# Task Partitioning and Load Balancing Strategy for Matrix Applications on Distributed System

Adeela Bashir†, Sajjad A. Madani†, Jawad Haider Kazmi†, Kalim Qureshi§
†Department of Computer Science, COMSATS Institute of Information Technology, Abbottabad, Pakistan
§Department of Information Science, Kuwait University, Kuwait
Email: {adeelams@gmail.com}

***Abstract*— In this paper, we present a load-balancing strategy (Adaptive Load Balancing strategy) for data parallel applications to balance the work load effectively on a distributed system. We study its impact on computation-hungry matrix multiplication application. The ALB strategy enhances the performance with features such as intelligent node selection, pre-task assignment, adaptive task sizing and buffer allocation, and load balancing. The ALB strategy exhibits reduced nodes idle time and inter process communication time, and improved speed up as compared to Run Time task Scheduling strategy.**

***Index Terms*— task partitioning, load balancing, heterogeneous distributed systems, matrix multiplication, and performance evaluation**

## I. INTRODUCTION

Distributed computing system (DS) is a better choice for multifarious computations of high processing demand scientific applications. Nodes in the DS show diverse behaviour in terms of performance under variations in workload [1]. Such variations mandate the effective distribution of tasks which is a major issue in distributed computing environment [2].

Parallel matrix multiplication is one of the most studied fundamental problems in distributed and high performance computing. Our focus is to minimize the total execution time of an application that can be achieve by *reducing inter process communication and nodes idle time*, and *incorporating effective load balancing components* [1]–[3]. Matrix multiplication is used in many scientific and engineering applications such as robots, remote sensing, and medical imaging etc. The matrix multiplication is an extensive data parallel application. We have been evaluating the performance of matrix multiplication on network of workstations [4].

Recently, the matrix multiplication computational complexity has been reduced using Strassens algorithm [5]. In [5], author replace the number of matrix multiplication by matrix additions that reduced the time complexity of matrix multiplication from $O(n^3)$ to $O(n^2.8)$. The proposed model of [4] is shown in Figure 1.

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

$$\begin{aligned} Q_1 &= (a_{11} + a_{22})(b_{11} + b_{22}) \\ Q_2 &= (a_{21} + a_{22})b_{11} \\ Q_3 &= a_{11}(b_{12} - b_{22}) \\ Q_4 &= a_{22}(b_{21} - b_{11}) \\ Q_5 &= a_{11} + a_{12})b_{22}) \\ Q_6 &= (a_{21} - a_{11})(b_{11} - b_{12}) \\ Q_7 &= (a_{12} - a_{22})(b_{21} + b_{22}) \end{aligned}$$

$$\begin{aligned} C_{11} &= Q_1 + Q_4 - Q_5 + Q_7 \\ C_{12} &= Q_3 + Q_5 \\ C_{21} &= Q_2 + Q_4 \\ C_{22} &= Q_1 + Q_3 - Q_2 + Q_6 \end{aligned}$$

Figure 1. Matrix multiplication by Strassen's algorithm [4]

The authors in [5] reduced the mathematical complexity of the application at the cost of supporting features of parallel data applications, for example, independency of sub-tasks and pattern repetition. The more is the independency and pattern repetitions, the less is the inter process communication time (this is not the case in [5] as shown in Figure 1). In [6] the authors have provided a new parallel matrix multiplication algorithm based on the Strassen's algorithm, which is named CAPS (Communication-Optimal Parallel Algorithm for Strassen's Matrix Multiplication), in which they have reduced the communication overhead of Strassen's algorithm. The authors also provided a comparison of different matrix multiplication algorithms which is shown in Table I. Here $\omega_0 = \log_2 7 \approx 2.81$ is the exponent of Strassen; $\ell$ is the number of Strassen steps taken.


doi:10.4304/jcp.8.3.576-584

Classical algorithms must communicate asymptotically more than an optimal Strassen-based algorithm. To compare the lower bounds, it is necessary to consider three cases for the memory size: when the memory-dependent bounds dominate for both classical and Strassen, when the memory-dependent bound dominates for classical, but the memory-independent bound dominates for Strassen, and when the memory-independent bounds dominate for both classical and Strassen. Briefly, the factor by which the classical bandwidth cost exceeds the Strassen bandwidth cost is $P^a$ where $a$ ranges from $\frac{2}{\omega_0} - \frac{2}{3} \approx 0.046$ to $\frac{3-\omega_0}{2} \approx 0.10$ depending on the relative problem size.

Various parallel classical matrix multiplication algorithms minimize communication relative to the classical lower bounds for certain amounts of local memory $M$. For example, Cannon's algorithm [7] minimizes communication for $M = O(n^2/P)$. Several more practical algorithms exist (such as SUMMA [8]) which use the same amount of local memory and have the same asymptotic communication costs. We call this class of algorithms "2D" because the communication patterns follow a two-dimensional processor grid.

Another class of algorithms, known as "3D" [9], [10] because the communication pattern maps to a three-dimensional processor grid, uses more local memory and reduces communication relative to 2D algorithms. This class of algorithms minimizes communication relative to the classical lower bounds for $M = \Omega(n^2/P^{2/3})$. As shown in [11], it is not possible to use more memory than $M = \Theta(n^2/P^{2/3})$ to reduce communication.

Recently, a more general algorithm has been developed which minimizes communication in all cases. Because it reduces to a 2D and 3D for the extreme values of $M$ but interpolates for the values between, it is known as the "2.5D" algorithm [12].

It is therefore evident that, in such applications, the distribution of tasks become very complex and effectively increases the network usage and nodes idle time [13]. Due to wide range usage of matrix multiplication in many scientific and engineering applications, there is a need to develop an efficient task partitioning, scheduling, and load balancing strategy for such resource hungry computations.

The static and dynamic (RTS) task distribution strategies are used to balance the loads among the WSs/PCs. Since WSs/PCs have a performance variation characteristic [2], [18], therefore, the static task distribution is not effective for HCC system [19]. RTS strategy can achieve nearly perfect load balancing [20], because the machines performance variations and non-homogeneous nature of the application (image) is adjusted at runtime [5]. The RTS strategy's performance depends upon the size of the sub-task. If sub-task size is too small then it generates a serious inter-process communication overhead. In other case, if the sub-task size is too large then it may create a longer master (client) machine waiting time due to the inappropriate sub-tasks size of slow performance machine [5]. Many researchers suggested enhancement in dynamic task scheduling strategies for homogeneous HCC system [21] [6-7]. However, these strategies do not work well for heterogeneous HCC system without further modifications. The crucial point is that they are based on fixed parameters that are tuned for the specified hardware. In heterogeneous systems, this tuning is often not possible because both the computational power and the network bandwidth are not known in advance, which may change unpredictably during runtime.

The strategies based on task distribution and then task migration from heavy loaded to light-loaded nodes are expressed [22], [23]. The task migration has two serious drawbacks [23]:

- All nodes should continuously monitor the status of other nodes.
- During the computations, a node has to search its load and float the information on the network, hence, produces a large amount of communication overhead.

In DS, static and dynamic task partitioning strategies are common. Static strategy is effective only in cases where there is no change in the nature of application and system during the dispensation of application [24]. To deal with nodes performance variation problem, Runtime Task Scheduling (RTS) strategy was proposed which provides effective load balancing, but fixed task size in RTS has raised many problems, i.e., appropriate determination of task size for each node is very difficult [4]. It increases the number of requests and replies and effectively increases the nodes idle time. Optimal task sizing is a complex problems small task size increases communication cost while large task size increases nodes idle time [4].

Dynamic scheduling for the purpose of load balancing scientific computations on homogeneous as well as heterogeneous systems has been extensively researched [25], [26]. In general, scientific applications adapt to variable work loads from one

TABLE I.
ASYMPTOTIC MATRIX MULTIPLICATION COMPUTATIONAL AND COMMUNICATION COSTS OF ALGORITHMS AND CORRESPONDING LOWER BOUNDS. [6]

| | | **Flops** | **Bandwidth** | **Latency** |
|---|---|---|---|---|
| Classical | Lower Bound [14] | $\frac{n^3}{P}$ | $\max\left\{\frac{n^3}{PM^{1/2}}, \frac{n^2}{P^{2/3}}\right\}$ | $\max\left\{\frac{n^3}{PM^{3/2}}, 1\right\}$ |
| | 2D [7], [8] | $\frac{n^3}{P}$ | $\frac{n^2}{P^{1/2}}$ | $P^{1/2}$ |
| | 3D [9], [10] | $\frac{n^3}{P}$ | $\frac{n^2}{P^{2/3}}$ | $\log P$ |
| | 2.5D (optimal) [12] | $\frac{n^3}{P}$ | $\max\left\{\frac{n^3}{PM^{1/2}}, \frac{n^2}{P^{2/3}}\right\}$ | $\frac{n^3}{PM^{3/2}} + \log P$ |
| Strassen-based | Lower Bound [11], [15] | $\frac{n^{\omega_0}}{P}$ | $\max\left\{\frac{n^{\omega_0}}{PM^{\omega_0/2-1}}, \frac{n^2}{P^{2/\omega_0}}\right\}$ | $\max\left\{\frac{n^{\omega_0}}{PM^{\omega_0/2}}, 1\right\}$ |
| | 2D-Strassen [16] | $\frac{n^{\omega_0}}{P^{(\omega_0-1)/2}}$ | $\frac{n^2}{P^{1/2}}$ | $P^{1/2}$ |
| | Strassen-2D [16], [17] | $\left(\frac{7}{8}\right)^{\ell} \frac{n^3}{P}$ | $\left(\frac{7}{4}\right)^{\ell} \frac{n^2}{P^{1/2}}$ | $7^{\ell} P^{1/2}$ |
| | 2.5D-Strassen | $\max\left\{\frac{n^3}{PM^{3/2-\omega_0/2}}, \frac{n^{\omega_0}}{P^{\omega_0/3}}\right\}$ | $\max\left\{\frac{n^3}{PM^{1/2}}, \frac{n^2}{P^{2/3}}\right\}$ | $\frac{n^3}{PM^{3/2}} + \log P$ |
| | Strassen-2.5D | $\left(\frac{7}{8}\right)^{\ell} \frac{n^3}{P}$ | $\max\left\{\left(\frac{7}{8}\right)^{\ell} \frac{n^3}{PM^{1/2}}, \left(\frac{7}{4}\right)^{\ell} \frac{n^2}{P^{2/3}}\right\}$ | $\left(\frac{7}{8}\right)^{\ell} \frac{n^3}{PM^{3/2}} + 7^{\ell} \log P$ |
| | CAPS [6] | $\frac{n^{\omega_0}}{P}$ | $\max\left\{\frac{n^{\omega_0}}{PM^{\omega_0/2-1}}, \frac{n^2}{P^{2/\omega_0}}\right\}$ | $\max\left\{\frac{n^{\omega_0}}{PM^{\omega_0/2}} \log P, \log P\right\}$ |

step to the next of their computation by using an iterative static approach. In this approach, repeated partitioning is performed, and between partitioning, computations on each partition segment is performed without further concern for load balancing until completion .

In heterogeneous systems, the differences in processor speeds, architectures and memory capacities can significantly impact performance. In case of heterogeneous NOW, loads and in general resource availability of workstations are unpredictable, and thus it is difficult to know in advance what the effective speed of each machine would be. For effective load balancing of scientific applications, algorithms that derive from theoretical advances in research on scheduling parallel loop iterations with variable running times have extensively been studied [29], [30]. As a result, dynamic loop scheduling algorithms based on a probabilistic analysis have been proposed, and successfully implemented for a number of scientific applications [29], [31].

In [2], a pre task assignment strategy is proposed in which more than one tasks are assigned to nodes according to their current performance. Assigning more tasks to a faster node is not very effective because there is a possibility that faster nodes complete the tasks and then waits for further assignment which essentially will increase idle time. In [24] a load balancing strategy is proposed for data parallel applications by combining static and RTS strategies. Data is divided at compile time and during execution, in case of load imbalance; task migration is carried out. Also to reduce communication between processors, boundary elements between adjacent processors are duplicated. This strategy has to face the overheads of task migration, and memory usage due to the duplication. Also, task migration [22] requires continuous monitoring of network and floating of load information of all nodes on the network which results in degraded performance due to overhead.

Adaptive Load Balancing (ALB) strategy has been developed to exploit data parallelism on distributed systems. As a benchmark, matrix multiplication application of large dimensions is considered.

## II. ADAPTIVE LOAD-BALANCING FOR DATA PARALLEL APPLICATIONS (APL)

The terms used to formulate the APL (Adaptive load-balancing for data Parallel applications) strategy are defined in Table II, while the components of the strategy are described below.

### A. Nodes selection

Due to the advancements in processing technologies in the previous years, the increase in computing power is far beyond the increase in the network speed. The network of workstations computing environment is available in various places, for example, in industries and in educational institutions. Such environments may consist of hundred of computers and using all of them for parallel computation is

TABLE II.
SYMBOLS USED AND THEIR DEFINITIONS

| Term | Definition |
|---|---|
| $T$ | Total number of tasks (size of matrix) |
| $i$ | Node number |
| $t_i$ | Node's assigned task or computed sub-task size |
| $T_b$ | Total number of unprocessed tasks ($T_b = Tt_i$) |
| $N_{p(i)}$ | Node's estimated current performance |
| $p_i$ | Node time taken to process $\|1x1\|$ matrix |
| $B_i$ | Node's buffer size |
| $N_{pslow}$ | Slowest node's estimated current performance |
| $F_i$ | Number of pre-tasks assign to nodes buffer |
| $W_i$ | Power weight factor of node |
| $N_T$ | Total number of nodes used in DS |
| $O_i$ | Client waiting time |
| $T_{pf}$ | The time taken by the fastest machine to process the application independently |
| $T_{DS}$ | Time take by the distributed system to process the application |
| $tc_i$ | Node's completion time for the task |
| $ts_i$ | Node's start time for the task |

not scalable due to the instability of nodes performance [32], limited network performance, and over utilization of network.

In a DS, the dynamic nature of operating systems and variation in computing power compel to distribute workload as per nodes capacity. But there is a possibility that faster nodes becomes idle after completing all assigned tasks while a slower node is still processing it. A better idea will be to choose well performing and more stable nodes [32].

The stability of a node is the ratio by which the node changes its performance in the presence of heavy work load [9]. To calculate the stability of a node, we measured the variance in performance of nodes by using unit of the task (one row multiply by one column) and measured the time taken by the node. Nodes average performance is measured by average the results after 5 runs of $p_i$. Equation 1 is used to compute the nodes variance in performance.

$$Variance = \sum_{i=1}^{n} \frac{(p_i - p_{avg})^2}{n} \quad (1)$$

If a node shows more variance in performance, it means the node is not stable, i.e., in loaded condition its performance will degrade. Variances of all nodes are sent to a master which calculates *variance percentage* by using the Equation 2.

$$Variance\ percentage = \frac{Variance}{n} \times 100 \quad (2)$$

The nodes with less than 33 percentages in performance-variation are considered stable nodes and can be use in the distributed system while the others are discarded. This reduction in number of nodes is not more than 20 percent of the total number of available nodes. Using the stable nodes in DS guarantee the DS will give a better scalability and speedup by reducing the overhead of task-migration [32].

### B. Dynamic task sizing

After selecting the stable nodes for DS, the task size for each node is calculated (by using Equation 3). As the node start processing her share of work, its runtime performance is estimated (by using Equation 4). The runtime performance is than used to calculated the dynamic task size for each subsequent assignments the the respective node. which will be used to calculate each node runtime task size.

$$N_{p(i)} = \frac{1}{p_i} \quad (3)$$

Each node computes its task size whenever the node performance varies in 25 percent as compare to current nodes performance. We consider that machine's performance have dynamic in nature and 25% variation is toleratable. The updated performance of slowest node is multicast to all nodes whenever it is need to update.

$$t_i = \frac{N_{p(i)}}{N_{pslow}} \quad (4)$$

### C. Adaptive semi de-centralized approach of resources management

Since centralized resource management strategy degrades the performance of master node, therefore, the master node cannot make new assignments or receive reports from slave nodes on immediate basis [5]. This results in increased idle time of the nodes and subsequently in degraded DS performance. In APL strategy, we use a semi-decentralized approach of resources management. Initially, the master node computes the initial task size for each node and then each node is responsible to compute its performance and current task size accordingly. Therefore, in our proposed strategy, the master node can make new assignments as well as receive reports from slave nodes thus enhancing DS performance by decreasing idle time.

### D. Dynamic buffering

To completely eliminate the nodes idle time, we introduce a buffer for each node that contains the number of tasks to be execute by the node. The number of tasks is dependent on nodes performance.. In APL strategy we classified the number node into *three* categories:

- **Category A** – Highest performance $P_i \leq 25\%$ $P_i$ nodes
- **Category B** – $P_i$ of $25\% \leq 50\%$ $P_i$ nodes
- **Category C** – All Remaining nodes

The size of the buffer to hold the number of task in advance is dependent upon the node performance and is determine by Equation 5.

$$B_i = t_i \times F_i \tag{5}$$

Where $F_i$ for
Category A node is 4
Category B node is 3
Category C node is 2

Therefore, when a node completes task processing and result reporting; another thread starts processing another task available in the buffer, thus minimizes the waiting time to start new task. In maximum, 4 tasks can be held by the buffer of the node. Increase in the number of tasks in buffer will result in load balancing and task migration overhead [2].

### E. Availability of tasks at the end of application

As discussed previously, the master node is responsible for assigning tasks and collecting results. The availability of unprocessed tasks in main task buffer is the key to guarantee that none of the nodes is idle. For this purpose, the master node assigns tasks to the nodes on the basis of their performance as described above. The master follows this mechanism till 70% of the application is completed. After that, master node reduces the sub task size by 30%, i.e., the nodes getting 4 tasks previously starts getting 3 tasks. The mechanism makes sure that the buffer is not empty and requests made by nodes are entertained until the application cycle is complete. We implemented the same idea in [2].

### F. Sub-task duplication

Once all the tasks are assigned and the buffer is empty, the faster nodes finish the tasks earlier compared to the slower ones; whereas, the slow nodes very often create a master / manager waiting time. To avoid such type of waiting times, the following slow nodes task duplication mechanism from [2] is implemented:

> *"...master / manager searches the least performance node in the current HDC configuration and duplicates its assigned task to the next idle node. It creates a competition between two nodes, the result of the node who completes the task first is considered and that of the other is ignored. Each least nodes task is duplicated only once. So all computing resources are utilized at the end of application. This also eliminates the excessive manager waiting time that would have occurred at the end of application.."*

### G. ALP strategy in steps

The breakdown of the ALP into steps is shown in the Table III.

TABLE III.
MAJOR STEPS IN ALP

| Step | Description |
|---|---|
| 1 | Assign task of size $p_i$ five times to each node $i$ to calculate the performance variation. Reduce $N_T$ upto 10% by dropping nodes with higher performance-variation (to reduce the task-migration overhead). |
| 2 | Multicast $N_{pslow}$ to all connected nodes. |
| 3 | Each node $i$ computes $t_i$, categorises itself on the basis of $N_{p(i)}$ and calculates $F_i$. |
| 4 | Each node $i$ informs master about $t_i$. The master then assign task according to $F_i$. |
| 5 | When a node $i$ experiences a performance variation higher than 25% it repeats Step *3* & *4*. |
| 6 | When 70% of $T$ is completed, $t_i$ is reduce by 30% for all $i$. |
| 7 | Sub-task duplication mechanism (Section II-F) remains in effect. |

## III. COMPARISON AND IMPLEMENTATION

**Runtime task scheduling strategy (RTS):** The RTS strategy uses a unit of task that is distributed at runtime. As the node completes the previously assigned sub-task, new task is assigned for processing. The performance of ALB strategy is compared with RTS in terms of number of requests made by nodes to the master, client waiting time, and speedup of the system.

**Client waiting time:** It is the amount of idle time that nodes have to wait between completion

of previous task and assignment of next task. It is directly proportional to the number of requests made by the nodes to master / manager [2], [4]. It can be calculated as

$$O_i = ts_i - tc_{i-1} \tag{6}$$

Where $O_i$ is the overhead time during which all the communication and data access delays are performed [33]. Thus the total client waiting time is as follows:

$$Client's\ waiting\ time\ cost = \sum_{i=1}^{W_t} O_i \tag{7}$$

The theoretical speedup and measured speedup of ALB strategy is also compared. The theoretical speedup can be calculated as follows.

**Power weight**: The power weight of a node is the nodes computing speed relative to the fastest node in the distributed system [33]. It can be calculated as:

$$Power\ weight(W_i) = \frac{T_{pf}}{T_{pi}} \tag{8}$$

Power weight is then used to compute the theoretical speedup of distributed system [32] as:

$$\sum_{i=1}^{N_T} W_i \tag{9}$$

**Measured speedup**: Speedup is used to measure the performance gain from parallel computation of an application over its execution independently on the fastest node in the system. It can be defined as:

$$\frac{T_{pf}}{T_{DS}} \tag{10}$$

All measurements are carried out on homogeneous distributed computing environment. The system is composed of 30 nodes with speed of 1 GHz and Linux Operating System. Dimensions 512 x 512 and 1024 x 1024 of matrices multiplication application has been selected for the performance evaluation of the proposed strategy. Measurements are taken in both un-loaded and loaded environments, i.e., with and without users using the network.

## IV. Results and discussions

The measured number of returns and replies made by the master node are listed in Table 2 for RTS and ALB strategies. From Table IV, we can observe that for each DS configuration, the total number of returns and replies occurs in ALB strategy is quite less as compared to RTS strategy. The less number of returns and replies means lesser utilization of network resources and lower overhead. Therefore, ALB has priority on RTS strategy.

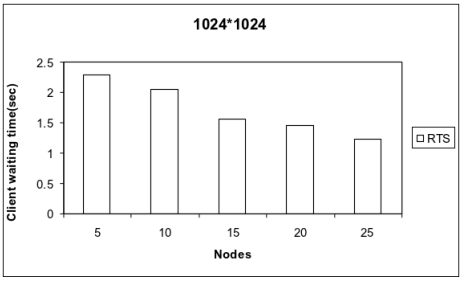


Figure 2. Measured client waiting time of five distributed system configurations in RTS strategy for matrix dimension 1024x1024.

TABLE IV.
Comparison of RTS and ALB strategies in term of number of requests made by the master to distribute the total task among the nodes

| Nodes | RTS Strategy | | ALB Strategy | |
|---|---|---|---|---|
| | Matrix Dimensions | | Matrix Dimensions | |
| | 512x512 | 1024x1024 | 512x512 | 1024x1024 |
| 5 | 512 | 1024 | 52 | 98 |
| 10 | 512 | 1024 | 84 | 152 |
| 15 | 512 | 1024 | 103 | 210 |
| 20 | 512 | 1024 | 167 | 330 |
| 25 | 512 | 1024 | 210 | 450 |

The measured results of non-optimal (using all the available nodes) DS and optimal nodes (omit the high performance variation nodes from the available nodes of DS) DS is shown in Table V. The measurement is carried out by five DS configuration. From Table V it show that not using the high performance variation nodes in DS give a better speed up.

The measured client waiting time occur in RTS strategy is shown in Figure IV. Due to the pre-task assignment and dynamic buffer sizing features of ALB strategy the client waiting time is not occur in ALB strategy.

The theoretical and measured speedup comparison is shown in figure. From Figure IV, we can conclude that measure speedup achieved by ALB strategy is closed to theoretical speedup.

The measured speedup for RTS and ALB strategy for five DS configuration is listed in Table VI. For each DS configuration the ALB strategy has better speedup as compared to RTS strategy as shown in Table VI.

TABLE V.
COMPARISON OF RTS AND ALB STRATEGIES IN TERM OF NUMBER OF REQUESTS MADE BY THE MASTER TO DISTRIBUTE THE TOTAL TASK AMONG THE NODES

| Nodes | | Speedup (Non-Optimal Nodes) | | Speedup (Optimal Nodes) | |
|---|---|---|---|---|---|
| Non-Optimal Nodes | Optimal Nodes | 512x512 | 1024x1024 | 512x512 | 1024x1024 |
| 5 | 4 | 2.18 | 2.58 | 2.53 | 2.85 |
| 10 | 8 | 3.78 | 3.28 | 3.92 | 3.42 |
| 15 | 13 | 3.24 | 4.24 | 4.19 | 4.23 |
| 20 | 18 | 3.75 | 5.15 | 4.45 | 4.93 |
| 25 | 22 | 4.26 | 5.96 | 5.02 | 5.37 |

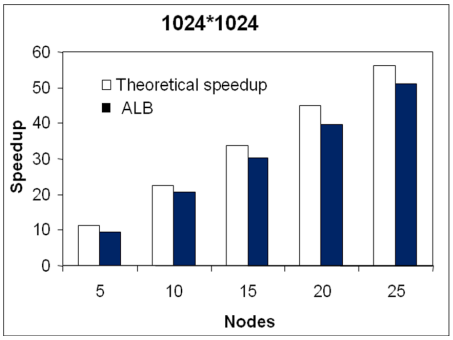


Figure 3. Comparison of theoretical and measured speedup using ALB of five distributed system configurations for matrix dimension 1024x1024.

TABLE VI.
MEASURED SPEEDUP FOR 5 DS CONFIGURATIONS IN RTS AND ALB STRATEGIES.

| Nodes | Speedup (RTS) | | Speedup (ALB) | |
|---|---|---|---|---|
| | Matrix Dimensions | | Matrix Dimensions | |
| | 512x512 | 1024x1024 | 512x512 | 1024x1024 |
| 5 | 2.241 | 2.476 | 2.53 | 2.859 |
| 10 | 3.713 | 3.061 | 3.923 | 3.429 |
| 15 | 3.981 | 3.833 | 4.194 | 4.236 |
| 20 | 4.234 | 4.634 | 4.456 | 4.938 |
| 25 | 4.601 | 5.300 | 5.021 | 5.739 |

## V. CONCLUSION

In this paper, we proposed and implemented adaptive load balancing strategy (ALB) for matrix applications on distributed systems. Experimental results show that ALB strategy has reduced number of requests made by master node and thus reduced the overhead. Due to the pre-task assignment and dynamic buffer sizing, ALB strategy also decreased the client waiting time. In case of using optimal nodes, ALB has gained 20% speedup over non-optimal number of nodes. In loaded environment ALB has gained 25% speedup over RTS. We have compared ALB speedup with theoretical speedup and the speedup of ALB is nearly equal to theoretical speedup. Besides matrix multiplication, the ALB strategy can also be applied in other data parallel applications like image process, looping, and other scientific applications.

## REFERENCES

[1] K. Qureshi, B. Majeed, J. H. Kazmi, and S. A. Madani, "Task partitioning, scheduling and load balancing strategy for mixed nature of tasks," *The Journal of Supercomputing*, vol. 19, no. 8, pp. 2228–41, 2011. [Online]. Available: http://www.springerlink.com/index/10.1007/s11227-010-0539-3

[2] S. Hussain, K. Qureshi, and H. Rashid, "Local predecimation with range index communication parallelization strategy for fractal image compression on a cluster of workstations," *Int Arab J Inf Technol*, vol. 6, no. 3, p. 293296, 2009. [Online]. Available: http://www.ccis2k.org/iajit/PDF/vol.6,no.3/13.pdf

[3] R. Shanaz and A. Usman, "Block-based spare matrix-vector multiplication on distributed memory parallel computers," *The International Arab Journal of Information Technology, Vol 8, No*, vol. 2, Apr. 2011.

[4] K. Qureshi and H. Rashid, "A practical performance comparison of parallel matrix multiplication algorithms on network of workstations," *IEE Transaction Japan*, vol. 125, no. 3, 2006.

[5] S. Hunold, T. Rauber, and G. Runger, "Combining building blocks for parallel multi-level matrix multiplication," *Parallel Computing 34*, vol. 24, pp. 411–426, Mar. 2008.

[6] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz, "Communication-Optimal Parallel Algorithm for Strassen's Matrix Multiplication," *ArXiv e-prints*, Feb. 2012.

[7] L. Cannon., "A cellular computer to implement the kalman filter algorithm." Ph.D. dissertation, Montana State University, Bozeman, MN, 1969.

[8] R. A. van de Geijn and J. Watts, "Summa: scalable universal matrix multiplication algorithm," *Concurrency - Practice and Experience*, vol. 9, no. 4, pp. 255–274, 1997.

[9] J. Berntsen, "Communication efficient matrix multiplication on hypercubes," *Parallel Computing*, vol. 12, no. 3, pp. 335–342, 1989.

[10] R. C. Agarwal, S. M. Balle, F. G. Gustavson, M. Joshi, and P. Palkar, "A three-dimensional approach to parallel matrix multiplication," *IBM Journal of Research and Development*, vol. 39, pp. 39–5, 1995.

[11] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz., "Strong scaling of matrix multiplication algorithms and memory-independent communication lower bounds," 2012., submitted to SPAA.

[12] E. Solomonik and J. Demmel., "Communication-optimal parallel 2.5d matrix multiplication and lu factorization algorithms." in *Euro-Par11: Proceedings of the 17th International European Conference on Parallel and Distributed Computing. Springer*, 2011.

[13] K. Qureshi and M. Hatanaka, "A Practical Approach of Task Scheduling and Load Balancing on Heterogeneous Distributed Raytracing System," *Information Processing Letter (IPL), Elsevier press*, vol. 79, no. 30, pp. 65–71, June 2001.

[14] D. Irony, S. Toledo, and A. Tiskin, "Communication lower bounds for distributed-memory matrix multiplication," *J. Parallel Distrib. Comput.*, vol. 64, no. 9, pp. 1017–1026, 2004.

[15] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, "Graph expansion and communication costs of fast matrix multiplication," in *SPAA 11: Proceedings of the 23rd annual symposium on parallelism in algorithms and architectures*, 2011, pp. 1–12.

[16] Q. Luo and J. Drake, "A scalable parallel Strassens matrix multiplication algorithm for distributed-memory computers," in *Proceedings of the 1995 ACM symposium on Applied computing, SAC 95*, 1995, pp. 221–226.

[17] B. Grayson, A. Shah, and R. van de Geijn, "A high performance parallel Strassen implementation," in *Parallel Processing Letters*, 1995, vol. 6, pp. 3–12.

[18] M. Hamid and C. Le, "Dynamic load-balancing of image processing applications on cluster of workstations," *Parallel Computing*, vol. 22, pp. 1477–1492, 1997.

[19] Y. Zhang, H. Kameda, and S. L. Hung, "Comparison of dynamic and static load-balancing strategies in heterogeneous distributed systems," *IEE Proceeding Comput. Digit. Tech.*, vol. 144, no. 2, pp. 100–107, 1997.

[20] J. Bloomer, *Power programming with RPC*. OReilly & Associates, 1991.

[21] C. Lee and M. Hamid, *Parallel image processing applications on a network of workstations*. Parallel Computing, 1995.

[22] P. Liu and C.-H. Yang, "Locality-preserving dynamic load balancing for data-parallel applications on distributed-memory multiprocessors," 2000.

[23] R. L. Cariiio, I. Banicescu, R. K. Vadapalli, C. A. Weatherford, and J. Zhu, "Parallel adaptive quantum trajectory method for wavepacket simulations," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS 2003) - PDSECA Workshop, IEEE Computer Society Press*, 2003, pp. 202–211.

[24] K. Schloegel, G. Karypis, and V. Kumar, "Dynamic repartitioning of adaptively refined meshes," in *Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM)*, ser. Supercomputing '98. Washington, DC, USA: IEEE Computer Society, 1998, pp. 1–8. [Online]. Available: http://dl.acm.org/citation.cfm?id=509058.509087

[25] A. Sarje and G. Sagar, "heuristic model for task allocation in distributed computing systems," *IEE Proceedings -E*, vol. 5, pp. 313–318, 1991.

[26] S. P. Dandamudi, "Sensitivity evaluation of dynamic load sharing in distributed systems," *IEEE Concurrency*, vol. 6, no. 3, pp. 62–72, July 1998. [Online]. Available: http://dx.doi.org/10.1109/4434.708257

[27] R.-Z. Khan, "Empirical study of task partitioning scheduling and load balancing for distributed image computing system," Ph.D. dissertation, Jamia Hamdard, Dheli, India, 2005.

[28] H. Rashid, "Parallel scientific applications scheduling on distributed computing system," Ph.D. dissertation, Preston University, Pakistan, 2006.

[29] S. F. Hummel, E. Schonberg, and L. E. Flynn, "Factoring: a method for scheduling parallel loops," *Commun. ACM*, vol. 35, no. 8, pp. 90–101, Aug. 1992. [Online]. Available: http://doi.acm.org/10.1145/135226.135232

[30] R. Biswas and L. Oliker, "Load balancing unstructured adaptive grids for cfd problems," in *8th IEEE Symposium on Parallel and Distributed Processing*, 1997, pp. 26–33.

[31] I. Banicescu and S. Flynn Hummel, "Balancing processor loads and exploiting data locality in n-body simulations," in *Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*, ser. Supercomputing '95. New York, NY, USA: ACM, 1995. [Online]. Available: http://doi.acm.org/10.1145/224170.224306

[32] S. Fu, "Failure-aware construction and reconfiguration of distributed virtual machines for high availability computing," in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, ser. CCGRID '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 372–379. [Online]. Available: http://dx.doi.org/10.1109/CCGRID.2009.21

[33] X. Zhang and Y. Yan, "Modeling and characterizing parallel computing performance on heterogeneous networks of workstations," in *Proceedings of the 7th IEEE Symposium on Parallel and Distributeed Processing*, ser. SPDP '95. Washington, DC, USA: IEEE Computer Society, 1995, pp. 25–. [Online]. Available:

http://dl.acm.org/citation.cfm?id=829516.830568

**Adeela Bashir** did her MS (Computer Science) from COMSATS Institute of Information Technology, Abbottabad, Pakistan, in 2010. Currently she is a faculty member of Abah University college, Saudi Arabia.

**Dr. Sajjad Ahmad Madani**, currently Associate Professor at COMSATS Institute of Information Technology (CIIT) Abbottabad Campus, has joined CIIT in August 2008 as Assistant Professor. Previous to that, he was with the institute of computer technology, Austria, from 2005 to 2008 as guest researcher where he did his PhD research. Prior to joining ICT, he taught at COMSATS institute of Information Technology for a period of 2 years. He has done MS in Computer Sciences from Lahore University of Management Sciences (LUMS), Pakistan with excellent academic standing. He has already done BSc Civil Engineering from UET Peshawar and was awarded a gold medal for his outstanding performance in academics. His areas of interest include low power wireless sensor network and application of industrial informatics to electrical energy networks. He has published more than 35 papers in international conferences and journals.

**Jawad Haider Kazmi** is currently working as Assistant Professor at the department of computer science, CIIT Abbottabad, Pakistan since 2008. He took MIT in 2003 and his MS in Computer Science in 2007, after many years of industry experience. His research interest includes medical imaging, distributed systems and computer networks.

**Dr. Kalim Qureshi**, currently Associate Professor at the Information Science Department of the Kuwait University, Kuwait. His research interests include network parallel distributed computing, thread programming, concurrent algorithms designing, task scheduling, and performance measurement. Dr. Qureshi receive his Ph.D and MS degrees from Muroran Institute of Technology, Hokkaido, Japan in (2000, 1997).