Qualitative Parametric Comparison of Matrix Multiplication algorithms in Parallel and Distributed Computing

Satya Prakash Singh,  Birla Institute Of Technology,(Noida Campus), Mesra Ranchi.  E-mail id: spsinghbit@yahoo.co.in

Abstract
Advance in computer network technologies have led to increased interest in the use of large scale parallel and distributed systems. One of the biggest issues in such systems is development of effective algorithms for distributed of the array elements to multiple processor based on various distributed scheme to achieve goal(s) such as minimize communication delays, minimize execution time maximize resource utilization and maximizing throughput.

The objective of this paper is to identify qualitative parameters for comparison of traditional matrix multiplication algorithms and systolic array algorithms.

Key words: Multidimensional Array, Data Distribution, Parallel Algorithm, Matrix Multiplication, Shared Memory.

I. INTRODUCTION TO PARALLEL AND DISTRIBUTED COMPUTING

Parallel Matrix multiplication is one of the most fundamental and important problem in Science and Technology, such as molecular dynamics[6], finite- element methods[7], atmosphere and ocean Sciences[8], etc.. Many methods have been proposed to implement these array operations efficiently. Algorithm allows the broadcast operations used in the preceding algorithm to be replaced with regular nearest-neighbor (“Systolic”) communications. However, data must be distributed among tasks in the different fashion. In this paper we comparison with traditional matrix multiplication (TMR)[9] to represent high dimensional arrays. This scheme has two drawbacks [9] for higher dimensional array operations, first costs of index computation of array operations increase as the dimension increased. Second, the cache miss rate for array operations increases as the dimensions increases due to more cache lines accessed.
II. PARALLEL MATRIX MULTIPLICATION ALGORITHMS

We start by examining algorithms for various distributions of $A$, $B$, and $C$. We first consider a one-dimensional, column wise decomposition in which each task encapsulates corresponding columns from $A$, $B$, and $C$. One parallel algorithm makes each task responsible for all computation associated with its $C_{ij}$. As shown in Figure, each task requires all of matrix $A$ in order to compute its $C_{ij}$, $N^2 / P$ data are required from each of $P-1$ other tasks, giving the following per-processor communication cost:

$$T_{\text{matrix-matrix} \text{ Id}} = (P - 1) \left( t_s + t_w \frac{N^2}{P} \right)$$

$$\approx t_s P + t_w N^2 / P.$$

Note that as each task performs $O(N^3)$ computation, if $N \ll P$, then the algorithm will have to transfer roughly one word of data for each multiplication and addition performed. Hence, the algorithm can be expected to be efficient only when $N$ is much larger than $P$ or the cost of computation is much larger than $t_w$.

Next, we consider a two-dimensional decomposition of $A$, $B$, and $C$. As in the one-dimensional algorithm, we assume that a task encapsulates corresponding elements of $A$, $B$, and $C$ and that each task is responsible for all computation associated with its $C_{ij}$. The computation of a single element $C_{ij}$ requires an entire row $A_{i*}$ and column $B_{*j}$ of $A$ and $B$, respectively. Hence, as shown in Figure, the computation performed within a single task requires the $A$ and $B$ submatrices allocated to tasks in the same row and column, $O(N^2 / \sqrt{P})$ data, considerably less than in the one-dimensional algorithm.
Figure: Matrix-matrix multiplication algorithm based on two-dimensional decompositions. Each step involves three stages: (a) an A submatrix is broadcast to other tasks in the same row; (b) local computation is performed; and (c) the B submatrix is rotated upwards within each column.

To complete the second parallel algorithm, we need to design a strategy for communicating the submatrices between tasks. One approach is for each task to execute

\[
\sqrt{P} - 1 \quad \text{for } j=0 \text{ to } \sqrt{P} - 1 \text{ in each row } i, \text{ the } \text{th task broadcasts } \]

\[
A' = A_{\text{local}}
\]

\[
B' = B_{\text{local}}
\]

accumulate \( A' \cdot B' \)
Each of the steps in this algorithm involves a broadcast to \( \sqrt{P} - 1 \) tasks (for \( A' \)) and a nearest-neighbor communication (for \( B' \)). Both communications involve \( \log \sqrt{P} \) steps using a tree structure, the per-processor communication cost is

\[
T_{\text{matrix-matrix 2d}} = (\sqrt{P} - 1) \left( \frac{\log P}{2} + 1 \right) \left( t_s + t_u \frac{N^2}{P} \right)
\]

\[
\approx t_s \frac{\sqrt{P} \log P}{2} + t_u N^2 \frac{\log P}{2 \sqrt{P}}.
\]

Notice that because every task in each row must serve as the root of a broadcast tree, the total communication structure required for this algorithm combines a hypercube (butterfly) structure within each row of the two-dimensional task mesh and a ring within each column.

![Diagram](image)

**Figure:** Reorganizing from a one-dimensional to a one-dimensional decomposition of a square matrix when \( P=16 \). Shading indicates one set of four tasks that must exchange data during the reorganization.

[A] Systolic Array

Systolic array [12] has been proposed over a decade. It is a special purpose parallel device composed of several processing elements (PE’s) who interconnection have the properties of regularity and locality.
Systolic arrays are arrays of DPUs which are connected to a small number of nearest neighbour DPUs in a mesh-like topology. DPUs perform a sequence of operations on data that flows between them. Because the traditional systolic array synthesis methods have been practiced by algebraic algorithms, only uniform arrays with only linear pipes can be obtained, so that the architectures are the same in all DPUs. The consequence is, that only applications with regular data dependencies can be implemented on classical systolic arrays. Like SIMD machines, clocked systolic arrays compute in "lock-step" with each processor undertaking alternate compute | communicate phases. But systolic arrays with asynchronous handshake between DPUs are called wavefront arrays. One well-known systolic array is CMU's iWarp processor, which has been manufactured by Intel. An iWarp system has a linear array processors connected by data buses going in both directions.

An application Example - Polynomial Evaluation Horner's rule for evaluating a polynomial is:

\[ y = (((a_n x + a_{n-1}) x + a_{n-2}) x + a_{n-3}) x \ldots a_1) x + a_0 \]

A linear systolic array in which the processors are arranged in pairs: one multiplies its input by x and passes the result to the right, the next adds aj and passes the result to the right:

III. Identification of Comparative Parameters

1. Nature

The nature or behavior of matrix multiplication algorithm, that is whether the systolic algorithm or traditional matrix multiplication algorithm nature. Preplanning sequencing or no planning sequencing.

Traditional Matrix Multiplication Algorithms (TMA) are planned nature as tasks are pre assigned. i.e at compile time in a planned manner at compile time to processors and these will be no redistribution of task takes place afterwards and out come of the program is deterministic as much of the job information is known as priori.
Systolic algorithms are no planning nature as tasks are assigned at run time to processors and task redistribution can take place if task assignment what was earlier done is not given good performance. The Direct Memory (DMA) unit is an example of such an another address generator.

2. Resource Utilization

This factor is used to check the resource utilization. TMA has lesser resource utilization just tries to assign tasks to processors in order to achieve minimize response time ignore the fact that may be using this task assignment can result into a situation in which some processors finish their work early and sit ideal due to lack of work.

In systolic algorithm have relatively better resource utilization as so that no processors should site idle.

3. Communication cost

This factor is related with determine communication cost. In TMA, in a massages-passing parallel environment such as these used in implementations of high performance Fortran90[3], typical array distributions would gain spread a matrix quadrant over many processors, thereby increasing communication cost[4].

In systolic (nearest-neighbor) communication, however data must be distributed among tasks in a different fashion. We assume that A, B and C are decomposed into $\sqrt{P} \times \sqrt{P}$ sub matrices. Each task $(i,j)$ contains sub matrices $A_{jk}$, $B_{ki}$, and $C_{ji}$, where $K = [(2P - 1 - i - j) \mod \sqrt{P}]$. Computation proceeds in $\sqrt{P}$ steps. In each step, contributions to C are accumulated in each task, after which values of A move down and values of B move right. The entire equation required a total of $2(\sqrt{P} - 1)$ massages per task, each of size $N^2/P$, for a cost of $T_{\text{matrix\_matrix\_systolic}} = (2\sqrt{P} - 1)(t_s + t_w N^2/P)$ communication cost are less by a factor about $(\log P)/4$.

(4) Computation Cost

In a parallel environment, the computation cost analysis how much cost is comes up during processing. In traditional matrix multiplication the computation cost is high due to execution more time. In systolic algorithm the computation cost is less due to less execution time.
IV. CONCLUSION

This is totally dependent upon comparison of traditional matrix multiplication and systolic array algorithms. In future work can be extended to develop experimental environment to study these matrix multiplication algorithms based on comparative parameters qualitatively.

REFERENCES