Estimating and Measuring Performance of Computational Agents

Roman Neruda, Pavel Krušina
Institute of Computer Science, Academy of Sciences of the Czech Republic,
P.O. Box 5, 18207 Prague, Czech Republic,
bang@cs.cas.cz

Abstract

We study and design multi-agent systems for computational intelligence modeling. Agents typically reside in a high-performance parallel environment, such as a cluster of computational nodes, and utilize a non-blocking asynchronous communication. The need of accurate predictions of run-time and other characterizations of complex parallel asynchronous processes bring us to design a new parallel model creation methodology. In this article our approach is briefly described and a test case is shown and discussed.

1. Introduction

In our project [4] we develop a multi-agent system aimed at hybrid computational intelligence models represented as a collection of autonomous agents in a multi-agent system [9]. One of the goals was to develop a unifying framework allowing time complexity estimates for agents encompassing computational methods on one hand, and a computer-aided performance analysis of the real agents behavior in a distributed environment on the other. Since a good fit of theoretical estimates to measured performance data is needed, we decided to sacrifice the simplicity of the model (as opposed to e.g. BSP [8] or LogP [1]) for better accuracy. This feature can be compensated for by a semi-automated way the parameters of the model are obtained for a particular computer architecture.

In this paper we demonstrate the above mentioned approach first on deriving a concrete complexity model called NASP which is further tuned for various computer architectures, from SMPs to clusters of workstations. Experimental performance evaluation has been made for a genetic algorithm [6] suite of agents.

2. Theoretical Model

While the theoretical RAM model [2] of serial computation is clear and widely accepted, the existing program developing tools target mainly real computers and their machine codes. We enhance the RAM model such that its programming language is a real computer assembler. This approach has the advantage of generating such models by machine compilation from high-level programming languages. Also the possibility of direct execution and measurements of these models is valuable especially for the complex computations.

A *x86 random access machine* (86-RAM) is based on the traditional Random Access Machine (RAM) model with the instruction and register sets modified to correspond to the real computer architecture IA-32 [3]. Proper definitions can be found in [5] and [7].

Definition 1 Generalized serial model *of an algorithm A is a function $\mathcal{A}$ on task size* $\mathbf{J}$ *and computer description* $\mathbf{M}$ *returning cost estimates* $\mathbf{c}$ *of the algorithm A implementation running on the* $\mathbf{M}$ *computer given the* $\mathbf{J}$*-large task to solve.*

We see that in order to be able to make predictions on algorithm costs, we need to obtain the model function $\mathcal{A}$, the computer description $\mathbf{M}$, and the task description $\mathbf{J}$ which is a generalization of the more traditional variable N meaning the input data size in bits. Now we focus on the machine independent model creation process.

At the beginning, an algorithm description $\mathcal{A}$ in a programming language is given. The goal is to estimate the $\mathcal{A}$ run costs on a target machine with the description $\mathbf{M}$. Let us further assume that we can run $\mathcal{A}$ on a different machine $\mathbf{M}'$ where we can do extra performance measurements of the execution. First choice is to select an atomic operations set $\mathbf{A}$. Those operations define basic metrics in which algorithm requirements are modeled and measured. This selection influences the accuracy the model can achieve: the larger the set is, the more accurate the model can be. The simplest atom set has the only member—an instruction, a two member set is a better selection—an integer instruction and a floating point instruction. More complex models count also cache misses, page faults, branch instructions, context switches, function invocations etc.

Next, the algorithm implementation is executed on the $\mathbf{M}'$ computer for various tasks $\mathbf{J}$, and the measurements of the atomic operations $\mathbf{A}$ are done. After that, a suitable regression technique is applied to get a function approximation F of these, depending on the task size $\mathbf{J}$. Notice that most of this work could be done automatically once the benchmarking tools are available. The function F maps task size description $\mathbf{J}$ in the atoms counts space $N^{|\mathbf{A}|}$. To obtain time, space and other costs $\mathbf{c}$ from these counts, the atomic operations benchmark on the target machine can be used. It is common to unify such machine benchmark results with the machine description $\mathbf{M}$. Finally, a functional composition of the approximation function F and machine description argument defines the generalized model function of $\mathcal{A} : \mathbf{M} \times \mathbf{J} \rightarrow \mathbf{c}$.

Definition 2 *An* asynchronous parallel machine *(ASP) is a universal computation device consisting of a fixed set of 86-RAM machines* $\mathsf{C} = P_0, P_1, ...P_{p-1}$ *and a communication interconnect* N.

The ASP machine description $\mathbf{M}$ *is a quartet* $\mathbf{M} = [p, \mathbf{A}, \mathbf{N}, \mathbf{c}]$. *The* p *is a number of processors. The* $\mathbf{A}$ *is a matrix* $p \times q_{\mathsf{C}}$*, the* q_{C} *is number of sequential atomic operations,* $\mathbf{A}_{i,j}$ *is the* j*-th atomic operation cost when executed on the* i*-th processor.* $\mathbf{N}$ *is basic communication routines cost functions* $\mathbf{N}_i : (|\delta|, \mathtt{src}, \mathtt{dst}, L_{\mathsf{C}}^{\mathtt{src}}, L_{\mathsf{C}}^{\mathtt{dst}}, L_{\mathsf{N}}) \rightarrow \mathbf{c}$ *vector of size* q_{N}*. And* $\mathbf{c}$ *is a 13-tuple containing a collection of private data and the following functions:*

Cost combination rules:	*Cost reduction operators:*
$SeqOver : (\mathbf{c}, \mathbf{c}) \rightarrow \mathbf{c}$	$T : \mathbf{c} \rightarrow \mathcal{R}^+$
$SeqNext : (\mathbf{c}, \mathbf{c}) \rightarrow \mathbf{c}$	$S : \mathbf{c} \rightarrow \mathcal{N}_0^p$
$SeqOver : (\mathcal{N}, \mathbf{c}) \rightarrow \mathbf{c}$	$U_{\mathsf{C}} : \mathbf{c} \rightarrow [0,1]^p$
$SeqNext : (\mathcal{N}, \mathbf{c}) \rightarrow \mathbf{c}$	$U_{\mathsf{N}} : \mathbf{c} \rightarrow [0,1]$
$ParOver : (\mathbf{c}, \mathbf{c}) \rightarrow \mathbf{c}$	$O_{\mathsf{C}} : \mathbf{c} \rightarrow \mathcal{R}_0^{+p}$
$ParNext : (\mathbf{c}, \mathbf{c}) \rightarrow \mathbf{c}$	$O_{\mathsf{N}} : \mathbf{c} \rightarrow \mathcal{R}_0^+$

The flexibility of ASP model definition leaves intentionally some parameters—like a number and meanings of atomic operations—unspecified. This makes from ASP something like an abstract template base upon which various but related models can be built. We define one representative of the ASP models family by selecting the ASP free parameters. It is called NASP and we will use it later in this work.

Definition 3 *A* normal asynchronous parallel machine *(NASP) is an ASP device with the following configuration. The* q_{C} *equals to 6 and the sequential atomic operations are: fast integer operation, fast float number operation, slow integer operation, slow float number operation, memory allocation, and memory deallocation. The* q_{N} *is 3 and the communication operations are: send, transfer, and receive.*

The ASP machine is a set of connected serial machines. Each computation node is modeled by a single 86-RAM. The ASP machine description $\mathbf{M}$ combines features of isolated computers such as the $\mathbf{A}$ atomic operations costs with the interconnect network description of $\mathbf{N}$. Further it contains the number of computation nodes p and the structure $\mathbf{c}$, which keeps track of all various costs relationships, combination rules, and reductions.

The $\mathbf{A}$ matrix can be understood as a vector of processors descriptions, each independent of the others. In each processor description, the first pair —the costs of single integer and floating point operations—is responsible for serial computation time estimates; while the other pair —the costs of dynamic memory allocation and deallocation—is used for memory space estimates.

Similarly the $\mathbf{N}$ vector provides cost $\mathbf{c}$ for various basic operations related to network communication.

The $\mathbf{c}$ cost structure—the last component of the $\mathbf{M}$ machine description— contains all the costs combination logic and its instances keep track of all various costs the algorithm in question pays for. The $\mathbf{c}$ object can have a dominant value on the time axis like the case of the integer operation cost, or it may have the time component zero and the memory space bigger like the memory allocation operation has. It can also have more important attributes as for example network send operation has— it takes some time, loads the processor and at the same time loads the interconnect network.

3. MAS implementation of a Genetic algorithm

Genetic Algorithm is an iterative optimization technique that works on a set of partial solutions—a population. In each iterations, relatively good solutions are taken from that set, optionally modified with various operators, and collected to create a new population. This process repeats till a sufficiently good solution approximation is found (Fig. 1).

A parallelization of Genetic Algorithm is often done via this scheme replication over the computational nodes and communicating best solutions among populations (Fig. 1).

Figure 2 shows a block diagram of one iteration of genetic algorithm. Each iteration starts with the fitness function evaluation for each genome that has changed from the last iteration—that is phase A. Then, the best genome is found and sent to the next one in the parallelization ring in phase B2. The B2 code is executed only if the elite genome has come to the actual process and only once for any such incoming genome. Next, in phase C, a new generation is

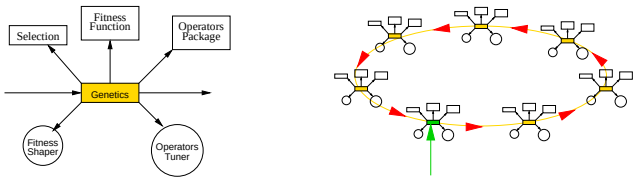


Figure 1. a) The generalized genetic algorithm decomposition. b) The generalized genetic algorithm parallelization via circular communication among per-machine subpopulations.

created in cooperation with the selection and genetic operators. When the new population is large enough, the iteration ends and a new one is started. Any time during the iteration, an elite genome may come up and be processed—phase B1. This completes the analysis of Genetic Algorithm code run. But other agents are executed as well. We name the fitness function code as F, the selection code S, and the metrics and genetic operators code M/O.

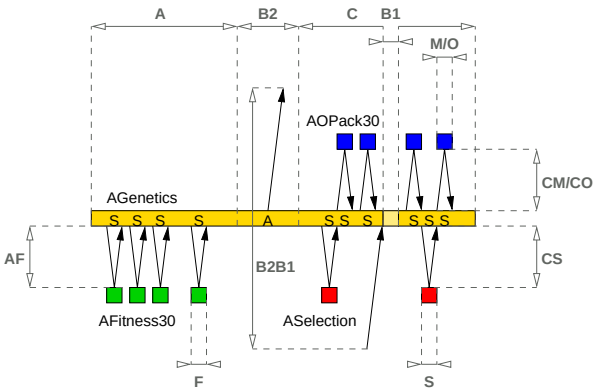
For each of the computation blocks (A,B1,B2,C,F,M/O,S), the following properties were measured: integer operations, floating point operations, processor cycles, the number and type of communications performed together with the sizes of transferred data. For the main blocks A,B1,B2,C, the amount of used memory was also measured.

4. Experiments

To test the constructed model of our genetic algorithm, we obtained descriptions of several machines, calculated predictions of iteration times, and measured them on the real GA running there. *Reference machine* is a regular IA32 computer with Celeron CPU, *SGI Origin 2000* is a RISC computer with two R12000 processors, *Sun ULTRA 10* is a RISC computer with one UltraSPARC IIi processor. The next machine is an *IBM PC* with Pentium II, and *Joyce* is a PC/Linux cluster of 16 Athlon XP nodes connected via a 100-Mbit Ethernet star-topology network.

First, experimental measurements of the genetic algorithm run are compared to the GA models predictions for all of the test computers. The average relative errors are between 1.16—in the case of PC—and 1.8—for the Sun computer. Table 1 and Figure 3 gather these results.

Next, the parallel model is compared to the parallel run of the genetic algorithm. To be able to construct the model of parallel genetic algorithm, a network inter-node communication time has to be analyzed. Similarly to the other distributed memory parallel machines, in the case of Joyce the communication time depends on the amount of transferred data. Measurements of these times have been done for a



A	The fitness function is evaluated.
B1	An elite genome is received from the previous AGenetics in the parallelization ring.
B2	If there is an elite genome from outside, it is added to the current population and the best genome is sent next through the ring.
C	The selection is called and the operator package applies genetic operators.
F	The fitness function evaluation.
M	The metric operator evaluation.
O	The genetic operator evaluation.
S	The selection process.
AF	The communication between AGenetics and AFitness30.
B1B2	The communication between two AGenetics in the parallelization ring.
CO	The communication between AGenetics and AOPack30 during evaluation of a genetic operator.
CM	The communication between AGenetics and AOPack30 during evaluation of a metric operator.
CS	The communication between AGenetics and ASelection.

Figure 2. Generalized genetic algorithm implementation.

whole scale of data sizes and the following formula approximates the obtained data best: $T(x) = max(1.65x, 5000 + 0.75x) - 2500\mu s$.

In our parallel model of GA, we want to predict how many iteration would proceed between successor elite genome arrivals into a particular node. To do so, we need to model the inter-node communication and the overall system behavior with respect to the asynchronous nature of its processes (see Figure 4).

For our test, we selected a configuration of 3 computational GA nodes. The computational nodes run unsynchronized, thus we can consider the elite genome income to occur in a random phase of the iteration. Since the elite genome can be emitted only from a fixed point of the iteration cycle the expected slow-down is one half (of the iteration cycle) per parallelization ring node. That is for the

Computer	Graph	Max E	Avg E	Min E
SGI	3	2.2	1.6	1.2
Sun	3	4.2	1.8	1.7
PC	3	1.21	1.16	1.04
Joyce	3	2.4	1.3	1.1

Table 1. Model evaluation on test computers.

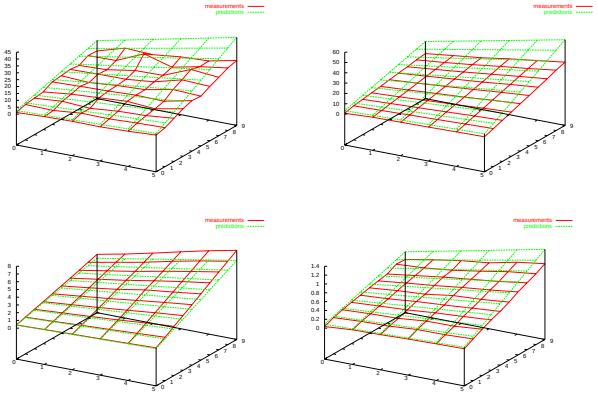


Figure 3. GA iteration time measurements and model predictions on Origin, Sun, IBM PC and a cluster of PCs.

time the elite genome spends waiting for the target process. Besides this, there is also the time spent communicating the elite genome among computational nodes which, if expressed in iterations, is $\frac{t_c}{t_G}$ per parallelization ring node (for the iteration times t_G and the communication time t_c). Thus it can be easily observed that the expected number of iterations between two elite genomes comings is: $X = \frac{3}{2} + 3\frac{t_c}{t_G}$.

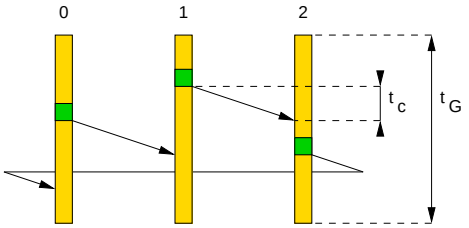


Figure 4. Parallel run of GA on 3 nodes.

The graph in Figure 5 shows the measured numbers of iterations between elite genomes comings X in the real parallel runs of GA together with two model predictions of these. Since the dependence of the elite genomes coming interval on the population size has shown to be small for the investigated parameters, we selected three more population sizes to try—2,5 and 10. Though the GA model is stressed

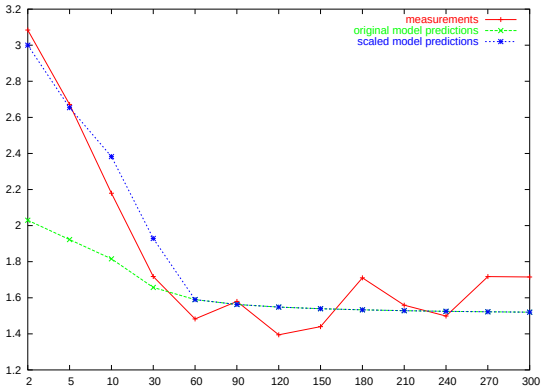


Figure 5. The number of iterations between elite genomes comings in real measurements and two models.

beyond the area it was built for, it still shows the correct tendency.

Acknowledgments

This research has been supported by the National Research Program Information Society project 1ET100300419.

References

[1] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practise of Parallel Programming*, 1993.

[2] S. Fortune and J. Wylie. Parallelism in random access machines. In *Proceedings of the 10th Symposium on theory of Computing*, pages 114–118, 1978.

[3] *IA-32 Intel Architecture Software Developer's Manual*, volume Volume 2: Instruction Set Reference. Intel Corporation.

[4] P. Krušina, R. Neruda, and Z. Petrová. More autonomous hybrid models in bang. In *International Conference on Computational Science*, pages 935–942, 2001.

[5] P. Krušina. *Models of Multi-Agent Systems*. PhD thesis, Charles University, Prague, 2004.

[6] Z. Michalewicz. *Genetic Algorithms+Data Structures=Evolution Programs*. Springer-Verlag, Berlin, 1994.

[7] R. Neruda and P. Krušina. A framework for modelling and estimating complexity in multi-agent systems. In T. Gonzales, editor, *Parallel and Distributed Computing and Systems*, pages 602–607. ACTA Press, 2004.

[8] L. Valiant. A bridging model for parallel computation. In *Communications of the ACM*, volume 33(8), pages 103–111, 1990.

[9] G. Weiss, editor. *Multiagent Systems*. The MIT Press, 1999.