N90-29071

Efficient Mapping Algorithms for Scheduling Robot Inverse Dynamics Computation on a Multiprocessor System

C. S. G. Lee

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

C. L. Chen

Department of Electrical Engineering
Purdue University
Indianapolis, Indiana 46205

ABSTRACT

This paper presents two efficient mapping algorithms for scheduling the robot inverse dynamics computation consisting of m computational modules with precedence relationship to be executed on a multiprocessor system consisting of p identical homogeneous processors with processor and communication costs to achieve minimum computation time. An objective function is defined in terms of the sum of the processor finishing time and the interprocessor communication time. The minimax optimization is performed on the objective function to obtain the best mapping. This mapping problem can be formulated as a combination of the graph partitioning and the scheduling problems, both have been known to be *NP*-complete. Thus, to speed up the searching for a solution, two heuristic algorithms were proposed to obtain fast but suboptimal mapping solutions. The first algorithm utilizes the level and the communication intensity of the task modules to construct an ordered priority list of ready modules and the module assignment is performed by a weighted bipartite matching algorithm. For a near-optimal mapping solution, the problem can be solved by the heuristic algorithm with simulated annealing. These proposed optimization algorithms can solve various large-scale problems within a reasonable time. Computer simulations were performed to evaluate and verify the performance and the validity of the proposed mapping algorithms. Finally, experiments for computing the inverse dynamics of a six-jointed PUMA-like manipulator based on the Newton-Euler dynamic equations were implemented on an NCUBE/ten hypercube computer to verify the proposed mapping algorithms. Computer simulation and experimental results are compared and discussed.

1. Introduction

Robot manipulators are highly nonlinear systems, and their motion control involves the computation of the required generalized forces/torques from an appropriate manipulator dynamics model. There are a number of ways to compute the generalized forces/torques among which the computation of joint torques from the Newton-Euler (NE) equations of motion [1,2] is the most efficient and has been shown to possess the time lower bound of $O(n)$ running in uniprocessor computers [3], where n is the number of degrees-of-freedom of the manipulator. It is unlikely that further substantial improvements in computational efficiency can be achieved. Nevertheless, some improvements could be achieved by taking advantage of particular computation structures [4], customized algorithms/architectures for specific manipulators [5,6], parallel computations [3,7], and scheduling algorithms for multiprocessor systems [8-11].

This paper presents two efficient mapping algorithms for scheduling the robot inverse dynamics computation of an n-jointed manipulator to be executed on a multiprocessor system with processor finishing time and interprocessor communication time considered in the system. The NE equations of motion are decomposed into m computational modules which are scheduled to be executed on p identical homogeneous processors to achieve minimum computation time. Several approaches to the general mapping problem have been proposed [13-18]. In this paper, an objective function is defined in terms of the sum of the processor finishing time and the interprocessor communication time. The minimax optimization of the objective function is performed to obtain the best mapping. This mapping problem is formulated as a combination of the graph partitioning and the scheduling problems; both have been known to be *NP*-complete. Thus, we first propose an efficient heuristic algorithm to obtain a fast but suboptimal solution. The heuristic algorithm utilizes the level and the communication intensity of the task modules to construct an ordered priority list of ready modules, and the module assignment is performed by a weighted bipartite matching algorithm. For a near-optimal mapping solution, the problem can be solved by a simulated annealing method with an efficient lower bound which indicates the minimum-finishing-time of the special scheduling problem. The simulated annealing

This work was supported in part by the National Science Foundation under Grant CDR 88-03017 to the Engineering Research Center for Intelligent Manufacturing Systems.

algorithm is a statistical optimization method which can solve various large-scale combinatorial optimization problems within a reasonable time. This approach transforms the mapping problem into a combined graph partitioning and scheduling problem. A partition of the task graph is first performed, and then the modules in each "block" of the partition are scheduled to be executed by the processor assigned to that block of the partition. Computer simulations and experimental results for computing the inverse dynamics of a six-jointed PUMA manipulator on an NCUBE/ten hypercube computer are compared and discussed.

2. Problem Formulation and Objective Function

The problem of computing manipulator joint torques based on a manipulator dynamic model is often referred to as the *inverse dynamics problem* and can be stated as: Given the joint positions and velocities $\{q_j(t), \dot{q}_j(t)\}_{j=1}^{n}$ which describe the state of an n-jointed manipulator at time t, together with the joint accelerations $\{\ddot{q}_j(t)\}_{j=1}^{n}$ which are desired at that time, solve the dynamic equations of motion for the joint torques $\{\tau_j(t)\}_{j=1}^{n}$ as follow:

$$\tau(t) = \mathbf{f}(\mathbf{q}(t), \dot{\mathbf{q}}(t), \ddot{\mathbf{q}}(t)) \tag{1}$$

where $\tau(t) = (\tau_1, \tau_2, \cdots, \tau_n)^T$, $\mathbf{q}(t) = (q_1, q_2, \cdots, q_n)^T$, $\dot{\mathbf{q}}(t) = (\dot{q}_1, \dot{q}_2, \cdots, \dot{q}_n)^T$, $\ddot{\mathbf{q}}(t) = (\ddot{q}_1, \ddot{q}_2, \cdots, \ddot{q}_n)^T$, the superscript T denotes transpose operation on vectors and matrices, and Eq. (1) indicates the functional representation of the manipulator dynamic model. Since the NE equations of motion have been known for their efficiency in computing the joint torques, our objective is to see how fast one can map the computation of the NE equations of motion on a multiprocessor system with p identical processors to achieve minimum computation time.

2.1. Representation of System Model

In general, a computational task can be represented by a directed acyclic task graph (DATG) $G_c = (V_c, E_c)$ consisting of a finite nonempty set of vertices V_c, $V_c = \{T_k \mid T_k \in G_c, k = 1, 2, \cdots, m\}$, and a set of finite edges, E_c, $E_c = \{e_{ij}^c \mid e_{ij}^c \in G_c\}$, connecting them. Each vertex represents a computational module (CM), and each edge represents the precedence constraint between two CMs. An edge connecting module T_i to module T_j is denoted by e_{ij}^c. The precedence constraint between CMs indicates which modules have to be completed before some other modules can be started. The ordered pair (T_i, D_i) † is introduced for labeling the module T_i which means that module T_i requires D_i units of execution time for completion. Similarly, a multiprocessor system can be defined by an undirected doubly weighted processor graph (UDPG) $G_p = (V_p, E_p)$, where V_p is a finite nonempty set of vertices denoting processors, $|V_p| = p \neq \varnothing$, and $E_p = \{(e_{ij}^s, e_{ij}^r) \mid e_{ij}^s, e_{ij}^r \in G_p\}$ is a set of finite double-weighted edges. (e_{ij}^s, e_{ij}^r) is an ordered pair associated with the edge connecting processor i to processor j in E_p, where e_{ij}^s indicates the message *sending* time incurred on processor i, and e_{ij}^r indicates the message *receiving* time incurred on processor j, if the necessary message communication is required to transfer from processor i to processor j. If we represent the sending time among the processors in the system in a matrix SM, then the (i, j) entry of this matrix, $SM(i, j)$, indicates the sending time between the processors i and j. Similarly, the matrix RM can be used to indicate the receiving time among the processors, and the (i, j) entry of this matrix, $RM(i, j)$, is the receiving time between processors i and j. In this paper, we assume that these two matrices are symmetric and that the diagonal elements of these matrices are zero to indicate the negligible message communication time within the same processor. Figure 1 shows a task graph, a processor graph, and their corresponding SM and RM matrices.

For a given DATG, if there is an edge from module i to module j, then module i is said to be an immediate predecessor of module j, and we denote it as $IPRED(j) = i$. If there is a directed path from module i to module j, then module i is said to be a predecessor of module j, and we denote it as $PRED(j) = i$. Initial modules are those modules with no predecessors, and terminal modules are those modules with no successors. The level l_i of a module T_i is the summation of the execution time associated with the modules in a path from T_i to a terminal module such that this sum is maximal. Such a path is called the *critical path* if the module T_i is the highest level in the DATG [12], and we define the critical path length as

$$D_{cp} \triangleq \max_{T_i \in V_c} l_i \tag{2}$$

where D_{cp} is the minimum possible finishing time for the multiprocessors to process all the modules in the given DATG. The physical meaning of the critical path, whichever scheduling method is employed, is the finishing time over all permissible schedules and cannot be shorter than the D_{cp}.

2.2. Processor Finishing Time and Interprocessor Communication Time

Two important parameters are usually considered in the mapping problem: the processor finishing time (PFT) and the interprocessor communication time (PCT). The processor finishing time of a processor k for a certain

† We also alternately write T_i to represent the module i.

mapping S, $PFT_k(S)$, $1 \leq k \leq p$, is the accumulated execution time of the modules assigned to that processor. If there exists some precedence constraint among these modules, then the $PFT_k(S)$ of processor k has to take the processor idle time into consideration. Determining the minimum processor finishing time with the precedence constraint taken into consideration is known as the minimum-finishing-time scheduling problem. The interprocessor communication time occurs only when two communicating modules are assigned to different processors such that the two modules residing on different processors can communicate through a communication channel or bus [15]. The message sent from one processor to other receiving processors causes the communication overhead among the sending and receiving processors. This results in the PCT which requires processing load on both the sending and receiving processors. For example, a specific module is sending a message from processor i to processor j. The sending processor i needs to spend time on message formatting and addresses initialization to the destination. Meanwhile, the receiving processor j needs to spend time on extracting the message contents from the sending processor. The message extracting time is usually much longer than the sending time which results in heavy traffic in the communication channel and causes the blocking of further messages arriving into the receiving processor j. Such PCT overhead can be eliminated if the two communicating modules are assigned to the same processor because both modules would locate at the same local memory, and the message communication time within the same processor is negligible and can be ignored.

2.3. Objective Function and Optimization Criterion

For most multiprocessor systems, minimizing the maximum processor finishing time is the most important performance measure and has the effect of evenly distributing the modules to all the processors to achieve the shortest possible computation time. In a task graph with the precedence constraint imposed on the modules, satisfying this mapping performance is known as the minimum-finishing-time scheduling problem. However, this performance criterion causes a negative effect which results in heavy communication costs among the processors (i.e., PCT). Minimizing the PCT alone may not produce a good assignment either because in a homogeneous system where all the processors are identical, a minimum PCT cost assignment will assign all the modules to a single processor, thus achieving zero PCT! The conflict of mapping performance between PFT and PCT has been studied [13].

The mapping problem is to find an optimal matching between a task graph and a processor graph, and this problem can be considered as a partition of the given DATG into several blocks with the modules in each block assigned to be executed by a corresponding processor. Figure 2 illustrates a partition of a DATG into three blocks to be executed on a three-processor computer system. Efficiency of a mapping can be improved by balancing the task modules among all the processors and minimizing the weight of the edges across the boundaries of the blocks which correspond to the communication costs among the processors.

Let S be a certain mapping and Ω be the set of all the mappings. Let ϕ_k, $1 \leq k \leq p$, be a partition of the given DATG such that $\bigcup_{k=1}^{p} \phi_k = V_c$ and $\phi_h \cap \phi_k = \varnothing$, $h \neq k$. The PCT incurred by processor k in the kth block, ϕ_k, is given by

$$PCT_k(S) = \sum_{i \in \phi_k,\, j,\, l \notin \phi_k} e^c_{ij} e^s_{ij} + e^c_{il} e^r_{il} \tag{3}$$

where modules i, j, and k are related by $IPRED(j) = i$ and $IPRED(i) = l$, and

$$e^c_{ij} = \begin{cases} 0\,, & \text{if } IPRED(j) \neq i \\ 1\,, & \text{if } IPRED(j) = i\,. \end{cases} \tag{4}$$

Let $E_k(S)$, $1 \leq k \leq p$, be the processor response time spent in processor k which consists of the sum of the processor finishing time $PFT_k(S)$ and the interprocessor communication time $PCT_k(S)$,

$$E_k(S) = PFT_k(S) + PCT_k(S)\,, \tag{5}$$

where the $PCT_k(S)$ is defined as in Eq. (3), and the $PFT_k(S)$ is the accumulated execution time of the modules assigned to that processor. The $PFT_k(S)$ is not simply the summation of the modules' execution time assigned to processor k; instead, it must include the processor idle time of that processor due to the precedence constraint. Usually, this value is greater than the summation of the modules' execution time assigned to that processor. If the system is initialized, the $PFT_k(S)$ is defined as the finishing time of executing the last module assigned to processor k.

Let $E(S)$ be the maximum processor response time. For a mapping S, $S \in \Omega$, the minimum-response-time mapping is the mapping S^* that minimizes the $E(S)$, that is,

$$E(S^*) = \min_{S \in \Omega} E(S) = \min_{S \in \Omega} \max_{1 \leq k \leq p} (PFT_k(S) + PCT_k(S)\,)\,. \tag{6}$$

This means that we want to minimize the maximum processor response time, resulting in the minimax optimization criterion.

2.4. Task Graph of Newton-Euler Equations of Motion

To achieve parallel processing with minimum response time, it is desirable to develop a directed task graph with maximum parallelism for the NE equations of motion. Unfortunately, a maximum-parallelism NE task graph may not yield a minimum-response-time mapping when the interprocessor communication time is taken into consideration. Thus, we perform a functional decomposition of the NE equations; that is, the equations are decomposed into computational modules, each of which calculates the kinematic and dynamic variables such as angular velocities, angular and linear accelerations, joint forces and moments, etc. The recursive structure of NE formulation with respect to the link coordinate systems is found to be in an inhomogeneous linear recurrence form (IHLR) which is not efficient for parallel processing [3,11]. On the other hand, when expressed in the base coordinate system, the NE equations are in a homogeneous linear recurrence form (HLR) which is more suitable for parallel processing [3,11]. The NE equations of motion expressed as HLR form with the detailed task modules for our mapping problem are shown in Fig. 3. For a six-jointed, PUMA-like manipulator, this task graph shows that the NE equations can be decomposed into 145 task modules [24].

Using the minimax optimization criterion as in Eq. (6), the mapping between the above NE task graph and a set of connected processors will be investigated. Our proposed mapping strategies for a multiprocessor system are substantially different from previous work, and their contributions can be seen in the following: (1) the precedence constraint and unequal module execution time of the task modules in the task graph are considered in the system; (2) unequal communication costs among processors in the processor graph are considered in the objective function; and (3) the minimax optimization criterion is used to obtain the minimum-response-time mapping. The proposed mapping algorithms are very general and can be applied to most multiprocessor systems.

3. The Mapping Algorithm

The mapping problem can be considered as decomposing a computational task into a set of task modules executed concurrently on a multiprocessor system. The design of this mapping is divided into two major steps: graph partitioning and module allocation. Both of these two steps are known to be *NP*-complete. Thus, we propose two heuristic mapping algorithms to obtain fast mapping solutions for computing the NE equations of motion. Both heuristic algorithms take the PFT, the PCT, the precedence constraint of the NE task graph, and the multiprocessor system structure into consideration.

3.1. Heuristic Mapping Algorithm with Weighted Bipartite Matching

Based on the given NE task graph in Fig. 3, the heuristic algorithm constructs a *dynamic* priority list containing all the task modules arranged in a descending order according to the weighted level l_i and the communication intensity of the task modules. Most of the priority lists developed in previous research do not include the communication costs [9,11]. To develop the dynamic priority list, let us denote $A(n)$ to be a set of modules that have been assigned to the processors at the nth insertion stage (i.e., the modules that have been inserted into the mapping-schedule from the dynamic priority list), and let $\bar{A}(n)$ be the complement of $A(n)$. Let $P_{mft}(n)$ be the processor(s) with the minimum finishing time at this stage, and let $K(n)$ denote the set of modules assigned to the remaining processors which have not yet finished processing. Let $FW(\bar{A}(n))$ be the function that returns the set of modules, $W(n)$, which are ready to be assigned to all the p processors, that is, for all modules $T_i \in \bar{A}(n)$, if and only if $PRED(T_i) \notin \bar{A}(n)$. Similarly, the function $FW(K(n) \cup \bar{A}(n)) - K(n)$ returns the set of modules, $R(n)$, which are ready to be assigned to the $P_{mft}(n)$. These notations will be useful for developing the proposed heuristic algorithm.

The mapping-schedule obtained from this heuristic algorithm starts from zero initially and gradually the ready modules are "inserted" into the mapping-schedule until all the modules have been inserted. Instead of randomly inserting the ready modules into the available processors $P_{mft}(n)$, the insertion of ready modules needs to consider the increased communication costs they created. In order to minimize the maximum processor response time, this insertion of ready modules has to be carefully selected so that it does not increase the PCT while maintaining the minimum-finishing-time schedule. This insertion process (or module allocation process) can be done by a weighted *Bipartite Matching Algorithm* [20] which will be discussed later. Using the weighted bipartite matching algorithm at each stage of the insertion of ready modules into the mapping-schedule, the PCT will be maintained to be the minimum.

Minimizing the maximum processor response time also requires us to consider the PFT at each stage of the insertion of ready modules. Due to the precedence constraint of the task graph, some idle modules might have to be assigned to the available processors during the insertion of the ready modules. Table 1 shows the PFT of the mapping-schedule in Fig. 1 when the new non-idle module is inserted to that processor. In order to include both the PFT and the PCT in the construction of the dynamic priority list to order the ready modules, we introduce two parameters, α and β, for weighting the level l_i and the communication intensity of task modules, respectively, in adjusting the assignment priority coefficient ρ_i. The ready modules are ordered into a priority list according to the descending order of the assignment priority coefficient ρ_i. Then the ready modules in the priority list can be allocated to the available processors using the weighted bipartite matching algorithm.

Algorithm HM (*Heuristic Mapping Algorithm*). Given a DATG and an UDPG, this algorithm constructs a dynamic priority list of all the task modules and inserts the modules one by one into the mapping-schedule according to the weighted bipartite matching algorithm.

BBH1.[Initialization.] Input the given DATG. Obtain the critical path D_{cp} and the total number of edges, *nedge*, in the DATG, where $nedge = |E_c|$.

H2. [Initialize Looping.] Set the mapping-schedule empty (i.e., $A(n) = \emptyset$); initialize the processor finishing time PFT and the interprocessor communication time PCT. Create two parameters α and β ($\alpha + \beta = 1$), α from one to zero and β from zero to one with an increment/decrement *ΔSTEP*. Set $n \leftarrow 1$.

H3. [Determine the level of modules in $R(n)$.] Determine the set of ready modules $R(n)$, and find the level l_i of each module T_i in the set $R(n)$.

H4. [Obtain the successors and the predecessors number.] For each module T_i in the set $R(n)$, find its number of successors, ns_i, and its number of predecessors, np_i.

H5. [Evaluate the assignment priority coefficient.] Evaluate the assignment priority coefficient, ρ_i, of module T_i from the equation

$$\rho_i = \alpha\,(\frac{l_i}{D_{cp}}) + \beta\,(\frac{ns_i + np_i}{nedge}) \quad , \text{ where } \alpha + \beta = 1 \,. \tag{7}$$

H6. [Order the priority list.] For all the ready modules in the set $R(n)$, construct an ordered priority list $R_l(n)$ according to the descending order of ρ_i.

H7. [Assign the modules.] Determine the number of available processors $p_{av} = |\ P_{mft}(n)|$. According to the weighted bipartite matching algorithm, assign the first p_{av} modules to the available processors based on the priority list $R_l(n)$. Record the modules in each processor, the PFT, and the PCT.

H8. [End of mapping-schedule?] If $\bar{A}(n) \neq \emptyset$, then set $n \leftarrow n + 1$, and go to step H3; otherwise, continue.

H9. [Obtain minimum cost among all mappings.] If $\bar{A}(n) = \emptyset$, record the mapping-schedule and the costs of this mapping. If $\alpha \neq 1$, then go to step H2 to obtain another mapping; otherwise, stop and obtain the minimum cost among all the mappings.

END HM.

In the above HM algorithm, based on the level l_i and the communication intensity of the task modules in the DATG, the weighting parameters, α and β, are used to evaluate the assignment priority coefficient ρ_i which is used to order the priority list $R_l(n)$ (steps H5 and H6). The parameters α and β are used to weight the level and the communication intensity of the task modules, respectively. In the extreme case, when $\alpha = 1$, the ready modules are ordered according to their level in the task graph. Similarly, when $\beta = 1$, the ready modules are ordered according to the communication intensity of the task modules in the task graph. By suitably adjusting α and β, various orderings of the ready modules in the priority list at each stage of the insertion can be generated, thus providing a better mapping solution.

The weighted bipartite matching problem is also known as the "job-worker" assignment problem. Each worker must be assigned to exactly one job, and each job must have one assigned worker. If job i is assigned to worker j, then a benefit of the objective function is realized. It is desirable to make an assignment such that the total benefit of the objective function is optimized. In our mapping problem, the jobs and workers are, respectively, corresponding to the ready modules and the available processors at each stage of the insertion. Since there are p_{av} available processors at each stage of the insertion, the *weights* or benefits produced by job i (or ready module i) assigned to worker j (or available processor j) is not simply a scalar value. Instead, the weights are expressed in a p_{av}-tuple, each of the kth element in this p_{av}-tuple is the benefit obtained by the kth processor in this assignment. We want to minimize the maximum element of this p_{av}-tuple to obtain the minimum cost of the assignment. Before we formulate the weighted bipartite matching algorithm, two operations are defined on the ordered p-tuples. Let $D = (d_1, d_2, \cdots, d_p)$ be an ordered p-tuple and $U = (u_1, u_2, \cdots, u_p)$ be an another ordered p-tuple. Then, the sum operation of these two ordered p-tuples, written as $D \oplus U$, results in another p-tuple, W,

$$W = (w_1, w_2, \cdots, w_p) = D \oplus U \triangleq (d_1+u_1, d_2+u_2, \cdots, d_p+u_p). \tag{8}$$

The maximum value of this p-tuple, written as $\max_p$, is the maximum element in W, that is, $\max_p \triangleq \max\limits_{1 \le i \le p} w_i$. The summation of a series of p-tuples is written as $\sum_{\oplus}$.

Algorithm WBM (*Weighted Bipartite Matching Algorithm*). Given an ordered priority list of the ready modules $R_i(n)$, the available processors, the number of available processors, the mapping-schedule at the nth stage, the first p_{av} ready modules denoted as $R_l^{p_{av}}(n)$, the matrix RM, the p_{av}-tuple processor finishing time $PFT^{p_{av}}(n-1)$, and the p_{av}-tuple interprocessor communication time $PCT^{p_{av}}(n-1)$, this algorithm assigns the ready modules to the available processors such that the PCT among all the processors is minimum.

W1. [Obtain communication time vector.] For module y in $R_l^{p_{av}}(n)$, obtain the number of its predecessors, n_{yk}, located in processor k, $1 \le k \le p_{av}$. Construct a $p_{av} \times 1$ vector; each entry of this vector is a p_{av}-tuple. The kth element of the ith entry of this vector, $k \ne i$, is the PCT incurred on processor k if module y is assigned to processor i and is calculated as $\sigma_k = RM(i,k)n_{yk}$. The ith element of the ith entry of the communication time vector is $\sum_{k=1,k\ne i}^{p_{av}} \sigma_k$. This is due to the PCT incurred on processor i and is defined as the sum of the weights of the edges across the boundary of the block i as we indicated in Eq. (3).

W2. [Obtain communication time matrix.] For all the ready modules in $R_l^{p_{av}}(n)$, construct the $p_{av} \times p_{av}$ communication time matrix $\Gamma(n)$ by collecting all the communication time vectors. Note that the ith column of this matrix corresponds to the communication time vector which is created by the module y, and this module is located at the ith position of the $R_l^{p_{av}}(n)$ priority list.

W3. [Obtain cost matrix.] Construct a $p_{av} \times p_{av}$ cost matrix $C(n) = [c_{ij}(n)]$ by

$$c_{ij}(n) = \Gamma_{ij}(n) \oplus (0, \cdots, PFT_i^{p_{av}}(n-1), \cdots, 0) \oplus PCT^{p_{av}}(n-1) \oplus (0, \cdots, D_y, \cdots, 0), \tag{9}$$

where $i, j = 1, 2, \cdots, p_{av}$, $PFT_i^{p_{av}}(n-1)$ is the ith element of the p_{av}-tuple at the $(n-1)$th stage, D_y is the execution time of module y and is located at the ith position of the p_{av}-tuple. Note that c_{ij} is a p_{av}-tuple.

W4. [Weighted bipartite assignment problem.] Solve the module assignment problem by finding an assignment matrix $X(n) = [x_{ij}(n)]$, $i, j = 1, 2, \cdots, p_{av}$ to

$$\min_{X(n)} \left(\max_{p_{av}} \sum_{i}{}_{\oplus} \sum_{j}{}_{\oplus} c_{ij}(n) x_{ij}(n) \right) \tag{10}$$

subject to $\sum_{j=1}^{p_{av}} x_{ij}(n) = 1$ for $i = 1, 2, \cdots, p_{av}$, $\sum_{i=1}^{p_{av}} x_{ij}(n) = 1$ for $j = 1, 2, \cdots, p_{av}$, and $x_{ij}(n) = 1$ or 0.

W5. [Assign modules to processors.] According to the module assignment matrix $X(n)$, assign the modules in $R_l^{p_{av}}(n)$ to the available processors.

END WBM.

When inserting the p_{av} ready modules into the mapping-schedule at the nth stage, the cost matrix $C=[c_{ij}(n)]$ in Eq. (9) includes the communication time the modules are going to create with the assigned modules in the $A(n)$, the processor finishing time of the mapping-schedule, $PFT^{p_{av}}(n-1)$, the interprocessor communication time of the mapping-schedule, $PCT^{p_{av}}(n-1)$, and the execution time of the modules to guarantee the best assignment.

As an example, consider the mapping-schedule of the given DATG in Fig. 1 ($m = 9$). The task modules are to be executed by 3 processors ($p = 3$). The level number and execution time of each module are given beside each module in the DATG, the e_{ij}^r and e_{ij}^s, $i, j = 1, 2, 3$, are shown in the UDPG. We use the HM algorithm and let $\alpha = 0$, $\beta = 1$ to determine a heuristic mapping-schedule. The $R(n)$, p_{av}, $R_l^{p_{av}}(n)$, $PFT(n)$, and $PCT(n)$ associated with each stage of insertion are listed in Table 2. The PFT and PCT of this mapping are found to be PFT $= (13, 15, 7)$ and PCT $= (6, 7, 7)$; the cost of this mapping is 22 units, and the Gantt chart of this suboptimal mapping is shown in Fig. 4.

3.2. Heuristic Mapping Algorithm with Simulated Annealing

In order to improve the mapping solution obtained by the above HM algorithm, the simulated annealing method [21], which incorporates an iterative improvement scheme and the Metropolis procedure, is introduced to find a near-optimal mapping. Consider that one starts with an initial state (i.e., a partition of the task graph DATG) of the optimization problem, instead of always rejecting the new state that increases the objective function, this new state with small conditional probability is accepted. In other words, if $\Delta E \le 0$, then accept this new state; if $\Delta E > 0$, then accept this new state with probability $e^{-\Delta E/T}$, where $\Delta E = E(S_{new}) - E(S)$ is the amount of the increase in the maximum processor response time $E(S)$, S_{new} and S are, respectively, the new and old states, and T is a control parameter. Initially, one starts with a large value of T; after the system is approaching the equilibrium at this T value, then T is reduced to a lower value and the system will approach to another equilibrium. This procedure is stopped at a certain desirable T, or no more improvement can be expected. By conditionally accepting the increased $E(S_{new})$ and by varying the control parameter T, one can escape the pitfall of the local optimal and obtain a better mapping.

Applying this simulated annealing method to our mapping problem, a random state is generated when all the modules in the DATG are randomly assigned to the processors and the PCT of this assignment is obtained (this is the graph partitioning problem); then based on the modules assigned to these processors, we schedule the computation of these modules in the processors according to the precedence constraint imposed on them and the PFT of this assignment is obtained (this is the scheduling problem). To compute the objective function of a new state, one needs to compute both the PCT and the PFT. The PCT of a state is simply the weights of all the edges across the boundaries of each block while the PFT of this state is difficult to obtain because obtaining the optimal schedule is difficult. Thus, one may use the heuristic scheduling algorithm [11] to obtain a suboptimal schedule or derive the lower bound of the PFT of the schedule. Since the graph partitioning randomly assigns task modules to be executed on specific processors, the scheduling problem that follows needs to address the restriction of executing these modules on their specific processors. This scheduling problem can be solved by the proposed processor-restricted dynamical-highest-level-first/most-immediate-successors-first algorithm (Algorithm *PRDHLF/MISF*).

Algorithm PRDHLF/MISF. (*Processor-Restricted Dynamical Highest Level First/Most Immediate Successors First Algorithm*). Given a task graph and restricting the execution of the modules on specific processors, this algorithm constructs a dynamic priority list of all the modules and inserts the modules one by one into the suboptimal schedule.

D1. [Initialization.] Initially, the schedule is empty (i.e. $A(n) = \varnothing$). Let P_i be the set of modules that are pre-assigned to the processor i , $i = 1, 2, \cdots, p$. Let $P^i_{mft}(n)$, $i = 1, 2, \cdots, p$, be the processor i having the minimum-finishing-time at the nth stage of the insertion.

D2. [Determine the set $R(n)$ and the sets $R_i(n)$.] Determine the set $R(n)$ and obtain $R_i(n) = R(n) \cap P_i$, $i = 1, 2, \cdots, p$, where $R_i(n)$ is the the set of ready modules assigned to the processor i.

D3. [Determine the levels of modules in $R_i(n)$.] Find the level l_k, $T_k \in R_i(n)$, for each module in the set $R_i(n)$, $i = 1, 2, \cdots, p$.

D4. [Construct the priority lists.] Construct the ith dynamic priority list of processor i in a descending order of l_k. If the levels of the modules are tied, then the module having the largest number of immediately successive modules is assigned to the highest priority.

D5. [Assign the modules.] Assign the modules to the $P^i_{mft}(n)$ on the basis of the ith priority list, $i = 1, 2, \cdots, p$. If $\bar{A}(n) = \varnothing$, then stop; otherwise, go to step D2.

END PRDHLF/MISF.

Although the *PRDHLF/MISF* algorithm provides a fast but suboptimal schedule, it is still very time-consuming to obtain the schedule when the number of task modules is large. Since our objective is to compute the PFT instead of finding a complete schedule, it is more desirable to find the lower bound of the PFT of the schedule. The lower bound of the PFT of the schedule without the restriction of executing the modules on specific processors has been discussed [22]. The lower bound of the PFT of the schedule with the restriction of executing the modules on specific processors is given by

$$PFT_i^{LB}(S) = \max_{T_j \in P_i} [\tau_l^u(T_j) + q_i] \tag{11}$$

where the notation and the proof of this lower bound can be found in [24].

With the lower bound of the PFT in Eq. (11) and the PCT obtained from the graph partitioning, the objective function of a state S can be easily obtained by

$$E(S) = \max_{1 \le k \le p}(PFT_k^{LB}(S) + PCT_k(S)). \tag{12}$$

Using the objective function in Eq. (12), the heuristic mapping algorithm with simulated annealing to obtain a near-optimal mapping is summarized in the following.

Algorithm HMSA (*Heuristic Mapping Algorithm with Simulated Annealing*) Given a DATG, an UDPG, the matrices *SM* and *RM*, and the control parameter T, this algorithm generates a near-optimal solution of the mapping problem.

S1. [Initialization.] Select an initial state S, and evaluate the objective function $E(S)$ in Eq. (12).

S2. [Looping with T.] If the equilibrium with respect to the control parameter T is reached, go to step S6; otherwise continue.

S3. [Generate new state.] Generate a new state S_{new}, and calculate $E(S_{new})$ of this new state according to Eq. (12). Obtain $\Delta E = E(S_{new}) - E(S)$. Select a random variable γ, $0 \le \gamma \le 1$. If $\Delta E \le 0$, go to step S5; otherwise go to step S4.

S4. [$\Delta E > 0$.] If $\gamma < e^{-\Delta E/T}$, then accept this new state with probability $e^{-\Delta E/T}$ and go to step S5; otherwise reject this new state and go to step S3.

S5. [$\Delta E \le 0$ or $\gamma < e^{-\Delta E/T}$.] Set $S \leftarrow S_{new}$. Record $E(S_{new})$ and go to step S2.

S6. [Looping with a smaller value of T.] If $T >$ *stopping criterion*, set $T \leftarrow T - \Delta T$, where ΔT is a user-designed variable, and go to step S3; otherwise, continue.

S7. [Obtain the mapping.] Obtain the state which has the smallest $E(S)$. Based on this state, schedule the modules according to the precedence constraint of the DATG and obtain the mapping.

END HMSA.

The HMSA algorithm is used to find a near-optimal mapping for the example in Fig. 1. An initial partition of the task graph assigns the modules $\{T_1, T_2, T_3\}$, $\{T_7, T_8, T_9\}$, and $\{T_4, T_5, T_6\}$ to processors 1, 2, and 3 for execution, respectively. At this initial state, the PFT_i^{LB} for $i = 1, 2, 3$ are respectively 10, 21, 15, and the PCT_i are 5, 12, and 21. The total cost of this state is 36 units. In this example, 273 iterations were required to reach the near-optimal mapping. The PFT_i^{LB} for the final state are 12, 15, and 13, and the PCT_i are 8, 7, and 9. The total cost of this final state is 22 units, and the partition of the task graph for this mapping assigns the modules $\{T_3, T_6, T_7\}$, $\{T_5, T_9\}$, and $\{T_1, T_2, T_4, T_8\}$ to processors 1, 2, and 3 for execution, respectively. It is interesting to note that both the HM and HMSA algorithms reached the same total cost of 22 units with different mapping solutions. Further computer simulations and actual implementation of the proposed algorithms on an NCUBE/ten multiprocessor system are described in the next section.

4. Computer Simulations and Experimental Results

Since the optimal mapping solution is *NP*-complete, the design of computer simulations for evaluating the efficiency of the proposed HM and HMSA algorithms as compared with the optimal solution is constrained by the number of task modules and processors that our computer can process within a reasonable time. A total of 100 random DATGs, each with the number of modules ranging from 5 to 10, was generated for mapping onto multiprocessor systems with 2 to 3 processors. The ratio of average module execution time of the modules in the DATG and the average interprocessor communication time between paired processors, P/C, was introduced to determine the communication overhead affecting the performance of the algorithms. In our computer simulations, all the optimal solutions were obtained by the exhaustive search method. Table 3 shows the efficiency of the HM algorithm with a P/C ratio ranging from 10 to 0.1 together with different numbers of processors. To evaluate the efficiency of the heuristic algorithms, an average relative error ε over N mappings is defined as

$$\varepsilon = \frac{1}{N} \sum_{i=1}^{N} \frac{h_i - o_i^*}{o_i^*} \tag{13}$$

where h_i is the ith heuristic solution and o_i^* is the ith optimal solution.

From the computer simulation, when the P/C ratio is less than 1, the solutions obtained by the HM algorithm were found to be unsatisfactory. This is due to the fact that the HM algorithm always distributes the ready modules to the available processors to achieve the minimum communication costs. When the average communication time is much greater than the average module execution time or the communication link density is higher, evenly distributing the ready modules to all the available processors will not produce a better mapping. Instead, it may happen that all the ready modules may be assigned to the same processor to reduce the large average communication time and sacrifice relatively small average module execution time to yield a better response. Thus, when the P/C ratio is less than one, a sequential computation of all the task modules on a single processor will yield a better result than parallel processing. Hence the performance of a multiprocessor system is closely related to the P/C ratio. To improve the performance of a multiprocessor system, it is suggested that the computational task should be decomposed such that the P/C ratio is greater than one. In our computer simulation, when the P/C ratio is greater than one, all the solutions obtained by the HM algorithm approached to the near-optimal solutions. The mapping solutions obtained from the HMSA algorithm for all the 100 DATGs agreed with the optimal solutions obtained by the exhaustive search method.

To further evaluate and validate the effectiveness of our proposed mapping algorithms, the computation of the robot inverse dynamics based on the NE task graph in Fig. 3 is implemented on an NCUBE/ten multiprocessor system. Once the sending and receiving processors are given, the path is obtained by the operating system of the machine. The performance of communication activities between two processors in the hypercube computer has been addressed [23]. It is shown that the hypercube machine has a high communication cost compared with the processing time of computing elementary multiplication or addition operation. For transferring a 12-byte (or 96-bit)† message, the ratio

† For a functional decomposition, each entity of the computation is 3×1 vector (e.g., ω_i etc.) and each scale value requires a 4-byte (32-bit) computation; thus the standard size to transfer a vector is 12 bytes.

of multiplication (or addition) operation and the message passing between two adjacent processors is approximately $P/C \approx 50/510 \ll 1$. This indicates that for implementation on NCUBE machines, it is not advisable to decompose the NE equations of motion into elementary operations. Thus, we perform functional decomposition on the NE equations of motion which results in the task graph in Fig. 3. The communication time between two processors for our NCUBE/ten machine has been measured experimentally and is found to be 0.34 milliseconds for sending a 12-byte message from a source processor to any destination processor. For receiving the message, we found that it takes 1.02, 1.7, and 2.2 milliseconds to receive a 12-byte message between one, two, and three hops‡, respectively.

A recent result indicates that it takes 24.8 milliseconds to compute the robot inverse dynamics on a uniprocessor system [8]. Due to the high communication time, if we implement the robot inverse dynamics computation on the hypercube computer, it may not generate a faster result than that running on a uniprocessor computer. In order to avoid mapping all the modules to a single processor by the proposed heuristic algorithms, we artificially set the execution time of a 12-byte elementary operation (multiplication or addition) to 3.4 milliseconds which is ten times of the time for sending a 12-byte message. Based on the NE task graph in Fig. 3, for a 6-jointed PUMA-like manipulator, the NE equations can be decomposed into 145 task modules. The detailed computation time and precedence relationship of each module can be found in [24]. Using the sending time matrix *SM* and the receiving time matrix *RM* of a subcube of four processors of our NCUBE/ten computer, we simulated on a VAX-11/780 computer the HM algorithm for finding our mapping solution. Based on the 145-module NE task graph, a suboptimal mapping was obtained from this computer simulation. The minimum processor response time of each processor from the suboptimal mapping was found to be 704.82, 701.42, 706.52, and 703.12 milliseconds for processor 0, 1, 2, and 3, respectively. We then implemented the mapping solution from the computer simulation into the actual NCUBE/ten machine. The minimum processor response time of each processor from the mapping was found to be 679.71, 676.69, 675.78, and 669.23 milliseconds for processor 0, 1, 2, and 3, respectively. Table 4 compares the computer simulation result with the actual implementation result and shows a maximum of five percent relative error between them.

In order to achieve the critical-path-length computation of the 145-module NE task graph, the hypercube dimension was increased from $N = 2$ to $N = 3$; that is, 8 processors were used for the computations. The details of the allocation of the 145 modules in each of the eight processors can be found in [24]. Using $p = 8$ processors, our HM algorithm achieves the critical-path-length computation of 400.18 milliseconds which incurred in processor number 3; with a communication time of 27.86 milliseconds, it gave a shortest possible response time of 428.04 milliseconds. The critical-path-length computation of 400.18 milliseconds means that the use of more than 8 processors for parallel processing will never obtain a shorter processing time. The comparison of the computer simulation result and the implementation result is shown in Table 5.

5. Conclusions

The problem of determining the optimal mapping between a computational task consisting of m modules with precedence constraint and a set of connected p identical processors with interprocessor communication time is formulated as an equivalent problem of solving the graph partitioning and the scheduling problems. Both of these problems are known to be *NP*-complete. Two efficient mapping algorithms, the HM algorithm and the HMSA algorithm, were proposed to determine fast and suboptimal or near-optimal mapping solutions. Minimizing the maximum of the sum of the processor finishing time and the interprocessor communication time is used as an optimization criterion for determining the best mapping. The proposed HM algorithm and the HMSA algorithm greatly reduce the time complexity of determining the mapping. Computer simulation results indicated that the proposed mapping algorithms are efficient and practical and that they can provide suboptimal as well as near-optimal solutions. Computer simulation was also conducted to simulate the operation of an NCUBE/ten hypercube computer for determining the mapping of the robot inverse dynamics computation of a 6-jointed PUMA-like manipulator. Actual implementation of the HM algorithm on the NCUBE/ten hypercube computer was also performed.

References

[1] J. Y. S. Luh, M. W. Walker, and R. P. Paul, "On-line Computational Scheme for Mechanical Manipulator," *Trans. ASME, J. Dynam., Syst., Meas., Contr.*, vol. 120, pp. 69-76, June 1980.

[2] D. E. Orin, R. B. MaChee, M. Vukobratovic, and G. Hartoch, "Kinematics and Kinetic Analysis of Open-Chain Linkages Utilizing Newton-Euler Methods," *Math. Biosci.*, vol. 43, pp. 107-130, 1979.

[3] C. S. G. Lee and P. R. Chang, "Efficient Parallel Algorithm for Robot Inverse Dynamics Computation," *IEEE Trans. Syst. Man, and Cybern.*, vol. SMC-16, no. 4, pp. 532-542, July 1986.

[4] C. S. G. Lee, T. N. Mudge, and J. L. Turney, "Hierarchical Control Structure Using Special Purpose Processor for the Control of Robot Arm," *Proc. 1982 Conf. Patt. Recog. and Image Processing*, pp. 634-640, June 1982.

‡The hops is the number of edges of the hypercube which the message must traverse when the message is sending from the source processor to the destination processor.

[5] R. Nigam and C. S. G. Lee, "A Multiprocessor-Based Controller for Control of Mechanical Manipulators," *IEEE J. of Robotics and Autom.*, vol. RA-1, no. 4, pp. 173-182, Dec. 1985

[6] T. Kanade, P. K. Khosla, and N. Tanaka, "Real-Time Control of the CMU Direct Arm II Using Customized Inverse Dynamics," *Proc. of IEEE Conf. on Decision and Contr.*, pp. 1345-1352, Dec. 1984.

[7] L. H. Lathrop, "Parallelism in Manipulator Dynamics," *Int'l J. of Robotics Res.*, vol. 4, no. 2, pp. 80-102, Summer 1985.

[8] J. Y. S. Luh and C. S. Lin, "Scheduling of Parallel Computer for a Computer-Controlled Mechanical Manipulator," *IEEE Trans. Syst. Man, and Cybern.*, vol. 12, pp. 214-234, 1982.

[9] H. Kasahara and S. Narita, "Parallel Processing of Robot-Arm Control Computation on a Multiprocessor System," *IEEE J. of Robotics and Autom.*, vol. RA-1, no. 2, pp. 104-113, June 1985.

[10] J. Barhen, "Robot Inverse Dynamics on a Concurrent Computation Ensemble," *Proc. of 1985 ASME Int'l Conf. on Computers in Engineering*, vol. 3, pp. 415-429, 1985.

[11] C. L. Chen, C. S. G. Lee, and E. S. H. Hou, "Efficient Scheduling Algorithms of Robot Inverse Dynamics Computation on a Multiprocessor System," *IEEE Trans. Syst. Man, and Cybern.*, vol. SMC-18, no. 5, September/October 1988.

[12] E. G. Coffman, *Computer and Job-Shop Scheduling Theory*, Wiley, New York, 1976.

[13] K. Efe, "Heuristic Models of Task Assignment Scheduling in distributed Systems," *Computer*, vol. 15 pp. 50-56, July 1982.

[14] S. H. Bokhari, "On the Mapping Problem," *IEEE Trans. Computer*, vol. C-30, pp. 207-214, Mar. 1981.

[15] W. W. Chu and L. M-T Lan, "Task Allocation and Precedence Relations for Distributed Real-Time Systems," *IEEE Trans. Computer*, vol. C-36, pp. 667-679, June 1987.

[16] S. Y. Lee and J. K. Aggarwal, "A Mapping Strategy for Parallel Processing," *IEEE Trans. Computer*, vol. C-36, pp. 433-441, Apral 1987.

[17] E. R. Barnes, "An Algorithm for Partitioning the Nodes of a Graph," *SIAM J. Alg. Disc. Math.*, vol. 3 no. 4, pp. 541-550, Dec. 1982.

[18] W. E. Donath and A. J. Hoffman, "Lower Bounds for the Partitioning of Graphs," *IBM J. Res. Develop.*, vol. 17, pp. 420-425, Sept. 1973.

[19] B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graph," *Bell System Tech. Journal*, vol. 49, no. 2, pp. 291-307, Feb. 1970.

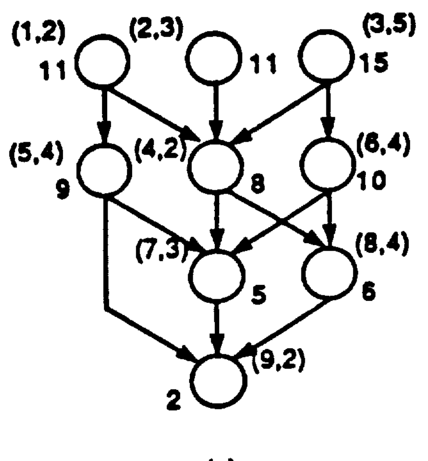
[20] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization, Algorithms and Complexity*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982.

[21] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp. 671-680, 1983.

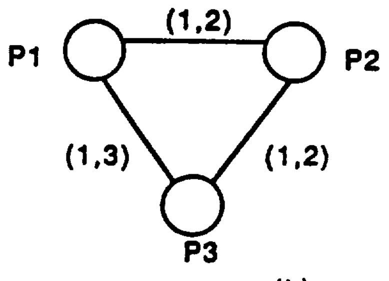
[22] E. B. Fernandez and B. Bussell, "Bound on the Number of Processors and Time for Multiprocessor Optimal Schedules," *IEEE Trans. Computer*,

[23] A. Beguelin and D. J. Vasicek, "Communication between Nodes of a Hypercube," *Proc. 2nd Conf. Hypercube Multiprocessors*, Knoxville, TN, pp. 162-168, Sept. 1986.

[24] C. L. Chen, "Efficient Mapping Algorithms for Scheduling Autonomous Vehicles and Robotic Computations," Ph.D. dissertation, School of Electrical Engineering, Purdue University, August 1988.



(a)



(b)

$$SM = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \; ; \; RM = \begin{bmatrix} 0 & 2 & 3 \\ 2 & 0 & 2 \\ 3 & 2 & 0 \end{bmatrix}$$

Figure 1. **(a) A direct task graph.**
(b) An undirected processor graph with *SM* and *RM* matrices.

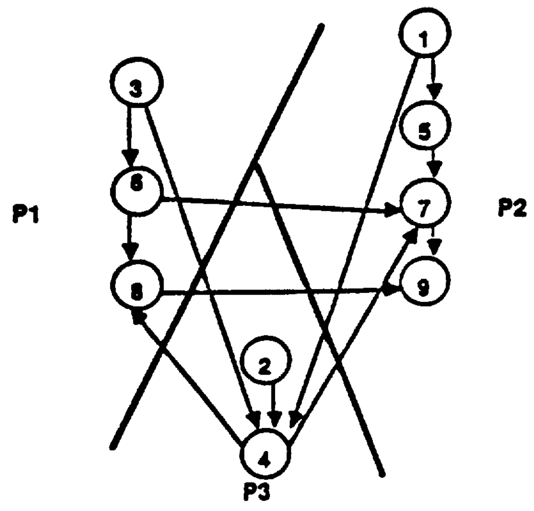


Figure 2. Partitioning a task graph into three blocks.

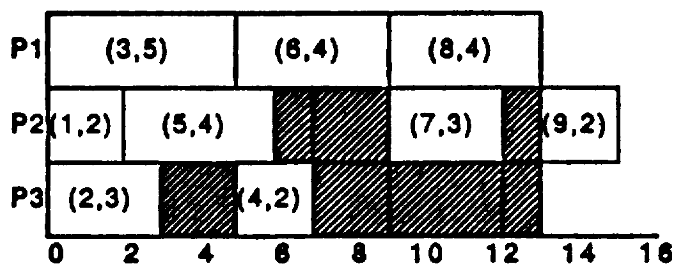


Figure 4. The Gantt chart for the Fig. 1 example.

Modules Description

$T_1 = {}^0\mathbf{R}_i$

$T_2 = \mathbf{p}_i^* = {}^0\mathbf{R}_i\ {}^i\mathbf{p}_i^*$

$T_3 = \mathbf{z}_{i-1}\dot{q}_i\ (1-\lambda_i)$

$T_4 = \boldsymbol{\omega}_i$

$T_5 = \mathbf{s}_i = {}^0\mathbf{R}_i\ {}^i\mathbf{s}_i$

$T_6 = (\mathbf{z}_{i-1}\ \ddot{q}_i + \boldsymbol{\omega}_{i-1} \times \mathbf{z}_{i-1}\ \dot{q}_i)\ (1-\lambda_i)$

$T_7 = \dot{\boldsymbol{\omega}}_i$

$$T_8 = \dot{\boldsymbol{\omega}}_i \times \mathbf{p}_i^* + \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times \mathbf{p}_i^*) + (\mathbf{z}_{i-1}\ \ddot{q}_i + 2\ \boldsymbol{\omega}_i \times (\mathbf{z}_{i-1}\ \dot{q}_i))\ \lambda_i$$

$T_9 = \ddot{\mathbf{p}}_i$

$T_{10} = \ddot{\mathbf{r}}_i$

$T_{11} = \mathbf{F}_i$

$T_{12} = {}^i\boldsymbol{\omega}_i = {}^i\mathbf{R}_0\boldsymbol{\omega}_i$

$T_{13} = {}^i\dot{\boldsymbol{\omega}}_i = {}^i\mathbf{R}_0\dot{\boldsymbol{\omega}}_i$

$T_{14} = {}^i\mathbf{N}_i = {}^i\mathbf{J}_i\ {}^i\dot{\boldsymbol{\omega}}_i + {}^i\boldsymbol{\omega}_i \times ({}^i\mathbf{J}_i\ {}^i\boldsymbol{\omega}_i)$

$T_{15} = \mathbf{N}_i$

$T_{16} = \mathbf{f}_i$

$T_{17} = \mathbf{N}_i + (\mathbf{p}_i^* + \mathbf{s}_i) \times \mathbf{F}_i + \mathbf{p}_i^* \times \mathbf{f}_{i+1}$

$T_{18} = \mathbf{n}_i$

$T_{19} = \tau_i$

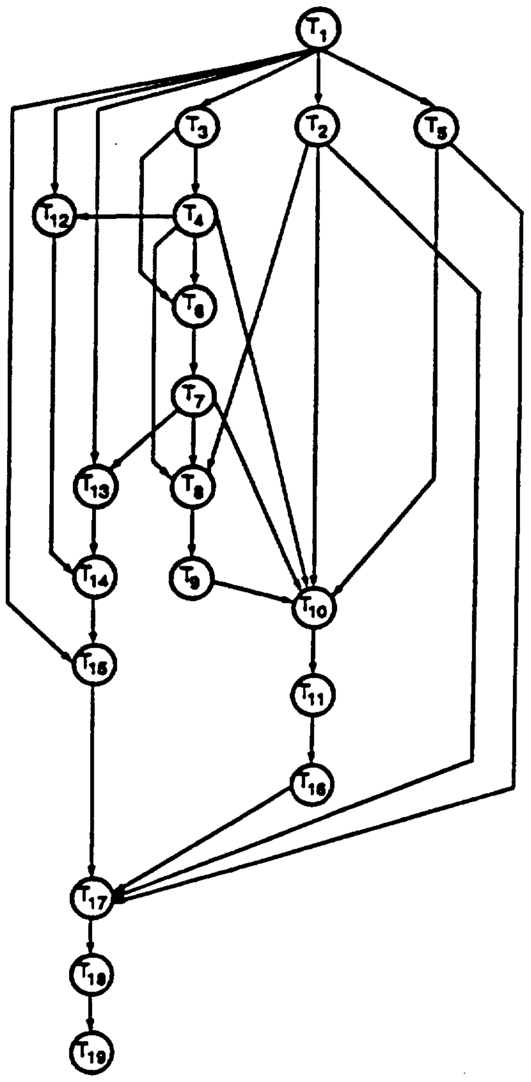


Figure 3. Task graph of Newton-Euler equations of motion.

Table 1. The PFT of the mapping-schedule in Fig. 1 example.

Insertion Stage	$PFT_1(n)$	$PFT_2(n)$	$PFT_3(n)$
1	2	2	2
2		6	6
3	7	7	7
4	10	10	
5		11	11
6		13	13
7		17	
8		20	
9		22	

Table 2. Cost value of each insertion stage for the Fig. 1 example.

n	$R(n)$	$P_{mft}(n)$	p_{av}	$R_i^{p-}(n)$	PFT	PCT
1	{1,2,3}	1,2,3	3	{3,2,1}	(5,2,3)	(0,0,0)
2	{ 5 }	2	1	{ 5 }	(5,6,3)	(0,0,0)
3	∅	3	1	∅	(5,6,5)	(0,0,0)
4	{4,6}	1,3	2	{6,4}	(9,6,7)	(0,0,0)
5	∅	2	1	∅	(9,7,7)	(1,1,5)
6	∅	2,3	2	∅	(9,9,9)	(1,1,5)
7	{7,8}	1,2,3	3	{8,7,idle}	(13,12,12)	(1,1,5)
8	∅	2,3	2	∅	(13,13,13)	(5,5,7)
9	{ 9 }	1,2,3	3	{9,idle,idle}	(13,15,7)	(5,5,7)
10						(6,7,7)

Table 3. Simulation results of the HM algorithm with different P/C ratios.

P/C	Optimality Percentage (%)		Relative Error (%)	
	Number of Processors = 2	Number of Processors = 3	Number of Processors = 2	Number of Processors = 3
10	71.43	90.48	3.05	0.44
5	71.43	90.48	3.67	0.29
2	71.43	95.24	5.61	0.05
1	52.38	66.67	13.47	3.09
0.5	30.77	15.38	17.34	16.65
0.2	0	0	102.02	92.26
0.1	0	0	224.32	254.36

Table 4. Comparison between simulation and implementation results for the four-processor case.

Processor Number	Simulation Result	Hypercube Result	Relative Error (%)
0	704.82 *ms*	679.71 *ms*	3.56
1	701.42 *ms*	676.69 *ms*	3.53
2	706.52 *ms*	675.78 *ms*	4.35
3	703.12 *ms*	669.23 *ms*	4.82

Table 5. Comparison between simulation and implementation results for the eight-prpcessor case.

Processor Number	Simulation Result	Hypercube Result	Relative Error (%)
0	379.70 *ms*	361.53 *ms*	4.79
1	330.08 *ms*	309.85 *ms*	6.13
2	383.12 *ms*	359.46 *ms*	6.17
3	428.04 *ms*	415.10 *ms*	3.02
4	406.98 *ms*	400.10 *ms*	1.70
5	376.50 *ms*	360.42 *ms*	4.27
6	376.16 *ms*	354.28 *ms*	5.82
7	374.50 *ms*	355.36 *ms*	5.11