IJCSC International Journal of Computer Science & Communication Vol. 1, No. 2, July-December 2010, pp. 47-50

Object Oriented Vs Procedural Programming in Embedded Systems

Sunil Dutt[1], Shubhnandan S. Jamwal[2] & Devanand[3]

[1,2,3]PG Department of Computer. Science and IT, University of Jammu
Email: [1]jkpenpal@gmail.com, [2]jamwalsnj@gmail.com, & [3]dpadha@rediffmail.com

ABSTRACT

The use of object-oriented technology (OOT) has been shown to be of great value in many market sectors, but to the use of such technology within embedded systems remain a challenge. Such systems require high execution speed and have high memory constraints. Object oriented model severely conflicts with the highly static model that is required in high integrity systems to meet the goals of time- and memory-boundedness assurance, and of determinism in data transformation due to code operation. The object-oriented approach is known to introduce a significant performance penalty in terms of memory and time compared to classical procedural programming. In this paper, we have analyzed the execution speed of the different programs generated by the compilers of C and Java. The investigation shows that the speed of execution shows considerable difference between the Object Oriented and procedural programming languages. It is concluded that the Object Oriented Programming languages proves to be slower than the procedural programming languages in terms of execution speed than there counterpart.
Keywords: Object oriented, procedural programming, embedded system, real-time system, high integrity system, garbage collection, locality of reference.

1. INTRODUCTION

A large percentage of the world-wide market for microprocessors is filled by micro-controllers that are the programmable core of embedded systems. Embedded system consists of microcontrollers and field programmable gate arrays as well as other programmable computing units such as digital Signal Processors (DSPs). Since embedded systems interact continuously with an environment that is analog in nature, they must typically have real time processing. A significant part of the design problem consists of deciding the software and hardware architecture for the system, as well as deciding which parts should be implemented in software running on the programmable components and which should be implemented in more specialized hardware. Embedded systems often are used in life critical situations, where reliability and safety are more important criteria than performance. Use of higher level languages such as C helps somewhat, but with increasing complexity, it is not sufficient. Formal verification and automatic synthesis of implementations are the surest ways to guarantee safety. However, both formal verification and synthesis from high levels of abstraction have been demonstrated only for small, specialized languages with restricted semantics.

The programming languages widely accepted for embedded systems are Assembly, C, C++, and JAVA. Sun's Java language [8, 9, 10] resembles C++ but is incompatible. Like C++, Java is object-oriented, providing classes and inheritance. It is a higher-level language than C++ since it uses object references, arrays, and strings instead of pointers. Java's automatic garbage collection frees the programmer from memory management but it is a very costly process. Java provides concurrent threads.

2. LITERATURE REVIEW

Despite the potential benefits, safety-critical software developers have largely avoided object-oriented methods, preferring instead to use procedural or modular approaches. However, a number of companies in safety-critical sectors are now moving to object-oriented software development, or planning such a move. This is particularly evident in the North American aerospace community. In recognition of this trend, a number of interested parties including the Federal Aviation Administration and NASA have set up the Object Oriented Technology in Aviation (OOTiA) program to address safety and certification issues when object oriented software is used in airborne applications[1].

There are a number of practical concerns over the use of OOT within the development and verification of high integrity systems that cannot be addressed directly by language level solutions. The two major concerns are Obscurity and Dead-code/Deactivated code [3].

Seragiotto, C. and Fahringer, T. reported challenging problem of performance analysis for Java programs. They describe procedures and requirements for instrumenting,

This document was created using SOLIDPDFTools To remove this message, purchase the product at www.SolidDocuments.com

monitoring, and analyzing distributed Java codes, and introduces Aksum, a highly customizable and flexible system for performance analysis that helps programmers to semi-automatically locate and understand performance problems in parallel and distributed Java programs. They also described sophisticated agent architecture as part of Aksum for static and dynamic instrumentation of Java programs. Experiments are presented for a widely distributed application running on a heterogeneous set of machines with different operating systems to prove the performance of the approach[4].

Dingle, A. Hilderbrandt, T.H. observed that Object oriented programs are simpler to implement and maintain than those using traditional programming methods. At the same time, object oriented programs create and destroy objects, incurring overhead costs. They also cause unmanned temporary objects of the same type to be created in the scope of the calling routine. They observed that both of these factors affect the performance of object oriented programs compared to procedural programs. Experimental results shows that programmers view object oriented programming as wasteful compared to procedural programming. When runtime efficiency is important, developers have a legitimate reason to reject OOP[5].

B. M. Barry studied the arguments for choosing an OOPS for implementing AMEP, which is ESM signal processor. AMEP is a large system which includes both hard real-time and knowledge-based subsystems. Extensive software metrics are presented for each subsystem and used to compare the characteristics of code designed for different purposes. For example, analysis of this data suggests that knowledge-based applications may be more difficult to port to an object-based language such as Ada than hard real-time systems. In his paper he also discussed other factors such as team programming, productivity, documentation standards and other software engineering issues[6].

JAVA lacks support for user-defined types with value semantics increases the need for dynamic memory allocation. The Microsoft language C# is in many ways similar to Java but supports value types. The provision of a garbage collector in both languages is a boon to commercial software developers but is likely to be unacceptable in real-time systems [7].

3. ANALYSIS

The common reasons studied for the slow uptake of Object Oriented Technology in Real time, Embedded and Highly integrity system is the speed of the execution of the executable code produced by procedural programming and object oriented programming style.

3.1. Processing Speed

Speed of execution of the object oriented programming languages is a major factor which is not acceptable in the embedded systems. Inability of the object oriented languages to meet the real time performance bounds is not acceptable. Object oriented technology facilitates the creation of a real-time system, but does not guarantee the final result will be real-time; this requires correct development of the software. The critical response time, called the *flyback time*, is the time that system takes to queue a new ready task and restore the state of the highest priority task. In a well-designed RTOS (real time operating system), readying a new task will take 3-20 instructions per ready queue entry, and restoration of the highest-priority ready task will take 5-30 instructions. On a 20MHz 68000 processor, task switch times run about 20 microseconds with two tasks ready. 100 MHz ARM CPUs switch in a few microseconds. In all the tests shown in table 1, between procedural and object oriented style shows that the speed of execution of object oriented programming is less then procedural programming style.

Table 1

	Speed of Execution	
	C (Procedural)	*C++ (OOPS)*
test 1	7.48	9.58
test 2	22.1	22.3
test 3	8.29	7.85
test 4	5.4	7.18
test 5	8.25	10.85
test 6	6.36	7.86
test 7	19.68	19.77
test 8	21.71	22.03

A test of the multiplication of double data type for the checking the speed of execution is performed on Operating System Microsoft windows XP Professional Version 2002 service Pack 2, CPU Pentium® 4-3.00GHZ Driver Version 5.1.25.35.0 Microsoft, Secondary Storage medium HDS728080PLA380 Version 5.1.25.35.0 Microsoft, Intel® 82801FB Ultra ATA Storage Controller 2651 Driver Version 7.0.0.1011. The compiler use for the tests for OOP's is JAVA version 1.3.0 (TM) 2 runtime Environment, Standard Edition(build 1.3.0-C) JAVA hotspot(TM) client VM (build 1.3.0-C, mixed mode) and in case of the procedural programming is TURBO C Version 3.0 Copyright(C) 1990, 1992 by Borland International, Inc. was used. In this test the random numbers were fetched by using inbuilt random function in both the compilers. Because of the primary memory constraint the numbers were stored in a secondary storage media in two different data files and then the individual numbers were fetched for multiplication. The result of the multiplication was stored in the third data file. Result of the tests is tabulated in table 2 bellow.

This document was created using SOLIDPDFTools To remove this message, purchase the product at www.SolidDocuments.com

Table 2

*5000*5000*	*10000*10000*		*15000*15000*		*20000*20000*		*25000*25000*			
	JAVA	*C*	*JAVA*	*C*	*JAVA*	*C*	*JAVA*	*C*	*JAVA*	*C*
RUN 1	1265	6	2500	10	3718	22	4953	38	6203	38
RUN 2	1250	6	2485	16	3734	22	4969	27	6204	33
RUN 3	1250	5	2500	17	3719	21	5000	28	6172	38
RUN 4	1250	6	2485	11	3718	23	4968	27	6188	33
RUN 5	1282	7	2486	11	3734	22	4969	33	6203	33
RUN 6	1250	11	2500	15	3719	22	4969	27	6187	33
RUN 7	1250	5	2500	17	3782	22	4953	27	6156	38
RUN 8	1250	6	2485	17	3719	22	4985	28	6219	33
RUN 9	1266	11	2484	15	3719	22	4969	33	6188	33
RUN 10	1250	8	2532	16	3784	22	4953	27	6187	33

4. RESULT AND DISCUSSION

The object-oriented system is heavily constrained by the nature of the object sizes. Flushing of the objects by the virtual memory manager must defragment the primary storage before the new object can be stored. It is observed that the speed of the execution of the benchmarks in case of procedural programming style is 8% more as compared to the object oriented programming. There is a tremendous amount of difference between the speed of the execution of the programs in JAVA and C. The performance of the JAVA is very poor as compared to C. This is because, the object oriented technology does not have good locality of reference. Locality of Reference means that an application does not access all of its data at once with equal probability. Instead, it accesses only a small portion of it at any given time. An application can exhibit temporal and/or spatial locality. If some data is referenced, then there is a high probability that it will be referenced again in the near future. This is called temporal locality. If some data is referenced, then there is a high probability that data nearby will be referenced in the near future.

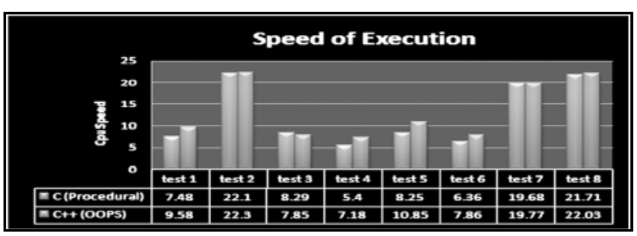

Fig. 1: Different Runs Conducted on JAVA and C Compiler on Different Sets of Data

This is called spatial locality. An application can take advantage of Locality by keeping copies of the most often or more recently used data in a faster medium. This scheme is commonly known as "caching". Generally, the locality of reference in garbage-collected systems has been very poor. In virtual memory systems, this poor locality of reference generally causes a large amount of wasted time waiting on page faults or uses excessively large amounts of main memory. Speed of the execution of the procedural style program is also more as compared to object oriented programs. So while an object-oriented paging scheme may make more sense in the context of more logical use of locality of reference, those benefits come at significant cost in terms of algorithmic complexity and overhead.

5. CONCLUSION

The development of high-performance devices not only depends on the underlying hardware architecture but also on programming languages. However, the object-oriented approach is known to introduce a significant performance penalty compared to classical procedural programming. Profiling results indicate that object oriented programs are slower than their corresponding procedural programming. It is analyzed that this is mainly due to the increased instruction count, larger code size and increased number of accesses to the data memory for the object-oriented versions.

REFERENCES

[1] OOTiA (2003), "Handbook for Object-Oriented Technology in Aviation (draft)", *OOTiA Workshop Proceedings*, March 5, 2003.

[2] B. Liskov, J. Wing, "A Behavioural Notion of Subtyping", *ACM Transactions on Programming Languages and Systems*, **16**, No. 6, November 1994.

[3] Janet Barnes, Brian Dobbing, Red Chapman, "On the Principled Design of Object Oriented Programming Language for High Integrity System", *ACM SIGPLAN PLDI*, Conference, 1994.

[4] Seragiotto, C. Fahringer, T., Performance Analysis for Distributed and Parallel Java Programs with Aksum, Inst. for Sci. Comput., Univ. of Vienna, Austria, *Cluster Computing and the Grid*, 2005. CCGrid 2005. IEEE International Symposium, 2005.

[5] Dingle, A. Hilderbrandt, T.H., *Improving C++ Performance using Temporaries*, Seattle Univ., WA, USA, 1998.

This document was created using SOLIDPDFTools To remove this message, purchase the product at www.SolidDocuments.com

[6] B. M. Barry, "Prototyping a Real-time Embedded System in Smalltalk", *Conference on Object Oriented Programming Systems Languages and Applications Archive*, New Orleans, Louisiana, United States, Year of Publication : 1989.

[7] RTJ (2003). http://www.rtj.org (30 September 2003).

[8] Ken Arnold, James Gosling, and David Holmes. *The Java Programming Language*. Addison-Wesley, Reading, Massachusetts, Third Edition, 2000.

[9] James Gosling, Bill Joy, Guy Steele,, and Gilad Bracha. *The Java Language Specification*. Addison-Wesley, Reading, Massachusetts, Second Edition, 2000.

[10] Tim Lindholm and Frank Yellin. *The Java Virtual Machine Specification*. Addison-Wesley, Reading, Assachusetts, 1999.

This document was created using SOLIDPDFTools

To remove this message, purchase the product at www.SolidDocuments.com