ELSEVIER

Future Generation Computer Systems 12 (1997) 391–406

FGCS
FUTURE GENERATION COMPUTER SYSTEMS

Load balancing by redundant decomposition and mapping

J. F. de Ronde*, A. Schoneveld[1], P. M. A. Sloot[2]

Department of Mathematics and Computer Science, University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands

Abstract

In this paper a new methodology for load balancing parallel processes on parallel systems is proposed. The problem of load balancing is considered to be an NP-hard optimization task. Taking static parallel finite element applications as a case study, the benefits and losses that follow from applying the methodology are studied. It is found that the proposed methodology can be especially useful for load balancing in asymmetric processor topologies, and therefore is of importance for work load balancing in workstation clusters.

Keywords: Redundant domain decomposition; Mapping; Graph based parallel process/processor modelling

0. Background

Experience gained in the CAMAS[3] project [27], indicates that in the community of parallel application developers, a strong need exists for various tools to support efficient code development. One can distinguish, e.g. tools for code analysis, performance evaluation and load balancing of parallel applications. The CAMAS project has focussed on the development of methodologies on which such tools can be based. The methods that were developed have been implemented in an integrated workbench.

Fig. 1 shows an overview of the CAMAS tool set. The following tools can be identified:

(i) *Code Analysis tools*: Inter procedural dependency analyzer (IDA) [19] and Fortran to Symbolic application description translator (F2SAD) [29].
(ii) *Performance evaluation tools*: Parallel machine modeling (Parasol I) [4] and parallel performance prediction (Parasol II) [8, 29].
(iii) *Load balancing tools*: Domain decomposition tool (DDT) [9] and Process mapping tool (MAP) [6].

In this paper the focus is on load balancing of parallel processes on parallel systems. Within the tool set, Fig. 1, the grey boxes and ovals denote the various stages in which load balancing was approached in the CAMAS project. A preprocessing

* Corresponding author. Tel.: + 31 20 525 7463; fax: + 31 20 525 7490; E-mail: janr @wins. uva.nl.

CAMAS: Computer Aided Migration of Applications System, ESPRIT III project number 6756, September 1992/September 1995.

[1] E-mail: arjen@wins.uva.nl.
[2] E-mail: peterslo@wins.uva.nl.

SSDI 0167-739X(97)00030-5

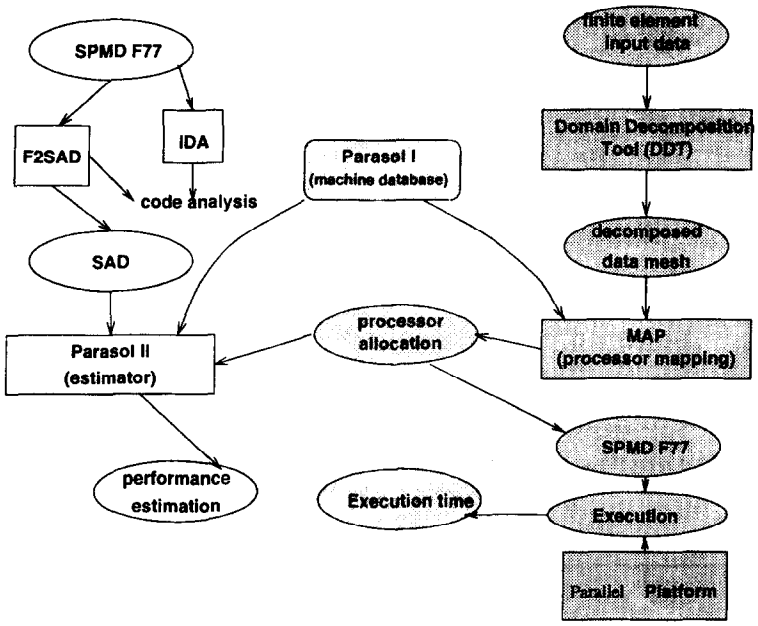


Fig. 1. The CAMAS workbench.

step creates a domain decomposition of a (finite element) mesh using DDT, followed by a mapping phase, where the emerging parallel processes are mapped on a parallel machine using MAP. A major unresolved problem in this load balancing approach is the optimal cardinality of the domain decomposition, given a certain target parallel architecture, which is the topic of this work.

This paper is structured as follows. In Section 1 the load balancing problem is introduced in a formal manner. An introduction to the methodology of redundant decomposition and mapping to approach the load balancing problem is given. Next, machine and application models are formulated that can be used to express the load balancing problem in terms of an NP-hard combinatorial optimization task. Furthermore, the properties of two well-known cost models, that can be used to indicate the quality of process allocations are summarized. In Section 2 the optimization kernel of the MAP tool is presented. Three heuristic optimization strategies that can be formulated within this framework are given. Section 3 is used to present a number of experiments on cost based mapping using MAP. Amongst others, the benefits and losses due to redundant decomposition on the mapping quality are investigated. In Section 4, a summary is given, followed by a discussion of the experimental results. Finally, some directions for future work are suggested.

1. Load balancing by redundant decomposition and mapping

The problem of finding an efficient mapping of a set of parallel tasks is generally referred to as the *load balancing problem*. In [7] it is posed that a practical approach to the load balancing problem is to solve the problem in two distinct phases: domain decomposition followed by mapping. Applications that are expected to benefit from large scale parallel computing generally work on data domains of considerable size. In general the intrinsic parallelism (denoted by the problem size N) is much higher than the available number of processors (P) in a parallel

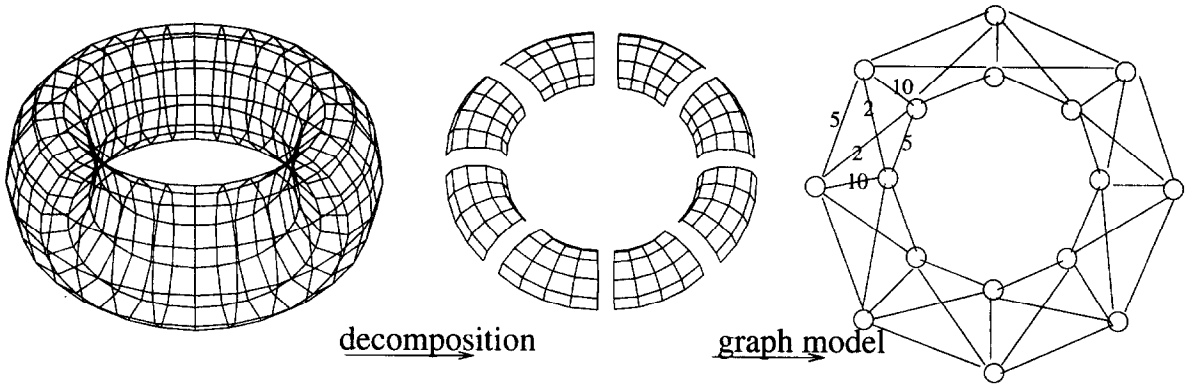


Fig. 2. Decomposition into 16 partitions of a 320 element torus, and its translation into a graph model.

system. Finding the optimal mapping is an NP-hard problem, which requires, in the worst case, $\bar{O}(P^N)$ different mappings to be evaluated. The size of the solution space is so large that it inhibits finding the best mapping in a reasonable time. Therefore, it is essential to reduce the size of this solution space. This can be done by clustering N parallel tasks into M clusters where M is of $\bar{O}(P)$ (decomposition). The decomposition determines the connectivity between the clusters as well as their relative computational weight. In this way the load balancing problem is reduced to the mapping of M parallel tasks instead of the N atomic parallel tasks. The M clusters can then be grouped into (a maximum of) P *super clusters*, that is, mapping them on a parallel machine consisting of P processors.

We can motivate the two phase approach as follows. Decomposition is necessary to separate the domain of an application into an acceptable number of parallel tasks. Mapping is necessary for optimization of the parallel execution time. We pose that a redundant decomposition ($M > P$) can be used to create a solution space that contains more solutions that are near the optimal mapping (which can only be found in case that the granularity of the decomposition is equal to the intrinsic parallelism (N)). However, this will induce an increased size of the solution space and consequently the problem becomes more difficult to optimize. Therefore, the trade-off between solution quality and optimization time has to be kept in mind.

1.1. Modelling the cost of mappings

In order to allow for evaluation of a given decomposition and mapping, two models are needed: a parallel application model and a parallel machine model [20]. Both models must be of moderate complexity, allowing for quick evaluation of candidate mappings. However, the model still has to carry enough richness to allow for comparison with the real system. The models that are used in MAP are instances of a more generic formalism used in dynamic complex system studies [26].

1.1.1. Parallel application model

A widely used model for static parallel processes is the parallel task graph. The vertices of the graph correspond to computational tasks and are attributed with a work load while the edges model communication load between the tasks. For example, Fig. 2 shows the modelling procedure for an application that works on a domain with a torus shape consisting of $32 \times 10 = 320$ quadrilateral elements. The torus is partitioned into 16 parts (only the top half is shown) and these 16 partitions are consecutively represented in a static task graph. In this case, the work load attributes of the vertices can be set equal. The edges in the graph have relative weights of 2, 5 or 10.

1.1.2. Parallel machine model

A parallel machine can be modelled analogously. Now a vertex corresponds to a processor and the

attribute is processing power, while an edge corresponds to the physical network connection between processors and the attribute there is link speed or bandwidth. The processor graph is fully connected, since every processor can communicate with every other processor, although they are not necessarily linked to each other directly in the physical topology.

1.1.3. Cost of a mapping

Several cost models that use a graph representation of the parallel application and machine can be found in literature. The following cost function (1)[14], is known to model the actual execution time for a given task allocation with reasonable accuracy. Of course, it is a simplification of the real situation, i.e. message latencies and network congestion are neglected.

$$H = \max_{q \in \mathcal{Q}} \left(\sum_{u_i \in U^q} \beta W_{u_i}/S_q + \max_{u_i \in U^q, u_j \in A(u_i)} W_{u_i u_j}/S_{pq} \right) \quad (1)$$

where u_i is a task in the parallel task graph;, $\mathcal{Q}$ the set of processors; $\mathcal{A}(u_i)$ the set of tasks connected to task u_i; $\mathcal{U}^q$ the set of tasks u_i residing on processor q;, W_{u_i} the work associated with task u_i (expressed in flop); S_q the processor speed of processor q (expressed in flop/s); $W_{u_i u_j}$ the number of bytes to be sent from processor p to processor q, due to the connectivity between task u_i, residing on processor q, and task u_j, residing on processor p; S_{pq} the bandwidth of the route between processor p and q (expressed in bytes/s) and β is a control parameter, equivalent to the calculation–communication ratio $(\tau_{\text{calc}}/\tau_{\text{comm}})$ [10].

Eq. (1) estimates the turn-around time by the execution time of the "slowest" processor in the parallel machine. The discontinuous appearance of this cost function is known to hamper the convergence behaviour of optimization algorithms. An alternative cost function (2), first introduced by Fox [10], has better optimization properties.

$$H = \beta \sum_p W_p^2 + \sum_{p>q} C_{pq}, \quad (2)$$

where, $W_p := A_p/S_p$, with $A_p := \Sigma_{u_i \in \mathcal{U}^p} W_{u_i}$, total work on processor p in terms of flop and, $C_{pq} := M_{pq}/S_{pq}$, with $M_{pq} := \Sigma_{u_i \in p, u_j \in q} W_{u_i u_j}$.

An incremental change to a given task allocation (moving one task to another processor), necessitates a complete recalculation of the cost for Eq. (1). On the other hand, Eq. (2) has the locality property, which means that incremental changes in a task allocation can be propagated into the cost without having to re-calculate the whole cost function. Only the difference has to be calculated instead [17]. This is specifically useful if an optimization algorithm that is based on incremental changes is applied, and as such can exploit the direct calculation of these increments, thus decreasing the computational cost of the optimization process. A disadvantage of using (2) is the fact that it is not a correct model for the absolute cost. However, the suboptimal mappings that are found with both functions approximately coincide, which is satisfactory in most practical situations. Both cost models are available within MAP.

2. Optimization methods for mapping

Since the decomposition and mapping phases are processed separately, dedicated methods for each can be developed. Decomposition of mesh based application domains can be performed with considerable efficiency using deterministic graph partitioning methods. In this study the library of graph partitioning methods that is offered by the DDT tool [9] of the CAMAS workbench, is used to create the domain decompositions.

The problem of evaluating every possible mapping for a problem with granularity N on a parallel platform of P processors is an intractable task for realistic problems. Even if the granularity of the application is first reduced by means of a redundant decomposition, the amount of possibilities still grows unacceptably. It has been shown that heuristic methods like genetic algorithms (GA) and simulated annealing (SA) are good approaches for finding, suboptimal mapping [18]. Motivated by arguments like parallelizability, generic applicability, cleanliness and extendability we have chosen to base the optimization kernel of MAP on a framework which can incorporate both types of algorithm

as well as a deterministic *greedy* search method known as steepest descent (SD).

2.1. The MAP kernel: A generic optimization framework

Many problems in science and engineering can be considered as optimization problems. One approach to solve these problems is to use deterministic or numerical methods. Another attractive approach is to use stochastic or natural solvers (NS) [26]. Two well-known natural solvers are SA [15] and GA [13]. A fundamental problem is that both methods are difficult to parallelize to a level of high scalability.

Classical GAs use global knowledge for their selection process. There does not exist a spatial relation between the different chromosomes. An essential problem in SA is that the method is inherently sequential. Our approach to parallelize both methods is to introduce adjustable ranges of locality by using an explicit mapping onto Cellular Automata. Examples in which a GA is mapped onto Cellular Automata can be found, for example, in [12,16,28].

In the general case it is not possible to map SA onto Cellular Automata. However, locality can be imposed on SA by applying a population based algorithm [11]. Another approach is to use simultaneous independent searches, which is basically the same method without interactions [2].

In [1] a generic algorithm, the so-called abstract genetic algorithm (AGA), for both SA and GA was introduced. In the MAP kernel an abstract cellular framework is utilized, that can be parallelized efficiently. Three different optimization methods can be invoked within MAP that fit directly onto this framework.

2.1.1. Solution encoding

A mapping is coded as a sequence, where each letter in this sequence is a number from the alphabet $\{1,2,\ldots,P\}$. The index of this sequence corresponds to the vertex number of the task in the task graph, while the letter in the sequence corresponds to the processor allocation number of the given task. Each of the three optimization methods that can be invoked within MAP manipulates this solution encoding in its own characteristic manner in the optimization process.

2.1.2. An abstract cellular genetic algorithm (ACGA)

To avoid the use of global information which is necessary in the AGA of Aarts et al. [1], we have introduced a local spatial neighbourhood structure [23]. In this way we make an analogy between the chromosome (or solution vector) and a cell in Cellular Automata. Each chromosome is assigned to a cell, which explicitly defines its neighbourhood structure. Interaction between cells (and consequently mixing of solutions in GA, for example) is restricted to a local neighbourhood. We can formulate the pseudo-code for the ACGA as follows:

```
Initialize
DO
  FOR EACH cell in the population
    Choose a parent list (choice)
    Recombine parent list (production)
    Mutate the offspring
    Evaluate offspring
    IF offspring meets some criterion
(selection)
      accept offspring
    ELSE
      leave the current chromosome in
its place
    ENDIF
  ENDFOR
UNTIL maximum number of generations
(iterations)
```

2.1.3. A cellular genetic algorithm (CGA)

From the ACGA pseudo-code above a parallel CGA with local selection can be derived straightforwardly. We only have to select the various genetic operators.

First, the selection operator. A conventional GA uses a global method to select the parents. One example is roulette wheel selection. With a CGA the parents are selected from a neighborhood of size $(2r+1)^2$, where r is the interaction radius. The fitness $F(x_i)$ of a specific mapping x_i is given by $F(x_i) = H_{\max} - H(x_i)$ where $H_{\max}$ corresponds to the cost of the most expensive individual in the genetic population and H_{x_i} denotes the cost associated with

individual x_i. A cell is chosen as a parent by picking out a uniformly distributed random number $\xi \in [0, 1)$ which satisfies the following rule:

$$\xi < \frac{F(x_m)}{\Sigma_{x_j \in A_{k(r)}} F(x_j)}, \tag{3}$$

where $A_{k(r)}$ is the neighborhood with radius r of cell x_k, including x_k, and $x_m \in A_{k(r)}$. In a previous work [24] we have coined the term *local roulette wheel* (LRW) selection for this mechanism.

Another possibility is tournament selection, which we coin *local tournament selection* (LTS) in the case of CGA. There is an advantage in using LTS over LRW in small neighbourhoods, because LRW suffers from sampling errors when used on small populations. In [24] we have shown that the use of structured populations and local selection does not induce major deviations from panmictic selection pressure. As a recombination operator we take the popular 1-point GA cross-over operator. In Fig. 3 the cross-over of two mappings is depicted. The mutation operator induces random changes to a solution vector. In our case it operates as follows. Each letter in the solution encoding is randomly changed with a small (mutation) probability to a new value, which is chosen uniformly from the set $\{1, \ldots, P\}$.

A GA has several other parameters, that can be used to steer its behaviour. The most important ones that can be distinguished are: The convergence length L and the population size N. If the optimal solution in the population has not changed during the last L evolution steps, the system is assumed to have converged. The population size N is equal to the number of chromosomes or solution encodings in the genetic population. Usually, it can be assumed that a population size of the order of the problem size (in this case the number of parallel processes) is a sensible choice. The cross-over operation generally is applied with a probability of about 0.7. This has also been adopted as the default cross-over probability in the GA algorithm used in MAP. The mutation probability is taken to be equal to $1/n$, where n is the length of the encoding sequence, or equivalently, the number of parallel tasks in the task graph.

2.1.4. *Cellular simulated annealing (CSA)*

Another optimization method that can be embedded in the cellular framework is a special variant of simulated annealing: cellular simulated annealing (CSA). To introduce locality in the SA algorithm the following approach is taken. Several SA solutions or configurations exist together on a two dimensional

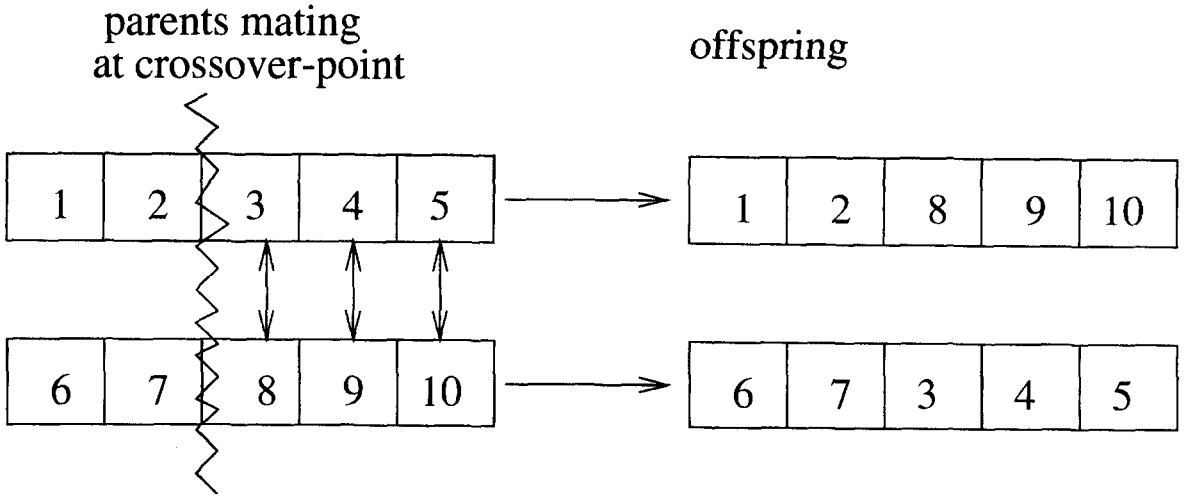


Fig. 3. 1-point cross-over of two sequences of length 5.

(2D) grid. These configurations only know of the existence of other configurations in their direct neighborhood. This neighbourhood is defined as a localized spatial structure on the grid. If a new configuration has to be evaluated for acceptance, not only the previous configuration but also its neighborhood set is taken into account. Rejection of a new configuration can cause any of the configurations in a neighbourhood set to take over the current spatial grid location. For details on the implementation of SA and CSA the reader is referred to [23].

2.1.5. *Cellular steepest descent (CSD)*

The steepest descent (SD) algorithm, is comparable to gradient based optimization methods in continuous optimization. It evaluates the cost of each mapping that can be obtained from the present solution vector by a single mutation. The mutation that gives the highest decrease in cost (steepest descent) is accepted. This procedure is repeated and continued, until no one-step mutations are left that decrease the cost. Such a mechanism is also known as a *greedy* method. Although the method is fast in comparison with SA and GA, and straightforward to implement, it is likely to get stuck in local minima, and therefore the least attractive for global optimization of complex combinatorial optimization problems. SD can be embedded into the cellular framework, leading essentially to an embarraisingly parallel implementation of the algorithm, that is, multiple non-interacting SDs are performed at the same time.

3. Experiments

In this section, several mapping experiments are presented. Firstly, in Section 3.1, mapping on hypothetical machines, with either infinitely fast processors, or infinitely fast communication networks, is considered.

The ratio of calculation time and communication time of parallel applications (expressed by β in Eqs. (1) and (2)) is an important factor in the optimization process. An experiment on the sensitivity of the mapping process to this parameter is presented in Section 3.2. In Section 3.3, results are given on experimentation with the concept of redundant decomposition and mapping. This is done on a finite element problem, that is to be processed in parallel.

3.1. *Mapping on hypothetical machines*

Initially, experiments are carried out on the simplest possible cases. For this purpose, two different hypothetical parallel machines are distinguished. One is equipped with infinitely fast processors, which implies that the work load term in the cost functions is deleted. The optimal mapping is sequential: all processes are allocated to one processor. The other hypothetical machine is located on the other side of the machine spectrum. In this case communication between processors can be performed infinitely fast. Therefore, the optimal mapping requires the work (in terms of computation time) to be equally balanced over the processor topology; each processor gets an equal part of the calculation time (not necessarily equal work distribution, since in principle it is allowed to have an asymmetric processor topology).

As a test application model we take the car grid depicted in Fig. 4. We consider decomposition in 16, 32 and 64 parts, using recursive spectral bisection (RSB) [25], followed by mapping of the corresponding task graphs on both hypothetical topologies. Cost function (2) is used as the objective function by the CGA optimization process.

In Fig. 5 the evolution of the work load distribution for the fast-cpu topology is depicted for the best individual in the GA population, for mapping 32 partitions on a 16 processor topology. For each processor the evolution of the work load allocation is displayed as a line. it can be observed that the total work load is assigned to one processor after 75 generations.

Fig. 6 shows the evolution of the work load distribution, but now for mapping on the fast-network topology. In this case the work load is distributed evenly over the available processors.

In Fig. 7 the evolution of the cost associated with the best individual in the population is shown for the following problem instances; the car grid partitioned into 16, 32 and 64 parts, respectively, map-

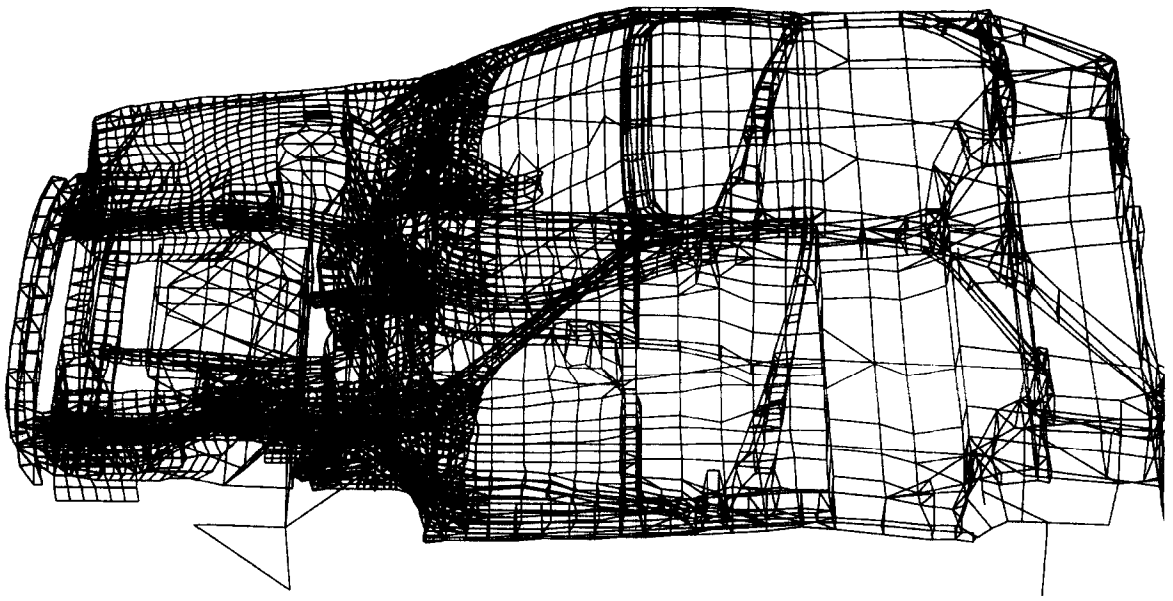


Fig. 4. Car grid, 7938 elements, courtesy provided by ESI-Paris.

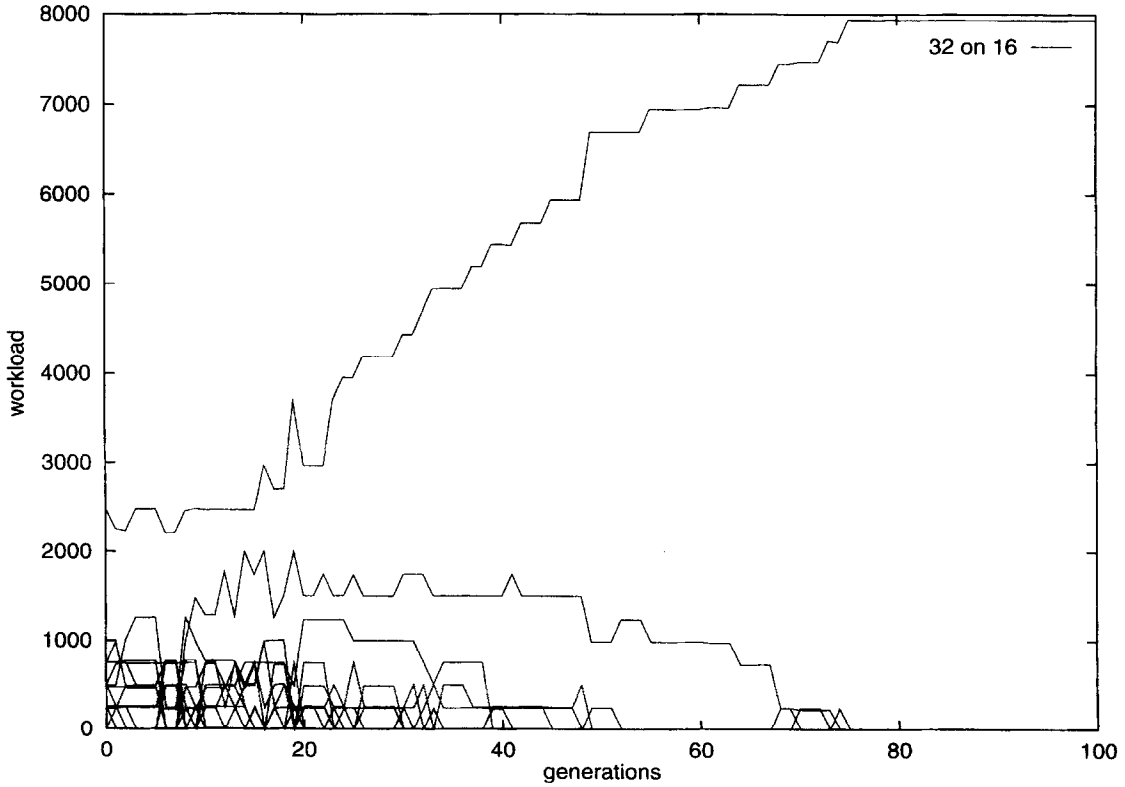


Fig. 5. Evolution of the work load distribution of the best individual in the population for mapping 32 partitions on a 16 processor, fast-cpu, topology.

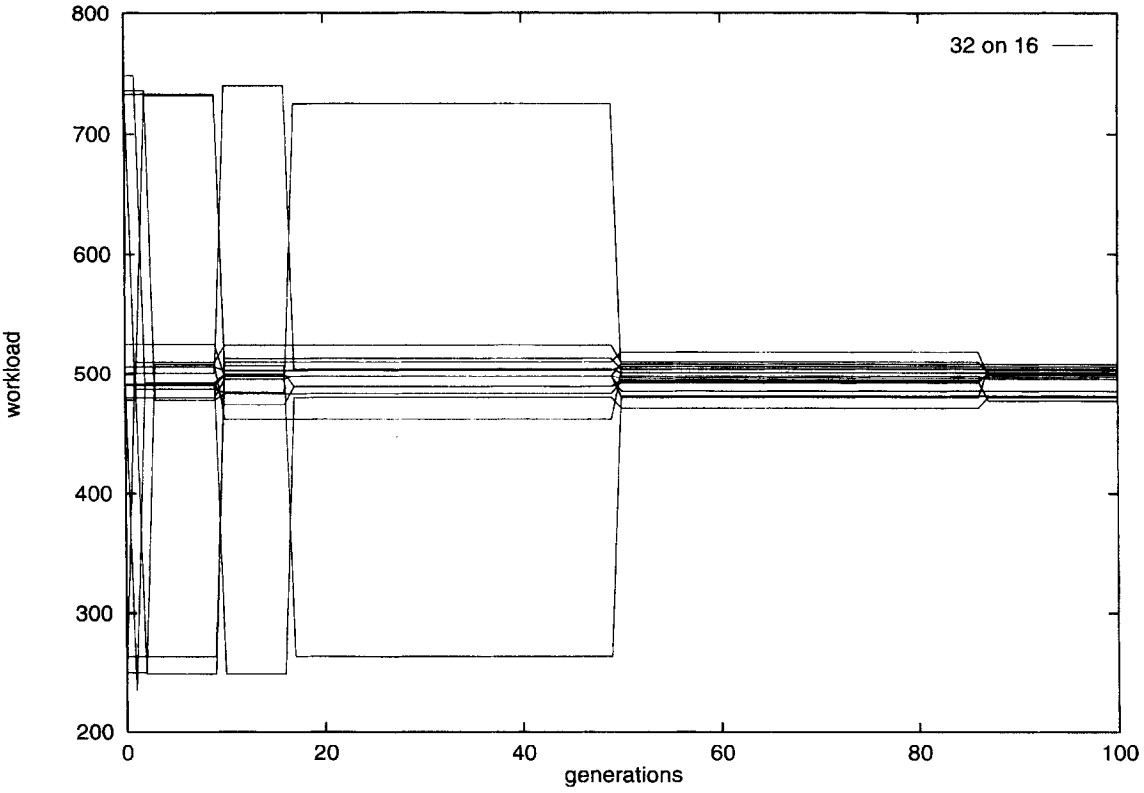


Fig. 6. Evolution of the work load distribution of the best individual in the population for mapping 32 partitions on a 16 processor, fast-network, topology.

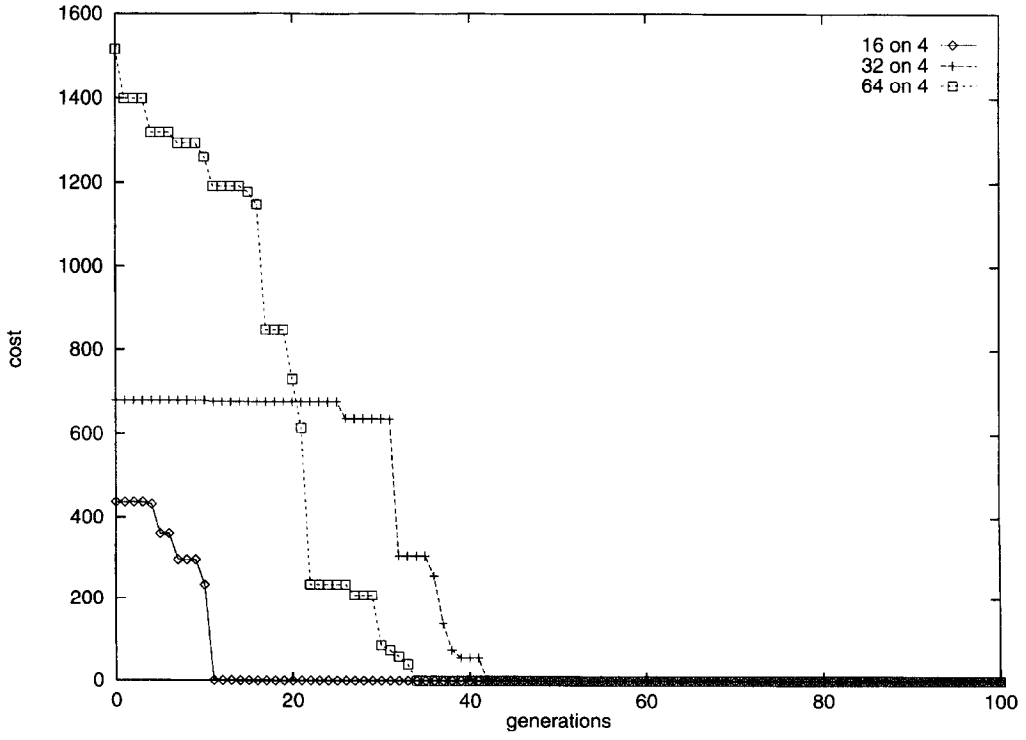


Fig. 7. Evolution of the cost of the best individual in the population for mapping 16, 32 and 64 partitions on a 4 processor, fast-cpu, topology.

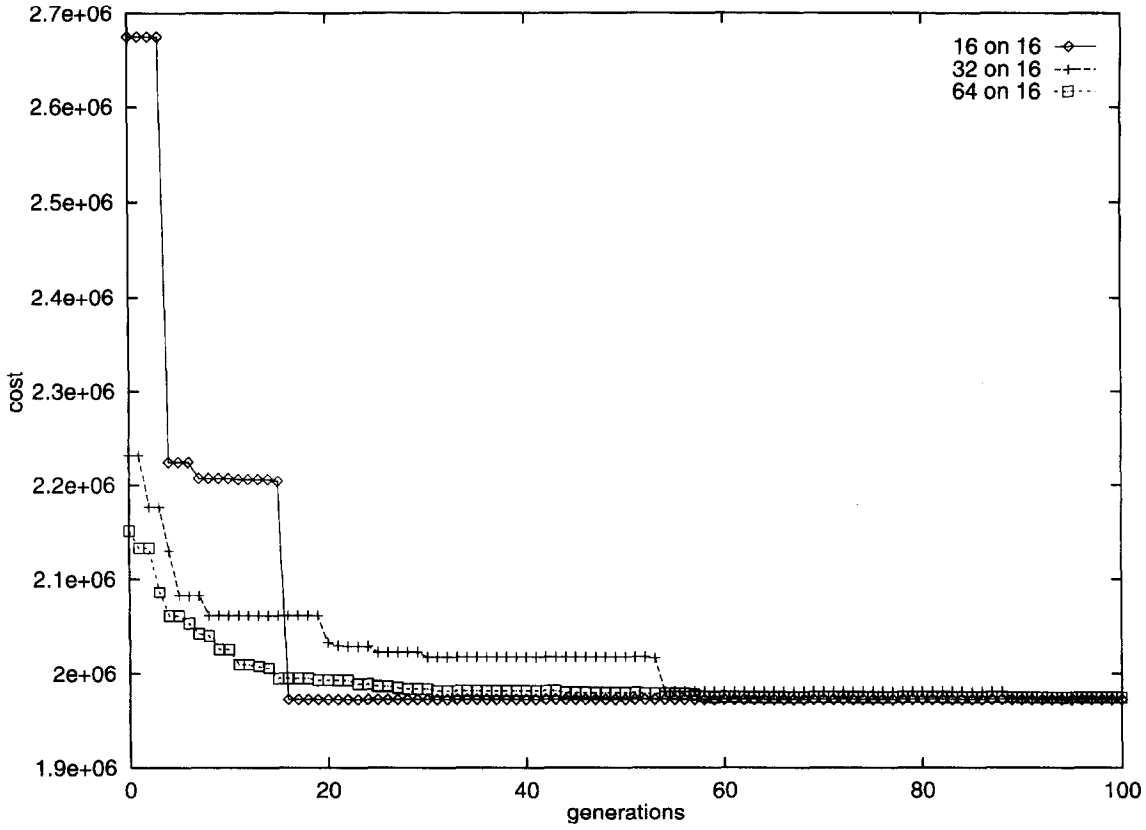


Fig. 8. Evolution of the cost of the best individual in the population for mapping 16, 32 and 64 partitions on a 16 processor, fast-network, topology.

ped on a 4 processor, fast-cpu, topology. Fig. 8 shows the cost evolution for mapping the same decompositions on a 16 processor, fast-network, topology. Both figures have been created using CGA. It should be noted that the experimental data depicted in all figures correspond to single optimization runs. Therefore, one should not draw general conclusions on the convergence speed of the genetic algorithm on the basis of these figures.

β, which has been introduced in the cost functions, can be used to vary the characteristics of a parallel machine in the machine spectrum between the two hypothetical machines. In the following, we will study the implications of β variations to the load balancing problem in more detail.

3.2. β Sensitivity

The competition between the work and calculation term in the cost functions can be varied using β. It gives a natural parameter to study the mapping problem for a given application and machine topology over the whole spectrum of possible machines. In Fig. 9 the number of processors that is used in optimal mapping vs. the value of β in the cost functions is shown. Eq (1) was taken as the objective function, mapping the 320 torus grid, partitioned into four parts on a 4 processor ring topology. The small solution space in this case, allows for an exhaustive search for the optimal values. The allocations that were found to be optimal in this case, therefore could be proven to be the global optima.

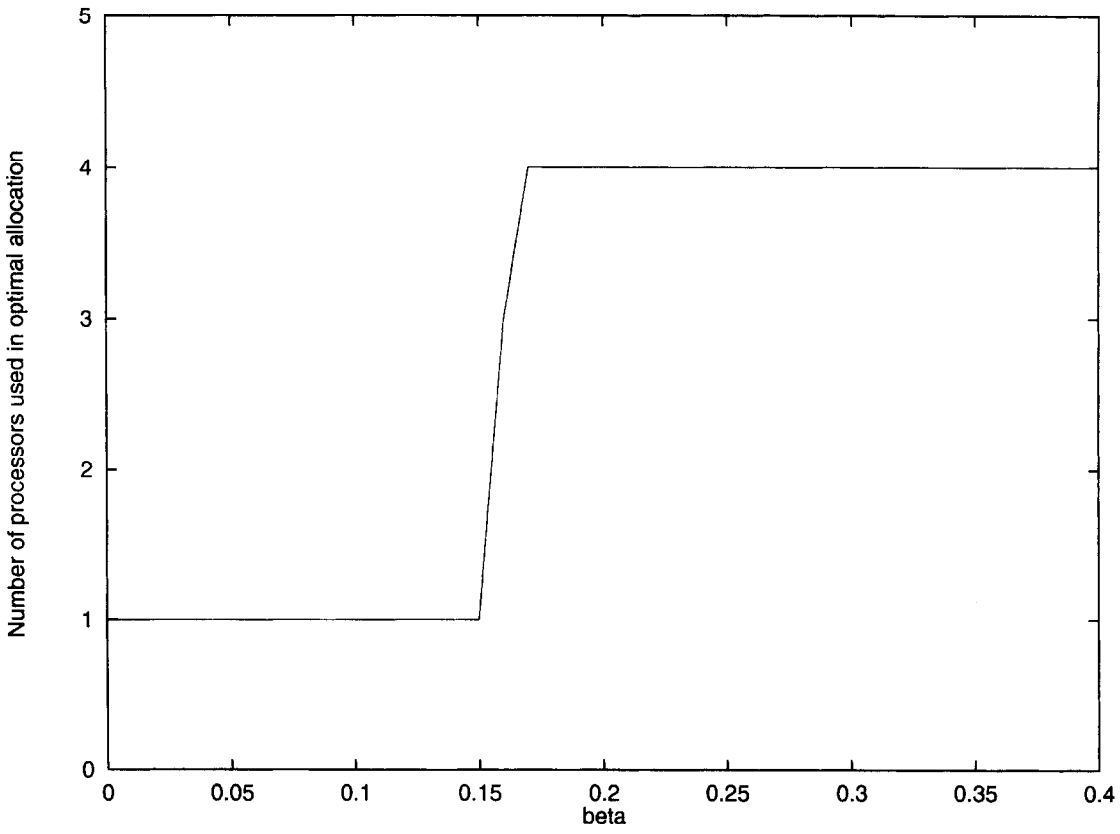


Fig. 9. Number of processors in optimal allocation vs. β.

Before we continue with the experiments on redundancy, let us reflect on the consequences of this section. We observe that there is a transition in the optimal number of processors that is used, that is dependent on β. In a recent work [5] we have argued that the load balancing problem essentially only exists in the transition region. Denoting the value of β where the transition is halfway by β_c, we can state that in practice it can be expected that mapping of a number of interconnected tasks is trivial in the region where $\beta < \beta_c$. In this case it makes no sense to search for parallelism. Sequential task mapping is optimal.

On the other hand if $\beta > \beta_c$, the amount of communication between the parallel tasks is small in comparison with the amount of work per task. Therefore, parallelism can be exploited. Finding the optimal allocation in this situation is not trivial though. If we have M tasks that are not all of equal weight, we are confronted with the NP-complete *job shop* problem [3]. Furthermore, if the processor topology is asymmetric, which for example can be the case in a heterogeneous cluster of workstations, extra difficulty is introduced.

Within the transition region ($\beta \approx \beta_c$), the work and communication term in the cost function are in strong competition with one another and the optimal value will be even harder to find, than in case of $\beta > \beta_c$. In Section 1 it has been posed that a redundant decomposition creates additional degrees of freedom for the mapping process, enabling it to find better optimal solutions. In Section 3.3, we will study the benefits of redundant decomposition, where β is restricted to be larger than β_c.

Table 1
The (suboptimal) cost for varying decomposition cardinality (M) and number of processors (P) with $\beta \approx 10$

M/P	4	5	6	7	8
8	1619.6	1596.8	1586.2	1564.0	856.8
16	1623.8	1526.2	1198.4	1158.6	822.0
32	1625.2	1340.4	1121.8	975.0	821.0
64	1642.4	1314.4	1095.2	932.0	824.4

Table 2
The (suboptimal) cost for varying decomposition cardinality (M) and number of processors (P) using a fast network topology

M/P	4	5	6	7	8
8	2006.0	1976.0	1968.0	1934.0	1053.0
16	1996.0	1883.0	1495.0	1433.0	1016.0
32	1992.0	1663.0	1417.0	1205.0	1014.0
64	1992.0	1594.0	1331.0	1147.0	1006.0

3.3. *Load balancing by redundant decomposition and mapping*

In this section we present some experiments on redundant decomposition. Again we consider the car grid. Tables 1 and 2 summarize the quality of mapping, calculated using Eq. (1), for various values of M and P. We consider a target machine that is fully connected and homogeneous.

The decomposition method that is used to create the M partitions is taken to be recursive coordinate bisection [25]. For Table 1 $\beta = \tau_{calc}/\tau_{comm} \approx 10$, whereas a fast network topology is used for the results presented in Table 2. Each number in these tables is the minimal value over 200 SD runs and is calculated using cost function (1), while the SD process is steered by Eq. (2).

From Table 1 we can observe that redundancy generally allows us to find better optima, although for the situation where $P = 4$ this is not the case. In the discussion below, some practical problems associated with mapping real finite element partitions are discussed, that account for this deficiency. For the fast network topology, redundancy is beneficial in all cases, see Table 2.

Finally, we consider mapping on an asymmetric fully connected 8-node topology, where the relative processing power of each CPU is inversely proportional to the processor identifier, $S_p = 1/p\,(p \in \{1,2,3,4,5,6,7,8\})$ and $\beta \approx 10$. In Fig. 10 the cost evolution for mapping 8, 16, 32 and 64 partitions of the car grid is depicted. We have used CGA for the optimization process. Note that increasing redundancy initially results in better suboptima, but that the solution quality degrades for large redundancy, which is presumably due to the size of the solution space.

4. Discussion and concluding remarks

4.1. *Summary and discussion*

For the purpose of mapping parallel processes onto parallel machines, we have developed a tool (MAP), which utilizes a generic graph model for parallel applications and machines. The quality of a proposed mapping is expressed using a cost function of which several examples are available within MAP. The mapping problem is approached as a NP-hard combinatorial optimization task. From within MAP several (well-known) parallel heuristic optimization kernels can be invoked to "solve" the mapping problem. The idea to use a redundant decomposition to create search freedom for the mapping optimization process is posed as a possible method to enhance the performance of parallel processes, opposed to a partitioning cardinality equal to the number of available processors.

4.1.1. *Mapping on hypothetical machines*

The GA converges to (sub) optimal mappings under the default parameter settings, when (2) is used as the objective function. This follows from experiments such as mapping on processor networks with respectively infinitely fast processors (all processes are mapped on one processor) and on topologies with an infinitely fast network (work load balance). A selection of the corresponding experimental results is shown in Figs. 5–7. It is clear that the optimization task becomes more problematic as the number of solutions increases. Furthermore, the optimization in the case of fast-cpu topologies appears to be much harder than in case of the fast-network topologies. It takes in general

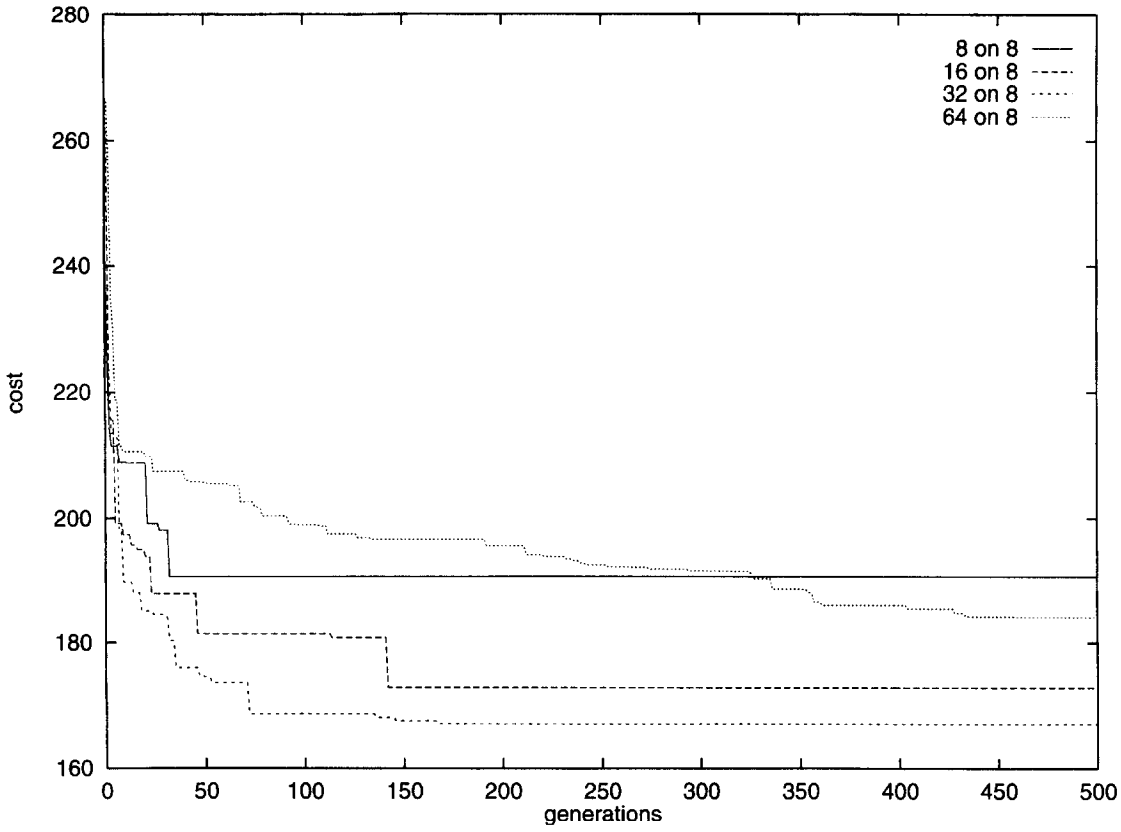


Fig. 10. Evolution of the optimal mapping in the CGA population for mapping 8, 16, 32 and 64 partitions on an asymmetric 8 node topology with $\beta \approx 10$.

much more evolution steps or a larger population size to arrive at the optimal value.

This can be explained from the fact that the number of optima is much smaller in the fast-cpu case (P) than in the case of the fast-network ($P!$), although the size of the solution space is equal in both cases. Therefore, the chance than an optimal solution is found (by random search) for the former is much smaller than for the latter. On the other hand the optimal mapping of parallel processors with an infinitely fast network is only exactly solvable in the case that each process has equal weight. As soon as this property is violated it is known that the problem becomes NP-complete, analogous to job scheduling [3]. However, good suboptimal solutions (minimizing the variance of the work load distribution) are found.

Comparable experiments indicate that SA has less difficulty with finding the optima in the fast-cpu case. The nature of the SA algorithm, where incremental changes to the solution encoding are made, is the main reason for this. Both SA and GA have more trouble optimizing (1), which is caused by the discontinuous form of this function. In this case *brute force* random search or multiple SD runs are possible alternatives.

4.1.2. β Sensitivity

The competition between communication and calculation time is an important parameter in the mapping process. Fig. 9 clearly shows that variation of β induces a transition from optimal sequential to parallel allocation. The steepness of this transition is rather striking. It was found that the optimal map-

ping appears either to be purely sequential or maximally parallel. Effectively, outside the transition region the problem of mapping respectively is reduced to sequential allocation of the parallel tasks and job scheduling.

4.1.3. *Load balancing by redundant decomposition and mapping*

From Tables 1 and 2 we can see that the quality of mapping can significantly benefit from redundant decomposition if we consider mappings where M/P is not an integer. This can be expected, since the relative weight of each chunk of data will be smaller in case of more redundancy which makes it easier to balance the work load distribution.

If M/P is an integer it is clear that the applicability of redundant decomposition strongly depends on the quality of the decomposition method. In case of a homogeneous processor topology, and a locality preserving decomposition method like RSB, one can only expect a marginal cost gain by applying mapping. On the other hand, reasonable cost improvements can be expected for mapping on heterogeneous topologies as is shown in Fig. 10. Note that there is a trade-off between cost decrease due to increased redundancy and the size of the search space, which increases with the degree of redundancy. For a redundancy of 64 one can expect a better optimal solution cost. However, due to the increased complexity, CGA is not able to find it in reasonable time.

4.2. *Concluding remarks and future work*

A significant practical problem in mapping real parallel processes using redundant decomposition is illustrated by the following example. Given a practical finite element mesh that is decomposed in M, and $2M$ parts, where the second partitioning equals the first partitioning, with each subdomain bisected. The first partitioning can therefore directly be constructed out of the second one by assembling the appropriate parts together. This collection operation is computationally so expensive that it must be left out of a mapping algorithm. The cost of a mapping is estimated by the individual contributions of each of the $2M$ partitions to the cost function. The contribution to the communication cost of a processor pair is approximated by the summation of the communication volume between each pair of data partitions on these processors. As a consequence, we are confronted with double counting of shared mesh points, resulting in the situation that the cost associated with the mapping of the M parts will be lower than that of the identical mapping of the $2M$ parts. Therefore it is not possible to quantitatively compare the evolution of the mapping costs for varying redundancy.

We have observed a trade-off between the useful redundancy and the size of the solution space. Although increased redundancy allows us to find better solutions it is not said that the performance gained by it will compensate the effort that we have to put into finding a better solution. In Section 3.2 it was noted that the specific parallel system and parallel application parameters are very important for the shape of the search space of the mapping problem.

The methodology of redundant decomposition and mapping described above is applicable to the load balancing problem for applications that display a static work load distribution combined with parallel machines that have static processor characteristics. An additional strong point, of our load balancing approach, is that it allows for more flexibility in handling asymmetry present in the processor topology.

However, many applications cannot be described in terms of a static parallel task graph. Furthermore, parallel machines like workstation clusters are by no means static resources. Extensive research on the subject of dynamic load balancing in our group has resulted in a prototype dynamic load balancing system for cluster computing, called *Dynamic PVM* [22].

It can be expected that the inherent dynamic asymmetry in workstation clusters will have a significant degrading effect on the performance of applications that work on nonredundantly decomposed meshes. Redundant decomposition, and thus in practice a redundant set of parallel tasks, can be used by a system like *Dymanic PVM* to adapt to changes in the CPU utilization. Part of the future work in our group consists of embedding a (parallel) mapping algorithm, into the task scheduler of *Dynamic PVM* [21].

Acknowledgements

The DDT tool of the CAMAS workbench was used for creating the domain decompositions. It has been developed by Nick Floros, of the University of Southampton. We are grateful that we were allowed to use this tool for our purposes.

References

[1] E.H.L. Aarts, A.E. Eiben and K.H. van Hee, Global convergence of genetic algorithms: A markov chain analysis, in ed. H.P. Schwefel (1990) 4–12 *Parallel Problem solving from Nature I,*.

[2] R. Azencott, *Simulated Annealing: Parallelization Techniques* (Wiley, New York, 1992).

[3] D.P. Bovet and P. Crescenzi, *Introduction to the theory of complexity*, International Series in Computer Science (Prentice-Hall, Englewood cliffs, NJ, 1994).

[4] A. De Mes, P.M.A. Sloot and J.F. de Ronde, Camas-tr-2.1.1.4 parasol i progress report, Technical Report, University of Amsterdam, March 1994.

[5] J.F. de Ronde, A. Schoneveld and P.M.A. Sloot, Properties of the task allocation problem, Technical Report CS-96-03, University of Amsterdam, 1996.

[6] J.F. de Ronde and P.M.A. Sloot, Camas-tr-2.1.3.4 map final report, Technical Report, University of Amsterdam, October 1995.

[7] J.F. de Ronde, A. Schoneveld, P.M.A. Sloot, N. Floros and J. Reeve, Load balancing by redundant decomposition and mapping, in: *High Performance Computing and Networking, Lecture Notes in Computer Science*, eds. H. Liddell, A. Colbrook, B. Hertzberger and P. Sloot, Vol. 1067 (1996) 555–561.

[8] J.F. de Ronde, B. van Halderen, A. de Mes, M. Beemster and P.M.A Sloot, Automatic performance estimation of spmd programs on mpp in: *Massively Parallel Processing Applications and Development*, eds. L. Dekker, W. Smit and J.C. Zuidervaart, *EUROSIM*, (June 1994) 381–388.

[9] N. Floros, Camas-tr-2.2.2.8 user manual, Technical Report, University of Southampton, April 1995.

[10] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon and D. Walker, *Solving Problems on Concurrent Processors*, Vol. 1 (Prentice-Hall, Englewood cliffs, NJ, 1988).

[11] D.E. Goldberg, A note on boltzmann tournament selection for genetic algorithms and population oriented simulated annealing, *Complex Systems* 4 (1990) 445–460.

[12] M. Gorges-Schleuter, An asynchronous parallel genetic optimization strategy, in: *3rd Int. Conf. on Genetic Algorithms* (1989) 422–427.

[13] J.H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan, Ann Arbor, 1975.

[14] J. De Keyser and D. Roose, Load balancing data parallel programs on distributed memory computers, *Parallel Computing* 19 (1993) 1199–1219.

[15] P. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, Optimization by simulated annealing, Research Note RC 9355, IBM, 1982.

[16] B. Manderick and P. Spiessens, Fine grained parallel genetic algorithms, in: *3rd Int. Conf. on Genetic Algorithms*, (1989) 428–433.

[17] N. Mansour and G. Fox, A hybrid genetic algorithm for task allocation, in: *Proc. 4th Int. Conf. on Genetic Algorithms*, (1991) 466–473.

[18] N. Mansour and G. Fox, Allocating data to multicomputer nodes by physical optimization algorithms for loosely synchronous computations, *Concurrency: practice and experience*, 4 (7) (1992) 557–574.

[19] J. Merlin, Camas-tr-2.2.1.2 ida's user's guide (year 1 deliverable report), Technical Report, University of Southampton, September 1993.

[20] M.G. Norman, Models of machines and computation for mapping in multicomputers, *ACM Computing Surveys* 25 (1993) 263–302.

[21] University of Amsterdam, Dynamite: Dynamic task migration execution environment, ESPRIT project nr. 23499, 1997.

[22] Benno J. Overeinder, Peter M.A. Sloot and Robbert N. Heederik, A dynamic load balancing system for parallel cluster computing, *Future Generation Computer Systems*. 1996, Published in Special Issue on Resource Management in Distributed Systems.

[23] A. Schoneveld, An abstract cellular genetic algorithm, Master's Thesis, University of Amsterdam, June 1994.

[24] A. Schoneveld, J.F. de Ronde, P.M.A. Sloot and J.A. Kaandorp, A parallel cellular genetic algorithm used in finite element simulation, in: *Parallel Problem Solving from Nature IV, Lecture Notes in Computer Science*, eds. H-.M. Voigt, H-. P. Schwefel, I. Rechenberg, and W. Ebeling, (1996) 533–542.

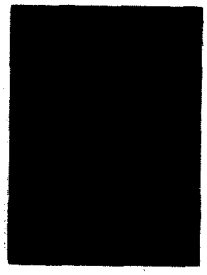
[25] H.D. Simon, Partitioning of unstructured problems for parallel processing, *Computing Systems in Engineering* 2 (2/3) (1991) 135–148.

[26] P.M.A. Sloot, J.A. Kaandorp and A. Schoneveld, Dynamic complex systems (dcs) a new approach to parallel computing in computational physics, Technical Report TR CS 95, University of Amsterdam, November 1995.

[27] P.M.A. Sloot and J. Reeve, Camas-tr-2.3.7 executive report on the camas workbench, Technical Report, University of Amsterdam and University of Southampton, October 1995.

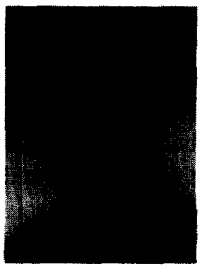
[28] M. Tomassini, The parallel genetic cellular automata: Application to global function optimization, in: *Artificial Neural nets and Genetic Algorithms* (1993) 385–391.

[29] B. van Halderen and P.M.A. Sloot, Camas-tr-2.1.1.7 sad/parasol final report, Technical Report, University of Amsterdam, October 1995.



Jan. F. de Ronde received a Masters degree (cum laude) in Physics from the University of Amsterdam in 1992. After his graduation he joined the Parallel Scientific Computing and Simulation Group at the same University as a Ph.D. student studying general aspects of mapping in High Performance Computing and Simulation. His research focusses on theoretical and practical aspects of task allocation on parallel architectures. Presently, he is working on the implementation of a parallel finite element simulation code, the design of which is inspired by the findings of his work on mapping irregular task-graphs on parallel systems. He has published several papers on the following topics: performance modelling of parallel applications, mapping of parallel tasks on parallel machines and parallelization of explicit finite element programs.

Arjen Schoneveld obtained a Masters degree in Computer Science from the University of Amsterdam in 1994. In the same year he joined the Parallel Scientific Computing and Simulation Group at the University of Amsterdam as a Ph.D. student. Currently, he is studying the concept of Dynamical Complex Systems (DCS) as a common denominator of different kinds of physical and biological systems, in which the locality, inherently present, can be exploited for allocation onto parallel platforms. Parallel machines themselves are not considered separately, rather as another instance of DCS. Features like frustration and disorder caused by mutually interacting agents, can be found throughout different DCS. Important issues include the localisation of complexity regions within "phase-diagrams" of such systems. He is the author of several papers on the topic of task allocation complexity and parallel heuristic optimization methods.



Peter M.A. Sloot gained a Masters degree in Chemical Physics and Theoretical Physics at the University of Amsterdam in 1983. He was then appointed as a research fellow at the Duth Cancer Institute. In 1988 he received a Ph.D. from the Department of Mathematics and Computer Science at the University of Amsterdam. In 1991 he became an assistant professor and in 1993 an associate professor in parallel scientific computing in the department. He has written and managed a large number of externally funded projects (funded by NWO, STW, Biophysics, Esprit eo). In 1990 he founded the interdisciplinary working group Parallel Scientific Computing and Simulation. He has published over 100 papers on various theoretical and experimental topics in the field of computational science. He has been in the organising committee of a large number of national and international conferences and workshops on scientific computing. His current interest is in the modelling and implementation of dynamic complex systems for massively parallel computers.