ソフトウェア工学　94－4
（1993．9．21）

多重ループに対するプロセッサ割当てアルゴリズムOPTALの高速化

李 暁傑　　原田 賢一
li@hara.cs.keio.ac.jp　　harada@hara.cs.keio.ac.jp

慶應義塾大学大学院 理工学研究科 計算機科学専攻
〒223　横浜市　港北区　日吉 3-14-1

あらまし

並列処理の問題に、プログラムのどの部分をどのプロセッサに割り当てれば、並列計算機の速度を最大限発揮させせることができるのかという割当ての問題がある。任意の多重ループに対するプロセッサの最適割当てに関しては、OPTALと呼ばれるアルゴリズムがPolychronopoulosらによって提案されている。本論文では、このアルゴリズムを元に、多重ループの実行にはすべてのプロセッサを使用するという制約の下でOPTALの高速アルゴリズムを提案する。この方法による割当て結果は、従来の配置法による結果とほぼ一致し、割当ての効率が向上できたことを示す。

和文キーワード　並列化多重ループ、プロセッサ割当て、最適化、アルゴリズム

An Optimal Allocation of the Fixed Number of Processors to Nested Parallel Loops

Shiao-Chieh LEE　　Ken'ichi HARADA
li@hara.cs.keio.ac.jp　　harada@hara.cs.keio.ac.jp

Department of Computer Science, Graduate School of Science and Technology,
Keio University
3-14-1, Hiyosi, Kouhoku-ku, Yokohama, 223, Japan

Abstract

An important issue on the efficient use of multiprocessor system is how to allocate each processor to nested parallel loops. For parallelizing compilers, it is desirable at minimal cost to get allocations which reduce the execution time of parallel loops. In this paper, we propose an efficient algorithm for such allocation under the constraint that the fixed number of processors are fully used for parallel loops. This algorithm can be executed much faster than the existing algorithm that dose not always use all processors to minimize the total execution time of loops. Using this algorithm, the physical load balance among the processors can be guaranteed.

英文 key words　Nested parallel loop, Processor allocation, Optimization, Algorithm

1 Introduction

Automatic parallelization of sequential programs is an active research area. The reason is the need of a powerful parallel programming environment of the efficient use of multiprocessor computers. Such an environment should allow the programmer to free from the architecture details of the machine, and would be especially useful for large scientific programs. The parallelizing compiler is an efficient tool.

Intuitively, it might be said that increasing of the number of processors in a multiprocessor system will improve execution of parallel programs. It is observed, however, that existing commercial multiprocessor systems usually contain only a small number of processors. This fact is mostly due to the inexperience of using a large number of processors to execute a single parallel program efficiently. There are several important issues which need to be further studied. One of these important issues is the scheduling of independent processors to execute a single parallel program as fast as possible.

Unfortunately, no feasible solution exists for the general problem. Very efficient solutions are possible only for specific case [2][4][6][7]. Since parallel programs provide the greatest potential of parallelism to be exploited by multiprocessor systems, it is reasonable and effective to focus our attention on parallel loops. In this paper, we concentrate only on the problem of static processor allocation for a arbitrarily nested parallel loops. A processor allocation algorithm is used by the parallelizing compiler to decide the number of processors that need to be allocated to each individual loop, such that the parallel execution time of the entire loop construct is minimized. Although several other issues, such as memory management and data locality, also affect the performance of scheduling algorithms, we do not address these issuses in this paper.

The most important processor assignment algorithm in the literature is the algorithm OPTAL [1][3]. It has been claimed that OPTAL can generate optimal processor allocations if the loop bounds are known at compile-time. In many cases, however, the loop bounds are unknown at compile-time. This problem can be alleviated at run-time because loop bounds must be known before the loop is entered. Thus, it would be appropriate to perform processor allocation at run-time just before the beginning of execution of a nested loop. Unfortunately, the overhead incurred by the algorithm OPTAL may greatly degrade the performance of the multiprocessor system when there is a large number of processors.

Sometimes, what we are concerned with is how to execute a given parallel programs as fast as possible on a fixed number of processors but not how to use available processors as efficiently as possible executing a given parallel program. In other words only the optimal allocation of fixed number of processors to loops is needed. Based on this consideration, we propose an efficient processor allocation algorithm for nested parallel loops. Considering the use of fixed number of processors, the algorithm adopts less comparison of parallel execution time than OPTAL. Therefore, it is faster than the exsiting algorithm. By executing this algorithm in parallel, the time complexities can be further reduced.

The rest of this paper is organized as follows. Section 2 gives some basic background and necessary definitions. Important previous work is also reviewed in this section. Section 3 proposes a new processor allocation algorithm and parallel counterpart. Performance evaluation is shown in Section 4. Finally, Section 5 gives the conclusion of this paper.

2 Background

In this section, we shall give some basic information and necessary definitions. The algorithm OPTAL will be briefly reviewed in this section as well.

As mentioned in the previous section, we focus our attention on the execution of nested parallel loops on multiprocessor systems. In a parallel program, loops can be one of the following three types: Do loops (serial loops), Doall loops (all iterations of the loop can be executed in parallel), Doacross loops (successive iterations can be partially overlapped). The parallel programs will be executed on multiprocessor systems such that iterations of Doall and Doacross loops can be distributed over different processors. Furthermore, execution of loops of different types can be overlapped.

We introduce definitions and notations relevant to nested parallel loops. To simplify our notations, each loop is assumed to be normalized (i.e. its iteration space is of the form [1, ..., N], $N \in Z^+$). We may assume that any loop has the following form:

```
DO I = 1, N
    { B }
ENDDO
```

where I is the loop index, N is the loop bound, and B is the loop body. The loop body may contains other loops. If there is no other loop contained in the loop body, the Do loop is called an *innermost* loop. Otherwise, it is called an *outer* loop. In a nested loop, an individual loop can be enclosed by many outer loops. The *nest level* of an individual loop is equal to one plus the number of the enclosing outer loops. The *nest depth* of a nested loop is the maximum nest level of loops in the nested loop. In a perfectly (one-way) nested loop of nest depth m, there is exactly one loop at each nest level i (i = 1, 2, ..., m). Therefore, a perfectly (one-way) nested loop of nest depth m is a loop of the form

```
DO $I_1$ =1, $N_1$
  DO $I_2$ =1, $N_2$
    ......
      DO $I_m$ =1, $N_m$
          { B }
      ENDDO
    ......
  ENDDO
ENDDO
```

A loop is k-way if there exist k disjoint loops at the same level. For convenience, it is assumed that individual loops in an arbitrarily nested loop are numbered increasingly in lexicographic order. An arbitrarily nested loop containing m individual loops is denoted by $L_{1,m}$.

As mentioned above, a Doacross loop can be informally defined as a parallel loop in which data dependence allow for partial overlap of execution of successive iterations. That is, if iteration i starts at time t, on a given processor, iteration $i+1$ can start at time $t+d$, where d is a constant. The constant d is called the delay and represents the difference in time instant between initiations of successive iterations. If b is the serial execution time of the loop body B, the d/b is defined to be the overlap ratio . When $d = b$, the loop is serial, while if $d = 0$, the loop is reduced to a Doall. Therefore, serial loops and Doall loops are special cases of Doacross loops. The parallel execution time of a Doacross loop i on P processors is given by the following expression:

$$T_P^i(b_i) = (\lceil N_i/P\rceil - 1)\times max\{b_i, P\times d_i\} + d_i\times((N_i-1)\ mod\ P) + b_i \quad (1)$$

Where N_i is the loop bound, d_i is the delay, and b_i is the serial execution time of the loop body. An arbitrarily nested loop can be uniquely represented as a k level tree where k is the maximum nest depth. The leaves of the tree correspond to blocks of assignment statements (BAS's) in the nested loop and intermediate nodes correspond to Doacross loops. Accordingly, each intermediate tree nodes at level m correspond to loops at nest level m, respectively.

Now the allocation of processors to an arbitrarily nested parallel loops is introduced. First, consider the allocation of P processors to a perfectly nested loop of nested depth 2 with loop bounds N_1 and N_2, respectively. One possible processor allocation is to partition P processors into Q clusters where each cluster contains $R = \lfloor P/Q\rfloor$ processors and then assign Q clusters to the outer loop and R processors to the inner loop. Under this processor allocation, each time the inner loop is executed as if there are R processors availble. Note that the inner loop will be executed N_1 times – each time corresponds to an iteration of the outer loop. Since there are Q clusters of R processors, at most Q iterations of the outer loop can be executed simultaneously. This means that N_1 iterations of the outer loop can be executed as if there are Q "processor" is used generically in this paper and refers to clusters of physical processors of different sizes.

Since each time the inner loop is executed as if there are R processors available, each execution of the inner loop will take $T_R^2(b_2)$ execution time as given in (1). Similarly, each execution of the outer loop will take $T_Q^1(b_1)$ execution time. Because each execution of the loop body of the outer loop is an execution of the inner loop, we have $b_1 = T_R^2(b_2)$. Consequently, the parallel execution time of the above processor assignment should be $T_Q^1(T_R^2(b_2))$.

In general, an allocation of P processors to a perfectly nested loop of nest depth k can be represented as a k-tuple (P_1, ..., P_k) such that loop i is assigned P_i processors and for each path from the root node to a leave node the product of the number of processors assigned to loops in that path is less than or equal to P. Given a processor allocation, we can compute its parallel execution time. An optimal processor allocation is a processor allocation whose parallel execution time is minimum.

Algorithm OPTAL proposed in [3] can be used to find the optimal processor allocation. This algorithm is briefly outlined here. The assignment function $G_i^j(q)$ is defined to be the parallel execution time of an optimal allocation of q processors to loop j at level i. The algorithm contains two steps. During the first step we compute the parallel execution time of each innermost loop j on the tree as follows:

$$G_i^j(q) = T_q^i(b_j) \quad for\ \ q = 1, 2, ..., P \qquad (2)$$

where $T_q^i(*)$ is given by (1). The second step is defined recursively. Let k be the maximum nest depth. During the second step, the parallel execution time of the optimal allocation of q processors to loop j at level i is then computed as follows, here NJ is the number of children of loop j.

$$G_i^j(q) = min_{1\le r\le q}\ \{T_r^j(\sum_{n=1}^{NJ} G_{i+1}^n(\lfloor q/r\rfloor))\} \quad for\ \ q = 1, 2, ..., P \qquad (3)$$

where (3) is computed for all subtrees rooted at loop nodes j at level i, and $T_r^j(*)$ is given by (1). The summation in (3) accounts for all loop nodes at level $i+1$ that are descendants of loop node j, that is, all loops nested inside loop j. The parallel execution time of the optimal allocation of P processors to an arbitrarily nested loop $\mathrm{L}_{1,m}$ is given by $G_1^L(P)$. The detailed optimal processor allocation is automatically constructed during the evaluation of (3). For each loop j at level i, the optimal number of processors allocated to the minimum term in (3) and the number of processors allocated to loops nested in loop j is $\lfloor$q/r$\rfloor$. Hence, the optimal allocation can be constructed recursively.

It should be noted that all optimal allocations of 1, 2, ..., $P-1$ processors to an arbitrarily nested loop $\mathrm{L}_{1,m}$ are computed as intermediate results of the computation of $G_1^L(P)$. This can be used to find the maximum number of useful processors. A number of processors P is said that to be $useful$ with respect to an arbitrarily nested loop $\mathrm{L}_{1,m}$ if then exists an allocation which can allocate exactly P processors to the loops of $\mathrm{L}_{1,m}$. Given P processors for an arbitrarily nested loop $\mathrm{L}_{1,m}$, the maximum number of useful processors is the minimum Q, such that $1 \le Q \le P$ and $G_1^L(Q) = G_1^L(P)$. Accordingly, the maximum number of useful processors can be computed easily by the algorithm OPTAL. In Figure 1, we show the action of the OPTAL for a perfectly nested loop $\mathrm{L}_{1,2}$.

Let the set of possible allocations of q processors, denoded by A_q, be defined as { (r, s) | $s = \lfloor q/r\rfloor$ and r = 1, 2, ..., q }, then (3) can be rewritten as follows:

$$G_i^j(q) = min_{(r,s)\in A_q}\ \{T_r^j(\sum_{n=1}^{NJ} G_{i+1}^n(\lfloor q/r\rfloor))\} \quad for\ \ q = 1, 2, ..., P \qquad (4)$$

The computing time spent in (4) for each loop j at level i can be computed as follows:

$$\sum_{q=1}^{P} |A_q| = \sum_{q=1}^{P} q = \frac{P(P+1)}{2} = O(P^2) \qquad (5)$$

where $|A_q|$ denotes the number of elements in the set A_q.

As discribed above, all allocations of 1, 2, ..., $P-1$ processors to an arbitrarily nested loop $L_{1,m}$ are computed by the algorithm OPTAL as intermediate results of the computation of G_1^L. However, not all processor allocation algorithm necessarily possess this property. A processor allocation algorithm is *complete* if it computes all optimal allocation of 1, 2, ..., $P-1$ processors to an arbitrarily nested loop $\mathrm{L}_{1,m}$ as intermediate results of the computation of $G_1^L(P)$. Otherwise, it is *incomplete*.

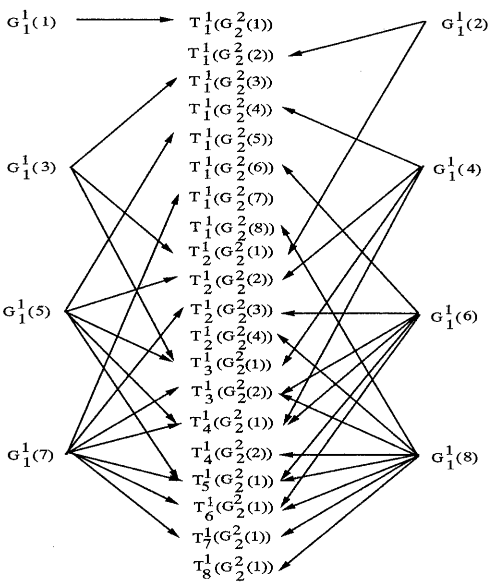

Figure 1: OPTAL algorithm for $L_{1,2}$

3 Allocating the fixed processors to nested loops

Sometimes, we take another point of view of processor allocation. When a dynamic allocation is required, the allocation algorithm must be faster than existing algorithm. In fact, we always allocate all physical processors to nested loops, because using all processors a high speedup also can be obtained [6][8], and algorithm becomes more practical. Under this circumstances, an incomplete algorithm is sufficient. Because incomplete algorithms compute only these allocations of P processors to $L_{1,m}$, they can be faster than complete algorithm. For example, the assignment function $G^j_i(q)$ of loop j at level 1 is useful only for $q = P$. Let the set of dominant numbers of q, denoted by D_q, be define as $\{ \lfloor q/r \rfloor \mid 1 \le r \le q \}$. From (3), we known that the assignment function $G^j_i(q)$ of loop j at level 2 is useful only for $q \in D_P$, and the assignment function $G^j_i(q)$ at level 3 is useful only for $q \in D_t$ where $t \in D_P$, and so on. At first glance, incomplete algorithm seems to be very complicated. However, the following theorems can greatly simplify the implementation of an incomplete algorithm.

Lemma 1 *D_s is a subset of D_q if $s \in D_q$.*

Proof: If $s \in D_q$, then by definition there exist r and t such that $q = rs + t$, $0 \le t < r$. For every $u \in D_s$,there exist v and w such that $s = uv + w$, $0 \le w < v$. Accordingly, we have $q = (uv + w)r + t = u(rv) + (rw + t)$. Since $0 \le rw + t < rv$, we can derive

$$u = \lfloor q/(rv) \rfloor \text{ and } u \in D_q$$

Therefore, for every $u \in D_s$, $u \in D_q$. Hence, D_s is a subset of D_q if $s \in D_q$. □

Theorem 1 *The assignment function $G^j_i(q)$ of any loop j at any level i, $i \ge 2$, is useful only for $q \in D_P$.*

Proof: We prove this using induction on the level i.
Basis: When i is equal to 2, this theorem is apparently true.
Hypothesis: Let us assume that when i is equal to k, the assignment function $G^j_k(q)$ of any loop j at level k is useful only for $q \in D_P$.
Inductive step: We shall prove this theorem is also true for $i = k + 1$. From (3) and the induction hypothesis, we known that the assignment function $G^j_{k+1}(q)$ of any loop j at level $k+1$ is useful only for $q \in D_t$ where $t \in D_P$. According to lemma 1, we known that for every $q \in D_t$ where $t \in D_P$, we have where $q \in D_P$. Therefore, the assignment function $G^j_{k+1}(q)$ of any loop j at level $k + 1$ is useful only for $q \in D_P$.

This completes the proof of this theorem. □

Theorem 1 insures that (3) is also useful in the set D_P of incomplete optimal allocation. However, in a optimal allocation, cluster p_i of $P = p_1 p_2 ... p_k$ must in the set D_P. We prove the property of this algorithm.

Lemma 2 *For each loop i, $T^i_r(b_i) \le T^i_s(b_i)$ if $r \ge s$.*

Proof: From (1) we know that when $r \ge s$,

$$(N_i - 1) \ mod \ r \le (N_i - 1) \ mod \ s$$

Therefore, $T^i_r(b_i) \le T^i_s(b_i)$. □

Theorem 2 *For each loop j at level i, there exists r, $r \in D_q$ such that*

$$G^j_i(q) = T^j_r(\sum_{n=1}^{NJ} G^n_{i+1}(\lfloor q/r \rfloor)).$$

Proof: Without loss of generality, we may assume that

$$G^j_i(q) = T^j_s(\sum_{n=1}^{NJ} G^n_{i+1}(\lfloor q/r \rfloor)).$$

By definition, we have $q = s \times \lfloor q/s \rfloor + (\ q \ mod \ s \ )$. Let r be the quotient of dividing q by $\lfloor q/s \rfloor$. Since $\lfloor q/s \rfloor \le q$, we have $r \in D_q$. Furthermore, since $(\ q \ mod \ s \ ) \ge 0$, we can conclude that $r \ge s$ and $\lfloor q/r \rfloor = \lfloor q/s \rfloor$. from lemma 2, we have

$$T^j_s(\sum_{n=1}^{NJ} G^n_{i+1}(\lfloor q/r \rfloor)) \ \ge \ T^j_r(\sum_{n=1}^{NJ} G^n_{i+1}(\lfloor q/r \rfloor))$$

This completes the proof of this theorem. □

Based on theorem 1 and 2, (3) can be rewritten as follow:

$$G^j_i(q) = min_{r \in D_q} \ \{T^j_r(\sum_{n=1}^{NJ} G^n_{i+1}(\lfloor q/r \rfloor))\} \quad for \ q \in D_P \qquad (6)$$

According to the (6), the incomplete algorithm is shown in Figure 3. As an example, the computation of the optimal allocation of 8 processors to a perfectly nested loop $L_{1,2}$ by the algorithm is shown in Figure 2. Here, $D_P = \{ 1, 2, 4, 8 \}$.

The time complexity is analyzed in the following.

Lemma 3 *For any positive integer q, we have*

$$D_q = \{r \mid 1 \le r \le \lfloor \sqrt{q} \rfloor\} \bigcup \{\lfloor q/r \rfloor \mid 1 \le r \le \lfloor \sqrt{q} \rfloor\}$$

and

$$|D_q| = \begin{cases} 2\lfloor \sqrt{q} \rfloor & if \sqrt{q} \ is \ not \ in \ Z^+ \ and \ q/\lfloor \sqrt{q} \rfloor \ge \lfloor \sqrt{q} \rfloor + 1 \\ 2\lfloor \sqrt{q} \rfloor - 1 & otherwise \end{cases}$$

Proof: From definition, $D_q = \{ \lfloor q/r \rfloor \mid 1 \leq r \leq q \}$, it is obvious that numbers of r ($1 \leq r \leq \lfloor\sqrt{q}\rfloor$) are successively in the set D_q. When r is greater than $\lfloor\sqrt{q}\rfloor$, the values of $\lfloor q/r \rfloor$ can be obtained from the quotients of dividing s by $\lfloor q/s \rfloor$ ($1 \leq s \leq \lfloor\sqrt{q}\rfloor$). Therefore,

$$D_q = \{r \mid 1 \leq r \leq \lfloor\sqrt{q}\rfloor\} \bigcup \{\lfloor q/r \rfloor \mid 1 \leq r \leq \lfloor\sqrt{q}\rfloor\}$$

When $\sqrt{q} \in Z^+$, hence the number $\sqrt{q}$ in subset $\{r \mid 1 \leq r \leq \lfloor\sqrt{q}\rfloor\}$ is equal to the number $\lfloor q/\sqrt{q} \rfloor$ in subset $\{\lfloor q/r \rfloor \mid 1 \leq r \leq \lfloor\sqrt{q}\rfloor\}$, $|D_q| = 2\lfloor\sqrt{q}\rfloor$ - 1. When $\sqrt{q}$ does not belong to Z^+ and $q/\lfloor\sqrt{q}\rfloor < \lfloor\sqrt{q}\rfloor + 1$, $|D_q|$ is the same as $\sqrt{q} \in Z^+$. Therefore,

$$|D_q| = \begin{cases} 2\lfloor\sqrt{q}\rfloor & \textit{if } \sqrt{q} \textit{ is not in } Z^+ \textit{ and } q/\lfloor\sqrt{q}\rfloor \geq \lfloor\sqrt{q}\rfloor + 1 \\ 2\lfloor\sqrt{q}\rfloor - 1 & \text{otherwise} \end{cases}$$

□

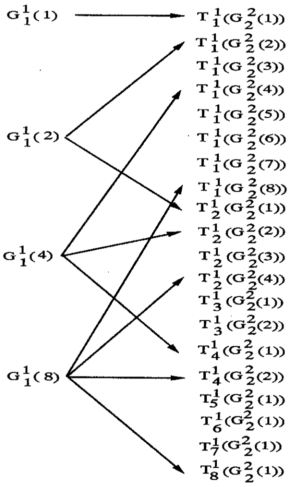

Figure 2: Incomplete allocation algorithm for $L_{1,2}$

```
INPUT:
    1. $L$ is a nested loop;
    2. $k$ is the nest depth of $L$;
    3. $P$ is the number of available processors.
OUTPUT:
    An incomplete optimal processor allocation.
METHOD:
    for each innermost loop $j$
        for $q \in D_P$ do
            begin
                $G_i^j(q) = T_q^j(b_j)$
            end;
    for $i = k-1$ to 1
        for each loop $j$ at level $i$
            for $q \in D_P$ do
              begin
              $G_i^j(q) = min_{r \in D_q} \ \{T_r^j(\sum_{n=1}^{NJ} G_{i+1}^n(\lfloor q/r \rfloor))\}$
              end;
    return the table of the optimal allocation.
```

Figure 3: The incomplete allocation algorithm

Theorem 2 em The time complexity of the algorithm is $O(P)$, where P is the number of processors.

Proof: From lemma 2, the computing time spent in (6) for each j at level i can be computed as follows:

$$\sum_{q \in D_P} |D_P| \leq \sum_{i=1}^{\lfloor\sqrt{P}\rfloor} 2\lfloor\sqrt{P/i}\rfloor + \sum_{i=1}^{\lfloor\sqrt{P}\rfloor} 2\lfloor\sqrt{i}\rfloor$$

$$\leq \sum_{i=1}^{\lfloor\sqrt{P}\rfloor} 2\sqrt{P/i} + \sum_{i=1}^{\lfloor\sqrt{P}\rfloor} 2\sqrt{i}$$

$$\leq 4 + \sum_{i=1}^{\lfloor\sqrt{P}\rfloor} 2\sqrt{P} = O(P)$$

This completes the proof of this theorem. □

The time complexity can be further reduced by executing the algorithm in parallel. Let us consider the computation of assignment function $G_i^j(q)$ at level i for loop j, the function is dependent on values of $2\lfloor\sqrt{q}\rfloor$ T_r^j's. Because $2\lfloor\sqrt{q}\rfloor$ ($q \in D_P$) is less than P, this step can be parallel executed. Theoretically the minimum of T_r^j's also can be obtained in parallel. Though the parallel execution, the evaluation of $G_i^j(q)$ becomes faster than the original. We give the parallel counterpart in Figure 4.

```
INPUT:
    1. $L$ is a nested loop;
    2. $k$ is the nest depth of $L$;
    3. $P$ is the number of available processors.
OUTPUT:
    An incomplete optimal processor allocation.
METHOD:
    for each innermost loop $j$
        for $q \in D_P$ do
            begin
                $G_i^j(q) = T_q^j(b_j)$
            end;
    for $i = k-1$ to 1
        for each loop $j$ at level $i$
            for $q \in D_P$ do
              begin-co
              $Reg_r = T_r^j(\sum_{n \text{ child of } j} G_{i+1}^n(\lfloor q/r \rfloor))$,   $r \in D_q$
              $G_i^j(q) = min \ \{ \ Reg_r \mid r \in D_q \ \}$
              end-co;
    return the table of the optimal allocation.
```

Figure 4: The parallel algorithm

The time complexity is analyzed in the following.

Lemma 3 *A decision of the minimum from M values can be done in $O(logM)$ on P ($P \geq M/2$) processors.*

Proof: Let the data set be $\{ \ r_1, r_2, ..., r_{m-1}, r_m \ \}$. At first step, compare r_1 with r_2 and replace the minimum to r_1, ...,

r_{m-1} with r_m and replace the minimum to r_{m-1} at parallel, here $M/2$ processors are necessary, then the set becomes { r_1, r_3, ...,r_{m-1} }. We use the same step recursively on the data set, and at $log_2(M/2)$ step the minimum of the set can be obtained. Therefore, the time complexity is $O(logM)$. □

Theorem 3 *The time complexity of parallel algorithm is $O(\sqrt{P}\ log\sqrt{P})$, where P is the number of processors.*

Proof: The computing time spent in (6) in parallel can be divided into two parts, one is the computation of T_i^j, another is the evaluation of minimum. Because the number of T_i^j's is less than P, the computing time is regarded as a constant. From lemma 3 and lemma 4, we have

$$\begin{aligned}\sum_{q\in D_P}|D_q| &= \sum_{i=1}^{\lfloor\sqrt{P}\rfloor} log(2i) + \sum_{i=1}^{\lfloor\sqrt{P}\rfloor} log(2\sqrt{P/i}) \\ &\le \sum_{i=1}^{\lfloor\sqrt{P}\rfloor}(log(i)+log(\sqrt{P/i})) \\ &= \sum_{i=1}^{\lfloor\sqrt{P}\rfloor}(log(i)/2+log(\sqrt{P})) \\ &\le \sum_{i=1}^{\lfloor\sqrt{P}\rfloor} log(i) \le \sum_{i=1}^{\lfloor\sqrt{P}\rfloor} log(\sqrt{P}) \\ &= O(\sqrt{P}log\sqrt{P})\end{aligned}$$

This completes the proof of this theorem. □

The algorithms shown above also constructs optimal processor allocations if the loop bounds are known at compile–time. This is frequently the case in numerical software where loop bounds usually reflects the problem size. However, there are cases where the loop bounds are not known at compile–time, it is possible to allocate processors dynamically using parallel counterpart of this algorithm.

4 Performance evaluation

In this paper, two allocation strategies were described: complete optimization algorithm OPTAL[3] and our incomplete optimization algorithm. The overall time complexity of latter is $O(P)$, which is derived by assuming that P processors are used. In fact, the claimed value reflects only the "observed" worst case. Consider the proof of Theorem 3, when P is a large number, $\sum_{i=1}^{\lfloor\sqrt{P}\rfloor} 2\lfloor\sqrt{P/i}\rfloor + \sum_{i=1}^{\lfloor\sqrt{P}\rfloor} 2\lfloor\sqrt{i}\rfloor$ is much less than $\sum_{i=1}^{\lfloor\sqrt{P}\rfloor} 2\lfloor\sqrt{P}\rfloor$, therefore, $O(P)$ is the threshold of the time complexity. The exact time complexity is fairly difficult to show. As a result, we give performance measures of two algorithm obtained through simulation runs, and compare the optimized allocations.

A. Algorithm Efficiency

It is clear that OPTAL and our strategy is different in the total of computations of $T_q^i(b_j)$ ($1\le q\le P$). This is the most important factor which affects the performance of the algorithms. In the other hand, both the two algorithms obtain the optimized allocations from a tree, in which $T_q^i(b_j)$ is a node. To get the $G_i^j(q)$'s, all $T_q^i(b_j)$'s should be saved. So memory allocating all mediate values must be considered here. In our strategy, the computations of $T_q^i(b_j)$ are significantly decreased, the time complexity and storage efficiency are much better than OPTAL. We plot the computation time for two algorithms in Figure 5. The storage efficiency is shown in Figure 6. It gives a ratio of memory consumption of OPTAL and incomplete algorithm.

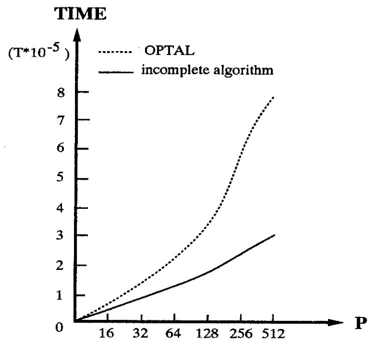

Figure 5: Performance of algorithms

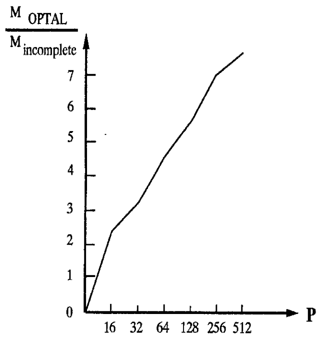

Figure 6: Efficiency of storage

From figures, it is obviously that our strategy always takes a considerably less time and memory.

B. Result Analysis

For two strtegies, perhaps the following question would be asked, how different are allocations between two allocating strategies? Indeed, sometimes allocation results are different, On strictly speaking, the allocations constructed by OPTAL is a "most" optimal result, but the allocations constructed by our strategy are very similar to that of OPTAL. Because allocation results are depend on nested depth, boundaries, synchronization method, and structure of nested loop, we cannot give a theoretical analysis. From a lot of numerical program, we choose some nested loops, based on which the allocation results of two strategies are compared. Here the following notations are used. SAM/3 means that a loop in program SAM is a 3-nested loop, 2-2-2 expresses that two clusters are assigned to each level of a 3-nested loop. Table 1 shows allocation comparison corresponding to different number of processors, the result of OPTAL is shown above the line, another is the result of our algorithm.

OPTAL ——— incomplete	P=16	P=32	P=64	P=128	P=256	P=512
CCD/2	8-2	8-4	8-8	8-16	16-16	16-32
	8-2	8-4	8-8	8-16	16-16	16-32
AMD/2	16-1	32-1	32-2	32-4	32-8	32-16
	16-1	32-1	32-2	32-4	32-8	32-16
SAM/3	2-8-1	2-2-8	63-1-1	2-4-16	2-8-16	2-8-32
	2-8-1	2-2-8	64-1-1	2-4-16	2-8-16	2-8-32
MDG/3	4-4-1	8-4-1	63-1-1	8-8-2	8-16-1	8-8-6
	4-4-1	8-4-1	64-1-1	8-8-2	8-16-2	8-8-8
SPICE/4	16-1-1-1	32-1-1-1	64-1-1-1	120-1-1-1	208-1-1-1	508-1-1-1
	16-1-1-1	32-1-1-1	64-1-1-1	128-1-1-1	256-1-1-1	512-1-1-1
ARC2D/4	1-1-1-16	1-1-1-32	1-1-1-63	1-1-1-128	1-1-1-255	1-1-1-510
	1-1-1-16	1-1-1-32	1-1-1-64	1-1-1-128	1-1-1-256	1-1-1-512

Table 1: Allocation comparison

Bounderies
of
Nested Loops
	N1=100
	N2=100
	N3=100
		N1=900
		N2=900
		N3=900
			N1=1700
			N2=1700
			N3=1700
				N1=2500
				N2=2500
				N3=2500

MDG/3(P=63)	0.965	1.521	2.274	2.644
MDG/3(P=64)	1.033	1.608	2.266	2.749

Table 2: Parallel execution time

From the benchmark suit, we have found that allocations of two algorithms are very similar, especially, when P is small, almost the same results are generated. The different allocations often occurred when P is a large number. By our observation, for simple allocations (all processors are assigned to one loop), the possibility of different results is higher than complex allocation (processors are assigned to more than one loop). In 80 loops observed by us, over 70% allocations are the same. We cannot give the all time comparisons between two different allocations. Here, only one case is simulated, which is MDG/3 at P=64. In OPTAL, the allocation is 63-1-1. It means that 63 clusters, one of which contains one processor, are assigned to the first loop. While our strategy is 64-1-1. The parallel performance time estimated for MDG/3 are shown in Table 2. It is found that the time of P=63 is faster than the time of P=64, but the gap of them is very small and can be accepted.

5 Conclusion

In this paper, we presented an algorithm for allocating processors to an arbitrarily nested parallel loop that parallel execution time is minimized. we have shown that the existing optimal processor allocation algorithm in the literature is inefficient. Based on an assumption that full processors are used, some computations are eliminated, the incomplete algorithm proposed in this paper is more efficient.

In addtion, we analyzed the algorithm efficiency and compared the allocations constructed by two algorithms. Although we only consider using all processors, our algorithm also yields a high efficient allocation. This alogrithm can be implemented in parallelizing compilers. It will result in improvements in compilation speedup.

References

[1] C.D. Polychronopoulos, D.J. Kuck and D.A. Padua : *Execution of parallel loops on parallel processor systems, in Proc. 1986 Int. Conf. Parallel Processing*, 1986, pp.519-535.

[2] C.D. Polychronopoulos and D.J. Kuck : *Guided self-scheduling: A practical scheduling scheme for parallel supercomputers, IEEE Tran. Comput*, Vol.36, No.12, pp.1425-1439, Dec. 1987.

[3] C.D. Polychronopoulos, D.J. Kuck and D.A. Padua : *Utilizing multidementional loop parallelism on large-scale parallel processor systems, IEEE Tran. Comput*, Vol. 38, No. 9, pp.1285-1296, Sept. 1989.

[4] J.A. Fisher, J.R. Ellis, J.C. Ruttenberg, and A. Nicolau : *Parallel processing: A smart compiler and a dumb machine, ACM SIGPLAN Notices*, Vol.19, No.6, pp.37-47, June 1984.

[5] K.P. Allen and P. Banerjee : *A scheduling algorithm for parallelizable dependent tasks*, in *Proc. 5th Int. Parallel Processing Symposium*, April 1991.

[6] N. Tawbi and P. Feautrier : *Processor allocation and loop scheduling on multiprocessor computers*, in *Proc. 1992 ACM Int. Conf. Supercomput.*, 1992, pp.63-71.

[7] R. Manner : *Hardware task/processor scheduling in a polyprocessor environment*, *IEEE Tran. Comput*, Vol.33, No.7, pp.626-636, July 1984.

[8] S. Hiranandani, K. Kennedy and C. Tseng : *Evaluation of Compiler Optimizations for Fortran D on MIMD Distributed -Memory Machines*, in *Proc. 1992 ACM Int. Conf. Supercomput.*, 1992, pp. 1-14.