Parallel Machine Code Genetic Programming

Markus Brameier **Frank Hoffmann** **Peter Nordin** **Wolfgang Banzhaf** **Frank Francone**

Abstract

AIMGP is a very fast linear genetic programming approach that evolves machine code programs. We report on a parallelization of AIMGP for a parallel transputer system resulting in an almost linear speedup.

In *linear* genetic programming (GP) computer programs of imperative programmming languages like C or machine code are evolved (Banzhaf et al. 1998). AIMGP (*Automatic Induction of Machine code by GP*) is a variant of linear GP where the evolving programs are represented as variable length sequences of binary machine code instructions that are directly executed during fitness calculation without interpretation. The method results in a significant speedup compared to interpreting GP systems.

For the parallelization of AIMGP described here we employ a steady state evolutionary algorithm using tournament selection. Parallelization of evolutionary algorithms is usually based on the observation that a population of solutions may be broken up into sub-populations (*demes*) while each of these subpopulations is run on a separate processor. Migration of individuals among the various demes causes evolution to occur in the population as a whole. This approach is inspired by the *island model* in biology and has been applied by Andre and Koza for the first multiprocessor implementation of a (traditional) GP system on a network of transputers (Andre et al. 1996).

The processing units of the Parsytec Power Explorer (sixteen here) are arranged in a matrix topology in that every node is connected to exactly four adjacent neighbors. These links determine the possible migration paths of the individuals. The migration technique used here is motivated by nature where migration is more-or-less *continuous*. During each migration, only *one* individual is selected from each deme node following a non-elitist migration strategy. An identical *copy* of that individual is moved to *all* four adjacent nodes in the transputer network. In this way, the deme from which the emigration originated is unchanged. The rate of migration is controlled by the frequency with which migrations occur (every 1000 tournaments here).

In this contribution results in relation to scalability are documented for a regression problem using the two dimensional objective function:
$f(x, y) = 5(x^4 + y^3) - 3x^2$.
In general a combination of parallel hardware and parallel algorithm is *scalable* if on average the product of the overall runtime and the number of processors remains constant with varying numbers of processors. In a perfectly scalable system the overall speed grows linearly with the number of processing units. In real systems scalability is restricted by the communication overhead between the nodes.

Table 1 shows the overall runtime of the parallel system until the optimal solution (fitness 0) has been found for different numbers of processors and demes respectively. Increasing the number of processing units results in a scaling factor of about 3/4 here which comes close to a linear improvement of speed performance (scaling factor 1).

Table 1: Average runtime (Ø) and standard deviation (σ) in 10^6 evaluations for different processor numbers and a constant overall population size (16000).

#Processors	1	4	16
#Individuals/Processor	16000	4000	1000
Runtime (Ø)	983	331	112
Runtime (σ)	276	101	34

References

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

W. Banzhaf, P. Nordin, R. Keller and F. Francone (1998) *Genetic Programming — An Introduction. On the Automatic Evolution of Computer Programs and its Application.* dpunkt/Morgan Kaufmann, Heidelberg/San Francisco.