NAVSWC TR 91-538

TASK ALLOCATION AND SCHEDULING FOR HIGH LEVEL SYNTHESIS

BY ERIC J. OGATA AND EDWARD A. COHEN

SYSTEMS RESEARCH AND TECHNOLOGY DEPARTMENT

21 MARCH 1994

NAVAL SURFACE WARFARE CENTER
DAHLGREN DIVISION • WHITE OAK DETACHMENT
Silver Spring, Maryland 20903-5640

NAVSWC TR 91-538

TASK ALLOCATION AND SCHEDULING FOR HIGH LEVEL SYNTHESIS

BY ERIC J. OGATA AND EDWARD A. COHEN
SYSTEMS RESEARCH AND TECHNOLOGY DEPARTMENT

21 MARCH 1994

NAVAL SURFACE WARFARE CENTER
DAHLGREN DIVISION • WHITE OAK DETACHMENT
Silver Spring, Maryland 20903-5640

FOREWORD

The results documented in this report are part of the Office of Naval Technology (Code ONR-227A) Engineering of Complex Systems (ECS) Technology Block effort. This work was done at the White Oak Detachment of the Dahlgren Division of the Naval Surface Warfare Center. The ECS block was developed to integrate systems engineering capabilities for developing large-scale, real-time, computer intensive systems. The goal of the ECS block is to improve the way in which the United States Navy currently creates, maintains and upgrades systems by incorporating automated systems engineering support technologies. These technologies will facilitate the design, analysis and life-cycle management of large, complex, real-time systems. For example, the technologies developed by the ECS block must be able to handle a system with a complexity level equivalent to the United States Navy Aegis missile defense system or greater. The ECS block is divided into four projects: Systems Design Synthesis Technology (RS34P11), Systems Evaluation and Assessment Technology (RS34P12), Systems Re-engineering Technology (RS34P13), and Engineering Application Prototype (RS34P14). This work is being performed under the Systems Design Synthesis Technology project by the Resource Allocation subtask.

The goal of this work is to develop algorithms and techniques for partitioning, scheduling and allocating logical task models of large systems onto the physical resources from which the system is constructed. These techniques will allow the high level synthesis of designs that satisfy the system requirements in an optimal, or near-optimal way.

This document covers the progress of the Resource Allocation Algorithms Task during the first ten months. This task is anticipated to continue for three years and documentation will be provided each year to reflect the current status of work. Since this task was initiated, much of the work accomplished consists of research surveys, and collection of background information. As the project continues the emphasis will shift towards the implementation and evaluation of the various resource allocation techniques.

Approved by:

D. B. COLBY, Head
Systems Research and Technology Department

ABSTRACT

This report addresses the development of automated techniques for solving resource allocation problems in the high level synthesis of system designs. These techniques are developed with the objective of supporting the design of highly complex systems that are characterized by an extremely large number of physical components of many different types, with complex interconnections and interdependencies. The purpose of the systems that are developed using these techniques is to implement a set of logical functions that define the overall system behavior. These logical functions can be described as a set of communicating tasks that pass data and control signals from one to another. The set of logical tasks must be mapped onto the physical resources from which the system is constructed. There may be many different ways to map logical system tasks onto the hardware resources. A particular mapping can be scored according to how well it satisfies some overall system design goal such as fault tolerance or rapid response time. The question as to how to identify optimal mappings that maximize or minimize some design parameter is important. This report presents the results of investigations of the performance of four different techniques for identifying optimal or near-optimal allocations given a particular optimization goal.

CONTENTS

CONTENTS

ILLUSTRATIONS

TABLES

CHAPTER 1

INTRODUCTION

The purpose of the Resource Allocation subtask of the Systems Design Synthesis project is to develop techniques for partitioning, allocating and scheduling logical task models of large, mission-critical systems onto the physical resources that compose the system. A system of this size and complexity is constructed from many disparate types of resources and can be examined at many different levels of detail. At the higher levels, components in the design may include ships, aircraft, satellites and communications links, onboard computer systems, as well as the human operators of the system. At lower levels the system components include microprocessors, electro-mechanical control systems, relays, motors, etc. Given a description of the system hardware structure, and a description of the system logical structure, there may be many ways to allocate tasks from the logical model onto the hardware resources. Some allocations are better than others. The question of how to determine whether one allocation is optimal, in some formal sense, becomes an important one.

This report is divided into four chapters: Chapter 1 provides some background material; Chapter 2 presents some mathematical terminology and the derivations of cost functions for a run-time performance optimization problem; Chapter 3 discusses the resource algorithm development and the results of some preliminary experiments using the cost functions derived in Chapter 2; Chapter 4 provides a summary and conclusion.

1.1 SYSTEM DESIGN

A system design can be represented in many different ways, and at many different levels of abstraction. A design description will include a model of the hardware components and a model of the logical task structure. The hardware model describes all the physical components of the design, and their interconnections. The logical description defines the tasks required to perform the system's mission, what information is needed to perform these tasks, and how the tasks pass information or control signals from one to another.

The resource allocation problems considered in this document begin with a hierarchical description of the physical resources that implement the system. This is called the *Implementation Model* (IM). A hierarchical description of the system's logical design is taken by the resource allocation algorithms to be mapped onto the IM. This logical description is called the *Logical Model* (LM). The primary focus of the work documented here is to address the problem of finding optimal mappings of the logical tasks described by the LM onto the physical resources described by the IM.

1.1.1 Implementation Model

The implementation model is a hierarchical description of the system's physical (hardware) resources. The hierarchical structure of the implementation model allows the model to be expanded, contracted, decomposed or recombined without limitation as the model evolves. This structure allows the model to be examined at many different levels of detail. The IM encapsulates the information about the *physical* properties, and characteristics of the system model. This includes weapons systems, communication links, sensors, processors and data-links – with related information such as their data rates, capacities and connectivity.

Figure 1-1 shows a graphic representation of an implementation model. The vertices in the IM graph represent processors in a distributed computer system. The vertices are labelled with their processor types. The edges represent data interconnections. The characteristics of each data link are expressed by a pair of numbers. The first number gives the data volume that can be carried by the link in bits per second. The second number indicates the setup time incurred to start a data transfer.

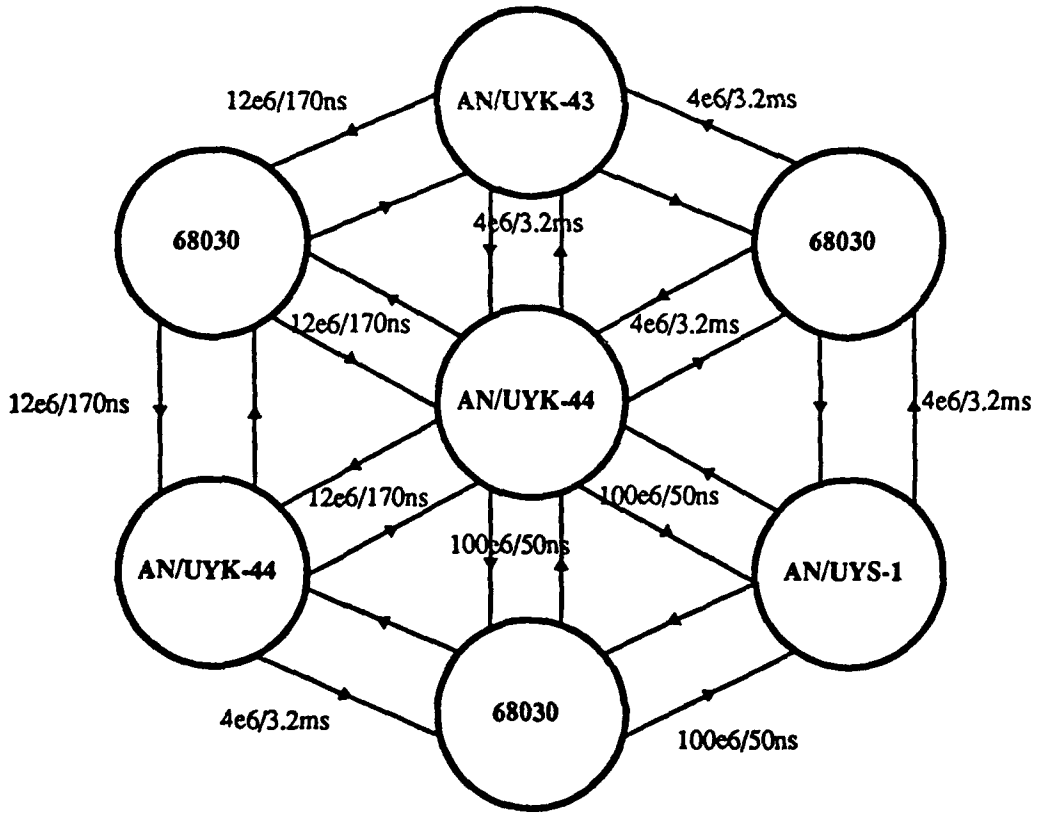


FIGURE 1-1. IMPLEMENTATION MODEL GRAPH

1.1.2 Logical Model

The logical model is a hierarchical description that captures the logical properties, attributes and characteristics of the system design. The LM is constructed in a hierarchical manner in much the same way that the IM is constructed. This model encapsulates the information about the data structures, and data flow, and describes the structure from a functional, or procedural point of view. The LM may be expanded, contracted, and restructured to display the many levels of detail encoded in the model

Figure 1-2 shows a graphic representation of a logical system model. Vertices in the LM graph represent system tasks. Edges in the LM graph represent intertask information flow. Edges are weighted with a description of the type and volume of data that flows across the edge. Information communicated along an edge may be *control* information, *data*, or both. Edges entering a node may be combined with the logical operators *and* and *or*. Logical *and* means that all of the inputs under the *and* operation must be present for the task to execute. Logical *or* implies that one or more of the inputs under the *or* operation must be present for the task to begin execution. Edges leaving a task node may also be combined using *and,* and *or* operators. When output edges are *and*ed together, both data paths are followed at task completion. When output edges are *or*ed together, the task may send data out one or more edges at task completion time.

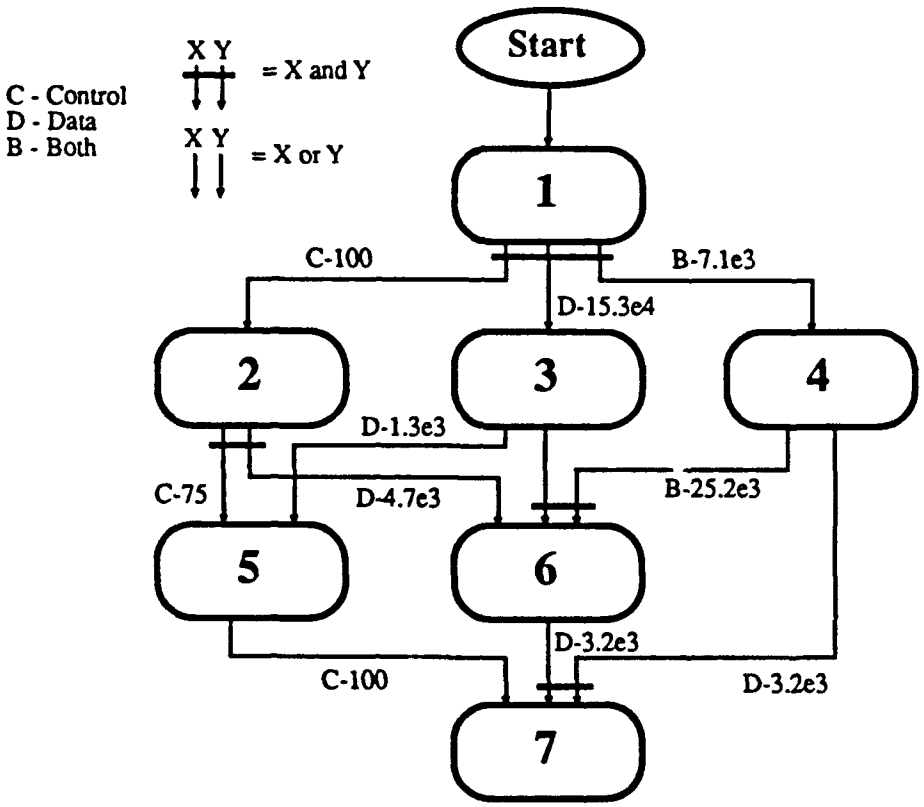


FIGURE 1-2. LOGICAL MODEL GRAPH

1.2 PARTITIONING, ALLOCATION AND SCHEDULING

The concepts of partitioning, allocation and scheduling are often discussed in the study of multi-tasking or multi-programming, especially for parallel or distributed processors. These concepts encompass many of the critical design decisions required to solve the resource allocation problems that occur in design synthesis problems. Definitions of these, and some related terms are provided in this section. Differences between static and dynamic techniques are also described.

1.2.1 Partitioning

Partitioning is the act of dividing a program into a set of sequential tasks that can be executed in parallel. A sequential task is a fundamental unit that is allocated and scheduled for execution on a particular processing node, at a particular time. A task has several important characteristics:

- A task has a sequential execution time. The value for execution time for a particular task depends on which resource from the IM the task is assigned to.

- A task has an associated overhead due to scheduling and communication costs.

- A task requires certain resources for its execution. These might be memory or I/O resources for a computer, power for an electrical system, or fuel resources for a mechanical system.

- A task may have *deadlines* associated with it. A deadline is an upper bound on the interval during which some critical action must be taken. If the task does not perform the associated action before the deadline expires, the system performance may be compromised.

- A task has precedence constraints that specify a logical combination of control signals or data that must be provided by predecessor tasks before execution can begin.

A parallel program may be partitioned into many small tasks, into a small number of large tasks, or somewhere between these two extremes. The problem of determining appropriate sizes and numbers for the partition's tasks is of fundamental importance. Partitions need to be matched to the type of hardware on which they will be executed. For example, it may be inefficient to partition a program into thousands of tasks for execution on a multicomputer with only 20 processors.

1.2.2 Allocation

Allocation is the process of assigning a task from the partitioned LM to a specific node in the IM. There are many factors that require consideration when performing the allocation. A fundamental consideration in allocating tasks to different nodes is the requirement for internode communication to exchange information from one task to another. Internode communication is likely to be slow compared to communication between tasks that reside on the same node. This introduces a significant overhead cost associated with the parallelization of a system. An efficient allocation must try to

balance the load evenly over all the available resources, but must also keep the amount of internode message traffic low. In a heterogeneous system there may be many tasks that are constrained to be performed by one particular physical resource, or by one of a limited number of the resources given. For example, a SONAR pulse emitted by the system might be constrained to be produced by one particular transceiver, and could not be generally allocated to any resource element. It is also important to consider that one type of resource may be much more efficient at executing a particular task even though that task could be more generally allocated. For example, a Fast Fourier Transform (FFT) would probably execute many times faster on a special purpose signal processing computer than it might on a general computer.

1.2.3 Scheduling

Scheduling is concerned with the precise plan for executing a set of tasks on the nodes in the IM. A parallel system will require a set of schedules, one for each resource in the system. In addition, a schedule must satisfy the following conditions to be considered valid:

- All tasks in the schedule must meet their deadlines.

- The task precedence constraints must be met.

- The total number of tasks executing at any moment in time does not exceed the number of resources (e.g. , processors) available to execute them

- The total amount of a limited resource (e.g. , memory) consumed by all executing tasks at any node, at any particular moment in time must not exceed the total quantity of that resource available.

1.2.4 Static vs. Dynamic Techniques

An important distinction must be made between *static* and *dynamic* methods for partitioning, allocation and scheduling. Static techniques work off-line during the design stage (or during the compile stage for a computer program). Dynamic techniques operate during the system execution. For example, in a typical multi-tasking computer system, scheduling is dynamic. Tasks are executed by being selected by a special program called the *scheduler* based on their priority. The scheduler determines execution order and allocates time slices to each task. In static scheduling, the execution schedule is predetermined, and there is no need for a scheduler task. This eliminates some overhead, but also reduces the flexibility of the system to respond to varying load situations. Most of the problems examined initially will involve static techniques. Dynamic techniques may be investigated as the project progresses.

1.3 PROBLEM OVERVIEW

The resource allocation problem considered here is to partition, allocate and schedule the LM modules onto the IM nodes to provide an effective system design. This problem is very similar in its nature to the partitioning and scheduling problems considered for parallel program design. Also, since the systems considered by the ECS block are often very heavily computer-based, much of the work done on partitioning and scheduling problems in parallel processing is directly relevant to this task. In this document, techniques for determining allocations of parallel *computer* programs to parallel *computer* hardware are often examined. However, the overall problem that is addressed in this work goes far beyond the computer-based systems that are often used as examples. The techniques will be developed in a general way that allow their direct application to problems outside the range of parallel/distributed computing. These techniques must be capable of expansion to handle the large, complex systems design problems considered by ECS. The target systems will include computer resources, as well as many other types of resources in an overall integrated system design. Another significant difference between the resource allocation problems considered here and the ones considered in the general literature on parallel/distributed processing is that many different criteria, or measures of effectiveness (MOE), for the allocations are involved. Most of the parallel/distributed processing work only considers optimization to minimize run-time. Run-time is, in general, a very important criterion, but there are other criteria to be considered, such as fault-tolerance, security and predictability.

1.4 PROBLEM COMPLEXITY

Many techniques have been developed to address task allocation problems in parallel and distributed computing. For some special cases, polynomial time algorithms are known that can optimally allocate and schedule a set of tasks for parallel execution. The special cases for which polynomial time algorithms are known cover a very small fraction of the kinds of systems with which the ECS block will be concerned. These cases will be handled separately, and the techniques developed here will be reserved for those cases where no polynomial time algorithms for task partitioning and allocation are known.

To understand the size of the problem space under investigation, consider that if there are n nodes in the IM, and m nodes in the LM, and if any node from the LM can be mapped to any node in the IM, then there are n^m different ways of allocating LM nodes onto IM nodes. Allowing a task to be redundantly allocated to more than one node in the IM, results in $(2^n - 1)^m$ possible allocations. The determination of an exactly optimal allocation may require a search through all possible allocations. In some cases there will be constraints that limit the number of resources to which a particular task can be allocated. Cases such as these will help to reduce the number of possible allocations, but the general problem still needs to be considered.

The general multiprocessor task allocation and scheduling problem under resource constraints was shown to be a member of the class of *NP-complete* problems by M. R. Garey and D. S. Johnson[1] in 1975. There are no known algorithms for finding exact solutions to NP-complete problems with less than $O(c^n)$ time complexity where n is a measure of the size of the problem and c is some posi-

tive constant. This limits the magnitude of problems that can be solved exactly to n less than 1000 or so. The problems considered by the ECS block may often require allocations where n is on the order of 1K to 1M. This requires relaxation of the requirement that solutions be exactly optimal. Techniques that produce solutions that are good enough (satisfy the system requirements and measures of effectiveness) will have to be sufficient. Many different algorithms have been developed to produce solutions to problems of this complexity. Some of these algorithms use rules of thumb, or heuristics, to produce good solutions. Some algorithms are based on stochastic hill climbing techniques. In this work algorithms of both types are examined.

1.5 ALGORITHMS CONSIDERED

To date, four different algorithms have been investigated. Two of the algorithms examined are stochastic algorithms. These algorithms are the simulated annealing and genetic algorithm methods. Two of the techniques discussed are based on heuristics that derive from common observations about the nature of good solutions to the task allocation problem. These methods are the simulation based and communications based algorithms.

1.5.1 Simulated Annealing

Annealing is a physical process that can improve the strength and resilience of metals and glasses. Annealing is performed through a series of heating and cooling cycles based on a controlled schedule. The material is usually heated to its melting point, and then cooled slowly to allow its molecules to assume a stable configuration. The stable configuration desired would typically be that which optimizes the strength of the material at some temperature. The material is then reheated to a lower temperature, and cooled slowly to allow generation of a better stable state. This process is repeated many times according to a precise annealing schedule. This allows the gradual formation of a molecular arrangement that optimizes the strength of the material under normal operating conditions. The annealing schedule specifies the heating and cooling rates for each phase.

Simulated annealing is a technique that attempts to exploit a useful analogy between statistical mechanics, and combinatorial optimization problems. Statistical mechanics is concerned with the behavior of systems with many variables, or degrees of freedom, in thermal equilibrium at some temperature. Combinatorial optimization problems involve the minimization of a cost function that depends on many parameters. Simulated annealing algorithms are designed to find solutions to combinatorial optimization problems by appealing to an appropriate temperature cooling schedule that insures convergence to the global optimum for the cost function. This is analogous to the process of annealing in metals or glasses, and provides a powerful computational technique for the optimization of the characteristics of large, complex systems. The resource allocation problems examined here are combinatorial optimization problems, and simulated annealing techniques provide a good candidate for generation of good solutions. The cost function that describes the solution fitness is substituted for the energy function that is minimized by the annealing process, and a stochastic model of the annealing process taken from the area of statistical mechanics is used.

The technique of simulated annealing dates back to the work of Kirkpatrick, Gelatt and Vecchi,[2] and Stuart and Donald Geman.[3] Their work, in turn, was based on the work of N. Metropolis, et al.,[4] and the work of J. Willard Gibbs, who introduced a probability distribution governing the energy macrostates of a large system of particles. If such a system of particles is defined by a set of atomic positions $\{r_i\}$, then the probability associated with the presence of that configuration is given by $exp(-E\{r_i\})/k_BT)$, where $E(\{r_i\})$ is the energy of the configuration, k_B is the Boltzmann constant, and T is the temperature in degrees Kelvin. There is a fundamental question in statistical mechanics which concerns what happens to a large system of particles in the limit of low temperature. At elevated temperatures, ground states and configurations close to them in energy, are extremely rare. As the temperature is reduced, the Boltzmann distribution collapses into the lowest energy state of the system. Low temperature is not a sufficient condition for finding ground states though. If a system of particles is melted, and then cooled too rapidly, the configuration that arises may correspond to a locally, rather than globally, optimal structure.

There are two types of simulated annealing algorithms which have been popularized recently: The first is the Classical Simulated Annealing distribution (CSA). The second is the Fast Simulated Annealing distribution (FSA).

Classical Simulated Annealing. The classical simulated annealing algorithm, as introduced in Kirkpatrick, Gelatt and Vecchi[2] and Stuart and Donald Geman[3] can be described more formally as follows.

There exists a set of N sites $S = \{s_i\}$, $1 \le i \le N$, and one considers a family of random variables $X = \{X_s, s \in S\}$. For simplicity, a common state space is assumed, say $\Lambda = \{0,1,2, \ldots, L\text{-}1\}$, so that each X_s can assume only a finite number of values. Letting $\omega = (x_{s_1}, x_{s_2}, \ldots, x_{s_N})$, where $x_{s_i} \in \Lambda$ for every i, and letting $U(\omega)$ be a cost function to be minimized, the Gibbs probability distribution is given by the following:

$$p(w) = exp\{-U(w)/T(t)\}/Z,$$

where Z is the sum of the numerator of $p(\omega)$ over all ω. In addition let

$$U^* = max_\omega U(\omega),$$
$$U_* = min_\omega U(\omega),$$
and $$\Delta = U^* - U_* .$$

Geman and Geman[3] proves that the Gibbs distribution, given any starting vector for N sites, converges as time tends to infinity to the uniform distribution on the set of minimizing points of U, provided that the following conditions hold:

1. There exists a time interval τ during which all sites are visited, regardless of the starting point.

2. The temperature $T(t)$ tends to zero as time tends to infinity.

3. $T(t) \geq N\Delta/\log t$ for all $t \geq t_0$, where $t_0 \geq 2$.

Obviously, when U has a unique absolute minimum, the Gibbs distribution converges with certainty to the absolute minimum of U.

A more general situation arises when Λ represents a continuum of values rather than a finite set of points. In this context, it would be desirable to have an algorithm for generating points that makes sense with respect to the annealing schedule. If one specifies an ε-covering of the entire space (i.e., a covering by spherical neighborhoods of radius ε) then one may ask for a time-dependent distribution function governing change of the underlying state vector such that the time-unconstrained random walk that evolves from the process visits every ε-neighborhood infinitely often with probability 1. This time-dependent distribution should be naturally coupled to the annealing temperature schedule. When the distribution is coupled in this way an acceptance distribution that is intimately connected with the underlying cost function is obtained. This provides a logical correspondence between state generation and state acceptance, the former being necessary to obtain appropriate inputs to the acceptance distribution.

The classical simulated annealing distribution, uses a Gaussian probability law with a diagonal covariance matrix such that all component variances are identical, and equal to $T_0/(1 + \log t)$, $t \geq 1$. The numerical results obtained using the classical methodology depend somewhat on the choice of the initial temperature T_0. Choosing a value for T_0 that is too large relative to the problem under consideration is wasteful because of the excessive computer time required for the solution to converge. A value for T_0 that is too small would not allow the satisfaction of requirement (3) above, and this would prevent the convergence to the optimum point, or points. An upper bound on Δ, the difference between the maximum and minimum of the cost function, would allow us to ascertain feasible choices for T_0. The better this bound is the better our choice for T_0, and the better the efficiency of our annealing process.

Fast Simulated Annealing. Since the annealing schedule temperature decreases logarithmically as a function of time and the tails of the Gaussian distribution are rather modest, the classical algorithm converges rather slowly. This provides the motivation for the second annealing algorithm. The second type of generating distribution is called a Fast Simulated Annealing distribution. The Fast Simulated Annealing algorithm (FSA) was first described by Szu and Hartley[5] and is based upon the Cauchy probability law. The distribution has infinite variance, and the annealing schedule is T_0/t, $t \geq 1$. Now the temperature decreases much more rapidly and the variance is infinite. These two facts imply that an algorithm based on the Cauchy generation law and a Gibbs acceptance law would be much faster than those based on a Gaussian generation law and a Gibbs acceptance criterion.

A rigorous convergence theory for simulated annealing which uses two distributions, one for generation and another for acceptance, still needs to be developed . The work of Geman and Geman[3] does not invoke separate distributions for these processes.

1.5.2 GeneticAlgorithm

Another search algorithm that has proven useful for solving combinatorial optimization problems is the *genetic algorithm* (GA). Genetic algorithms were developed at the University of Michigan by John Holland, his associates and students. Part of the motivation for the GA comes from the observation that the most complex systems in nature are the result of evolutionary processes. The same principles that control adaptation and survival in nature are applied in the genetic algorithm to provide a robust search of complex problem spaces. The GA attempts to exploit principles of evolution and natural selection, to produce good solutions to optimization problems. Many of the operators used to develop solutions are modeled after the basic principles of gene replication and recombination in biological organisms.

A genetic algorithm works by maintaining a population of *chromosomes* each of which represents one point in the solution space. The chromosomes are ranked according to their fitness through the evaluation of the objective function that is to be optimized. Chromosomes are selected from the population according to their fitness and recombined into new chromosomes through a *crossover* operation. The crossover operation exchanges material from two parent chromosomes to create a child chromosome that combines the best features of its parents. During the crossover operation a child may occasionally experience *mutation* by having one or more of its elements randomly altered from those of its parents. This helps to prevent the search from becoming stuck at a local optimum.

Chromosome Encoding. A chromosome can be modeled as a fixed length vector, or string. Each element is a member of a finite alphabet. In nature the DNA alphabet is made up of four nucleotides, thymidine (T), guanosine (G), cytidine (C), and adenosine (A). A fragment of a DNA strand might be expressed as a string of nucleotides, for example,

...AATCCGACGGTGATCTT... .

At a higher level of abstraction, a chromosome is made up from series of genes. Each gene codes for a different attribute of the organism and can take on one of a fixed number of possible values, or alleles. One allele may result in blues eyes, and another in brown eyes.

In the computer simulations finite-length arrays of values chosen from an appropriate alphabet are used. For example,

<01100011101010010000100> .

Each location encodes a particular indicator and is analogous to a gene. The alphabet provides the values, or set of alleles, from which one is chosen for each location to specify a particular simulated chromosome.

Genetic Operators. There are three primary genetic operators that work together to form the most basic genetic algorithm. They are selection, crossover and mutation.

Selection. The chromosomes are selected from the population for breeding according to their fitness as determined by evaluation of the cost function. One simple selection strategy is known as roulette wheel selection. Given a population containing N chromosomes, if the fitness of the ith chromosome is f_i, then the probability of selection is given by the following:

$$p_{sel} = \frac{f_i}{\sum_{j=0}^{N-1} f_j}$$

This is analogous to performing selection with a roulette wheel, where each individual is assigned a pie shaped slice whose size is proportional to the fitness of the chromosome.

Crossover. Crossover is a technique for choosing material from each parent chromosome and recombining it to produce the child chromosome. An appropriate crossover operator creates more highly fit offspring by combining the best features of its parent chromosomes. The simplest example of a crossover operator is known as single-point crossover. After two individuals have been selected from the population by the selection operator, a decision is made whether to perform crossover on them based on a predefined probability of crossover, p_c. If crossover is to be performed then a crossover-site is selected by choosing a random number between 1 and L -1, where L is the length of the chromosomes. Two new chromosomes are constructed by exchanging material from the two parent chromosomes on either side of the crossover point. For example,

parent 1: <110100110010>
parent 2: <000111011000>

Here $L = 12$, and we choose a crossover site is randomly chosen, such as 4. This gives us the following situation:

parent 1: <1101|00110010>
parent 2: <0001|11011000>

crossover-site

child 1: <1101|11011000>
child 2: <0001|00110010>

Child 1 is constructed by taking the first four elements of parent 1 and the last eight elements of parent 2. Child 2 is constructed by taking the first four elements of parent 2 and the last eight elements of parent 1. The two child chromosomes replace their parents in the next generation. If crossover is not performed the two parent chromosomes are copied unchanged into the next genereation. With an appropriately designed coding, the crossover operator works to combine the best features of the two parent chromosomes to produce offspring with even higher fitness.

Mutation. Mutation acts by randomly changing features of the child chromosome from those of its parents and helps to provide a mechanism for introducing genetic material into a population. This can prevent a population from converging to a local optimum. Mutation is usually performed by deliberate mis-copying during the crossover operation. As each element is copied from the parent to the child chromosome it may be changed, with low probability, to another random value from the genetic alphabet.

Example. Figure 1-3 shows an overall picture of the basic operation of a simple genetic algorithm using single-point crossover with N=18, L=18 and a crossover site of 7. Parent 1 and parent 2 have been chosen according to their fitness using roulette wheel selection and are recombined to create two child chromosomes. Notice that one of the chromosome values in child 1 has been changed by the mutation operator.

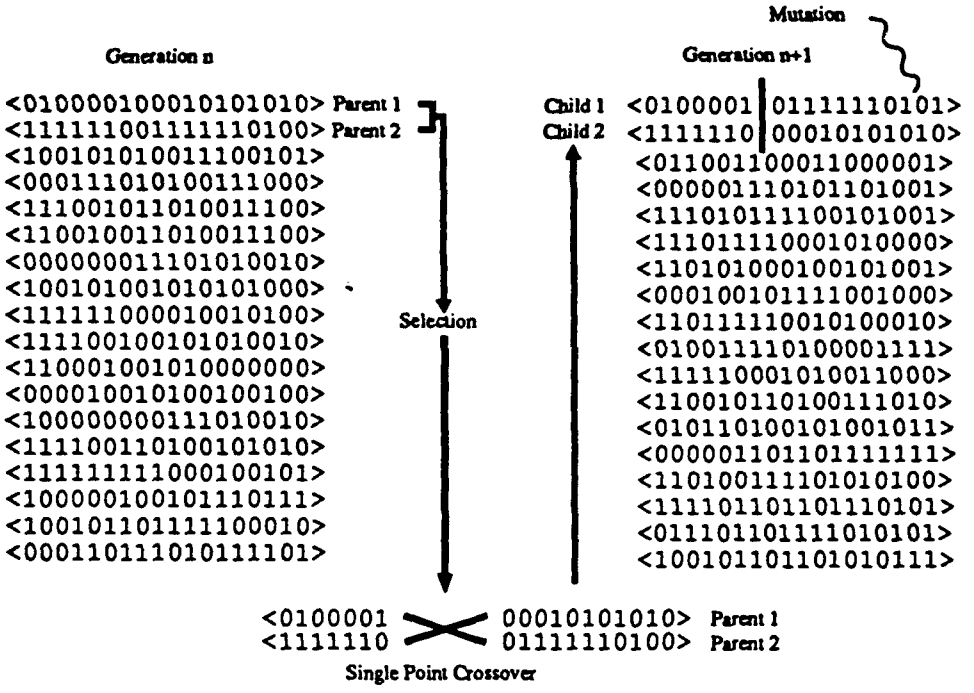


FIGURE 1-3. A SIMPLE GENETIC ALGORITHM

The Schema Theorem. Given this simplistic and rather artificial definition for a genetic algorithm, the question arises as to whether it really provides a useful analogy to the adaptive processes that operate in nature. To provide some feeling for why the genetic algorithm is a good optimization technique, a brief introduction to the concept of schema processing, first described by John Holland[6] is given. In addition, an outline of the schema theorem as described by David Goldberg[7] is also presented. Some notational conventions to describe the elements of the genetic algorithm will be required.

A chromosome is defined as a fixed-length vector of length L over some fixed alphabet of cardinality M. Each position can be referred to by its index.

$C = \langle a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9 \rangle$

To keep the notation simpler the example of a binary alphabet, $A = \{0, 1\}$, can be used without loss of generality.

Holland suggests that a simple genetic algorithm works by processing a set of similarity templates or *schema*. Schema are constructed from an augmented alphabet, $A+ = \{0,1,*\}$, where the * represents a *don't care* condition. The schema,

<11********>

represents the set of chromosomes that have ones in the first two elements. Each chromosome represents 2^L schema since schema can be constructed with values that take on a *don't care*, or the actual value, at each location. A chromosome constructed from an alphabet of cardinality M represents M^L schema. The introduction of schema allows the consideration of the *similarities* between different chromosomes. Presumably, highly fit individuals in a population of chromosomes share similar characteristic values at certain array locations. Schema can be thought of as *building blocks* for the genetic algorithm and represent the fundamental units of information that are processed by the genetic algorithm. A population of size P processes somewhere between M^L and $P \cdot M^L$ schemata in each generation, depending on the population diversity. The fact that a population of size P can process information about such a large number of schemata with comparatively few function evaluations (P for each generation) is referred to as *implicit parallelism*.

There are two very important characteristics of schemata that have a large impact on whether they will propagate successfully in succeeding generations.

• The order of a schema, $o(S)$, is taken as the number of fixed positions (non * positions) in the schema. For example:

o(<**12*11*>) = 4
o(<1***3***>) = 2

• The defining length $\delta(S)$ is given by the difference between the first specific schema value, and the last specific schema value. For example:

δ(<1*1**3*>) = 5
δ(<10*****>) = 1
δ(<***1***>) = 0

The order of a schema provides information about its generality. Low order schemata have few fixed positions and so are represented by many chromosomes. Low order schemata also have a greater chance of surviving the mutation operator and therefore of being propagated to the next

generation. If the probability of mutation for an individual bit is given by p_m, then the probability of a schema surviving the mutation operator unchanged is

$$p_s = (1 - p_m)^{o(S)} .$$

assuming statistical independence. Assuming very small values for p_m, ($p_m << 1$) then p_s can be approximated by

$$p_s = 1 - o(S) \cdot p_m .$$

The defining length gives an indication of how susceptible a schema is to disruption by the crossover operator. When two parents are chosen using the selection operator, crossover may be performed with probability p_c. When the crossover operation is applied to a pair of chromosomes, there is a high likelihood that a schema contained by one of the chromosomes will be disrupted if it has a long defining length, since the crossover site has a high probability of falling somewhere within it. If the probability of crossover is given by p_c, then a schema will survive the crossover operation with probability

$$p_s \geq 1 - p_c \cdot \frac{\delta(S)}{L - 1} .$$

The *schema theorem* for genetic algorithms can now be presented. The schema theorem defines the probability that a schema is propagated from one generation to the next assuming the simple genetic algorithm with single point crossover as described previously. If the number of chromosomes containing a particular schema S, in generation n, is given by $m(S, n)$ then the number of chromosomes representing this schema expected in the next generation is given by:

$$m(S, n + 1) \geq m(S, n) \cdot \frac{f(S)}{\bar{f}} \left(1 - p_c \frac{\delta(S)}{L - 1} - o(S) p_m \right) \qquad (1\text{-}1)$$

where

$$\bar{f} = \sum_{i=0}^{N-1} \frac{f_i}{N}$$

is the population average fitness. The form of Equation (1-1) shows that the genetic algorithm allocates numbers of schema representations in a population in proportion to the ratio of a schema's fitness to the average population fitness. The bracketed term gives a lower bound on the combined probability that a schema will pass unscathed through the crossover and mutation operators. This term depends on the order and defining length of the schema in a manner such that low-order, short defining length schemata are passed with high probability. The result demonstrated by the schema theorem is that highly fit schemata with low order and short defining length experience exponential growth as the population advances through successive generations. Conversely, schemata with low

fitness receive exponentially decreasing numbers in succeeding generations. The exponential growth/decay characteristics demonstrated by the schema theorem are analogous to the population dynamics exhibited in nature as a species adapts to its environment.

Two characteristics of the GA that make it particularly powerful are its implicit parallelism and its ability to work from a large population of possible solution points. Implicit parallelism allows the GA to search through an enormous number of schemata with comparatively few function evaluations. Working from a population of solutions allows the GA to search from many points simultaneously.

The description of a genetic algorithm presented was based on a specific representation for a simulated chromosome and on specific techniques for simulating the selection, crossover and mutation operators. In nature, the chromosome is actually a much more complex structure and there are many other operators that work at the DNA level during the processes of reproduction and recombination. The simple GA described here serves as a baseline for the techniques that will be considered in this work. There are many choices to be made as to how to represent the problem domain by simulated chromosomes and how to implement the various genetic operators. This is an active area of research in the field of genetic algorithms and careful thought and analysis will be required to develop techniques that are especially suited to the resource allocation problems that are addressed by this task.

1.5.3 Simulation Based

The simulation based allocation approach is developed from a greedy scheduling algorithm. The algorithm works by traversing the logical model task graph and attempting to assign tasks to the available node that can execute them most quickly. It does this through a discrete event simulation process. As a task is assigned to a node the node is marked as executing and becomes unavailable for other tasks until the task execution time expires. At that time the node is returned to a ready state and another task may be allocated to it. This results in the construction of a task execution schedule similar to a Gantt chart, and provides both an allocation, and a schedule for the logical model execution. The algorithm is *greedy* since it always chooses the fastest available node for the next task assignment. This is an easy strategy to implement but is based on the assumption that good solutions can be found simply by taking the *locally* optimal step at each point in the solution generation. This assumption can lead to the generation of good solutions, but there is no guarantee that these solutions will be optimal or even near-optimal. An advantage of this approach is that it generates a schedule that can be checked for validity and that resource constraint violations can be checked relatively easily as the algorithm executes.

1.5.4 Communications Based

The communications based algorithm works by performing an analysis and ranking of the intertask communication burden. When possible, task pairs that have a high volume of message traffic are placed on the same node. An upper bound is placed on the number of tasks that can be

allocated to any one node. This provides a crude method for balancing the execution load. If two communicating tasks cannot be placed on the same node they are assigned to the available nodes that provide the fastest communication link between them. The assignment process is performed by traversing the LM and keeping track of the previous assignments made. As an unassigned task is encountered, the algorithm first attempts to place it on the same node with the predecessor task that has the largest communication volume. If that fails various alternatives are tried in succession to determine an allocation that provides the fastest available communication links for the transfer. This is another greedy allocation algorithm that can generate good allocations but provides no guarantee of optimality.

1.6 BACKGROUND ON THE PARALLEL PROGRAM PARTITIONING AND SCHEDULING PROBLEM

The problem of partitioning, allocating and scheduling programs into parallel tasks for execution on a multicomputer is one that has been studied in various forms by many people since the development of parallel computers. This work is of direct relevance to the problems considered by the ECS block. A great deal of effort in the first year of this task has been directed toward a comprehensive study of some of this work. This section provides a brief summary of some of the relevant conclusions from previous work. By far the most frequently studied problem has been the minimization of run-time. This is just one of the many optimization criteria that will be examined by this task. Run-time minimization is an extremely important issue and is one of the primary factors driving the push toward the use of parallel hardware. Run-time minimization is a good choice for an initial attempt to develop usable allocation tools. This provides a good test problem and allows comparison of the results with the results of previous work.

Shahid Bokhari[8] describes a mapping problem for processor arrays. Bokhari shows that this problem is equivalent to the graph isomorphism problem and also shows that it is very similar to both the bandwidth reduction problem and quadratic assignment problems. These are combinatorial optimization problems for which no polynomial time solutions are known. The processor array architectures considered by Bokhari are a somewhat limited example of the types of IM structures with which this task will be concerned. Bokhari attempts to limit communications traffic by mapping adjacent task modules to adjacent processor nodes. The optimization criterion is obtained by assessing the number of intertask edges (representing intertask communication) that fall onto processor array edges. This gives rise to a direct analogy with the graph isomorphism problem. Two graphs are said to be *isomorphic* if there is an exact one-to-one correspondence between the vertices of both graphs, and an exact one-to-one correspondence between the edges of both graphs. Given two graphs, the graph isomorphism problem is to determine an isomorphic mapping from one graph onto the other, if one exists. This is one of the classic unsolved combinatorial decision problems for which no exact polynomial time solution is known. For the purposes of this task it is often desirable to allocate two communicating tasks onto the *same* node; allocating them to adjacent nodes may well be a second best alternative. The example used by Bokhari is a finite element analysis performed in parallel on a specialized array processor. Bokhari describes a heuristic algorithm for finding good mappings for these kinds of problems when the array sizes are relatively small (e.g., 6 by 6).

S. Bollinger and S. Midkiff[9] describe an approach to the mapping problem for multicomputers using the simulated annealing algorithm. They perform the allocation in two phases. The first phase is a *process annealing* phase in which tasks are allocated to processors. The second phase is a *connection annealing* phase which maps interprocessor communication onto the communication links in an attempt to minimize the communication overhead. This two phase approach effectively separates the process of task assignment to balance execution loads from the process of link assignment to balance interprocessor traffic. The results presented in this paper are most easily applied to a homogeneous multiprocessor, such as a hypercube, and so are not directly applicable to the more general case of a heterogeneous system.

The process annealing phase is done purely on the basis of reducing communications cost because of the assumption that all processors are of the same type. The cost function used for this phase consists of two parts: The first is the sum of the interprocessor delays over all the communicating tasks. An intertask communications delay is estimated by taking the product of the data volume transmitted, with the distance in *hops*, or the number of link crossings necessary to get to the node where the receiving task resides. The observation is made that this does not adequately estimate the delays due to overloaded link traffic on some communication links. To address this problem a second function is defined that estimates the traffic on the most saturated link. The process annealing then attempts to minimize a weighted sum of these two terms. This has the effect of minimizing overall communications traffic at the same time as minimizing the worst case link traffic load. In the link annealing phase, process assignments are fixed and different routes for the intertask communication are considered in an attempt to balance the interprocessor traffic flow.

N. Mansour and G. Fox[10] describe a hybrid genetic algorithm approach to solving a partitioning problem for multiprocessors. Their approach is based on the assumption that all processors are running the same code and alternate between calculation and communications phases. A homogeneous architecture such as a hypercube is also assumed. These assumptions cover a small subset of the kinds of allocation problems that this task will consider, but there are number of useful ideas related to application of a GA to this problem. The hybrid GA used consists of three stages. The first phase attempts to cluster the tasks so that communication costs are minimized. The second phase concentrates on reducing the execution time costs and balancing the execution loads. The third stage uses a hill climbing technique where the genetic algorithm is abandoned in an attempt to zero in on the local minimum. This hybrid approach may be worth further investigation. Mansour and Fox also use an *inversion* operator that occasionally reverses a contiguous section of a chromosome. The chromosome representation used is similar to that used in experiments with GAs described in Chapter 3, with the exception that the chromosome is treated as a ring during the crossover and inversion operations. The cost function used considers both execution time and communication time factors in a manner similar to that described in Chapter 2.

H. Muhlenbein et al.,[11] describe an evolutionary approach to mapping parallel programs to parallel processors. Their approach is based on a *replicator equation* that describes population growth in the form of a differential equation on a set of fitness functions. They apply this technique to the Graph Partitioning Problem (GPP) and the Traveling Salesmen Problem (TSP). The GPP attempts to partition a graph into clusters such that the communications cost between clusters is minimized. This is a combinatorial optimization problem that arises in several areas. It has a direct

mapping to the load balancing problem in parallel computing. It is an NP-hard problem. The replicator equation technique described attempts to provide a mathematical model of population growth and is based on very different principles from those exploited by the genetic algorithm. The statement of the parallel task assignment problem as a graph partitioning problem and the cost functions used to describe it are very useful though.

C. Price[12] presents a very similar problem to the one examined by this task. Price states the problem as a 0-1 quadratic programming problem. An execution time matrix is used that provides estimates of the run-time cost of assigning a task to a particular processor. This is the same as the W matrix defined in Section 2.1. Price assumes that the communication network for her problem is fully connected and attempts to minimize an expression that is a sum of a term describing the run-time cost derived from W , and a term that describes the communications cost. Price expresses the allocation in terms of an assignment matrix that performs the same function as the mapping vector, M defined in Chapter 2, and presents an iterative hill-climbing technique that converges to a local optimum when given a starting position. The assignment array matrix used could provide a coding for allocations as simulated chromosomes for a genetic algorithm. The hill-climbing technique described by Price may be useful at the end of simulated annealing, or genetic algorithm execution, to home in on a final solution.

This provides a very brief summary of some related work in the area of parallel and distributed computing. The volume of prior work in this area is enormous, particularly in the area of run-time optimization. An important part of this task will continue to be the evaluation of the previous work. In the coming year this task will examine work done in the areas of optimizing fault tolerance and optimal scheduling for real-time systems.

CHAPTER 2

PROBLEM STATEMENT

This chapter defines some terminology to allow the concise statement of the resource allocation problems addressed by this task. Some cost functions are then derived for use with the simulated annealing and genetic algorithms.

2.1 SOME NOTATION FOR THE MAPPING PROBLEM

To describe the task partitioning problem and the necessary cost functions more formally, it is helpful to develop some standard notation.

A task decomposition graph can be derived from the logical model. This graph is essentially a simplified version of the logical model that contains only the features that are relevant to the derivation and analysis of the cost functions. This is referred to as the Logical Graph, $L = \langle V_L , E_L \rangle$. Figure 2-1 defines the terms related to the logical graph.

N_T - The number of tasks in L = $| V_L |$.

t_j - the jth task in L, $0 \le j \le N_T - 1$.

C=(c_{jk}) - Connectivity matrix: where $c_{jk} = \{0,1\}$ indicates a link in G_L from task t_j to task t_k. $0 \le j,k \le N_T - 1$.

D=(d_{jk}) - Data volume matrix: where d_{jk} indicates the quantity of information (bits) flowing from t_j to t_k. $0 \le j,k \le N_t - 1$.

W=(w_{jk}) - Run time matrix: the quantity w_{jk} is the runtime cost to execute task t_j on node n_k. $0 \le j,k \le N_t - 1$.

FIGURE 2-1. LOGICAL GRAPH

Similarly, an Implementation Graph, $I = <V_I, E_I>$, can be derived from the Implementation Model. Figure 2-2 defines the terminology that describes the characteristics of the Implementation Graph.

N_p - The number of nodes in $I = |V_I|$.

n_j - the jth node in I, $0 \le j \le N_p - 1$.

L=(l_{jk}) - Connectivity matrix: where $l_{jk} = \{0,1\}$ indicates a data link from node n_j to node n_k, $0 \le j,k \le N_p - 1$.

T=(t_{jk}) - Setup time matrix: the quantity t_{jk} is the setup time for data transmission from n_j to n_k, $0 \le j,k \le N_p - 1$.

R=(r_{jk}) - Communication rate matrix: the quantity r_{jk} represents the transmission rate (Bits/Sec.) for data transmission from n_j to n_k, $0 \le j,k \le N_p - 1$.

S=(s_{jk}) - Distance matrix: the quantity s_{jk} gives the minimum distance in hops from n_j to n_k, $0 \le j,k \le N_p - 1$.

FIGURE 2-2. IMPLEMENTATION GRAPH

A mapping, **M** , of the Logical Graph onto the Implementation Graph is defined as a vector that specifies which node n_j, each task t_j is executed on.

M=(m_j) - A vector such that the quantity m_j gives the index of the node in I onto which the jth task from L has been allocated.

2.2 OBJECTIVE FUNCTION FORMULATIONS

The development of a set of objective functions that characterize the expected performance of a particular mapping is probably the most difficult single problem addressed by this research. This is the primary area of investigation at this early stage of the project. The objective, or cost function, is required for both the simulated annealing and genetic algorithm techniques. This function will be evaluated many times during the execution of these algorithms, and must evaluate as quickly as possible for the search to converge in a reasonable time. It is necessary to develop objective functions that are simple to evaluate, and yet provide a good estimate of the quality of an allocation. The requirement that the objective function execute rapidly tends to rule out careful analysis of communications traffic and schedule validity, at least in the initial stages.

Initial research has concentrated on the minimization of run-time. The focus on run-time minimization allows this task to get the greatest leverage from previous work done in the area of parallel

and distributed computing. This also provides a good base problem to insure that the allocation algorithms are functioning properly. In this section consideration is given to the development of the objective functions independently from considerations about specific implementations of simulated annealing, or genetic algorithms.

2.2.1 Run Time Optimization

Run time performance is one of the most important criteria when considering an allocation. Two characteristics of the allocation tend to dominate other factors when considering run-time. The first is the time spent executing the tasks in the LM on the nodes from the IM that are specified by the allocation. The second consideration is the amount of time spent performing intertask communication. Most of the work on partitioning and allocation for parallel or distributed programs has concentrated on optimizing one or both of these two factors. The problem can be stated as follows. Given a valid mapping, M , find:

$$\min \quad T_{total}(M) = T_{exec}(M) + T_{comm}(M) \quad . \tag{2-1}$$

Equation (2-1) expresses the total run-time for mapping M as the sum of two terms: one that describes the execution time cost of the mapping, and one that describes the communication time cost of the mapping.

Some of the fundamental information required to determine T_{exec} for a particular mapping is contained in the execution time matrix W, and the Logical Graph. The actual time spent in raw task execution can be determined by looking up the appropriate element in W. Another critical factor in determining T_{exec} is the amount of parallelism that can be obtained from the mapping. An accurate estimate of the time saved due to parallel execution requires some consideration of the task precedence constraints, and the schedule used to execute the mapping. In the worst case, this may require an actual simulation of the task execution for mapping M. In fact, the mapping does not provide any information about the schedule, or even guarantee that a valid schedule exists. To keep the problem relatively simple in the early stages of this task, realistic considerations of schedules and effective parallelism will be ignored. As the work progresses, this issue will be examined more carefully and an attempt will be made to determine more accurate methods for addressing this problem.

A first approximation to T_{exec} can be taken by simply summing the appropriate elements of the execution time matrix, as follows:

$$T_{exec} = \sum_{j=0}^{N_T - 1} w(j, m(j)) \, . \tag{2-2}$$

This expresses the cost due to raw execution time, with no consideration given to the potential time savings due to parallel execution. Equation (2-2) may undeservedly reward allocations that use a small number of nodes, since these have lower communications costs. It does not properly encourage

allocations that balance the load more evenly. If we ignore communications costs ($T_{comm} = 0$), then optimal allocations can be found by placing each task on the processor that executes it most quickly.

Another way of estimating T_{exec} that does give some consideration to the potential parallelism is the following:

$$T_{exec} = \frac{1}{\varepsilon \cdot N_{par}} \sum_{j=0}^{N_T - 1} w(j, m(j)), \qquad (2\text{-}3)$$

where N_{par} is the number of nodes used by the allocation, and ε is a measure of the *parallel efficiency*. The efficiency of a parallel program is often defined in the literature on parallel processing as the ratio of the *speedup* obtained by using several processors, to the number of processors used. The efficiency of a parallel program is usually somewhat less than 1. This happens because overhead costs due to communications and scheduling detract from the time saved by parallel execution. An ε of 1 implies a linear speedup as the number of processors is increased and this is usually not attainable. Equation 2-3 tends to give lower (better) scores to allocations that use more nodes, since this indicates greater parallelism. The value of ε will vary for different allocations and is difficult to determine accurately. For our purposes we can arbitrarily assume some fixed number less than or equal to 1.

The second term from Equation (2-1), T_{comm}, is also quite difficult to estimate accurately. The communications time cost depends on the mapping, M, and the precise characteristics of both the logical and implementation graphs. When two communicating tasks are placed on different nodes, delays are introduced when sending data from one to the other. These delays depend on many factors. There are many different ways to send information between two nodes, and often times many different paths along which it can be sent. Some nodes may be connected by dedicated links, and some nodes may be connected through general communications networks. Circuit switching, or packet switching networks may be employed. It may be very important to consider the actual traffic flowing on each link, at each moment in time, to balance the link traffic evenly. Otherwise, allocations may be accepted that have saturated links and incur a greater communications cost than anticipated.

A useful start can be made to define some of the factors that affect T_{comm} by making some simplifying assumptions: (1) Problems with link saturation will be ignored for the moment. (2) Internode communications will be assumed to always use a *shortest path* route, where path length is measured as the number of hops from one node to the next. If we also assume that all communications links are of the same type then a particularly simple equation for T_{comm} can be presented that describes the most basic characteristic of communications cost. In some fundamental sense, the communication cost for two tasks that reside on different nodes is related to the product of their *distance* from one another, and the volume of data that must be sent. If the communications links are identical, then a simple measure of internode distance is given by the distance matrix, S, and a simple value for time spent transmitting data between the tasks is given by the following:

$$t_{task\ j,k} = d_{jk} \cdot s_{jk} , \tag{2-4}$$

where d_{jk}, is the data volume between tasks j and k, and s_{jk} is the minimum distance in hops. T_{comm} is:

$$T_{comm} = \sum_{j,k} d_{jk} \cdot s_{jk} . \tag{2-5}$$

In general, communication links in the implementation graphs are not homogeneous. To address this problem consideration needs to be given to the characteristics of each link as expressed in the values for *setup time* and *communications rate*. The setup time is a delay incurred to initiate a data transfer on the link. The communication rate describes the bandwidth of the datalink. A similar approach can be taken to the one used to derive Equation (2-5). Again, the assumption is made that interprocessor communication always uses the shortest path between the two communicating nodes. When more than one such shortest path exists, one is arbitrarily chosen. A function $p_{jk}(l_{mn})$, is defined such that,

$p_{jk}(l_{mn}) = 1$ if data link l_{mn} is used to transmit information from node j to node k,
$= 0$ otherwise.

The cost for transmitting d_{jk} bits across link l_{mn} is given by the following:

$$t_{link\ m,n} = t_{mn} + \frac{d_{jk}}{r_{mn}} . \tag{2-6}$$

This is the sum of the setup time for link l_{mn}, and the time to transmit d_{jk} bits, at r_{mn} bits/second across the link. The time spent communicating between task j and k is given by summing over all the links:

$$t_{task\ j,k} = \sum_{m,n} p_{jk}(l_{mn}) \cdot t_{link\ m,n} . \tag{2-7}$$

A function $q(j,k)$ is defined such that,

$q(j,k) = 1$: if task j communicates with task k.
$= 0$: otherwise.

Now T_{comm} can be found by summing over all the tasks:

$$T_{comm} = \sum_{j,k} q(j,k) \cdot t_{task\ j,k} . \tag{2-8}$$

Because Equation (2-8) assumes shortest path communications, it ignores difficulties of scheduling link traffic. It also does not guarantee that the fastest link configuration is used, since the shortest path is measured in hops, and does not consider the link transmission speeds. Even when there is no other traffic on the links, the shortest *transmission time* path between two nodes will depend on the message size. It may be advantageous to pay a high setup time cost to use a fast data link if the message is large, and to use a different path with low setup time and slower data rate when the message is small.

Equation (2-1) provides a useful description of the important factors to consider when optimizing run-time performance. The initial attempts to estimate the values for T_{exec} and T_{comm} are still somewhat primitive, but have the advantage of being relatively easy to evaluate. This is an important factor to consider since it directly affects the amount of time required for the simulated annealing and genetic algorithms to generate allocations. Despite the many complexities involved in determining accurate values for the components of Equation (2-1), enough information is available to begin some preliminary experiments. It may be that the result of allocations based on simple measures, such as the ones developed here, can be used to help prune the search space. These allocations may provide useful starting points for more rigorous analysis by other techniques. These simple cost functions provide a good starting point for the initial development of the algorithms, since they provide reasonably predictable kinds of allocations. This will allow checks of the simulated annealing and genetic algorithms to be made to insure that they are working correctly. The cost functions derived here also serve to point out some of the important issues related to run-time performance optimization, and will be developed further in subsequent work. Three important concerns that still need to be addressed are (1) A closer examination of the allocation's schedulability and the effective parallelism obtained (2) Checks to insure that an allocation does not violate resource constraints; (3) Link traffic scheduling.

2.2.2 Optimizing Fault Tolerance

Optimization for fault tolerance will be examined in the coming year's work.

2.2.3 Other Optimizations

There are many other criteria that need consideration when generating allocations. Some of the other criteria of interest are the following:

- Reliability
- Predictability
- Security
- Maintainability
- Cost.
- Weight

Examination of some of these other criteria will be performed in subsequent work.

2.2.4 Multiple Criteria Optimization

As the number optimization criteria under consideration increases it will be necessary to develop techniques for simultaneously optimizing several of these factors. This will be addressed in subsequent work.

CHAPTER 3

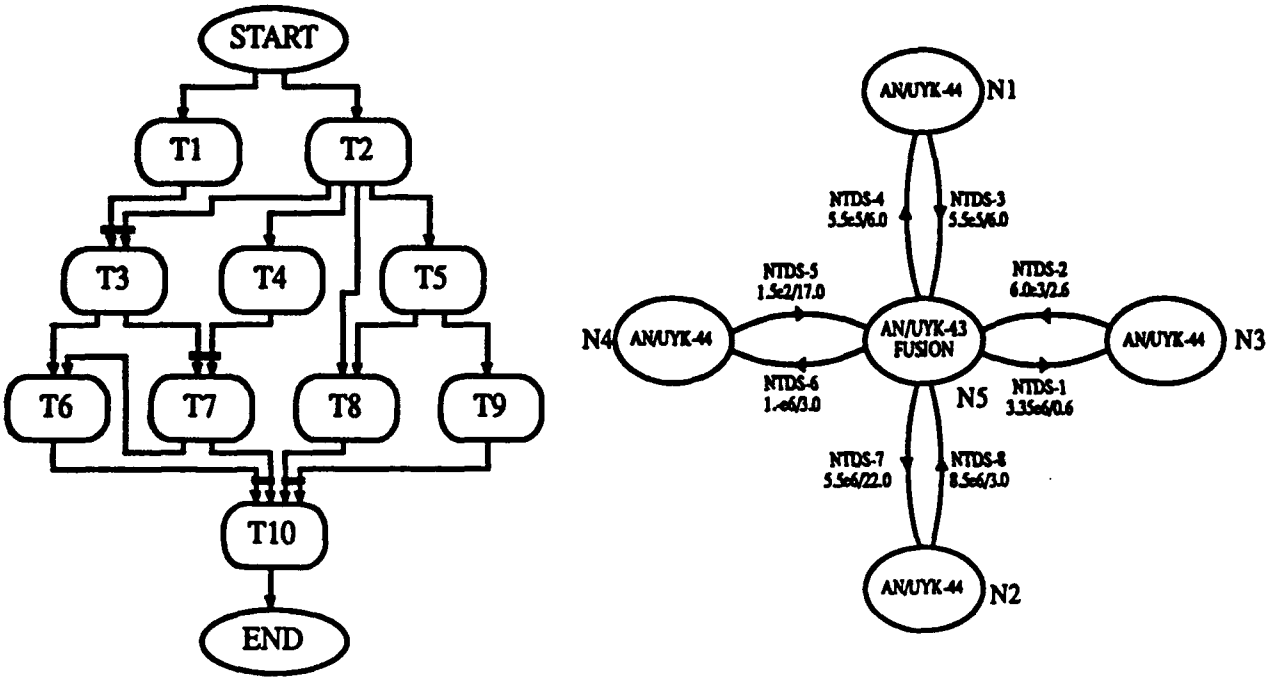
ALLOCATION ALGORITHMS

In the initial approach to the task allocation problem, techniques will be examined for finding near-optimal, single criteria solutions to task partitioning and allocation problems. Initial work has concentrated on the problem of *allocating* tasks from the LM onto the IM. As the project progresses, consideration will be given to problems related to the partitioning and scheduling of the LM tasks. The initial work has concentrated on the issues of run-time optimization. Four techniques are currently being investigated to perform the allocation. The simulation based and communications based techniques described in Chapter 1 are in the early stages of development. The current status of work on the simulated annealing and genetic algorithm allocation techniques is described in this chapter.

3.1 AN EXAMPLE PROBLEM

This section defines some example allocation problems. These problems provide test cases for the algorithms and cost functions under development. The problems are relatively small ones, and the results are presented primarily to allow the algorithm implementations to be checked to see that they are performing as expected. The results also provide some information about the performance of the cost functions defined in Chapter 2. The problems are really too small to allow any confidence that the techniques will scale to problems of the size that eventually need to be considered. However, these problems are useful for the initial debugging of the algorithms themselves. An important goal in next year's work will be to develop some large test cases to check that the algorithms do scale properly. The example problems presented here all use the same logical and implementation graphs, but use different data-volume matrices. This allows evaluation of the performance of the cost functions derived in Chapter 2. Larger intertask data-volumes put more emphasis on the importance of reducing the communications time factor of the cost function, while smaller intertask data-volumes put the emphasis on the execution time factor. The three test problems defined here will be referred to as Problem 0, Problem 1, and Problem 2.

Figure 3-1 shows the logical and implementation graphs for the test problems. The number of tasks in the logical graph shown in Figure 3-1(a) is $N_T = 10$. The implementation graph shows 5 nodes that represent various processor types connected in a star configuration. This gives a value for N_p of 5. Even for this relatively small problem there are $5^{10} = 9{,}765{,}625$ possible allocations. Notice that the communication links are unidirectional so $l_{jk} \neq l_{kj}$. Table 3-1 shows the execution time matrix, W, for the test problems. The same execution time matrix was used for all three problems. The tasks, numbered 1 through 10, are listed by column and the processor resource nodes are listed by row.



(a) LOGICAL GRAPH (b) IMPLEMENTATION GRAPH

FIGURE 3-1. TEST PROBLEM GRAPHS

Data volume weights have been omitted from the edges in Figure 3-1(a) and are presented in tabular form. In the first problem the data-volumes are all zero. This results in a communication cost of zero for all allocations and causes the allocation to be done strictly on the basis of minimizing execution time.

TABLE 3-1. EXECUTION TIME MATRIX

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
N1	5.01	3.44	53.5	17.45	.39	86.21	33.35	25.91	11.44	2.37
N2	4.35	5.83	49.0	22.99	.11	89.31	37.22	26.62	14.55	4.41
N3	6.55	1.22	37.62	10.27	.19	92.5	31.37	23.23	16.32	7.64
N4	5.84	2.78	46.45	25.17	3.65	99.99	21.15	29.16	11.49	4.68
N5	5.6	7.9	45.91	25.25	2.09	87.25	30.01	33.18	10.1	8.77

The data-volume matrix, D, for Problem 1 is given by Table 3-2. Now the optimization must try to balance the cost due to execution time with the cost due to communication time.

TABLE 3-2. DATA VOLUME MATRIX FOR PROBLEM 1

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
T1	0	0	21	0	0	0	0	0	0	0
T2	0	0	77	5.0e3	3.3e6	0	0	.5e6	0	0
T3	0	0	0	0	0	6.2e6	2.3e6	0	0	0
T4	0	0	0	0	0	0	1.2e6	0	0	0
T5	0	0	0	0	0	0	0	6.9e2	3.7e7	0
T6	0	0	0	0	0	0	0	0	0	7.8e4
T7	0	0	0	0	0	4.9e4	0	0	0	5.5e3
T8	0	0	0	0	0	0	0	0	0	7.4e4
T9	0	0	0	0	0	0	0	0	0	9.9e2
T10	0	0	0	0	0	0	0	0	0	0

The values of the data-volume matrix given in Problem 1 have been multiplied by a factor of 100 to create the data-volume matrix for Problem 2. This has the effect of causing intertask communication costs to dominate the cost function when two communicating tasks are allocated to different nodes. The data-volume matrix for Problem 2 is shown in Table 3-3.

The cost function used in the initial experiments is based on Equation (2-1), namely:

$$T_{total} = T_{exec} + T_{comm} .$$

T_{exec} is found using Equation 2-2, and T_{comm} is found using Equation (2-8).

The problem then is to find:

$$\min g(M) = T_{total}(M) = T_{exec}(M) + T_{comm}(M) \quad , \tag{3-1}$$

where M is an allocation mapping vector.

TABLE 3-3. DATA VOLUME MATRIX FOR PROBLEM 2

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
T1	0	0	2.1e3	0	0	0	0	0	0	0
T2	0	0	7.7e3	5.0e5	3.3e8	0	0	.5e8	0	0
T3	0	0	0	0	0	6.2e8	2.3e8	0	0	0
T4	0	0	0	0	0	0	1.2e8	0	0	0
T5	0	0	0	0	0	0	0	6.9e4	3.7e9	0
T6	0	0	0	0	0	0	0	0	0	1.8e6
T7	0	0	0	0	0	4.9e6	0	0	0	5.5e5
T8	0	0	0	0	0	0	0	0	0	7.4e6
T9	0	0	0	0	0	0	0	0	0	9.9e4
T10	0	0	0	0	0	0	0	0	0	0

3.2 SIMULATED ANNEALING

The application of the simulated annealing algorithm to the task partitioning and scheduling problem is straightforward and requires the following ingredients: (1) A description of the system configuration which can be developed from the logical and implementation models and represents one particular allocation of logical modules onto hardware nodes, (2) A random generator of moves, or new allocations, (3) An objective, or cost function that ranks the fitness of the allocation is required, and (4) An annealing schedule that specifies temperatures and number of iterations for which the system is to be executed.

The Simulated Annealing (SA) algorithm takes as inputs the Implementation Graph and the Logical Graph described in Chapter 2 and produces a near optimal mapping of the logical graph modules onto the implementation graph nodes. The initial allocation may be generated by the user or can be generated in an arbitrary manner. This provides the first ingredient for the simulated annealing algorithm. Once an initial mapping is provided the simulated annealing algorithm begins its work.

The algorithm begins by calculating the number of possible binary swaps between each pair of logical tasks and implementation nodes for the given graphs. This value is remembered so an estimate of the total percentage of swaps made can be maintained. The algorithm chooses a statistically

significant fraction of the possible swaps and attempts them while keeping a running sum of those that are successful. When all the swaps are complete the percentage of successful swaps is calculated.

The second ingredient for the SA algorithm is the random move generator. A move is performed in the following manner: A random jump is made into a list of the logical tasks and the task pointed to is selected. A random logical task is selected from a hardware node adjacent to the one where this task is currently allocated. The effect of the proposed exchange is evaluated. The new value of the objective function is computed and compared to the last value. If the new allocation is better than the last one the exchange is accepted. If the new value of the objective function is not better than the last value the exchange is rejected with probability of $1 - e^{-\Delta E/T}$, where ΔE is the change in efficiency of the new mapping and T is the temperature. This means that allocations that are not better are kept with a probability that is proportional to the temperature. At high temperatures they may be accepted frequently but as the temperature decreases the algorithm demands a higher degree of fitness from the new allocation. The addition of controlled randomness to the mapping scheme helps to avoid premature convergence to some local optimum.

The third ingredient of the SA algorithm, the cost function, is described in the previous section.

The final ingredient of the SA algorithm is the annealing schedule. At a given temperature, exchanges take place until some target number of successful swaps have occurred, or a sufficient portion of the solution space has been examined. When this occurs the temperature is decreased according to an annealing schedule and the exchange process begins again. The process terminates when the annealing schedule is completed or when the solution freezes and no successful exchanges take place within some set number of tries. The definition of a suitable annealing schedule for a particular problem is given in a somewhat ad hoc manner. Initially the temperature is set to a level where most exchanges are accepted. This is the *melting point* temperature. Cooling then proceeds according to the annealing schedule. Usually a longer schedule that allows slow cooling over many iterations will provide better solutions. As the problem size grows, the amount of computation required for a long schedule becomes unmanageable, and a shorter schedule must be used.

The simulated annealing algorithms are still under development at this time. As they become more mature they will be tested on the example problems described in Section 3.1 so that a comparison of their performance can be made with the performance of the other algorithms. Fast Simulated Annealing algorithm described in Chapter 1 will also be further developed.

3.3 GENETIC ALGORITHM

A simple genetic algorithm has been coded similar to the one described in Chapter 1. This algorithm has been used to generate allocations for the 3 test problems previously described. Three genetic operators have been used to date to solve this problem:

(1) Selection: Roulette wheel selection is performed on the basis of a modified version of the cost function presented earlier. The modification is necessary because the problem has been defined

as a minimization problem and it is necessary to map this into a maximization problem so the genetic algorithm can be applied. This can be done by defining a new problem,

$$\max f(M) = G_{max} - g(M) \text{ when } g(M) < G_{max}$$
$$= 0 \text{ otherwise.}$$

G_{max} is given a large initial value and then reset whenever a larger $g(m)$ is encountered. The initial determinaton for G_{max} is performed in an ad hoc manner. The genetic algorithm used to date is *elitist*. This means that the best chromosome in a population is copied unchanged into the next generation before the selection process begins. This strategy can help to prevent some thrashing that may occur when the genetic algorithm has converged to a stable point, and insures preservation of the best allocation found.

(2) Crossover is performed using the single point crossover operator described in Chapter 1.

(3) Mutation is performed using the method described in Chapter 1.

3.3.1 Codings

A chromosome structure must be defined that represents task allocations, and a coding that maps the chromosome representation to an allocation from task modules to resource nodes must be developed. The coding must be one such that the application of the genetic operators always generates chromosomes that represent valid allocations. At this time, only one encoding has been examined. This encoding uses the same form as the mapping vector, M, described in Chapter 2. The chromosome length is determined by the number of tasks in the logical graph, and an alphabet whose cardinality is given by the number of nodes in the implementation graph is used . For example if the logical graph has 10 tasks, and the implementation graph has 5 nodes (as our example problem does) then one chromosome might be,

<1125324521> .

This would have the interpretation as an allocation where tasks 1 and 2 are mapped onto node 1, task 3 is mapped to node 2, task 4 is mapped to node 5, etc.... This mapping has the advantage of being simple, and of always producing valid allocations when operated on by the crossover and mutation functions. It may not be the best way to represent allocations though, since the communications cost factor of our objective function may be highly sensitive to intertask dependencies that result in highly fit schemata of long defining length. For example, if task 1 transmits a large volume of data to task 10, a highly fit schemata for the allocation problem might be,

<1********1> ,

because tasks 1 and 10 are allocated to the same processor and do not pay a penalty for intertask communication. According to the schema theory presented in Chapter 1, this schema has a high probability of being disrupted by the crossover function. This can make it very difficult for the GA

to find solutions containing this schema. Although the mapping may not be the best from this point of view, the GA does converge rather nicely, at least for this problem. It is important to be aware of the sensitive dependence of the GA performance on the exact mapping used from chromosomes to allocations. This is an area where a great deal of study will be required to develop mappings that work well with the genetic operators employed and that permit the rapid convergence of the solutions to good allocations. This is another issue that requires further analysis.

3.3.2 Some Experiments and Results

The GA was run on each of the three problems presented earlier. A population size of 30 chromosomes and a crossover probability, p_c, of 0.6 were used for all runs. The mutation probability was $p_m = 0.001$. These values were chosen in a somewhat arbitrary manner, and some more experimentation will be required to determine the effect of changes to these fundamental parameters on the convergence of the GA.

Problem 0. Figure 3-2 portrays the allocation found by the GA for Problem 0. The gray shaded areas indicate the nodes that the tasks were assigned to by the allocation. Figure 3-3 shows the performance of the GA on Problem 0. The best solution found had a runtime of 196.63 seconds. The GA had converged to this solution after approximately 78 generations. Figure 3-3(a) shows the graph of T_{total} vs. the generation number. Figure 3-3(b) shows the graph of T_{exec} vs. the generation number. In Problem 0 all communication costs were zero, so these two graphs are equal. The GA simply tries to place each task on the node that will execute it most quickly. Examination of the execution time matrix in Table 3-1 shows that this is exactly what happened. This solution is therefore a global optimum for this problem. The value of f(M), the fitness function that is maximized by the GA, is shown in Figure 3-3(c).

Problem 1. The allocations are shown in Figures 3-4(a) and 3-4(b). It can be seen that as the communication costs rise, the GA prefers allocations that use fewer processors, and clusters task groups with high communications volumes onto the same node. These allocations point out a deficiency in the calculation of T_{exec}. In reality T_{exec} would probably be somewhat lower for allocations that use more nodes since the execution can proceed with a higher degree of parallelism. The calculation of T_{exec} does not properly reward these kinds of allocations and this is an important factor to consider in the next year's work. At that time this task will investigate the results of using a T_{exec} calculation similar to that given by Equation (2-3). This should result in allocations that show a greater degree of parallelism. Figure 3-5 shows the performance of the GA on Problem 1. The data volumes for intertask communication for Problem 1 are given in Table 3-2. They are non-zero so the GA must try to optimize the communication costs as well as the execution time cost. The ordinates of the run-time graphs shown in Figures 3-5(a) and 3-5(d) are the sums of the ordinates of the execution time and the communications time graphs below them, as given by Equation (3-1). Two different runs of the GA on Problem 1 are shown side by side. Allocation 1-A converged to a solution with a total run-time of 234.9 seconds after approximately 145 generations. Allocation 1-B converged to a run time of 225.6 seconds after approximately 141 generations.

Problem 2. Two different allocations are shown in Figures 3-6(a) and 3-6(b). In allocation 2-A all tasks were allocated to node 2. The total run time is 254.4 seconds. In allocation 2-B the GA found an allocation that uses 3 nodes and had a total run time of 254.0 seconds. Figure 3-7 presents the performance results for Problem 2. Problem 2 puts the most emphasis on the T_{comm} component of Equation 3-1 because it has the highest intertask data volumes.

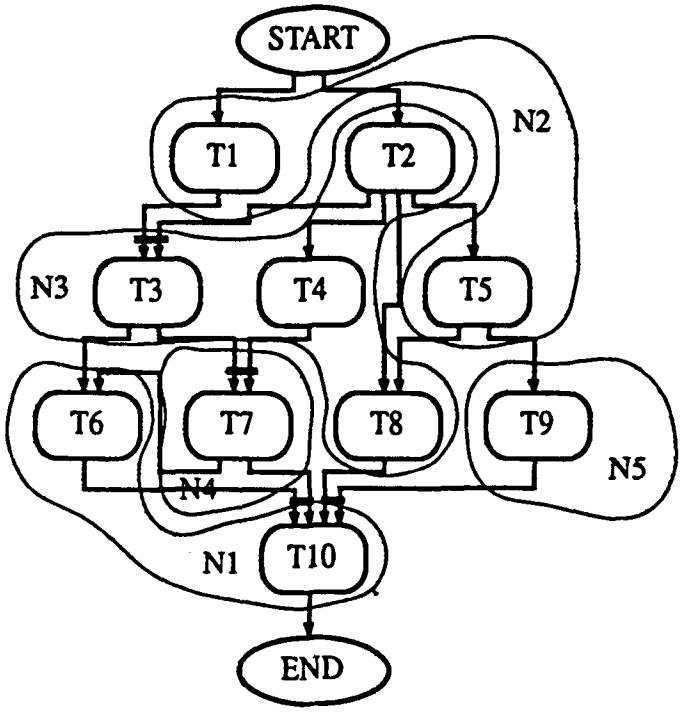
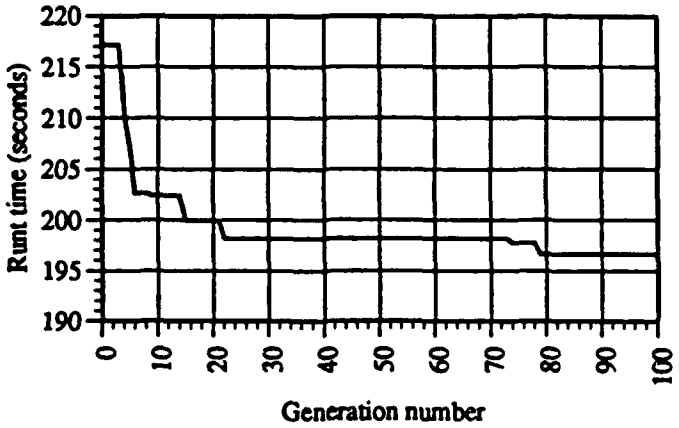
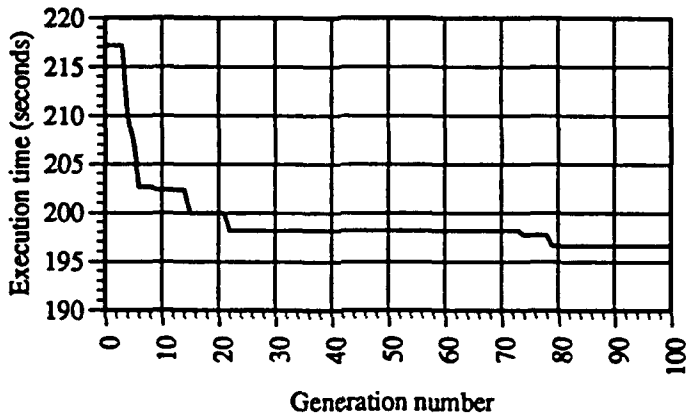


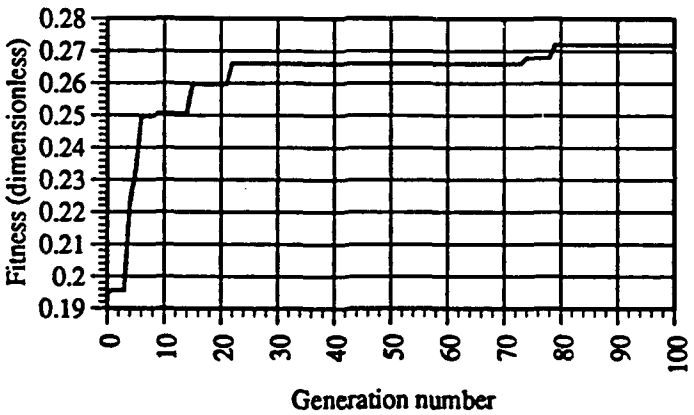
FIGURE 3-2. PROBLEM 0 ALLOCATION



(a) TOTAL RUNTIME VS. GENERATION



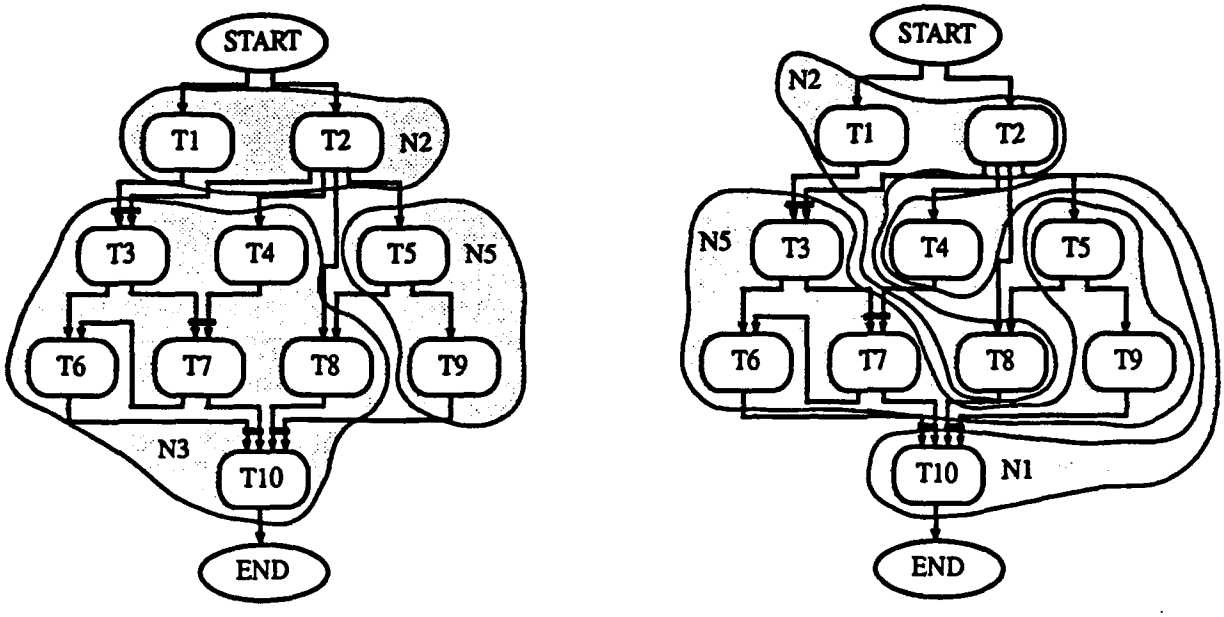
(b) EXECUTION TIME VS. GENERATION



(c) FITNESS VS. GENERATION

Best allocation – <2333214351>

FIGURE 3-3. PROBLEM 0 PERFORMANCE

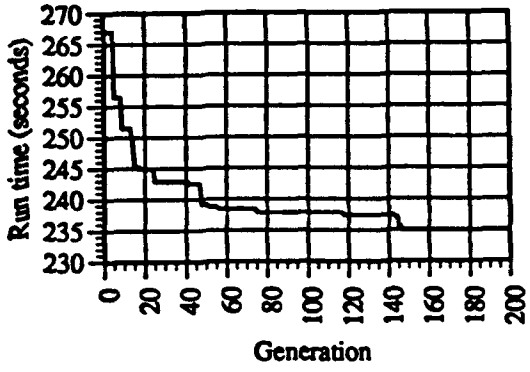


(a) ALLOCATION A

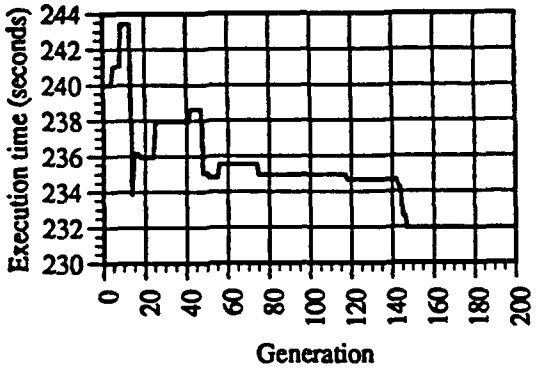
(b) ALLOCATION B

FIGURE 3-4. PROBLEM 1 ALLOCATIONS

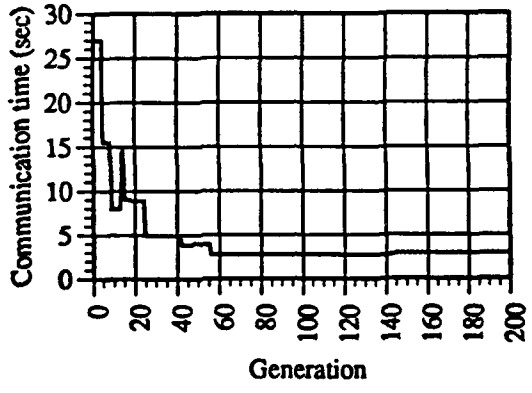
ALLOCATION 1-A



(a) RUN-TIME



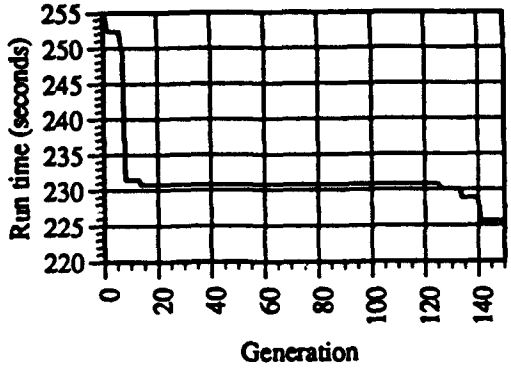
(b) EXECUTION TIME



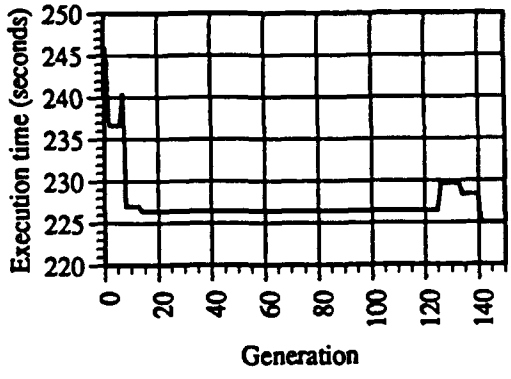
(c) COMMUNICATIONS TIME

Best allocation – <2251555251>
Run time – 234.9

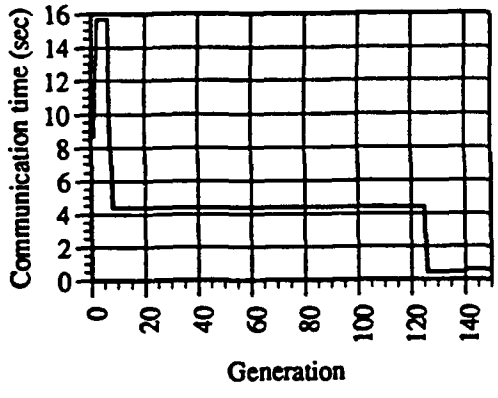
ALLOCATION 1-B



(d) RUN-TIME



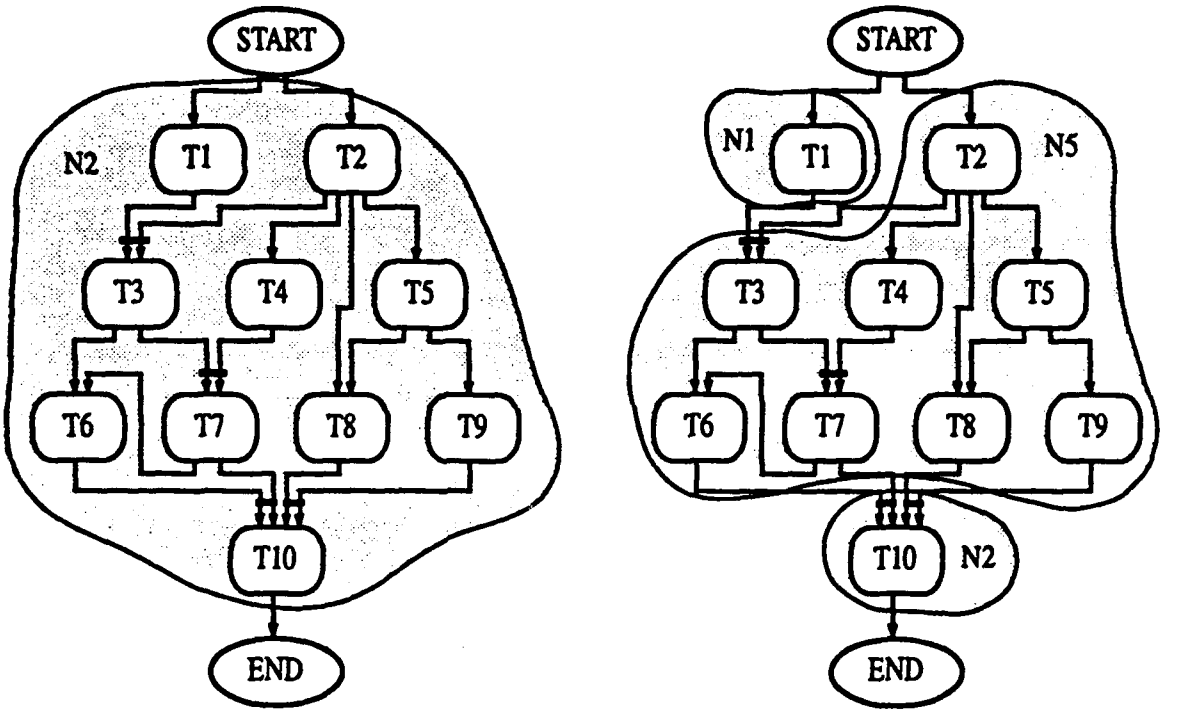
(e) EXECUTION TIME



(f) COMMUNICATIONS TIME

Best allocation – <2233533353>
Run time – 225.6

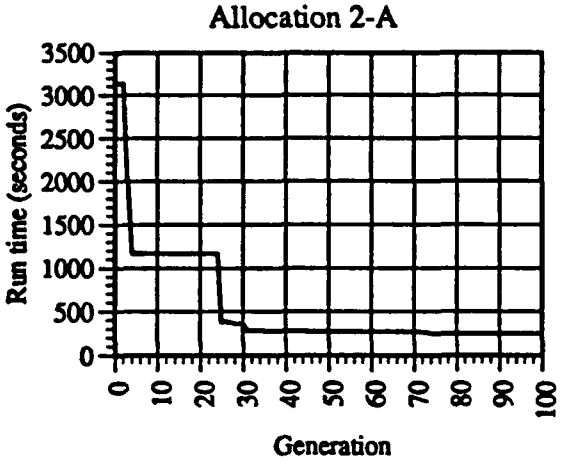
FIGURE 3-5. PROBLEM 1 PERFORMANCE



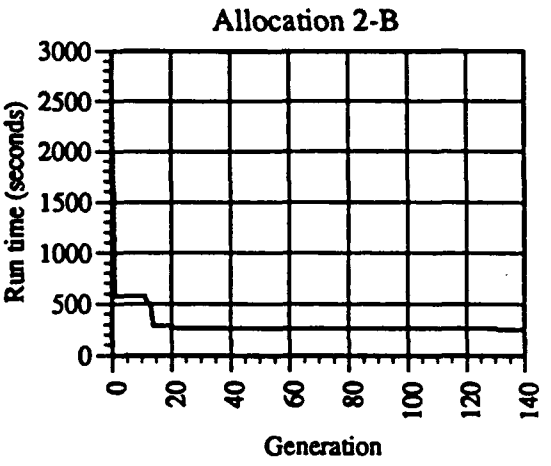
(a) PROBLEM 2 ALLOCATION A.

(b) PROBLEM 2 ALLOCATION B.

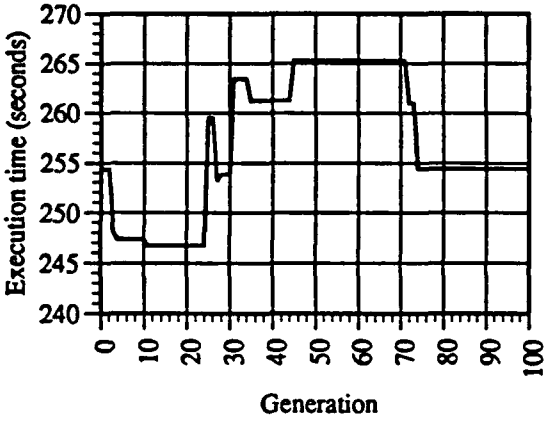
FIGURE 3-6. PROBLEM 2 ALLOCATIONS



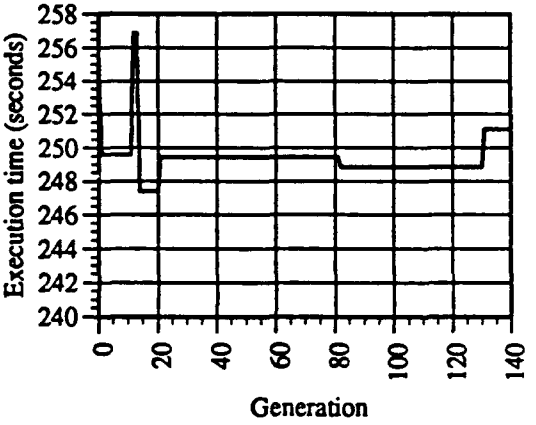
(a) RUN TIME



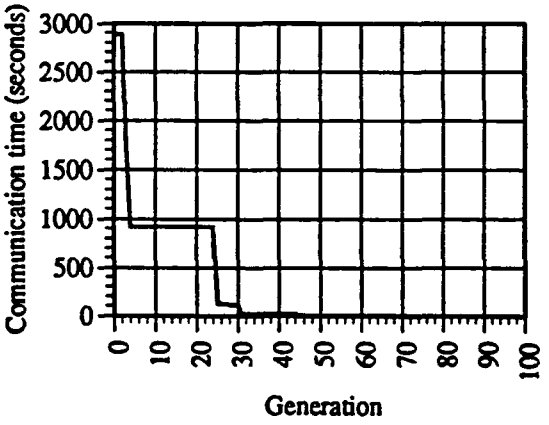
(d) RUN TIME



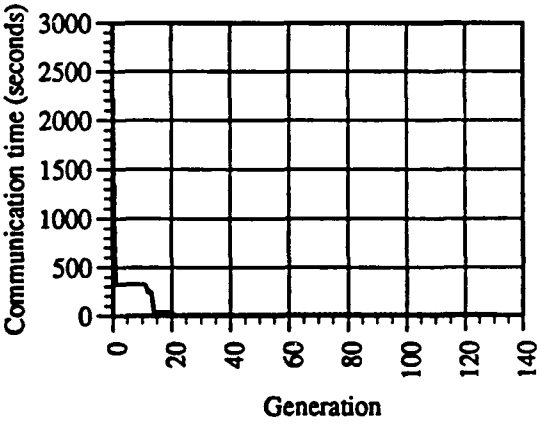
(b) EXECUTION TIME



(e) EXECUTION TIME



(c) COMMUNICATIONS TIME



(f) COMMUNICATIONS TIME

Best allocation – <2222222222>
Run time – 254.4

Best allocation – <1555555552>
Run time – 254.0

FIGURE 3-7 PROBLEM 2 PERFORMANCE

CHAPTER 4

CONCLUSION AND DIRECTIONS FOR FUTURE WORK

This task is still in the early stages of development and much work remains to be done. During the first year the focus has been on a survey of previous related work and on the initial implementation of some optimization algorithms. Some preliminary work on the development of cost functions for run-time performance optimization has been performed. In this section the current status of the optimization algorithms is summarized and issues that will require further investigation are identified.

4.1 SIMULATED ANNEALING

The simulated annealing algorithms are still under development. A basic implementation of simulated annealing using the classical simulated annealing algorithm described in Section 1.5.1 has been generated. Its performance on the three test problems defined in Chapter 3 still needs to be examined to provide a basis for comparison with the other allocation algorithms under consideration. The fast simulated annealing algorithm described in Section 1.5.1 needs further development and this will be an area of investigation in subsequent work. Also, investigation of techniques for parallelizing the simulated annealing algorithms will be performed.

4.2 GENETIC ALGORITHM

A test bed for the evaluation of genetic algorithms has been developed and some preliminary experimentation has been performed using the cost functions derived in Chapter 2. To date, one coding for the allocation problem as a simulated chromosome has been examined. This coding was operated on by the three basic genetic operators described in Chapter 1. The results of the experiments presented in Chapter 3 provide some hope that the genetic algorithm is an appropriate technique for solving the task allocation optimization problems under consideration. A great deal of work remains to determine efficient codings and genetic operators that are especially suited to the problems of interest.

The exact implementation details of the genetic algorithm have a large effect on how well it will perform on a given problem. This can be seen at almost every stage in the definition of codings and genetic operators. A possible deficiency in the coding was pointed out earlier in that it can result in highly fit schemata of long defining length. Actually, the deficiency arises because of the combi-

nation of this coding with the single-point crossover operator used. Other ways of performing crossover may not have the same vulnerability with this coding. In general very tight co-dependencies exist between the coding used and the exact implementation of the genetic operators that act on that coding. Another deficiency of the coding defined earlier is that it cannot conveniently express allocations where the same task is redundantly allocated to more than one node. These kinds of allocations may become important when fault tolerant allocations are examined. Additionally, codings that will allow the expression of execution schedules and link allocations need to be developed so link traffic can be optimized and schedule validity can be checked. This will allow more precise estimates of the factors that need to be considered in the evaluation of the cost functions. Much more examination of codings and genetic operators will be required to insure the development of efficient techniques that are well suited to the allocation problems addressed by this task.

Codings that allow constraints on the allowed values at various positions need to be developed. This would allow the definition of cases where some tasks need to be restricted to assignment to certain nodes. Some of these codings may require more sophisticated representation strategies than the simple string or array representations experimented with so far. The development of multiple-stranded chromosomes, or matrix representations may be required. If this becomes necessary, new methods for performing crossover, mutation and any other genetic operators will need to be developed.

Hybrid strategies that perform the optimization in several different phases need to be examined. For example, it may be useful to concentrate specifically on task clustering optimization in the first phase. An objective function that specifically rewards the clustering of related tasks that can be allocated to the same resource might be of great usefulness. This would allow determination of the large scale structure of the solution space. Once good clustering has been performed the objective function could be changed to encourage smaller scale optimizations. Objective functions may be changed over different phases of the allocation process to encourage the evolution of solutions that address somewhat different criteria at the different phases of the optimization process. Finally, it may be useful to abandon the GA approach for a gradient descent type optimization during the final stage.

Some experimentation needs to be performed to determine the effect of varying key parameters such as population size, crossover rate and mutation rate. It may be useful to vary these parameters during the course of the genetic algorithm execution. Also other genetic operators, such as inversion, should be investigated.

4.3 SIMULATION BASED

The simulation based allocation technique has been developed and is operational. Work on this technique is still in the preliminary stages. An evaluation of the algorithm performance on the test problems has not yet been completed.

4.4 COMMUNICATIONS BASED

The communications based algorithm has been implemented. Additional development is in progress and its performance on the standard test problems will be evaluated in subsequent work.

4.5 COST FUNCTIONS

Development of appropriate object functions for performing the optimizations under consideration is one of the most difficult problems encountered in the execution of this task. Some preliminary work has been performed to develop cost functions for the run-time performance optimization problem. Closer examination of the performance of the cost functions developed for run-time minimization is required. There is a lot of room for improvement in the cost functions used. The calculation of T_{exec} is a particular problem in the experiments presented in Chapter 3, since it does not properly reward parallel allocations in the presence of high communications costs. This results in allocations that use too few processor nodes and have an uneven load distribution. An improvement could be made by calculating T_{exec} using a technique like that suggested in Equation (2-3). This brings up other difficult issues though, such as how to estimate the parallel efficiency, ϵ, of an allocation. On a higher level, this problem is related to the artificial division of the cost equation into separate terms for T_{exec} and T_{comm}. This is a convenient abstraction at some philosophical level but in reality the cost of communication cannot be so cleanly separated from the cost for execution. The two costs are closely coupled and an accurate estimate for the run-time of an allocation depends on a full analysis of the exact execution schedules and link communication traffic. The values provided by the cost functions based on this ad hoc division are highly artificial. The fundamental reason for relying on a cost function of this form is the requirement for rapid execution. This is very important for both the simulated annealing and genetic algorithms. This form for the cost function does have the advantage of being relatively easy to evaluate by comparison with a full simulation of the execution of every allocation. It will be necessary to develop cost functions that have this property for the simulated annealing and genetic algorithms. These functions may appear somewhat contrived, but if properly designed will provide a *heuristic*, or rule of thumb, that will allow identification of regions containing promising allocations from the space of all possible allocations. These regions can then be investigated thoroughly by more stringent methods.

Much work remains to be done in developing and refining the cost functions used by the various optimization algorithms. Cost functions that address other design factors, such as fault tolerance, security, reliability and the others mentioned in Chapter 2, still need to be developed. Also, as the number of optimization criteria expands it will be necessary to consider techniques for performing multiple criteria optimizations.

4.6 TEST PROBLEM SUITE

A more comprehensive set of test problems is required, including some very large problems. This would permit a better evaluation of the performance of the different allocation algorithms. This

exercise presents its own difficulties. Care must be taken to insure that the test suite is large, and well balanced to avoid the development of unrealistic strategies that perform well on the test suite, but that fail miserably in the real world.

REFERENCES

1. Garey, M. R., Johnson, D. S., "Complexity Results for Multiprocessor Scheduling Under Resource Constraints," *SIAM J. Comput.*, Dec 1975, Vol. 4, No. 4.

2. Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P., "Optimization by Simulated Annealing," *Science*, 13 May 1983, Vol. 220, No. 4598.

3. Geman, S., Geman, D., "Stochastic Relaxation, Gibbs Distributions, and Bayesian Restoration of Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Nov 1984, Vol. PAMI-6, No. 6.

4. Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., Teller, E., "Equation of State Calculations by Fast Computing Machines," *The Journal of Chemical Physics*, Vol. 21, No. 6, Jun 1953.

5. Szu, H., Hartley, R., "Fast Simulated Annealing," *Physics Letters A*, Vol. 122, No. 3,4, 8 Jun 1987.

6. Holland, J. H., *Adaptation in Natural and Artificial Systems*, Univ. of Mich. Press, 1975.

7. Goldberg, D. E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, 1989.

8. Bokhari, S. H., "On The Mapping Problem," *IEEE Transactions on Computers*, Vol. C-30, No. 3, Mar 1981.

9. Bollinger, S.W., Midkiff S. F., "Processor and Link Assignment in Multicomputers Using Simulated Annealing," In *Proceedings of International Conference on Parallel Processing*, Aug 1988.

10. Mansour, N., Fox, G. C., "An Evolutionary Approach to Load Balancing Parallel Computations," In *Proceedings of the Sixth Distributed Memory Computing Conference*, Apr 1991.

11. Muhlenbein, H., Gorges-Schleuter, M., Kramer, O., "New Solutions to the Mapping Problem of Parallel Systems: The Evolution Approach," *Parallel Computing*, North Holland, 1987.

12. Price, C., Garner, J., *"Task Assignment Modeled as a Quadratic Programming Problem,"* Dept. of Maths and Statistics, Steven F. Austin University, Technical Report No.DMS-7, 1983

BIBLIOGRAPHY

Aho, A. V., Hopcroft, J. E., Ullman, J. D., *The Design and Analysis of Computer Algorithms*, Addison Wesley, 1974.

Bertsekas, D. P., Tsitsiklis, J. N., *Parallel and Distributed Computation: Numerical Methods*, Prentice Hall, 1989.

Bokhari, S. H., "On The Mapping Problem," *IEEE Transactions on Computers*, Vol. C-30, No. 3, Mar 1981.

Bollinger, S.W., Midkiff S. F., "Processor and Link Assignment in Multicomputers Using Simulated Annealing," In *Proceedings of International Conference on Parallel Processing*, Aug 1988.

Davis, L., Ed., *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, 1991.

Garey, M. R., Johnson, D. S., "Complexity Results for Multiprocessor Scheduling Under Resource Constraints," *SIAM J. Comput.*, Dec 1975, Vol. 4, No. 4.

Geman, S., Geman, D., "Stochastic Relaxation, Gibbs Distributions, and Bayesian Restoration of Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Nov 1984, Vol. PAMI-6, No. 6.

Goldberg, D. E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, 1989.

Holland, J. H., *Adaptation in Natural and Artificial Systems*, Univ. of Mich. Press, 1975.

Holland, J. H., Holyoak, K. J., Nisbett, R. E., Thagard, P. R., *Induction: Processes of Inference, Learning, and Discovery*, MIT Press, 1986.

Huck, J. C., Flynn, M. J., *Analyzing Computer Architectures*, IEEE Computer Society Press, 1989.

Hwang, K. Xu, J., "Mapping Partitioned Program Modules onto Computer Nodes Using Simulated Annealing," In *Proceedings of the International Conference on Parallel Processing*, 1990, Vol. III.

Ibaraki, T., Katoh, N., *Resource Allocation Problems: Algorithmic Approaches*, MIT Press, 1988.

Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P., "Optimization by Simulated Annealing," *Science*, 13 May 1983, Vol. 220, No. 4598.

Ly, T. A., Mowchenko, J. T., "Applying Simulated Evolution to Data Path Allocation in High Level Synthesis," In *Proceedings of Canadian Conference on VLSI*, Oct 1990.

Ly, T. A., Mowchenko, J. T., "Applying Simulated Evolution to Scheduling in High Level Synthesis," In *Proceedings of the 33rd IEEE Midwest Symposium on Circuits and Systems*, Aug 1990.

Ly, T. A., Mowchenko, J. T., "Bottom Up Synthesis Based on Fuzzy Schedules," In *Proceedings of the Design Automation Conference*, Jun 1991.

Mansour, N., Fox, G. C., "An Evolutionary Approach to Load Balancing Parallel Computations," In *Proceedings of the Sixth Distributed Memory Computing Conference*, Apr 1991.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., Teller, E., "Equation of State Calculations by Fast Computing Machines," *The Journal of Chemical Physics*, Vol. 21, No. 6, Jun 1953.

Muhlenbein, H., Gorges-Schleuter, M., Kramer, O., "Evolution Algorithms in Combinatorial Optimization," *Parallel Computing*, North Holland, 1988.

Muhlenbein, H., Gorges-Schleuter, M., Kramer, O., "New Solutions to the Mapping Problem of Parallel Systems: The Evolution Approach," *Parallel Computing*, North Holland, 1987.

Price, C., Garner, J.,*"Task Assignment Modeled as a Quadratic Programming Problem,"* Dept. of Maths and Statistics, Steven F. Austin University, Technical Report No.DMS-7, 1983.

Sarkar, V., *Partitioning and Scheduling Parallel Programs for Multiprocessors*, MIT Press, 1989.

Stone, H. S., *High-Performance Computer Architecture*, Addison Wesley, 1987.

Szu, H., Hartley, R., "Fast Simulated Annealing," *Physics Letters A*, Vol. 122, No. 3,4, 8 Jun 1987.

DISTRIBUTION

	Copies
Defense Technical Information Center Cameron Station Alexandria, VA 22314	12
Library of Congress Attn: Gift and Exchange Division Washington, DC 20540	4
Naval Air Development Center Attn: Code 7033	
(Dr. C. Schmiedekamp)	1
(P. Zombori)	1
Warminster, PA 18974-5000	
Office of Naval Technology Attn: Code 227	
(Elizabeth Wald)	1
(Cmdr. Gracie Thompson)	1
800 N. Quincy Street Arlington, VA 22217-5000	
Center for Naval Analyses 4401 Fort Avenue P.O. Box 16268 Alexandria, VA 22302-0268	2
Naval Research Laboratory Attn: Code 5534	
(Connie Heitmeyer)	1
(Bruce Labaw)	1
Washington, DC 20375	
Naval Research Laboratory Attn: Code 5543	
(Catherine Meadows)	1
Washington, DC 20375	
Naval Postgraduate School Attn: Code AS/RA	
(Balasubramaniam Ramesh)	1
Administrative Sciences Department Monterey, CA 93943	
Naval Postgraduate School Attn: Code EC/LE	
(Chin Hwa Lee)	1
Monterey, CA 93943	
NUWC Attn: John Rumbut Jr. Bldg. 1171-3 Newport, RI 02841-2047	1
Trident Systems Incorporated Attn: Nicholas Karangelen 10201 Lee Highway Suite 300 Fairfax, VA 22030	1
Research Triangle Institute Attn: Geoffrey Frank P.O. Box 12194 Research Triangle Park, NC 27709-2194	1
Advanced Technology & Research Corp.	
Attn: Adrien J. Meskin	5
Goerge Stathopoulos	1
14900 Sweitzer Lane Laurel, MD 20707	
Internal Distribution:	
D4	1
E231	2
E232	3

DISTRIBUTION (CONTINUED)

		Copies
Internal Distribution:		
A10	(R. Scalzo)	1
B		1
B02		1
B04		1
B05	(H. Crisp)	1
B10		1
B10	(W. Farr)	1
B20		1
B40		1
B44		1
B44	(M. Edwards)	1
B44	(N. Hoang)	1
B44	(S. Howell)	10
B44	(M. Jenkins)	1
B44	(T. Moore)	1
B44	(C. Nguyen)	20
B44	(T. Park)	1
B44	(H. Roth)	1
B44	(M. Trinh)	1
B44	(C. Yeh)	1
B44	(M. Wilson)	1
B44	(E. Cohen)	1
B44	(H. Szu)	1
E342	(GIDEP)	1
F01		1
G07	(F. Moore)	1
K02		1
K14	(D. Clark)	1
N35	(M. Masters)	1
N35	(R. Riedl)	1

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	21 March 1994	

4. TITLE AND SUBTITLE
Task Allocation and Scheduling for High Level Synthesis

5. FUNDING NUMBERS

6. AUTHOR(S)
Eric J. Ogata and Edward A. Cohen

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Surface Warfare Center (Code B40)
10901 New Hampshire Avenue
Silver Spring, MD 20903-5640

8. PERFORMING ORGANIZATION REPORT NUMBER
NAVSWC TR 91-538

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report addresses the development of automated techniques for solving resource allocation problems in the high level synthesis of system designs. These techniques are developed with the objective of supporting the design of highly complex systems that are characterized by an extremely large number of physical components of many different types, with complex interconnections and interdependencies. The purpose of the systems that are developed using these techniques is to implement a set of logical functions that define the overall system behavior. These logical functions can be described as a set of communicating tasks that pass data and control signals from one to another. The set of logical tasks must be mapped onto the physical resources from which the system is constructed. There may be many different ways to map logical system tasks onto the hardware resources. A particular mapping can be scored according to how well it satisfies some overall system design goal such as fault tolerance or rapid response time. The questions as to how to identify optimal mappings that maximize or minimize some design parameter is important. This report represents the results of investigations of the performance of four different techniques for identifying optimal or near-optimal allocations given a particular optimization goal.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Resource Allocation Logical Model Implementation Model	58
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL