International Journal of Psychosocial Rehabilitation, Vol. 24, Issue 01, 2020

ISSN: 1475-7192

Optimizing compilers through parallel processors and memory performance observing as combined approach

[1]Dr Sandeep Kulkarni, [2]Dr. K P Yadav

Abstract

Phases of compilers for tokenizing the input using lexical analysis and regular expression. Abstract syntax tree in the form of parsing.. Abstract might go to error state if it has more than one input. Hence automata uses the phases of compiler with the help of algorithms which is mathematical. Processing of source code which is human readable to machine readable code which is translated at the time of runtime. While translating code should be readable which is done with the help of compilers and interpreters. Therefore it requires less memory because there is no specific code for platform. Taking string input as symbols changes state as per instructions is called finite automata. It uses regular expressions. It recognizes regular expressions. After processing all the state according to instruction it reaches final state and it is known as accepted state. If it is self compiling kind of compiler in any programming language is known as bootstrapping. Using very little part of language we could generate bootstrap compiler is many programming language. For example languages like Pascal, Haskell, C, OCaml, Java etc uses bootstrap compiler. Features containing discrete properties in mathematics like calculus, algebra that includes set theory , matrix and so on. Before runtime occurs in programming language some interpretation happens in some languages, that is translation occurs . Interpreted code can be executed without the help of machine code. It can run in many operating systems. Optimization is good , because they are interpreted as soon as they are interpreted. To one of the problem is inefficiency of compiled programming language. Small talker is of the programming language it is known for most productive for many years. Language complexity is considered seriously. Now a days we using Swift for reducing the complexity of the language.

Keywords: *Memory performance, Parallelism, Matrix, Permutation, Scope Management, Unified.*

I. Introduction

We should consider discrete mathematics which indirectly solves problem in efficiency in any programming language. The supporting mathematics such as set theory, number theory , counting theory are

[1] *Post-doctoral scholor, Computer Science Department, Sangum University, Bilwara ,Rajasthan, INDIA-311011*

[2] *Vice Chancellor, Computer Science Department, Sangum University, Bilwara ,Rajasthan, INDIA-311011*

Received: 09 Nov 2019 | Revised: 18 Dec 2019 | Accepted: 27 Jan 2020

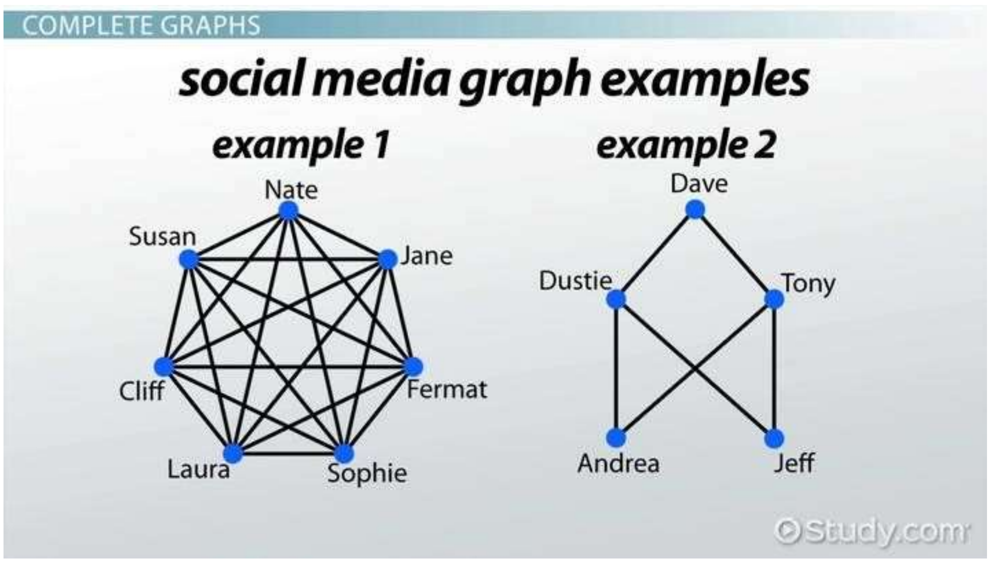
some of the things which is supported by computer science . With the help of boolean algebra digital circuit can be designed. For development of software mathematics helps in sets,trees for data structure. Cryptography uses number theory. Artificial intelligence uses logical mathematics. Induction proof helps in theory of computation and well as compiler design. For hardware design theory of probability is used. To develop our algorithm we need idea for it, we need to see random file cannot be compressed programs. It includes good understanding of binary trees, linked list. Regular expressions knowledge is very important used in parallelization optimizations in computers and regular expression is also required. By the help of discrete mathematics we can solve the problems in algorithms , computability. For analysing the algorithms set theory, combination and permutation is very important.

II. Literature Survey

Symbol table structure is kept in semantic analyser. Order of searching a string is current symbol ,parent symbol table, global symbol table. Perl and Python has best lexical and syntax analyser , they producing good quality product.Writing the intermediate code such as syntax tree and generating a code. Structure is arranged in such a manner that even it is reverse manner by traversal it becomes correct.

Graphs: Graph consists of nodes and vertex. Graphs are used in real life situations such as social networking, suppose user is considered as a node and his activates is considered as edges between the node. To find the roads in particular area, source and destination addresses are considered as nodes, and lines represents edges using graph theory we try to find the possible various approaches to reach the destination in google maps. Using graph theory various pop up or recommendations for buying or selling products which guides us to idea about the products in ecommerce. In terms of resource allocation process with the help of vertices as resources we can use graph theory in operating system.

For pseudocode which are having proof which are studied in discrete maths. Sets are used in data structures, similarly trees are used in arrays.



Discrete mathematics can be used in computer graphics, deep learning , recursion, code optimization.

For solving 15 puzzle arranging the square box . Its behaviour is studied by mathematics. With the help of permutations and combination, with solvable move the arrangement is made and moves to next step.

Combinatory is used in statistics and machine learning. Such as dice game that is predicting the fractions of square for example it is used in loops in the python language. Another example we can take is each one is having one each phone number against them , is that algorithm is running fine this can be addressed by combinatory. Random variables, suppose we are not caring weather its negative or not, but we interested quantitative outcome. Randomness picking the people among the crowed. This is known as random variables.

Software which analysis distribution of water , to which area water has been distributed evenly or not, for which area we need more water. This can be used in traffic system it can road or air. This can be used in hospitals and schools also. This is known as flows and matching algorithms. Algorithm which can send secret key form one destination to other. Its application we can find in payment in internet this is done with the help of number theory. Euclid algorithm is the basics for cryptography. Chinese remainder theorem using this algorithm suppose we want to secure from secure from attacks like RSA we have send code in secret manner. It was used in Word war 1 and 2 because there everybody was spying on each one. This kind of techniques is used frequently. Optimization in compiler design is using less CPU or memory in it with high delivery speed which helps efficiently in low level program, if we replace the code the output of code should remain same and it should be done in minimal resource available. Machine dependent optimization involves references , especially memory references optimizes with the help of mathematical formula .

III. Proposed System

Optimizing compilers using algorithms on parallelism on memory locations. When the string is passed to the compilers for many inputs scalable performance becomes an issue. So distribution among the memory is good technique to implement. Solving both parallelism and memory distribution in single architecture is most difficult and challenging one. Some operations it should perform such as nested loops, multi dimensional arrays should be distributed for each processors, communication between processors for messaging, cache functionality should be implemented. For examples Convex Exemplar, Intel Paragon, SGI Power Challenge XL.

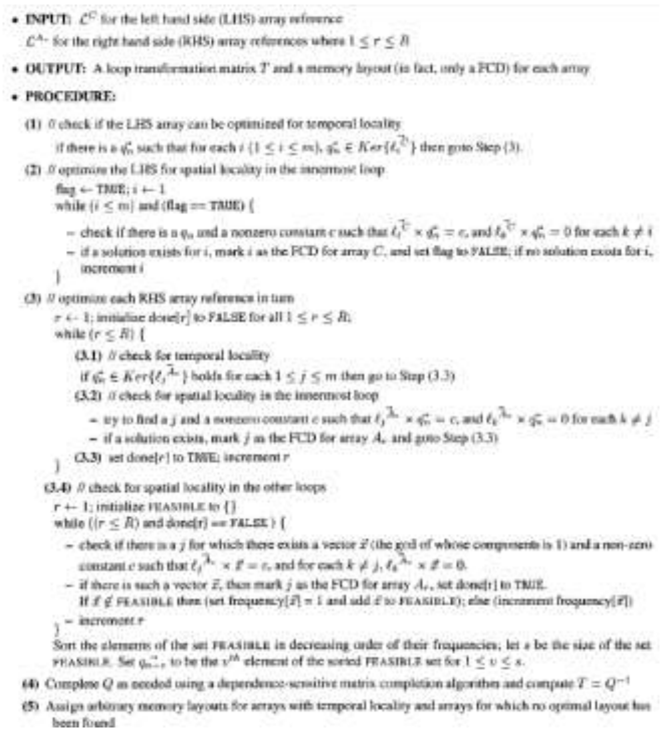
In previous years processors speed has been improving along with memory and caching. Interrelated issues still needs to be addressed block transfers and loop system this all issues should be under one framework. Some of the best programming languages still have memory access problems, loops should be the solutions to it. It should support multidimensional arrays. Even if processor changes it should be supporting. As memory and parallelism co ordinate with each other so synchronization is very important. If it comes to iteration , we can label inner and outer iteration loops, it can be bounded with limits. It can be subscripted with an array. So we can say m dimensional arrays in a dimension loop of n.

We mainly concentrate on multidimensional arrays with some constants. Suppose if we see as all the matrixes are stored in some order some block are stored for future research purposes. Fastest changing dimension which is related to inner circle on array. If we take an array it is distributed across all dimension is known as Distributed dimension, which we tag with fastest changing dimension with respect to column wise. Loop optimization is very important , can be done by techniques such as loop fusion, permutation and distribution. Space transformation is directly related to speedup the process.

In mathematical aspects vector spaces are defined to matrix algorithms used for inner circles of matrixes. Performance is graded in kernels in operating system. Reusing the vectors means reusing the matrices , that is reusing the vectors , it reuses the inner most loops . Looping permutations with optimizing parallelism can be done. We try to rectify the spatial locality with memory considering different arrays.

IV. Enchancing cache locality

It better to access data in cache than in memory because of performance. Loop can be in cache and arrays can be in configured respectively. Algorithm needs to be implemented single and multiprocessor. This is done by the technique outermost loop is sent to processors and distributed arrays is sent to all processors to optimize



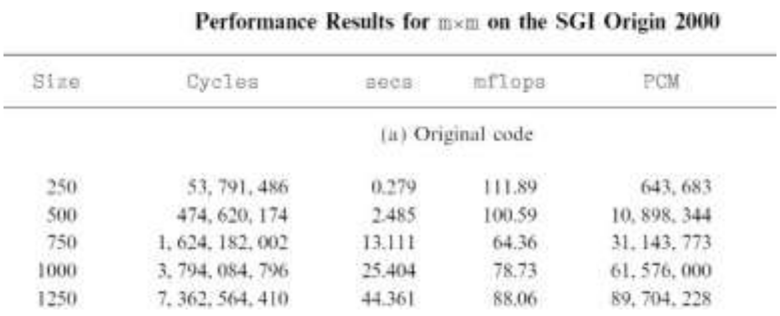
```
• INPUT: $\mathcal{L}^{C}$ for the left hand side (LHS) array reference
  $\mathcal{L}^{A_r}$ for the right hand side (RHS) array references where $1 \le r \le R$
• OUTPUT: A loop transformation matrix $T$ and a memory layout (in fact, only a FCD) for each array
• PROCEDURE:
  (1) // check if the LHS array can be optimized for temporal locality
      if there is a $\vec{q}_n$ such that for each $i$ ($1 \le i \le m$), $\vec{q}_n \in Ker\{\vec{\ell}_i^{\,C}\}$ then goto Step (3).
  (2) // optimize the LHS for spatial locality in the innermost loop
      flag ← TRUE; i ← 1
      while ($i \le m$) and (flag == TRUE) {
        – check if there is a $\vec{q}_n$ and a nonzero constant $c$ such that $\vec{\ell}_i^{\,C} \times \vec{q}_n = c$, and $\vec{\ell}_k^{\,C} \times \vec{q}_n = 0$ for each $k \ne i$
        – if a solution exists for $i$, mark $i$ as the FCD for array $C$, and set flag to FALSE; if no solution exists for $i$,
          increment $i$
      }
  (3) // optimize each RHS array reference in turn
      $r$ ← 1; initialize done[$r$] to FALSE for all $1 \le r \le R$;
      while ($r \le R$) {
        (3.1) // check for temporal locality
         if $\vec{q}_n \in Ker\{\vec{\ell}_j^{\,A_r}\}$ holds for each $1 \le j \le m$ then go to Step (3.3)
        (3.2) // check for spatial locality in the innermost loop
          – try to find a $j$ and a nonzero constant $c$ such that $\vec{\ell}_j^{\,A_r} \times \vec{q}_n = c$, and $\vec{\ell}_k^{\,A_r} \times \vec{q}_n = 0$ for each $k \ne j$
          – if a solution exists, mark $j$ as the FCD for array $A_r$ and goto Step (3.3)
        (3.3) set done[$r$] to TRUE; increment $r$
      }
    (3.4) // check for spatial locality in the other loops
      r ← 1; initialize FEASIBLE to {}
      while (($r \le R$) and done[r] == FALSE ) {
        – check if there is a $j$ for which there exists a vector $\vec{x}$ (the gcd of whose components is 1) and a non-zero
          constant $c$ such that $\vec{\ell}_j^{\,A_r} \times \vec{x} = c$, and for each $k \ne j$, $\vec{\ell}_k^{\,A_r} \times \vec{x} = 0$.
        – if there is such a vector $\vec{x}$, then mark $j$ as the FCD for array $A_r$, set done[r] to TRUE.
          If $\vec{x} \notin$ FEASIBLE then (set frequency[$\vec{x}$] = 1 and add $\vec{x}$ to FEASIBLE); else (increment frequency[$\vec{x}$])
        – increment $r$
      }
      Sort the elements of the set FEASIBLE in decreasing order of their frequencies; let $s$ be the size of the set
      FEASIBLE. Set $\vec{q}_{n-v}$ to be the $v^{th}$ element of the sorted FEASIBLE set for $1 \le v \le s$.
  (4) Complete $Q$ as needed using a dependence-sensitive matrix completion algorithm and compute $T = Q^{-1}$
  (5) Assign arbitrary memory layouts for arrays with temporal locality and arrays for which no optimal layout has
      been found
```

V. Improvement in future

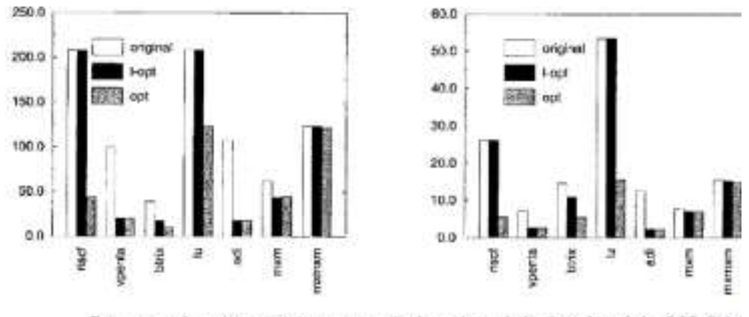
To improve the cache we can use nonsingular transformation matrices. This can also be done with Fourier-Motzkin elimination method. Restructure the code is more focused. Anderson method for distributed- shared- memory by shared memory space. This involves parallelism decisions which is made by earlier SUIF compiler , so it is not directly related to locality in transformation matrix. But it is not clear interms of uniprocessors, distributed and shared memory in this approach. Stride vector is made for optimization and for mapping vectors which is helpful for loop transformation. Finally equality got is TTv=LTx. That is L, the array access matrix. Now find T, iteration space transformation matrix; x, mapping vector which can be assumed m!.

VI. Results

This Unified Algorithm is presented by greedy heuristic which handles arrays which also helps to take decisions during optimization. Here how it works, outermost loop runs into processors, arrays are distributed throughout the memory. Interprocess communication is useful for optimization with the help of vectorization. Iteration space transformation for every array is required for cache optimization. As experimental results are considering matrix , LU uses 3 matrices and classical ijk multiplication of matrix routine. Here are some of the results experimentally on the shared memory machine which is SGI and IBM SP-2 with 32 K bye L1 data cache and 4 instructions per cycle.


Performance Results for m×m on the SGI Origin 2000

Size	Cycles	secs	mflops	PCM
		(a) Original code		
250	53, 791, 486	0.279	111.89	643, 683
500	474, 620, 174	2.485	100.59	10, 898, 344
750	1, 624, 182, 002	13.111	64.36	31, 143, 773
1000	3, 794, 084, 796	25.404	78.73	61, 576, 000
1250	7, 362, 564, 410	44.361	88.06	89, 704, 228



Primary cache misses (PCM) on (a) a single node and (b) 8 nodes of the SGI Origi
values are in millions. These results are obtained using the performance counters in the Origin.

VII. Conclusion

Running parallel both the things in terms of loop optimization is very difficult, might be because hardware configuration are not same although we consider architectures in parallel will make profit in optimization of compiler as well as improvement in locality. We should handle parallelism should be in one must be working for all parallel compilers. As with the help of looping of matrix and determines works in memory array for loop parallelism. We have experimented on SGI Origin 2000 and IBM SP-2. Data distributions along one and diagonal-wise distributions can work in single and multiple cases. We tried providing evidence and errors found in different systems it can be solved by unified techniques.

References

[1] M. O'Boyle and P. Knijnenburg, Non-singular data transformations: definition, validity, applica- tions, in ``Proc. 6th Workshop on Compilers for Parallel Computers, ,"pp. 287_297 - 2003.

[2] D. Palermo and P. Banerjee, Automatic selection of dynamic data partitioning schemes for distributed-memory multicomputers, in ``Proc. 8th Workshop on Languages and Compilers for Parallel Computing, Columbus, ," pp. 392_406 -2003.

[3] C. Polychronopoulos, M. B. Girkar, M. R. Haghighat, C. L. Lee, B. P Leung, and D. A. Schouten, Parafrase-2: an environment for parallelizing, partitioning, synchronizing, and scheduling programs on multiprocessors, in ``Proc. the International Conference on Parallel Processing, St. Charles, IL,," pp. 39_48 - 2000.

[4] J. Ramanujam, Non-unimodular transformations of nested loops, in ``Proc. Supercomputing 92, Minneapolis, MN, ' pp. 214_223 - 2000.

[5] J. Ramanujam and A. Narayan, Integrating data distribution and loop transformations for dis- tributed memory machines, in ``Proc. 7th SIAM Conference on Parallel Processing for Scientific Computing - 2015.

[6] J. Ramanujam and A. Narayan, Automatic data mapping and program transformations, in ``Proc. Workshop on Automatic Data Layout and Performance Prediction, Houston, TX,. - 2018"

[7] V. Sarkar, G. R. Gao, and S. Han, Locality analysis for distributed shared-memory multiprocessors, in ``Proc. the Ninth International Workshop on Languages and Compilers for Parallel Computing, Santa Clara, CA" -2019.