The Fourteenth International Symposium on Artificial Life and Robotics 2009 (AROB 14th '09),
B-Con Plaza, Beppu, Oita, Japan, February 5 - 7, 2009

Development of Programming Language Espace and Its Application to Parallel and Distributed Evolutionary Computation

Takehiko IWAKAWA Satoshi ONO Shigeru NAKAYAMA
Deptartment of Information and Computer Science, Faculty of Engineering
Kagoshima University
1-21-40, Korimoto, Kagohima 890-0065, Japan
{sc100007, ono, shignaka}@ics.kagoshima-u.ac.jp

Abstract

This paper proposes a programming language called "Espace" for parallel and distributed computation. In general, it is difficult to code a distributed, parallel program due to multi threading, message passing, managing clients, and so on. Espace involves a few simple syntax rules added to Java. Developers do not need to know how to write a parallel, distributed program source code in detail. This paper applies Espace to parallelize an evolutionary computation program and shows that the Espace compiler enables to convert an evolutionary computation program written in Java into a distributed, parallel system by adding a few words to the program.

1 Introduction

General evolutionary computation algorithms are stochastic, population-based generate-and-test algorithms which require huge computational cost. They are not an essential way to solve combinatorial explosion but practical resources for cost reduction to develop an evolutionary computation program as a distributed, parallel system.

Distributed models of evolutionary computation are mainly classi ed into two models: island model and evaluation-distributed model. Island model involves some subpopulations, and a few solution candidates immigrating between subpopulations at de ned intervals. Island model is a rough-grained model which can be parallelized by subpopulations and has an advantage in control of search diversi cation. Evaluation-distributed model is a ne-grained parallel and distributed model appropriate for problems which require much computation time to evaluate a solution candidate because evaluation of solution candidates can easily be parallelized.

In the case using the existing libraries or languages, although the above models are e ective to reduce computation time, developers have to code for parallelization procedures such as multi threading, fault tolerance and so on

In general, it is difficult to code a distributed, paralleled program due to multithreading, message passing, managing clients, and so on. To reduce the development cost of distributed, parallel system, some libraries for programming languages producing shared memory or message passing functions are used[1]. There are also other methodologies using compilers, such as programming languages, macros, directions to compilers, which are speci ed to address these problems reducing the cost of distributed, parallel system development [2, 3, 4, 5, 6].

Using these libraries or languages is unfortunately not so easy because of their complicated syntax and discords between base language and extended module. Construction of distributed runtime environment is also a cumbersome procedure. For example, a developer has to set up an operating system on personal computers only for parallel, distributed computation as a cluster, or install middleware to maintain the system's security level[7].

This paper proposes a programming language for parallel and distributed computation called "Espace"[8], and shows that the Espace compiler enables to convert evolutionary computation program written in Java into distributed, parallel system by adding a few words of Espace to the program.

2 Programming Language Espace

2.1 Design policy

Design policies of Espace language proposed in this paper are as follows:

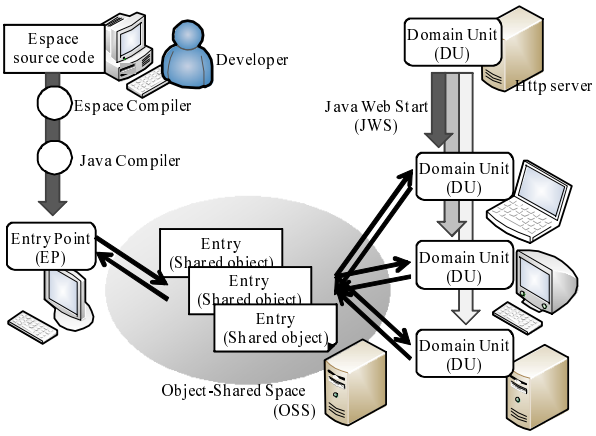

Figure 1: System structure of Espace runtime environment

1. Espace utilizes simple, few syntax rules.
2. Espace abstracts distributed and parallel processing function.
3. Espace produces a distributed runtime environment which can be built up easily.

Espace involves a few simple syntax rules added to Java. The added syntax rules are mainly invocation rules of distributed method and access rules to a distributed object. Therefore, developers do not need to know how to write a parallel, distributed program source code in detail.

2.2 System structure

The distributed runtime environment of Espace is made up of an Object-Shared Space(OSS) server and its clients. Clients communicate with each other via reading or writing shared objects called Entry. The system structure of Espace is shown in Fig. 1. An Espace program is compiled by an Espace Compiler into a Java program called Entry Point(EP). At the same time, EP is conducted with anonymous computation units called Domain Units(DUs) via Java Web Start[9]. DU works as computation resource or as a holder of distributed object. The only operation users need to do is accessing an HTTP server for deploying the DU.

2.3 Syntax rules

Developers can allocate or access distributed objects by inserting the keyword "remote", which indicates that a field modified by the keyword has a remote attribute (remote field) and can be accessed by any client joining the Espace network.

```
distribute-statement =
'distribute'
'{' block-statement* '}'
['doing' '{' block-statement* '}' ] ;
```

Figure 2: EBNF definition of distribute statement.

Invocation syntax rules of distributed method enables developers to write distributed, parallel processing program with extremely reduced cost; adding the keyword "espace" to distributed methods and invoking them in a "distribute" statement. The EBNF definition of a distribute statement is shown in Figure 2. In a "distribute" section, a method modified by the keyword has an espace attribute (espace method), and then each espace method is invoked on parallel threads. In addition, it is not necessary for developers to adjust task granularity of parallelization and implement fault tolerance functions on connection or unforeseen situation. Processes defined at "doing" section are also executed at one time on EP .

The keyword can be used to modify a field. Field instances modified by the keyword is shared by DUs while running a "distribute" statement, and the fields become able to be used in espace method.

3 Application to Image Filter Generation by Genetic Programming

3.1 Overview

Over the past few years, studies have been performed on image filter generation(IFG) with genetic algorithm (GA) or genetic programming (GP)[10]. The existing methods generate various filters by combining existing primitive filters. Although filters generated by GP show better output quality than filters generated by GA, chromosome size becomes huge in GP. Uncontrolled chromosome growth, called bloat, makes filter application time worse, and makes it impossible for users to analyze and modify generated filters. The authors of this study propose a GP based image filter generation method that tunes numeric parameters of the primitive filters and restrains surplus chromosome growth[10].

3.2 Process Flow

GP for IFG recieves source and objective image sets as input, and outputs a generated image lter. Each node in a tree-structured chromosome corresponds to a basic lter such as smoothing, edge extraction, inverse, and so on. Some of the basic lters have a parameter such as binarization, addition, maximization and so on, and the parameter should be adjusted so that the basic lter works well.

The process ow of GP for IFG is basically an iteration of recombination of individuals by applying operators of selection, crossover, and mutation, evaluation of the individuals, and parameter adjustment in basic lters.

The parameter adjustment is conducted in a similar way to local search:

Step 1 Choose R_{PA} individuals randomly.

Step 2 Repeat following Step 3 through 4 S_{PA} times.

Step 3 For each individual i, choose a basic lter and generate N_{PA} o spring by changing the parameter of the chosen basic lter.

Step 4 Choose the best o spring and replace i by the best o spring.

In the parameter adjustment, R_{PA} S_{PA} N_{PA} times tness calculation must be conducted, and this is more time-consuming than all other processes in the iteration involving evaluation of all individuals.

3.3 Implemantation

We applied Espace to parallelization of GP for IFG. Here GP for IFG distributed by Espace is called as DGPIFG . In DGPIFG, EP distributes parameter adjustment process for one chromosome to DUs via OSS. DUs get chromosomes from EP via OSS and return the result to EP in every generation.

To parallelize existing GPIFG code with Espace, the following procedure is necessary.

Step1 Add the "espace" keyword to modify the method de ning parameter adjustment processes.

Step2 Add the "distribute" statement outside of the loop invoking the parameter adjustment method repeatedly.

Step3 Add the "espace" keyword to variables read in the parameter adjustment method.

The source code for DGPIFG is shown in Figure3. Method tuneParam is an espace method, and is accessed on a distribute statement. Method calFitnessN is also an espace method, and is accessed on method tuneParam. Each espace method is executed on DUs. Field orgFit, pixGs, pixSs, and weight are espace eld and are accessed on calFitnessN. Instances of these espace elds are assigned to OSS, and shared between DUs. In addition, it is necessary to change declared type of an espace eld to Object class of Java due to inhibited number of connections to OSS. If an espace eld were to be decleared as array, each element in the array would be written to OSS as Entry splitted by dimension, consequentry number of connections would extremely increase causing performance decrement in the case shown in Figure. 3. It is also nessesary to insert cast operation to espace eld. To run the system, the user starts OSS, DUs, and EP, in this order. Because DUs and EP automatically search and connect to OSS, users need not con gure network settings.

3.4 Evaluation

The Espace source code shown in Figure. 3 contains ve espace elds, six espace methods, and a distribute statement. The conversion from Java to Espace was done by adding 12 keywords and a block, changing 4 declared types and adding 24 cast operators. Although it was vexatious to add cast operators, the program structure almost was not changed. It, therefore, seems resonable to conclude that the source code of DGPIF is easy to understand for Java programmers.

4 Conclusions

We have proposed a programming language called Espace which involves simpli ed syntax rules for distributed parallel computation. The experimental result has shown that the Espace compiler enables to convert an evolutionary computation program written in Java into distributed, parallel system by adding a few words of Espace to the program.

In the future we plan to work up experimental implementations of distributed system written in Espace to evaluate usability of the language.

Acknowledgements

Part of this study was supported by IPA Exploratory Software Project.

```
class GPIF_gp{
  espace Object orgFit;//double[]
  espace Object pixGs;//Pixels[]
  espace Object pixSs;
  espace Object weight;//double[][][][]
  void ptNotE() {
    int num=(int)(nC*HIT_RATE);
    int[] targets=new int[num];
    IdvTree tmpElite=null;
    for (int i=0; i<num; i++) {
      targets[i]=(int)((nC)*Math.random()
          *0.99);
    }
    distribute{
      for (int i=0; i<num; i++) {
        tmpElite =tuneParam(popula[targets
            [i]], num);
        popula[targets[i]] = tmpElite;
      }
    }
    ...
  }
  espace IdvTree tuneParam(IdvTree elite,
      int candiNum,
    int prmNum, boolean isRplc) {
    calFitnessN(...);
    ...
  }
  espace IdvTree calFitnessN(IdvTree tree1
      ){
    double[][][][] w = (double[][][][])weight;
    Pixels[] pixs = ((Pixels[])pixGs);
    Pixels[][][] pixSsl = (Pixels[][][])pixSs;
    double[] orgFitl = ((double[])orgFit);
    double fitSum;
    int[][][][] outPix;
    fitSum = 0.0;
    if (!tree1.isCalculatedFlg()) {
      for (int k = 0; k < NUM_IMG; k++) {
        flt.initImg(pixSsl[k]);
        outPix = filtering(tree1.getChromo
            ());
        ...
      }
      tree1.setFitness(fitSum / inImgNum);
      tree1.setCalculatedFlg(true);
    }
    return tree1;
  }
  ...
}
```

Figure 3: Espace source code example.

References

[1] MPICH2. http://www.mcs.anl.gov/index.php.

[2] Open MP. http://www.openmp.org/drupal/.

[3] Omni OpenMP Compiler Project. `http://phase.hpcc.jp/Omni/`.

[4] Philippe Charles, Christopher Donawa, Kemal Ebcioglu, Christian Grotho , Allan Kielstra, Christoph von Praun, Vijay Saraswat, and Vivek Sarkar. X10: An Object-oriented Approach to Non-Uniform Cluster Computing. In *Proc. 20th Ann. ACM SIGPLAN Conf. Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 05)*, pages 519–538. ACM Press, 2005.

[5] Eric Allen, David Chase, Joe Hallett, Victor Luchangco, Jan-Willem Maessen, Sukyoung Ryu, Guy L. Steele Jr., and Sam Tobin-Hochstadt. The fortress language speci cation version 1.0, 2008.

[6] Takeshi KAMOGAWA and Takaya YUIZONO. A proposal of programming Language JavaPAN with Static Network Scope. *The transactions of the Institute of Electronics, Information and Communication Engineers. D-I*, 88(2):326–329, 2005.

[7] I Foster and C Kesselman. Globus: A metacomputing infrastructure toolkit. 11(2), 1997.

[8] Takehiko IWAKAWA, Satoshi ONO, and Shigeru NAKAYAMA. Development of a Programming Language Espace for Distributed Parallel Processing. *Transactions of ISCIE*, 19(7), 2006.

[9] Java SE Desktop Technologies - Java Web Start Technology. `http://java.sun.com/javase/technologies/desktop/javawebstart/index.jsp`.

[10] Masaki MAEZONO, Satoshi ONO, and Shigeru NAKAYAMA. Automatic Parameter Tuning and Bloat Restriction in Image Processing Filter Generation Using Genetic Programming. *Transactions of JSCES*, 8(20060021), 2006.