International Journal of Research in Engineering, Science and Management
Volume-2, Issue-4, April-2019
www.ijresm.com | ISSN (Online): 2581-5792

Compiler Optimization Techniques to Improve the Performance for Large Data

K. S. Shivakumara[1], V. M. Hemanth[2], Akhthar Unnisa[3], A. Parkavi[4]
[1,2,3]*Student, Department of CSE, MSRIT, Bangalore, India*
[4]*Assistant Professor, Department of CSE, MSRIT, Bangalore, India*

***Abstract*: Parallel programming is a computation in which execution is carried out simultaneously and distribution of the data over disk and perform some task on it simultaneously. When we have large volume of data then the processing and analyzing of these data is highly important. The distribution of these data over the disk and processing is the highly challenging task. In this paper we present some of the compiler techniques that shows how the parallel programing is done for large data and how to increase the performance of the computer to work with such a huge data. The data placement is the major task that the disk should be able to store all the data in the memory when the execution starts. In this paper, we present the techniques for storing and accessing the data so that the compiler could be cable of doing tasks accurately without getting interrupted.**

***Keywords*: compiler, parallel programing, loop fusion.**

1. Introduction

In today's world data is the one that got most attention and analyzing this data and performing the computation is the challenging task. In most of the computer the task of handling large data and analyzing is done by the compiler only. This will be overhead for a compiler and will take more time for retrieving the data that it want. The compiler can do a better performance when the burden become less because the complier takes more time for collecting the data than the compilation. In this paper, we present the techniques that are used for the better performance of the compiler when we are working with the large data. With the parallelism and optimization of a compiler we can get the better performance.

As discussed above, nowadays the most emerging task is to deal with data analysis for business improvement. It is simple while working with a less number of data, but the trend changes as the data are large in its number which is too complex in handling them to execute them with less compilation time and acquire a minimum storage in the disk. To accomplish this task we will be implementing the concept of parallel execution of the program. This is not a newly introduced concept but to this field of data analysis, parallel execution is a new technology and best method to fit the task of reducing the execution time and storage usage.

2. Related Work and Discussion

Abelardo López-Lagunas.et.al [1], The data movement in the program should be efficient so that we can get the high performance of the computer. By using the stream descriptors, we provide a memory access pattern for a compiler. In the stream descriptor the data is divided into small pieces and mapping is done between the memory units so that access to this memory should not take overhead by the compiler. By defining the dependencies between the memory address and the data, we can access the data easily by the DMA (Direct Memory Access) by using this stream descriptor. When the input is given to access the memory a AGU (Address Generation Unit) generates a bus address by using the descriptors and the remaining request are put into address queue. The line buffer stores the elements when the request is granted by the bus. The stream buffer returns the data in the order that are requested.

Steve Carr Kathryn S.et.al [2]. To improve the execution time the data in the memory should be organized in a hierarchy so the compiler can easily access it or else it will take more time for searching data itself. With the techniques of loop permutation, loop fusion and distribution we can reduce the access time of the compiler. The adjacent loops can be combined so the compiler get less overhead and can improve its runtime performance.

Renato Ferreira.et.al [3]. Performing local reduction and tiling output space are the advanced techniques that can improve the compiler performance. Dense and sparse characterization of access pattern gives the better performance. In sparse execution different strategies are used for removing the replicated space and converting to tiles to fit in the memory during the execution. The data that is required is brought into a memory i.e. ADR (Active Data Repository) and mapping is done at run time. With these advanced techniques we can extract the required information from the source and selecting the appropriate algorithm for execution can improve the compiler performance.

F. Kuijlman.et.al [4]. The compiler framework is developed and it can find a placement for the data and perform the appropriate functions on the data simultaneously. Data placement should be done in the memory and provide the annotation for the program to access the data while compiling the program. This can be done in two steps i.e. analysis phase and placement selection. In analysis the information of data placement is used to get placement information of each piece of

code. In selection phase all the selected placement data is used with the annotations to convert the program to parallel version. By constructing parse tree, we distribute the data into different levels and with the help of mapping techniques we can link to the memory to access the data.

Manish gupta.et.al [5]. The communication cost between the compiler and memory should be reduced. No compiler provides a good data partition of the data so an automated partitioning of the data can used so it will reduce the programmer overhead for developing the program for efficient partition of data and the compiler could easily get this information for processing. This system performs some task in dealing with this data processing, paraphrase used for parse tree structure, control flow information and data dependencies. The alignment is used for mapping the array dimension to a processor, the execution time and cost is determined by the computational and communication cost estimator.

Jay Patel.et.al [6].To meet the good efficient program elimination redundant optimization and manage resources efficiently in order to meet the efficient code we use some methods information flow analysis, peer hole optimization in information flow analyses tries to discover how information is flow throw the program the information flow analysis is process of collecting details of the variables used and define here the minimizing the transformation of assignment statement in the local block of code during the global optimization is to take the global variables and expression determine globally and removing unnecessary code is called dead code elimination in loop variant the in loop will take more execution time the goal of combined code motion and register is the instruction in less frequently and less frequently accessible blocks

Neeraj Kumar.et.al [7].In complier the grammar is the set of action to do when we combine the both Lex and yac parser it will become high level routine called yylex() when it need to take a token from input the Lex will scan the input through input recognition token when it find token it return the code and value of yylex() in the storage management it contains the collection of objects temporary variable and their lifetime the important goal of the storage management is more economical use of the memory and there accessibility towards the function to individual objects in static memory management the compiler will provide the some fixed set of memory address to the object at the time of the program translations in dynamic memory management we allocate the memory by using heap and stack in stack at the time of procedure call or block entry the activation record of the object are pushed into the stack the disadvantage of heap storage management system when the system allocate the more memory then required at this time the memory will be waste

Ch. Raju.et.al [8],proposed a system to improve the optimization of the code which select the program or some part of the code which gone optimized after selection it perform transformation and it apply some of the techniques which is suitable for the code after apply the algorithm the optimized code will generated and it will give some warning message code Is generated the new code is saved and after that complexity comparison is performed between old program source code and the new generated optimized source code

Enyindah P.et.al [9] to analysis the parsing tree and complier applications in parsing algorithm the parsing define and analysis the text which contain the sequences of the token which determine the structure for the grammar when the structure of the grammar is valid it will generate the abstract syntax tree for the source code in fuzzy parsing we take some part of the source instead of take whole of the source as input to find the efficiency of the parse so perform analysis of the selected code part grammar in compiler design when the input are split into token and design a common grammar to describe the performance language to develop a translation for the program developing these grammar is concur about the ability to find the equivalent grammar the grammar is said to be equivalent grammar if the two grammar if it describe the same language that generate the same sentences of output the compiler design have many applications in network ,OS, embedded system

Aastha Singh.et.al [10],finding some of the code optimized techniques which are machine dependent and machine independent machine dependent means the optimization of the code is not respect to compilers and processor in machine dependent code must consider some of the attributes regarding to the compiler and processer of the system the ANN(artificial neural network) in this popular machine learning algorithm which is capable of pattern recognition the system is connected in the form of network with more of the inter connected neurons to compute the input and output and feeding information in network in this paper the proposed system in front end it takes the code and it generate the features and generate the profiles applying ANN using 4cast_xl compiler apply optimization and calculate the speed up of the code at the backend.

Xiaogang Li.et.al [11]. Data mining algorithm focuses on finding the pattern among the large dataset, but the size of the dataset is larger in size it takes more computational time to complete the task. To overcome this, we apply a technique as a parallel execution of the algorithm. This is shown with the most complex algorithm in data mining as K-means, KNN, and Apriori. Here to do this task we are just overwriting the algorithm in a Java language with an intention that executes in parallel. By doing this the algorithms are being executed faster with better accuracy than executing with original as with handwritten codes and reduce the gap between the compiled and manual code.

Dhruva R.et.al [12]. At present run-time library support to handle irregular access but it is necessary to prove for regular access with is not yet proven. So the author focuses on regular access application and evaluates the performance using techniques as PILLAR and CHAOS/PARTI. In which by using this technique the regular access code using libraries comes closer to the performance of code generated by a compiler.

Abdourahmane SENGHOR.et.al [13]. In this authors discuss the improvement of compiler performance with the help of JOMP's and JAVAR's. But the author faces issues while implementing these two compilers as JOMP is implemented in Java whereas JAVAR is in C language. Working with two different languages for design and proposing a compiler is a difficult job. In the intention of these issues, they decided to convert the JAVAR into java language which can be done by using Lexical analyser. By using this instruction are written in parallel.java which contains files as a parallel loop and multi-way recursive node. By doing these results to give a better or equal performance to the best one between JAVAR and JOMP with the help of parallelizing of matrix sort program.

Jun Cao.et.al [14]. A major goal in this paper is to develop the model with the reference of quantum chemistry expecting to increase the ability to predict the properties of the formulation. It is different as compare to constructing the formulas manually because as we know the chemistry formula consists a lot of algebraic expression and equation where the system should remember all of them and should work with it which is difficult and consumes more storage along with compilation time. As to overcome this issues author made a step to make the system to design the formula with the help of compiler to which the expression and the algebraic equation have been imported. By doing this it helps to reduce the complexity of the code as it was generated by the system and is been executed with high performance and better accuracy.

3. Conclusion

When we are working with the huge amount of data, data placement is very important so the compiler could able to process it without any distraction and with less time. To improve the performance, we have to use some techniques as discussed earlier and if we do both accessing the data and processing simultaneously we can get a better performance with less time. As discussed some techniques like steam descriptor, active data repository, loop fusion, automatic partition of data and the implementation of compiler framework etc. all these techniques can be used for compiler optimization to handle large data and can get a better performance.

References

[1] A. Lopez-Lagunas and S. M. Chai, "Compiler manipulation of stream descriptors for data access optimization," *2006 International Conference on Parallel Processing Workshops (ICPPW'06)*, Columbus, OH, 2006, pp. 8 pp.-344.
[2] Steve Carr Kathryn S. McKinley Chau-Wen Tseng, "Compiler Optimizations for Improving Data Locality", 1994.
[3] Renato Ferreira, Gagan Agrawal, Joel Salt Z. "Advanced Compiler and Runtime Support for Data Intensive Application."
[4] F. Kuijlman, H.J. Sips, C. van Reeuwijk, and W.J.A. Denissen. A Unified Compiler Framework for Work and Data Placement.
[5] Manish Gupta, Prithviraj Banerjee. Paradigm: A compiler for automatic data distribution on multicomputer, 1993.
[6] Jay Patel, Mahesh Panchal, "Code Optimization in Compilers using ANN," vol. 3, no. 5, pp. 557-561, May 2014.
[7] Neeraj Kumar, Saroj Hiranwal, "Improving Code Efficiency by Code Optimizing Techniques," in International Research Journal of Engineering and Technology, vol. 3, no. 4, pp. 362-366, April 2016.
[8] Ch. Raju, Thirupathi Marupaka, Arvind Tudigani, "Analysis of Parsing Techniques &Survey on Compiler Applications," in International Journal of Computer Science and Mobile Computing, vol. 2, no. 10, pp. 115-125, October 2013.
[9] Enyindah P., Okon E. Uko, "The New Trends in Compiler Analysis and Optimizations," in International Journal of Emerging Trends & Technology in Computer Science, vol. 46, no. 2, May 2017.
[10] Aastha Singh, Sonam Sinha, Archana Priyadarshi, "Compiler Construction," in International Journal of Scientific and Research Publications, vol. 3, no. 4, pp. 1-6, April 2013
[11] Xiaogang Li, Ruoming Jin and G. Agrawal, "A compilation framework for distributed memory parallelization of data mining algorithms," *Proceedings International Parallel and Distributed Processing Symposium*, Nice, France, 2003, pp. 8.
[12] D. R. Chakrabarti, P. Banerjee and A. Lain, "Evaluation of compiler and runtime library approaches for supporting parallel regular applications," *Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing*, Orlando, FL, USA, 1998, pp. 74-79.
[13] A. Senghor and K. Konate, "A Java Hybrid Compiler for Shared Memory Parallel Programming," *2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies*, Beijing, 2012, pp. 131-136.
[14] J. Cao, A. Goyal, S. P. Midkiff and J. M. Caruthers, "An Optimizing Compiler for Parallel Chemistry Simulations," *2007 IEEE International Parallel and Distributed Processing Symposium*, Rome, 2007, pp. 1-10.