Map design report

de Ronde, J.F.; Sloot, P.M.A.

Publication date
1994

Commission of the European Communities

****

ESPRIT III

PROJECT NB 6756

****

CAMAS

COMPUTER AIDED MIGRATION OF APPLICATIONS SYSTEM

****

CAMAS-TR-2.1.3.2
MAP Design Report

****

Date: September1994

ACE - U. of Amsterdam - ESI SA - ESI GmbH - FEGS - PARSYTEC - U. of Southampton.

Technical Report CAMAS-TR-2.1.3.2

Map Design Report

J.F. de Ronde and P.M.A. Sloot
University of Amsterdam

September1994

This design report has been submitted for evaluation to the CAMAS industrial partners of Workpackage 1: ESI and FEGS.

Contents

1 Introduction

The purpose of MAP is to aid in modelling the behaviour of parallel (SPMD) applications for which the process graph can be specified statically and is undirected. One should think of for example finite element or finite difference applications. Many of such applications can be seen to have a direct correspondence to a regular domain decomposition. For example a grid (one or more dimensions). Modern massively parallel architectures often offer as physical or virtual topology such regular structures onto which the static process graph that emerges from a regular decomposition can be mapped easily.

However for problems that do not have a regular communication pattern / process graph the optimal mappping may not be obvious in advance. For developing data parallel applications that contain irregular communicaton patterns it is useful to be able to experiment with various data mapping strategies and see what the effects are on the performance of the application. MAP will be utilized in combination with SAD and PARASOL to enable application developers to do qualitative analysis of application behaviour under different decomposition strategies. A near optimal process to processor graph mapping will be obtained by means of Genetic Algorithm (GA) optimization. Simple fitness functions will be used for this purpose. Modelling of the effects on performance of dynamic properties such as contention of message traffic in a processor network will be done by simulation (see SAD/PARASOL). The choice of using GA as optimization technique is based on the following.

- Physical optimization technique such as Simulated Annealing and Genetic Algorithms have shown to be succesfull in finding near optimal solutions for discrete optimization problems that show NP completeness in their solution space, see for example [1] and [4].
- The parallelizability of Genetic Algorithms is much better (in general) than e.g. Simulated Anealing.
- It is relatively simple to build genetic operators and add these modularly to a genetic algorithm kernel.
- For example Fox et al. (see [3] and [2]) have shown that genetic algorithms can work well for the problem of mapping finite element meshes on hypercubic topologies.

Basically these were the motivations to take GA as optimization strategy within the MAP tool. The next sections of the design report handles the following. In section 2 the way that MAP is supposed to be used within the CAMAS workbench is discussed. In chapter 3 the formats as chosen within the design for specifying a processor topology and a process graph are presented and clarified using a simple example. In chapter 4 the idea of mapping using a genetic algorithm of the example finite element mesh on the example processor topology is explained in some detail. Section 5 briefly adresses a time schedule for development of the specific work for the MAP subtask. In the appendix a pseudo genetic algorithm is presented as it is to be implemented within MAP. Furthermore briefly the interfacing of e.g. PAM-CRASH input files and DDT output to the MAP tool is discussed. Test results for a preliminary version of MAP are presented in appendix C.

2 MAP embedded in the toolset

In figure 1 the synergy of the toolset is shown with emphasis on MAP.

Within the Parasol I virtual machine one can specify a virtual or real processor topology. This topology can be represented in terms of a connectivity matrix. For each processor A to every other processor B a factor is given that corresponds to the route taken by a message sent from A to B and e.g. the relative throughput per link in this route. The route that is

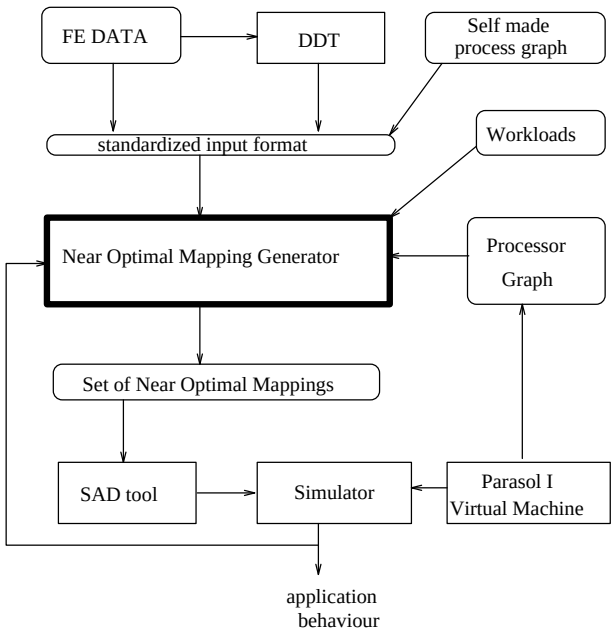

Figure 1: MAP embedded in the toolset

taken depends on the routing algorithm that has been build in by the manufacturer of the hardware platform under consideration. Below the routing problem is adressed when the formats for processor topology specification are presented.

On the other hand one can supply a static process graph that is handcrafted or derived from a finite element mesh (elements + connectivities) or a “super element”-mesh, that is a set of sub-domains of a mesh that has been arrived at via a domain decomposition technique (for example by Recursive Spectral Bisection (RSB), Orthogonal Recursive Biseciont (ORB) or scattered decomposition). Symbolically this is represented by a box DDT.

We consider the following possibilities in studying mappings:

- Mapping of a finite element mesh.
- Mapping sub-domains that are generated by a decomposition module (for example from DDT) from a finite element mesh.
- Mapping of a user defined connectivity graph which expresses the static process graph of an arbitrary problem that requires investigation.

The mapping procedure is constructed as follows: Using a relatively cheap (meaning that it requires relatively little time to evaluate) fitness function a genetic optimization is done. Intermediate solutions can be picked out. From such a specific individual solution a SAD simulation program is generated. Dynamic behaviour such as network contention can then be investigated using the SAD/PARASOL simulation environment. The result of this can be fed back in into the optimization procedure.

3 Cosmetics

The following section is dedicated to the form in which MAP expects input and a more detailed view into the construction of MAP.

3.1 Finite Element Mesh

A finite element mesh is characterized by its logical connectivity. Basically, the logical connectivity enumerates for every element in the mesh by which gridpoints it is constituted. The number of constituting gridpoints of a specific element depends on the shape of the element. Examples of possible element shapes are shown in figure 2.

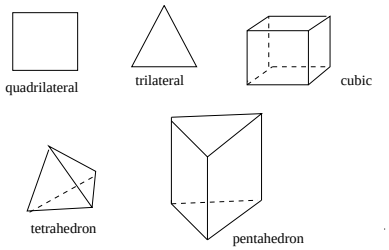

Figure 2: different element shapes

A logical connectivity that is to be read from file may appear in the following format:

Element Number Element Type

1	type	$a_{11}\ a_{12}$ ....a_{1m_1}
2	type	$a_{21}\ a_{22}$ ....a_{2m_2}
.	.	. .....
.	.	. .....
N	type	$a_{N1}a_{N2}$ ....a_{Nm_N}

For every element the type is given followed by the numbers of the gridpoints that constitute it. The number of gridpoints of an element depends on its "shape" property. So in the example logical connectivity table element 1 is formed by the gridpoints $a_{11}a_{12}....a_{1m_1}$. Figure 3 shows an example mesh. The elements are numbered from 0 to 15. The gridpoints are numbered from 1 .... 25. Only the numbers for the gridpoints that lie on the boundaries of the mesh are displayed.

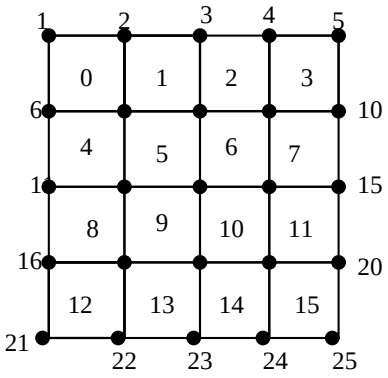

Figure 3: mesh example

The logical connectivity for this mesh is given for the first few elements by:

Element Number Element Type

0	quad	1 2 6 7
1	quad	2 3 7 8
2	quad	3 4 8 9

etc....

From this logical connectivity the element connectivity as it will be used in MAP can be derived. For this specific example this will become for the first few elements:

Element
0 1 2 4 2 5 1
1 0 2 2 2 4 1 6 1
2 1 2 3 2 5 1 7 1
etc....

This means: Element 0 shares 2 gridpoints with element 1, 2 gridpoints with element 4 and 1 gridpoint with element 5 etc... If after mapping a finite element mesh to a processor network two connected elements reside on different processors, (in first approximation) a message that is proportional to the number of gridpoints they share has to be exchanged between those processes.

Furthermore, with every element type a static workload must be associated. The determination of this (relative) workload is a task of the user of the MAP environment. A possible path in determining the work for a specific element type on a specific machine is by applying the performance estimation tools of the CAMAS Workbench.

3.2 Processor Topology

A processor configuration for MAP will be characterized by a connectivity matrix. An element $P_{pq}(p \neq q)$ denotes the relative (or absolute) time it takes for a unit sized message to travel from processor p to processor q.

Within the PARASOL I virtual machine model one will be allowed to specify a parallel machine by the processors that it is constituted from with the nearest neighbour links and their relative throughput. Figure 4 shows an example of a mesh of processors with different link types (a and b) as well as different processor types (X and Y).

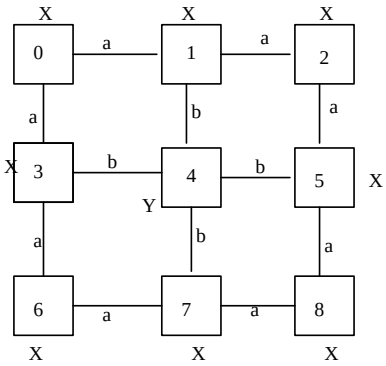

Figure 4: processor config example

In the following table the representation of such a mesh as how it could be initialized within the PARASOL I tool is given.

processor connected to other processors with link speeds.

0 1 a 3 a
1 0 a 2 a 4 b
2 1 a 5 a
3 0 a 4 b 6 a
4 1 b 3 b 5 b 7 b
5 2 a 4 b 8 a
6 3 a 7 a
7 4 b 6 a 8 a
8 5 a 7 a

This has to be read as processor zero has two directly neighbouring processors (1 and 3). Both processors are connected to processor 0 via links with relative throughput a.

Unfortunately this information is not sufficient to find out what costs are associated with sending a message between processor 0 and 8 for example. The route that is taken by this message is dependent on the routing mechanism of the hardware (e.g. simple X-Y routing where the route between two processors is defined by alternating steps in X and Y direction via the links of the grid topology. We assume in our modelling that every processor pair is connected via a unique route when exchanging messages. It is not possible to incorporate dynamically changing routing schemes. What can be done is using different routing schemes and implementing these within PARASOL I.

Given the processor mesh example in figure 4. So e.g. a message send from processor 0 to processor 8 migrates consecutively over links with speed of a, b, b and a. The processor connectivity matrix $P_{pq}(p \neq q)$ is what is used by MAP. It has to be build within PARASOL I in which routing strategies can be embedded.

3.3 Units

A finite element mesh may first be run through a preprocessing module that partitions the mesh in a set of M so called "units". Examples of partitioning algorithms are RSB, ORB or scattered decomposition.

Figure 5 shows the example mesh that is partitioned into 4 parts. A unit connectivity has the same format as a normal element connectivity. For every part (unit) the row of constituting elements is enumerated. Furthermore the "unit connectivity" is given.

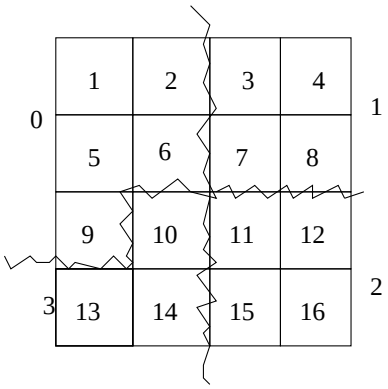

Figure 5: a mesh partitioned into 4 units

In case of the example mesh

Unit
0 1 3 2 1 3 4

1 0 3 2 3 3 1
2 0 1 1 3 3 3
3 0 4 1 1 2 3

Meaning that unit 0 shares 3 gridpoints with unit1, 1 gridpoint with unit 2 etc....

The Workload associated with a unit is the summation of the workloads of the individual elements that constitute the unit.

4 Mapping

The search for a near optimal mapping of elements or units onto a processor configuration will be done by means of a genetic algorithm. The form of a chromosome on which genetic operators will act is encoded as a string of integers, where an integer refers to a processor and its position in the string represents the allocated data element.

So for example for the finite element mesh that is mapped on the processor configuration (see figure 6) the corresponding chromosome is given by:

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \end{pmatrix}$$

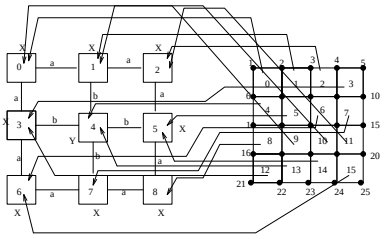

Figure 6: a map of a FE mesh on a processor grid-topology

An initial population of chromosomes can be generated randomly. Work by Fox et al. (see [3]) has shown that this is a reasonable heuristic to initiate the genetic optimization from for the type of problems that we are adressing.

4.1 Fitness

Two examples of costfunctions functions were given in technical report CAMAS-TR.2.1.3.1 : (units are equivalent with elements)

$$C = \max_{q \in \mathcal{Q}} \sum_{u_i \in \mathcal{U}^q} W_{ii}$$

where $\mathcal{Q}$ is the set of processors and $\mathcal{U}^q$ is the set of units u_i residing on processor q.

This costfunction only takes workloads into account. Thus basically it represents a model of a fully connected machine with infinite communication bandwidth.

A more realistic costfunction might be:

$$C = \max_{q \in \mathcal{Q}} \left(\sum_{u_i \in \mathcal{U}^q} W_{ii} + \max_{u_i \in \mathcal{U}^q, u_j \in \mathcal{A}(u_i)} P_{pq} W_{ij} \right)$$

where $\mathcal{A}(u_i)$ is the set of neighbouring units for unit u_i.

In this case the costfunction takes into account the connectivities in the workload graph as well as the processor topology via the factor P_{pq}. This will force the maximum distance between the processors onto which connected units are mapped to be minimal while at the same time for optimizing the workload distribution.

It should be noted that the multiprocessors in the above two models are assumed to consist of processors that have identical performance rate. That's why the factor describing this rate P_{qq} can be omitted in the costfunctions above.

Since many evaluations of the above costfunctions are necessary within genetic algorithm optimization it is important to find a function of which the evalution is not too expensive. Fox et al, see [2] have proposed an approximate function that has shown to achieve good results. In the genetic algorithm that is to be used by MAP a similar objective function is going to be used. At points where it becomes necessary to be more exact in the solution a more expensive function is taken.

Finally the fitness of a specific mapping is defined as the reciprocal of the "costfuntions" introduced above.

4.2 The Genetic Algorithm

Generally a genetic algorithm consists of several operations that transform a population of chromosomes into a new one. The reproduction operation basically is the operation that is responsible for the tendency that consecutive populations that are generated evolve in the direction of higher average fitness. The other operations are introduced to create populations with high diversity.

GA is a blind search technique. Very often the implementation of an efficient GA is confronted with the problem of premature convergence into local optima. On the other hand an implementation that is not specifically efficient may take very long times to reach a good suboptimal solution.

Incorporating problem specific knowlegde in the blind GA search to direct it to the fruitfull regions in the search space is a possible techique to get more efficient GA's. So called hybrid techniques take can take into account domain specific knowledge in their search. The hybrid genetic algorithm for our purposes has to satisfy the following conditions: The likelyhood of premature convergence has to be minimized, the search efficiency must be high while computational costs are kept as low as possible, and problem specific knowlegde has to be utilized. For example work by Fox et al, see [2] has shown that such a hybrid genetic algorithm can provide good results.

Several operators can be distinguished that are typical for the genetic algorithm kernel.:

- Reproduction
- Crossover
- Mutation and Inversion

4.2.1 Reproduction

Reproduction of a population of size of N individuals produces a new population of size N. The probability that an individual that appeared in the old population will be reproduced and put in the new one is higher when the fitness of the individual is higher. This can be done e.g.according to the so called biased "wheel of fortune" trick, see [1], or "ranking", see [2]. Ranking will be adopted as reproduction scheme for the genetic algorithm within MAP. It has shown to be able to control premature convergence and to be computationally relatively cheap compared to other methods.

4.2.2 Crossover

In “Crossover” the population that was generated by “Reproduction” is manipulated into a new population. Randomly two chromosomes are picked out of the population. Then two new chromosomes are generated using a crossover operation. Figure 7 gives some crossover operations that may be applied: one point crossover and two point crossover.

A so called two-point ring-like crossover will be used within MAP instead of a more sophisticated crossover technique in order to avoid too much computational demand.

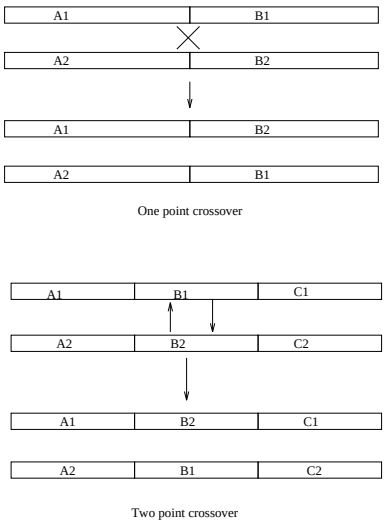

Figure 7: crossoverexample

4.2.3 Mutation and Inversion

Mutation allows a small fraction of a chromosome to change spontaneously. These operations allow the genetic search to spontaneously direct the search to a region in the space of solutions that otherwise wouldn’t be reached. In the specific problem that is adressed within mapping of a task graph formation of clusters of “adjacent tasks” on the same processor is higly desirable. Mutation therefore is restricted to tasks/elements that reside on the boundaries of “clusters”. A task is said to be on a boundary when at least one of its nearest neighbour tasks is residing on another processor.

Inversion within a chromosome is also allowed with some inversion probability. The chromosome is considered as a ring. A contiguous section of this ring gets inverted. In figure 8 this situation is depicted.

4.3 Hill Climbing

A simple hill climbing technique to direct the blind GA search into the more profitable regions of the search space is added to genetic algorithm of MAP. In the hill climbing “boundary elements” are considered one at a time. A boundary element is transferred from the processor where it resides to a processor on which a neighbouring element resides if the fitness of the chromosome increases or stays the same. In this way locality within the task graph can be utilized to approach optimal solutions faster.

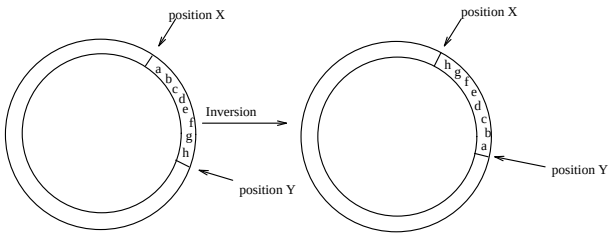

Figure 8: A ring-like chromsome with a substring inverted

4.4 Simulation Generation

From an individual that was generated by the genetic algorithm a simulation program for the Parasol II simulator can be generated. The results of such a simulation can again be fed back into the genetic algorithm as a "higher level" fitness function value.

5 Time Schedule

The following time schedule for the progress of MAP in the CAMAS project is proposed:

- Review 4: Deliverable: MAP Design Report (this report)
- Review 4: Pseudo Code for MAP Genetic Algorithm Kernel(included in report)
- Review 4: Interface for reading PAM grids and DDT output (logical connectivities) and transforming into format required by MAP.
- Review 5: Deliverable: Report on Progress of MAP.
- Review 5: Demonstration of Mapping example finite element grids to processor configuration (directly or partitioned by DDT).
- Review 5: SAD level 3 simulation code generation from a proposed mapping
- Review 6: Deliverable : final report on MAP
- Review 6: Demonstration of MAP:
  - Mapping finite element grids (preprocessed or not) using genetic mapping module
  - choice of various fitness functions for mapping
  - generating from intermediate mapping solution a SAD level 3 simulation program
  - Simulation using PARASOL II (simulator)
  - Feedback of simulation result into mapping module
  - Evaluation example: direct mapping of a finite element grid versus mapping of same but now partitioned grid

A Pseudo Genetic Algorithm for process-processor mapping

```
Choose a Fitness Function
Choose number of Chromosomes in a population
Choose mating strategy ( one or two point crossover, mutation probability, ...)
Read in Unit or Element information
Read in processor topology information
Create Initial Population
While (No convergence)
        Reproduce :
                Calculate Total Fitness of Population
                For Number of Chromosomes
                    Reproduce one of the Chromosomes using ranking
                EndFor
        Crossover:
                Generate (Number of Chromosomes/2) pairs
                randomly
                do crossover on every pair according to the chosen
                crossover strategy
        Mutation:
                For all Chromosomes in the population
                    mutate a boundary - element in the chromosome with
                    mutation probability
                Endfor
        Inversion
                apply inversion with small probability
        Apply hill-climbing for newly generated chromosomes
        If intermediate solution is requested
                Filter out momentarily "best" solution and
                feed it to the SAD program generator
        Endif
        If convergence criterium is satisfied
                set Convergence TRUE
End While
```

B On Input format

Perl (Practical Extraction and Report Language) is very well suited for building reasonably fast an interface that can extract from for example a PAM-CRASH inputfile the logical connectivity and cast it into such a format that the MAP tool can handle it. The same goes for data that is generated by DDT. Therefore Perl is used for interfacing between real data files (such as for PAM-CRASH) and the MAP tool.

C MAP Preliminary Version

A preliminary version of MAP has been implemented. Results of a first test are presented here. The test took as input the logical connectivity of a mesh consisting of 640 quadrilateral finite elements. The mesh is the PAM-CRASH input mesh "box.inp" and has been provided by ESI.

Furthermore a simple processor topology of 8 processors was taken as the processor graph. All processors have equal processor power. Figure 9 shows this processor mesh.

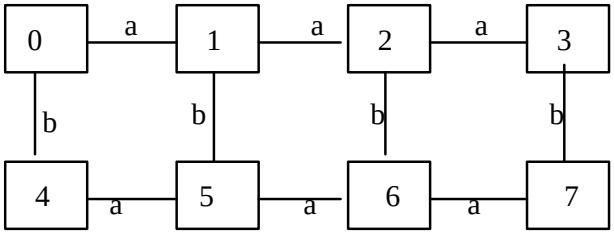

Figure 9: a processor grid with 8 identical processors and 2 types of links, type a and type b

It is assumed that the routing strategy is X-Y routing which has as a result the following processor connectivity matrix:

$$\begin{pmatrix} 0 & a & 2a & 3a & b & a+b & 2a+b & 3a+b \\ a & 0 & a & 2a & a+b & a & a+b & 2a+b \\ 2a & a & 0 & a & 2a+b & a+b & b & a+b \\ 3a & 2a & a & 0 & 3a+b & 2a+b & a+b & b \\ b & a+b & 2a+b & 3a+b & 0 & a & 2a & 3a \\ a+b & a & a+b & 2a+b & a & 0 & a & 2a \\ 2a+b & a+b & b & a+b & 2a & a & 0 & a \\ 3a+b & 2a+b & a+b & b & 3a & 2a & a & 0 \end{pmatrix}$$

The preliminary version of MAP consists of an interface that reads in the logical mesh connectivity, workload file per element type, a processor connectivity matrix and several operator rates for the genetic algorithm. The genetic algorithm that has been implemented is equivalent to the pseudo code given above, except that the "Hill-Climbing" is not included yet.

Figure 10 shows an example of how the fitness function of this system evolves in the range of 1000 iterations, where the link speeds are taken a = 1 and b = 2. Such parameters will become available from within Parasol in the future. For now they are simply user defined. The population contains 100 chromosomes. The fitness function used is the reciprocal of the function:

$$C = \max_{q \in \mathcal{Q}} \left(\sum_{u_i \in \mathcal{U}^q} W_{ii} + \max_{u_i \in \mathcal{U}^q, u_j \in \mathcal{A}(u_i)} P_{pq} W_{ij} \right)$$

that was introduced above.

It is clear that the fitness evolves in the direction of higher fitness (due to the reproduction procedure).

We are planning to optimize the algorithm where possible. The algorithm appears to work fine although for large problems it is likely to be very slow. At such a point it probably will not be feasible anymore to look at mapping of "individual" elements, but look at "units".

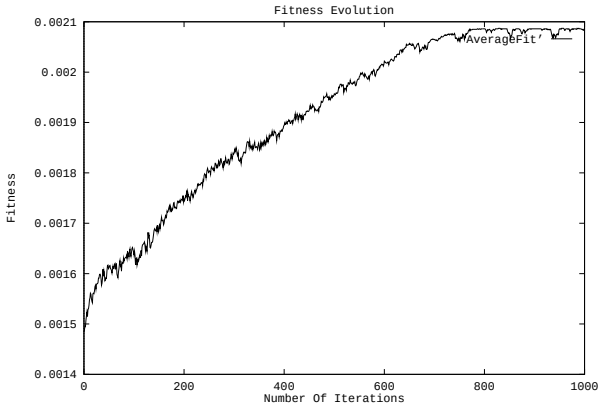

Figure 10: Fitness Evolution - 1000 Iterations

References

[1] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, 1989.

[2] N. Mansour and G. Fox. A hybrid genetic algorithm for task allocation. In *Proceedings of the 4th International Conferencen on Genetic Algorthims*, pages 466–473, July 1991.

[3] N. Mansour and G. Fox. Allocating data to multicomputer nodes by physical optimization algorithms for loosely synchronous computations. *CONCURRENCY: PRACTICE AND EXPERIENCE*, 4(7):557–574, OCTOBER 1992.

[4] P. J. M. van Laarhoven and E. H. Aarts. *Simulated Annealing: Theory and Applications*. D. Reidel Publishing Company, 1987.