Angewandte Mathematik und
Informatik
Universität zu Köln

Report No. 97-269

Dynamic Load Balancing for Simulations of Biological Aging

by

F. Meisgen

1997

published in: *International Journal of Modern Physics C*, Vol. 8, Nr.3, June 1997, pages 575–582

F. Meisgen
Department of Computer Science
Cologne University
50923 Köln, Germany
E-mail: meisgen@informatik.uni-koeln.de

1991 Mathematics Subject Classification: 68Q22,68U20
Keywords: dynamic load balancing; heterogeneous system; simulation; biological aging

Dynamic Load Balancing for Simulations of Biological Aging

Frank Meisgen
Department of Computer Science
Cologne University
50923 Köln, Germany
E-mail: meisgen@informatik.uni-koeln.de

April 10, 1997

Abstract

The efficient usage of parallel computers and workstation clusters for biologically motivated simulations depends first of all on a dynamic redistribution of the workload. For the development of a parallel algorithm for the PENNA model of aging we have used a dynamic load balancing library, called PLB. It turns out that PLB manages a nearly balanced load situation during runtime taking only a low communication overhead. We compare different architectures like parallel computers and nondedicated heterogeneous networks, and give some results for large populations.

1 Introduction

Biologically motivated simulations need parallel computing for large populations. In Darwinistic evolution, selection of the fittest may finally lead to the effect that all survivors are offsprings of a comparatively small set of ancestors. This yields the effect that all individuals are simulated on the same processor, leaving the other nodes of the processor network idle. Moreover simulations on workstation clusters may lead to further imbalances if some nodes are more used or slower than others. Thus an efficient load balancing is necessary for fast long time simulations on large populations.

We consider a parallel version of the PENNA model [6] of aging, which is widely used for Monte Carlo studies of age-structured populations [7, 1]. In this model an individual is composed of a bit-string of length $l = 32$ representing its genes and its actual age. If a bit has value one, this gene is damaged by a mutation. An individual of age j dies if there at least k mutations of the first j bits. It also dies if it reaches its maximal age l or if it is killed by external

influences like starvation. This is modeled by the Verhulst factor $1 - N/N_{max}$, where N is the actual population size and N_{max} is the maximal capacity of the ecosystem. After an individual reaches its reproduction age a, every year it gives b offsprings. Every offspring inherits the genes from its mother, further a randomly chosen bit is set to one. In our experiments, we choose a=8, b=1 and k=4.

The parallel algorithm distributes the population on n machines of a heterogeneous or homogeneous network. After each processor has executed an evolution step, the total size $N = \sum_{i=1}^{n} N_i$ of the population must be calculated to determine the Verhulst factor. This leads to a synchronization of the processors after each simulation step, causing idle times due to different runtimes. These differences can be created by

1. different computing power (hardware, utilization)
2. different workload (population size, age structure)

For reducing these effects we use a dynamic load balancing procedure, the Precomputation-based Load balancing algorithm (PLB [2, 3, 5]), which redistributes parts of the population if the workload is too imbalanced. In the experimental part (Sect. 4) we will first consider homogeneous dedicated networks, then heterogeneous local area networks and finally heterogeneous wide area networks.

2 Precomputation-based Load Balancing

We give a short description of the PLB algorithm on heterogeneous networks, for more details see [5]. Readers not interested in these aspects may proceed to section 4. The processor network is represented by a vertex-weighted connected graph $G(V, E, w)$ with $w : V \to \mathbb{R}^+$. Each node $v \in V$ is identified by a processor, which has a *power weight* $w(v)$, defined as follows. A sequential process, which executes T time units on processor v will need $w(v) \cdot T$ time units on the benchmark processor v_1. Hence, $w(v_1) = 1.0$ and $w(v) < 1.0$ if v is slower than v_1. The power weight of a subset $V' \subseteq V$ of the network is defined as $w(V') := \sum_{v \in V'} w(v)$. Two nodes are connected by an edge, iff the processors are connected in the virtual topology created by MPI.

Denote by $\lambda_t(v)$ the amount of workload of processor v at time t. We refer to $\lambda_t(v)$ as the *load distribution* function of the network. The load $\lambda_t(v)$ of processor v is assumed to be proportional to the time for executing the next simulation step $t + 1$ for this population on processor v_0. Then $\overline{\lambda_t} := \frac{1}{w(V)} \sum_{v \in V} \lambda_t(v)$ is called the *mean load*. If time t is fixed, we omit the index t. The load of a processor v is called *optimal*, if $\lambda(v) = w(v) \cdot \overline{\lambda} =: \lambda_{opt}(v)$.

The PLB algorithm calculates in a fast precomputation phase for each processor the amount of load, which should be exchanged to achieve an equalized

load distribution. In the following we describe the precomputation phase on a weighted processor tree $T = (V, E, w)$ of height h and root r. T_v denotes the subtree of T rooted at v.

1. Every processor $v \in V$ determines the load and weight of his subtree by first receiving these information from its children and afterwards sending $\lambda(T_v)$ and $w(T_v)$ to its parent:

$$\begin{aligned} \lambda(T_v) &= \lambda(v) + \sum_{u \in children(v)} \lambda(T_u) \\ w(T_v) &= w(v) + \sum_{u \in children(v)} w(T_u) \end{aligned}$$

2. The root r of T broadcasts the *mean load* $\overline{\lambda} := \frac{\lambda(T_r)}{w(T_r)}$ to all processors.

With this information every processor can determine the amount of load it has to send to (or receive from) its parent and to its children. If $\lambda(T_v) - w(T_v) \cdot \overline{\lambda}$ is positive, v has to send this amount of load to its parent, otherwise v has to receive it. The same holds for every child u with $w(T_u) \cdot \overline{\lambda} - \lambda(T_u)$.

During the balancing phase load is exchanged according to this scheme. The execution time for this step depends primarily on the maximum of load a processor has to move and the diameter of the network [3].

3 Details of Implementation

We use the number of individuals on a processor as rough load estimation of the workload in the next time step. This is only an approximation of the real workload, a newborn child cause more work than simply increasing the age of an individual, thus an individual which will be reproductive in the next time step is expected to cause more work than a younger animal.

We have compared the PLB strategy with a no load balancing algorithm (NLB), distributing the total initial population in equal parts on the processors and then performing no more load balancing. The coarse-grained structure of the algorithm is listed below, every processor executes this code, a synchronization is done in the second step.

foreach *year* **do**

1. Simulate year

2. Calculate total population size	**NLB**

or

2a. Precomputation phase 2b. Balance if necessary	**PLB**

done

The balancing phase (2b.) is only executed if the highest execution time differs from the lowest one in more than α percent. We have investigated in our experiments the influence of the parameter α and call the corresponding algorithm PLB-α. It will turn out that for workstation clusters much higher values are necessary for α than for dedicated parallel computers.

All algorithms are implemented in C++ and use the mpich 1.0.13 implementation of the MPI 1.1 standard [4]. The program was compiled with the -O4 option of the GNU C++ compiler, version 2.7.2 (2.7.0 for PARIX). The simulation algorithm is based on an implementation of P.M.C. DE OLIVEIRA.

4 Experimental Results

For all network types a population of initially 300 000 animals per processor and a final total size of 6.6 (3.2) million on 16 (8) processors after 4096 simulation steps was used as test instance. For heterogeneous networks the initial population of 2.4 million individuals (8 processors) was shared out according to the productivity of the machines. In all experiments the processors were arranged as a linear array in the virtual MPI topology.

4.1 Homogeneous Networks

Our test platform is a dedicated parallel system at the PC^2 (University of Paderborn), a GCPP with 32 nodes each containing two Power-PCs type 601 (80 MHz) with 64MB RAM together.

Table 1: Results for an homogeneous network with 16 nodes, where t_{bal} is the average time needed for balancing, the average number of exchanged individuals correspond to λ_{bal} and t_{bal}/λ_{bal} is the average time for exchanging a million individuals.

Parameter	runtime (avg)	idle (avg)	balance count	t_{bal}	λ_{bal} $(\cdot 10^6)$	t_{bal}/λ_{bal}
NLB	5923.71s	4297.32s	-	-	-	-
PLB-50	1885.53s	252.16s	15	13.41s	1.524	8.80s
PLB-10	1711.32s	86.95s	71	13.66s	1.643	8.31s
PLB-05	1697.92s	71.07s	155	16.38s	1.792	9.14s
optimal	1677.15s	56.55s	-	-	-	-

For determining the influence of inaccurate time measurement and communication overhead we have executed the same program with the same random seed on all processors, so that during runtime all population sizes are equal. The values for this experiment are listed in the last row (called optimal), this is a lower bound for a parallel algorithm. It is notable that the PLB-05 algorithm

reaches nearly this optimal value. Furthermore the runtime could be reduced by a factor of three compared with the NLB algorithm. In this case the distribution of population was extremely imbalanced at the end of simulation (year 4096), it scatters from 21 055 to over 2 million. The quality of the load balancing done by PLB can be adjusted by the balance parameter. Smaller values for α lead to higher numbers of balancing runs (column 4), a small increase of balancing time (t_{bal}) and insignificant more exchanged animals (λ_{bal}). In the PLB-05 case only 425 individuals per processor are exchanged after each simulation step on the average, this is less than 0.01% of the average population on a processor.

4.2 Heterogeneous Networks

The local area network of the test environment is shown in the upper part of Fig. 1, it consists of different types of Suns SPARCstations varying from a Sparc ELC to a Hypersparc with 90MHz connected by a 10MBit Ethernet. The wide area network is shown beneath, it is composed of four SPARCstations and four Sun ultras. Processors, which are part of the same LAN, are enclosed by a rectangle marked by the domain name.

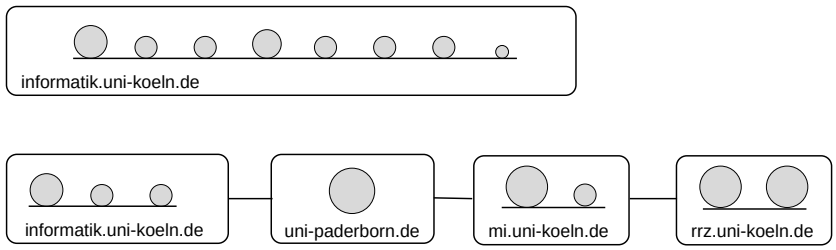

Figure 1: *Test environment, the diameter of a processor is proportional to its power weight, the position corresponds to the order in the virtual MPI topology. The LAN is shown in the upper part, the WAN below.*

To measure the power weight of different machines, we performed three test runs doing a simulation of 1024 years with an initial population of 300 000 animals. For every machine the run with minimum time was recorded and compared with the run on the benchmark processor, a SPARCstation-10 with a 90MHz hyperSPARC CPU. These experiments were made while the workstations were performing their usual tasks, thus we were working in a nondedicated environment. Furthermore all tests were run with low priority.

4.2.1 Local Area Heterogeneous Networks

Table 2 contains results for the algorithm without load balancing (NLB), PLB with parameter $\alpha = 50$ and finally PLB with a dynamic adaption of power

weights (PLB-50-d). For details of the dynamic adaption strategy please refer to [5]. In comparison with dedicated parallel computers the idle times of

Table 2: Results for an heterogeneous LAN with 8 processors

Parameter	runtime (avg)	idle (avg)	balance count	t_{bal}	λ_{bal} $(\cdot 10^6)$	t_{bal}/λ_{bal}
NLB	5604.18s	3478.91s	-	-	-	-
PLB-50	2735.03s	551.18s	4091	92.17s	2.73	33.76s
PLB-50-d	2563.33s	369.23s	266	82.95s	3.78	21.94s

nondedicated heterogeneous networks are significantly higher. This is caused by the higher variation of runtimes which depends on the scheduling of competing tasks done by the operating system. Thus it was necessary to choose higher values for α to avoid unnecessary balancing steps. Another interesting observation is that the time needed for balancing is more influenced by the number of balancing steps than by the amount of exchanged load.

4.2.2 Wide Area Heterogeneous Networks

We have performed the same three tests for WANs as for local area networks, the results are collected in Tab. 3. It can be observed that the time needed to

Table 3: Results for an heterogeneous WAN with 8 processors

Parameter	runtime (avg)	idle (avg)	balance count	t_{bal}	λ_{bal} $(\cdot 10^6)$	t_{bal}/λ_{bal}
NLB	4835.86s	3616.15s	-	-	-	-
PLB-50	2727.37s	1191.66s	3809	419.21s	1.368	306.44s
PLB-50-d	3363.57s	1297.58s	2082	1045.80s	17.044	61.36s
PLB-100-d	2245.15s	918.10s	452	244.67s	3.339	73.28s

exchange problems has grown (last column), probably caused by the latency of the network. Furthermore the variance of runtimes increases, causing significantly higher idle times and more balancing steps. To avoid these unnecessary steps we fix the parameter $\alpha = 100$ and calculate the power weights on a bigger sample, leading to the results shown in the last row of Tab. 3.

Thus for the efficient usage of nondedicated distributed networks a more coarse-grained structure of the parallel algorithm is necessary then for dedicated parallel computers, so in our application the time of a simulation step should be increased.

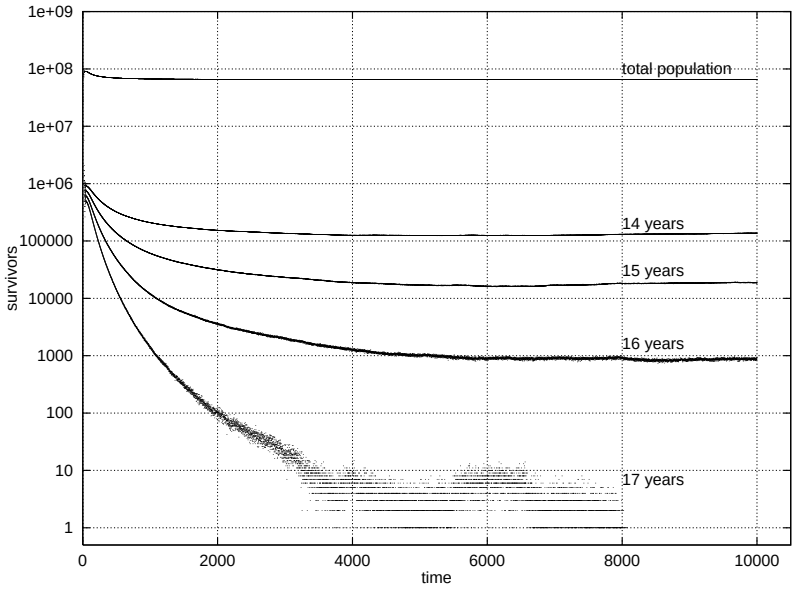

Figure 2: *Equilibration of total population size and number of survivors for ages 14,15,16 and 17.*

4.3 Results for Large Populations

To demonstrate the usefulness of the PLB algorithm on larger populations and networks we have simulated about 65 million animals over a period of 10 000 years on the GCPP with 32 nodes. In Fig. 2 the total population size and the number survivors for different ages are plotted as functions of simulation years. It is remarkable that with increasing age the function comes much later into equilibrium.

The survival rate $S_t(a)$ is defined as $N_t(a)/N_{t-1}(a-1)$ where $N_t(a)$ corresponds to the number of individuals of age a at time-step t. The scaled mortality rate defined as $ln(S_1/S_t)$ is plotted in Fig. 3, showing a nearly exponential increase like a Gompertz law.

5 Conclusion

It has been shown that a dynamic load balancing makes it possible to use large parallel computers and workstation networks for longtime simulations without heavy losses of efficiency. For small populations it is much more difficult to achieve a reasonable speedup on nondedicated systems with unpredictable load changes and the result of the load balancing algorithm depends strongly on the used parameters. In our actual research we develop adaption strategies for

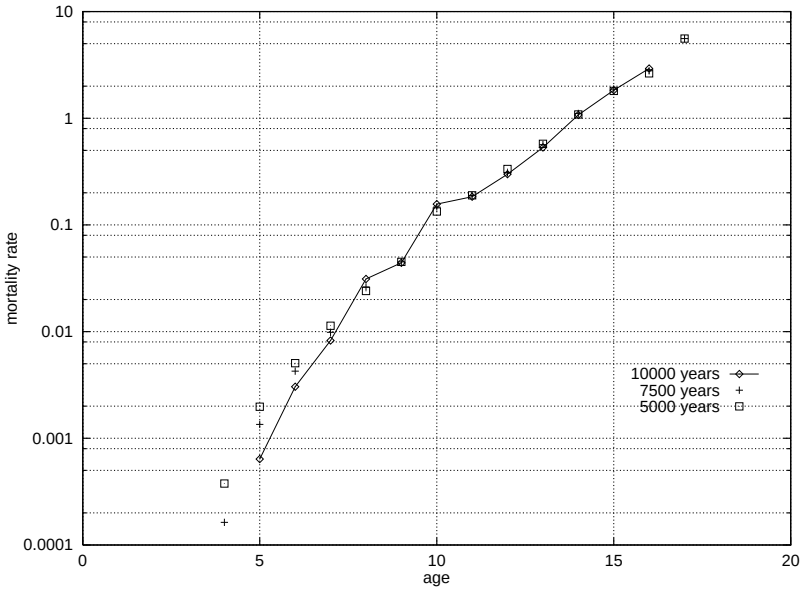

Figure 3: *Mortality rate at different simulation time steps.*

parameters and investigate the influence of the choice of the virtual topology. Furthermore we are going to extend the number of applications to more complicated simulations, where parts of the simulation graph have to be distributed with respect to local dependencies.

6 Acknowledgments

The author thanks D. Stauffer for all the helpful comments and discussions and P.M.C. de Oliveira for his sequential algorithm.

References

[1] A. T. Bernardes. Monte Carlo Simulations of Biological Aging. In D. Stauffer, editor, *Annual Reviews of Computational Physics*, page 359, Singapore, 1996. World Scientific.

[2] M. Böhm and E. Speckenmeyer. A fast parallel SAT–solver — efficient workload balancing. *Annals of Mathematics and Artificial Intelligence*, 17:381–400, 1996.

[3] M. Böhm and E. Speckenmeyer. Precomputation based load balancing. Technical Report 96.219, Universität zu Köln, 1996. To appear in *Proceedings of the 4th PASA Workshop,1996.*

[4] The Message Passing Interface Forum. The Message Passing Interface Standard. http://www.mcs.anl.gov/mpi/, June 1995.

[5] F. Meisgen and E. Speckenmeyer. Dynamic Load Balancing on Heterogenous Workstation Clusters. Technical Report 97.261, Universität zu Köln, 1997. submitted to: *Euro-Par'97.*

[6] T. J. P. Penna. A Bit–String Model for Biological Aging. *J. Stat. Phys.*, 78:1629, 1995.

[7] D. Stauffer. Getting older - Monte Carlo simulations of biological aging. *Computers in Physics*, 10:341, 1996.