Event Detection In A Fully Distributed Wireless Sensor Network using Message Passing Interface and Open Multi-processing

Priscilla Ai Ching Tha
Monash University
Sunway, Malaysia
ptha0007@student.monash.edu

***Abstract*—A fully distributed wireless sensor network (WSN) is achievable by implementing MPI which address the memory limitation of an individual sensor nodes. Further solidifying the IPC architecture by arranging the nodes in nearest neighbors control model allows better bandwidth utilization. Additionally, encryption and decryption schemes are added into the architecture and accompanied with OpenMP to improve performance and efficiency.**

Keywords—event detection, wireless sensor node, OpenMP, MPI, inter-process communication, distributed computing, parallel processing.

I. INTRODUCTION

Parallel computing is the state of executing multiple processes simultaneously. The paradigm expanded its influence when frequency scaling failed to improve performance which led to utilization of multi-core processors. Furthermore, the application of parallelism in this system not only improve performance but also efficiency by applying inter-process communication (IPC). A wireless sensor network (WSN) perfectly simulates the environment by having the sensor nodes act as the processors.

WSN generates data from surrounding information then sends to a higher-level node for processing. It is typically employed in smart technology, search and rescue mission and geographical predictions[1]. However, these nodes have very limited processing power and memory capacity[1] which becomes the motivation to redefine its architecture in a distributed memory system. The architecture is possible with IPC and realized using Message Passing Interface (MPI) library in C language[2].

The simulation aims to recognize the possible issues in the decided IPC protocol design with 20 nodes and a base station. A grid of size 4x5 organizes the 20 nodes in question. Following the fatal attack of WannaCry and on Equifax, the encryption process is significant to obey data confidentiality and data integrity before passing the message to adjacent nodes and the base station. Parallelisation is realized using Open Multi-processing (OpenMP) to increase performance and efficiency of each node[2] during the computationally intensive encryption process.

II. IPC PROTOCOL DESIGN

A. Topology Design

The network takes on the 4x5 rectangular grid arrangement in nearest neighbours control manner with the last node as the base station (Figure I). Each node is connected to the minimum number of nearest neighbours[3]. Subsequently, MPI allows for distributed memory approach where memory allocated for each node are independent of other nodes. This approach addresses the limited memory capacity of sensor nodes.

0	**1**	**2**	**3**	**4**
5	**6**	**7**	**8**	**9**
10	**11**	**12**	**13**	**14**
15	**16**	**17**	**18**	**19**

Figure I Topology Design.

The grid is created with MPI_Cart_create with period and reorder set to false. Setting period to false prevents each node from communicating with nodes that are not directly adjacent to it (henceforth nearest neighbours)[3] whereas, setting reorder to false keeps each node rank in the MPI backend. A call to MPI_Cart_shift with the correct parameter allows nodes to retrieve information such as rank of their directly adjacent nodes.

B. Message Passing System

Message passing is also handled by MPI which guarantees portability as MPI standard is supported in different platforms. The network uses asynchronous communication mode between nodes using MPI_Isend and MPI_Irecv. Consequently, it reduces the risk of deadlock aside from ensuring data exchanges[4]. Combined with MPI_Wait and/or MPI_Waitall increases concurrency compared to MPI_Send and MPI_Recv as exchanges can happen regardless of sent or received acknowledgement.

On the other hand, encryption is necessary to fulfil the security feature of a good message passing system. The exchanges are also authenticated from both the sender's and receiver's end as part of the MPI_Isend and MPI_Irecv parameters. OpenMP here allows for thread-level parallelism by dividing each chunk of characters to the number of threads for encryption. Such shows the partition between threads using the following formula.

$$Characters\ per\ thread = \frac{Number\ of\ characters\ in\ string}{Number\ of\ thread} \quad (1)$$

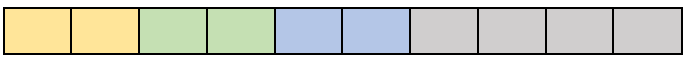

Figure II Division of characters per thread when number of characters = 10.

However, there may be the case where the characters cannot be divided equally among the threads. The last few characters are then dealt by the last thread in id to compute. It may not matter which thread computes the remaining characters, unlike the computation of the Mandelbrot set where workload distribution may be unequal, as the encryption calculation is constant throughout the characters in a word.

C. Encryption and Decryption Scheme

The Caesar cipher scheme is used in this simulation for simplicity and emphasize focus on the speed up factor when computation task is divided to multiple processors and/or threads. It is a symmetric encryption scheme where both the sender and receiver have decided to a prior secret key to encrypt and decrypt their messages[5]. Caesar cipher is a polyalphabetic cipher that uses the ascii values of the characters of a secret key to shift the characters of the plaintext.

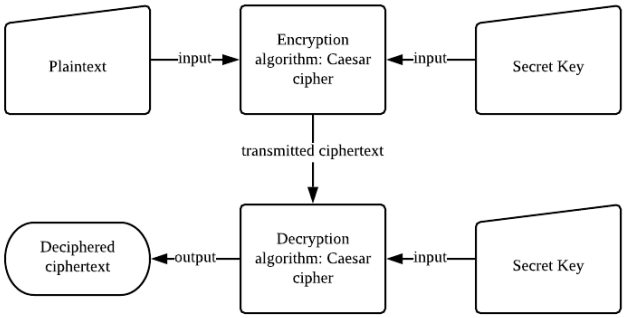

Figure III Symmetric encryption scheme

Given the letter ‘y’ (121 - 97), the operation is done by adding the value of letter ‘l’ (108 - 97) of the secret key to the value of letter ‘y’ which results in 35. The result is wrapped around 26 (35 mod 26) to ensure the ciphertext comprised of letters because the English language has 26 alphabets. The decryption operation is done by subtracting the value ‘l’ (108 - 927) of the secret key from the value of the cipher letter. Then adding 97 + 26 if it results in negative or 97 if it results in positive to obtain the ascii value.

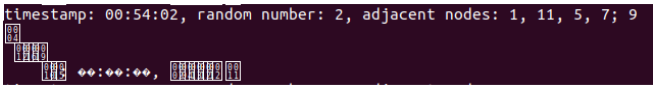

Figure IV The string excerpt before and after encryption.

To summarise, each node generates a random number which is encrypted before sent adjacent nodes and decrypted upon receiving. Should a node receive a common random number from at least three of their directly adjacent nodes, an ‘event’ is triggered which information is sent to the base station. Figure V illustrates the simulated IPC architecture with fully distributed WSN which mirrors its practicality in the real world.

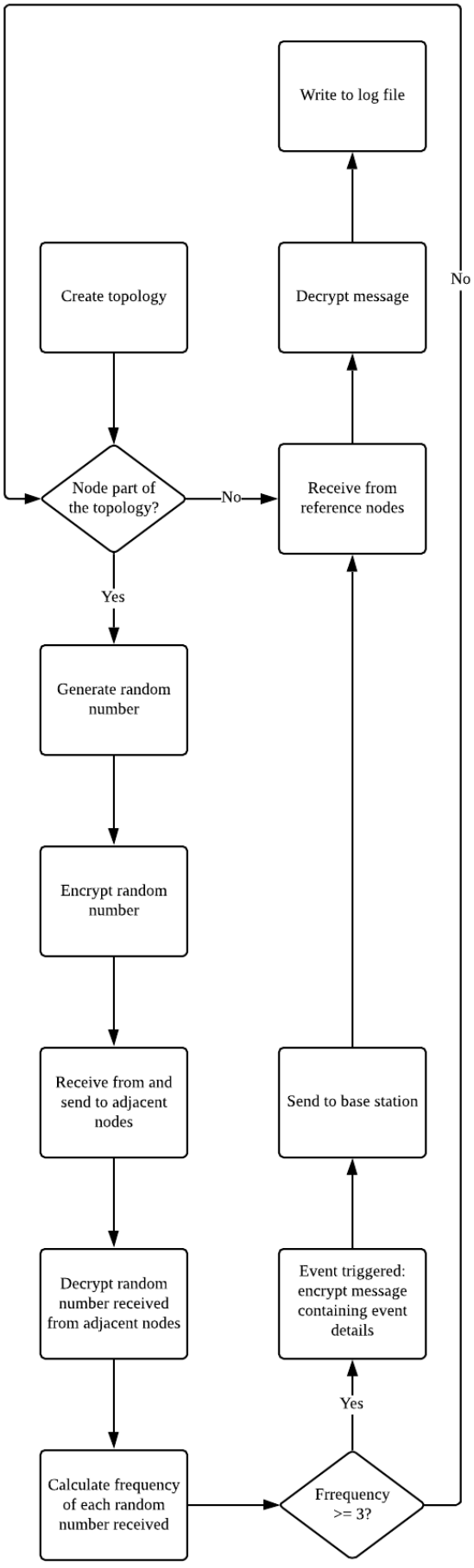

Figure V WSN IPC architecture.

III. RESULTS & DISCUSSION

The values recorded in the log file at the end of the simulation described in Section II is recorded in Table A in the appendices. Subsequently, the following equation is used to calculate the experimental speed up factors of the encryption algorithm when thread-parallelism was enabled using OpenMP.

$$S(p) = \frac{t_s}{t_p} \quad (9)$$

with

t_s = execution time using one thread

t_p = execution time using multiple threads

Table I shows the calculated experimental speed up factors S(p) for the 50th and 100th iteration.

TABLE I. EXPERIMENTAL SPEED UP FACTOR

Speed up factor, S(p)	
Number of threads, t = 4	
Iteration	
50	**100**
5.274303	1.290066

The values in Table I exceed one which indicates the increase in speed when parallelising the encryption. However, these speed up factors are obtained after five loop of encryptions whereas the serialised encryption outperforms the parallelised encryption when run once. The difference in hypothesis may be due to the overhead from establishing the parallelisable region. This happened during the forking of the master thread to multi threads and, later joining the multi threads contributes to the latent overhead.

Furthermore, thread parallelism is implemented in shared memory, hence, there is also the overhead from allocating memory during forking and deallocating memory during joining. The positive result obtained after five loop of encryptions shows that thread parallelism improves performance and efficiency for batch and computationally intensive programs. Justified, the encryption and decryption scheme used in the simulation only performs simple mathematical calculations whereas the most common encryption scheme practically, the Advanced Encryption Standard (AES) algorithm computationally complex.

In accord to the simulation, the asynchronous communication mode records low responsive property as the standard measure of usability for an event-driven program. Figure VI illustrates the average number of events to be 9% out of the total events if all nodes detected an event at each iteration. OpenMP then may not be suitable for this type of program as there are no similar functionality to MPI_Waitall or MPI_Wait thus, the master thread participating in the parallelism must wait for the other threads to finish computing before joining the threads into one again[6]. It may well slow down the program.

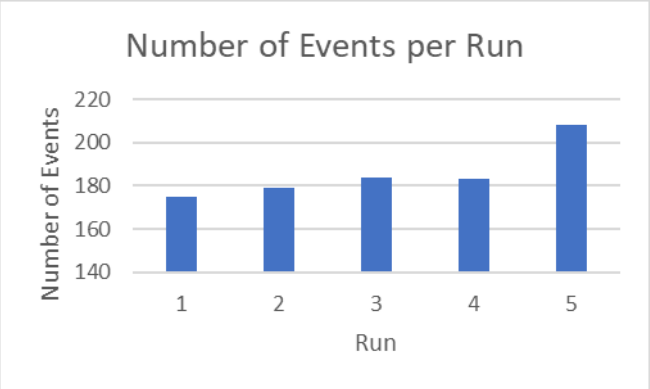

Figure VI Number of events per run

Finally, the messages sent could well be over the maximum transmission unit (MTU) of the network bandwidth. The messages then need to be broken down into packets before sending which is sent orderly. OpenMP could instead parallelised the sending process after the messages are broken down into packets by sending them simultaneously rather than in an orderly manner as it now complies to the bandwidth. Additionally, this proved thread parallelism using OpenMP reduces latency as smaller MTU values prevents network delay (Figure VII and Figure VIII). This must certainly accompany with OpenMP parallelisation at the receiving end as well.

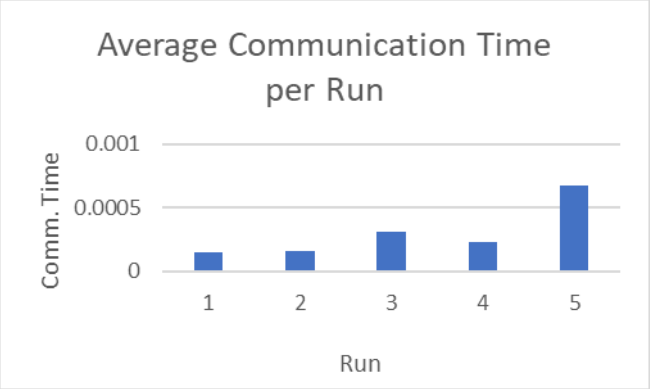

Figure VII Average Communication Time per Run

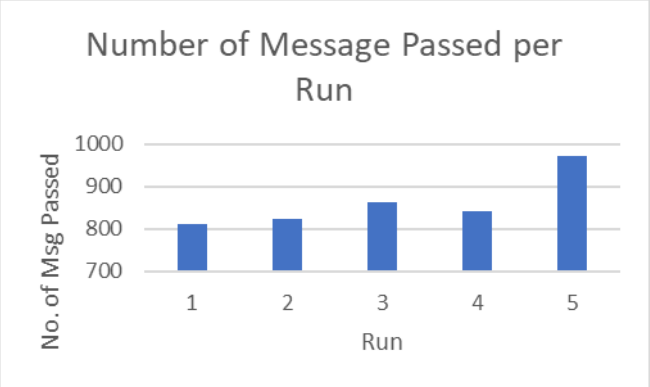

Figure VIII Number of Message Passed per Run

IV. CONCLUSION

Therefore, the IPC protocol design is implemented using MPI and included encryption and decryption process with OpenMP at encryption addresses the main limitation of a wireless sensor network. The distributed memory system achieved from using MPI will increase the memory capacity of each individual nodes. Combined with a solid encryption and decryption scheme, the message passing system in the IPC architecture guarantees data confidentiality and data integrity. Thread parallelism using OpenMP counters the computationally intensive nature of encryption schemes which satisfies the responsiveness property of an event-detection program aside from employing the asynchronous communication mode. It will clearly boost the performance of individual nodes as observed from the speed up factor obtained.

REFERENCES

[1] Farmani, M., Moradi, H. and Asadpour, M. (2012). A hybrid localization approach in wireless sensor networks using a mobile beacon and inter-node communication. *2012 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*.

[2] Jin, S., Chen, Y., Wu, Di. and et. al. (2015). Implementation of Parallel Dynamic Simulation on Shared-Memory vs. Distributed-Memory Environments. *2015 IFAC (International Federation of Automatic Control).*

[3] M. Gerharz, C. de Waal, P. Martini and P. James, "A Cooperative Nearest Neighbours Topology Control Algorithm for Wireless Ad Hoc Networks", *Telecommunication Systems*, vol. 28, no. 3-4, pp. 317-331, 2005.

[4] D. Galinec and L. Luić, "Asynchronous Message-Passing and Inter-Application Communication Software for Process Improvement in Complex Systems", *International Journal of Knowledge-Based Organizations*, vol. 4, no. 4, pp. 36-50, 2014.

[5] W. Stallings and L. Brown, *Computer security*. .

[6] X. Fan, O. Sinnen and N. Giacaman, "Towards an Event-Driven Programming Model for OpenMP", *2016 45th International Conference on Parallel Processing Workshops (ICPPW)*, 2016.