Implementation and Analysis for a parallelized distributed wireless sensor network with Message Passing Interface

written by Takayuki Kimura

19/10/2019.

Monash University

Malaysia

tkim0004@student.monash.edu

Abstract—A wireless sensor network (WSN) is a communicational architecture that consists of a base node and corresponding sensor nodes, which is designed to observe environmental states based on the information gathered from all sensor nodes. With the increase of its applicability to various situations, WSN needs to have more efficient and adaptable implementation to deal with a various type of problems or dataset. Thus, the aim on this report is to experiment the parallelization of WSN in order to achieve computational improvement as well as flexibility of the WSN with experimental methods such as sliding window approach. In order to guarantee the security of the messages between nodes, Vigenere encryption is applied.

Keywords—Wireless sensor network, MPI, parallel computing, sliding window, Vigenere cipher.

I INTRODUCTION

In this report, it aims to apply inter-process communication system (IPC) to WSN with Open MPI structure and analyse its functionality and implement how it is done. IPC plays an important role of supporting distributed computing architecture providing inter process communication in different clusters (Bagchi, 2014). Hence, IPC is used to implement WSN with OpenMPI parallelization.

In basic WSN mechanism following the assignment specifications, sensor nodes in WSN pass random values to adjacent nodes and if more than 3 collected values are same, the node has to send a message including details of the event to base node. In order to check the state of event, sliding window system is applied, which will be explained below section. In terms of security purpose, Vigenere cipher is utilized to encrypt messages passed between processors. This report also attempts speed up with parallelized encryption and decryption codes from serial code. This process is implemented with Open Multi processing (OpenMP) which is an API that provides a platform of parallelism in C language.

II Design scheme of IPC

A. illustration and description of the IPC architecture

IPC architecture is constructed using MPI_cart_create that is MPI function to create new communicator by building a form of matrix with processors in the communicator. Figure 1 represents how the sensor nodes in the group are communicated. Figure 2 shows the flow of the sensor node. In order to send and receive a message between nodes, MPI_Isend and MPI_Irecv are utilized.

MPI_Isend provides non-blocking interaction for passing a message, on the other hand, MPI_Irecv with MPI_wait implements blocking communication as it waits for the message to be arrived properly. Hence, the communication in the architecture is a hybrid communication between non-blocking and blocking. However, it is still faster than complete blocking implementation as sending itself does not have to wait.

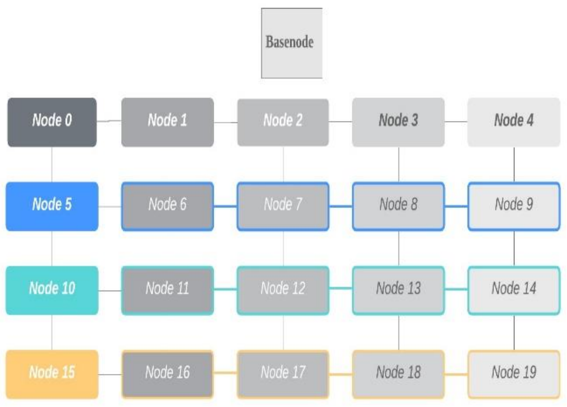


Fig 1 WSN architecture using MPI_Cart_create

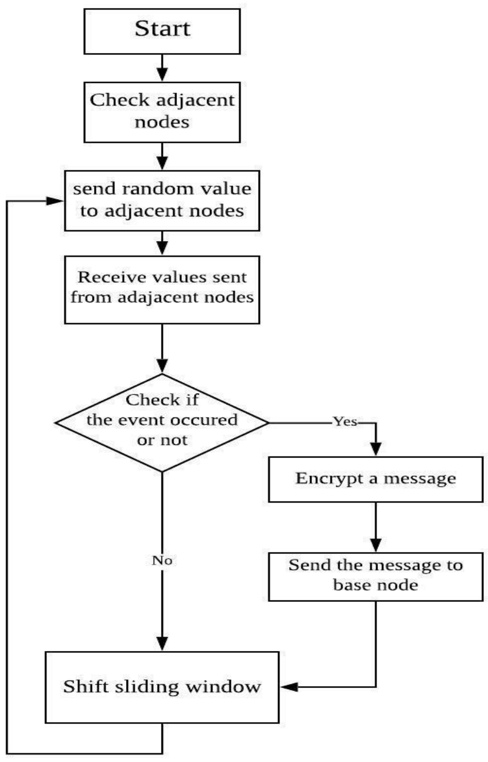

Start
Check adjacent nodes
send random value to adjacent nodes
Receive values sent from adajacent nodes
Check if the event occured or not
Yes
Encrypt a message
Send the message to base node
No
Shift sliding window

Fig 2 Flowchart of sensor node

IPC architecture in WSN also has to take its weakness into an account. Sharma et al. (2010) pointed out that generally sensor node has less capacity and resource than main node in WSN, thus the waste of these resources should be avoided. In order for that, sliding window approach. This method is implemented to identify subsequent values that satisfy the event requirement as the image below (Figure 3) represented. The frequency of the events is basically quite unstable as it is depending on the random value and it is being created in a moment. To improve the accuracy and stabilize the occurrence of the events, sliding window is one of the useful technique.

iteration	i = 8	i = 9	i = 10
node1	10	9	3
node5	9	11	6
node7	10	12	8
node11	8	14	10

Fig 3 an example of sliding window

B. Description of the applied encryption/decryption algorithm with OpenMP

For the cipher of message, Sharma et al. (2010) stated that confidentiality of passed messages should be maintained in WSN protecting the information from malicious attacks due to poor security. To solve this problem, Vigenere cipher is used in this report. This encryption simply encrypt character by character by substituting it based on the corresponding key as shown in the below image (Figure 4). To decrypt the encrypted string, it needs a same key used in encryption to know how many times a character needs to be shifted back to obtain the original character.

Original text	Key	distance	enxrypted text
H	A	0	H
E	B	1	F
L	C	2	N
L	A	0	L
O	B	1	P

Fig 4 Vigenere Encryption

The main reason why this cipher is chosen is that firstly it is designed for encrypting a string, secondly its availability and flexibility of the code to implement OpenMP with it. Thirdly, its safety in terms of security of encryption is still higher than other simple encryption such as Caesar's cipher in contrast to its simplicity of the encoding process. Bhateja et al. (2015) mentioned that the feature of Vigenere that each original letter has several different decrypted letter increase the difficulty to do cryptanalysis.

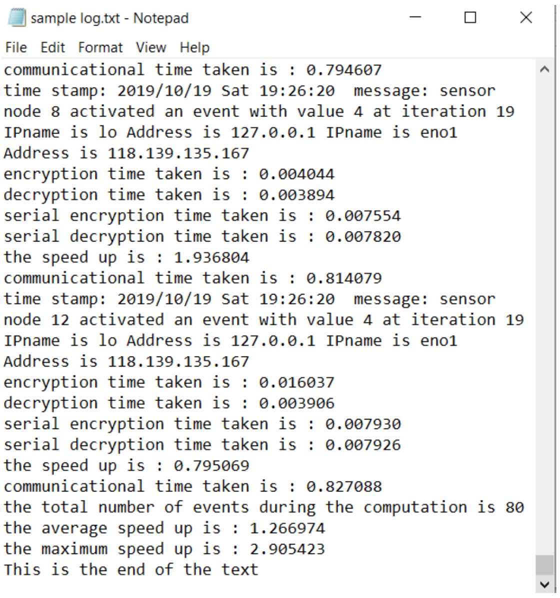
The code implemented in the assignment is retrieved from https://www.thecrazyprogrammer.com/2017/08/vigenere-cipher-c-c.html.

C. Speed up of parallelized cipher with OpenMP

As mentioned above section, encryption is used for security purpose. However, encrypting and decrypting all messages passed between processors may affect the computational time. Thus, parallelizing the cipher is attempted with OpenMP.

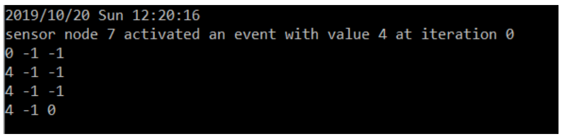
III RESULTS AND DISCUSSIONS

In this section, the tabulated results are represented to discuss the performance of the implemented code. As the sample log file shows (Figure 5), the details of the events are recorded. The number of iterations is set as 20. The number of reported messages between sensor node and base node when an event occurred is same with the number of events as it sends only 1 single string as a message per event. All the required time such as encryption and decryption time for both parallel and serial codes respectively. At the end of the computation, it also records the average speed up and the maximum speed up happened in the iterations. For the further details of the nodes, the state of the sliding window array is displayed in the command line (Figure 6). As far as it can be seen, the implementation works with WSN and does not have any serious issues.



```
communicational time taken is : 0.794607
time stamp: 2019/10/19 Sat 19:26:20  message: sensor
node 8 activated an event with value 4 at iteration 19
IPname is lo Address is 127.0.0.1 IPname is eno1
Address is 118.139.135.167
encryption time taken is : 0.004044
decryption time taken is : 0.003894
serial encryption time taken is : 0.007554
serial decryption time taken is : 0.007820
the speed up is : 1.936804
communicational time taken is : 0.814079
time stamp: 2019/10/19 Sat 19:26:20  message: sensor
node 12 activated an event with value 4 at iteration 19
IPname is lo Address is 127.0.0.1 IPname is eno1
Address is 118.139.135.167
encryption time taken is : 0.016037
decryption time taken is : 0.003906
serial encryption time taken is : 0.007930
serial decryption time taken is : 0.007926
the speed up is : 0.795069
communicational time taken is : 0.827088
the total number of events during the computation is 80
the average speed up is : 1.266974
the maximum speed up is : 2.905423
This is the end of the text
```

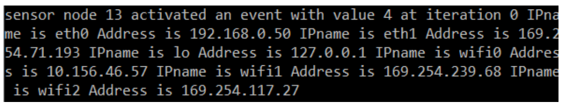
Fig 5 Sample log text



```
2019/10/20 Sun 12:20:16
sensor node 7 activated an event with value 4 at iteration 0
0 -1 -1
4 -1 -1
4 -1 -1
4 -1 0
```

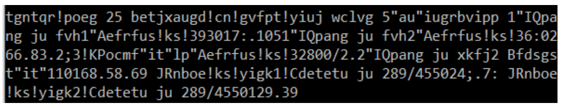
Fig 6 Sample displayed information on Terminal

For the encryption part, the below two images show sample plain text and sample cipher text after encryption. It can be said that it is well decrypted and hardly be decoded without a key. However, it is still simple encryption as compared to complicated cipher such as advanced encryption standard (AES).



```
sensor node 13 activated an event with value 4 at iteration 0 IPna
me is eth0 Address is 192.168.0.50 IPname is eth1 Address is 169.2
54.71.193 IPname is lo Address is 127.0.0.1 IPname is wifi0 Addres
s is 10.156.46.57 IPname is wifi1 Address is 169.254.239.68 IPname
 is wifi2 Address is 169.254.117.27
```

Fig 7 sample plain text



```
tgntqr!poeg 25 betjxaugd!cn!gvfpt!yiuj wclvg 5"au"iugrbvipp 1"IQpa
ng ju fvh1"Aefrfus!ks!393017:.1051"IQpang ju fvh2"Aefrfus!ks!36:02
66.83.2;3!KPocmf"it"lp"Aefrfus!ks!32800/2.2"IQpang ju xkfj2 Bfdsgs
t"it"110168.58.69 JRnboe!ks!yigk1!Cdetetu ju 289/455024;.7: JRnboe
!ks!yigk2!Cdetetu ju 289/4550129.39
```

Fig 8 sample cipher text

Regarding to the speed up of encryption and decryption, the equation (1) is applied to calculate it. As a result of the iterations based on the log file shown (Figure 5), the average speed up is 1.266974 (Figure 5) that is mostly more than 1 though each speed up is still inconsistent. However, the speed up of parallel cipher was slower than the serial at first. The main 2 components that accelerate a computational time is using sleep() function and limiting the number of the assigned processors for the task. For sleep function, it is to let the processor sleep for a given time. My analysis is that serial code does not benefit from resting the processor itself as much as the parallelized code as serial code does not critically need to wait for other processors to complete its assigned tasks. Especially, the case where the actual amount of computation is not as huge as parallelization is designed for, it may cause speed down even. That is why, the number of the processors to be assigned for OpenMP is set as 2 for encryption and decryption parallelization. This constraint can contribute on reducing the communicational time between the assigned processors as compared to splitting computation to 4, 8 or 16 processors though it still can benefit from parallelization.

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

t_s = serial code computational time

t_p =parallelized code computational time

(1)

IV CONCLUSIONS

In conclusion, implementation of IPC in WSN works well with OpenMPI which requires well organized distributed architecture to allow each processor to communicate with the corresponding nodes. The proposed sliding window approach contributes on stabilize the frequency of the number of events with its functionality. The suggested Vigenere cipher manipulate encryption and decryption with OpenMP and it achieved more than 1 speed up using sleep function providing processors to rest and it can compute other assigned tasks. For future work, the method of how to maintain the consistency of speed up without sleep function and should be analysed and practiced. In addition, whether OpenMP parallelization can be applied to highly complex encryption or not could also be important to discover the flexibility of the parallelization.

References

[1] Bagchi, S. (2014). The Software Architecture for Efficient Distributed Interprocess Communication in Mobile Distributed Systems. Journal of Grid Computing, 12(4), 615–635. https://doi.org/10.1007/s10723-014-9304-9

[2] Bhateja, Ashok K, Bhateja, Aditi, Chaudhury, Santanu, & Saxena, P.K. (2015). Cryptanalysis of Vigenere cipher using Cuckoo Search. Applied Soft Computing Journal, 26, 315–324. https://doi.org/10.1016/j.asoc.2014.10.004

[3] Sharma, R., Chaba, Y., & Singh, Y. (2010). An IPC key management scheme for Wireless Sensor Network. 2010 First International Conference On Parallel, Distributed and Grid Computing (PDGC 2010), 251–255. https://doi.org/10.1109/PDGC.2010.5679906