AIM−GP and Parallelism

Peter Nordin
Physical Resource Theory
Chalmers University of Technology
Sweden

Frank Hoffmann
LS11
University of Dortmund
Germany

Frank D. Francone
RML
Technologies, Inc.
USA

Markus Brameier
LS11
University of Dortmund
Germany

Wolfgang Banzhaf
LS11
University of Dortmund
Germany

Abstract

Many machine learning tasks are just too hard to be solved with a single processor machine, no matter how efficient algorithms we use and how fast our hardware is. Luckily genetic programming is well suited for parallelization compared to standard serial algorithms. This paper describes the first parallel implementation of an AIM-GP system creating the potential for an extremely fast system. The system is tested on three problems and several variants of demes and migration are evaluated. Most of the results are applicable to both linear and tree based systems.

Keywords: Genetic Programming, Machine Code GP, Parallelism

1. Introduction

The hardware speed of desktop computers is growing exponentially. This means that every year new application areas can be addressed with machine learning and adaptive systems. The constant development of new algorithms and implementation methods also enables us to explore previously impossible terrain. However there are absolute physical limitations to even the fastest serial computer with the most efficient algorithm. The ultimate goal of artificial intelligence might be to build an algorithm with capabilities of the human mind. The problem is that if we were to simulate the maximum computing capability of the human brain—with all synapses active—a serial computer must be smaller than a single atom to accomplish this. This fact is due to limitations caused by the speed of light and there is nothing we can do about it except for extending our algorithms to parallel computers. In addition, parallel desktop computers are becoming mainstream with seamless support from operating systems making multi-processor systems uncomplicated to use.

Genetic programming has proven to be a very robust, efficient and broadly applicable method. The good news in this context is that it is also very suitable for parallelization which has been successfully demonstrated (Andre and Koza 1996). Some parallel approaches even display nearly "super linear speed-up" over the simple serial approach. This means that the parallel version of the algorithm is more efficient even if it is simulated on a purely serial machine (Andre and Koza 1996). This is a very beneficial property of a machine learning algorithm and it has led to unique and impressive projects such as the 1000 processor Alpha machine under construction in John Koza's team.

AIM-GP (Automatic Induction of Machine Code with Genetic Programming) is formerly know as CGPS. This method induces binary machine code directly without any interpreting steps enabling speed-ups of two orders of magnitude. AIM-GP is the fastest GP approach and the most studied linear GP method. But as seen above, even a very efficient algorithm will eventually benefit from parallelization. This paper documents the implementation of a parallel AIM-GP algorithm and experiments design to evaluate it. A large part of the results count for tree based systems, too.

2. Method

The method can be subdivided into AIM-GP implementation, hardware, and parallelization:

2.1 AIM−GP

AIM-GP uses a linear representation directly corresponding to binary machine code. Crossover is performed as a string crossover between instructions while mutation switches instruction or selected bits within the instruction. In the experiments we use the PowerPC chip from Motorola and the original version of AIM-GP without instruction blocks. For further details on AIM-GP see (Nordin 1997, Banzhaf et al . 1998, Nordin et al. 1999).

2.2 Hardware

To connect several processors into a parallel computer is not a trivial task. Several different architectures exist. In our experiments we have used the Parsytec Power Explorer, one of the most widespread commercial systems. The power explorer is a MIMD machine (multiple instruction multiple data system) with up to 64 power PC processors. The configuration used for the experiments was a 16 processor variant, but the approach will port directly to 64 processors.

Each node in the computer consists of a PowerPC601, a T414 support transputer processor, 32 MB of RAM and Boot ROM as seen in Figure 1. The support processor manages efficient serial communication between nodes, see Figure 2.

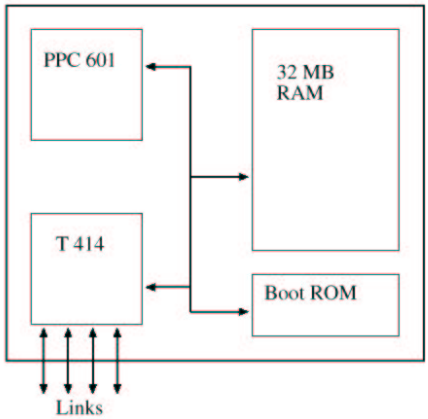

Figure 1: A node in the Power XPlorer

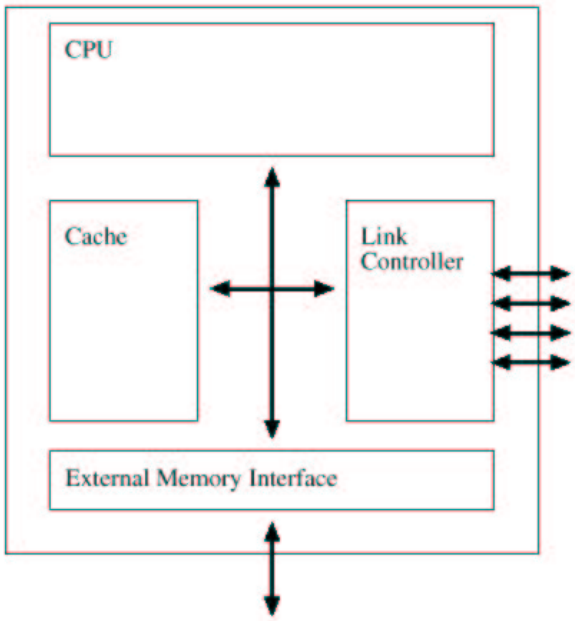

Figure 2: The T414 Transputer

The PowerPC is a conventional cached RISC processor also used in for instance Macintosh computers.

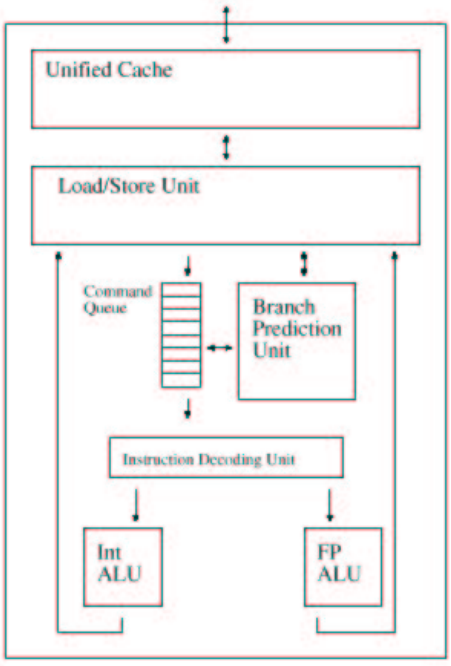

Figure 3: The PowerPC processor

The nodes of the the Parsytec Power Explorer are coupled together in a grid as seen in Figure 4.

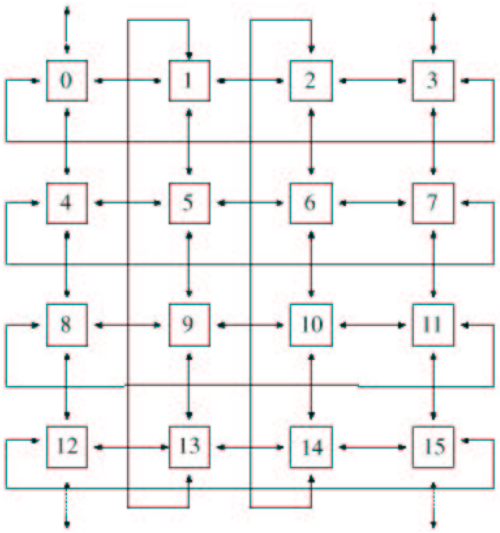

Figure 4: Connection of Nodes

We use the standard set-up with the Ultrix operation system.

2.3 GP and parallelization

There are several ways to parallelize the GP algorithm. It could be divided both with regard to fitness cases and to the population. However, the most common way to achieve parallelization is through "demes". The notion of demes was introduced by Wright in 1943 as a description of a phenomenon in natural evolution where an animal population is divided into subpopulations (Wright 1943). Between the otherwise separated subpopulations infrequent migration takes place. This model is often referred to as the island model, see Figure 5. Demes are argued to reduce the possibility for evolution to end up in a local minimum and this increase the efficiency of the paradigm. This "super linear speed-up" has been observed in simulations of evolutionary algorithms.

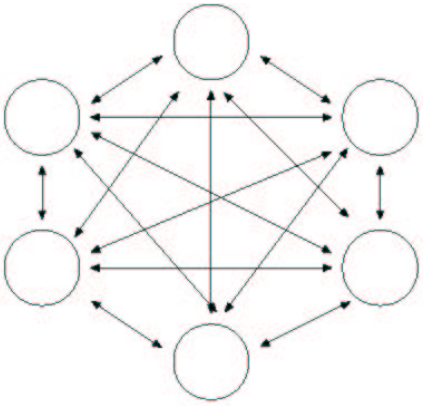

Figure 5: Island Model

The demes can be arranged in different patterns. Two of the most common are the "stepping stone model as ring" and "the stepping stone model as lattice" (Kimura 1953), see Figure 6.

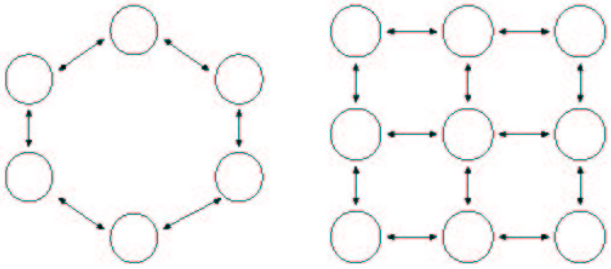

Figure 6: Stepping Stone model as ring and lattice

A related approach using demes is the neighborhood model. Here the individual can migrate to more than the closest island, they can move within a predefined "neighborhood distance", see Figure 7 (Gorges-Schleuter 1989).

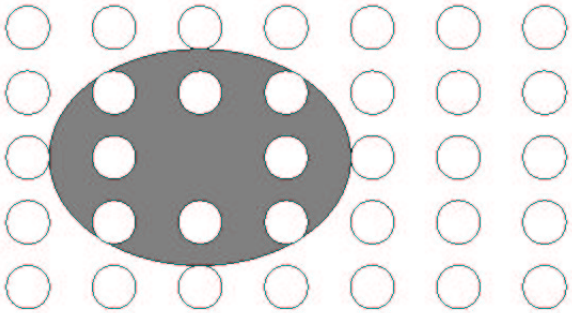

Figure 7: Neighborhood model

There exists several implementations of parallel EAs in research. Tanese (Tanese 1989) used an island model of a GA for a parallel machine. He concluded that migration rates have a decisive influence on the performance of the algorithm and that it is possible to achieve at least linear speed-up by adding new nodes.

Andre and Koza implemented a parallel GP system on a transputer network (Andre and Koza 1996).

2.4 Design of the AIM-GP system

To maximize performance we knew that we needed to keep communication between nodes to a moderate level. We also wanted to keep the system flexible and portable using standard Ultrix features. The main design of an AIM-GP node is illustrated in Figure 8.

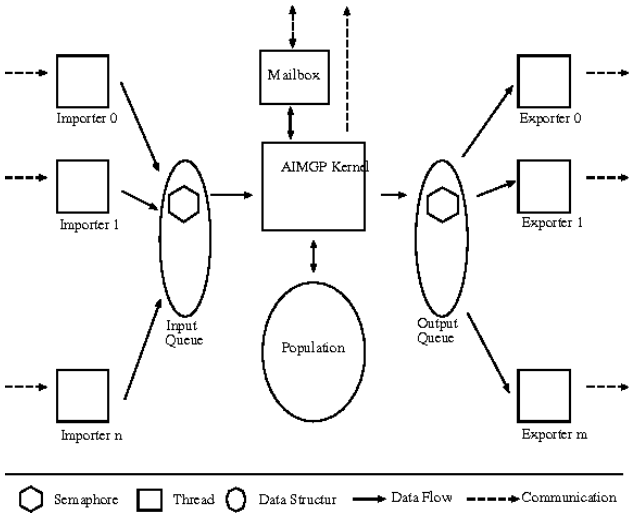

Figure 8: A node with AIM−GP

The emigration of individuals takes place through an input/output queue where the export thread is responsible for sending it to another node. In a similar way there is an immigration queue and an immigration thread on the receiving island. The immigration individual in the simplest case replaces a random existing individual.

AIMGP uses 8 of the 32 registers of the PowerPC (*r03 - r10*), while the machine instructions in the function set were *add, addi, subf, mullw, mulli, and, andi, or, ori, xor, xori, eqv, nand, nor, slw, srw, sraw*.

The output of the system can be presented both as assembler and decompiled into C, for example:

```
00000000: 7D032378 * or r03,r08,r04 ;
00000004: 7C051838 and r05,r00,r03 ;
00000008: 7CE85038 and r08,r07,r10 ;
0000000C: 7C883C30 srw r08,r04,r07 ;
00000010: 7C683C30 * srw r08,r03,r07 ;
00000014: 7CA54A14 add r05,r05,r09 ;
00000018: 7CE54BB8 nand r05,r07,r09 ;
0000001C: 7C8643B8 nand r06,r04,r08 ;
00000020: 7C644378 * or r04,r03,r08 ;
00000024: 7CA449D6 mullw r05,r04,r09 ;
00000028: 7CE623B8 nand r06,r07,r04 ;
0000002C: 7C880278 * xor r08,r04,r00 ;
00000030: 7D0A1838 * and r10,r08,r03 ;
00000034: 7C842038 * and r04,r04,r04 ;
00000038: 7C895430 * srw r09,r04,r10 ;
0000003C: 7CE019D6 mullw r07,r00,r03 ;
00000040: 7C834B78 or r03,r04,r09 ;
00000044: 7C834B78 * or r03,r04,r09 ;
00000048: 4E800020 blr ;

int individual_function(int r03, int r04, int
r05, int r06, int r07, int r08)
{
r03 = r08 | r04;
r08 = r03 >> r07;
r04 = r03 | r08;
r08 = r04 ^ r00;
r10 = r08 & r03;
r04 = r04 & r04;
r09 = r04 >> r07;
r03 = r04 | r06;
return r03;
}
```

3. Experiments

3.1 Evaluation problems

We selected three problem classes to evaluate the properties of the parallel AIM−GP:

1. A boolean problem
2. A function regression problem
3. An image classification problem

Twenty runs with different random seeds were performed for each of the problem set−ups. All problems were either run for a maximal number of tournaments or until a perfect solution (fitness 0) was found. The fitness function always used the sum of the absolute values of errors. Both mutation and crossover probabilities were set to 90%.

Function regression

For function regression we used a polynomial which allowed for scaling of difficulty:

$$f(x, y) = 5(x^4 + y^3) - 3x^2$$

Each time the fitness cases consisted of 200 randomly selected values. The instructions in the function set consisted of the operations *add, sub, mul*.

Parity function

The parity function takes a number of N bits and outputs "1" if the number of "ones" is even or "0" otherwise:

$$e = \bigoplus_{n=0}^{N} b_n$$

In this case the function set was {*and, or, nand, nor*} with the constants 1 and 0. We deliberately sustained from using the *xor* instruction to make the problem harder. Using the *xor* AIM-GP solves the problem almost immediately. The parity function belongs to the hardest class of Boolean function with a very rough search space.

Partitioning of Images

Images classification is a growing application domain due to increasing use of digital images and due to decreasing cost of hardware.
The objective of the problem is to divide the pixels of an image into two classes depending on properties of the original pixels in the image. Figure 8 illustrates the output of an individual segmenting the image in two areas with two gray-tones. The individual is feed by x and y coordinates and expected to give the class of that pixel as output. 500 randomly selected pixels are used for individual fitness and another 500 are used for generalization. The output of the individual is interpreted as one of two cases depending on if its greater than or less than a certain threshold value. The individual is free to use all of the machine code instruction in the PPC AIM-GP implementation.

Figure 8: Exampe image segmentatio

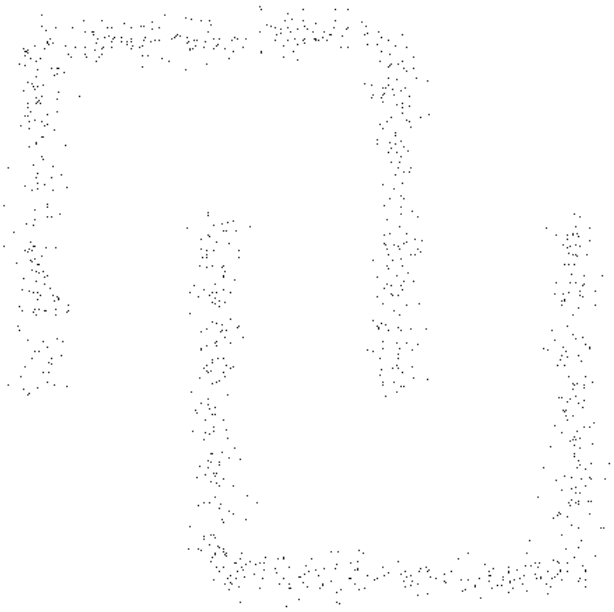

Figure 9: Input image data

Initial populations show individuals usually overfitting stripes to certain points as in Figure 10. Later more complex, often fractal patterns appear, see Figure 11. Finally a fitting geometry evolves as in Figure 8.

Figure 10: Stipes: uses only one input

Figure 11: Pattern which uses both inputs

4. Results

In this work the parallel AIMGP system has been investigated in relation to the influence of migration on the learning process. The system has been configured using a sub−population size of 1000 individuals for all 16 processing nodes. Each experimental result is documented as an average of the 20 runs performed.

4.1 Migration Effects

The migration between demes has a qualitative and a quantitative effect. The migration frequency specifies how many individuals migrate each time interval. The qualitative part defines which individuals are allowed to migrate.

4.2 Migration Frequency

In all our experiments migration rate is varied by changing the frequency of migration instead of the number of individuals that are emigrated from each deme during each migration phase. This is different from (Andre and Koza 1996) who took the latter approach. Our migration technique is strongly motivated by nature where migration is a rather *continuous* process. During each migration step only *one* individual is selected from each deme and moved to all adjacent nodes in the transputer network as shown in Figure 4. In addition to the motivation by nature there are some technical advantages: The workload of the links is reduced and synchronization problems occur less frequently.

A test run without migration is shown in Figure 12 where the fitness development of all isolated demes can be seen. Effectively, this means running the problem with different random seeds 16 times simultaneously. The GP system uses tournament selection. The migration rate is measured in relation to number of tournaments. With a very high migration rate (every 250th tournament) we get a development as in Figure 13. The high migration rate results in a worse final fitness. With a migration every 1000nd tournaments we get a graph as depicted in Figure 14. We conclude that migration is important for the parallelization of a GP system and we can see a 15 fold increase in performance in the best migration setting compared with runs without migration. On the other hand too much migration is not optimal. In all test cases we see worse performance with too high migration.

4.3 Migration Strategies

The method by which an individual is selected for emigration controls the quality of the emigrating individual. In our experiments we evaluated several different approaches. We used random emigration, emigration of the best and tournament emigration. Tournament emigration chooses the winner of the last tournament as emigrant giving a random distribution of better individuals but seldomly the best individual in the population.

Tables 1-9 summarize our results for the three problems and the three migrations strategies respectively. Tables 1 and 2, for instance, show how random migration and tournament migration tend to perform similarly in relation to the best fitness found, the number of evaluations and the number of executed instructions per run. Average values and standard deviations (σ) are given for all three measures.
In comparison, emigration of the best performs worse (see Table 3) probably due to a higher tendency to get stuck in local optima. Only runs with very fast search might do better with emigration of the best since a more elitistic approach is better as a global optimum is being approached.

There seems to be a trade-off between fast progress in fitness and loss of diversity with emigration of the best as illustrated in Figure 19. Loss of diversity lets the evolutioary process run into local sub-optima because no new genetic material can be created. This is especially the case if best individuals are reproduced in different demes with high migration rates. In contrast, Figure 18 shows a rather continuous improve in fitness for the random strategy when the migration rate is increased.

The selection of individuals which are to be replaced by the immigrants can also affect performance. We have studied three basic methods: random selection, selection of the worst individual and tournament selection where the loosers of the tournament are selected for replacement. We found that there is little difference in performance. Even random selection appears to be a robust method.

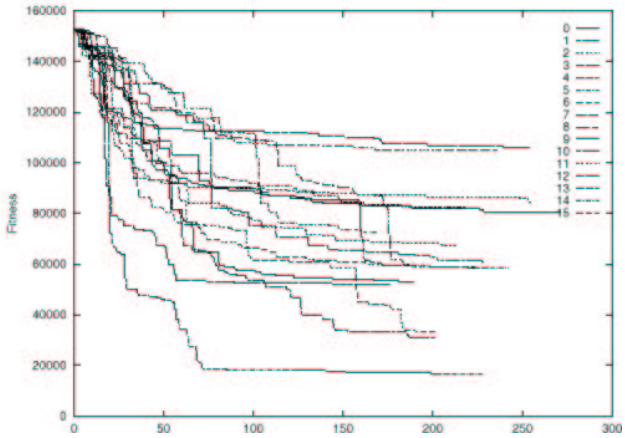

Figure 12: Fitness over time (in million evaluations) of best individual in all 16 demes without migration

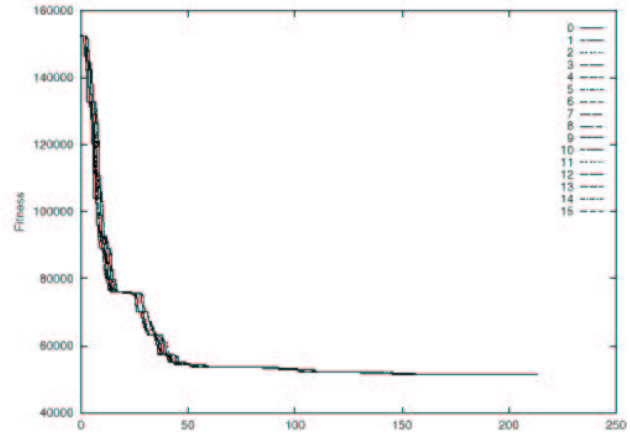

Figure 13: Fitness over time of best individual in all demes with migration every 250 tournaments

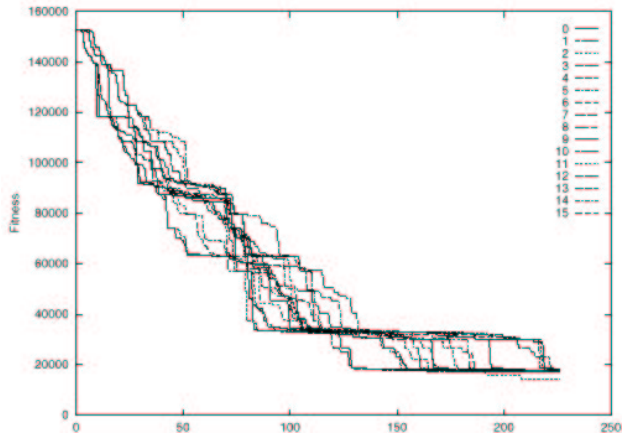

Figure 14: Fitness over time of best individual in all demes with migration every 1000 tournaments.

Migration distance	250	500	1000	2000	3000	∞
Fitness	0	0	0	0	0	355
σ Fitness	0	0	0	0	0	185
% No solution	0	0	0	0	0	30
Evaluations*10^6	162	240	198	219	346	2569
σ Evaluations	13	49	25	20	99	588
Instructions*10^6	7790	12823	9852	10504	19523	161755
σ Instructions	856	3078	1658	1836	6349	37683

Table 1: Random emigration (function regression)

Migration distance	250	500	1000	2000	3000	∞
Fitness	0	0	72	0	0	355
σ Fitnes	0	0	72	0	0	185
% No solution	0	0	5	0	0	20
Evaluations*10^6	272	164	480	381	272	2569
σ Evaluations	82	14	304	125	39	588
Instructions*10^6	9811	7650	28086	14309	14954	161755
σ instructions	1068	921	19492	2655	2480	37683

Table 2: Tournament emigration (function regression)

Migration distance	250	500	1000	2000	3000	∞
Fitness	79	0	0	72	218	355
σ Fitness	55	0	0	72	150	185
% No solution	10	0	0	5	10	20
Evaluations*10^6	1371	444	252	637	982	2569
σ Evaluations	472	137	46	321	415	588
Instructions*10^6	80121	25400	13374	38064	60188	161755
σ instructions	30563	8821	2910	20575	26577	37683

Table 3: Emigration of the best (function regression)

Migration distance	250	500	1000	2000	3000	∞
Fitness	0,5	0,6	0,6	0,9	1,7	6,2
σ Fitnes	0,2	0,3	0,3	0,4	0,8	0,8
% No solution	20	20	15	20	25	95
Evaluations*10^6	988	930	865	834	1085	2014
σ Evaluations	172	169	171	160	164	78
Instructions*10^6	233447	211982	213823	198620	258084	477506
σ Instructions	45273	42213	43789	41124	43065	29289

Table 4: Random emigration (parity function)

Migration distance	250	500	1000	2000	3000	∞
Fitness	1	1	0,9	0,8	1,1	6,2
σ Fitnes	0,5	0,5	0,5	0,4	0,4	0,8
% No solution	20	20	20	20	35	95
Evaluations*10^6	710	765	835	973	1144	2014
σ Evaluations	179	164	162	172	171	78
Instructions*10^6	166962	177185	198122	234007	277529	477506
σ Instructions	45829	43833	41783	44068	43968	29289

Table 5: Emigration of the best (parity function)

Migration distance	250	500	1000	2000	3000	∞

Fitness	0,2	12	12	8	10	6,2
σ Fitnes	0,1	1	1,2	1,15	1,2	0,8
% No solution	10	20	25	25	20	95
Evaluations*10^6	801	790	1063	999	998	2014
σ Evaluations	157	184	177	187	156	78
Instructions	190131	187678	256207	240870	240227	477506
σ Instructions	40300	47350	45369	47971	40384	29289

Table 6: Tournament emigration (parity function)

Migration distance	100	400	1000	2000	∞
Fitness	34	34	37	57	97
σ Fitness	8	9	8	14	9

Table 7: Random emigration (image partitioning)

Migration distance	100	400	1000	2000	∞
Fitness	24	49	54	68	97
σ Fitness	8	14	23	14	9

Table 8: Emigration of the best (image partitioning)

Migration distance	100	400	1000	2000	∞
Fitness	41	33	44	58	97
σ Fitness	5	3	3	5	9

Table 9: Tournament migration (image partitioning)

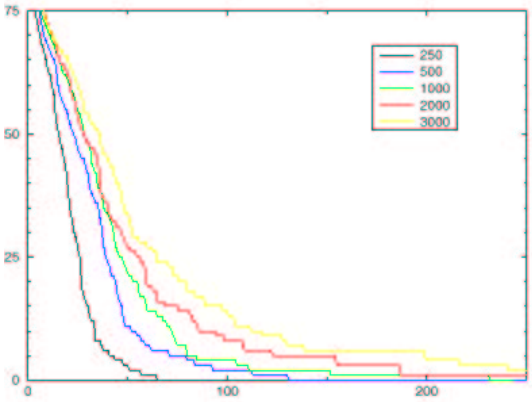

Figure 18: Best fitness over time (in million evaluations) for random migration and different migration frequencies (function regression)

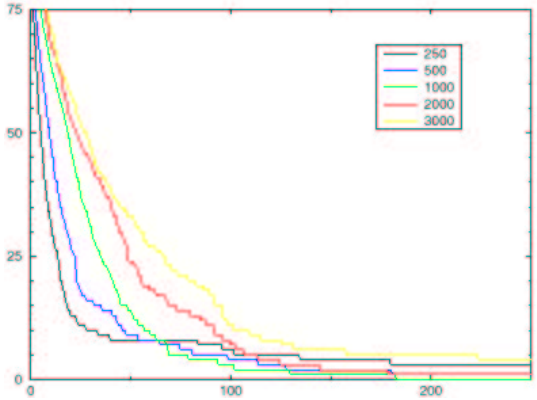

Figure 19: Best fitness over time for elitist migration and different migration frequencies (function regression)

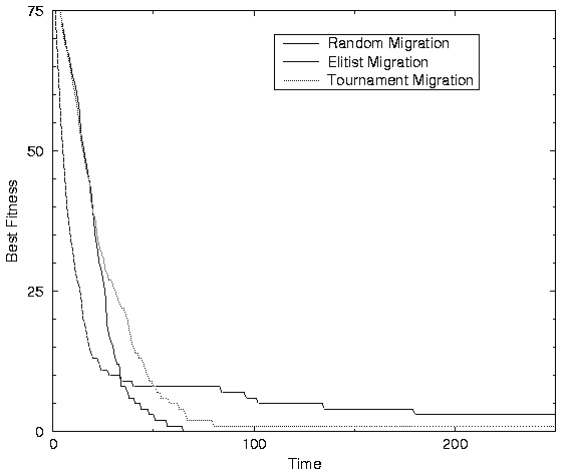

Figure 20: Progress of fitness for the different migration strategies and migration frequency 250 (function regression)

5. Summary

Our AIM−GP system has been parallized efficiently on a transputer network. Results have been presented for different problem domains using different migration strategies and migrations rates. In general the deme approach has proven a usefull concept for parallization .

Acknowledgement

Peter Nordin gratefully acknowledges support from the Swedish Research Council for Engineering Sciences.

Bibliography

[Andre and Koza 1996] Andre, D. and Koza, J. (1996) *Parallel Genetic Programming: A Scalable Implementation Using The Transputer Network Architecture*. In Angeline, P.J. and Kinnear, K.E. (eds.), *Advances in Genetic Programming 2*, MIT Press, Cambrige.

[Banzhaf et al., 1998] Banzhaf, W., Nordin, P. Keller, R. E., and Francone, F. D. (1998). *Genetic Programming — An Introduction. On the automatic evolution of computer programs and its applications.* Morgan Kaufmann, San Francisco and dpunkt Verlag, Heidelberg.

[Gorges-Schleuter 1990] Gorges-Schleuter, M. (1990) *Genetic algorithms and population structure — A massively parallel algorithm*. Ph.D. thesis, University of Dortmund, Germany.

[Kimura 1953] Kimura, M. (1953) *Stepping Stone Model of Population.* Annual Report of Nat. Gent. Japan, p. 62-63.

[Nordin 1997] Nordin, J.P. (1997) *Evolutionary Program Induction of Binary Machine Code and its Application*. Krehl Verlag, Münster, Germany.

[Nordin et al. 1999] Nordin, P., Banzhaf, W., and Francone, F. (1999) *Effective Evolution of Machine Code for CISC Architecture using Blocks and Homologous Crossover.* In Spector, L., Langdon, W.B., O`Reilly, U.-M., and Angeline, P.J. (eds.), *Advances in Genetic Programming 3,* MIT Press, in press.

[Tanese 1989] Tanese, R. (1989*) Distributed Genetic Algorithms.* In Schaffer, J.D. (ed.), *Proceedings of the 3rd International Conference on Genetic Algorithms*, Morgan Kaufmann.

[Wright 1953] Wright, S. (1943) *Genetics* 28, p.114.