International Journal of Information Technology and Knowledge Management
July-December 2010, Volume 2, No. 2, pp. 515-517

QUANTIFYING TAIL RECURSION, RECURSION AND ITERATION

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

In this paper we developed and measured the performance of recursive code, tail recursive code, and iterative code on the compilers of JDK5 and Borland C. We executed the different version of three types of sorting techniques that is bubble sort (BuS), selection sort (SelS) and Quick sort (QkS). The average of all the benchmarks in JDK5 compiler comes to 30.63 ms for recursive, 57.26 ms for tail recursive and 85.33 ms for iterative version. On the compiler of the Borland C the average of all the benchmarks is 1.27 ms for recursive, 4 ms for tail recursive and 8.4 ms for iterative version. There fore in this paper it is concluded that the recursive version is fastest than the tail recursive and iterative version. We present evidence to show the utility of recursion over tail recursion and iteration.
Keywords: Recursive, tail recursive, iterative, sort, performance, execution speed.

INTRODUCTION

Recursion is a simple mechanism for expressing solutions then a loop. Programmers also feel that recursion is a stylistically preferable way to write loops because it avoids assigning variables. A recursive call requires the compiler to allocate storage on the stack at run-time for every call that has not yet returned but tail recursion is a form of recursion that can be implemented much more efficiently than general recursion. It is the special case of recursion that is semantically equivalent to the iteration constructs. Tail recursion is equivalent to iteration and tail-recursive programs can be compiled as efficiently as iterative programs. The proposed study carried out in this paper shows that the tail recursive programs performs well than the iterative ones but not by the recursive ones.

The memory consumption makes recursion unacceptably inefficient for representing repetitive algorithms having large or unbounded size but it is understood the in terms of the speed of the execution of the programs the efficiency of the recursion is more as compared to iterative counterparts [2]. In this paper we provide a quantitative analysis of recursion, tail recursion and iterative solution in procedural language C and object oriented language JAVA to measure the efficiency of the three constructs.

[1]Asst. Professor, PG Department of Computer Science and IT, University of Jammu

[2]Research Associate, PG Department of Computer Science and IT, University of Jammu

[3]Professor and Head, PG Department of Computer Science and IT, University of Jammu

Email: jamwalsnj@gmail.com[1], jkpenpal@gmail.com2, dpadha@rediffmail.com

RELATED WORK

Clinger, W. D. [1] observed that functional languages make heavy use of tail recursion, and recursion in general, and most implementations ensure that tail recursion is handled efficiently. In general this means a constant use of space. In general, these implementations rely on heap allocation. Bentley J. [3] observed that in C it is the code can be converted into tail recursion into simple iteration by the programmer only. This optimization can be performed at the source level. Greg Stitt et al.[4] showed that the convenience of recursive algorithms has made recursion a widespread programming practice, therefore limiting the applicability of high-level synthesis tools.

They introduced a new synthesis technique called recursion flattening that reduces the limitations caused by recursion for high-level synthesis. Recursion flattening can eliminate many instances of recursion by determining recursion depth, and then in-lining recursive calls. MIT group headed by Leiserson [5] developed “Cache Oblivious algorithms” that used recursive layouts and algorithm for automatic data locality for complicated memory hierarchies. Recently, Chatterjee et al. [6] evaluated five recursive data layouts for two dimensional arrays. Four of these are different Morton variants (Z, U, X, G) and the last one is the Hilbert layout for a space-filling curve5. These recursive layouts are used in three recursive blocked algorithms for matrix multiplication. One of their conclusions is that recursive array layouts significantly outperform traditional array layouts for the blocked recursive standard algorithm, where splitting is used in all three problem dimensions. Another contribution of their work is explicit expressions for the recursive layout functions. Moreover, they claim that the overheads due to addressing these recursive layouts are only marginal. Many studies carried out the worst-case execution time analysis for real-time systems [8, 9] and determine the maximum execution time of a recursive program for a given input.

HARDWARE AND SOFTWARE

The performance of the different bench marks is measured on Intel core(R) Core(TM) 2 DUO CPU E4500 @ 2.20 GHz intel(R) EM64T capable, 32 bit system and 1GB RAM. The frequency of the memory is 667 MHz. Microsoft Windows Vista™ Business copy right(c) 2006 with Service pack 2 was used as an operating system. L2 cache RAM in the system is 2048KB. The compilers used for the measuring the performance of three different benchmarks is Java version "1.5.0" Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0-b64), Java HotSpot(TM) Client VM (build 1.5.0-b64, mixed mode). Another compiler used for testing the results is Borland C Version 5.02.

ANALYSIS

Table 1
Performance on JDK5

Rec.	Tail rec.	Iterative	
BuS	62.5	109.4	128.2
SelS	40.6	47	46.6
QkS	12.8	15.4	81.2
Average	38.63	57.26	85.33

Table 2
Performance on Borland C

Rec.	Tail rec.	Iterative	
BuS	1.8	8.2	10.4
SelS	1.6	2.6	2.2
QkS	0.4	1.2	12.6
Average	1.27	4	8.4

Most of the researchers find out that the performance of the recursive code in terms of the space and time is poor as compared to the iterative code. But in our understanding the performance of the recursive code in terms of space is less as it is well know fact that the function call needs a space for the formation of stack. But in terms of the execution speed the performance of the recursive code is better. We have measured execution speed of the recursive, tail recursive and iterative version and it is observed that the recursive and tail recursive solution performs better than iterative version of there counterpart. We developed and executed the different version of three types of code bubble sort (BuS), selection sort (SelS), Quick sort (QkS). Different arraylets of random numbers of 5000 are created. Then each of the different sorting technique is applied on them and the execution speed of the three types of the solution is measured in ms. Table 1 shows the performance of bubble sort (BuS), Selection Sort (SelS) and quick sort (QkS) for recursive, tail recursive and iterative version of different benchmarks. Pictorial representation of table 1 is depicted in figure 1 to and the figure 2 depicts the performance of the different benchmarks of table 2 for Borland C compiler. Each of the entry in the two tables is the average of ten runs for each version of the different benchmarks.

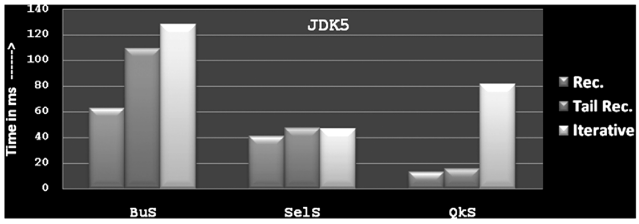

Fig. 1: Pictorial Representation of Table 1

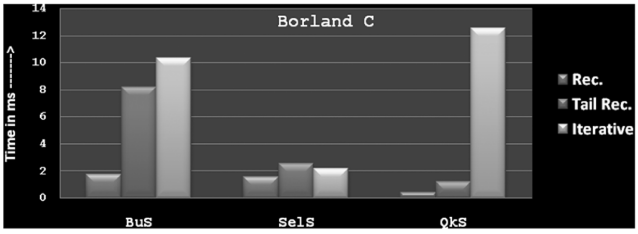

Fig. 2: Pictorial Representation of Table 2

CONCLUSION

N.Park, B.Hong, and V.K. Prasanna [7] showed that recursively blocked algorithm improve on temporal data locality and the hybrid data formats improve on spatial data locality to the extent that is possible. Each block gives optimal temporal data locality when in L1 cache. A good understanding of recursion is an essential asset for programmers. Recursion is a powerful problem solving technique in programming and helps to better understand and manipulate complex data structures. There fore in this paper it is concluded that the recursive version is fastest than the tail recursive and iterative version.

REFERENCES

[1] Clinger, W. D. Proper Tail Recursion and Space Efficiency. In Proceedings of the ACM SIGPLAN '98 Conference on Programming Language Design and Implementation (1998), pp. 174–185.

[2] Shubhnandan S. Jamwal, Pawnanesh Abrol and Devanand, "Recursion in Java: a Quantative Analysis", Advances in Computational Sciences and Technology (ACST), 3, No.1, pp. 9-16, 2010.

[3] Bentley J. The Cost of Recursion. Dr. Dobbs Journal (1989), 111–114.

[4] Great Lakes Symposium on VLSI Proceedings of the 18th ACM Great Lakes symposium on VLSI, Greg Stitt, Jason Villarreal, ACM New York NY, USA. 2008.

[5] M. Frigo, CE Leiserson, Hprokop and S. Ramachandern Cache Olivious Algorithm in Proceedings of 40th Annual IEEE Symposium on Foundations of Computer Science, New York 1999.

[6] S. Chatterjee, A. R. Lebeck, P. K. Patnala, and M. Thottethodi, "Recursive Array Layouts and Fast Matrix Multiplication", IEEE Trans. Parallel Distrib. Systems, 13, 2002,pp. 1105–1123.

[7] N.Park, B.Hong, and V.K. Prasanna, "Tiling Block Data Layout, and Memory Hierarchy Performance", IEEE Trans Parallel Distribution System, 2003.

[8] Malik, S., Martonosi, M., and Li, Y.. "Static Timing Analysis of Embedded Software". In Proceedings of the Design Automation Conference (DAC), pp. 174-162, 1997.

[9] Giusto, P., Martin, G., and Harcourt, E. Reliable Estimation of Execution Time of Embedded Software. In Proceedings of the Conference on Design, Automation and Test in Europe (DATE), pp. 580-589, 2001.