as a node-edge-weighted undirected graph, called task graph. The task allocation problem becomes a problem of mapping the set of tasks to the set of processors such that the completion time is minimized, considering both processor load and communication overhead.

The main theme of our approach is to traverse a state-space tree that enumerates all possible task assignments. The key idea of the efficient task allocation algorithm is that we apply two pruning rules on each traversed state to check whether traversal of a given sub-tree is required by taking advantage of dominance relation and task clustering heuristics. The pruning rules try to eliminate partial assignments that violate the clustering on tasks but still keep some optimal assignments in the future search space. Experiment shows that our proposed pruning rule makes state-space searching approach feasible for practical use.

2. Modeling the Task Allocation Problem

2.1. Formulating the task allocation problem

We follow [1][2][3] to formulate the task allocation problem.

The input of a task allocation algorithm is a task graph $G(T, E, c)$ and a machine configuration $M(P, d)$. A parallel program is represented as a task graph $G(T, E, c)$ in which a node represents a program module, called a task, and an edge represents communication between tasks. Weight on a task, denoted $e(t_i)$, represents the execution time of the task and weight on an edge, denoted $c(t_{i,j})$, represents the amount of data transferred between the two tasks. The machine configuration is represented as $M(P, d)$. $P = (p_1, p_2, \ldots, p_m)$ is the set of all processors. For each pair of processors $(p_i, p_j) \in P$, a distance $d(p_i, p_j)$ is associated to represent the latency of transferring one unit of data between $p_i$ and $p_j$.

The output of the task allocation algorithm, called a complete assignment, is a mapping that maps the set of tasks $T$ to the set of processors $P$. An optimal assignment is a complete assignment with minimum cost. To find an optimal

1. Introduction

As part of the joint project “Study of Single Chip Multiprocessors Design,” the goal of this project is to optimize the benchmark program for the MP-chip based system. We investigate the task allocation problem of mapping a parallel program to a multiple MP-chip systems. A parallel program is modeled

---

**Abstract**— As part of the joint project “Study of Single Chip Multiprocessors Design,” we propose a task allocation algorithm that aims at finding an optimal task assignment for any parallel programs on the MP-chip based multiprocessor systems. The main theme of our approach is to traverse a state-space tree that enumerates all possible task assignments. The key idea of the efficient task allocation algorithm is that we apply two pruning rules on each traversed state to check whether traversal of a given sub-tree is required by taking advantage of dominance relation and task clustering heuristics. The pruning rules try to eliminate partial assignments that violate the clustering on tasks but still keep some optimal assignments in the future search space. In contrast to previous state-space searching methods for task allocation, the proposed pruning rules significantly reduce the time and space required to obtain an optimal assignment and lead the traversal to a near optimal assignment in a small number of states. Experiment shows that our proposed pruning rule makes state-space searching approach feasible for practical use.

**1. Introduction**

As part of the joint project “Study of Single Chip Multiprocessors Design,” the goal of this project is to optimize the benchmark program for the MP-chip based system. We investigate the task allocation problem of mapping a parallel program to a multiple MP-chip systems. A parallel program is modeled...
assignment, the branch-and-bound algorithm will go through several partial assignments, where only a subset of the tasks has been assigned. The cost of an partial complete assignment \( A \) is the turn-around time of the last processor finishes its execution. The turn-around time of processor \( p_i \), denoted \( TA_i(A) \), is the time to execute all tasks assigned to \( p_i \) plus the time that these tasks communicate with other tasks not assigned to \( p_i \), defined as follows:

\[
TA_i(A) = \sum_{t_j \in A} c(t_j) + \sum_{t_j \in A, t_k \in A} \sum_{t_l \in T} c(t_l, t_j) \cdot d(p_k, A(t_l))
\]

2.2. Transforming to the state-space searching problem

![State-space tree](image)

**Figure 1.** State-space tree

We traverse the state-space tree, as illustrated in Figure-1, to find an optimal assignment. During the traversal, an active set, denoted ActiveSet, is used to keep track of all partial complete assignments that have been explored but not visited. We follow the approach in Shen and Tsi [1] to determine the traversal order. For each partial complete assignment \( A \), a lower-bound (denoted \( LA(A) \)) on all complete assignments extended from \( A \) or \( A \) itself in case that \( A \) is a complete assignment) is estimated. The partial complete assignment in ActiveSet with minimum \( LA(\bullet) \) is removed for visiting in each iteration. \( LA(A) \) is computed according to the additional cost of assigning tasks not assigned in \( A \), defined as follows:

\[
AC_i(t_j \rightarrow p_k, A) = c(t_j) + \sum_{t_l \in A(t_l) \neq p_k} c(t_l, t_j) \cdot d(p_k, A(t_l))
\]

\[
AC_i(t_j \rightarrow p_i, A) = \sum_{t_l \in A(t_l) \neq p_k} c(t_l, t_j) \cdot d(p_k, p_i) \quad \text{if } p_k \neq p_i
\]

For a partial assignment \( A \), the cost lower-bound \( LA(A) \) for all complete assignments extended from \( A \) is estimated to be

\[
LA(A) = \max_{\text{processor } p_k} \left\{ \begin{array}{l}
TA_i(A) + \sum_{t_j \in A, t_k \in A} \sum_{t_l \in T} c(t_l, t_j) \cdot d(p_k, A(t_l))
\end{array} \right.
\]

3. Dominance Relation for State-Space Pruning

We first develop a dominance relation [6] to serve as the basis for developing pruning rules. The proposed dominance relation checks whether a partial assignment can be pruned or not according the estimated turn-around time difference lower-bound:

\[
TADL_i(A_i, A_j) = TA_i(A_i) - TA_j(A_j)
\]

\[
+ \sum_{t_j \in A_i} \min_{p_k \in P} (AC_i(t_j \rightarrow p_k, A_i) - AC_j(t_j \rightarrow p_k, A_j))
\]

**Theorem 1 (Dominance relation for space pruning).** Let \( A_i \) and \( A_j \) be two partial assignments assigning the same set of tasks. If \( TADL_i(A_i, A_j) \geq 0 \) for each processor \( p_k \), then \( A_i \) dominates \( A_j \).

4. Space Pruning by Detecting the Clustering on Tasks

The dominance relation proposed in Section 3 is only effective when a small cut can be found. To overcome this drawback, we develop a further pruning rule that integrates the detection of clustering on tasks as well as the dominance relation.

**Algorithm Prune Test \((A, A_i)\)**

- **input:**
  - \( A, A_i \): partial assignments.
  - depth\((A)\): depth\((A)\)
  - \( A_i \): a complete assignment
- **output:**
  - prune=True if \( A \) can be pruned, otherwise prune=False
- **method:**
  1) perform Compute PA\((A, A_i)\) to determine \( PA_i \)
  2) /* exclude extensions violating PA */
     21) success=False
     22) for each processor \( p_k \), do
        23) if \( TAL(A_i, A_i, P \cdot A) \cdot cost(A_i) \) then
           24) success=True
           break
     25) if success=False then \( PA_i \leftarrow \)
  3) \( A \) : the ancestor of \( A_i \) in the same level with \( A \)
  4) prune=False
  5) /* dominate extensions obeying PA */
     for each processor \( p_k \), do
        if \( TADL_i(A_i, A_i, P \cdot A) \cdot 0 \) then
           prune=False
           break
  6) return prune

**Figure 2.** Algorithm to examine the partial assignment using the pruning rule
Algorithm Compute_PA(A, AＡ)

- **input:**
  - A, AＡ: partial assignments, depth(A):
  - depth(A):

- **output:**
  - PA: for each task t not assigned in A (P is the set of all processors)

- **method:**
  1. If two or more partial assignments in the partial assignment A are visited, instead of only one killer. To obtain killers, a link to the deepest descendant node is associated with each visited partial assignment. For each visited partial assignment A, we associate a pointer deep(A) pointing to the deepest partial assignment visited in the sub-tree of A. If two or more partial assignments in the same level of the state-space tree are visited, deep(A) points to the first one visited. The KillerSet is the set of all deep(A) for each ancestor of A along with the complete assignment A.A.

Figure-3. Algorithm to predict the clustering on tasks

Figure-2 presents the algorithm to examine a partial assignment A. It calls procedure Compute_PA presented in Figure-3, to detect the task clustering. Two additional inputs are required: (1) partial assignment A, called the **killer**—reflecting the clustering on tasks, and (2) complete assignment A.A, serve as an upper bound on the optimal cost, which is obtained by the greedy search.

We determine whether the candidate partial assignment A can be pruned or not according to the following quantities: TAL(A) violate PA =

\[ TAL(A) = TAL_k(A, A.A) = TAL_k(A) - TAL_k(A.A) \]

\[ + \sum_{j \in \text{assigned in } A} \left( \min_{p_j \in P_j} AC_k(t_j \rightarrow p_j, A) \right) \]

\[ + \left( \sum_{p_j \in P_j} \min_{t_j \in \text{assigned in } A} AC_k(t_j \rightarrow p_j, A) \right) \]

The killers are obtained as follows. To increase the possibility of pruning a partial assignment, we may find multiple killers, called a **KillerSet**, instead of only one killer. To obtain the killers, a link to the deepest descendant node is associated with each visited partial assignment. For each visited partial assignment A, we associate a pointer deep(A) pointing to the deepest partial assignment visited in the sub-tree of A. If two or more partial assignments in the same level of the state-space tree are visited, deep(A) points to the first one visited. The KillerSet is the set of all deep(A) for each ancestor of A along with the complete assignment A.A.

The branch-and-bound algorithm for task allocation

To exploit the effectiveness of the pruning rule, tasks should be enumerated in such an order that tasks with high communication are enumerated first. This can be achieved by performing the max-flow min-cut algorithm recursively.

The branch-and-bound algorithm is described in Figure-4. Optimal assignment will be obtained if no overflow on the time and space required.

**Theorem 2 (Correctness of our proposed algorithm).** Our proposed branch-and-bound algorithm will end up with an optimal assignment if neither overflow on the time and space nor time-out occurs.
The ActiveSet is implemented as an array of heaps. To assign $n$ tasks to $m$ processors, the ActiveSet is a two dimensional array heap[$i$][$j$] for $1 \leq i \leq n$ and $1 \leq j \leq m$. A (partial) assignment assigning tasks $\{t_1, t_2, \ldots, t_m\}$ to $j$ of the $m$ processors is placed in heap[$i$][$j$]. The complexity of the branch-and-bound algorithm is controlled by the size of heap[$i$][$j$], denoted $size(i, j)$, which is a polynomial function of $i$ and $j$. When the number of (partial) assignments in the ActiveSet assigning $\{t_1, t_2, \ldots, t_m\}$ using $j$ processors exceeds $size(i, j)$, the one in heap[$i$][$j$] with maximum $L(*)$ will be dropped.

6. Experimental Evaluation

The performance and allocation quality are evaluated using 240 task graphs and three hierarchical machine configurations. On generating task graphs, the distribution on weights and edge density are chosen to cover all degree of clustering on tasks. On selecting the machine configuration, the processor distances are chosen such that the parallelism in optimal assignments ranges from using a few processors within a MP-chip to using all processors across multiple MP-chips.

We use the term performance to refer to the execution time that the task allocation algorithm spends to obtain an optimal assignment without time and space constraint. The metric is:

$$\text{Speed-up} = \frac{\text{number of states traversed by the A* algorithm}}{\text{number of states traversed by our proposed algorithm}}$$

The evaluation shows that the speed-up ranges from 1.03-2.20, depending on the degree of clustering on tasks and parallelism.

We use the term allocation quality to refer to how good the complete assignment returned by the task allocation algorithm is subject to time and space constraint. The metric is:

$$\text{Allocation quality} = \frac{\text{cost of the complete assignment returned}}{\text{cost of the optimal assignment}}$$

Time and space complexity are controlled by setting ActiveSet size and time-out threshold. In the experiment, the time-out threshold is set to be $\frac{n!}{m!}$, where $n$ is the number of tasks and $m$ is the number of processors, and the size of heap[$i$][$j$] is set to be $i^j$. Each test yields an allocation quality within 1.14.

7. Conclusion

In this report, we proposed a two-stage task allocation algorithm that aims at finding an optimal assignment. The first stage is a recursive partitioning procedure to form a task enumerating order such that we can exploit the task clustering property. The second stage is a branch-and-bound algorithm with pruning rule to traverse the state-space tree. The pruning rules keep some optimal assignments in the future search space and hence an optimal assignment will be obtained if neither time-out nor overflow on the ActiveSet occurs.

The key idea to the efficient task allocation is the pruning rule, which is a combination of a dominance relation and task clustering heuristic. The pruning rule reduces the time and space required to obtain an optimal assignment. Moreover, cooperated with the space efficient ActiveSet design, the traversal procedure can reach a near optimal assignment within a low order polynomial number of states.

The task allocation algorithm is evaluated on randomly generated task graphs. The experiment shows that our proposed pruning rule is effective to prune the search space and lead the traversal to a near optimal assignment within a low order polynomial number of states. This makes the state-space searching approach feasible for practical use.

Reference:


