COMPARATIVE ANALYSIS OF C, C++, C# AND JAVA PROGRAMMING LANGUAGES

Justin Ogala
University of Delta, Agbor, Delta State, Nigeria
4 PUBLICATIONS **0** CITATIONS
SEE PROFILE

GSJ: Volume 8, Issue 5, May 2020, Online: ISSN 2320-9186
www.globalscientificjournal.com

COMPARATIVE ANALYSIS OF C, C++, C# AND JAVA PROGRAMMING LANGUAGES

By

Ogala, Justin Onyarin & Ojie, Deborah V

1&2 Department of Computer Science,

College of Education, P.M.B 2090, Agbor, Delta State, Nigeria.

E-mail: justinoo2001@gmail.com

ABSTRACT

With the emergence of software industry in recent times, more people are interested in learningprogramming languages. But nowadays there are more than 250 programming languages available, only a few of them can be applied comparatively widely. In this paper, the research in programming language was conducted. Four of the most popular programming languages C, C++, C# and Java are chosen to be the objects to study. The technical features of these four programming languages were summarized and compared with each other. To know the actual performance of these four chosen programming languages, an experiment was carried out by implementing the benchmark for each programming language. The result from the experiment was recorded and analyzed. The research concluded the most suitable application fields for these four of the most popular programming languages according to the technical features and the result from the experiment. C is suitable for systems-programming applications, hardware related applications, embedded device, chip designing, and industrial automation products. C++ is appropriate for the software development such as application software, device drivers and high-performance server. C# is proper for application development and the development of web application. Java has three different forms, Java2 Standard Edition (J2SE), Java2 Micro Edition (J2ME), and Java2 Enterprise Edition (J2EE). J2SE is suitable for the desktop applications. J2ME is proper for embedded systems development for mobile phones, wireless application and PDA programming. Finally, J2EE is appropriate for the development of server programming.

Key word: Programming language, C, C++, C#, Java, TBB

INTRODUCTION

A programming language is a set of English-like instructions that includes a set of rules (syntax) for putting the instructions together to create commands.A programming language is a formal computer language or constructed language designed to communicate instructions to a machine, particularly a computer. Programming languages can be used to create programs to control the behavior of a machine or to express algorithms.

The development in the technology of programming languages is also rapid. Since the first generation of programming languages emerges in early 1950, programming languages have five generations. The latest generation of programming languages is aimed to "make the computer solve a given problem without the programmers. However, according to the TIOBE Programming Community index that "gives an indication of the popularity of programming languages, third generation of programming language is still be widely used. The Table 1.0 shows the TIOBE Programming Community index for May 2016.

Table 1.0. TIOBE Programming Community index for May 2016.

May 2016	May 2015	Change	Programming Language	Ratings	Change
1	1		Java	20.956%	+4.09%
2	2		C	13.223%	-3.62%
3	3		C++	6.698%	-1.18%
4	5	↑	C#	4.481%	-0.78%
5	6	↑	Python	3.789%	+0.06%
6	9	↑	PHP	2.992%	+0.27%
7	7		JavaScript	2.340%	-0.79%
8	15	⇈	Ruby	2.338%	+1.07%

May 2016	May 2015	Change	Programming Language	Ratings	Change
9	11	▲	Perl	2.326%	+0.51%
10	8	▼	Visual Basic .NET	2.325%	-0.64%
11	13	▲	Delphi/Object Pascal	2.008%	+0.71%
12	22	▲▲	Assembly language	1.883%	+1.12%
13	10	▼	Visual Basic	1.828%	-0.07%
14	4	▼▼	Objective-C	1.597%	-3.80%
15	18	▲	Swift	1.593%	+0.48%
16	12	▼▼	R	1.334%	-0.11%
17	38	▲▲	Groovy	1.288%	+0.90%
18	14	▼▼	MATLAB	1.287%	+0.00%
19	17	▼	PL/SQL	1.208%	+0.08%
20	30	▲▲	D	0.975%	+0.39%

From the table above, this research chooses C, C++, C# and Java as the four of the most popular programming languages as the subjects to study.

In this paper, five generations of programming languages were briefly described by time sequence.

1. Machine languages

The first generation of programming languages is machine languages appearing in early 1950's.

It was written in binary, a series of zeros and ones. Binary is difficult to understand for human-being and was very prone to errors. The main problem of machine languages is machine dependency because "Machine languages were created differently for different for each CPU". [7]

2. Symbolic assembly languages

The second generation of programming languages is symbolic assembly programming languages which was written in a more simplistic form and at a higher level of abstraction machine languages, instead of a series of zeros and ones there were symbols (percent, dollar and portions of a word and number combination used to make commands. But symbolic assembly programming languages still have the problems of high hardware dependency and lack of portability which means assembly codes could not implemented on the different machines.

3. Problem-oriented languages

The period from the early 1960's till 1980's brought us third generation programming languages. Third generation of the programming languages is called problem-oriented languages and is also considered as the high level languages. Third generation programming languages were converted from English into machine languages; compilers were used to convert these instructions. C, C++, C and Java are all examples of third generation programming languages. Most of third generations of programming languages have the compilers or the interpreters. The advantage of this is that the programs can run very fast after compiling. The problem of "machine dependency which was encountered by the first and second generation of programming languages is no longer for the third generation of the programming languages any more. The main problem of third generation programming languages is that different type of processors need different source codes of third generation programming languages and third generation programming languages are quite hard to work out.

4. Non-procedural languages

Fourth generation programming languages are more focused on problem solving. The main difference between other generation programming languages is that they are more concerned with what is to be done than the actual how. Features evident in fourth generation languages quite clearly are that it must be user friendly, portable and independent of operating systems, usable by non-programmers, having intelligent default options about what the user wants and allowing the user to obtain results fasts using minimum requirement code generated with bug-free code from high-level expressions (employing a data-base and dictionary management which makes applications easy and quick to change). The most well-known examples of fourth Generation programming languages are SQL, MYSQL.

5. Fifth-generation programming languages

A fifth-generation programming language (abbreviated 5GL) is a programming language based around solving problems using constraints given to the program, rather than using an algorithm written by a programmer. With using the fifth generation programming languages, the computers can have their own ability to think and their own inferences can be work out by using the programmed information in large databases. The dream of a robot with artificial intelligence and fuzzy logic came true by using this generation programming languages.

BACKGROUND

Developing applications can be done in several different programming languages but implementing parallelism with the help of their respective API is quite new. These API tools were developed for current software developers on existing programming languages so that upgrading existing programs and systems for more concurrent functionality would be easy but this was not necessary until recently.

The task is to perform a comparative analysis of how parallel models are implemented in different programming languages with their respective APIs. This report will focus on C/C++ and Java/C# and compare which of these languages give the best performance. These programming languages according to TIOBE are currently the most popular and widely used programming languages by software developers and are appropriate to compare.

BRIEF HISTORY OF C, C++, C# and JAVA PROGRAMMING LANGUAGES

Brief history of C language

C is a programming language which born at 'AT & T's Bell Laboratories' of USA in 1972. It was written by Dennis Ritchie. This language was created for a specific purpose: to design the UNIX operating system (which is used on many computers). From the beginning, C was intended to be useful--to allow busy programmers to get things done.

After that, C began to be used by more and more people outside the Bell Laboratories because it is more efficient than other programming languages at that time. In the late 70's, C took the dominant position of programming languages. The committee formed by the American National Standards Institute (ANSI) approved a version of C in 1989 which is known as ANSIC. With few exceptions, every modern C compiler has the ability to adhere to this standard. ANSI C was then approved by the International Standards Organization (ISO) in 1990.

Brief history of C++ language

C++ was written by Bjarne Stroustrup at Bell Labs during 1983-1985. C++ is an extension of C. Prior to 1983; Bjarne Stroustrup added features to C and formed what he called 'C with Classes'. He had combined the Simula's use of classes and object-oriented features with the power and efficiency of C. The term C++ was first used in 1983.

C++ was designed for the UNIX system environment, it represents an enhancement of the C programming language and enables programmers to improve the quality of code produced, thus making reusable code easier to write.

Brief history of C# language

The primary architects of C# were Peter Golde, Eric Gunnerson, Anders Hejlsberg, Peter Sollichy and Scott Wiltamuth. Of course, the principal designer of the C# language was Anders Hejlsberg, a lead architect at Microsoft.

C# was designed to be a pure object-oriented programming language. C# debuted in the year 2000 at the Professional Developers Conference (PDC) where Microsoft founder Bill Gates was the keynote speaker. At the same time, Visual Studio .NET was announced.

Brief history of Java language

Java started to be developed in 1991 by James Gosling from Sun Microsystems and his team.

The original version of Java is designed for programming home appliances. In 1994, James Gosling started to make a connection between Java and internet. In 1995, Netscape Incorporated released its latest version of the Netscape browser which was capable of running Java programs.

The original name of Java is Oak. But it had to change its original name because Oak had been used by another programming language. The new name Java was inspired by a coffee bean. While Java is viewed as a programming language to design applications for the Internet, it is in reality a general all-purpose language which can be used independent of the Internet.

MATERIALS AND METHODS

The method chosen for testing and comparing the programming languages parallel model is by execution time performance. The benchmarking programs were tested in different programming languages.

Execution time performance was chosen as the main testing criteria as todays focus when developing multi core processors is speed. It is then only reasonable to test the execution time performance of different parallel models for the differentprogramming languages.

The benchmarks were executed on the College of Education, Agbor, ICT centre, multi core computer with a total of 48 cores divided on 4 sockets. Each socket has 2 NUMA-nodes, each with 6 processors and 8 GB of local RAM. Each of the processors is based on the AMD x86-64 architecture (Opteron 6172).

The benchmarks were executed 5 times to get a median value and then shifted up to the next amount of threads. The numbers of threads tested were 1, 2, 4, 8, 16, 24, 36 and 48. This was done by a simple script that executed the benchmark several times with different input for the number of threads used.The speedup was calculated as $S(n) = T(1)/T(n)$, where n is number of threads, $T(1)$ is the time it takes to execute with 1 thread, $T(n)$ for n threads and $S(n)$ is the achieved speedup.
Analyzing the execution time was done simple. Each test provided a speed result of its parallel and serial sections. The results were then used to compare each programming language

parallel execution time performance compared to its sequential counterpart giving a perceptual speedup depending on the number of threads used. This was one type of test that provided a simple overhead of the speedup each language obtained when increasing the number of threads. The second type of test was to compare the overall speedup with a specific number of threads, comparing the languages with each other. The third type of test was to find outhow well each language could handle different granularities on tasks without losing its overall speedup.

Testing C

The benchmark SparseLU from BOTS was originally written in C with the OpenMP 4.0 API for the parallel regions of the code. There was no need to rewrite or change the structure of the program since a task centric version of the code was available.

Minor changes before it was put to use was made for instance the program was ported all into one file for easy reading and future porting to other languages. All the variables and functions were all in one file.

To test the benchmark with different number of threads as well as different granularities, the program was edited once more to fetch input of the amount of threads, matrix size and sub matrix size to be used for easier testing. This led to the program able to run the benchmark program line *sparselu 4 50 100* where 4 is the number of threads, 50 is the matrix size and 100 is the sub matrix size.

Testing C++ TBB

The TBB version of the benchmark was also available so no porting was made. It used the original testing and compiling principles as the original BOTS package. This automatically provided measurements for the parallel sections of the bench- mark.

Compiling and running the complete package provided by BOTS was done in several steps. To compile and run it on the multi core server, the command *source /opt/intel/bin/compilervars.sh intel64* was used to configure the compiler *tbb(icc)*. TBB (ICC) stands for Intel's C Compiler for the TBB version. A make file was already provided to compile the code. To run it the following file *sparselu.icc.tbb -w* was executed with the flag *-w 48* where 48 is the number of threads to be used.

Testing C++

Unfortunately testing C++ was proven difficult. Porting the sequential code from C to C++ was easy but implementing this new parallel standard was difficult without changing the parallel model it originally had in C, since C++ does not fully support task centric parallelism. C++ focused on recursive parallelism and even the asynchronous functions of the language focused on the same functionality. Because of this it was not tested and no results will be presented for C++. The standard is fairly new and rarely used in the programming community, it is mainly researched on.

Testing C#

First, sparseLU was ported from C to C# in Visual Studio 2015 where it was built, compiled and debugged. Since C# didn't support pointers and pointer arithmetic's like C, an approach using *out* and *ref* was chosen to minimize the number of data copies. Out and ref are used to send secure pointers to variables as parameters to functions in C#. Another problem was that C# didn't allow free control of the number of threads in the thread pool. It was a crucial problem that was finally solved by restricting the number of tasks run by the thread pool instead of restricting the number of threads. This came with a bit of performance loss as the overhead became bigger. The tasks were created and managed by the Task Parallel Library (TPL).

To test the benchmark with different number of threads, the program was edited to fetch input of the amount of threads to be used for easier testing. To run it on server, the following line was used: *mono sparselu.exe numThreads,* where numThreads are the number of threads to be used in the parallel version.

Testing Java

C# and Java have similar syntax and when the C# version of sparseLU finished it was easy to port from C# to Java in Eclipse. It was built, compiled and debugged in Eclipse. To counter the loss of pointer arithmetic, the matrix was made global so every thread had access to it. To control the number of threads the ForkJoinPool class in the concurrent package was used.

A*ForkJoinPool* object controls an underlying thread pool with a specified number of threads and has a queue for tasks. If the number of tasks is greater than the number of threads, the rest of the tasks wait in the queue until a thread goes idle and can execute another task.

To be able to test the benchmark with different number of threads, the program was edited to fetch input of the amount of threads to be used for easier testing. To run it on server, the following line was used: *java Main numThreads*, where numThreads are the number of threads to be used in the parallel version. The benchmark was edited once more to be able to fetch input for different matrix sizes. To run the program with granularity size input the following line was used *java Main numThreads size subsize*.

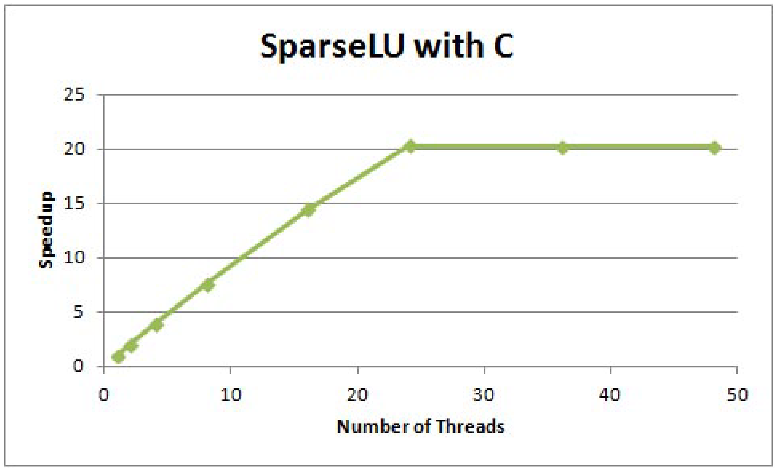
RESULTS AND DISCUSSIONS

The discussion is split in two parts. The first part is about the tests with different number of threads. The second part is about the tests with 48 threads run on different granularity. Both parts focus on the two factors speedup performance and execution time performance as well as some more overall discussion.

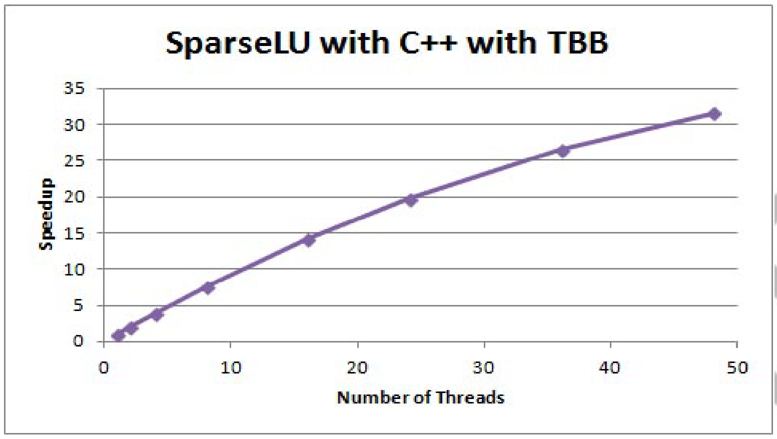
Both Java and C# uses garbage collection but C and C++ don't. This is important to take into account when comparing low-level languages to mid- or high-level languages and when comparing mid- or high-level languages with themselves as the garbage collection may differ. Unfortunately the tests in this research couldn't measure and compare the garbage collection on Java and C#.

Performance with number of threads

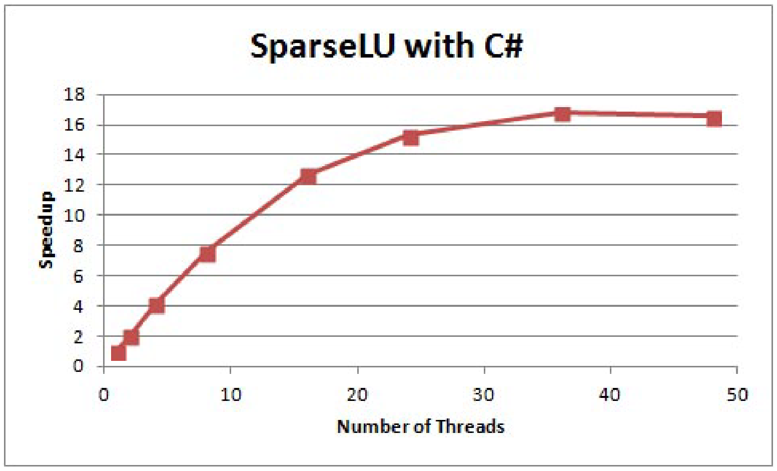
The discussion on performance with number of threads will mainly be around the two combined graphs 1, 2, 3 and 4 below.



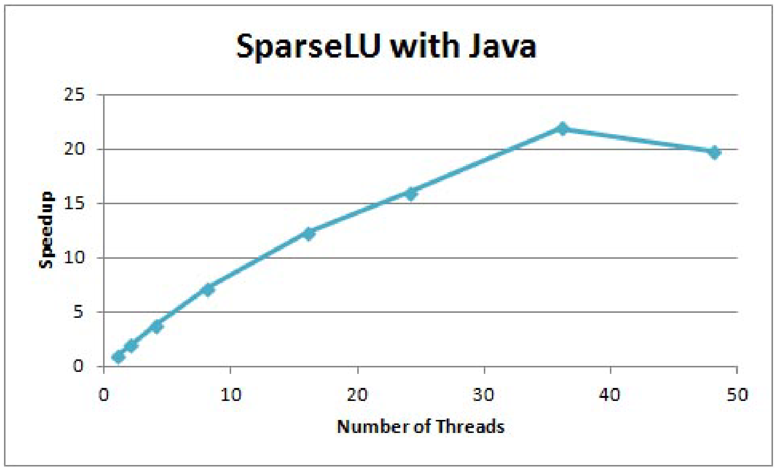
1. SparseLU written in C



2. SparseLU written in C++ with TBB



3: SparseLU written in C#



4: SparseLU written in Java

Graphs showing the achieved speedup with different number of threads on the SparseLU benchmark

Speedup

The programming language C++ using the TBB API has shown the greatest speedup with just a slight loss after larger number of threads. This is most likely due to the overhead of each task the TBB model has to use to execute them compared to the programming language C where the increasing number of threads doesn't affect its linear speedup.

A peculiar result produced by the programming language C with OpenMP is that after a certain amount of threads available the linear speedup stops flat. This is because at 24 threads it has reached an execution time of 1 second and because of this the increased overhead is canceling out the gains of more threads.

The same goes for the Java benchmark using fork/join. Although it is a bit slower than C it reaches the same speedup and an execution time of 1 second at 36 threads. It then drops a little in speedup due to the increased overhead being bigger than the gains of more threads.

In order to test the true potential of C and Java a bigger data-set is needed because measuring execution times at 1 second and bellow is inaccurate. A bigger data-set would take more time to execute giving more accurate measuring of the speedup gains at 24 threads and above for C and Java.

C# didn't perform as well as the three other in terms of speedup. It may depend on the problem limiting the number of threads running in parallel forcing a solution where the tasks were limited instead. This had an unknown impact on the performance and in order to fairly test the thread pool in C# another approach is needed.

Execution Time

In terms of execution time the OpenMP parallel model for the programming language C dominated the benchmarking tests with the fastest execution time while C# performed the worst.

Comparing the low level programming languages C and C++, C dominated the tests due to the fact that it made use of its pointer arithmetic and with no dependencies each task could handle the matrix without moving any of the sub matrices. For the programming language C++ the TBB API had much extra code compared to C in order to extract each task in a TBB model making it a bit slower than C. The overall speedup is most likely affected by the overhead each task is given by the TBB model.

Comparing the mid-level programming languages Java and C#, Java performed the best. It even performed better than the low level programming language C++. Java has a JIT (Just In Time) compiler which makes optimizations before it is executed on a specific machine. This makes it much faster than C# which also handles a virtual machine in between like Java but with no JIT. This leaves Java in the same result category as C as it also stops improving after a certain amount of threads due to the fast execution time.

Both C and Java performed really well on the test, C being a little faster than Java. The surprise is that Java performed so well when the hypothesis stated that the opposite was expected.

CONCLUSION AND RECOMMENDATIONS

Conclusion

The programming language C as well as Java dominated the tests in terms of execution time while C++ and C dominated the overall speedup gained. The granularity tests showed that Java could handle small tasks while still keeping its overall speedup in comparison to the other three languages. In all of the tests the programming language C# performed the worst in terms of execution time, overall speedup and the amount of granularity the language could handle before the overhead of the tasks took over.

According to the results the most effective programming language in parallel programming is C with the OpenMP API and Java with the fork/join method. C and Java gave the fastest execution time of all languages and handled granularity in a very efficient way without wasting execution time or overall speed up, especially Java that could handle very small granularity for each task and thread.

Recommendations

Further studies within this area have great potential, the comparison between programming languages and their respective parallel programming models would prove beneficial to future software development especially when hardware development today are focused in developing multi core processors rather than faster single core processors.

This research could also be beneficial for institutes researching in developing parallel programming APIs. This research gives an overview of how each programming language performs from a parallel point of view, either lacking in support and performance or exceling in these areas.

REFERENCES

A Brief History of the C Language, available at
http://hubpages.com/technology/A-Brief-History-of-the-C-Language (Accessed May 25th, 2016)

A Brief History of C++, available at
http://mathbits.com/MathBits/CompSci/Introduction/history.htm (Accessed May 25th, 2016)

Artur Podobas. Performance-driven exploration using task-based parallel programming frameworks. Licentiate thesis, KTH Royal Institute of Technology, Information and

Communication Technology, Sweden

Beginning Java - Unit 1 Brief History - MathBits.com, available at http://mathbits.com/MathBits/Java/Introduction/BriefHistory.htm (Accessed May 25th, 2016)

E. Balagurusamy, Fundamentals of Computers, Mcgraw Hill Education (India), 2009, ISBN 978-0070141605

Paul Jansen and Bram Stappers. Tiobe programming community index, April 2013.

Richard Grigonis. "FIFTH-GENERATION COMPUTERS". Retrieved 2016-03-05.

The Evolution of Computer Programming Languages, available at http://www.associatedcontent.com/article/369862/theevolutionofcomputer_programming.html (Accessed May 25th, 2016)

TIOBE Programming Community Index for May 2016, available at http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html (Accessed May 25t h, 2016)

Wikipedia, Fifth-generation programming language, available at http://en.wikipedia.org/wiki/Fifth-generationprogramminglanguage (Accessed May 25t h, 2016)

http://en.wikipedia.org/wiki/Comparison_of_programming_languages (Accessed May25t h, 2016)

http://www.jvoegele.com/software/langcomp.html (Accessed May25t h, 2016)

http://www.mathsisfun.com/combinatorics/combinations-permutations.html (Accessed May 25t h, 2016)

http://www.threadvisors.com/lang cmparison.htm (Accessed May25t h, 2016)

http://userweb.cs.utexas.edu/users/djimenez/utsa/cs3343/lecture25.html (Accessed May25t h, 2016)

http://www.tutors4you.com/circularpermutations.htm (Accessed May25t h, 2016)