Parallel Algorithm for Time Series Based Forecasting on OTIS-Mesh

Sudhanshu Kumar Jha
Senior Member, IEEE
Department of Computer Science and Engineering
Indian School of Mines, Dhanbad 826 004, India

Prasanta K. Jana
Senior Member, IEEE
Department of Computer Science and Engineering
Indian School of Mines, Dhanbad 826 004, India

ABSTRACT
Forecasting plays an important role in business, technology, climate and many others. As an example, effective forecasting can enable an organization to reduce lost sales, increase profits and more efficient production planning. In this paper, we present a parallel algorithm for short term forecasting based on a time series model called weighted moving average. Our algorithm is mapped on OTIS-mesh, a popular model of optoelectronic parallel computers. Given m data values and n window size, it requires $5(\sqrt{n}-1)$ electronic moves + 4 OTIS moves using n2 processors. Scalability of the algorithm is also discussed.

Keywords
Parallel algorithm, OTIS-mesh, time series forecasting, weighted moving average

INTRODUCTION
OTIS-mesh is a popular model of Optical Transpose Interconnection Systems (OTIS) [1]. In an OTIS-mesh, n^2 processors are organized into n groups in a two dimensional layout (as shown in Figure 1)

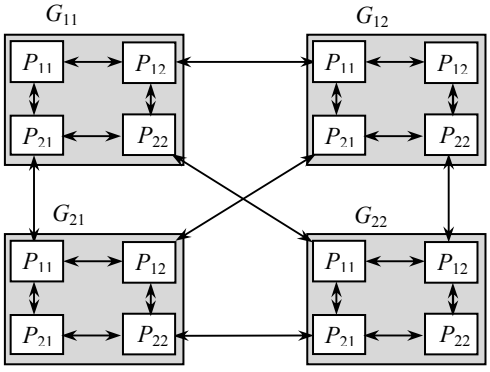

Figure 1. OTIS-Mesh network of 2^4 processors.

where each group is a $\sqrt{n} \times \sqrt{n}$ 2D-mesh. The processors within each group are connected by electronic links following mesh topology where as the processors in two different groups are connected via optical links following transpose rule discussed afterwards. Let G_{xy} denote the group placed in the x^{th} row and y^{th} column, then we address the processor placed in the u^{th} row and v^{th} column within G_{xy} by (G_{xy},P_{uv}). Using transpose rule, (G_{xy},P_{uv}) is directly connected to (P_{uv}, G_{xy}). In the complexity analysis of a parallel algorithm on a OTIS model, the electronic and the optical links are distinguished by counting the data movement over electronic and the optical links separately which are termed as electronic moves and OTIS moves respectively.

In the recent years, OTIS has created a lot of interests among the researchers. Several parallel algorithms for various numeric and non-numeric problems have been developed on different models of OTIS including image processing [2], matrix multiplication [3], basic operations [4], BPC permutation [5], prefix computation [6], polynomial interpolation and root finding [7], Enumeration sorting [8], phylogenetic tree construction [9] randomized algorithm for routing, selection and sorting [10].

Among different quantitative forecasting models available for successful implementation of decision making systems, time series models are very popular. In these models, given a set of past observations, say $d_1, d_2, \ldots, d_m$, the problem is to estimate $d(m + \tau)$ through extrapolation, where τ (called the lead time) is a small positive integer and usually set to 1. The observed data values usually show different patterns, such as constant process, cyclical and linear trend as shown in Figure 2. Several models are available for time series forecasting. However, a particular model may be effective for a specific pattern of the data, e.g. moving average is very suitable when the data exhibits a constant process. Weighted moving average is a well known time series model for short term forecasting which is suitable when the data exhibits a cyclical pattern around a constant trend [11].

In this paper, we present a parallel algorithm for short term forecasting which is based on weighted moving average of time series model and mapped on OTIS-Mesh. We show that the

algorithm requires $5(\sqrt{n}-1)$ electronic moves + 4 OTIS moves for m size data set and n window size using n^2 processors.

This is important to note that the exponential weighted moving average is more widely accepted technique method for short term forecasting than the (simple) weighted moving average. However, our motivation to parallelize weighted moving average with the fact that both the exponential weighted moving average and the simple moving average (MA) are the special cases of the weighted moving average as will be discussed in section 2. Moreover, in order to find the optimum value of the window size, it involves $O(m)$ iterations where each iteration requires $O(n^2)$ time for calculating $(m - n + 1)$ weighted averages for a window size n and m size

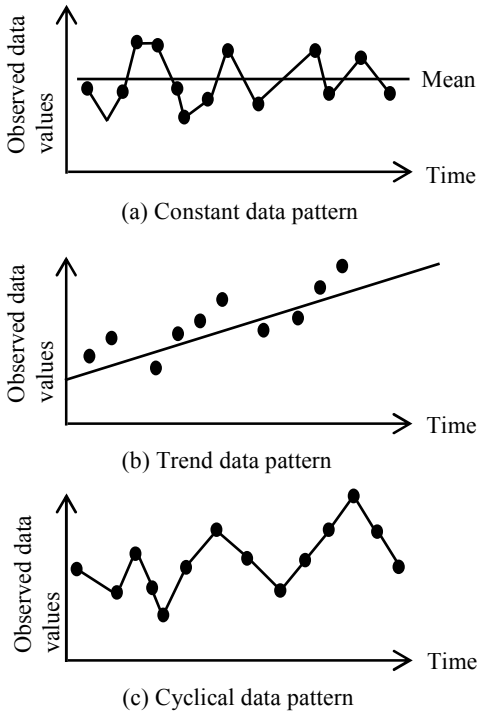

Figure 2. Illustration of different types of data pattern

data set. This is expensive when the data size is very large.

Quite a few parallel algorithms have been reported for short term forecasting. The parallel algorithms presented in [12] are based on weighted moving average and shown to implement on a linear array in $m + 1$ steps using n processors and on a tree architecture in $(m - n + 2) + \log n$ steps with $(2n - 1)$ processors. The algorithms have also been extended to map on a ST-array and ST-tree in $\frac{n}{p}(m-n+1)+p-1$ and $\frac{n}{p}[(m-n+2)+\log_2 p]$ steps respectively when only p processors are available. The systolic algorithm [13] for moving average was shown to require $m - n + 1$ steps with $n + 1$ processors. To the best of our knowledge, no other parallel algorithms have been reported for short term forecasting.

The rest of the paper is organized as follows. Section 2 describes the time series forecasting and the method of weighted moving average with its special cases. In section 3, we present our proposed parallel algorithm followed by the conclusion in section 4.

1. WEIGHTED MOVING AVE-RAGE TECHNIQUE

For the completeness of the paper, we describe here the forecasting methodology using weighted moving average as given in [12]. In this method, for a set of n data values d_t, d_{t+1}, …, d_{t-n+1} and a set of positive weights w_1, w_2, …, w_n, we calculate their weighted moving average at time t by the following formula

$$W^M(t)=\frac{w_n d_t + w_{n-1}d_{t-1} + \cdots + w_1 d_{t-n+1}}{w_n + w_{n-1} + \cdots + w_1} \qquad (2.1)$$

where $w_n \geq w_{n-1} \geq \ldots \geq w_1 \geq 0$. We then use $W^M(t)$ to estimate the forecast value $\hat{d}(t+\tau)$ at time $t+\tau$, i.e., $\hat{d}(t+\tau) \approx W^M(t)$. The quality of the forecast depends on the selection of the window size (n). Therefore, in order to find the optimum value of n, we calculate $m - n + 1$ weighted averages for a specific value of n by sliding the window over the data values and the corresponding mean square error (*MSE*) is also calculated using

$$MSE = \sum_{t=n+\tau}^{m} \frac{[d_t - \hat{d}_t]^2}{(m-n-\tau+1)} \qquad (2.2)$$

We then vary the window size (n) to obtain the corresponding *MSE* with the newly calculated weighted moving averages. The same process is repeated for $n = 1, 2, 3, \ldots, m$. The value of n for which *MSE* is least is chosen for forecasting.

Some Special Cases:

Simple Moving Average: In this method, equal importance is given to each data value. Hence we assign equal weight $1/n$ to all the data values and obtain the following moving average

$$S^M(t)=\frac{d_t + d_{t-1} + d_{t-2} + \ldots + d_{t-n+1}}{n} \qquad (2.3)$$

As we have discussed in section 1, this method is best when the data pattern shows a constant process.

Exponential Moving Average: In this method, the more recent observations are given a larger weight to face smaller error and thus the weights are assigned in decreasing order. The formula for exponential moving average is as follows

$$E^M(t) = \alpha d_t + \alpha(1-\alpha)d_{t-1} + \alpha(1-\alpha)^2 d_{t-2} + \cdots + \alpha(1-\alpha)^{n-1} d_{t-n+1} \qquad (2.4)$$

where weight $w_i = \alpha\,(1-\alpha)^{n-i}$, $1 \le i \le n$ and $0 \le \alpha \le 1$. This method is suitable for a cyclical pattern around a constant trend and is widely accepted specially for business environment. However, the method suffers form the proper selection of the value of the α parameter and there is no easy method to do it.

3. PROPOSED ALGORITHM

Assume $\tau = 1$. Then $(m - n + 1)$ weighted moving averages are obtained form equation (2.1) for a given window size n along with their error term as follows

Weighted Moving Average

$$\left.\begin{aligned} W^M(n) &= \frac{w_1 d_1 + w_2 d_2 + \cdots + w_n d_n}{w_1 + w_2 + \cdots + w_n} \\ W^M(n+1) &= \frac{w_1 d_2 + w_2 d_3 + \cdots + w_n d_{n+1}}{w_1 + w_2 + \cdots + w_n} \\ W^M(n+2) &= \frac{w_1 d_3 + w_2 d_4 + \cdots + w_n d_{n+2}}{w_1 + w_2 + \cdots + w_n} \\ &\vdots \\ W^M(m) &= \frac{w_1 d_{m-n+1} + w_2 d_{m-n+2} + \cdots + w_n d_m}{w_1 + w_2 + \cdots + w_n} \end{aligned}\right\} \quad (2.5)$$

Error Terms

$$\left.\begin{aligned} E_{n+1} &= d_{n+1} - W^M(n) \\ E_{n+2} &= d_{n+2} - W^M(n+1) \\ E_{n+3} &= d_{n+3} - W^M(n+2) \\ &\vdots \\ E_m &= d_m - W^M(m-1) \end{aligned}\right\} \quad (2.6)$$

$$MSE = \sum_{t=n+1}^{m} \frac{E_i^2}{m-n} \quad (2.7)$$

This is easy to note that the sequential implementation of the above computation requires $O(n^2)$ time. For a different value of n say n_i, $1 \le i \le m$, we require to compute different set of $(m - n_i + 1)$ weighted moving averages (as given above) for a maximum of m iterations. However, our target is to parallelize the above computation for a single iteration so that the overall time complexity can be significantly reduced. The basic idea is as follows. We initially feed the data values and the weight vector through the boundary processors. Then using suitable electronic and OTIS moves, they are stored in the D and W registers respectively. Next we calculate their products for each processor in parallel. The products are then used to form the local sum in each group which are finally accumulated using suitable electronic and OTIS moves to produce weighted moving averages. The algorithm is now formally described as follows.

Algorithm: Parallel_WMA

Step 1. /* Data Input */

1.1 Feed the data values d_i's, $1 \le i \le m$ to the boundary processors in the 1st column position of each group G_{xy}, $1 \le x, y \le \sqrt{n}$ as shown in Figure 3.

1.2 Feed the weights w_j's, $1 \le j \le n$ to the boundary processors in the 1st row position of the group G_{1y}, $1 \le y \le \sqrt{n}$ as shown in Figure 3.

Step 2. /* Data distribution into D-registers */

Shift the data values row-wise to store them in D-registers in a pipeline fashion (as data storing for mesh sort [14]).

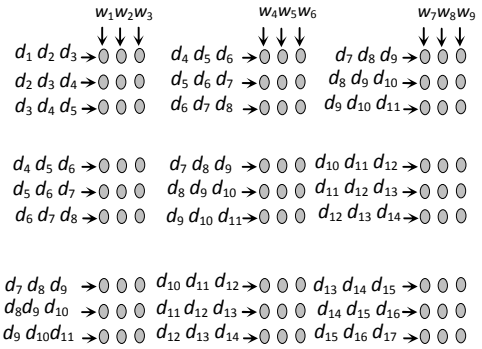

Figure 3. Data input of d_i's and w_i's value.

Step 3. /* Distribution of weights */

3.1 Perform column-wise broadcast on the weights fed in step 1.2 and store them in W register.

***Illustration* 1:** Contents of D and W registers after this step are shown in Figure 4.

3.2 Perform one OTIS move on the contents of W registers stored in step 3.1.

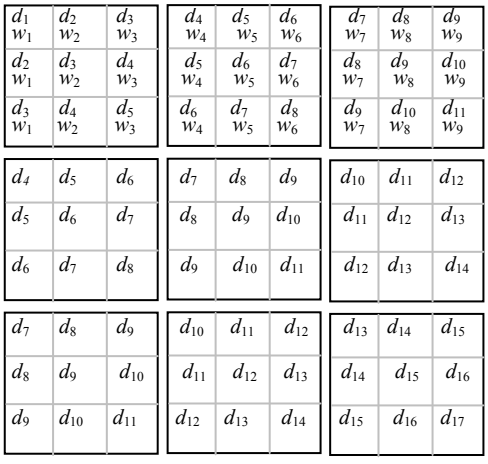

Figure 4. After row-wise shift of d_i's and column wise broadcast of w_j's.

3.3 Perform column-wise broadcast on *W* register contents stored in step 3.2.

3.4 Perform OTIS move on *W* registers.

***Illustration* 2**. The results after step 3.3 and 3.4 are shown in Figures 5 and 6 respectively.

d_1 w_1	d_2 w_4	d_3 w_7	d_4 w_2	d_5 w_5	d_6 w_8	d_7 w_3	d_8 w_6	d_9 w_9
d_2 w_1	d_3 w_4	d_4 w_7	d_5 w_2	d_6 w_5	d_7 w_8	d_8 w_3	d_9 w_6	d_{10} w_9
d_3 w_1	d_4 w_4	d_5 w_7	d_6 w_2	d_7 w_5	d_8 w_8	d_9 w_3	d_{10} w_6	d_{11} w_9
d_4 w_1	d_5 w_4	d_6 w_7	d_7 w_2	d_8 w_5	d_9 w_8	d_{10} w_3	d_{11} w_6	d_{12} w_9
d_5 w_1	d_6 w_4	d_7 w_7	d_8 w_2	d_9 w_5	d_{10} w_8	d_{11} w_3	d_{12} w_6	d_{13} w_9
d_6 w_1	d_7 w_4	d_8 w_7	d_9 w_2	d_{10} w_5	d_{11} w_8	d_{12} w_3	d_{13} w_6	d_{14} w_9
d_7 w_1	d_8 w_4	d_9 w_7	d_{10} w_2	d_{11} w_5	d_{12} w_8	d_{13} w_3	d_{14} w_6	d_{15} w_9
d_8 w_1	d_9 w_4	d_{10} w_7	d_{11} w_2	d_{12} w_5	d_{13} w_8	d_{14} w_3	d_{15} w_6	d_{16} w_9
d_9 w_1	d_{10} w_4	d_{11} w_7	d_{12} w_2	d_{13} w_5	d_{14} w_8	d_{15} w_3	d_{16} w_6	d_{17} w_9

Figure 5. After column-wise broadcast of w_j's.

d_1 w_1	d_2 w_2	d_3 w_3	d_4 w_4	d_5 w_5	d_6 w_6	d_7 w_7	d_8 w_8	d_9 w_9
d_2 w_1	d_3 w_2	d_4 w_3	d_5 w_4	d_6 w_5	d_7 w_6	d_8 w_7	d_9 w_8	d_{10} w_9
d_3 w_1	d_4 w_2	d_5 w_3	d_6 w_4	d_7 w_5	d_8 w_6	d_9 w_7	d_{10} w_8	d_{11} w_9
d_4 w_1	d_5 w_2	d_6 w_3	d_7 w_4	d_8 w_5	d_9 w_6	d_{10} w_7	d_{11} w_8	d_{12} w_9
d_5 w_1	d_6 w_2	d_7 w_3	d_8 w_4	d_9 w_5	d_{10} w_6	d_{11} w_7	d_{12} w_8	d_{13} w_9
d_6 w_1	d_7 w_2	d_8 w_3	d_9 w_4	d_{10} w_5	d_{11} w_6	d_{12} w_7	d_{13} w_8	d_{14} w_9
d_7 w_1	d_8 w_4	d_9 w_7	d_{10} w_2	d_{11} w_5	d_{12} w_8	d_{13} w_3	d_{14} w_6	d_{15} w_9
d_8 w_1	d_9 w_2	d_{10} w_3	d_{11} w_4	d_{12} w_5	d_{13} w_6	d_{14} w_7	d_{15} w_8	d_{16} w_9
d_9 w_1	d_{10} w_2	d_{11} w_3	d_{12} w_4	d_{13} w_5	d_{14} w_6	d_{15} w_7	d_{16} w_8	d_{17} w_9

Figure 6. After OTIS move on w_j's.

***Remark* 1:** The distribution of w_j's can be similarly implemented as the data values d_i's by feeding them in the 1st column position of each group. However, it would increase the total number of I/O ports.

Step 4.. ∀ processors *do in parallel*

Form the products with the contents of *D* and *W* registers and store it in *C*-register.

Step 5. ∀ groups *do* steps 5.1 and 5.2 *in parallel*

5.1 Sum up the contents of *C*-registers row-wise and store the partial sum into *C*-register of the 1st column processors of each group.

5.2 Sum up the contents of *W*-register row-wise and store the partial sum into *W*-register of the 1st column processors of each group.

***Illustration* 3**: The results after this step is shown in Figure 7 in which C_i^j indicates the i^{th} partial sum of the j^{th} computation and W^j denotes the j^{th} partial sum of the weights. We also show the detailed results of C_i^j's and W^j's processor-wise within each group in Table 1.

Step 6. Perform OTIS move on the contents of both *C* and *W*-registers stored in step 5. Result is shown in Figure 8.

Step 7. Same as step 5.

***Step* 8:** Perform OTIS move on *C* and *W*- registers to rearrange them.

Step 9: ∀ processors *do in parallel*

Divide the content of *C*-register by the *W*-register to store in *R*-registers

***Remark* 2:** The final results emerge from the *R*- registers of processors (G_{x1}, P_{u1}), $1 \le x \le \sqrt{n}$, $1 \le u \le \sqrt{n}$.

Time Complexity: Each of the steps 2, 3.1, 3.3, 5, 7, requires $\sqrt{n}-1$ electronic moves, steps 3.2, 3.4, 6, 8 require one OTIS moves for each and rest of the steps are completed in constant time. Therefore, the above algorithm requires $5(\sqrt{n}-1)$ electronic moves + 4 OTIS moves.

C_1^1 W^1	-	-	C_1^2 W^2	-	-	C_1^3 W^3	-	-
C_2^1 W^1	-	-	C_2^2 W^2	-	-	C_2^3 W^3	-	-
C_3^1 W^1	-	-	C_3^2 W^2	-	-	C_3^3 W^3	-	-
C_4^1 W^1	-	-	C_4^2 W^2	-	-	C_4^3 W^3	-	-
C_5^1 W^1	-	-	C_5^2 W^2	-	-	C_5^3 W^3	-	-
C_6^1 W^1	-	-	C_6^2 W^2	-	-	C_6^3 W^3	-	-
C_7^1 W^1	-	-	C_7^2 W^2	-	-	C_7^3 W^3	-	-
C_8^1 W^1	-	-	C_8^2 W^2	-	-	C_8^3 W^3	-	-
C_9^1 W^1	-	-	C_9^2 W^2	-	-	C_9^3 W^3	-	-

Figure 7. Contents of *C* and *W* registers after step 5.

Table 1. Showing the result after row-wise addition in step 5.

	P_{00}	P_{10}	P_{20}
G_{00}	$C_1^1 = d_1w_1 + d_2w_2 + d_3w_3$ $W^1 = w_1 + w_2 + w_3$	$C_1^2 = d_4w_4 + d_5w_5 + d_6w_6$ $W^2 = w_4 + w_5 + w_6$	$C_1^3 = d_7w_7 + d_8w_8 + d_9w_9$ $W^3 = w_7 + w_8 + w_9$
G_{01}	$C_2^1 = d_2w_1 + d_3w_2 + d_4w_3$ $W^1 = w_1 + w_2 + w_3$	$C_2^2 = d_5w_4 + d_6w_5 + d_7w_6$ $W^2 = w_4 + w_5 + w_6$	$C_2^3 = d_8w_7 + d_9w_8 + d_{10}w_9$ $W^3 = w_7 + w_8 + w_9$
G_{02}	$C_3^1 = d_3w_1 + d_4w_2 + d_5w_3$ $W^1 = w_1 + w_2 + w_3$	$C_3^2 = d_6w_4 + d_7w_5 + d_8w_6$ $W^2 = w_4 + w_5 + w_6$	$C_3^3 = d_9w_7 + d_{10}w_8 + d_{11}w_9$ $W^3 = w_7 + w_8 + w_9$
G_{10}	$C_4^1 = d_4w_1 + d_5w_2 + d_6w_3$ $W^1 = w_1 + w_2 + w_3$	$C_4^2 = d_7w_4 + d_8w_5 + d_9w_6$ $W^2 = w_4 + w_5 + w_6$	$C_4^3 = d_{10}w_7 + d_{11}w_8 d_{12}w_9$ $W^3 = w_7 + w_8 + w_9$
G_{11}	$C_5^1 = d_5w_1 + d_6w_2 + d_7w_3$ $W^1 = w_1 + w_2 + w_3$	$C_5^2 = d_8w_4 + d_9w_5 + d_{10}w_6$ $W^2 = w_4 + w_5 + w_6$	$C_5^3 = d_{11}w_7 + d_{12}w_8 + d_{13}w_9$ $W^3 = w_7 + w_8 + w_9$
G_{12}	$C_6^1 = d_6w_1 + d_7w_2 + d_8w_3$ $W^1 = w_1 + w_2 + w_3$	$C_6^2 = d_9w_4 + d_{10}w_5 + d_{11}w$ $W^2 = w_4 + w_5 + w_6$	$C_1^3 = d_{12}w_7 + d_{13}w_8 d_{14}w_9$ $W^3 = w_7 + w_8 + w_9$
G_{20}	$C_7^1 = d_7w_1 + d_8w_2 + d_9w_3$ $W^1 = w_1 + w_2 + w_3$	$C_7^2 = d_{10}w_4 + d_{11}w_5 + d_{12}w_6$ $W^2 = w_4 + w_5 + w_6$	$C_7^3 = d_{13}w_7 + d_{14}w_8 d_{15}w_9$ $W^3 = w_7 + w_8 + w_9$
G_{21}	$C_8^1 = d_8w_1 + d_9w_2 + d_{10}w_3$ $W^1 = w_1 + w_2 + w_3$	$C_8^2 = d_{11}w_4 + d_{12}w_5 + d_{13}w_6$ $W^2 = w_4 + w_5 + w_6$	$C_8^3 = d_{14}w_7 + d_{15}w_8 d_{16}w_9$ $W^3 = w_7 + w_8 + w_9$
G_{22}	$C_9^1 = d_9w_1 + d_{10}w_2 + d_{11}w_3$ $W^1 = w_1 + w_2 + w_3$	$C_9^2 = d_{12}w_4 + d_{13}w_5 + d_{14}w_6$ $W^2 = w_4 + w_5 + w_6$	$C_9^3 = d_{15}w_7 + d_{16}w_8 d_{17}w_9$ $W^3 = w_7 + w_8 + w_9$

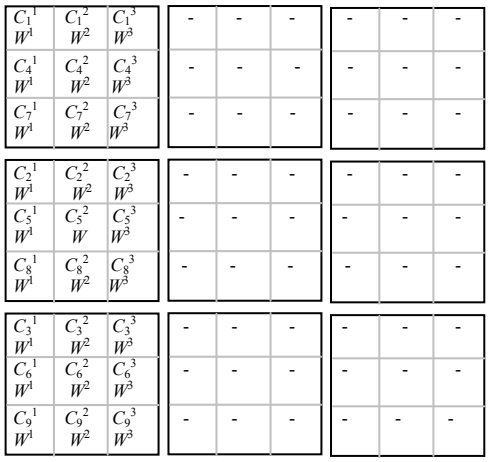

Figure 8. After one OTIS move.

Scalability: Now we consider any arbitrary size of the window to map the above algorithm on a $\sqrt{n} \times \sqrt{n}$ OTIS-mesh. In other words, we consider the case when the window size is independent of the number of processors. For the sake of simplicity and without any loss of generality, let us assume it to be *kn*. Note that in this case, the size of the data set will be $2kn - 1$. Then we can partition the weight set into *k* subsets: $\{w_1, w_2, \ldots, w_n\}$, $\{w_{n+1}, w_2, \ldots, w_{2n}\} \ldots$, $\{w_{(k-1)n+1}, w_{(k-1)n+2}, \ldots, w_{kn}\}$. Accordingly the data set is also partition into *k* subsets: $\{d_1, d_2, \ldots, d_n\}$, $\{d_2, d_3, \ldots, d_{n+1}\}, \ldots, \{d_{2kn-n}, d_{2kn-n+1}, \ldots, d_{2kn-1}\}$. Given a subset of the data, its corresponding weight subset is fed to the $\sqrt{n} \times \sqrt{n}$ OTIS-mesh. We then run the above algorithm (Parallel_WMA) and store the result temporarily. Next we input another data subset along with the corresponding weight subset, execute Parallel_WMA and update the current result with the previously calculated partial result. This process is repeated *k* times to yield the final result. This is obvious to note that this version of the algorithm requires $5k(\sqrt{n} - 1)$ electronic moves + 4*k* OTIS moves, which is *k* times more than time complexity of parallel_WMA.

4. CONCLUSION

In this paper, we have presented a parallel algorithm for short term forecasting using weighted moving average technique. The algorithm is mapped on n^2- processor OTIS-mesh. We have shown that it requires $5(\sqrt{n} - 1)$ electronic moves + 4 OTIS moves. The algorithm is also shown to be scalable.

REFERENCES

[1] Zane F., Marchand P., Paturi R. and Esener S., 2000. Scalable network architectures using the optical transpose interconnection system (OTIS), J. of Parallel and Distributed Computing, 60, 521-538.

[2] Wang C. F. and Sahni S., 2000. Image processing on the OTIS-Mesh optoelectronic Computer, IEEE Trans. on Parallel and Distributed Systems, 11, 97-109.

[3] Wang C. F. and Sahni S., 2001. Matrix Multiplication on the OTIS-Mesh Optoelectronic Computer, IEEE Transactions on Computers, 50(July 2001), 635 – 646.

[4] Wang C. F. and Sahni S., 1998. Basic operations on the OTIS-Mesh optoelectronic computer, IEEE Trans. on Parallel and Distributed Systems 9(Dec. 1998) 1226–1998.

[5] Wang C. F. and Sahni S., 1998. BPC Permutations on the OTIS-Hypercube, Optoelectronic Computer, Informatica, 22(3).

[6] Jana P. K. and Sinha B. P., 2006. An Improved parallel prefix algorithm on OTIS-Mesh, Parallel Processing Letters, 16, 429-440.

[7] Jana P. K.,2006 Polynomial Interpolation and Polynomial Root Finding on OTIS-Mesh, Parallel Computing, 32(4), 301-312.

[8] Lucas K. T. and Jana P. K., 2009. An Efficient Parallel Sorting Algorithm on OTIS Mesh of Trees, Proc. IEEE Intl. Advance Computing Conference , (6-7 March, 2009), Patiala, India, 175-180.

[9] Lucas K. T., Mallick D. K. and Jana P. K., 2008. Parallel algorithm for conflict graph on OTIS triangular array, Lecture Notes in Computer Science, 4904, 274-279.

[10] Rajasekaran S. and Sahni S., 1998. Randomized routing selection, and sorting on the OTIS-mesh, IEEE Transaction on Parallel and Distributed Systems, 9, 833-840.

[11] Wheelwright S. C., and Makridakis S., 1980 Forecasting Methods for Management, John Wiley and Sons.

[12] Jana P. K., Sinha B. P., 1997. Fast Parallel Algorithms for Forecasting, Computers Math. Applic. 34(9) 39-49.

[13] Evans D.J. and Gusev M., 1994. New linear systolic arrays for digital filters and convolution, Parallel Computing 20 (1), 29-61.

[14] Nassimi, D., and Sahni, S., 1979. Bitonic sort on a mesh-connected parallel computer, IEEE Trans. Comput. C-28(1), 2-7.