Modeling the Communication Behavior of Distributed Memory Machines by Genetic Programming*

L. Heinrich-Litan, U. Fissgus, St. Sutter, P. Molitor, and Th. Rauber

Computer Science Institute, Department for Mathematics and Computer Science,
Martin-Luther-Universität Halle-Wittenberg, D-06099 Halle (Saale), Germany
<*name*>@informatik.uni-halle.de

Abstract. Due to load imbalance and communication overhead the behavior of the runtime of distributed memory machines is very complex. The contribution of this paper is to show that runtime functions predicting the execution time of the communication operations can be generated by means of the genetic programming paradigm. The runtime functions generated dominate those presented in literature, till today.

1 Introduction

Distributed memory machines (DMMs) provide large computing power which can be exploited for solving large problems or computing solutions with high accuracy. Nevertheless, DMMs are still not broadly accepted. One of the main reasons is the costly development process for a specific parallel algorithm on a specific DMM. This is due to the fact that parallel algorithms on DMMs may show a complex runtime behavior caused by communication overhead and load imbalance. Thus, there is considerable research effort to model the performance of DMMs. This includes modeling the runtimes of communication operations with parametrized formulas [6, 5, 4, 8, 1]. The modeling of the execution time of communication operations can be used in compiler tools, in parallelizing compilers or simply as concise information of the performance behavior for the application programmer.

Genetic algorithms (GAs) are stochastic optimization techniques which simulate the natural evolutionary process of beings. They often outperform conventional optimization methods when applied to difficult problems. We refer to [2] for an overview on complex problems in the area of industrial engineering, where GAs have been successfully applied. The genetic programming approach (GP) has been introduced by Koza [7] and is a special form of a genetic algorithm.

The contribution of this paper is to show that runtime functions of high quality, which model the execution time of communication operations, can be modeled by the genetic programming approach. To illustrate the effectiveness of the approach we have chosen [8] for comparison. In [8], collective communication

* This work has been supported in part by DFG grant Ra 524/5 and Mo 645/5.

operations from the communication libraries PVM and MPI are investigated and compared. Their execution time is modeled by runtime functions found by curve fitting. In this paper we demonstrate the effectiveness of GPs to model the execution time of communication operations on DMMs, by example. Especially we demonstrate that GP generates runtime functions of higher quality than those published in literature till now.

The paper is structured as follows. Section 2 gives a brief overview on the investigations made by [8]. Section 3 presents how performance modeling can be attacked by GPs. Experimental results are shown and discussed in Section 4.

2 Runtime Functions Generated by Curve Functions

First, we give an overview on the communication operations which we use in our experiments. Then we present the model of the communication behavior on the IBM SP2 obtained by curve fitting.

The specific execution platform that [8] used for their experiments is an IBM SP2 with 37 nodes and 128 MByte main memory per processor from GMD St.Augustin, Germany, with IBM MPI-F, Version 1.41. They investigate and compare different communication operations. Due to space limitation, we only report on two communication operations.

Single-transfer operation: In MPI, the standard point-to-point communication operations are the blocking `MPI_Send()` and `MPI_Recv()`. For blocking send, the control does not return to the executing process before the send buffer can be reused safely. Some implementations use an intermediate buffer.

Scatter operation: A single-scatter operation is executed by the global operation `MPI_Scatter()` which must be called by all participating processes. As effect, the specified root process sends a part of its own local data to each other process.

In [8], the runtimes of MPI communication operations on the IBM SP2 are modeled by runtime formulas found by curve fitting that depend on various machine parameters including the number of processors, the bandwidth of the interconnecting networks, and the startup times of the corresponding operations. The investigations resulted in the parameterized runtime formula $f^{cf}_{scatter}(p,b) = 10.28 \cdot 10^{-6} + 91.59 \cdot 10^{-6} \cdot p + 0.030 \cdot 10^{-6} \cdot p \cdot b$ [μsec] for the scatter operation, and $f^{cf}_{single}(b) = 211.8 \cdot 10^{-6} + 0.030 \cdot 10^{-6} \cdot b$ [μsec] for the single-transfer operation. The parameters b and p are the message size in bytes and the number of processors, respectively.

3 The Genetic Programming Approach

The usual form of genetic algorithm was described by Goldberg [3]. Genetic algorithms are stochastic search techniques based on the mechanism of natural selection and natural genetics. They start with an initial population, i.e., an initial set of random solutions which are represented by chromosomes. The chromosomes evolve through successive iterations, called generations. During each

generation, the solutions represented by the chromosomes of the population are evaluated using some measures of fitness. To create the next generation, new chromosomes, called offsprings, are formed. The chromosomes involved in the generation of a new population are selected according to their fitness values. Fitter chromosomes have higher probabilities of being selected. After several generations, the algorithm converges to the best chromosome which hopefully represents the optimal or suboptimal solution to the problem. GAs whose chromosomes are syntax trees of formulas, i.e., computer programs, are GPs.

In the next sections we present the GP we have applied for the problem of finding runtime functions of high quality modeling the execution time of the MPI communication operations. We use terminology given in [7].

3.1 The Chromosomes

In genetic programming, a chromosome is an element of the set of all possible combinations of functions that can be composed recursively from a set F of basic functions and a set T of terminals. Each particular function $f \in F$ takes a specified number of arguments, specifying the arity of f. The basic functions we allowed in our application are addition $(+)$ and multiplication $(\cdot)$ both of arity 2, and the operations *sqr*, *sqrt*, *ln*, and *exp*, all of arity 1. The set T is composed of one or two variables p and b (p specifies the number of processors and b specifies the message size), and the constants 1 to 10, 0.1, and 10^{-6}. We represent each chromosome by the syntax tree corresponding to the expression.

In the following, we identify a chromosome with the runtime function which is represented by that chromosome in order to make the diction easier.

3.2 Fitness Function

Fitness is the driving force of Darwinian natural selection and, likewise, of genetic programming. It may be measured in many different ways. The accuracy of the fitness function with respect to the application is crucial for the quality of the results produced by GP. In the application handled here experiments have shown that the average percental error of a chromosome as fitness results in best solutions. Of course, as the aim is to minimize the average percental error, a chromosome is said to be fit if its average percental error is small. Let Pos be the set of measuring points and $m(\alpha)$ be the measured data at measuring point α, then the percental error $error_f(\alpha)$ of a runtime function f at point α and the average percental error $error_f$ of f are given by

$$error_f(\alpha) = \frac{abs(m(\alpha) - f(\alpha))}{m(\alpha)} \cdot 100 \quad \text{and} \quad error_f = \frac{\sum_{\alpha \in Pos} error_f(\alpha)}{\mid Pos \mid},$$

respectively.

There are two problems we have to care about when applying this fitness function. The populations can contain chromosomes whose average percental error is greater than the maximal decimal value (which is about $1.79 \cdot 10^{308}$ in GNU

g++). In this case, we have to redefine the error of these chromosomes to be some large value. The other problem is that some operations of the syntax trees are not defined, e.g., $\ln a$ with $a \leq 0$. The fitness function of our GP replaces these faulty operations by constants and adds a penalty to the error. Chromosomes with faulty operations are not taken as final result of the GP.

3.3 Crossover and Mutation Operators

The crossover operator for GP creates variation in the population by producing new offsprings that consist of parts taken from each parent. It starts with two parental syntax trees selected with error-inverse-proportionate selection, and produces two offspring syntax trees by independently selecting, using an uniform probability distribution, one random point in each parent to be the crossover point for that parent. The offsprings are produced by exchanging the crossover fragments. To direct the algorithm to generate smooth runtime functions we restricted the solution region to syntax trees of depth up to a constant c.

The mutation operator introduces random changes in structures randomly selected from the population. It begins by selecting a point within the syntax tree by random. Then, it removes the fragment which is below this mutation point and inserts a randomly generated subtree at that point. The depth of the new syntax tree is also limited by c.

3.4 Initialization and Termination

There are several methods to initialize the GP. At the beginning of our experiments we generated each chromosome of the first population recursively top down. The semantics of a node v of depth less than c is taken from $F \cup T$ by random. If a node has depth c, the semantics is randomly taken from the set T of terminals. Experiments showed, that the results are better if we generate a part of the initial chromosomes by approximation, and the rest of them as described above. These initial approximation functions represent linear dependencies between the measuring points and the measured data.

In our experiments, the GP stops after having considered a predefined number of generations.

4 Experimental Results

We begin by quantifying the quality of the runtime functions given in [8]. Then we compare these runtime functions to those generated by GP and discuss the results. We close the section by presenting a runtime formula generated by GP.

We evaluated the runtime functions given in [8] (see Section 2) by using the fitness function specified in Section 3.2. The test set of measured data contained communication times for $p = 4, 8, 16, 32$ processors and message sizes between 4 and 800 bytes with stepsize 20, between 800 and 4.000 with stepsize 400, and between 4.000 and 240.000 with stepsize 4.000.

Table 1. Deviations of the runtime functions

Operation	Approach	$error_f$	% of $\alpha \in Pos$ with $error_f(\alpha) < x$						
			< 0.01	< 0.1	< 1	< 10	< 25	< 50	< 100
scatter	curve fitting	57.36	0.0	0.6	2.6	23.8	49.0	72.5	83.4
	GP	9.68	0.0	2.3	14.2	62.9	93.7	97.7	100.0
single-transfer	curve fitting	71.90	0.0	1.2	20.9	47.4	59.4	67.4	73.4
	GP	14.64	0.3	1	17.6	59.6	75.6	93	100.0

To generate runtime functions by GP, we have to set some parameters. By setting the size of each population to 50, the maximum number of generations to 12000, the maximum depth of syntax trees to 15, and the probabilities for applying crossover, mutation, and the copy operation to 80%, 10% and 10% respectively, we obtained runtime functions by GP which dominate by far the runtime functions from [8] obtained by curve fitting.

In Table 1 we compare the deviations of the runtime functions found by curve fitting to the deviations of the best runtime functions generated after having run the GP algorithm 3 times, which takes about 6 hours on a SUN SPARC station Ultra 1, 128 MByte RAM. The first column specifies the communication operation, the second column the used approach and the third column the average percental error of the corresponding runtime function. Columns 4 to 10 give the percentage of measuring points $\alpha \in Pos$ for which the percental error $error_f(\alpha)$ is less than x for $x = 0.01$, 0.1, 1, 10, 25, 50 and 100, respectively.

Figure 1 graphically compares the deviation of the runtime function from [8] and the deviation of the runtime function generated by GP for the scatter operation with respect to some set of measured data. The curves differ very much for small values of parameter b whereby the curve generated by GP dominates by far the curve found by curve fitting. For large parameter values, both runtime functions predict the execution time of the scatter operation roughly alike. Figure 2 shows the comparison of the deviations of the runtime functions found by curve fitting and GP, respectively, for the single-transfer operation (which depends only on parameter b) with respect to several sets of measured data. Figure 2 shows only the cut-out determined by the small message sizes (up to 500 byte).

Let us discuss the results. The disadvantage of curve fitting is that usually only one simple function is selected to model a communication operation for a large range of message sizes. This may lead to poor results in some regions since often different methods are used for different message sizes. GP on the other hand allows for generating complex functions.

We close the section by presenting the runtime formula $f^{GP}_{scatter}$ predicting the execution time of the scatter operation which has been generated by GP :

$$f^{GP}_{scatter}(p,b) = 10^{-6} \cdot (b \cdot p \cdot (p^{1/2} - 1)^{1/2})^{1/2} \cdot (b \cdot p \cdot 0.000625 + 0.00005 \cdot b + 0.00375 \cdot p^2 + 0.003 \cdot p + \ln(e^{p/2}) + 49)^{1/2}.$$

It shows that the runtime formulas generated by GPs are rather complex. To our opinion, this is the only disadvantage of the approach we presented.

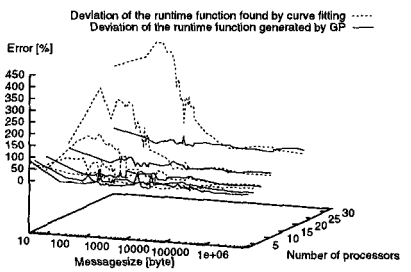

Fig. 1. Deviation of the runtime function found by curve fitting and deviation of the runtime function generated by GP for the scatter operation.

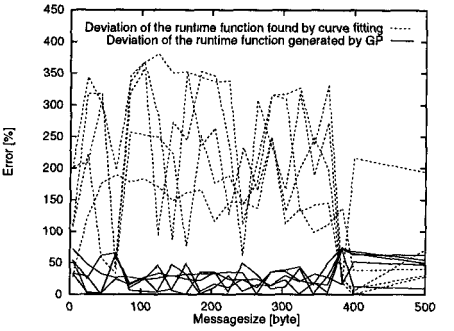

Fig. 2. Deviation of the runtime function obtained by curve fitting and deviation of the runtime function generated by GP for the single-transfer operation.

5 Conclusions

We showed by example that genetic programming provides runtime functions predicting the execution time of communication operations of DMMs which dominate the runtime functions found by curve fitting.

The next step of our investigations we will make is to automatically generate different runtime functions for different intervals of the parameters. The intervals themselves have to be determined by genetic programming, too. Using such non uniform runtime functions seems to be necessary to predict the execution functions more accurately, as DMMs use various communication protocols depending on the message size.

References

1. Foschia, R., Rauber, Th., and Rünger, G.: Prediction of the Communication Behavior of the Intel Paragon. In *Proceedings of the 1997 IEEE MASCOTS Conference*, pp.117-124, 1997.
2. Gen, M., and Cheng, R.: *Genetic Algorithms* & Engineering Design. John Wiley & Sons, Inc., New York, 1997.
3. Goldberg, D.: *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, Reading, MA, 1989.
4. Hu, Y., Emerson, D., and Blake, R.: The communication performance of the Cray T3D and its effect on iterative solvers. *Parallel Computing*, 22:829-844, 1996.
5. Hwang, K., Xu, Z., and Arakawa, M.: Benchmark Evaluation of the IBM SP2 for Parallel Signal Processing. *IEEE Transactions on Parallel and Distributed Systems*, 7(2):522-536, 1996.
6. Johnson, L.: Performance Modeling of Distributed Memory Architecture. *Journal of Parallel and Distributed Computing*, 12:300-312, 1991.
7. Koza, J.: *Genetic Programming.* The MIT Press, 1992.
8. Rauber, Th., and Rünger, G.: PVM and MPI Communication Operations on the IBM SP2: Modeling and Comparison. In Proceedings of HPCS 97, 1997.