Dynamic Load Distribution Optimization in Heterogeneous Multiple Processor Systems

Emile Haddad

TR 93-02

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

January 21, 1993

DYNAMIC LOAD DISTRIBUTION OPTIMIZATION IN HETEROGENEOUS MULTIPLE PROCESSOR SYSTEMS

Emile Haddad
Computer Science Department
Virginia Polytechnic Institute and State University
2990 Telestar Court, Falls Church, Virginia 22042.
Tel. (703)-4502156, (703) 6986023

Abstract

We examine the problem of optimizing the distribution of the m interacting modules of a given workload on a parallel system with p heterogeneous processors. Average-valued parameters are used to model the intermodule coupling of the workload and its execution and communication times on the diverse system processors. We derive an analytical optimality criterion to minimize a multi-metric objective function representing a weighted combination of workload completion time, communication cost, resource utilization cost, and processor idle-time. The efficacy of a processing element is defined as a composite measure of its cpu speed, memory speed, and the degree of coupling among the modules of the given workload. The optimal load distribution is found to be the apportionment of the total load among a subset of the system processors composed of the $q \leq p$ most efficacious processors in direct proportion to their efficacies. The criterion provides a straightforward procedure for determining the number q of processors to be engaged in executing the given workload. In the absence of synchronization delays, the optimal distribution results in equal execution time for all engaged processors, thus eliminating idle wait-time and therefore representing the ideal load "balancing" on the heterogeneous system. The p - q least efficacious processors remain unengaged in workload execution and can be allocated to other jobs. The mathematical derivations are exact, with no analytical approximations used in the optimization process. The analysis starts by assuming the total load to be continuously partitionable, and then derives an integer-valued distribution from the the optimal continuous-valued solution. The optimization procedure is highly efficient, with O(p) and O(p2) computational complexities for the continuous and integer-valued solutions respectively. The procedure can be invoked dynamically to implement optimal load migration whenever the benefit of redistribution outweighs its cost. Updated procedure parameters can be efficiently computed in a recursive manner during run-time. Repeated application of the dynamic procedure tends to persistently correct the deviations from optimality resulting from averaging the load parameters and from ignoring synchronization delays in workload time execution modeling.

1. Introduction and Background

Consider a computer workload consisting of m interacting tasks or program modules, $\{M_j\} = \{M_1, M_2, \ldots M_m\}$, to be run on a parallel processing system comprising p processing elements, $\{P_i\} = \{P_1, P_2, \ldots P_p\}$, by allocating to each P_i a module assignment A_i subset of $\{M_j\}$. Let $A = (A_1, A_2, \ldots Ap)$ denote the vector of assignments over the p processors. The coupled module pairs interact through the exchange of data, supported by local resources when the modules reside on the same processor and by shared system resources when the modules run on different processors. Each processor computes its assignment of modules A_i, carries out the internal communication among them, and cooperates with other processors on the external communication between A_i and all other assignments.

1.1 Task assignment versus load distribution

There are various phases associated with the management of this problem: *task definition,* to specify the workload modules $\{M_j\}$; *task assignment,* to specify the initial load distribution $\{A_i\}$; *task scheduling,* to allocate modules of A_i to the local cpu for execution; and *task migration,* to dynamically reassign $\{M_j\}$ by respecifying $\{A_i\}$ during execution. In the management of these problem aspects, one seeks to meet desired goals, usually stated in the form of optimizing certain *objective functions* reflecting specified metrics of problem behavior in *performance* and *cost,* such as minimizing the workload turnaround time or the cost of communication.

The task assignment problem has received a great deal of attention over the past 15 years. Starting with the classic work of Stone [1], [2], several formulations and approaches have been studied. In its most general form, where the workload modules are nonuniform and the system is heterogeneous, the problem is analytically intractable except in the most restrictive cases, and the solution is normally achieved through algorithmic methods whose computational complexity grows exponentially with the dimensions of the problem [22]. Various efficient heuristics have been proposed and studied [4], [5], [17], [18].

A simplified variant of the task assignment problem is the *load distribution problem* [7]-[9], [10]-[14], [23]. In this formulation, one is not primarily concerned with assigning specific modules to specific processors but rather with determining the *loading level* distribution as expressed by the *number of modules* x_i allocated to processor P_i. If we denote the *load distribution* of the m modules over the p processors by the vector $x = (x_1, x_2, \ldots x_p) = (|A_1|, |A_2|, \ldots |A_p|)$, the problem may be stated as follows: how to determine the load distribution $x^* = (x_1^*, x_2^*, \ldots x_p^*)$ which optimizes the appropriate objective function.

1.2 Deterministic versus random modeling

This problem would be meaningful only if, in the modeling of the workload and the analysis of its behavior on the given system, the *identity* of specific load modules is inconsequential. Two types of *workload models* have been used in the literature to supress the identity of the individual load modules and to analyze the load distribution problem:

1. A uniformly partitioned and uniformly coupled *deterministic workload* of equal-sized modules which exchange equal amounts of communications data. Examples of this approach are found in [8], [12], [14].
2. A *random workload* model in which the module size is a random variable identically distributed for all modules and the existence/amount of communication data between pairs of modules are random variables identically distributed for all module pairs. Examples of this approach are found in [7]-[10], [13], [23].

In this paper, we adopt an approach which is akin to the two approaches just described. We consider a nonuniform deterministic workload in which the size of the load modules can vary and the module pairwise couplings, as represented by the existence and magnitude of intermodule communications, can also vary. The analysis, however, uses only the *average values* of the workload parameters. This approach is analogous to the one used for the random workload model referred to above, except that we do not use the stochastic analytical formalism of that model. It should be emphasized that the analysis and results of this paper can be adapted in a straightforward manner to the random model found in [7]-[10], [13], [23]. Our preference not to do so is primarily for the sake of brevity, and to focus attention on the other aspects of the load distribution problem presented in this paper. It should be stressed that our results, as well as any results obtained for the stochastic model, are necessarily approximate in any given situation since they only apply "in the mean". By modeling the average behavior of the workload on the given system, the most we could hope for is to derive approximate rule-of-thumb criteria or guidelines for "optimally" distributing the load modules over the system processors. The accuracy of such criteria improves as the deviation of the individual module parameters from their mean values decreases.

1.3 Optimization objectives

A key aspect of both the task assignment and load distribution problems is the selection of the objective function to be optimized. Investigators have used various cost and performance metrics, including total execution cost [1], [2], [5], [6]; total communication cost [7], [17]; "interference cost" [4], degree of load imbalance [15], [17]-[20], workload execution time [8], [11], and combinations thereof. An important consideration in this regard is the appropriateness and accuracy

of modeling the intended metrics in a parallel/distributed processing environment. Investigators have found it simpler to use *cost* metrics because such metrics are generally *additive,* and are therefore easily modeled, in a concurrent system. Most *performance* metrics, on the other hand, are related to the time behavior of the system which is difficult to model in a parallel setting because of the concurrency that exists among the various time activities in the system, as well as the existence of synchronization time delays. Workload completion time, as a performance measure, has been used for serially executing loads[1], [5], or for parallel loads with simplifying assumptions and restrictions on the overlap among the time activities, and by totally ignoring synchronization delays [7], [8].

This paper examines the load distribution problem in a *heterogeneous* multiple-processor system . We adopt a multimetric objective function H(x) to be minimized, expressed in terms of the load distribution vector $x = (x_1, x_2, \ldots x_p)$, composed of a *weighted combination* of a performance metric and three cost metrics:

$$H(x) = d_t t(x) + d_c c(x) + d_u u(x) + d_w w(x)$$

. where t(x) is a metric estimating the workload *completion time,* c(x) is a metric representing the total interprocessor *communication cost,* u(x) is a measure of the total resource *utilization cost* which may include storage and processing costs, and w(x) is a measure of the cost of *wait-time* for processors that complete their execution before the overall workload completion time (wasted time, idle time). The values of the weight coefficients d_t, d_c, d_u, d_w reflect the *relative importance* of the four metrics in the overall optimization process for a given situation.

1.4 Continuous versus discrete modeling and analysis

The solution to the optimal load distribution problem we have posed is *conceptually* straightforward. There is a finite number of ways in which the m modules can be distributed among the p processors. It can be shown that this number is equal to number of unordered samples of size m (repetition allowed) from a p-element set, which can be expressed as

number of possible distributions = $N(p,m) = p(p+1)(p+2)\ldots(p+m-1)/m!$

By listing all the possible load distributions and evaluating the objective function for each, we can pick the one (or ones) for which H(x) is minimum. This approach of exhaustive enumeration would be tolerable only when p and m are small, since N(p,m) increases exponentially with these parameters. What is more important, however, is that such a "brute force" approach would reveal to us nothing about the *relationship* between the computed optimal distribution and the given parameters of the workload and the system. We may seek to formulate algorithmic or heuristic procedures that reduce the computational effort required for arriving at an optimal or suboptimal solutions. Although such procedures may shed some light on why the prescribed iterative process

converges to the targeted solution, they do not, in general, provide a *closed-form analytical criterion* that relates the optimal solution to the problem parameters.

Alternatively, we may model the problem by relaxing the integer restriction on the load distributions $\{x_i\}$ and allowing them to be *continuous variables* satisfying the constraint $\Sigma_i x_i = m$. In this case, the number of possible distributions becomes infinite, and exhaustive enumeration for optimization would not be feasible. We may, however, be able to derive a closed-form criterion for the optimal solution using the methods of continuous analysis. Such a criterion would be highly desirable because, in addition to providing the *absolute* optimal solution, it would provide insight into the exact analytical dependence of the optimal distributions on the workload and system parameters. The values of x_i thus obtained for the optimal load distribution may not be integer-valued. If the given load modules are fragmentable into smaller tasks, one may attempt to break down some of the modules into smaller partitions in order to *fit* the fractional values of the absolute optimal distribution, thus arriving at a *better* performing load allocation compared to the integer-restricted allocation. If fragmentation of some modules is not possible, then one may obtain an integer-valued distribution by rounding the fractional distributions to the their "closest" integer-valued allocations. The deviation from absolute optimality resulting from integer rounding should not very objectionable if one is, in the first place, using average parameters to characterize the load modules, including the parameters related to the "size" of a module.

In this paper we adopt such an approach. We start by allowing the load distributions to be continuous variables, and derive a rather simple closed-form analytical criterion that specifies the optimal load distribution in relationship to the given workload and system parameters. We find that the total load of m modules should be allocated to a *specific subset* of the system processing elements, such that each processor gets a fraction of the total load proportional to its "efficacy". The notion of efficacy is a defined as a metric related to the processor speed, its memory speed, *and* the degree of intermodule coupling in the given workload. The subset of processors that is engaged for workload execution is represented by the q processors having the largest values of efficacy, where q is an integer determined by the criterion from the given problem parameters. The remaining p - q unengaged (unloaded) processors are free to be assigned to other jobs, if permitted by the systems resource allocation policy. We then formulate simple procedures for obtaining integer-valued distributions, to be derived from the ideal optimal solution if module fragmentation is totally inadmissible.

1.5 Load distribution as preliminary optimization

The load distribution problem as described above may be regarded as a *preliminary* optimization step aimed at determining the optimal *number of modules* x_i to be loaded on each

processor P_i, irrespective of the identity of such modules. This first-step optimization may then be followed by further optimization, focusing on the selection of the *specific* x_i modules, using any of the other known techniques for optimal task assignment presented in the literature. The advantage of carrying out such a preliminary optimization step is that the computational complexity of the problem presented to the task assignment technique would be substantially reduced, since now the number of modules to be assigned to each processor is predetermined and is no longer a variable of the optimization procedure. This would be especially true if the load distribution prescribes assigning zero load to some of the processors. In some cases, as we shall demonstrate later, no further optimization would be required beyond the optimal load distribution prescribed by the techniques of this paper. This would evidently be the case when the optimal load distribution happens to be of the form $(x_1, x_2, \ldots x_p) = (m, 0, \ldots, 0)$, viz. allocating all modules to one processor.

1.6 Dynamic versus static load distribution

A *static* load distribution policy prescribes how the given m load modules should be initially allocated to the p available processors. This is done once at the time of loading the modules for execution on the given system. No further redistribution or migration of modules is allowed during run-time. A *dynamic* load distribution policy, on the other hand, would allow the redistribution of the load modules during the execution period, and specifies when and how should load migration take place. This is normally implemented by monitoring the course of workload execution in real time, and deciding whether load redistribution at some points would be "advantageous". The decision is typically based on a policy that requires the *benefit* from redistribution to outweigh the *cost* of migration, using some appropriate quantification of benefit and cost.

In this paper, we shall first develop a static load distribution criterion which can be used to initially allocate the given modules over the available processors. The distribution minimizes the objective function H(x) described in a previous section.We then extend the applicability of the criterion to run-time, such that the optimal redistribution of the *remaining* (unexecuted) modules can be determined for any specified time t during execution. The resulting dynamic strategy stipulates that migration from the existing distribution to the optimal distribution is carried out only if the resulting reduction in the objective function, i.e. benefit, exceeds the cost of implementing the load redistribution. Although the static procedure can deal with non-integer load distributions, it is likely that a less costly dynamic implementation would result if preemption of partially executed modules i.e. fractional modules, is execluded from redistribution in order to avoid the relatively high overhead of context switching that would otherwise be entailed. We show that the prescribed dynamic load distribution procedure can be efficiently implemented, and that its repeated application during run-time would persistently correct the deviations from true optimality that arise

from using average load parameters instead of module-specific values. If greedily applied, the dynamic procedure tends also to reduce the inaccuracies resulting from ignoring synchronization delays in the time modeling of the concurrent execution of the system processors, as well as from ignoring partially-executed modules as candidates for load migration.

1.7 Organization of the paper

The remainder of the paper is organized as follows. Section 2 presents a computational model of the workload based on a precedence graph representation of the load modules and their interaction, and derives the expressions for the average values of the workload coupling parameters. In Section 3 a discussion of the optimization "decision model" is presented, and expressions are formulated for the components of the multi-faceted objective function H(x): completion time, communication cost, resource utilization cost, and processor idle-time; and the notion of processor efficacy is defined. Section 4 presents in detail the mathematical optimization process for the continuous-valued load distributions, leading to the principal criterion which specifies the optimal load distribution. This section also formulates procedures for obtaining integer-valued load distributions. The dynamic load distribution strategy is elaborated in Section 5. Conclusions and further research are discussed in section 6. The proofs for three lemmas used in the mathematical optimization derivation are presented in the Appendix.

2. Workload and System Models

We assume that the interaction among the set of load modules $M = \{M_i\}$ is represented by a *noncyclic directed graph* $R = (M , E) = (\{M_j\},\{E_{kg}\})$. An edge E_{kg} represents an *intermodule communication* from M_k to M_g indicating a *precedence constraint* in the execution of the corresponding modules, such that M_k must complete execution before M_g can start. We assume that an edge E_{kg} is unique, i.e. there is at most one edge from M_k to M_g. If module M_k happens to send data to M_g more than once, all such communications can be lumped into a single equivalent communication from M_k to M_g.

We shall assume that the transfer of data form M_k to M_g as represented by the edge E_{kg}, is modeled by the following mechanism in a *loosely-coupled* multiple processor system. If M_k and M_g are assigned to the same processor, M_k generates during its execution the data D_{kg} intended for module M_g and *stores* it in a block of local memory. Subsequently, M_g during its execution *retrieves* the same data D_{kg} from local memory. If M_k and M_g are assigned to different processors,

M_k generates during its execution the data D_{kg} intended for module M_g and *stores* it in a block of local memory. After the generation of D_{kg} is completed, the data is transfered in a block, as an interprocessor message, from the local memory of of the source processor to the local memory of the destination processor, where it can be subsequently *retrieved* whenever the execution of M_g gets underway. We assume that this interprocessor transfer between local memories takes place with no, or very little, involvement from the corresponding processors. A typical implementation might use a direct memory access (DMA) mechanism to transfer the block of D_{kg} from local memory to a communication buffer (sender), then via the interconnection network to a receiver communication buffer at the destination processor, then to local memory by another DMA transfer. The inter-buffer transfer implemented by the interprocesor communication protocol, as well as the DMA transfers, require little, if any, time from the processors concerned. Therefore, we shall assume that the communication tasks assigned to the processors involve only the *writing* and *reading* of communication data to and from their local memories. In a *tightly-coupled* system with global memory, the same considerations apply, except that now communications data are exchanged via shared memory.

Let $m = |M|$ and $e = |E|$ be respectively the the total number of nodes and edges, which correspond to the total number of modules and communications among them. Let δ_j denote the *degree* of M_j, viz the total number of edges connected to M_j. Let δ be the *average* value of δ_i over all nodes:

$$\text{Average number of communications per module} \equiv \delta = (\Sigma_j \delta_j)/m = 2e/m \qquad (1)$$

where we have used the familiar relationship, $\Sigma_j \delta_j = 2e$, applicable to any graph. Define the *workload coupling factor*, denoted by λ, as

$$\text{workload coupling factor} \equiv \lambda = \delta/(m-1) = 2e/m(m-1) = e/[m(m-1)/2] \quad , \quad 0 \le \lambda \le 1 \qquad (2)$$

Note that λ is a factor representing the ratio of the average number of edges per node (δ) to the maximum possible value that δ can ever have, namely (m-1), which would be the case if every node were connected to all the other m-1 nodes. Hence $\lambda \le 1$. Another interpretation of λ is given by the expression $e/[m(m-1)/2]$ which is the ratio of the total number of edges to the maximum possible number of edges, namely $m(m-1)/2$, which would be the case if R were fully connected. Thus the factor λ measures the overall average degree of interaction, or coupling, among the modules of the workload. It should be noted that if we were to use the random workload model found in [7]-[10], [13], [23], λ would represent the *probability* t hat an edge exists between any pair of nodes picked at random, i.e. the probability that an arbitrary node pair are coupled.

3. Optimization Decision Model

We now elaborate on the objective function H(x), the minimization of which is the goal of the optimization we are seeking. A load distribution $x^* = (x_1^*, x_2^*, \ldots x_p^*)$ is said to be optimal if

$$H(x^*) = \min_x H(x) \tag{3}$$

The function H(x) we propose is a linear combination of four functions t(x), c(x), u(x), and w(x); representing respectively the workload *completion time*, the total *communication cost*, the total resource *utilization cost*, and the total cost of *wait-time* for idled processors:

$$H(x) = d_t t(x) + d_c C(x) + d_u u(x) + d_w w(x) \tag{4}$$

where d_t, d_c, d_u, and d_w are non-negative constant multipliers. Each of the four component metrics represents a specific problem attribute, whose minimization is desirable. The minimization of each metric *separately* will generally result in four distinct optimal solutions x^* corresponding to the individual metrics. Alternatively, we seek a "compromise" solution by minimizing a *weighted sum* of the various metrics. The weight coefficients can be chosen to reflect the *relative importance* of each component as perceived for any particular situation. We now define each of the four metrics in a precise fashion and derive an explicit expression for each in terms of the load distribution x.

3.1. Workload completion time

The metric t(x) stands for the *workload completion time* . We shall *estimate* the magnitude of this metric by the total execution time of the busiest processor i.e. the processor with the largest execution time:

$$t(x) \equiv \max_i\{(r_i+\delta\tau_i)x_i\} = \max_i\{x_i/a_i\} \tag{6}$$

$r_i \equiv$ mean value of total time spent by processor P_i *per module computation*

$\tau_i \equiv$ mean value of total time spent by processor P_i *per intermodule communication*

$T_i \equiv r_i+\delta\tau_i \equiv$ mean value of total time spent by processor P_i *per module execution*

$a_i \equiv 1/T_i = 1/(r_i+\delta\tau_i) \equiv$ *Efficacy* of processor P_i with respect to the given workload. (7)

Recall from Section 2 the discussion on the modeling of the processor activity related to

intermodule communication. During the execution of a module, a processor attends to an average of δ intermodule communications (graph edges), each requiring access to a data block in memory. If we represent the average size of the data block by D, then τ_i represents the time processor P_i spends accessing D words in memory. Write-access (store) is needed for data to be sent to another module (emanating edge), and read-access (retrieve) is needed for data received from another module (incident edge). The total communication time spent by P_i for every module assigned to it is therefore $\delta\tau_i$. The parameter a_i, which we describe as the "efficacy" of P_i relative to the given workload, depends on the *processor speed*, the *memory speed*, and the degree of *workload coupling* as reflected in the parameters r_i, τ_i, and δ respectively. In all subsequent analysis, we shall assume, without loss of generality, that the set of system processors $\{P_i\}$are indexed according to the non-increasing values of their efficacies :

$$a_1 \geq a_2 \geq \cdots \geq a_p \tag{8}$$

We have described t(x) as being an *estimate* , rather than an exact measure, of the workload completion time because the busiest processor might experience *synchronization delays*. Such *scheduling* delays would occur if the processor happens to attempt starting the execution of a module that needs communication data yet to be received from another processor. When synchronization delays cannot be totally eliminated for the busiest processor by appropriate scheduling, t(x) becomes an *underestimate* of the workload completion time.

3.2. Communication cost

We turn next to the metric c(x)which stands for the *total external communication cost* incurred during workload execution. The minimization of this cost metric has been an optimization goal in many investigations of the load assignment/load distribution problem [7], [4], [17], [21]. Let c be the average cost incurred from the use of the communication network, (or more generally the shared structure), for a single interprocessor data exchange between non-resident modules. For any given load distribution $x = (x_1, x_2,...x_p)$, we express c(x) as:

$$C(x) \equiv N_{ext}(x)c = [(1/2)\lambda\Sigma_i x_i(m-x_i)]c \tag{9}$$

where N_{ext} is the total number of interprocessor communications for the given load distribution x. To explain the expression for $N_{ext}(x)$, consider the δ edges connected to one of the x_i modules allocated to processor P_i. The other ends of these edges are connected to a possible m-1 modules of which x_i-1 are internal to P_i and the remaining $m - 1 - (x_i-1) = m - x_i$ are external modules. Thus

the δ edges are split into internal and external edges in the following ratios:

$$\text{Internal edges per module} = \delta(x_i - 1)/(m - 1) = \lambda(x_i - 1)$$

$$\text{External edges per module} = \delta(m - x_i)/(m - 1) = \lambda(m - x_i)$$

$$\text{External edges for } x_i \text{ modules} = \lambda x_i(m - x_i)$$

$$\text{Total number of external edges} = (1/2)\lambda\Sigma_i x_i(m - x_i)$$

The factor (1/2) is applied to the expression of $N_{ext}(x)$ because every external edge is counted twice when the summation over all i is carried out. Note that the above expression for $N_{ext}(x)$ with $\lambda = 1$ gives the correct number of edges for a fully connected graph, which can be independently verified. It should be emphasized again that the above expressions are identical with the corresponding expressions obtained for the random workload model used in [7]-[10], [13], [23]; with λ representing the probability that an edge exists between any pair of nodes picked at random.

3.3. Resource utilization cost

We assume that a load module placed with processor P_i incurs an average *resource utilization cost* of u_i. This cost might include such charges as the cost per module for *cpu time* and *storage* at the site of processor P_i. The consideration of this cost takes on a special significance when one is dealing with *heterogeneous systems*, where the differences in processing and storage costs at the various processor sites could be important factors influencing the decision of how to distribute the total load. The total resource utilization cost for a given load distribution x is expressed as

$$u(x) = \Sigma_i u_i x_i \tag{11}$$

We shall assume that

$$u_1 \leq u_2 \leq \cdots \leq u_p \tag{12}$$

Recalling the ordering of the processors according to the values of their efficacies $\{a_i\}$ indicated in (8), the assumption in (11) implies that u_i is smaller for a machine with a larger value of a_i. This may be justified by thinking of the parameter u_i not as the cost per load module of "purchasing" the storage and processing hardware of P_i but rather as a "charge" for allocating storage space and cpu time. We should emphasize that this assumption is being made in order to *simplify the presentation* of a two-stage optimization procedure to be described next. This procedure still holds and can be adapted to the cases where the assumption in (12) of ordering the values of u_i is not made.

3.4 Cost of idleness

For any given load distribution x, the job completion time is equal to the completion time of the longest running processor $t(x) = \max\{T_i\}$, where is the execution time of processor P_i. Any other processor engaged in job execution that completes the execution of its load allocation x_i in time $T_i < t(x)$ has to remain idle for a period of time $t(x) - T_i$. We are assuming that the system resource allocation and scheduling policies preclude the utilization of an idle processor P_i until the completion time of *all* other processors engaged with it on the execution of a given workload. Under such a policy, the wait-time period $t(x) - T_i$ represents wasted time for processor P_i, to which we attach a cost $(t(x) - T_i)w_i$, where w_i is a weight factor depending on the particular processor. One expects w_i to have higher values for the more powerful machines in the system, representing higher penalty for their wasted idle time. Thus, the total cost w(x) of wasted time (or wait-time) is expressed as

$$w(x) = \Sigma_i e(x_i)[t(x) - T_i]w_i = \Sigma_i e(x_i)[t(x) - (x_i/a_i)]w_i \quad (13)$$

$$e(x_i) = 0 \quad \text{for } x_i = 0,$$

$$= 1 \quad \text{for } x_i > 0 \quad (14)$$

Note that we have included the *unit- step function* $e(x_i)$ as a multiplier in order to exclude the cost for those processors, if any, that are allocated no load, $x_i = 0$. Such processors are, in effect, *not engaged* with the other processors in the execution of the workload, and may therefore be utilized for any other task.

It should be emphasized that the minimization of idle time in a *heterogeneous* system is tantamount to *load balancing* in a homogeneous multiple-processor sytem, a topic which has been extensively studied in the literature.

3.5 Objective function expression

Substituting the expressions for t(x), c(x), u(x), and w(x) from (6), (9), (11), and (13) into the expression of H(x) in (4), we obtain

$$H(x) = d_t \max_i\{x_i/a_i\} + d_c\lambda\Sigma x_i(m-x_i)]c/2 + d_u\Sigma u_i x_i + d_w\Sigma e(x_i)w_i[\max_i\{x_i/a_i\} - (x_i/a_i)] \quad (15)$$

where Σ stand for "summation over i from 1 to p". In the next section, we shall examine the problem of determining the optimal load distribution vectors $x^* = (x_1^*, x_2^*, \ldots x_p^*)$ which minimize H(x).

4. Optimal Load Distributions

Our goal is to find the load distribution vector $x^* = (x_1^*, x_2^*, \ldots x_p^*)$, such that $\Sigma x_i^* = m$, which minimizes H(x). Define D' as the domain of all *feasible distributions* i.e. the set of all x such that $\Sigma x_i = m$:

$$D'(m,p) \equiv \{ x : \text{Integer } x_i \geq 0, \quad \Sigma_i x_i = m, \quad i = 1, 2, \ldots p \}$$

Note that this implies $0 \leq x_i \leq m$. In the minimization procedure to be shortly described, we shall first allow the load distributions x_i to vary *continuously* over their domain [0 , m], and then show how to obtain an integer-valued distribution. Accordingly, we define D(m,p) as

$$D(m,p) \equiv \{ x : x_i \geq 0, \quad \Sigma x_i = m, \quad i = 1, 2, \ldots p \}$$

Let the symbol ε stand for "element of". We are seeking all $x^* \varepsilon D(m,p)$ such that

$$H(x^*) = \min_{x \varepsilon D} H(x) \qquad , \qquad x^* \varepsilon D(m,p)$$

(16) Define a new function F(x) as

$$F(x) \equiv H(x) - d_w w(x) = d_t \max_i\{x_i/a_i\} + d_c \lambda \Sigma x_i(m - x_i)]c/2 + d_u \Sigma u_i x_i \qquad (17)$$

We shall first determine the minimum of F(x) then show how this leads to the minimum of H(x).

4.1 Two-stage minimization procedure

Expanding and simplifying the expression in (17) by noting that $\Sigma x_i = m$ and setting $\max_i\{x_i/a_i\} = t(x)$, we obtain

$$F(x) = d_t t(x) + d_u \Sigma u_i x_i - A\Sigma x_i^2 + Am^2 = d_t t(x) + G(x) + Am^2 \qquad (18)$$

where we have intoduced the new constant parameter A and the function G(x) defined as

$$A \equiv d_c \lambda c/2 \quad , \quad G(x) \equiv d_u \Sigma u_i x_i - A\Sigma x_i^2 \qquad (19)$$

Our goal is to find the load distribution vector $x^* = (x_1^*, x_2^*, \ldots x_p^*)$, such that $\Sigma x_i^* = m$, which minimizes F(x):

$$F^* \equiv F(x^*) = \min_{x \varepsilon D} F(x) = \min_{x \varepsilon D}[d_t t(x) + G(x) + Am^2] \quad , \quad x^* \varepsilon D \qquad (20)$$

For a given *fixed* value of t, define D(t) as the following subset of D(m,p):

$$D(t) \equiv \{ x : x_i \geq 0, \quad \Sigma x_i = m \ , \ \max_i\{x_i/a_i\} = t \ , \ \text{some given } t \} \qquad (21)$$

Thus, for a given value of completion time t, the set D(t) represents all the admissible load distributions for which the completion time is equal to t. Note that the additional constraint

$\max_i\{x_i/a_i\} = t$ implies the following:

$$x_i \le a_i t \quad \text{for all } i \ , \quad \text{and} \quad x_j = a_j t \quad \text{for some } j = i \tag{21}$$

Thus, fixing the value of t places an *upper bound* $a_i t$ on the load x_i of processor P_i. Thus, D(t) is equivalently defined as

$$D(t) \equiv \{ x : 0 \le x_i \le a_i t \ , \ \Sigma x_i = m \ , \ x_j = a_j t \text{ for some } j = i \} \tag{22}$$

We shall carry out the minimization indicated in (20) in *two stages*. In the first stage we *fix t* and find the minimum of F(x) over D(t), denoted by f(t):

$$f(t) \equiv \min_{x\varepsilon D(t)} F(x) = \min_{x\varepsilon D(t)} [d_t t + G(x) + B] = d_t t + Am^2 + \min_{x\varepsilon D(t)} G(x) \tag{23}$$

where the minimization is applied only to G(x) since the other terms are constant for a given fixed value of t. In the second stage we minimize f(t) over all t

$$F^* = \min_t f(t) \tag{24}$$

The minimization in (24) should be carried over all admissible values of t. However, if we can show that the minimum of f(t) can occur *only* for values of t belonging to an interval [t' , t"], then we need to carry the minimization only over the specified interval. We find that this, in fact, is true as stated in the following lemma.

Lemma 1: *If* $f(t^*) = \min_t f(t)$, *then* $t^* \varepsilon [t_p , t_1]$ *where* $t_1 \equiv m/a_1$, $t_p \equiv m/(a_1 + a_2 + \cdots + a_p)$

The proof of the lemma is presented in the Appendix. In all the subsequent analysis we shall restrict the domain of minimization of f(t) to the values of t in the interval $[t_p , t_1]$.

We now determine the expression for f(t) in (23) as an *explicit* function of t:

$$f(t) = d_t t + \min_{x\varepsilon D(t)} G(x) + Am^2 = d_t t + g(t) + Am^2 \tag{25}$$

Where g(t) is the function of t defined as

$$g(t) \equiv \min_{x\varepsilon D(t)} G(x) = \min_{x\varepsilon D(t)} (d_u \Sigma u_i x_i - A \Sigma x_i^2)$$

We shall determine the expression of g(t) *explicitly* in terms of t, which we then substitute in (25) to obtain the expression of f(t) as an explicit function of t. We shall show that, for any given fixed value of t, the minimum of the function G(x) over all $x \varepsilon D(t)$ is obtained for the specific load distribution $x = \chi(t) = (\chi_1(t), \chi_2(t) , \ldots , \chi_p(t))$:

$$g(t) \equiv \min_{x\varepsilon D(t)} G(x) = G(\chi(t))$$

$$\chi(t) \equiv (a_1 t , a_2 t , \ldots , a_{n(t)} t , m - \Sigma^{n(t)} a_i t , 0 , 0 , \ldots , 0) \quad , \text{ any given } t \tag{26}$$

where the notation Σ^n stands for "summation over i from 1 to n(t)" and where n(t), for any given t, is defined as

$$n(t) \equiv \text{largest integer j such that } m - \Sigma^j a_i t \ge 0$$

$$n(t) \equiv \max \{ j : m - \Sigma^j a_i t \geq 0 \ , \text{ any given } t \} \tag{27}$$

$\Sigma^{n(t)} \equiv$ "summation from 1 to n(t)"

Figure 1 shows the *staircase* variation of n(t) as a function of t over the interval $[t_p , t_1]$. Note that the jump discontinuities of n(t) occur for every $t = t_k \equiv m/\Sigma^k a_i$ with k=1, 2, . . p, at which points the value of the function is equal to k:

$$n(t_k) = k \ , \ n(t_k^+) = k-1 \ , \quad t_k \equiv m/\Sigma^k a_i \ , \qquad k=1, 2, \ldots p \tag{28}$$

where t_k^+ is any value such that $t_k < t_k^+ \leq t_{k-1}$.

Recall from the definition of D(t) in (22) that $x \varepsilon D(t)$ implies $x_i \leq a_i t$ for all i. Thus, for the given completion time t, the load distribution $\chi(t)$ which minimizes G(x), as expressed in (26), gives each of the first n(t) processors its maximum load allocation $a_i t$, and the remainder $m-\Sigma^{n(t)}a_i t$ is allocated to the next processor. Recall from (8) that the first n(t) processors are those with highest n(t) values of efficacy a_i, viz. the n most "efficacious" processors, in our terminology. We now formalize these results in the following Lemma, deferring the presentation of its proof to the Appendix.

Lemma 2: *For any given value of* $t \varepsilon [t_p , t_1]$, *the value of* g(t) *is expressed as*

$$g(t) \equiv \min_{x \varepsilon D(t)} G(x) = G(\chi(t)) = d_u \Sigma u_i \chi_{i(t)} - A\Sigma(\chi_{i(t)})^2 \tag{29}$$

$$\chi(t) \equiv (a_1 t , a_2 t , \ldots , a_{n(t)} t , m-\Sigma^{n(t)} a_i t , 0, 0 , \ldots , 0) \tag{30}$$

$$n(t) \equiv \max \{ j : m - \Sigma^j a_i t \geq 0 \} \tag{31}$$

and therefore the explicit expression of g(t) *in terms of* t *is*

$$g(t) = d_u \Sigma^{n(t)} u_i a_i t - A\Sigma^{n(t)} (a_i t)^2 + d_u u_{n(t)+1} (m-\Sigma^{n(t)} a_i t) - A(m-\Sigma^{n(t)} a_i t)^2 \tag{32}$$

The expression for g(t) in (32) results from substituting the expression for $\chi(t)$ from (30) into (29). Substituting the expression of g(t) from (32) into (25), we obtain f(t) explicitly as a function of t:

$$f(t) = d_t t + g(t) + Am^2 = d_t t + d_u \Sigma^{n(t)} u_i a_i t - A\Sigma^{n(t)} (a_i t)^2 + d_u u_{n(t)+1} (m-\Sigma^{n(t)} a_i t) - A (m-\Sigma^{n(t)} a_i t)^2 + Am^2$$

Noting that t is fixed under the summations with resoect to i , and regrouping the terms, we have

$$f(t) = -A[\Sigma^{n(t)} a_i^2 + (\Sigma^{n(t)} a_i)^2] t^2 + [d_t + d_u \Sigma^{n(t)} u_i a_i + (2mA - d_u u_{n(t)+1}) \Sigma^{n(t)} a_i] t + [d_u u_{n(t)+1} m]$$

We introduce the notation a(n(t)), b(n(t)), and c(n(t)) to represent the three terms in the square brackets above:

$$f(t) = - a(n(t)) t^2 + b(n(t)) t + c(n(t)) \tag{34}$$

$$a(n(t)) \equiv A[\Sigma^{n(t)} a_i^2 + (\Sigma^{n(t)} a_i)^2] \ , \ b(n(t)) \equiv [d_t + d_u \Sigma^{n(t)} u_i a_i + (2mA - d_u u_{n(t)+1}) \Sigma^{n(t)} a_i] \tag{35}$$

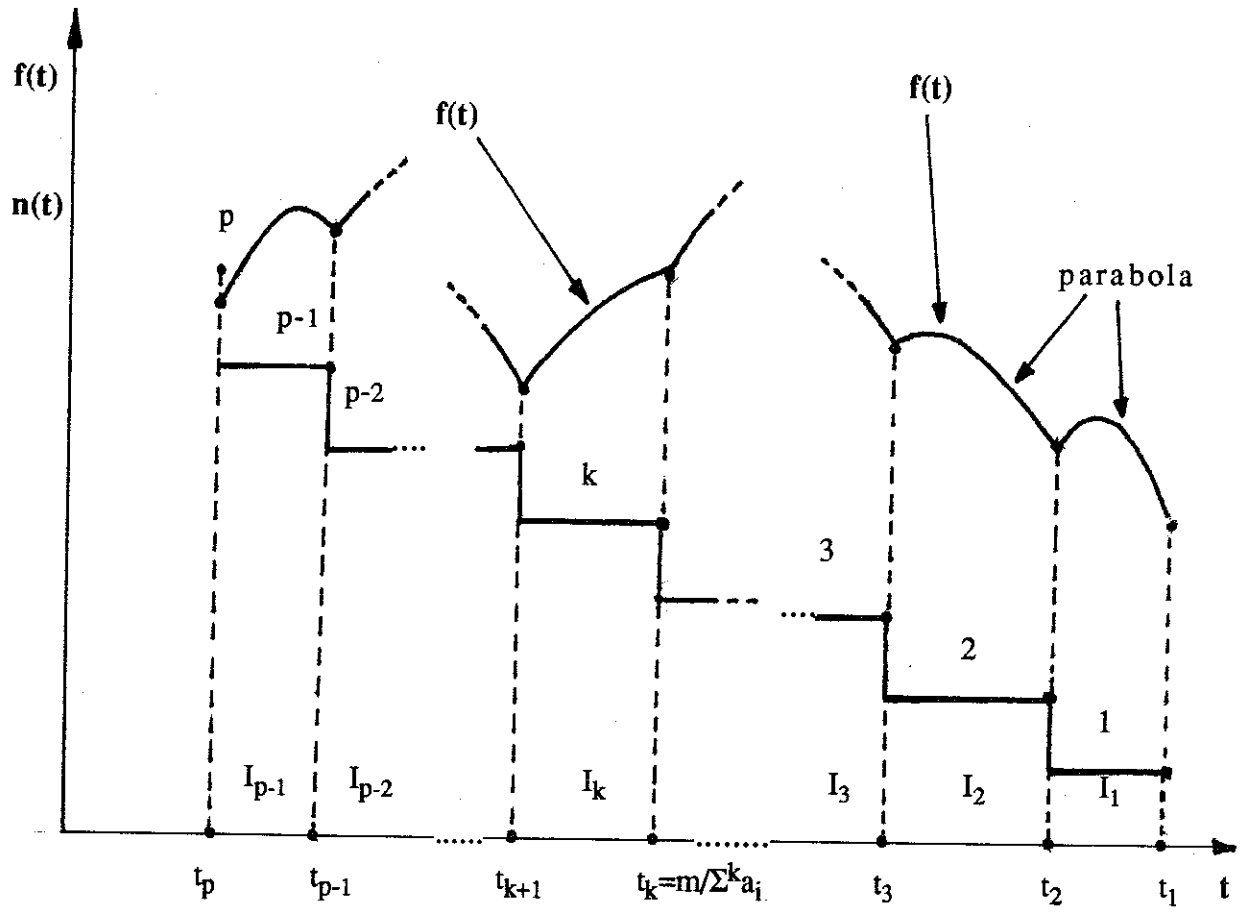

Figure1. Variation of the functions n(t) and f(t) with t

$$c(n(t)) \equiv d_u u_{n(t)+1} m \tag{36}$$

From the definition of n(t) in (31) and the staircase function in Fig.1, we note that the value of n(t) is *piecewise constant* over every interval of t of the form $I_k \equiv (m/\Sigma^{k+1}a_i\ ,\ m/\Sigma^{k}a_i] \equiv (t_{k+1}\ ,\ t_k]$

$$n(t) = k \quad , \quad \text{for all } t \in I_k \equiv (t_{k+1}\ ,\ t_k] \ , \quad t_k \equiv m/\Sigma^{k}a_i \ , \quad k = 1, 2, \ldots, p. \tag{37}$$

Note that the values t_k thus defined represent the values of t at which the function n(t) exhibits a unit-jump discontinuity. Thus, the functions a(n(t)), b(n(t)), and c(n(t)) are also constant over each interval I_k and exhibit a stairecase variation over t.This leads to the interesting conclusion that the function f(t) as expressed in (34), which seemed initially to have a rather complex expression, is in fact a *second degree polynomial* over each of the intervals I_k, as shown in Figure 1. Since n(t) is discontinuous at all t_k, it follows that a(n(t)), b(n(t)), and c(n(t)) are also discontinuous functions of t at the same points. We should emphasize that, despite these discontinuities, the function f(t) in (34) *is continuous* . It so happens that the effects on f(t) of the discontinuities of a(n(t)), b(n(t)), and c(n(t)) at the points t_k *cancel each other*.To verify this fact, evaluate $f^+(t_k)$ and $f^-(t_k)$, representing the values of f(t) as t approaches the point t_k from the left and the right respectively, and verify that $f^+(t_k) = f^-(t_k)$. We formalize this result in the following Lemma, deferring the proof to the Appendix.

Lemma 3: *The function* f(t) *is continuous for all* t *in the domain* $[t_p\ ,\ t_1]$

Recall that the expression for f(t) over each of the intervals I_k is a quadratic polynomial whose graph is an inverted (convex) parabola, as shown in Figure 1. Hence, the minimum of f(t) over the *closed* interval $I'_k \equiv [t_{k+1}\ ,\ t_k]$ must occur at one of its *end points*:

$$\min_{t \in I'_k} f(t) = \min\{f(t_k)\ ,\ f(t_{k+1})\} \tag{38}$$

Finally, the *absolute minimum* of f(t) over the domain $t \in [t_p\ ,\ t_1]$ is found by identifying the minimum of all the interval minima:

$$\min_{t \in [t_p, t_1]} f(t) = \min_k \min_{t \in I'_k} f(t) = \min_k\{f(t_k)\} \quad , \quad k = 1, 2, \ldots, p. \tag{39}$$

Recall from (20) and (24) that min f(t) is the desired minimum value F^* of the function F(x):

$$F^* = \min_{t \in [t_p, t_1]} f(t) = \min_k\{f(t_k)\} \quad , \quad k = 1, 2, \ldots, p. \tag{40}$$

The p values in the set $\{f(t_k)\}$ can be computed from the expression for f(t) in (34)-(36) in a straightforward fashion, and the minimum value among them is the desired minimum of F(x). Let q be a value of k for which $f(t_q)$ is equal to the minimum value:

$$q \in (1, 2, \ldots, p) \text{ such that } f(t_q) = \min_k\{f(t_k)\} \quad , \quad k = 1, 2, \ldots, p. \qquad (41)$$

The corresponding *optimal load distribution* is $\chi(t_q)$ corresponding to $t = t_q$ as expressed in (30):

$$\chi(t) \equiv (a_1 t , a_2 t , \ldots , a_{n(t)} t , m - \Sigma^{n(t)} a_i t , 0 , 0 , \ldots , 0) \quad , \quad \text{any given } t$$

Recall from (37) that $n(t_q) = q$ and $t_q = m/\Sigma^q a_i$, hence

$$n(t_q) = q \ , \quad m - \Sigma^q a_i t_q = m - \Sigma^q a_i (m/\Sigma^q a_i) = m - m = 0$$

and the optimal load distribution is

$$\chi(t_q) \equiv (a_1 t_q , a_2 t_q , \ldots , a_q t_q , 0 , 0 , 0 , \ldots , 0) \qquad (42)$$

If there is more than one value of q satisfying (41), there would be a corresponding optimal distribution as in (42) for each, all of which would give the same minimum value F^*.

We now show that the optimal load distribution $\chi(t_q)$ which minimizes F(x) also minimizes the overall objective function $H(x) = F(x) + d_w w(x)$ by demonstrating that $w(\chi(t_q)) = 0$, which is the absolute minimum value that w(x) can have. Substituting the expression for $\chi(t_q)$ from (42) into the expression for w(x) in (13), and recalling from (14) that $e(x_i) = 1$ for $x_i > 0$ and $e(x_i) = 0$ for $x_i = 0$, we obtain

$$w(\chi(t_q)) = \Sigma e(x_i) w_i [t - (x_i/a_i)] = \Sigma^q w_i [t_q - (a_i t_q / a_i)] = 0 \qquad (43)$$

Thus $\chi(t_q)$ is the desired optimal load distribution which minimizes the objective function H(x).

Note that this optimal distribution allocates the total load of m modules among a *subset of the system processors* consisting of the q *most efficacious processors* in the system, with no load assigned to the remaining p - q processors. Each processor P_i for i = 1, 2, . . . q gets a load $\chi_i = t_q a_i$ that is *directly proportional to its efficacy* a_i. The number q and the proportionality factor t_q are readily found from the relationship $f(t_q) = \min_k\{f(t_k)\}$ with k =1, 2, . . .p as indicated in (41), where $t_k = m/\Sigma^k a_i$ and f(t) is the function defined in (34). All q processors run for exactly the *same execution time* $t_i = \chi_i/a_i = t_q a_i/a_i = t_q$, thus eliminating all idle wait-time. These conclusions are recapitulated and formalized in the following procedure, which relates *all parameters* back to the basic *given parameters* of the system.

4.2 Procedure : Optimal distribution

(1) *Arrange the* p *processors of the system in the nondecreasing order of their efficacy :*

$$a_1 \geq a_2 \geq \cdots \geq a_p \quad \textit{where} \quad a_i = 1/(r_i + \delta\tau_i) \quad , \quad \delta = 2e/m \tag{44}$$

(2) *Determine the integer* $q \leq p$ *and the value* t_q *from the relationship*

$$f(t_q) = \min_k\{f(t_k)\} \quad , \quad k = 1, 2 \ldots p \tag{45}$$

where

$$t_k = m/\Sigma^k a_i \quad , \quad f(t_k) = -a(k)t_k^2 + b(k)t_k + c(k) , \tag{46}$$

$$a(k) = d_c(c/2)\lambda[\Sigma^k a_i^2 + (\Sigma^k a_i)^2] \quad , \quad b(k) = [d_t + d_u\Sigma^k u_i a_i + (d_c c\lambda m - d_u u_{k+1})\Sigma^k a_i] \tag{47}$$

$$c(k) = d_u u_{k+1} m \tag{48}$$

(3) *Determine the optimal load distribution as*

$$\chi(t_q) = (t_q a_1 , t_q a_2 , \ldots , t_q a_q , 0 , 0 , \ldots , 0) \quad , \quad m = \Sigma^q t_q a_i \tag{49}$$

by allocating the entire load to the first q *processors in proportion to their respective efficacies.*

4.3 Integer-valued solution

The load allocations $t_q a_i$, with $\Sigma^q t_q a_i = m$, prescribed by the optimal distribution are not in general integer-valued. If module fragmentation is allowed, then one may wish to use partitioned modules to fit the non-integer values of the optimal load allocations. Otherwise, one can derive an integer-valued solution from among those integer-valued distributions that are "closest" to the optimal $\chi(t_q)$ by rounding some of the values in the set $\{t_q a_i\}$ to their nearest lower or higher integer value. Define $[t_q a_i]$ and $[t_q a_i]'$ as the rounded values of $t_q a_i$:

$$[t_q a_i] \equiv \text{largest integer less than or equal to } t_q a_i \tag{50}$$

$$[t_q a_i]' \equiv \text{smallest integer greater than or equal to } t_q a_i \tag{51}$$

Let R be the set of all possible distributions of the m modules over the q processors such that x_i is equal to the lower or upper *rounded value* of $t_q a_i$

$$R \equiv \{ x = (x_1 , x_2 , \ldots x_q , 0, \ldots 0) : \Sigma^q x_i = m \ , \ x_i = [t_q a_i] \text{ or } x_i = [t_q a_i]' \} \tag{52}$$

We shall choose our integer-valued solution, denoted x_{min}, as that element in the set R which minimizes our objective function H(x) given in (15)

$$\min_{x \in R} H(x) = H^* = H(x_{min}) \quad , \quad x_{min} \in R \tag{53}$$

This requires an exhaustive enumeration of the elements comprising the set R and evaluating H(x)

for each. We can determine the total number of such distributions, |R|, as follows. Let d be the integer equal to the excess of m over the summation of $[t_q a_i]$:

$$d = m - \Sigma^q [t_q a_i] \quad , \quad 0 \le d < q \tag{54}$$

Thus, each distribution $x \in R$ is composed of d components of the form $[t_q a_i]'$ and the remaining q - d components of the form $[t_q a_i]$. Hence, |R| is equal to the number of choices of d elements from a set of q elements (repetition not allowed and order does not matter) :

$$|R| = \binom{q}{d} = q!/d!(q-d)! \tag{55}$$

This number may be quite large, depending on the value of q and d, as shown in the following tabulations for q = 5 and q = 10

q = 5	d :	0	1	2	3	4					
	|R| :	0	5	10	10	5					
q = 10	d :	0	1	2	3	4	5	6	7	8	9
	|R| :	0	10	45	120	210	252	210	120	45	10

In those cases where the number of possible distributions in R happens to be large, an alternative much less expensive procedure can be used. This is a *greedy heuristic* that is based on the essential characteristic of the optimal distribution, namely that each load $t_q a_i$ is directly proportional to the value of a_i. We start by rounding each $t_q a_i$ down to $[t_q a_i]$ and determining the value of d from (54). We want to choose the d processors that have to be given one extra module each. For each processor P_i, define the *gain* g_i which results when its allocation is augmented from $[t_q a_i]$ to $[t_q a_i]+1$ as

$$g_i \equiv \{t_q - [t_q a_i]/a_i\} - \{([t_q a_i]+1)/a_i - t_q\} = 2(t_q - [t_q a_i]/a_i) - 1/a_i \tag{56}$$

where $\{t_q - [t_q a_i]/a_i\}$ and $\{([t_q a_i]+1)/a_i - t_q\}$ measure the deviations of $[t_q a_i]$ and $[t_q a_i]+1$ from true optimality. *Greediness* dictates that we should choose the processors to be augmented as those which have the largest values of g_i. Thus, we arrange $\{g_i\}$ in decreasing order and pick the first d elements in the array:

$$\{ g_{i(1)}, g_{i(2)} \ldots g_{i(k)}, \ldots g_{i(d)}, \ldots g_{i(q)} \} \quad , \quad g_{i(k)} \in \{g_i\} \quad , \quad g_{i(k)} \ge g_{i(k+1)} \tag{57}$$

The desired integer-valued distribution is

$$x_i = [t_q a_i] + 1 \quad \text{for} \quad i \in \{ i(1), i(2), \ldots i(d) \} \quad , \quad x_i = [t_q a_i] \quad \text{otherwise} \tag{58}$$

The time complexity of this procedure is $O(q^2)$ corresponding to the process of sorting the q elements of $\{g_i\}$. We shall illustrate the application of this procedure by a numerical example in Section 4.5 which follows the next section.

4.4 Computational complexity and utilization

The effort needed to determine the optimal load distribution in any given situation is rather minimal, especially when compared to the computationally intensive algorithms and heuristics for optimizing task assignment that one finds in the literature. To use the procedure for a given problem, we simply compute the p values in the set $\{f(t_k)\}$ as indicated in (46), idetntify their minimum value which gives the values of q and t_q as indicated in (45), and allocate the m load modules to the q most efficacious processors as indicated in (49), giving each a load $t_q a_i$ in direct proportion to its efficacy a_i. This simplicity of the technique and the resulting low overhead of its use derives from the fact that we are modeling the workload parameters *in the mean*. As such, the technique must be viewed as an approximate rule-of-thumb criterion of optimality useful in guiding load distribution. The criterion may be used alone or in conjuction with other more precise task assignment techniques which take into account the parameters of the individual load modules. In the latter case, the criterion at hand is used first to determine the *optimal level of loading*, i.e. number of modules, that should be allocated to each processor. The task assignment technique is then employed to specify the individual modules for each processor. The pay-off to be gained from this strategy is the dramatic reduction of overhead resulting from running the task assignment algorithm/heuristic with *fixed* processor load levels. In some cases, this reduction of overhead may be as high as 100%. That would be the case when the application of the procedure results in a value of q = 1, implying an optimal distribution that allocates all the load modules to the most efficacious processor, thus rendering the running of the task assignment algorithm unnecessary.

4.5 Example

The following tabultaion for a numerical example illustrates the procedure for finding the optimal subset of the given p =10 processors to be engaged in the execution of the given workload of m = 55 modules and the apportionment of the total load among them. In the first row, the 10 processors are arranged in the decreasing order of their efficacies a_i. For each value of k from 1 to 10; we compute $t_k = 55/\Sigma^k a_i$ and the corresponding values of $f(t_k)$ as shown in rows 3 and 4. We determine the smallest value of $f(t_k)$, namely 5.3 shown in boldface, which corresponds to k=7. Thus the number of processors to be engaged in execution is q = 7, and the 55 modules are apportioned among the first 7 processors in proportion to their efficacy, with $t_q = t_7 = 1.112$ being the proportionality factor (shown in boldface). The optimal loads $t_q a_i = 1.122 a_i$ are shown in row 5. To find the integer number of modules for each *engaged* processor, we compute the value

a_i :	10	9	8	7	6	5	4	4	3	3
k :	1	2	3	4	5	6	7	8	9	10
$t_k = 55/\Sigma^k a_i$:	5.50	2.89	2.04	1.62	1.37	1.22	**1.122**	1.04	0.98	0.93
$f(t_k)$:	8.5	6.3	9.4	9.2	7.8	8.2	**5.3**	9.6	7.1	8.1
$t_q a_i = 1.122 a_i$:	11.22	10.98	8.98	7.85	6.73	5.61	4.49	0	0	0
g_i :	-.056	-.089	**.119**	**.101**	**.077**	**.044**	-.006			
$[t_q a_i]$ or $[t_q a_i]'$:	11	10	9	8	7	6	4			

of $d = m - \Sigma^q[t_q a_i] = 55 - 51 = 4$, which means we have to round the load of 4 processors up to $[t_q a_i]'$ and round the load of the remaining 3 processors down to $[t_q a_i]$. To make the selection, we compute the values of g_i as shown in row 6, and we select the 4 processors with largest values of g_i (shown in boldface) to be rounded up to $[t_q a_i]'$, as shown in the last row which lists number of modules that should be allocated to each of the engaged processors.

5. Dynamic Load Distribution

The load distribution policy that we have formulated so far may be described as a *static* strategy, to be invoked at the time of compiling or loading the given workload represented by the graph R. Using the given problem parameters p, m, δ, λ, r_i, τ_i, a_i , c, u_i , d_t, d_c, d_u; we were able to determine an *initial* optimal distribution $x^* = (x_1^*, x_2^*, \ldots x_p^*)$ of the m load modules over the p available processors which minimizes the prescribed objective function H(x). If we assume that load execution starts at time t = 0, each processor P_i proceeds for $t > 0$ with the execution of its load of x_i^* modules, progressively terminating them one by one until its load assignment execution is completed. (The reader should distinguish the symbol t, used in this section to denote time as an independent variable, from the symbol t of t(x) used in the previous sections to denote completion time for the load distribution x). from We assume that the processor does not perform context switching among the modules assigned to it, and that a module execution once started is taken to completion with no interruption. During workload execution, the static load distribution policy does not allow any load redistribution of the remaining unexecuted modules at any time $t > 0$. We now describe a *dynamic* load distribution policy which allows such redistribution during workload execution, with the aim of optimizing the remaining computation to be executed after any given time t. By monitoring the state of the *remaining computation* at time t, a decision is made on

whether redistribution is warranted by comparing the *benefit* that would accrue from redistribution to the *cost* of such redistribution, using some appropriately defined measures of benefit and cost.

5.1 The state of remaining computation

Let $x(t) = (x_1(t), x_2(t), \ldots x_p(t))$ represent the load distribution vector at time t, where $x_i(t)$ denotes the number of *unexecuted* modules remaining on processor P_i at time t. Thus, $x_i(0) = x_i^*$, and $x_i(t)$ is a decreasing function of time, with $x_i^* - x_i(t)$ representing the number of executed modules (including the currently executing module) for P_i. Let $m(t) = \Sigma x_i(t)$ denote the total number of unexecuted modules remaining on all processors at time t. Thus $m(0) = m$, and m(t) is a decreasing function of time, with $m - m(t)$ representing the total number of executed modules at time t. Let $R(t) = (M(t), E(t))$ be the graph representing the *remaining computation* at time t, where M(t) is the set of unexecuted load modules and E(t) is the set of unexecuted edges composed of elements E_{kg} such that M_k and M_g are both in M(t). Thus $M(0) = M$, $E(0) = E$ and $M - M(t)$, $E - E(t)$ represent the sets of executed modules and edges respectively, at time t. Note that $E - E(t)$ is the set of all edges in the initial graph R that are connected to the set of modules $M - M(t)$. Thus, we obtain R(t) from R by removing the modules $M - M(t)$ and all edges connected to them. Let $I = \{1, 2, \ldots, j, \ldots m\}$ represent the set of subscripts indexing the modules M_j of M, and let I(t) be the subset of I corresponding to the subscripts of M(t). Note that $|I| = |M| = m$ and $|I(t)| = |M(t)| = m(t)$. Let $|E(t)|$ be denoted by e(t), and let $\delta_j(t)$ denote the degree of M_j in R(t) for every $j \in I(t)$. We can now define the average degree $\delta(t)$ and the workload coupling factor $\lambda(t)$ for the graph R(t) of the remaining computation in a similar fashion to the earlier definitions of δ and λ given in (1) and (2)

$$\delta(t) \equiv \text{Average value of } \delta_j(t) = (\Sigma_{j \in I(t)} \delta_j(t))/m(t) = 2e(t)/m(t) \qquad (60)$$

$$\lambda(t) \equiv \delta(t)/[(m(t) - 1)] = 2e(t)/m(t)(m(t) - 1) \qquad (61)$$

Note that in our identification of "unexecuted" or "remaining" modules for a processor P_i, we have opted not to include the remaining computation of a *partially executed* module that may be currently executing at time t. This is so because the dynamic load distribution policy that we shall adopt will target for redistribution the remaining modules that have not started execution. By deliberately excluding the modules currently executing on their respective processors from possible migration to other processors, we avoid the the overhead cost of *context switching* which tends to offset, or possibly negate, the benefit that might be obtained from such migration. Thus, in our dynamic load redistribution strategy, to be described next, any currently executing module will stay with its processor until completion.

We now represent the other parameters pertaining to the remaining computation R(t) in a

fashion similar to that used in defining the average number of unexecuted communictions per module, $\delta(t)$, and the remaining workload coupling factor, $\lambda(t)$. The following are the relevant parameters:

$R(t) \equiv$ remaining computation as defined above

$p(t) \equiv$ number of system processors available at time t for $R(t)$

$r_i(t) \equiv$ mean computation time spent by processor P_i per module of $R(t)$

$\tau_i(t) \equiv$ mean time spent by processor P_i per intermodule communication of $R(t)$

$\delta'(t) = (\Sigma_{j\epsilon I(t)}\delta_j(0))/m(t) \equiv$ mean number of intermodule communications per module of $R(t)$

$T_i(t) = r_i(t) + \delta'(t)\tau_i(t) =$ mean execution time spent by processor P_i per module of $R(t)$

$a_i(t) \equiv 1/(r_i(t) + \delta'(t)\tau_i(t)) \equiv$ Efficacy of processor P_i with respect to the workload of $R(t)$

Note first that have made the number of processors available for the remaining computation, $p(t)$, a function of t. This number could be less than p if some of the initial set of processors received zero load and were subsequently engaged with other tasks. Or $p(t)$ may be larger than p if some of the system processors were already engaged at the time of intitiating the workload at hand but were freed later. Note also the introduction of the new parameter $\delta'(t)$ and compare it to the parameter $\delta(t)$ previously defined in (60). Whereas $\delta(t)$ represents the average number of *unexecuted* communications per module of $R(t)$, $\delta'(t)$ represents the total number of communications that a module in $R(t)$ has to deal with, which includes reading from memory the communication data received from *previously executed* modules that have been removed from $R(t)$. This is why in the expression for $\delta'(t)$ we use $\delta_i(0)$ rather than $\delta_i(t)$.

We are now ready to present the criterion for the optimal redistribution of the load represented by the remaining computation $R(t)$ at time t. All we have to do is apply the previously established criterion of section 4.2 using the appropriate parameters for $R(t)$ we have defined in this section. We present the result as a procedure in the next section.

5.2 Procedure : Optimal redistribution

(1) *Arrange the* $p(t)$ *processors available at time* t *in the nondecreasing order of their efficacy :*

$$a_1(t) \geq a_2(t) \geq \cdots \geq a_{p(t)}(t) \quad \textit{where} \quad a_i(t) = 1/(r_i(t) + \delta'(t)\tau_i(t)) \ , \ \delta'(t) = (\Sigma_{j \in I(t)} \delta_j(0))/m(t)$$

(2) *Determine the integer* $q(t) \leq p(t)$ *and the value* $t_q(t)$ *from the relationship*

$$f(t_q(t)) = \min_k \{ f(t_k(t)) \} \qquad , \qquad k = 1, 2 \ldots p(t)$$

where

$$t_k(t) = m(t)/\Sigma^k a_i(t) \quad , \quad f(t_k(t)) = -a(t,k)[t_k(t)]^2 + b(t,k)t_k + c(t,k) \ ,$$

$$a(t,k) = d_c(c/2)\lambda(t)[\Sigma^k [a_i(t)]^2 + (\Sigma^k a_i(t))^2] \ ,$$

$$b(t,k) = [d_t + d_u \Sigma^k u_i a_i(t) + (d_c c \lambda(t) m(t) - d_u u_{k+1}) \Sigma^k a_i(t)] \quad , \quad c(t,k) = d_u u_{k+1} m(t)$$

(3) *Determine the optimal load distribution as*

$$\chi(t_q(t)) = (t_q(t)a_1(t) , t_q(t)a_2(t) , \ldots , t_q(t)a_q(t) , 0 , 0 , \ldots , 0) \quad , \quad m(t) = \Sigma^q t_q(t) a_i(t)$$

by allocating the entire load to the first $q(t)$ *processors in proportion to their respective efficacies.*

Note that this dynamic redistribution procedure is identical to the static distribution procedure we derived earlier, with all parameters now reflecting the state of the remaining computation $R(t)$.

5.2 Redistribution policy

For any given t with a current distribution $x(t) = (x_1(t), x_2(t), \ldots x_p(t))$ and $\Sigma_i x_i(t) = m(t)$, the remaining $m(t)$ unexecuted modules can be redistributed using the above dynamic allocation procedure to obtain a new optimal distribution $x^*(t) = (x_1^*(t), x_2^*(t) \ldots x_p^*(t))$ with $\Sigma_i x_i^*(t) = m(t)$, which minimizes the objective function $H(x(t))$. The redistribution from $x(t)$ to $x^*(t)$ involves transferring a certain number of modules and their data from their current processors to other processors. This redistribution can be justified only if its *cost* does not exceed its *benefit*. We now define metrics to assess the redistribution cost and benefit, and use them to formulate a simple policy for deciding whether to go ahead with the module migration at any given time t. Let $n(t)$ denote the number of modules that must be transferred to affect the redistribution from $x(t)$ to $x^*(t)$:

$$n(t) \equiv \text{number of modules to be redistributed} = (1/2)\Sigma_i |x_i(t) - x_i^*(t)|$$

The expression for $n(t)$ is a direct consequence of the fact that $\Sigma_i (x_i(t) - x_i^*(t)) = m(t) - m(t) = 0$. We shall express the cost of redistribution as the sum of the cost of transferring the $n(t)$ modules plus

the cost of transferring the *communication data already received* by each module from previously executed modules:

$$\text{Cost of transferring n(t) module} \equiv C_{mod}(t) = k_1 n(t)$$

$$\text{Cost of transferring communication data} \equiv C_{data}(t) = k_2 n(t)[\delta'(t) - \delta(t)]$$

where k_1 and k_2 are proportionality/scale factors, and $\delta'(t) - \delta(t)$ is the average number of communication data received by a module in R(t) from already executed modules. Thus

$$\text{Total cost of redistribution} \equiv C(t) \equiv C_{mod}(t) + C_{data}(t) = k_1 n(t) + k_2 n(t)[\delta'(t) - \delta(t)]$$

We shall measure the benefit of redistribution by the decrease in the value of the objective function as a result of going from the current distribution x(t) to the optimal distribution $x^*(t)$:

$$\text{Benefit of redistibution} \equiv \Delta H(t) \equiv H(x(t)) - H(x^*(t))$$

The value of the benefit at time t can be computed from the expression for H(x) in (15). Thus, redistribution is carried out only if the benefit outweighs the cost , i.e. if

$$\Delta H(t) > KC(t)$$

where K is a proportionality/scale factor chosen in some appropriate manner.

5.3 Frequency of redistribution and optimality

An important issue in the implementation of the dynamic distribution policy is the question of how often should load redistribution be invoked. The procedure has been formulated such that it can be invoked at any time t during workload execution. Assuming the cost of running the procedure is not an issue, the utmost performance from employing this dynamic distribution strategy is attained if the procedure is invoked *greedily,* i.e.whenever a processor completes the execution of one module and before it begins the execution of another, at which time the value of m(t) decrements by 1, and a new state of R(t) emerges with new values for the various applicable parameters. Another event that warrants invoking the procedure arises when the set of available processors is augmented due to the freeing of one or more processors that were unavailable at the time of the last redistribution. Each application of the procedure represents , in effect, an *updating of optimality* for the remaining computation.

What is highly interesting about this procedure is that this adjustment for optimality goes beyond the updating of parameters in response to a change in the value of m(t) or a change in the set of available processors. This arises from the fact that the dynamic procedure tends, with each application, to correct for its inherent inaccuracies and the inaccuracies of the model on which it is based. Recall that there are three basic sources of inaccuracy underlying the dynamic load distribution strategy we have formulated: Using *average* workload parameters, ignoring

synchronization delays in modeling time execution, and excluding *partially-executed* modules from redistribution. The procedure tends to offset the deviation from optimality arising from these effects. To see why this is so, recall that the optimality criterion allocates to each processor a load proportional to its efficacy, such that all processors will have the *same value of execution time* completion : $T_i = t(x)$ for all i. Consider what might take place between invoking the procedure at time t_1 and subsequently at time t_2. If the *specific* modules allocated to processor P_i at time t_1 happen to be, say, larger than the average-parameter module, then P_i will *fall behind* in execution time and will have at time t_2 a larger number of unexecuted modules $x_i(t_2)$ than it would otherwise. Let $x_i^*(t_2)$ be the new optimal allocation. If $x_i^*(t_2) > x_i(t_2)$ then $x_i^*(t_2) - x_i(t_2)$ modules would be added to P_i, which is *less* than it would get otherwise. If $x_i^*(t_2) < x_i(t_2)$ then $x_i(t_2) - x_i^*(t_2)$ modules would be removed from P_i, which is *more* than it would loose otherwise. Thus, in every case the larger than average loading of P_i at time t_1 tends to be corrected at time t_2. The same arguments apply if P_i falls behind because it experiences synchronization delays or because it had at time t_1 a partially executed module that was not accounted for by the redistribution procedure. Thus repeated application of the procedure improves the accuracy of its performance by reducing the deviation from true optimality resulting from ignoring module specificity, synchronization delay, and partially-executed modules. This would especially true if the procedure is applied greedily as described above.

5.4 Implementation issues

One important issue in the implementation of the dynamic load distribution procedure is the task of maintaining current updated values for the parameters of the remaining workload R(t), such as $\delta(t)$, $r_i(t)$, and $\tau_i(t)$. To illustrate how this might be done *recursively*, assume that the redistribution algorithm is invoked every time a processor completes the execution of a module. Assume that the last redistribution occured at time t_1 when the remaining computation was represented by the graph $R(t_1)$ consisting of the set of unexecuted modules corresponding to the index set $I(t_1)$ whose cardinality is $|I(t_1)| = m(t_1)$. Assume that the next redistribution is invoked at time t_2 when the execution of module M_j is just completed.The updated value $r_i(t_2)$ can be computed from the *previous* value $r_i(t_1)$ by the following simple recursive relationship:

$$r_i(t_2) = [m(t_1)r_i(t_1) - r_ij]/[m(t_1) - 1]$$

where r_ij is the computation time of module M_j on processor P_i. The derivation of the above relationship is as follows:

$$r_i(t_2) = \Sigma_{k\varepsilon I(t2)}\, r_i^k/m(t_2) = [(\Sigma_{k\varepsilon I(t1)}r_i^k) - r_ij]/[m(t_1) - 1] = (\Sigma_{k\varepsilon I(t1)}r_i^k)/[m(t_1) - 1] - r_ij/[m(t_1) - 1]$$

$= \{m(t_1)/[m(t_1) - 1]\}(\Sigma_{k\epsilon I(t1)} r_i^k)/[m(t_1) - r_i j/[m(t_1) - 1] = \{m(t_1)/[m(t_1)-1]\} r_i(t_1) - r_i j/[m(t_1)-1]$

$= [m(t_1)r_i(t_1) - r_i j]/[m(t_1) - 1]$

The above relationship implies that if $r_i j = r_i(t_1)$ then $r_i(t_2) = r_i(t_1)$. In other words, if the computation time of the completed module is approximately equal to the current average, the updated average will be approximately equal to the preceeding value. Similar recursive relationships are applicable to $\delta(t)$ and $\tau_i(t)$.

6. Conclusions and Recommendations

We have developed in this paper simple criteria for optimizing the distribution of coupled load modules over heterogeneous processing elements. A simple computational model of the workload, with average values of the relevant parameters, enabled us to minimize a weighted combination of four metrics of performance and cost: workload completion time, communication cost, resource utilization cost, and idle-time cost. The weight coefficients can be used to "tune" the decision model to the specific objectives of a particular system/workload situation. A straightforward analytical criterion was derived which enables the determination of the optimal load distribution according to the numerical values of the problem parameters. The optimal load distribution was found to be the apportionment of the total load among the $q \leq p$ most efficacious processors in direct proportion to their efficacies. The efficacy of a processing element was defined as a composite measure of its cpu speed, memory speed, and the degree of coupling among the modules of the given workload. The criterion provides a straightforward procedure for determining the number q of processors to be thus engaged in executing the given workload. In the absence of synchronization delays, the optimal distribution results in equal execution time for all engaged processors, thus eliminating idle wait-time and therefore representing the ideal load "balancing" on the heterogeneous system. The p - q least efficacious processors remain unengaged in workload execution and can be allocated to other jobs.

Since average values were used in the analysis, the criterion should be regarded as an *approximate* rule-of-thumb policy for guiding load distribution.The policy may be implemented on its own; or as a first step in a two-level optimization procedure employing an adaptation of any of the known task-specific assignment algorithms, with a resulting substantial savings in the overhead cost of such algorithms running on their own.

Another source of inaccuracy in the results stems from ignoring the effects of synchronization delays experienced by processors while waiting for interprocessor communications. Since synchronization delay creates the potential of prolonging the load completion time, it follows that

the estimate of workload completion time used in our objective function might actually be an underestimate of the desired metric.

Exact continuous-variable analysis was employed in the derivation of the closed-form criterion which therefore provides the absolute optimal distribution comprising non-integer-valued module allocations. If load module fragmentation for fitting the ideal distribution is not possible, integer-valued allocations can be derived by a straightforward and computationally efficient algorithm.

The optimal load distribution procedure can be applied dynamically at any time during workload execution using the appropriate parameters pertaining to the remaining workload computation. The procedure can be greedily invoked at the time of individual module completions or whenever the set of available system processors changes. Updated values of the needed parameters can be efficiently computed in a recursive fashion from one invokation to the next. Decision to carry out the prescribed redistribution is governed by whether its cost is outweighed by its benefit as measured by the would-be reduction in the decision objective function value. Although the analytical procedure can handle non-integer load allocations, it is suggested that pre-emption of partially executed modules be excluded from redistribution because of the high overhead of context switching it might entail. This would represent a third source of deviation from true optimality, in addition to using average load parameters and ignoring synchronization delay. Nonetheless, repeated applications of the dynamic distribution algorithm during workload run-time tends to persistently and adaptively correct the deviations from optimality resulting from these inaccuracies.

Further research is needed to extend the approach and the results of this paper to heterogeneous systems where the processor loading levels are *constrained* by ceilings dictated by limitations of physical resources, such as memory capacity. In this case the, the minimization of the objective function has to be performed under the additional constraints $x_i \leq L_i$, where L_i is the upper bound limiting the loading of processor P_i. This extention has been accomplished in [23] for*static* load distribution and using a simplified objective function consisting only of the workload completion time. Further effort is needed to bring this extension to bear on *dynamic* load distribution and the *multimetric* optimization decision model formulated in this paper.

7. Appendix

7.1 Lemma 1: *If* $f(t^*) = \min_t f(t)$, *then* $t^* \varepsilon [t_p, t_1]$ *where* $t_1 \equiv m/a_1$, $t_p \equiv m/(a_1+a_2+\cdots+a_p)$

Proof: we first show that for *any* value of t, we must have $t \geq t_p$. Assume, to the contrary, that $t < t_p$. Then we must have

$$t(x) = \max_i\{x_i/a_i\} < t_p \ , \ x_i/a_i < t_p \ \text{ for all i } \ , \ x_i < a_i t_p \ \text{ all i } \ , \ \Sigma x_i < \Sigma a_i t_p \ , \ m < t_p \Sigma a_i$$

which is a contradiction to the definition of $t_p = m/\Sigma a_i$. Hence we must have $t \geq t_p$. We next show that for and $t = t' > t_1$ we must have $f(t') > f(t_1)$ and therefore t' cannot be a minimum point of f(t). It follows from the definition of f(t) in (25) that, to have $f(t') > f(t_1)$, we must have

$$d_t t' + \min_{x \varepsilon D(t')} G(x) \ > \ d_t t_1 + \min_{x \varepsilon D(t1)} G(x)$$

Since $t' > t_1$, the above inequality would be true if can prove that

$$\min_{x \varepsilon D(t')} G(x) \ \geq \ \min_{x \varepsilon D(t1)} G(x) \qquad \text{(A0)}$$

Now consider three different distributions: a *specific* $x^1 \varepsilon D(t_1)$, *any* $x' \varepsilon D(t')$, and *any* $x'' \varepsilon D(t_1)$ defined as follows:

$$x^1 \ : \quad x^1_1 = m \ , \ x^1_i = 0 \ \text{ for } i \neq 1 \ , \ t(x^1) = \max_i\{x^1_i/a_i\} = m/a_1 = t_1$$

$$x' \ : \quad \Sigma x'_i = m \ , \ \ t(x') = \max_i\{x'_i/a_i\} = t'$$

$$x'' \ : \quad \Sigma x''_i = m \ , \ \ t(x'') = \max_i\{x''_i/a_i\} = t_1$$

Evaluate the difference $G(x') - G(x^1)$ using the expression for $G(x) = d_u \Sigma u_i x_i - A\Sigma x_i^2$ from (19):

$$G(x') - G(x^1) = d_u \Sigma u_i x'_i - A\Sigma x'^2_i - d_u u_1 m + Am^2 = d_u \Sigma (u_i - u_1) x'_i + A[(\Sigma x'_i)^2 - \Sigma x'^2_i] \geq 0$$

The last inequality follows from $u_i - u_1 \geq 0$, as implied by (12), and $(\Sigma x'_i)^2 - \Sigma x'^2_i \geq 0$ which is always true for a set $\{x'_i\}$ of non-negative numbers. Repeating the same for $G(x'') - G(x^1)$, we get

$$G(x'') - G(x^1) = d_u \Sigma u_i x''_i - A\Sigma x''^2_i - d_u u_1 m + Am^2 = d_u \Sigma (u_i - u_1) x''_i + A[(\Sigma x''_i)^2 - \Sigma x''^2_i] \geq 0$$

Thus we have $G(x'') \geq G(x^1)$ for all $x'' \varepsilon D(t_1)$, hence $G(x^1) = \min_{x \varepsilon D(t1)} G(x)$. Also we have $G(x') \geq G(x^1)$ for all $x' \varepsilon D(t')$, hence $\min_{x \varepsilon D(t')} G(x) \geq G(x^1)$. The relationship in (A0) follows from these two conclusions, and the proof of the lemma is complete.

7.2 Lemma 2 : *For any given value of t , the value of* g(t) *is expressed as*

$$g(t) \equiv \min_{x \varepsilon D(t)} G(x) = G(\chi(t)) = d_u \Sigma u_i \chi_{i(t)} - A\Sigma (\chi_{i(t)})^2 \qquad \text{(A1)}$$

where

$$\chi(t) \equiv (a_1 t, a_2 t, \ldots, a_{n(t)} t, m - \Sigma^{n(t)} a_i t, 0, 0 \ldots, 0) \quad (A2)$$

$$n(t) \equiv \max \{ j : m - \Sigma^{j} a_i t \geq 0 \} \quad (A3)$$

and therefore the explicit expression of $g(t)$ *in terms of* t *is*

$$g(t) = d_u \Sigma^{n(t)} u_i a_i t - A \Sigma^{n(t)} (a_i t)^2 + d_u u_{n(t)+1} (m - \Sigma^{n(t)} a_i t) - A (m - \Sigma^{n(t)} a_i t)^2 \quad (A4)$$

Proof: Recall the definition of D(t) from (22)

$$D(t) \equiv \{ x : 0 \leq x_i \leq a_i t , \Sigma x_i = m , x_s = a_s t \text{ for some } s = i \} \quad (A5)$$

Note that the distribution $\chi(t)$ satisfies all the conditions for D(t) in (A5), and therefore $\chi(t) \in D(t)$. Note also that distribution $\chi(t)$ in (A2) is characterized by two properties, denoted P1 and P2:

P1 : *The elements* $\chi_i(t)$ *of* $\chi(t)$ *are arranged in nonincreasing order, (because* $a_1 \geq a_2 \geq \cdots \geq a_p$).

P2: *For every element* $\chi_i(t \neq 0$, *the preceding element* $\chi_{i-1}(t)$ *is equal to its upper bound* $a_{i-1} t$.

These two properties uniquely identify $\chi(t)$. No other $x \neq \chi(t)$ satisfies P1 and P2 simultaneously. We present a proof by contradiction. Assume, contrary to the assertion of the lemma, that the minimum of G(x) does not occur for $\chi(t)$ but occurs for some other distribution $x' \neq \chi(t)$:

$$G^* = \min_{x \in D(t)} G(x) = G(x') < G(\chi(t)) , \quad x' \in D(t) , \quad x' \neq \chi(t) \quad (A6)$$

There are two mutually exclusive cases for x', denoted by C1 and C2:

C1 : The elements of x' are arranged in non-increasing order, viz. $x'_1 \geq x'_2 \ldots \geq x'_p$.

C2 : The elements of x' are not arranged in non-increasing order.

We show that either case will lead to a contradiction. Consider case C1 first. Since x' satisfies P1 and $x' \neq \chi(t)$, it follows that x' does not satisfy P2.This implies that there is at least one nonzero element x'_j whose preceding element x'_{j-1} is less than its upper bound $a_{j-1} t$:

$$0 < x'_j \leq x'_{j-1} < a_{j-1} t \quad (A7)$$

Define the number d as

$$d \equiv \min \{ x'_j , a_{j-1} t - x'_{j-1} \} > 0 \quad (A8)$$

Consider a new distribution x" derived from x' by subtracting d from x'_j and adding d to x'_{j-1}, while keeping all other elements unchanged

$$x'' = (x'_1, x'_2, \ldots, x'_{j-1} + d , x'_j - d, \ldots, x'_p) \quad (A9)$$

We now show that $G(x'') < G(x')$, which is a *contradiction* to the starting assumption of G(x') being the minimum value of G(x). But first we have to have check if $x'' \in D(t)$ by verifying that x" satisfies all the conditions in (A5). Note that $\Sigma x''_i = \Sigma x'_i = m$ and (A8) implies that

$$x''_{j-1} = x'_{j-1} + d \leq a_{j-1} t \quad , \quad x''_j = x'_j - d \geq 0 \quad (A10)$$

Thus x" satisfies all the conditions in (A5) except perhaps the requirement $x"_s = a_s t$ for some $s = i$. The only situation where this condition would be violated is when x'_j is the *only element* in x' such that $x'_j = a_j t$, hence $x"_j = x'_j - d < a_j t$, *and* if $x"_{j-1} = x'_{j-1} + d < a_{j-1} t$. This would mean $x"_i < a_j t$ for all i and x" is not in D(t). In this happens to be the case, we abandon x'_j and focus our attention on x'_{j-1} instead.which has same needed property as x'_j ,namely, a nonzero element whose preceding element is less than its upper bound $x'_{j-2} < a_{j-2} t$ (since in this case x'_j is the *only element* equal to its upper bound), and in addition we have $x'_{j-1} < a_{j-1} t$ as indicated in (A7).

Recall the expression of G(x) from (19):

$$G(x) = d_u \Sigma u_i x_i \; - \; A \Sigma x_i^2 \qquad \text{(A11)}$$

$$\begin{aligned} G(x') \; - G(x") &= d_u (u_j - u_{j-1}) d \; + \; A\{(x'_{j-1} + d)^2 + (x'_j - d)^2 \; - (x'_{j-1})^2 \; - \; (x'_j)^2 \} \\ &= d_u (u_j - u_{j-1}) d \; + \; 2Ad(x'_{j-1} \; - x'_j) + \; 2Ad^2 \; > \; 0 \end{aligned}$$

The positiveness of the last expression follows from the relationships $u_j - u_{j-1} \geq 0$, $x'_{j-1} - x'_j \geq 0$, and $d > 0$, as indicated in (8) ,(A7), and (A9) respectively.

Consider case C2. Since the elements of x' are not arranged in nonincreasing order, we can identify a pair of elements x'_j and x'_k such that

$$x' \; = \; (x'_1, x'_2, \ldots, x'_j, \ldots, x'_k, \ldots, x'_p) \qquad , \; x'_k > x'_j \; , \qquad k > j \qquad \text{(A12)}$$

Consider a new distribution x^1 derived from x' by swapping the magnitudes of x'_j and x'_k :

$$x^1 \; = \; (x'_1, x'_2, \ldots, x^1_j = x'_k, \ldots, x^1_k = x'_j, \ldots, x'_p) \qquad \text{(A13)}$$

Note that $x^1 \in D(t)$ since $\Sigma x^1_i = \Sigma x'_i = m$ and $x^1_j = x'_k \leq a_k t \leq a_j t$, $x^1_k = x'_j < x'_k \leq a_k t$ as implied by (8) and (A12). We now show that $G(x^1) \leq G(x')$:

$$G(x') \; - G(x^1) \; = \; u_j (x'_j - x'_k) + u_k (x'_k - x'_j) \; = \; (u_k - u_j)(x'_k - x'_j) \; \geq \; 0$$

where the non-negativeness of the last expression follows from $u_k \geq u_j$ and $x'_k > x'_j$. If $G(x') - G(x^1) > 0$, we have a contradiction to G(x') being the minimum of G(x), and the proof would be complete. On the other hand, if $G(x^1) = G(x')$, we repeat the *same steps* applied now to x^1 instead of x', provided that x^1 is still not arranged in nondecreasing order despite the swapping which previously ordered x'_k and x'_j in decreasing order. By swapping two elements in x^1 similar to what we did in (A12) and (A13), we get a new distribution x^2 leading again to a contradiction or to the conclusion that $G(x^2) = G(x^1) = G(x')$. We re-iterate the procedure $s \leq p$ times, which terminates, either with a contradiction or with a distribution fully arranged in nondecreasing order, whichever comes first. In the later case, we have

$$G(x^s) = G(x^{s-1}) = \cdots = G(x^2) = G(x^1) = G(x') = \min_{x \in D(t)} G(x)$$

Thus , x^s is a distribution arranged in nondecreasing order which minimizes G(x). But this also leads to a contradiction, as we have already shown in case C1 above. This completes the proof.

7.3 Lemma 3 : *The function* f(t) *is continuous for all* t *in the domain* $[t_p, t_1]$

Proof: We focus on the points $t_k = m/\Sigma^k a_i$, where the coefficients of f(t) are discontinuous, and verify that $f^+(t_k) = f^-(t_k)$. Recall that $n(t_k) = k$, and the expression for f(t) from (34)-(36):

$$f(t) = - a(n(t))t^2 + b(n(t))t + c(n(t))$$

$$a(n(t)) \equiv A[\Sigma^{n(t)}a_i^2 + (\Sigma^{n(t)}a_i)^2] \ , \ b(n(t)) \equiv [d_t + d_u\Sigma^{n(t)}u_i a_i + (2mA - d_u u_{n(t)+1})\Sigma^{n(t)}a_i]$$

$$c(n(t)) \equiv d_u u_{n(t)+1} m$$

$$f^+(t_k) - f^-(t_k) = [-a(k) + a(k-1)]t_k^2 + [b(k) - b(k-1)]\, t_k + c(k) - c(k-1)$$

$$= - t_k\{A[\, a_k^2 + (\Sigma^k a_i)^2 - (\Sigma^{k-1}a_i)^2]t_k - 2mAa_k + d_u(u_{k+1} - u_k)\, \Sigma^k a_i \} + d_u(u_{k+1} - u_k)m$$

Factoring the difference of the two squares and substituting $t_k = m/\Sigma^k a_i$:

$$(\Sigma^k a_i)^2 - (\Sigma^{k-1}a_i)^2 = a_k(\Sigma^k a_i + \Sigma^{k-1}a_i) = a_k(2\Sigma^k a_i - a_k) = 2a_k\Sigma^k a_i - a_k^2$$

$$f^+(t_k) - f^-(t_k) = -A(m/\Sigma^k a_i)\{[2a_k\Sigma^k a_i]m/\Sigma^k a_i - 2ma_k\} = -A(m/\Sigma^k a_i)\{2ma_k - 2ma_k\} = 0$$

Thus, f(t) is continuous at t_k .

8. References

[1] H.S. Stone, "Multiprocessor Scheduling with Aid of Network Flow Algorithms," *IEEE Trans. Software Eng.*, vol. SE-3, January 1977, pp.85-93

[2] H.S. Stone, "Critical Load Factors in Two-processor Distributed Systems," *IEEE Trans. Software Eng.*, vol. SE-4, May 1978, pp. 254-258.

[3] G.S. Rao, H.S. Stone, and T.C. Hu, "Assignment of Tasks in a Distributed Processor System with Limited Memory," *IEEE Trans. Computers*, vol. C-28, April 1979, pp. 291-299.

[4] V.M. Lo, "Heuristic Algorithms for Task Assignment in Distributed Systems," *IEEE Trans. on Computers*, vol. 37, no. 11, November 1988.

[5] S. Bokhari, "Partitioning Problems in Parallel Pipelined and Distributed Computing," *IEEEE Trans. Computers*, vol. 37, no. 1, Jan. 1988. .

[6] S. Bokhari, *Assignment Problems in Parallel and Distributed Computing*, Kluwer Academic Publishers, 1987.

[7] B. Indukhya, H.S. Stone, and L. Xi-Cheng, "Optimal Partitioning of Randomly Generated Distributed Programs," *IEEEE Trans Software Eng.*, vol. SE-12, March 1986, pp. 483-495.

[8] H.S. Stone, *High Performance Computer Architecture*, 2d ed., Addison-Wesley, 1990, pp.

309-25.

[9] D.M. Nicol, "Optimal Partitioning of Random Programs Across Two Processors," *IEEE Trans. on Software Eng,* Vol. 15, no. 2, February 1989.

[10] E.K. Haddad, "Optimal Partitioning of Random Workloads in Homogeneous Multiprocessor and Distributed Systems," *Proc of the Second IEEE Symposium on Parallel and Distributed Processing,* Dallas, Texas, Dec. 1990.

[11] E.K. Haddad, "Partitioned Load Allocation for Minimum Parallel Processing Execution Time," *Proc of the 19th International Conference on Parallel Processing,* St. Charles, Illinois, August 1989.

[12] E.K. Haddad, "A Criterion for the Optimal Assignment of Program Modules in Parallel and Distributed Systems. *International Journal Of Mini & Micro Computers,* Special Issue on Distributed and Parallel Computing, vol. 12 no. 3 , 1990.

[13] E.K. Haddad, "Optimizing the Parallel Execution Time of Homogeneous Random Workloads" *Proceedings of 21st International Conference on Parallel Processing,"* St. Charles, Illinois, August 1991.

[14] E.K. Haddad, "Variation of Parallel Processing Time with Continuously Partitioned Load Allocation." *Proc of Fourth SIAM Conference on Parallel Processing for Scientific Computing,* Chicago,11-13 December 1989.

[15] T.C. Chou and J.A. Abraham, "Load Balancing in Distributed Systems, *IEEE Trans. on Software Engineering,* vol. SE-35, 1982, pp. 401-12.

[16] J.L. Gaudiot and J.I.Pi, "Program Graph Allocation in Distributed Multicomputers," *Parallel Computing* , Northland, vol. 27, 1988, pp.227-47.

[17] K. Hwang and J. Xu, "Efficient Allocation of Partitioned Program Modules in a Message-Passing Multicomputer," *Proceedings of the ISMM International Conference on Parallel and Distributed Computing and Systems,* New York, 10-12 October 1990.

[18] J.Xu and K. Hwang, "A Simulated Annealing Method for Mapping Production Systems onto Multicomputers," *Proc. of Sixth IEEE Conference on Aritficial Intelligence Applications,* March 1990, pp. 130-136.

[19] A.N. Tantawi and D. Towsley, "Optimal Static Load Balancing in Distributed Computer System," *Journal of the ACM,* vol. 32, 1985.

[20] K.W. Ross and D.D. Yao, "Optimal Load Balancing and Scheduling in a Distributed Computer System" *Journal of the ACM,* Vol.38,No. 3, July 1991.

[21] W.W. Chu, L.J. Holloway, M.T. Lan and Kemal Efe, "Task allocation in distributed data processing," *IEEE Computer,* pp. 57-69, November 1980.

[22] V. M. Lo ,"Task assignment in distributed systems," Ph.D. Dissertation, Dep. Comput. Sci., Univ. Illinois, Oct. 1983.

[23] E.K. Haddad, "Optimal Distribution of Random Workloads over Heterogeneous Processors with Loading Constraints" *Proc of the 1992 International Conference on Parallel Processing,* St. Charles, Illinois, August 1992.