Performance Analysis of Matrix-Vector Multiplication in Hybrid (MPI + OpenMP)

Vivek N. Waghmare, Sandip V. Kendre and Sanket G. Chordiya

Assistant Professor
Sandip Institute of Tech. & Research Centre, Nashik
Maharashtra (INDIA)

ABSTRACT
Computing of multiple tasks simultaneously on multiple processors is called Parallel Computing. The parallel program consists of multiple active processes simultaneously solving a given problem. Parallel computers can be roughly classified as Multi-Processor and Multi-Core. In both these classifications the hardware supports parallelism with computer node having multiple processing elements in a single machine, either in single chip pack or on more than one distinct chip respectively. Parallel programming is the ability of program to run on this infrastructure which is still quite difficult and complex task to achieve. Out of many two different approaches used in parallel environment are MPI and OpenMP, each one of them having their own merits and demerits. Hybrid model combines both approaches in the pursuit of reducing the weaknesses in individual.
In proposed approach takes a pair of, Matrices produces another matrix by using **Matrix-Vector Multiplicatio**n **Algorithm**. The resulting matrix agrees with the result of composition of the linear transformations represented by the two original matrices. This algorithm is implemented in MPI, OpenMP, and Hybrid mode. The algorithm is tested for number of nodes with different number of matrix size. The results indicates that the Hybrid approach out performs the MPI and OpenMP approach.

Keywords: MPI, OpenMP, Hybrid (MPI+OpenMP), Matrix-Vector Multiplication Algorithm

1. INTRODUCTION
Matrices are a key tool in linear algebra. One use of matrices is to represent linear transformations, which are higher-dimensional analogs of linear functions of the form $f(x) = cx$, where c is a constant; matrix multiplication corresponds to composition of linear transformations. Matrices can also keep track of the coefficients in a system of linear equations [5]. For a square matrix, the determinant and inverse matrix (when it exists) govern the behavior of solutions to the corresponding system of linear equations, and eigenvalues and eigenvectors provide insight into the geometry of the associated linear transformation. Matrices find many applications. Physics makes use of matrices in various domains, for example in geometrical optics and matrix mechanics; the latter led to studying in more detail matrices with an infinite number of rows and columns. Graph theory uses matrices to keep track of distances between pairs of vertices in a graph. Computer graphics uses matrices to project 3-dimensional space onto a 2-dimensional screen [4]. Matrix calculus generalizes classical analytical notions such as

derivatives of functions or exponentials to matrices. The latter is a recurring need in solving ordinary differential equations. Serialism and dodecaphonism are musical movements of the 20th century that use a square mathematical matrix to determine the pattern of music intervals.

Mention the word supercomputer to someone and they automatically think of monstrously complicated machines solving problems no one really understands. Maybe they think of flashing lights and some super intelligence that can beat humans at chess or figure out the meaning of life, the universe, and everything. Back in the day, this was not an altogether untrue view of supercomputing. With an entry fee of at least seven figures, supercomputing was for the serious scientists and engineers who needed to crunch numbers as fast as possible. Today we have a different world. The custom supercomputer of yesteryear has given way to commodity-based supercomputing, or what is now called High Performance Computing (HPC). In today's HPC world, it is not uncommon for the supercomputer to use the same hardware found in Web servers and even desktop workstations.

The HPC world is now open to almost everyone because the cost of entry is at an all-time low. To many organizations, HPC is now considered an essential part of business success. Your competition may be using HPC right now. They won't talk much about it because it's considered a competitive advantage [3]. Of one thing you can be sure, however; they're designing new products, optimizing manufacturing and delivery processes, solving production problems, mining data, and simulating everything from business process to shipping crates all in an effort to become more competitive, profitable, and "green". HPC may very well be the new secret weapon. The main goal of writing a parallel program is to get better performance over the Serial version. With this in mind, there are several issues that one needs to consider when designing the parallel code to obtain the best performance possible within the constraints of the problem being solved.

1.1 MPI:
The generic form of message passing in parallel processing is the Message Passing Interface (MPI), which is used as the medium of communication. A standard Message Passing Interface (MPI) is originally designed for writing applications and libraries for distributed memory environments.

However, MPI does provide message-passing routines for exchanging all the information needed to allow a single MPI implementation to operate in a heterogeneous environment [1].

1.2 Open MP:
OpenMP is an Application Program Interface (API) that may be used to explicitly direct multi-threaded, shared memory parallelism. It is a specification for a set of compiler directives, library routines and environment variables [2]. The available

programming environment on most of the Multi-Core processors will address the thread affinity to core and overheads in OpenMP Programming environment.

1.3 HYBID:

Combining shared-memory and distributed-memory programming models are an old idea [1]. One wants to exploit the strengths of both models: the efficiency, memory savings, and ease of programming of the shared-memory model and the scalability of the distributed-memory model. Until recently, the relevant models, languages, and libraries for shared-memory and distributed-memory architectures have evolved separately, with MPI becoming the dominant approach for the distributed memory, or message-passing, model, and OpenMP [2, 3] emerging as the dominant "high-level" approach for shared memory with threads.

The idea of using OpenMP [3] threads to exploit the multiple cores per node while using MPI to communicate among the nodes appears obvious. Yet one can also use an "MPI everywhere" approach on these architectures, and the data on which approach is better is confusing and inconclusive. It appears to be heavily dependent on the hardware, the MPI and OpenMP implementations, and above all on the application and the skill of the application writer.

2. IMPLEMENTATION DETAILS

The Matrix product is the most commonly used type of product of matrices. Matrices offer a concise way of representing linear transformations between vector spaces, and matrix multiplication corresponds to the composition of linear transformations [4]. The matrix product of two matrices can be defined when their entries belong to the same ring, and hence can be added and multiplied [5], and, additionally, the number of the columns of the first matrix matches the number of the rows of the second matrix. The product of an $m \times p$ matrix A with an $p \times n$ matrix B is an $m \times n$ matrix denoted AB whose entries are,

$$(AB)_{i,j} = \sum_{k=1}^{P} A_{ik} . B_{kj}$$

Where $1 \le i \le m$ is the row index and $1 \le j \le n$ is the column index.

This definition can be restated by postulating that the matrix product is left and right distributive and the matrix units are multiplied according to the following rule:

$$E_{ik}E_{lj} = \delta_k l E_{ij}$$

Where the first factor is the $m \times n$ matrix with 1 at the intersection of the ith row and the kth column and zeros elsewhere and the second factor is the $p \times n$ matrix with 1 at the intersection of the lth row and the jth column and zeros elsewhere.

In general, matrix multiplication is not. C More precisely, AB and BA need not be simultaneously defined; if they are, they may have different dimensions; and even if A and B are square matrices of the same order *n*, so that AB and BA are also square matrices of order n, if n is greater or equal than 2, AB need not be equal to BA. For example,

$$E_{11}E_{12} = E_{12}, \text{ where as } E_{12}E_{11} = 0$$

However, if A and B are both diagonal square matrices of the same order then AB = BA.

- ***Matrix Multiplication is Associative:***

$$A (BC) = (AB) C$$

- ***Matrix multiplication is Distributive over matrix addition:***

$$C (A+C) = AB+AC,$$
$$(A+B) C = AC+BC.$$

Provided that the expression in either side of each identity is defined.

- ***Matrix product is compatible with scalar multiplication:***

$$C (AB) = (CA) B = A (CB)$$

Where C is a scalar (for the second identity to hold, C must belong to the center of the ground ring this condition is automatically satisfied if the ground ring is commutative, in particular, for matrices over a field).

- If A and B are both *n*x*n* matrices with entries in a field then the determinant of their product is the product of their determinants:

$$\det (AB) = \det (A) \det(B)$$

In particular, the determinants of AB and BA coincide.

- Let *U*, *V*, and *W* be vector spaces over the same field with certain bases, $S: V \rightarrow W$ & $T: U \rightarrow V$ be linear transformations and $ST: U \rightarrow W$ be their composition. Suppose that A, B, and C are the matrices of *T*, *S*, and *ST* with respect to the given bases. Then

$$AB = C$$

Thus the matrix of the composition (or the product) of linear transformations is the product of their matrices with respect to the given bases.

The figure 2.1 to the right illustrates the product of two matrices A and B, showing how each intersection in, the product matrix corresponds to a row of A and a column of B. The size of the output matrix is always the largest possible, i.e. for each row of A and for each column of B there are always

corresponding intersections in the product matrix [6]. The product matrix AB consists of all combinations of dot products of rows of A and columns of B.

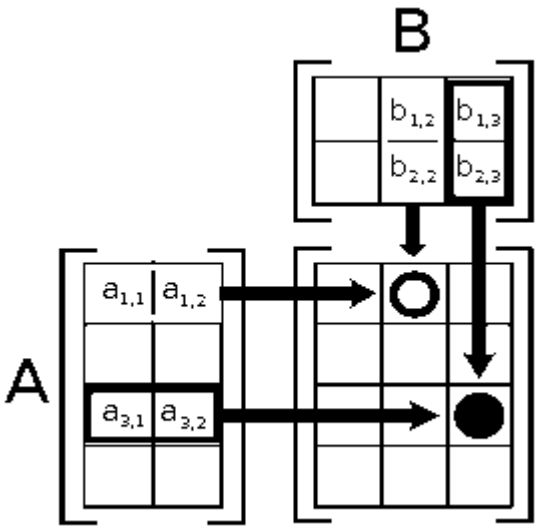

Figure 2.1 the product of two Matrices A & B.

The values at the intersections marked with circles are:

$$x_{1,2} = (a_{1,1}, a_{1,2}) \cdot (b_{1,2}, b_{1,2}) = a_{1,1} b_{1,2} + a_{1,2} b_{2,2}$$

$$x_{3,3} = (a_{3,1}, a_{3,2}) \cdot (b_{1,3}, b_{2,3}) = a_{3,1} b_{1,3} + a_{3,2}, b_{2,3}$$

3. RESULTS

Performance analysis pure MPI Vs HYBRID (MPI+OpenMP) using matrix multiplication for MPI (1+3) task on dual core and 2 task on each single core same for hybrid model. Use 2 number of threads and chunk =50 constant number of node =2**,** as shown in table 3.1 and figure 3.1 as follows.

Table 3.1 Performance of MPI time Vs HYBRID time on 2 nodes with matrix multiplication.

Matrix Size	MPI_Time (Sec)	HYBRID_Time(Sec)
100 * 100	0.1053	0.065180
200 * 200	1.60451	1.02532
400 * 400	12.451664	7.295252
600 * 600	27.401632	21.667435
1000 * 1000	81.102446	54.757333

Performance analysis pure MPI VS HYBRID (MPI+OpenMP) using matrix multiplication for MPI (1+3) task on dual core and 2 task on each single core same for hybrid model. We use 2 number of threads and chunk =50 constant number of node =2, 3, 4, as shown as in Table 3.2, 3.3, 3.4 & Figure 3.2, 3.3, 3.4.

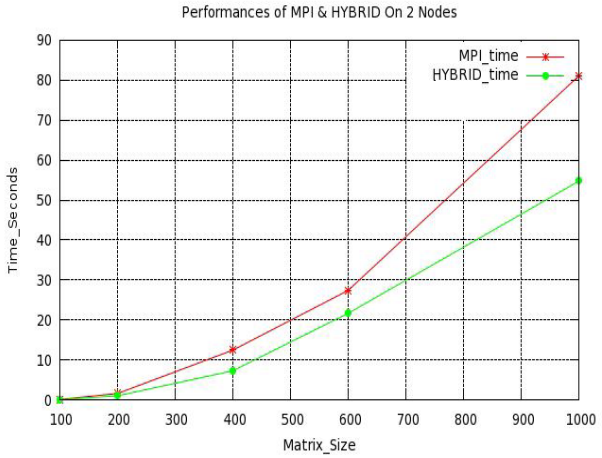

Figure 3.1 Performance of MPI time Vs HYBRID time on 2 nodes with matrix multiplication.

Table 3.2 performance of MPI time Vs HYBRID time on 4 nodes with matrix multiplication.

Matrix Size	MPI_Time (Sec)	HYBRID_Time (Sec)
1000 * 1000	74.3216	53.6614
2000 * 2000	356.2699	271.5930
4000 * 4000	2130.9338	1697.5930

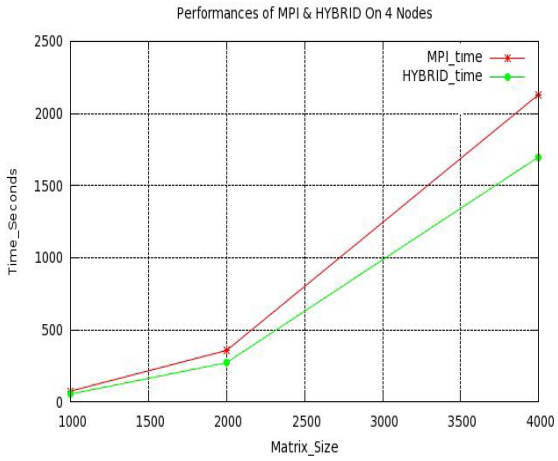

Figure 3.2 performance of MPI time Vs HYBRID time on 4 nodes with matrix multiplication.

Table 3.3 performance of MPI time Vs HYBRID time on 4 nodes with matrix multiplication.

Node_ Number	MPI_Time (Sec)	HYBRID_Time(Sec)
1	451.1239	311.9054
2	367.7262	304.9193
3	362.8631	284.3486
4	356.2699	271.5930

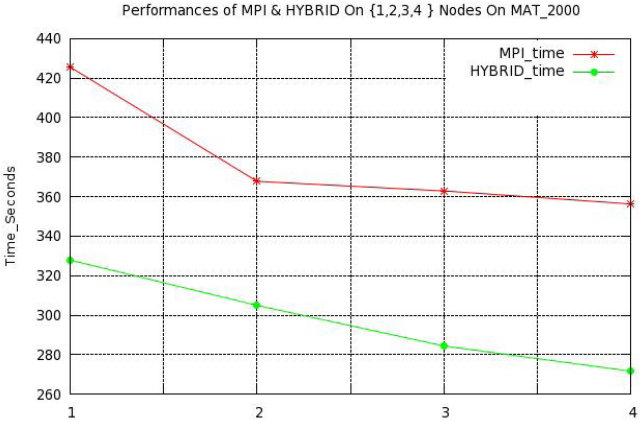

Figure 3.3 performance of MPI time Vs HYBRID time on 4 nodes with matrix multiplication.

Table 3.4 performance of MPI time Vs HYBRID time on 4 node with matrix multiplication.

Node_ Number	MPI_Time (Sec)	HYBRID_Time(Sec)
1	27.9311	20.9770
2	1.60665	7.895560
3	7.98376	6.174830
4	6.56270	4.843306

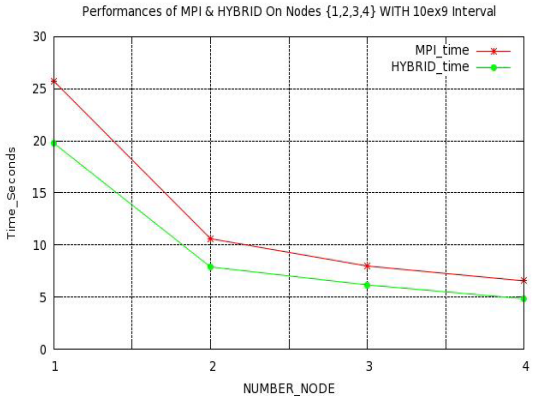

Figure 3.4 performance of MPI time Vs HYBRID time on 4 node with matrix multiplication

As seen in the figure 3.1,3.2,3.3 and 3.4 the result obtain from the Hybrid programming is gives better result than that of MPI programming due the load balancing problem of MPI programming which is reduced due to the use of OpenMP threads within MPI communication.

4. CONCLUSION

This paper compares the performance for program by using MPI, OpenMP, and Hybrid (MPI+OpenMP). It is observed that the Hybrid mixed mode programming model gives better performance than that of MPI and OpenMP programming model for the number of task and thread assigned to each processor, which is scalable.

Hence a combination of shared memory and message passing parallelization paradigms within the same application (mixed mode programming) may provide a more efficient Parallelization strategy than pure MPI and OpenMP.

5. REFERENCES

[1] MPI, MPI: \A Message-Passing Interface standard" Message Passing Interface Forum, June 1995. http://www.mpi-forum.org.

[2] OpenMP, The OpenMP ARB. http://www.OpenMP.org/.

[3] Mixed-mode programming", D. Klepacki, T.J.Watson Research Center presentations, IBM 1999. http://www.research.ibm.com/actc/Talks/DavidKlepacki/ MixedMode/htm.

[4] Henry Cohn, Robert Kleinberg, Balazs Szegedy, and Chris Umans. Group-theoretic Algorithms for Matrix Multiplication. arXiv:math.GR/0511460. Proceedings of the 46th Annual Symposium on Foundations of Computer Science, 23–25 October 2005, Pittsburgh, PA, IEEE Computer Society, pp. 379–388.

[5] Horn, Roger A.; Johnson, Charles R. (1985), Matrix Analysis, Cambridge University Press, ISBN 978-0-521-38632-6.

[6] Ran Raz. On the complexity of matrix product. In Proceedings of the thirty-fourth annual ACM symposium on Theory of computing. ACM Press, 2002. doi:10.1145/509907.509932.

[7] Finite-size errors in quantum many-body simulations of extended systems", P.R.C. Kent, R.Q. Hood, A.J.Williamson, R.J. Needs, W.M.C Foulkes, G. Rajagopal, Phys. Rev. B 59, pp 1917-1929, 1999.24.

[8] P. Lanucara and S. Rovida, "*Conjugate-Gradient algortihms: An MPI-OpenMP implementation on distributed shared memory systems",* proceeding of the 1st European Workshop on OpenMP, Lund, Sweden, 1999.

[9] D.K. Tafti, "*Computational power balancing*", Help for the overloaded processor. http://access.ncsa.uiuc.edu/ Features/Load-Balancing/