Available online at www.sciencedirect.com

ScienceDirect

AASRI Procedia 5 (2013) 189 – 193

AASRI Procedia

www.elsevier.com/locate/procedia

2013 AASRI Conference on Parallel and Distributed Computing Systems

Matrix Multiplication using r-Train Data Structure

Bashir Alam*

Department of Computer Engineering, Jamia Millia Islamia New Delhi, 110025, India.

Abstract

A new dynamic data structure has been proposed recently in 2011. There are several algorithms for matrix multiplication. But none of them has used r-train data structure for storing and multiplying the matrices. In this paper algorithm for matrix multiplication using r-train for parallel machine has been proposed.

Keywords: R-Train, SIMD, Parallel Algorithm

1. Introduction

There are several model of parallel systems[2]. In SIMD model single instruction is executed on several processing elements simultaneously. In MIMD model several instructions are executed on several data. In this work SIMD model is used. There are a number of useful data structures developed by different authors for various purposes with different philosophy for different kind of requirements. Few important non-linear data structures are listed here. Graphs are one of the pervasive data structures in computer science. Sleator and Tarjan in [5,6] ntroduced The data structure Splay Trees and Dynamic Trees. hash tables were introduced by H.P.Luhn. A very useful data structure AVL Trees were introduced by G.M. Velskii and Landis [8] . B-Trees were introduced by Bayer and McCreight [10]. Bayer invented the data structure Red-black trees [11] .

* Corresponding author. Tel.: +91- 9810650497;
*E-mail address:*babashiralam@gmail.com (Bashir Alam)

doi:10.1016/j.aasri.2013.10.077

Seidel and Aragon proposed the Treaps[7]. A treap is a binary search tree in which each node has both a key and a priority: nodes are ordered in an inorder fashion by their keys (as in a standard binary search tree) and are heap-ordered by their priorities (so that the each parent has a higher priority than its children). Binary Search Tree seems to be independently discovered by a number of people as mentioned by Knuth in [4]. a dynamic array algorithm called Tiered Vectors for order preserving insertions or deletions from the middle of the array was presented by Goodrich [9]. A hashed array tree (HAT) is a dynamic array algorithm invented by Edward Sitarski [3] . A dynamic array is a random access, variable-size list data structure that allows elements to be added or removed. A dynamic array is not the same thing as a dynamically-allocated array, which is a fixed-size array whose size is fixed when the array is allocated, although a dynamic array may use such a fixed-size array as a back end. The simplest dynamic array is constructed by allocating a fixed-size array and then dividing it into two parts: the first stores the elements of the dynamic array and the second is reserved, or unused. We can then add or remove elements at the end of the dynamic array in constant time by using the reserved space, until this space is completely consumed. The number of elements used by the dynamic array contents is its logical size or size, while the size of the underlying array is called the dynamic array's capacity, which is the maximum possible logical size. In applications where the logical size is bounded, this data structure suffices. Resizing the underlying array is an expensive operation, typically involving copying the entire contents of the array. The dynamic array has performance similar to an array, with the addition of new operations to add and remove elements from the end. Compared to linked lists, dynamic arrays have faster indexing (constant time versus linear time) and typically faster iteration due to improved locality of reference. Even, in a highly-fragmented memory region, it may be expensive or impossible to find contiguous space for a large dynamic array, whereas linked lists do not require the whole data structure to be stored contiguously.

There are some engineering situations that require more than the simple rudimentary data structures like arrays, linked lists, hash tables, binary search trees, dictionary, etc. In present day database of huge size, in many situations, require a dash of creativity of a new or better equipped/powerful data structure of rudimentary in nature. There are some situations where we need to create an entirely new type of data structure of generalized nature, and creating such a new but simple data structure is not always a straightforward task. The objective of this research work is to introduce a new and a robust kind of dynamic data structure which encapsulates the merits of the arrays and of the linked lists, and at the same time inherits a reduced amount of their characteristic demerits. By the nature ‘dynamic’ we mean that it changes over time, it is not like static. Apparently it may seem that this new data structure is of hybrid nature, hybrid of linked list and array; but construction wise (by virtue of its architecture) it is more. The new data structure is called “r-train”. Using the data structure ‘r-train’, a large number of data elements can be stored and basic operations like insertion/deletion/search can be executed very easily, in particular parallel programming can be done very easily, in many situations, with improved time complexities and the performance. The term “train” has been here coined from the usual railways transportation systems because of a lot of similarities in nature of the object r-train with the actual train of railways; and analogously we use the terms coach, passenger, availability of seats, etc. in our way of discussion in this work. The notion of r-trains is not a direct generalization of that of the linked lists. It is not just linked list of arrays as it looks so, but something more. However, every linked list is an example of 1-train (i.e. r-train with r = 1). The main aim of introducing the data structure ‘r-train’ is how to store a large array in an alternative way. One basic demerit of r-train inherited from the properties of array is that new data can not be inserted in between two existing consecutive data, because we preserve the unique index of each and every data which remains unaltered althrough the time. However, the data structure r-train obviously dominates the data structures array and linked list in case of dealing with huge number of data. A new data structure r-Train proposed by R. Biswas[1] is neither a dynamic array [9] nor a HAT [3]. It is briefly describe below.

A r-train is basically a linked list of tagged coaches. This linked list is called the 'pilot' of the r-train. But the implementation of this pilot in memory may also be done using the data structure array in some cases which will be explained later on in this paper. The pilot of a train is thus, in general, a linked list. But, if we are sure that there will be no requirement of any extra coach in future, then it is better to implement pilot as an array. Details about these operations is explained at subsequent sections. The number of coaches in a r-train is called the 'length' of the r-train which may increase or decrease time to time. A 'r-train' may also be called by the name 'train' if there is no confusion. A r-train T of length l (>0) will be denoted by the following notation

$$T = < (C1, sC1), (C2, sC2), (C3, sC3), \ldots\ldots, (Cl, sCl) >,$$

where the coach Ci is (Ai,ei) with Ai being a larray of length r, ei being the address of the next coach Ci+1 (or an invalid address in case Ci is the last coach) and sCi being the status of the coach Ci, for i = 1, 2, 3, …., l.

For a r-train, START is the address of the pilot (viewing its implementation in memory as a linked list). Thus START points at the data C1 in memory. The length l of the pilot could be any natural number, but the larrays of the TCs are each of fixed length r which store data elements (including $\mathcal{E}$ elements) of common datatype where r is a natural number. Thus, by name, 1-train, 60-train, 75-train, 100-train, etc. are few instances of r-train, where the term 0-train is undefined.

The notion of the data structure r-train is a new but very simple data structure, a type of 2-tier data structure, having very convenient methods of executing various fundamental operations like insertion, deletion, searching etc., for huge number of data, in particular for parallel computing. The most important characteristic of the data structure r-train is that it is likely that it can store x number of elements of identical datatype even if an array of size x of same elements can not be stored in memory, at some moment of time. And also, the data can be well accessed by using the indices.

In a r-train, the coach names C1, C2, C3, ……. , Cl do also mean the addresses (pointers) serially numbered like in case of arrays, for example : the array z = (5, 9, 3, 2) can be easily accessed by calling its name z only.

Status of a coach reflects the availability of a seat (i.e. whether a data element can be stored in this coach now or not). Status may vary with time. Each coach of a r-train can accommodate exactly r number of data elements serially numbered, each data element being called a passenger. Thus each coach of a r-train points at a larray of r number of passengers. By definition, the data ei is not a passenger for any i.

Consider a coach Ci = (Ai, ei). In the larray Ai = <ei1, ei2, ei3, …….....,ei(r-1), eir>, the data element eij is called the "j th passenger" or "j th data element" for j = 1, 2, 3, 4,..., r. Thus we can view a r-train as a linked list of larrays. Starting from any coach, one can visit the inside of all the next coaches but not any of the previous coaches. The r-train is a forward linear object, not a type of circular one. It is neither a dynamic array nor a HAT. It has an added advantage over HAT that starting from one data element, all the next data elements can be read well without referring to any hash table or the pilot.

2. Representation of matrix using r-Train data structure

An NXN matrix A can be represented using N-train using two ways

1. Row Major N-train Representation
2. Column major N-train Representation

In Row major N-Train representation an NXN matrix A can be represented by N-train data structure Xa as given below.

$Xa = <A_0, A_1, \ldots\ldots\ldots\ldots\ldots\ldots A_{n-1}>$

Where $A_0=<a_{00}, a_{01}, a_{02}\ldots\ldots\ldots a_{0n-1}, e_1>$

$A_1=<a_{10},a_{11},a_{12}\ldots\ldots\ldots\ldots\ldots a_{1n-1}>$

$A_2=<a_{20}, a_{21},\ldots\ldots\ldots\ldots\ldots\ldots a_{2n-1}>$

……

……

$A_{n-1}=<a_{n-10}, a_{n-1\,1},\ldots\ldots\ldots\ldots\ldots\ldots a_{n-1}a_{n-1}>$

Here A_0, $A_1\ldots\ldots\ldots\ldots\ldots\ldots A_{n-1}$ are coaches , a_{00}, a_{01}, a_{20},$a_{22}\ldots\ldots$ are passenger of coaches and e_1 , e_2 ……e_{n-1} are address of coach A1, A2…….An-1 and A1.0 , A1.1……A1.n-1 represents a10, a11, a12……a1n-1 respectively.

In Column major N-Train representation an NXN matrix A can be represented by N-train data structure Xa as given below.

$Xa=<A_0, A_1,\ldots\ldots\ldots\ldots\ldots\ldots A_{n-1}>$

Where $A0=<a_{00}, a_{10}, a_{20}\ldots\ldots\ldots a_{n-10}, e_1>$

$A_1=<a_{01},a_{11},a_{21}\ldots\ldots\ldots\ldots a_{n-11}>$

$A2=<a_{02}, a_{12},\ldots\ldots\ldots\ldots\ldots a_{n-1\,2}>$

..

..

$A_{n-1}=<a_{0n-1}, a_{1\,n-1},\ldots\ldots\ldots\ldots\ldots\ldots a_{n-1}a_{n-1}>$

Here A_0,$A_1\ldots\ldots\ldots\ldots\ldots\ldots A_{n-1}$ are coaches , a_{00}, a_{10}, a_{02},$a_{12}\ldots\ldots$ are passenger of coaches and e_1 , e_2 ……e_{n-1} are address of coach A_1, $A_2\ldots\ldots\ldots\ldots\ldots\ldots A_{n-1}$. Further $A_{0.0}$, $A_{0.1}\ldots\ldots\ldots\ldots A_{0.n-1}$ represents a_{00}, a_{10}, $a_{20}\ldots\ldots\ldots a_{n-10}$ respectively.

3. Proposed Matrix Multiplication algorithm

Here is the proposed algorithm of matrix multiplication using r-train data structure on a parallel system having M processors. Each processor has a unique ID for identification. If a for loop is preceded by Par then the body of for loop has to be executed in parallel by all the processors. A, B, C are three NXN matrices. Aim of this work is to compute C=AXB using r-train data structure. Proposed Algorithm is given below.

1. Store matrix A as row major N-train data structure TA as discussed in previous section.
2. Store matrix B as column major N-train data structure TB as discussed in previous section.
3. Par for(id=0;id<M;id++)
   {
4. Read $TA_{\text{id mod N}}$
5. Read $TB_{\text{floor (id/N)}}$
6. temp=0;
7. For(i=0;i<N;i++)
8. temp=temp+ $TA_{\text{id mod N .i}}$ * $TB_{\text{floor (id/N).i}}$
9. C id/N.id mod N =temp
   }

Product Any two matrices of NXN order may be obtained by running this algorithm and verification of that is obvious. Complexity analysis of the Algorithm is given below.

Step 1 and Step 2 can be done in O (N) time. Steps 3, 4, 5, 6, 8, 9 can be done in O(1) time . Step 8 is repeated O(N) time because of loop of step 7. So applying the sum and product rule of computing complexity the complexity of the algorithm is found to be O(N) if $M=N^2$.

4. Conclusion

In this work a novel matrix storage scheme using N-train data structure and a novel algorithm of matrix multiplication of two NXN matrices using n-train data structure are given and their computational complexity has also been computed.

References

[1]Ranjit Biswas.r-Train (TRAIN): A new Flexible Dynamic Data Structure. INFORMATION : An International Journal (Japan) 2011;14 (4):1231-1246.
[2]Kai Hwang, Faye A. Briggs. Computer Architecture and Parallel processing. International edition .MCGraw Hill 1985; 32-49.
[3]Sitarski, Edward Algorithm Alley. HATs: Hashed array trees. Dr. Dobb's Journal 1996; 21 (11). http://www. ddj.com/architect/184409965?pgno=5
[4]Donald E. Knuth. The Art of Computer Programming. Addison Wesley1968;Vol - I, II, III.
[5]D.D.Sleator and R.E.Tarjan.A data structure for dynamic trees.Journal of Computer & System Sciences 1983;26(3): 362-391.
[6]D.D.Sleator and R.E.Tarjan.Self-adjusting binary search trees. Journal of the ACM 1985;32(3): 652-686.
[7]R.Seidel and C.R.Aragon. Randomized search trees.Algorithmica 1996;16: 464-497.
[8]G.M.Velskii & E.M. Landis.An algorithm for the organization of information. Soviet Mathemtics Doklady 1962;3:1259-1263.
[9]Goodrich, Michael T.; Kloss II, John G. ().Tiered Vectors: Efficient Dynamic Arrays for Rank-Based Sequences. Workshop on Algorithms and Data Structures 1999;1663: 205–216.
[10]R.Bayer and E.M.McCreight. Organization and maintenance of large ordered indexes.Acta Informatica 1972;1(3): 173-189.
[11]R.Bayer, Symmetric binary B-trees.Data Structures and maintenance algorithms.Acta Informatica 1972;1: 290-306.