Investigating the Run Time Behavior of Distributed Applications by Using Tiny Java Virtual Machines with Wireless Communications

Tsuyoshi Miyazaki, Takayuki Suzuki, and Fujio Yamamoto

Department of Information and Computer Sciences, Kanagawa Institute of Technology,
1030 Shimo-ogino, Atsugi-shi, Kanagawa, 243-0292 Japan
{miyazaki,suzuki,yamamoto}@ic.kanagawa-it.ac.jp

Abstract. From the viewpoint of programming education, distributed application programs carried out in a small JAVA machine group were considered. These computers are equipped with radio communication facility, multi-thread function, LEDs and various sensors. Parallel genetic algorithms and distributed search problems were targeted for the study here. About the latter, a detailed implementation method and the result of the experiment are shown. In such a computing environment, it was understood that the internal behavior and the data communication in the distributed application were easy to be grasped by an effect of visualizing them by the physical interface.

Keywords: Physical Computing, Distributed Computing, Software Education.

1 Introduction

In the present age when Web-related technical development is remarkable, the importance of the software education at the university rises still more. Above all, it is necessary to teach the basic technology of distributed systems or distributed applications practically. Conventionally, in this field, lectures on a basic concept and the basic technology of the distributed applications were performed in the classroom, whereas the practice that uses computers seems not to be performed very much. The one of the reasons is that there is not the environment where each person accesses a lot of PCs except one's PC. The second reason is that grasping the cooperative activities among PCs is not so easy, and consequently it is difficult to have overall image of the processing performed by that distributed application.

Recently small computer SunSPOT [5,6] equipped with JAVA virtual machine attracts attention. SunSPOT possesses an acceleration sensor and an illumination sensor, and can execute general JAVA programs on it. Therefore it is originally thought that it is a device to build a radio sensor network. However, as a notable thing, this device possesses a multi-thread function as well as a wireless communication function. These functions can be used in the JAVA language like the case of the normal PC. In other words, various network applications using multi-thread programming, socket communication and multicast can be made on this machine easily. Communication among

J.A. Jacko (Ed.): Human-Computer Interaction, Part I, HCII 2009, LNCS 5610, pp. 882–889, 2009.

SunSPOTs is enabled without being connected to the PC after an application program was developed. The power supply by the built-in battery lasts comparatively for a long time enough to demonstrate the applications anywhere.

By using radio communication facilities among SunSPOTs, various distributed applications can be performed. When it is compared with a general PC, the CPU performance and the transmission rate of SunSPOT are considerably low. Fortunately, such characteristics can be used effectively. In other words, an environment that can watch the behavior of the distributed application by slow motion is provided. SunSPOT is equipped with eight color LEDs and also can be attached an LCD display such as one in a mobile telephone. If they are used, the operation of the CPU and the movement of the transmission and reception can be grasped easily. Based upon this, utilization of SunSPOT for education of the distributed application technology is trying. In the following, distributed applications to work on a set of SunSPOTs developed this time are explained, and based on it, the possibility of the use by the future education is considered.

2 Distributed Application Programs for Educational Use

Two applications are taken up to teach the technology of the distributed programs in a computing environment mentioned above. The first is the parallel genetic algorithm, and second is the issue of search of the two-dimensional domain by Bentley [2]. About the former, only the aim and method are described, and the implementation will be reported in the near future. On the other hand, about the latter, the details on a method and the implementation are shown below.

2.1 Parallel Genetic Algorithms

The genetic algorithm (GA) has been used as convincing technique to solve the issue of complicated optimization for a long time. The potential concurrency of the processing attracted attention early and there are many studies on it [4,8]. By the parallel genetic algorithm, a population is divided into some partial groups, and each is allocated to each processor. Evolution processing is carried out independently in each group. The variety of the individual will be maintained in the whole population. However, in the evolution, it is very likely that each sub-population converges early to each local optimum status.

To prevent this problem, migration of individuals is necessary in the parallel genetic algorithm. In other words the information should be exchanged at a certain suitable period between sub-populations. Some individuals are moved to another sub-population. And let the individuals take an opportunity to do crossing-over with the individuals in the sub-population they arrived at. The individual to be migrated is limited to the elite who showed high fitness in its original sub-population. Because the elite evolved in a different way from the evolution performed in the sub-population it arrived at, a birth of child with new character is expected from the crossing-over with a native individual there. Such situation will develop new searching area for global optimized value.

Here, a problem to search for the combination pattern of the color was solved by such a parallel genetic algorithm. It is assumed that eight lamps are put to one line now. Each lamp can display 256 colors by changing each value of R, G and B. Then the combination of the color of eight lamps becomes the enormous number, that is, 256 to the 8th power (about 10 to the 19th power). It is the problem to find out one specific pattern among them. The difference between the answer pattern and the presumed pattern is assumed known as a value of the fitness function. Each subpopulation has 30 individuals. Each individual estimates one color combination pattern. From generation to generation, those individuals do crossing-over and the individuals having high fitness (near the correct color pattern) will survive. In this process, appropriate migration is important as shown above. By this migration, the stoppage to a local optimum as a whole population can be avoided.

2.2 Bentley's Searching Problems

The second application developed this time is a solution for the searching problem that Bentley, J. L. (1985) presented [2]. It is assumed that a two dimensional array of NxN contains the elements of plus and minuses numbers at random position. The problem is to find out the best rectangle among all possible rectangles covering the part of the array. The best rectangle here denotes the one that gives the maximum value resulting from the summation of all the elements contained in that rectangle. That maximum value is the final answer.

A naive algorithm. For example, in the case of an array A of 4x4 in Fig. 1, the grand total of the elements in the rectangle of the dot line frame becomes 3. On the other hand, the grand total of the elements in the rectangle of the black bold frame is 8, and this becomes the maximum. The empty rectangle that nothing includes is permitted, and, in that case, the sum of elements is considered to be a zero. Therefore, the maximum does not become the minus number. Generally, in this solution, computational complexity suddenly increases as size N of the array A grows big.

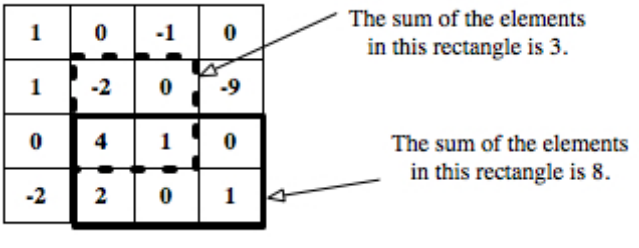

Fig. 1. Bentley's problem (for 4x4 array)

The most naive algorithm needs computational complexity of the order of N to the 6th. However, by some improvement, it can be reduced to the order of N to the 5th. The main part of this algorithm is shown below.

```
int maxsofar = 0;
int sum;
for (int is =0; is<N; is++){
  for (int js =0; js<N; js++){
    for (int ib =is; ib<N; ib++){
      sum = 0;
      for (int jb =js; jb<N; jb++){
        for (int i = is; i<=ib; i++) {
          sum = sum + A[i][jb];
          }
        if (sum > maxsofar) maxsofar = sum;
        }
      }
   }
}
```

An Optimized Algorithm. More effective algorithm is usable if much memory can be used. The main part of this algorithm is shown below. Here, the intermediate result of the calculation is stored in a work array W, and it is used effectively as a partial sum. This algorithmic computational complexity becomes the order of N cubed [1].

```
int globalmax =0;
int sum;
int i, j, k ;
for (j=0; j<N; j++){
  for (i=0; i<N; i++){
    sum = 0.0;
    for (k=i; k<N; k++){
      sum = sum + A[k][j];
      W[i][k] = max(0, W[i][k]+sum);
      globalmax = max(globalmax, W[i][k]);
     }
   }
}
```

However, because the SunSPOT machine does not have much quantity of memory, it was decided to use the algorithm of N to the 5th mentioned above. Here, the performance enhancement of the application is not necessarily assumed the main purpose. Therefore, this algorithm is useful enough as an exercise to grasp inside behavior of the distributed application.

3 Distributed Solution for the Bentley's Problem Using SunSPOTs

Here, an algorithm with the complexity of N to the fifth is adopted. And a JAVA program is developed that divides the search calculation into small parts and distributes them among several SunSPOTs. One SunSPOT is chosen as a host machine, and other SunSPOTs act as a worker that performs provided part of searching. The host sends appropriate partial search area to the worker that sent a "READY" message, and then waits for the searched result. The worker, which received a partial search area,

performs calculation and sends back the result (local maximum value) to the host, and then sends “READY" message again. During the computation with several workers, another new workers can be dynamically joined the computation. When all the results from the workers sent back to the host, the global maximum value should be taken as the final answer.

3.1 Observing Internal Processing and Data Transmission

In this distributed application, the host SunSPOT has two threads run simultaneously. The first thread receives “READY” from a worker, and then sends back partial search area to that worker. The second thread waits for the result that a worker calculated on the partial search area. Eight LEDs of SunSPOT is used to display the movement of this application. When a SunSPOT sends or receives data, LED-7 or LED-0 turns on blue respectively for a pre-defined period. While a SunSPOT performs calculation, four central LEDs turn on red, whereas while it is in “READY” they turn on green. Because a LCD device (a display part of a mobile telephone) is connected to the host SunSPOT, it is possible to display successive data sent from each worker and also the final answer. Fig. 2 shows the communications among a host and the workers.

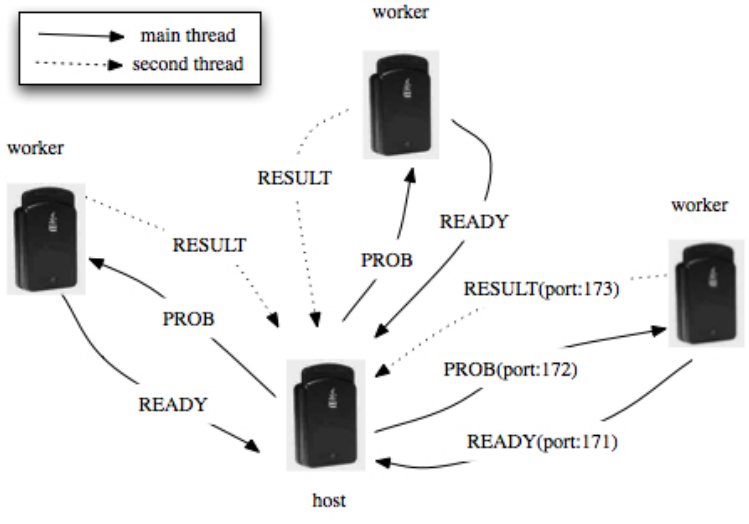

Fig. 2. Radio communications among SunSPOTs

Outline of the Host program. Pseudo-coding shows the outline of this method below. This illustrates the programs of the host SunSPOT. In this version, it is necessary to start this host program earlier than a worker.

```
public void run(){
  start listenReady() in the main thread;
  start listenResult() in the second thread;
 }
```

```
public void listenReady(){
  loop = true;
  while(loop){
    receive datagram at port 171;
    if(the message equals "READY"){
      turn on LED-0 for 100ms (receiving);
      extracts sender's address A;
      if(partial search area is still left){
        send a partial search area
          to the address A at port 172;
        turn on LED-7 for 100ms (sending);
      }else{
        loop = false;
        }
      }
    }
 }
public void listenResult(){
  loop = ture;
  while(loop){
    receive datagram at port 173;
    if(the message equals "RESULT" ){
      turn on LED-0 for 100ms (receiving);
      take that data as an local maximum;
      update the global maximum value;
     }
    if(all the answers have been received){
      loop = false;
     }
   }
 }
```

Outline of the Worker program. The following pseudo-coding illustrates the outline of the worker program on a SunSPOT. This worker can be added dynamically during processing with the host and other workers. The relationship among the host and workers are shown in Fig. 2.

```
public void run(){
  while(true){
    turn on green with LED2-5 (as "READY");
    receive datagram at port 172;
    if(it indicates a partial search area){
      if(the problem is to me){
        turn on LED-0 for 100ms (receiving);
        turn on red with LED2-5(as "BUSY");
        search the partial area (local max);
        send result to the host at port 173;
        turn on LED-7 for 100ms (sending);
        send "READY" with my address
          to the host at port 171;
```

```
            turn on LED-7 for 100ms (sending);
           }
        }
     }
  }
```

3.2 Observing the Behavior of the Application

With up to sixteen SunSPOTs (including host SunSPOT), demonstration of this application was performed in a classroom. Strong interest of students was attracted. Primarily, they can understand well the cooperative activities among JAVA machines that are small enough to put in their palm. They can clearly see which machine receives a problem (partial search domain) from a host, and when it goes busy or ready state. In other word, even in the case that the calculation (searching) completes instantly with a normal PC, they can observe steadily and slowly the activities when using SunSPOTs. The positional transparency in a distributed system can be presented simply by moving some of the machines running the application outside the door. Additionally they can understand some new machines join the calculation dynamically by turning on the power switch of them. Through these experiences they can be familiarized with distributed systems technology. It took 176 seconds when using one host and one worker in case of an array of 60x60. In contrast, a solution was obtained in 67 seconds when the number of workers was increased to three.

3.3 Considerations in Programming Education

The students may wrestle with the following problems after having had interest in this way, and it is thought that their software development ability can be improved. At first, in this application the amount of operations each machine takes care of may be unequal, depending on the division method of the search domain. This situation can be observed by the red lightning of LEDs, which denotes execution of operations. Students may have willing to do load balancing among workers. As a next problem, consider the case that a worker suddenly goes down while executing operations, for example by pushing the power button off. In this case, some of the expected result will never returned to the host forever. It may be understandable that some kind of transaction facility should be introduced. Namely, recovering mechanism that another worker can take care of the missing problem should be necessary.

There exits a bottleneck in the communication among the host and many workers. This problem may also be observable by frequent lightning of blue LEDs of the host when a lot of extra new workers are added. For that problem, a method that the local maximum value should be reserved in a worker by communications between workers in the sub-group will be thought about. By this method, only few workers send a result to a host and consequently communication traffic should be reduced.

4 Conclusion

By being familiar with a distributed application, actually holding portable JAVA machines in a hand in this way, most students probably wrestle with the acquisition of

the software technology based on multi-threads and network communications eagerly. The environment of physical computing except SunSPOT used this time is also regulated well recently [3]. With it, the application to education [7] would largely be extended.

References

1. Arisawa, M.: Algorithms and Their Analysis, Corona Publishing (1989) (in Japanese)
2. Bentley, J.L.: Programming Pearls. Addison-Wesley, Reading (1985)
3. Estrin, D., Culler, D., Pister, K., Sukhatme, G.: Connecting the Physical World with Pervasive Networks. Pervasive Computing (2002)
4. Juille, H., Pollack, J.B.: Massively Parallel Genetic Programming. In: Advances in Genetic Programming II, MIT Press, Cambridge (1996)
5. Simon, D., Cifuentes, C., Cleal, D., Daniels, J., White, D.: Java(TM) on the Bare Metal of Wireless Sensor Devices – The Squawk Java Virtual Machine, VEE, Ottawa (2006)
6. Smith, R.B.: SPOTWorld and the Sun SPOT. In: Proceedings of the 6th international conference on Information processing in sensor networks, pp. 565–566 (2007)
7. Yamamoto, F.: An Educational JavaSpaces Programming Environment with Phidgets Devices. In: Supplementary Proceedings of The 15th International Conference on Computers in Education, pp. 1–2 (2007)
8. Yamamoto, F., Araki, T.: A Parallel Genetic Algorithm with Diversity Controlled Migration and its Applicability to Multimodal Function Optimization. In: Proc. of the AFSS 1998, pp. 629–633 (1998)