response time of correlated multiplication and division arithmetic instructions on global variables is greater than the same instructions on local variables.

Figure 9 shows the time response for conditional instructions on local and global float variables executed on Desktop.

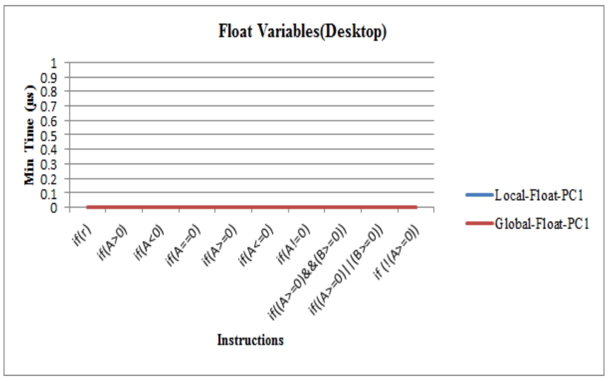

Figure 9: Time response patterns for conditional instructions on Float data type (Desktop)

The conditional instructions don't take time on Desktop.

Figure 10 shows the time response for basic instructions on float data type.

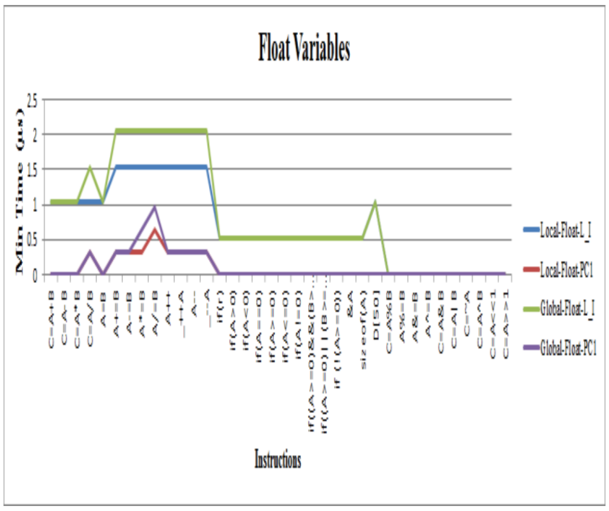

Figure 10: Time response patterns for basic instructions on Float data type

Time response patterns for local and global variables are the same for Desktop. For Laptop, local variables are better than global variables for non-correlated division and correlated arithmetic instructions.

6.3 Time response for local and global double variables

Time response for correlated and non-correlated arithmetic instructions on local and global double variables executed on Laptop, is shown in Figure 11.

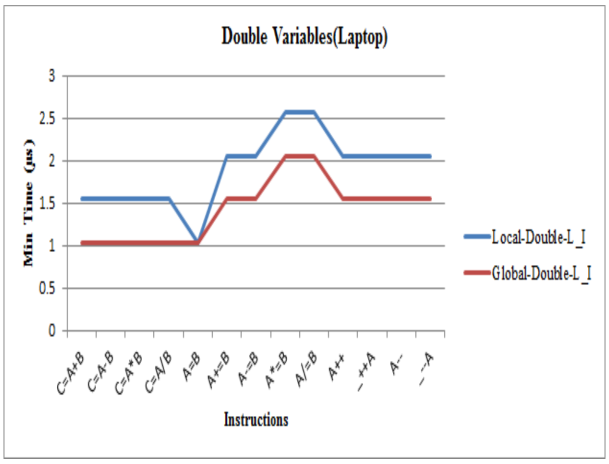

Figure 11: Time response patterns for arithmetic instructions on Double data type (Laptop)

The execution time of the basic non-correlated arithmetic instructions is the same. This is due to the use of FPU (Floating Point Unit). The difference is between correlated and non-correlated instructions, and the execution time of correlated division and multiplication instructions is greater than other correlated instructions.

Figure 12 shows the time response for conditional instructions on local and global double variables executed on Laptop.

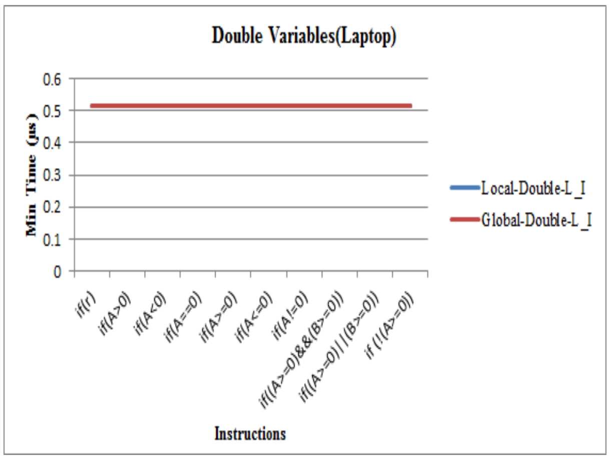

Figure 12: Time response patterns for conditional instructions on Double data type (Laptop)

No difference between local and global variables in all forms of conditional instructions.

Figure 13 shows the time response for arithmetic instructions on local and global double variables executed on Desktop.

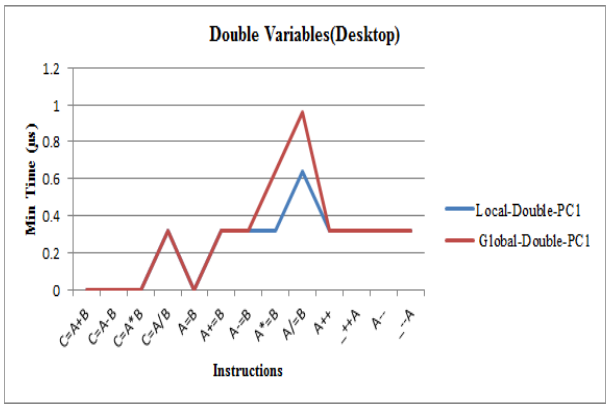

Figure 13: Time response patterns for arithmetic instructions on Double data type (Desktop)

On Desktop, the time response of double data type is the same as float data type, and the correlated multiplication and division instructions on local variables have better response time than global variables.

Figure 14 shows the time response for conditional instructions on local and global double variables executed on Desktop.

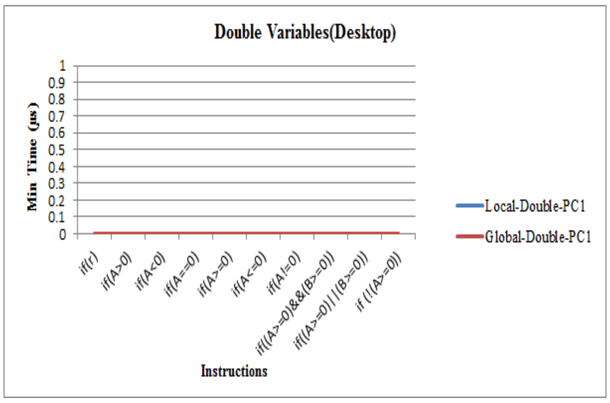

Figure 14: Time response patterns for conditional instructions on Double data type (Desktop)

The conditional instructions don't take time on Desktop.

Figure 15 shows the time response for basic instructions on double data type.

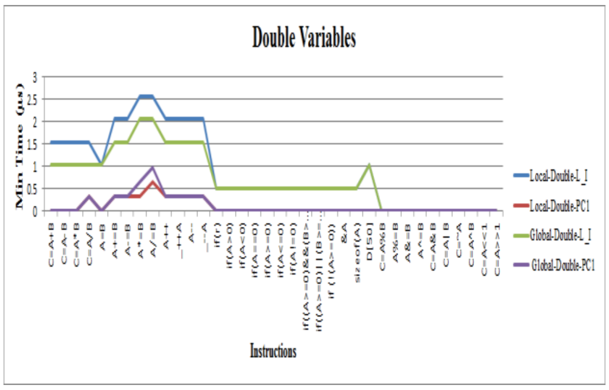

Figure 15: Time response patterns for basic instructions on Double data type

The response time of the correlated multiplication and division instructions differs from the execution time of the correlated form of the other arithmetic instructions. Time response for non-correlated and correlated division instructions is greater than non-correlated and correlated other basic instructions respectively.

For Laptop, global variables have better response time than local variables. It is not the case for Desktop, no difference is found between local and global variables, except for correlated multiplication and division instructions where local variables are better in the time response.

6.4 Time response to correlated and non-correlated instructions

Figure 4 shows the time response with respect to the number of instructions for correlated and non-correlated instructions. Non-correlated instructions have better time response than correlated instructions. This is due to the role that the multi-processor architecture plays in distributing non-correlated work over the available cores (Out-Of-Order execution) [12].

Out-Of-Order execution is an important and necessary feature of modern processor. It enhances the performance of programs. The processor selects a set of sequential but non-correlated instructions within the context of the program and executes them simultaneously on the available cores, allowing for high efficiency in execution.

This advantage is illustrated in the experimental results presented in table 3. These results show that the speed of execution of non-correlated instructions is much greater than the speed of execution of correlated instructions. For this reason, the number of non-correlated instructions executed in the unit of time is much greater than the number of correlated instructions executed in the same unit of time.

Table 3: shows the number of correlated and non-correlated instructions executed in the unit of time.

Table 3: Number of instructions executed in a time unit

	Instruction Count			
Time Unit	C=A+B	A=A+B	A=A/B	C=A/B
0.513277	388	125	37	67

Note that the speed of execution of correlated instructions is much greater than the speed of execution of non-correlated instructions, and this is due to the feature of out of order execution that allows the distribution of non-correlated instructions to the processor cores.

7. RESULTS AND DISCUSSION

Tables 4,5,6 summarize the average response time for all type of instructions.

Table 4: Average Response Time for Integer data types for all instructions on local and global variables

Statement	**Local L_I**	**Global L_I**	**Local PC1**	**Global PC1**
Average	1.2546	1.9247	0.1692	0.1692

Table 5: Average Response Time for Float data types for all instructions on local and global variables

Statement	**Local L_I**	**Global L_I**	**Local PC1**	**Global PC1**
Average	0.9475	1.12525	0.12336	0.14803

Table 6: Average Response Time Double data types for all instructions on local and global variables

Statement	**Local L_I**	**Global L_I**	**Local PC1**	**Global PC1**
Average	1.2618	1.0051	0.13364	0.16036

It noticed that instructions executed on local variables have better response time than those executed on global variables.

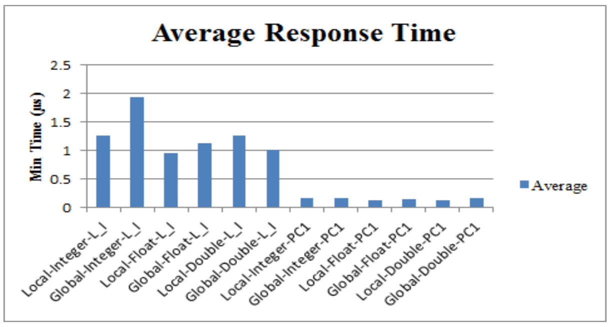

Figure 16: Average Time response for all data types for Local and global Variables

The obtained experimental results can be summarized as follows:

1- Current computer systems execute non-correlated instruction much faster than correlated instructions. This is due to modern architectural enhancements provided to the modern processors such as Out-Of-Order execution.

2- Time response patterns for C instructions and programming structures are similar when executed on computers having the same platform (processor family and operating system). In this paper, experimental study was done on computers with Intel processors and Windows operating system. Thus, the obtained results can be generalized to computer systems having similar platforms. This permits to propose a generic programming model for specific class of computer systems.

3- Time response of arithmetic and logical instructions is not related to variables values or data types, except for Double and Float data types. Instructions executed on all other data types have the same time response except for the division operation. Execution time of division is greater because of the use of more sophisticated and complex unit. Instructions with Double and Float data types have different time response from all other data types.

4- For float data type, the response time of addition, subtraction, multiplication and division arithmetic instructions is the same. This is due to the use of FPU (Floating Point Unit). The difference is between correlated and non-correlated instructions.

5- For double data types, arithmetic instructions have equal response time in the case of correlated and non-correlated instructions, except, for Laptop, correlated division has greater response time, and, for Desktop, correlated and non-correlated division have greater response time over all other arithmetic instructions. Note that correlated instructions have great response time over non-correlated instructions for both Laptop and Desktop.

6- Local and global variables have the same response time pattern for desktop computer and for all arithmetic instructions and data types. An exception is for float and double data types, where division and multiplication instructions on local variables has better response time than on global variables. For laptop, local variables have better response time than global variables for all instructions and data types except for double, where results showed that global variables has better response time than Local variables.

7- The response time of all conditional instructions is equal to the No Command time, and there is no difference between local, global variables.

8. REFERENCES

[1] Kumari, P. K. S., Kumar, S., and Sinha, S. 2016. VLSI systems energy management from a software perspective-A literature survey. ELSEVIER. pp. 611-613.

[2] Samuel, M. S. 2017. An Insight into Programming Paradigms and Their Programming Languages. Journal of Applied Technology and Innovation, vol. 1, no. 1, pp. 37-57.

[3] Alomari, Z. et al. 2015. Comparative Studies of Six Programming Languages. ArXiv. Computer Science.

[4] Pereira, R., et al. 2017. Energy Efficiency across Programming Languages How Does Energy, Time, and Memory Relate? Vancouver, BC, Canada : Association for Computing Machinery.

[5] Singh, P., et al. 2017. Performance Evaluation of Programming Languages. International Conference on Innovations in information Embedded and Communication Systems (ICIIECS).

[6] Nose, T. 2012. A Performance Comparison Method Of Programming Languages Using Source To Source Translation Technique. s.l. : Master Thesis.

[7] Chatzigeorgiou, A. , Stephanides, G. 2002. Evaluating Performance and Power of Object-Oriented Vs. Procedural Programming in Embedded Processors. Springer-Verlag Berlin Heidelberg.

[8] Kocher P., et al. 2018. Spectre attacks: Exploiting speculative execution. arXiv:1801.01203.

[9] Kattan, I., Bouchi A. , Naal, M. A. 2020. Role of Intrinsic Parallelism of Modern Processors on Performance Improvement , university of aleppo research journal, N:153.

[10] Sterling, T., Brodowicz, M., Anderson, M. 2018. High Performance Computing: Modern Systems and Practices.

[11] Pvt.Ltd, Tutorials Point (I). 2014. Learn C++ Programming Language. s.l. : tutorials Point.

[12] Fog, A. 2018. Optimizing software in C++ An optimization guide for Windows, Linux and Mac platforms.