Rjeas Research Journal in Engineering and Applied Sciences 3(4) 262-266 Rjeas
© Emerging Academy Resources (2014) (ISSN: 2276-8467)
www.emergingresource.org

# IMPACT OF MICROSOFT VISUAL C++ VERSION ON THE PERFORMANCE OF ARRAYS AND VECTORS

**K.K.L.B. Adikaram[1,2,3], M.A. Hussein[3], T. Becker[3]**
[1]Computer Unit, Faculty of Agriculture,
University of Ruhuna, Mapalana, Kamburupitiy, Sri Lanka.
[2]Institut für Landtechnik und Tierhaltung, Vöttinger Straße 36, 85354 Freising, Germany.
3Group Bio-Process Analysis Technology, Technische Universität München,
Weihenstephaner Steig 20, 85354 Freising,Germany.
**Corresponding Author:** *K.K.L.B. Adikaram,*

## ABSTRACT
Execution time of an algorithm is a critical factor in process optimization. We found that the Visual C++ version has a significant impact on the performance of memory access, especially on vectors. There is no surprise if the lower version has poor performance. However, we found that the performance of the Visual C++ 2012 is lower than that of version 2010. The results clearly show that there is a remarkable difference of performance in relation to the Visual C++ version with default compiler settings. Visual C++ 2010 reported the best performance in relation to array and vector versions. Visual C++ 2012 vector versions of the selected algorithms reported 185% to 244% clocks per element in relation to the Visual C++ 2010 vector versions. Visual C++ 2012 array versions reported 101% to 143% clocks per elements in relation to the Visual C++ 2010 array versions.



**KEYWORDS:** Fast algorithm, array vs. vector, performance of .Net versions, process optimization, Visual C++

## INTRODUCTION
The generally accepted phenomenon is that the new version of a certain programming language performs better than the previous version. We did performance analysis of several algorithms with Visual C++ 2010 and memory structures of primitive arrays and vectors. Due to upgrading of the system we got the new version of Visual C++ 2012 and expected much better performance. Unfortunately, the algorithms with vector memory structures reported poorer performance compared with the 2010 version. This observation motivated us to study the performance in relation with different versions of a specific programming language. Performance of an algorithm can be locally improved by reducing the arithmetic load of the algorithm (Burrus, 2008) and improving input / output speed (Karp, 1996). Domain factors such as hardware properties, operating system, and programming language (Fourment & Gillings, 2008) are the global factors that influence the performance of an algorithm. The version of a specific programming language has direct relation with memory management, which is considered as a part of input / output management. However, we were unable to find a detailed study on the effect of the different versions of a specific programming language on the performance of an algorithm. Therefore, we decided to conduct this study to understand the impact of Visual C++ version on the performance of primitive array and vector memory structures. Our findings may be important to researchers who are engaged in process optimization.

The cost of memory access is a critical factor with high impact on CPU performance (Gog & Petri, 2013). There are two ways of allocating memory for a series of variables: “slot of continuous memory elements” or “collection of non-continuous memory elements”. The term “stack” is the general technical term referring to “slot of continuous memory elements” and the term “heap” is the general technical term referring to “collection of non-continuous memory elements” (Lyons, 2004). Stack memory allocates a continuous slot of memory and provides easy reading and writing of variables. Therefore, stack memory is considered as fast. In contrast, heap memory is not physically continuous, making it necessary to have pointers to access the memory. Thus, reading and writing of variables in heap memory is considered as slow. Array is another term that refers to “slot of continuous memory elements” in most of the programming languages (Donovan, 2002; Mcconnell, 2004). Arrays are the simplest and most common type of data structure (Mcconnell, 2004).

C++ is one of the most common and powerful programming languages that has been in use since 1984 (Koenig & Stroustrup, 1995; Stroustrup, 2007). With the introduction of “Standard Library” many other methods of memory allocation were introduced as different container types such as list, vector and map (Stroustrup, 2007), in addition to the standard “slot of continuous memory elements” method in C++. All the containers are “collections of non- continuous memory elements”. The

memory layout shows the structure of the memory as illustrated in Figure 1 for the memory layout of C++ (Stroustrup, 1997, 2012). Heap and stack storages are the most frequently used memory for variables.

Microsoft Visual C++ provides different types of memory structures such as Deque, Hash_map, Hash_multimap, Hash_multiset, Hash_set, List, map, Multimap, Multiset, Set, Vector, Priority_queue, Queue, Stack, Array, ArrayList and many more (Microsoft). Vector is the most commonly used memory structure among those varieties of different structures (Mcconnell, 2004; Stroustrup, 2009). All these memory structures allocate memory form heap.

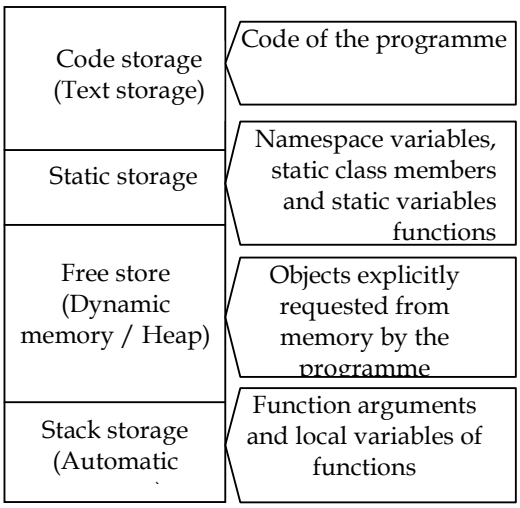


Figure 1 - Memory layout of C++

The structure of memory is first decided by the machine architecture (Karp, 1996), and the operating system is responsible for managing the memory. Different operating systems manage the same memory architecture in different ways. In addition, performance of the memory system depends on the selected programming language (Stroustrup, 2012). Programming languages provide different memory structures (Stroustrup, 2009) which handle the memory in different ways. Therefore, efficiency of the memory is the resultant output of performance of hardware, performance of the operating system, performance of the programming language and of the selected memory structure that is provided by the programming language. In addition, performance can be further tuned by changing the options provided by the compiler of the programming language.

## METHODOLOGY

Three algorithms were selected for evaluating the influence of C++ version on performance of arrays and vectors, namely "Elster's Bit-Reversal" algorithm (Elster, 1989), "Linear Bit-Reversal" algorithm of Elster (Elster & Meyer, 2009), and the algorithm by Rubio *et al*. (Rubio, Gomez, & Drouiche, 2002). The selected algorithms are used for creating bit reversal permutation, which is a sub-process of the Fast Fourier Transformation process. Furthermore, the majority of operations in the algorithms are related to operations on memory structures. Visual C++ 2008, Visual C++ 2010, and Visual C++ 2012 as specified in Table **1** were evaluated for performance.

We recently installed a PC with specifications given in Table **1** with default settings. To eliminate limits on memory and address space related to the selected platform, the Visual C++ compiler option *"/LARGEADDRESSAWARE*" was set (Microsoft) and the platform was set as "x64". Except for these two options, all other options of compiler were kept unchanged.

Table 1 – Specifications of PC, OS and Visual C++

| | |
|---|---|
| Processor | Intel Core 7i CPU 870 @ 2.93GHz |
| RAM size | 12 GB |
| L2 Cache | 256 KB per processor core |
| L3 Cache | 8192 KB |
| Brand and type | Fujitsu, Celsius |
| Bios and OS settings | Default |
| OS and Service pack | Windows 7 professional (64 bit) with SP1 |
| Microsoft Visual Professional 2008, Framework Version | Studio Version 20089.0.21022.8 RTM, .NET Version 3.5 SP1 |
| Microsoft Visual Professional 2010, Framework Version | Studio Version 10.0.40219.1 SP1Rel, .NET Version 4.5.50709 SP1Rel |
| Microsoft Visual Professional 2012, Framework Version | Studio Version 11.0.50727.1 RTMREL, .NET Version 4.5.50709 |

We considered the memory allocation techniques mentioned in Table 2 to implement algorithms. Theoretically, the fastest memory structure is the primitive array, as it uses stack storage for allocating memory. However, the primitive array does not support dynamic memory allocation, *i.e.* the size of the array needs to be declared. Even after setting the compiler option *"/LARGEADDRESSAWARE*, memory structures 3 and 4 mentioned in Table 2 did not support accessing memory greater than 2GB. Hence, we excluded those three memory structures and used memory structure 2 (array) and memory structure 5 (vector) to implement the selected three algorithms. Both the selected memory structures allocate memory in heap, where they have equal status.

Table 2 - Common Visual C++ syntaxes use to allocate memory for series of variables and memory location in memory layout

| No | Memory structure | Syntax | Memory layout |
|---|---|---|---|
| 1 | Array | int BRO[1000] | stack storage |
| 2 | Array | int* BRO =new int[N] | free storage |
| 3 | Array | array<int> ^BRO = gcnew array<int>(N) | free storage |
| 4 | Arraylist | ArrayList ^ BRO = gcnew ArrayList() | free storage |
| 5 | Vector | std::vector< int> BRO (N) | free storage |

The selected algorithms were implemented with selected memory structures (memory structure 2 and memory structure 5 in Table 2) with 2008, 2010, and 2012 versions of Visual C++. Performance of algorithms was evaluated considering the "*clocks per element*" (CPE) of each algorithm. To get this value, first average running times for each sample size (memory size) of $2^{21}$ to $2^{31}$ were calculated after executing each algorithm 100 times

**RESULTS**

Average CPE consumed by array and vector versions of Elster's Bit-Reversal algorithm, Linear Bit-Reversal algorithm of Elster, and algorithm by Rubio *et al*. are shown in Figure 2, Figure 3, and Figure 4. Vector version (A) and array version (B) are in the same scale of each figure.

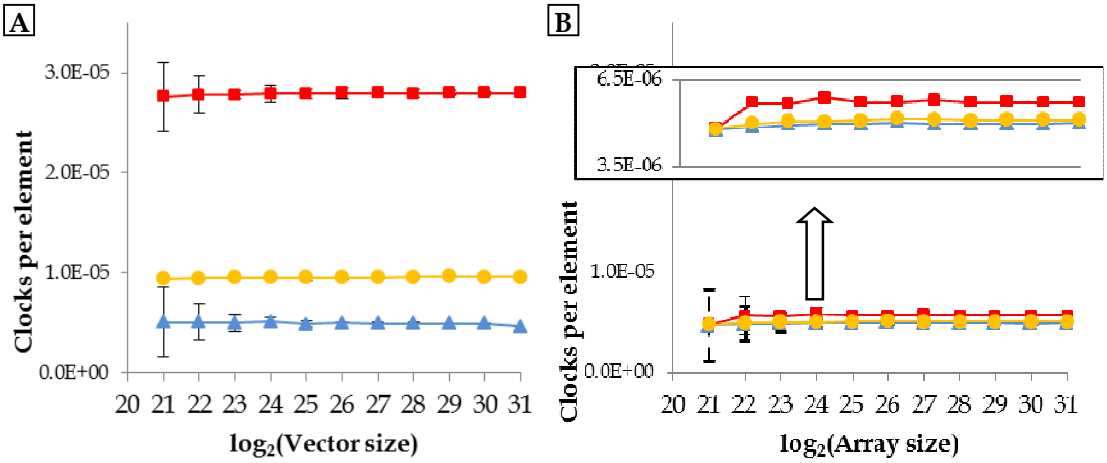


Figure 2: Average clocks per element vs. $\log_2$(size of memory structure) for the Linear Bit-Reversal algorithm of Rubio *et al.* . Where "A" corresponds to the vector version, "B" corresponds to the array version of the algorithm, (—■—) corresponds to the 2008 version, (—▲—) corresponds to the 2010 version, and (—●—) corresponds to the 2012 version of Visual C++.

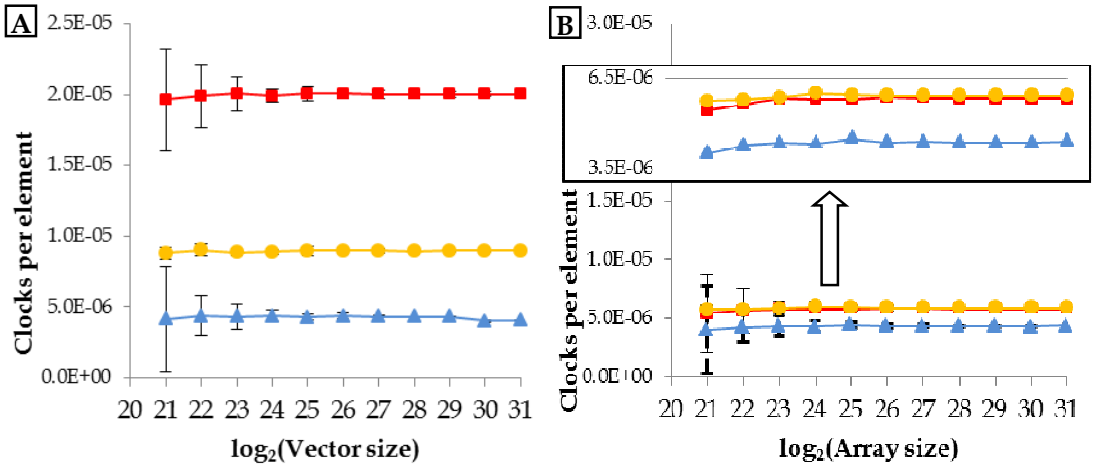


Figure 3: Average clocks per element vs. $\log_2$(size of memory structure) for the Linear Bit-Reversal algorithm of Elster. Where "A" corresponds to the vector version, "B" corresponds to the array version of the algorithm, (—■—) corresponds to the 2008 version, (—▲—) corresponds to the 2010 version, and (—●—) corresponds to the 2012 version of Visual C++.

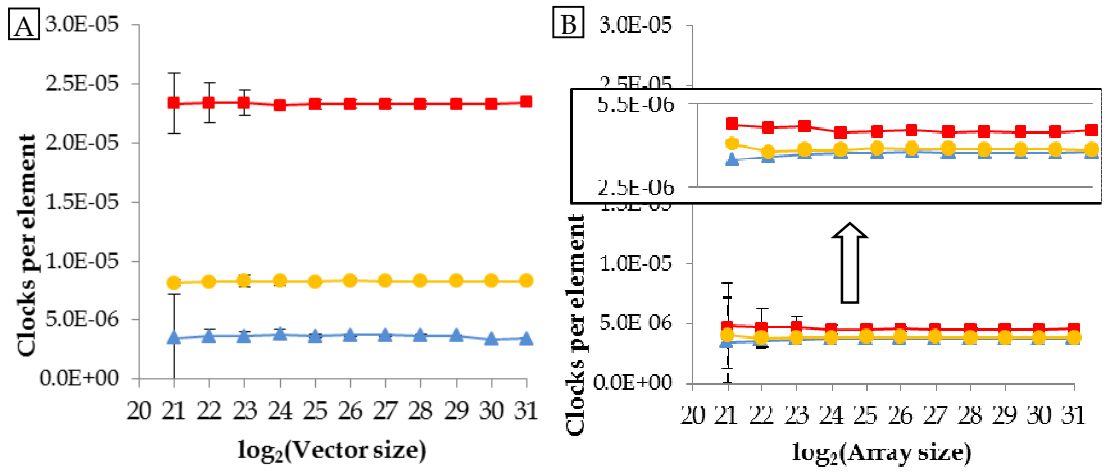


Figure 4: Average clocks per element vs. $\log_2$(size of memory structure) for the Elster's Bit-Reversal algorithm . Where "A" corresponds to the vector version, "B" corresponds to the array version of the algorithm, (—■—) corresponds to the 2008 version, (—▲—) corresponds to the 2010 version, and (—●—) corresponds to the 2012 version of Visual C++.

## DISCUSSION

The results clearly show that there is a remarkable difference in performance in relation to the Visual C++ version with default compiler settings. Visual C++ 2010 reported the best performance in relation to array and vector versions. Visual C++ 2008 vector versions of the selected algorithms reported 454% to 685% CPEs and Visual C++ 2012 vector versions of the selected algorithms reported 185% to 244% CPEs in relation to the Visual C++ 2010 vector versions. Visual C++ 2008 array versions of the selected algorithms reported 101% to 137% CPEs and Visual C++ 2012 array versions of the selected algorithms reported 101% to 143% CPEs in relation to the Visual C++ 2010 array versions. Since the version 2010 is the higher version, there is no hesitation about the low performance of version 2008. However, the low performance of the higher version 2012 in relation to version 2010 implies that it is not good practice to believe that a new version will always have better performance. On the other hand, standard deviations reported by the 2012 version of all selected algorithms were very low. This implies that the version 2012 is much more consistent than versions 2008 and 2010. The most possible reason for this observation is the performance of different techniques used in both versions for managing the memory. In the newer version, priority is given to the stability and consistency of the performance.

With the introduction of vectors, programmers used and were encouraged to use vectors as a memory structure because of their flexible nature. All the algorithms checked have no complex operations such as deleting and sorting, but only inserting at the end. The results show that the performance of vectors is better than arrays only with Visual C++ 2010. CPE ratio of array and vector of version 2010 is between 91% and 106%. However, it is between 148 % and 218% for the version 2008. For the version 2012, it is between 342% and 572%. These results imply that the selection of vector or array as a memory structure has considerable effect on performances of the algorithm. Results implied that when there are no complex operations on memory structures, it is better to use array for better performance. Due to the fact that the structure of vector gives priority to generality and flexibility rather than other performance factors such as execution speed and memory economy (Fog, 2014), vectors consume more execution time.

All the results we showed are related to default settings. However, version 2012 may perform much better than version 2010 with different compiler options instead of default options. Unfortunately, we could not find such information. Furthermore, different hardware architecture may show much better performance of Visual C++ 2012.

## CONCLUSION

Results clearly show that the generally accepted idea of "the latest version of a programming language performs better than the earlier version" is not always true. The best performance of both vector and array type memory structures was reported for Visual C++ 2010, not for the newer version 2012. Furthermore, results show that the impact of selecting vector memory structure is significant. Visual C++ 2010 is the only version that reported better results for vector in comparison with array. From the point of view of performance of Visual C++, arrays are better than vectors and the latest version is not always the best. Our current findings may be especially useful to researchers who are engaged in process optimization. However, to identify the real impact of the different versions on performance of memory, assembly code analysis is required. This result will be useful to programmers who are engaged in building compilers.

## REFERENCES

Burrus, C. S. (2008). Fast Fourier Transforms: Houston, Texas, Rice University.

Donovan, S. (2002). C++ by Example, UnderC (Learning Edition ed.): QUE Corporation.

Elster, A. C. (1989). Fast Bit-Reversal Algorithms. Paper presented at the Internat. Conf. Acoustics, Speech and Signal Processing.

Elster, A. C., & Meyer, J. C. (2009). A Super-Efficient Adaptable Bit-Reversal Algorithm for Multithreaded Architectures. 2009 Ieee International Symposium on Parallel & Distributed Processing, Vols 1-5, pp. 2131-2138.

Fog, A. (2014). Optimizing software in C++ Retrieved 2014, May, from http://www.agner.org/optimize/

Fourment, M., & Gillings, M. (2008). A comparison of common programming languages used in bioinformatics. BMC Bioinformatics, 9(1), 1-9. doi: 10.1186/1471-2105-9-82

Gog, S., & Petri, M. (2013). Optimized succinct data structures for massive data. Software: Practice and Experience, n/a-n/a. doi: 10.1002/spe.2198

Karp, A. H. (1996). Bit-Reversal on Uniprocessors. SIAM Review, 38(1), pp. 1–26.

Koenig, A., & Stroustrup, B. (1995). Foundations for native C++ styles. Software: Practice and Experience, 25(S4), 45-86. doi: 10.1002/spe.4380251304

Lyons, R. G. (2004). Understanding Digital Signal Processing: Prentice Hall PTR.

Mcconnell, S. (2004). Code Complete (Second Edition ed.): Microsoft Press.

Microsoft. Memory Limits for Windows Releases. Microsoft, USA Retrieved 2012, Oct., from http://msdn.microsoft.com/en-us/library/aa366778%28VS.85%29.aspx#memory_limits

Microsoft. STL Containers. Microsoft, USA Retrieved 2012, Aug., from http://msdn.microsoft.com/en-us/library/1fe2x6kt%28v=vs.110%29.aspx

Rubio, M., Gomez, P., & Drouiche, K. (2002). A new superfast bit reversal algorithm. International journal of adaptive control and signal processing, 16, 703–707

Stroustrup, B. (1997). The C+ + Programming Language (Third Edition ed.): AT&T Labs.

Stroustrup, B. (2007). Evolving a language in and for the real world: C++ 1991-2006.

Stroustrup, B. (2009). Programming - Principles and Practice Using C++: Addison-Wesley.

Stroustrup, B. (2012). Software Development for Infrastructure. IEEE Computer Society, 45, pp. 47-58.