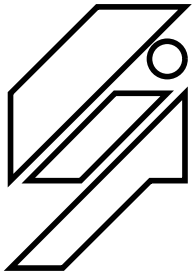
Task Ordering in Linear Tiles.

Fabrice Rastello, Amit Rao, Santosh Pande



Laboratoire de l'Informatique du Parallélisme

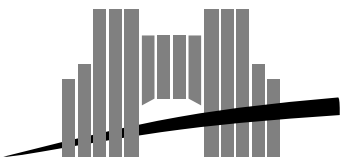
Ecole Normale Supérieure de Lyon
Unité de recherche associée au CNRS n°1398

Task Ordering in Linear Tiles

Fabrice RASTELLO
Amit RAO
Santosh PANDE

février 1998

Research Report N^{O} RR98-11



Ecole Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : (+33) (0)4.72.72.80.00 Télécopieur : (+33) (0)4.72.72.80.80
Adresse électronique : lip@lip.ens−lyon.fr

Task Ordering in Linear Tiles

Fabrice RASTELLO
Amit RAO
Santosh PANDE

février 1998

Abstract

In this report we address the issue of loop tiling to minimize the completion time of the loop when executed on multicomputers. We remove the restriction of atomicity of tiles and internal parallelism within tiles is exploited by overlapping computation with communication. The effectiveness of tiling is then critically dependent on the execution order of tasks within a tile. In this paper we present a theoretical framework based on equivalence classes that provides an optimal task ordering under assumptions of constant and different permutations of tasks in individual tiles. Our framework is able to handle constant but compile-time unknown dependences by generating optimal task permutations at run-time and results in significantly lower loop completion times. Our solution is an improvement over previous approaches and is optimal for all problem instances. We also propose efficient algorithms that provide the optimal solution. The framework has been implemented as an optimization pass in the SUIF compiler and has been tested on distributed and shared memory systems using a message passing model. We show that the performance improvement over previous results is substantial.

Keywords: automatic parallelization, tiling, nested loop, reordering, pipelined communications, uniform dependances, equivalence classes.

Résumé

Étant donné un nid de boucles 1-dimensionnel avec des dépendances uniformes et une distribution regulière des tâches sur une chaîne de processeurs. Nous adressons ici le problème du réordonnancement des tâches à l'intérieur même de chaque tuile afin de pipeliner les communications. En fait, nous cherchons à utiliser le parallélisme interne à chaque tuile afin de réduire la latence dans une direction critique ; ces résultats pouvant s'appliquer à des nids de boucles multidirectionnels. Les approches précedantes se tenant à chercher une permutation constante des tâches à l'intérieur de chaque tuiles, nous avons d'abord résolu ce problème de manière optimale (algorithme 3) puis comparé cet algorithme à un algorithme utilisant des permutations non constantes (algorithme 4). La construction de l'algorithme 3 à nécessité la mise en oeuvre d'une formalisation mathématiques du problème suivit de preuves substentielles. C'est ce qui constitue le corps de ce rapport.
Si clairement dans le cas 1-directionnel nos résultats montrent la supériorité de l'algorithme 4, certains paramètres laissent à penser que dans les dimensions supérieures, un algorithme de type 3 serait peut être plus efficace...

Mots-clés: parallélisation automatique, pavage, nids de boucle, réordonnancement, communications pipelinées, dépendances uniformes, classes d'équivalence.

Optimal Task Scheduling to Minimize Inter-Tile Latencies *

Fabrice Rastello
LIP, Ecole Normale
Supérieure de Lyon
46 allée d'Italie
69364 Lyon Cedex 07, France
fabrice.rastello@ens-lyon.fr

Amit Rao
Dept. of ECECS
University of Cincinnati,
Cincinnati, OH 45221
arao@ececs.uc.edu

Santosh Pande
Dept. of ECECS
University of Cincinnati,
Cincinnati, OH 45221
santosh@ececs.uc.edu

Abstract

In this paper we address the issue of loop tiling to minimize the completion time of the loop when executed on multicomputers. We remove the restriction of atomicity of tiles and internal parallelism within tiles is exploited by overlapping computation with communication. The effectiveness of tiling is then critically dependent on the execution order of tasks within a tile. In this paper we present a theoretical framework based on equivalence classes that provides an optimal task ordering under assumptions of constant and different permutations of tasks in individual tiles. Our framework is able to handle constant but compile-time unknown dependences by generating optimal task permutations at run-time and results in significantly lower loop completion times. Our solution is an improvement over previous approaches and is optimal for all problem instances. We also propose efficient algorithms that provide the optimal solution. The framework has been implemented as an optimization pass in the SUIF compiler and has been tested on distributed and shared memory systems using a message passing model. We show that the performance improvement over previous results is substantial.

*This work is supported in part by the National Science Foundation through grant no. CCR-9696129; by the CNRS–ENS Lyon–INRIA project *ReMaP*; and by the Eureka Project *EuroTOPS*

1 Introduction

One of the most popular techniques to partition uniform loop nests and map tasks to processors in a multicomputer is based on loop tiling. The motivation behind tiling is to increase the granularity of computations, locality of data references and data reuse [2, 14, 21]. Usually tiles are considered to be atomic *i.e.,* inter-processor communication is considered to take place only after the end of computation in each tile. A number of research approaches [1, 5, 12, 16, 17, 22] use this assumption to derive the optimal parameters *i.e.,* size and shape of the tile. However, the assumption of atomic tiles is needlessly restrictive when targeting parallel machines such as distributed memory multicomputers that are capable of overlapping communication with computation. Atomic tile considerations are justified when the cache size is small since tile size is chosen to fit the underlying data in the cache. However, if tiles are considered to be atomic, communication cannot be overlapped with computation. Therefore, intra-tile optimizations given that computation and communication can overlap poses an interesting problem.

Partitioning uniform loop nests to find communication free partitions or to minimize communication has motivated a significant amount of research recently [1, 17, 21]. These approaches involve distribution of data among processors and finding a communication optimal loop partition.

On the other hand we can avoid redistributing data among processors and order individual tasks on processors so as to minimize the completion time of the loop using tiling framework. In general, communication free or communication minimal partitioning of loops needs a more precise knowledge of dependences (such as dependence distance vectors); whereas tiling could be done with less precise dependence information (such as dependence direction vectors) [11]. The compiler can evaluate the loop completion time of the approach based on distributing data among processors and computing communication minimal loop partitions and compare it to an approach which avoids data redistribution and computes the optimal order of task execution on each processor [3]. Based on the above analysis, the compiler can then choose the corresponding technique. This indicates that there is a need to investigate the benefit of intra-tile optimizations.

Previous approaches by Chou & Kung [4] and Dion et al [10] to the problem of finding an optimal task ordering within loop tiles have relied on heuristics that do not yield the optimal solution for all instances of the problem even in one dimension. Thus, finding the *optimal* solution in one dimension is still an open issue. The central contribution of this work is to develop a new formulation of this problem along with a framework based on equivalence classes to derive optimal permutations. Using this framework we also develop efficient algorithms that result in an optimal solution for all problem instances in one dimension. Further, we take into account the possibility of having different task permutations in each tile and give the optimal solution for this case too.

The remainder of the paper is organized as follows. Section 2 presents the motivation through a simple example and the problem statement. Section 3 presents the related work and compares our solution with previous approaches. Section 4 formulates the problem based on equivalence classes. In section 5 we develop the theoretical framework, prove optimality results and give the algorithms. Section 6 deals with the more general case. Section 7 discusses the solution when different task permutations are considered in each tile. In section 8, we present and discuss the results of our implementation. Finally, section 9 provides concluding remarks.

2 Motivation and Problem Statement

2.1 Motivating Example

Consider the following example,
Example 1 :

```
Do l : 1 -> L
  Do i : 0 -> N
    A[i+1] <- f(A[i]);
  EndDo
EndDo
```

Let us suppose that we tile the `i`-loop in the above example and execute the loop nest on a multicomputer with processor caches. We could partition the above loop so that tasks with dependences execute on the same processor in order to fully eliminate communication of data involving array `A`. This can be achieved by allocating iterations `i` and `i+1` to the same processor. However, loop partitioning and data distribution will have to be done for every value of `l` which is very costly.

On the other hand, we could partition the iteration and data space using a method based on tiling which allocates consecutive iterations to the same processor. We do not redistribute data among processors. By exploiting the parallelism within loop tiles we overlap communication with computation. Assuming the tile size to be a constant across all processors for load balancing, the total loop completion time is critically dependent on the ordering of tasks within each tile. The task ordering cannot always be determined at compile-time for loops such as *example 1* where the dependences are unknown at compile-time. In such cases, it is necessary to determine the task ordering at run-time. Thus, determining optimal ordering of tasks within a tile to minimize the loop completion time for loops with constant but

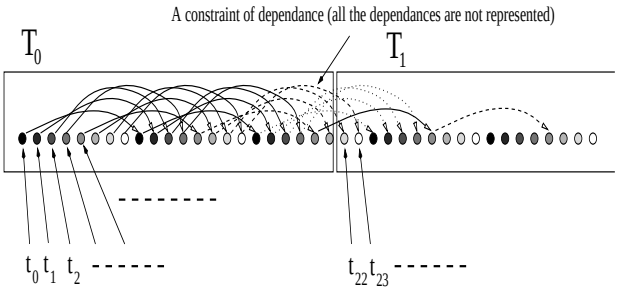


Figure 1: One dimensional tiling

unknown dependences is a key issue which is the focus of our work.

2.2 Statement of the problem

Consider a loop with loop size N, tile size n and dependence distance l. We assume that there is a single dependence distance vector and we have P processors. The N tasks are partitioned into $\lceil \frac{N}{n} \rceil$ tiles such that for $0 \leq i < \lceil \frac{N}{n} \rceil - 1$, the tasks contained in tile T_i are $\{t_{ni}, t_{ni+1}, ..., t_{n(i+1)-1}\}$ and $T_{\lceil \frac{N}{n} \rceil - 1} = \{t_{n(\lceil \frac{N}{n} \rceil - 1)}, ..., t_{N-1}\}$. Tile T_i is executed on processor P_i for every i. The problem is to find an optimal ordering (σ) of tasks in a tile, so as to minimize the overall completion time (T_{tot}) of the loop respecting the dependences imposed. The time to perform each task is τ_{calc} and the time required for each communication is τ_{comm}. Please note that the number of processors and tiles need not be the same.

To simplify the notation, consider an example in which there are 700 tasks $t_0, t_1, ..., t_{699}$ with a flow dependence between t_i and t_{i+8} for all i. In order to take advantage of temporal data locality we use tiling as discussed above. Let us assume that the tile size (n) is 22 and number of processors (P) is 32. For $0 \leq i \leq 30$, tile T_i contains the tasks $\{t_{22i}, t_{22i+1}, t_{22i+2}, ..., t_{22(i+1)-1}\}$ and $T_{31} = \{t_{682}, t_{683}, ..., t_{699}\}$. By executing T_i on processor P_i for every i, the loop space is mapped onto the processor space. Figure 1 represents the constraints of dependences for the first two tiles.

We now present a survey of previous work related to solving this problem and explain the significance of our contribution.

3 Related Work

Tiling bas been the focus of loop partitioning work in compilers recently. Many different approaches investigate different aspects of tiling such as determining size and shape of a tile, determining legality of tiled space to enforce dependences, use of tiling to minimize communication etc. We present a brief overview of some of the important recent tiling approaches below and contrast our work with them.

Irigoin and Triolet [11] and Jingling Xue [23] address the issue of legality of tiling. They introduce the notion of loop partitioning with multiple hyper-planes. They propose a set of basic constraints that should be met by any partitioning and derive the conditions under which the hyperplane partitioning satisfies the constraints.

Agarwal et al [1] address the problem of deriving the optimal tiling parameters for minimal communication in loops with general affine index expressions on cache-coherent multiprocessors. They assume tiles to be atomic and only consider DOALL loops factoring out issues of dependencies and synchronizations that arise from the ordering of iterations of a loop. Our work is not restricted to DOALL parallelism.

Ramanujam & Sadayappan [17] address the problem of compiling perfectly nested loops for multicomputers using tiling. However, they consider tiles to be atomic and therefore do not allow synchronization during the execution of a tile. They present an approach to partitioning the iteration and data space so that communication is minimized.

Tang and Zigman [20] use loop tiling coupled with chain-based scheduling and indirect message passing to develop efficient message passing parallel code for DOACROSS loop nests. Their work mainly deals with optimizing communication arising out of tiling which is not the focus of our work.

Desprez et al [7] investigate the cumulative idle time of processors during a parallel execution.

They study perfect loop nests with uniform dependences and determine the solution to parameters that relate the shape of the iteration space to that of the tiles, for all distributions of tiles to processors. Their work is different from ours as they assume tiles to be atomic.

D'Hollander [8] presents a partitioning algorithm to identify independent iterations in loops with constant dependence vectors. He also proposes a scheduling scheme for the dependent iterations. The iterations in each set are ordered lexicographically.

The focus of our work is on intra-tile optimizations to minimize the loop completion time. More specifically, we focus on the problem of determining optimal permutations of tasks within a tile to minimize loop completion time. The problem of determining permutations of tasks within a tile to minimize loop completion time has been attempted by Chou & Kung [4] and by Dion et. al. [9, 10]. Before discussing their approaches we introduce some terms and definitions.

3.1 Terms and Definitions

We denote the tile length by n, loop size by N and the dependence distance by l.

Definition 1 *Let us denote $\tau = \{t_0, t_1, ..., t_{N-1}\}$ as the task space. Each element of τ is a task to be executed.*

Definition 2 *The iteration space is mapped onto the processor or tile space. Each tile contains n tasks ; more precisely, a tile is a set $T_j = \{t_{nj}, t_{nj+1}, t_{nj+2}, ..., t_{nj+n-1}\} \cap \tau$ with $j \in \{0, ..., \lceil \frac{N}{n} \rceil - 1\}$. All the tasks of one tile are to be executed on one processor only.*

Let τ_{calc} be the time to perform one task with one processor, and let τ_{comm} be the time for one communication between two adjacent processors. The g.c.d of n and l is denoted by $n \wedge l$. P denotes the number of processors. k denotes $n \bmod l$ while p denotes $\lfloor \frac{n}{l} \rfloor$. Finally,

$$x \equiv y[l] \Longleftrightarrow x \bmod l = y \bmod l$$

Definition 3 *T is called the period of ordering such that for each i, tile T_i is executed from time iT to time $iT + n\tau_{calc}$.*

Since, the total loop completion time depends linearly on the value of T, the problem reduces to minimizing T.

Definition 4 *In each tile, the permutation of tasks is equal modulo n. Let σ denote the permutation of tasks in tile T_0.*

$$\{0, 1, ..., n\} \xrightarrow{\sigma} \{0, 1, ..., n\}$$

We have for all $i \in \{0, 1, ..., N-1\}$, t_i is executed by the processor $j = \lfloor \frac{i}{n} \rfloor$ between time $\tau_i = jT + \sigma(i - nj)\tau_{calc}$ and the time $\tau_i + \tau_{calc}$.

Definition 5 *For a given n, and a given l, and τ_{calc} and τ_{comm} being fixed, the minimum period of ordering T_{min} is the smallest reachable value of T satisfying the above constraints with an appropriate permutation within a tile.*

3.2 Chou & Kung's Solution

Let us first discuss Chou and Kung's [4] solution to the one-dimensional tiling problem. Chou and Kung propose the following ordering of tasks on each tile (processor),
For all $i \in \{0, .., N-1\}, j = \lfloor \frac{i}{n} \rfloor$ (we have $nj \leq i \leq nj + n - 1$), t_i is executed by processor P_j between time $\tau_i = (i - nj) \times \tau_{calc} + j \times T$ and $\tau_i + \tau_{calc}$. The period of ordering is given by

$$T = (n - l + 1) \times \tau_{calc} + \tau_{comm}$$

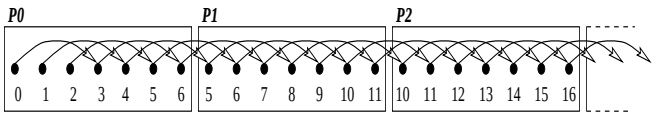
Hence, for all i, the processor P_i starts working at time $i \times T$ and ends working at time $i \times T + n \times \tau_{calc}$.

Chou & Kung's solution for the case $n = 7$ and $l = 3$ is presented in Figure 2.

In Figure 2 we use the following notation,

1. Each • represents a task to be executed. The tasks are numbered from left to right in increasing order starting at 0.

2. A rectangle is a tile.



Case n=7, l=3 & T=5tcalc+tcom

Figure 2: Chou & Kung's solution for the case $n = 7$ and $l = 3, T = 5$

3. The set of tasks contained in a tile are executed on the same processor.

4. The numbers below each task represent the times at which the tasks are executed, considering $\tau_{calc} = 1$ and $\tau_{comm} = 0$.

5. The arrows between tasks denote the dependences between tasks. For example, $t_1 \longrightarrow t_2$ means that t_1 must be executed before t_2.

As one can see their approach is non-optimal. Dion improved over Chou and Kung's solution by considering a better ordering of tasks within each tile so as to decrease the period T of ordering.

3.3 Dion's solution

First Dion shows the following results,

Lemma 1 *The best local permutation that permits reaching the minimum period T_{min} is independent of the values of τ_{calc} and τ_{comm}. Moreover, if $T^{1,0}_{min}$ is the minimum period for $\tau_{calc} = 1$ and $\tau_{comm} = 0$ then $T^{\tau_{calc},\tau_{comm}}_{min} = T^{1,0}_{min} \times \tau_{calc} + \tau_{comm}$.*

Lemma 2 *If $n \wedge l \neq 1$ the problem is equivalent to a smaller problem with $n' = \frac{n}{n \wedge l}$ and $l' = \frac{l}{n \wedge l}$.*

Hence, from now on we can choose n and l such that $n \wedge l = 1$, τ_{calc} will be equal to 1, and τ_{comm} will be 0, so that the discussion is simplified.

Dion's permutation leads to a smaller period of ordering than Chou & Kung's solution. In fact, her solution is optimal for the special case $l = 2$. For example, for the case $n = 9$ and $l = 2$ Dion's solution leads to $T = 7$ which is the optimal solution, while Chou and Kung's solution leads to

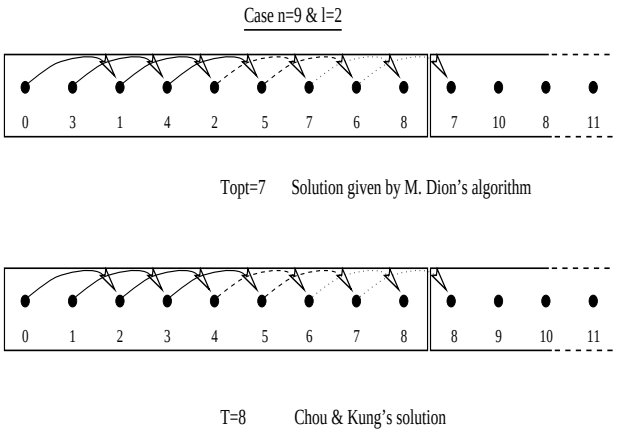


Figure 3: Comparison of Dion's solution and Chou & Kung's solution for $n = 9$ and $l = 2$

$T = 8$. Larger the value of n, greater will be the difference between the two solutions and thus, the loop completion times. The solutions are presented in Figure 3.

The following theorem summarizes Dion's contribution for the special case $l = 2$,

Theorem 1 *For $n = 2k + 1$, $k > 0$ and $l = 2$, the optimal ordering has a period of ordering,*

$$T_{opt} = \lceil \frac{3n-1}{4} \rceil$$

Dion also gives an algorithm for ordering the tasks for this special case in order to reach the optimal period of ordering. Let $j \in \{0, \cdots n-1\}$.

- for j odd, $\tau_j = \lceil \frac{3n-1}{4} \rceil - \frac{n-j}{2}$
- for j even,
  - if $j \leq 2(\lceil \frac{3n-1}{4} \rceil + 1) - n, \tau_j = \frac{j}{2}$
  - if $j > 2(\lceil \frac{3n-1}{4} \rceil + 1) - n, \tau_j = \frac{j+n-1}{2}$

She also derives bounds for T which is stated in the following theorem,

Theorem 2 *For $l \geq 3$ and $n \wedge l = 1$, we have*

$$2\lfloor \frac{n}{l} \rfloor - 1 \leq T \leq 2\lfloor \frac{n}{l} \rfloor + 2$$

For $l \geq 3$, Dion gives an algorithm called a *cyclic* algorithm that gives a correct permutation or schedule with $T = 2\lfloor \frac{n}{l} \rfloor + 2$. However, this is

not an optimal solution. This is one of the major limitations of the solution given by Dion. The other limitation is that she has used the simplifying assumption of a constant permutation in every processor similar to Chou and Kung's work. In later sections we show that the solution can be greatly improved if we remove this restriction.

Dion's and Chou and Kung's solutions are not optimal for all instances of the problem. Figures 4 & 5 compare their solution to ours, for the case $(N = 700, n = 22, l = 8)$. Figures 4(a) and 4(b) show Chou & Kung's solution and Dion's solution. Figures 5(a) and 5(b) show solutions obtained by our two proposed algorithms. The tasks are represented by dots. The horizontal lines correspond to the starting times of each individual tile. The vertical lines represent the boundaries of adjacent tiles. Tasks that have dependences are shaded in the same color.

However, determining the optimal solution for both constant and non-constant permutations is a non-trivial combinatorial problem as outlined below.

3.4 Complexity of the Problem

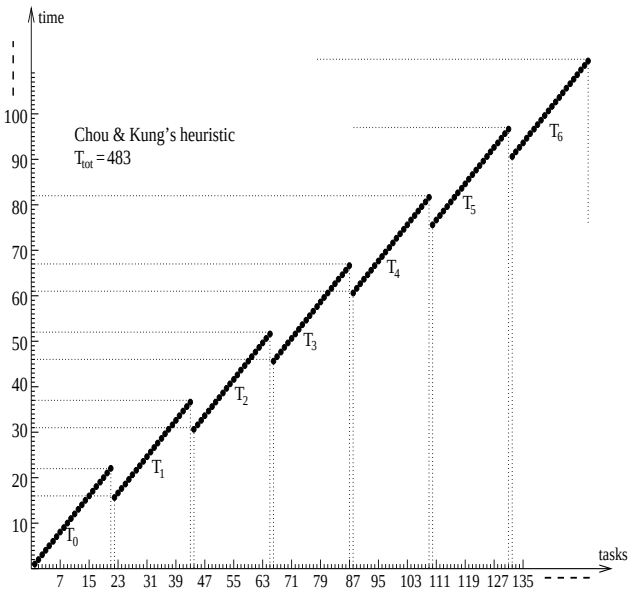
We first examine the complexity of the problem of determining optimal permutations of tasks within a tile.

Given a tile size (n) and a dependence distance (l), we calculate the total number of possible permutations corresponding to different orderings of tasks within a tile. For $n = 22,\ l = 8$, the dependence graph for a single tile is shown in Figure 6.

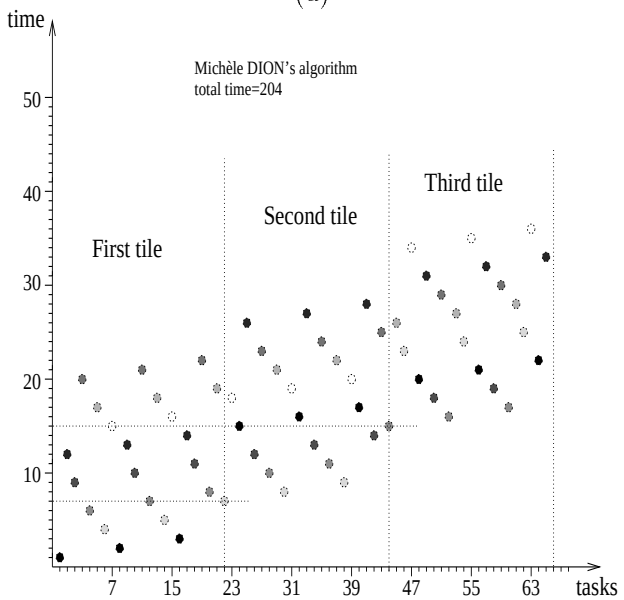
The dependence graph has l connected components. Let $p = \lfloor \frac{n}{l} \rfloor$ and $k = n \bmod l$. Then, k components contain $p + 1$ tasks, and $l - k$ components contain p tasks.

To begin with, there are $n!$ possible permutations of n tasks without considering dependences. The tasks within each component have to be executed in increasing order to satisfy dependency constraints. The total number of correct permutations taking dependences into account is

$$\frac{n!}{((p+1)!)^k (p!)^{l-k}} = \frac{n!}{(p+1)^k (p!)^l}$$



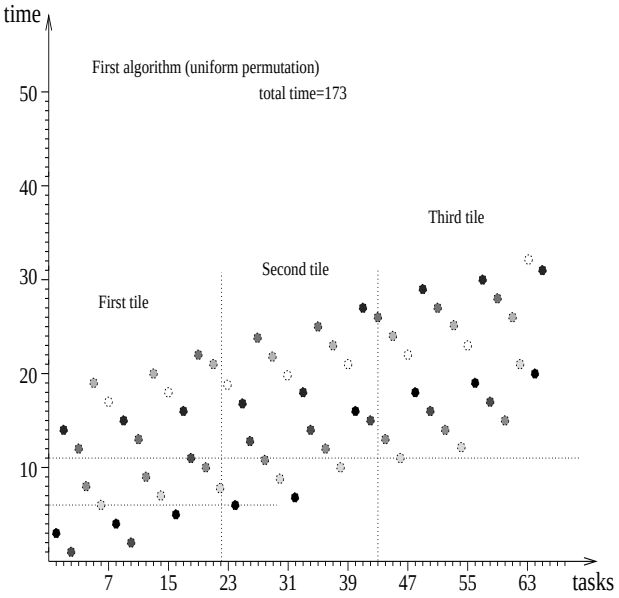
(a)



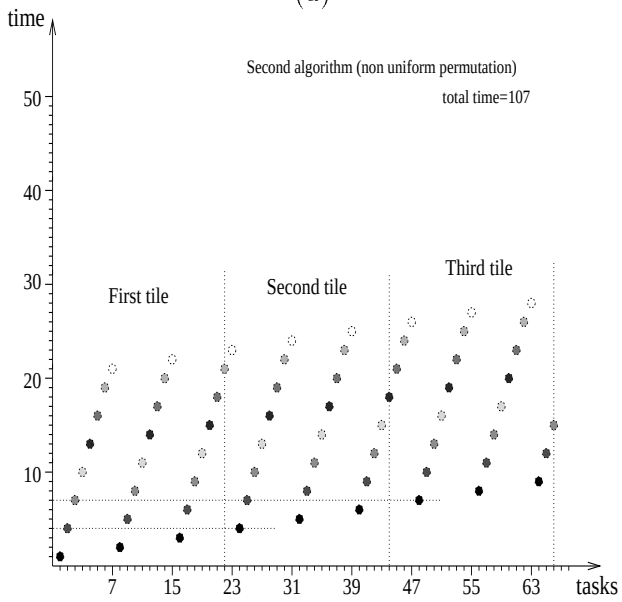
(b)

Figure 4: Timing graph showing solution obtained from (a) Chou & Kung's algorithm (b) Dion's algorithm $(N = 700, n = 22, l = 8)$

.



(a)



(b)

Figure 5: Timing graph showing solution obtained from (a) Algorithm I (b) Algorithm II ($N = 700, n = 22, l = 8$).

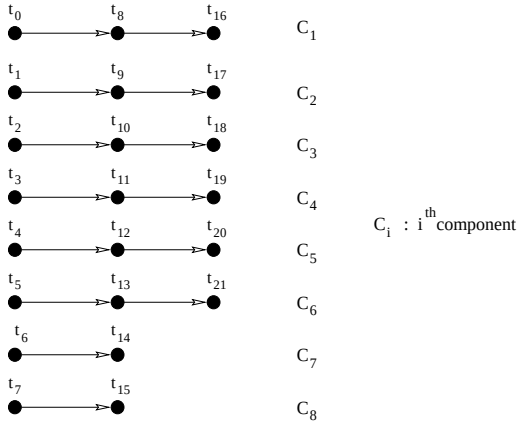


Figure 6: Task dependence graph for tile at origin

For $n = 22$, $l = 8$, the number of possible permutations is 6.02×10^{15}. Since this is a very large number even for a small problem, the problem is combinatorial in nature. In order to make it more tractable, a suitable framework must be developed to solve it in polynomial time.

In order to derive the optimal solution in polynomial time, we formulate the problem in a different way. We then develop a new framework that helps us derive the optimal period of ordering T_{min}, for all cases of l, in terms of n, k and l, where $n = lp + k$ & $0 \leq k < l$. The framework also gives us an efficient algorithm that always reaches the optimal period of ordering T_{min}.

4 Formulation of the Problem using Equivalence Classes

We introduce the following notation and definitions for our subsequent discussion,

1. The set of tasks of the first tile is given by $\Omega = \{t_0, t_1, ..., t_{n-1}\}$. Further, we say that $t_i \equiv t_j$ iff $i \equiv j[l]$. Hence, by using a simplified notation, we denote task t_i by the integer i.

2. The operator $\equiv$ is an equivalence relation that defines equivalence classes within $\{0, 1, ..., n-1\}$. This equivalence relation defines l components or equivalence classes,

$X_0, X_1, ..., X_{l-1}$. We denote the set of equivalence classes by $\Psi = \{X_0, X_1, ..., X_{l-1}\}$.

3. We say that, $X \rightarrow Y$ if,

$$\exists (x, y) \in (X, Y) \cdot x \equiv y + n[l]$$

4. The indices for the equivalence classes are chosen in a manner such that,

   - $X_0 = \{i \in \Omega \cdot i \equiv 0[l]\}$
   - $X_0 \rightarrow X_1 \rightarrow X_2 \rightarrow \cdots \rightarrow X_{l-1} \rightarrow X_0$

From the following definitions we have,

$$\forall X_i \in \Psi, p \leq |X_i| \leq p + 1$$

Moreover, there are k classes of size $p+1$ & $l-k$ of size p.

Definition 6 *For all i in $\{0, 1, \cdots, l-1\}$, we define $\lambda_i = |X_i| - p$. In other words, if $|X_i| = p+1$ then $\lambda_i = 1$, else $|X_i| = p$ and $\lambda_i = 0$.*

Definition 7 *For all $i \geq l$, $\lambda_i = \lambda_{i \bmod l}$*

Therefore, the $\lambda-$string can be extended to an infinite string (periodic with period l).
Example 2 : Consider $n = 7$ and $l = 3$. We have,

$$X_0 = \{0, 3, 6\}\ ,\ \lambda_0 = 1$$
$$X_1 = \{2, 5\}\ ,\ \lambda_1 = 0$$
$$X_2 = \{1, 4\}\ ,\ \lambda_2 = 0$$

The problem of finding an optimal ordering of tasks now reduces to finding a suitable permutation of the l equivalence classes. This permutation depends on the relative values of λ_i *i.e.,* the property of the string $\lambda_0 \lambda_1 \lambda_2 \cdots \lambda_{l-1} \cdots$ We derive the optimal solution in the next section.

5 Optimal Solution

5.1 Formal Framework

Let us recollect that the tile size is n, the dependence distance is l and $n \wedge l = 1$. The equivalence classes are called X_i and their size is $p + \lambda_i$, with $p = \lfloor \frac{n}{l} \rfloor$. We claim that the optimum period of ordering depends on the relative values of λ_i, more specifically λ_i and λ_{i+1}. Hence, the aim of theorem 3 is to discuss the property of the string $\lambda_0, \lambda_1, ..., \lambda_{l-1}$..... in terms of the tile size (n) and the dependence distance (l). This relationship is quantified in terms of Λ presented below.

Theorem 3 *If $l \geq 3$, $n \wedge l = 1$ and $\Lambda = \min_{i \in N}[\max_{i+1 \leq j, j+1 \leq i+l-1}(\lambda_j + \lambda_{j+1})]$, then*

- *If $l = 4$ & $k = 3$ then*

$$\Lambda = \max_{2 \leq j, j+1 \leq 4} (\lambda_j + \lambda_{j+1}) = 1$$

- *Else*

$$\Lambda = \max_{1 \leq j, j+1 \leq l-1} (\lambda_j + \lambda_{j+1}) = \left| \begin{array}{ll} 0 & \text{if } k = 1 \\ 1 & \text{if } 1 < k \leq \lceil \frac{l}{2} \rceil \\ 2 & \text{if } k > \lceil \frac{l}{2} \rceil \end{array} \right.$$

□

As seen above, the definition of Λ is a min-max definition. The above theorem gives the values of Λ for all cases of tile size and dependence distance. The relationship between the value of Λ and the optimal period of ordering (T_{min}) is presented in theorem 4. Thus, theorem 3 and theorem 4 together allow us to determine the lower bound on the period of ordering (T) given tile size (n) and dependence distance (l).

We now illustrate the result of theorem 3 through some examples.

Example 3 : Consider $n = 7$, $l = 3$ ($k = 1$)
$\lambda -$ string $= 1\mathbf{00}100100...$
$\Lambda = 0$.
Example 4 : Consider $n = 7$, $l = 4$ ($k = 3$)
$\lambda -$ string $= 11\mathbf{101}110...$
$\Lambda = 1$
Example 5 : Consider $n = 7$, $l = 5$ ($k = 2 < \lceil \frac{5}{2} \rceil$)
$\lambda -$ string $= 1\mathbf{0100}10100...$
$\Lambda = 1$
Example 6 : Consider $n = 9$, $l = 5$ ($k = 4 > \lceil \frac{5}{2} \rceil$)
$\lambda -$ string $= 1\mathbf{1110}11110...$
$\Lambda = 2$

The Λ values calculated above for different tile sizes and dependence distances allow us to determine the best period of ordering as per theorem 4

discussed in section 5.2. In order to prove theorem 3 we need a few results.

Lemma 3 gives us a working definition for X_i.

Lemma 3 *If* $X_i' = \{x \in \Omega \mid \exists \alpha \in I\!N \cdot (x \equiv \alpha l[n]) \wedge (in \leq \alpha l < (i+1)n)\}$, *then* $\forall i,\ X_i' = X_i$.

Proof Refer [18]

In order to prove *theorem 3*, we have to differentiate between the cases $\Lambda = 1$ and $\Lambda = 2$. For this we need to discuss whether there exist two different integers i & j in $\{0, 1, ..., l-1\}$ such that $\lambda_i\lambda_{i+1} = \lambda_j\lambda_{j+1} = 11$.

The next lemma formalizes this discussion. It states the condition that the tile size and the dependence distance should satisfy in order to have a sub-string 11 within the λ-string.

Lemma 4

$$\begin{aligned} \exists i \in \{1, ..., l-1\} \mid \lambda_i\lambda_{i+1} = 11 &\iff 2k > l+1 \\ &\iff k > \lceil \frac{l}{2} \rceil \end{aligned}$$

Proof Refer [18]

In order to prove theorem 3, we have to differentiate between the cases $\Lambda = 1$ and $\Lambda = 2$ (the case $\Lambda = 0$ is trivial). To achieve this task we need to discuss whether there exists i in $\{2, \cdots, l-1\}$ such that $\lambda_i\lambda_{i+1} = 11$ or not. Indeed, we will see that if $k > \lceil \frac{l}{2} \rceil$ then $\lambda_0\lambda_1 = 11$ and that $\lambda_{l-1} = 0$.

To formalize it, we need to introduce a new concept - the property of a string to be well balanced.

Let $\sum = \{0, 1\}$ be the alphabet of the λ-string. Let uv be the concatenation of the strings u & v (if $u = u_0u_1u_2.....u_n$ & $v = v_0v_1.....v_m$ then $uv = u_0u_1.....u_nv_0v_1.....v_m$).

Definition 8 (sub-string) *v is said to be a sub-string of u, if there exist two strings α & β, such that $u = \alpha v \beta$*

Definition 9 (length) *The length of a string $u = u_1u_2u_3....u_n$ is the integer n denoted by $|u|$.*

Definition 10 (weight) *Let $\alpha \in \sum$. If $u = u_1u_2.....u_n \in \sum^*$ is a string of length n, then $|u|_\alpha = |\{i \in \{1, ..., n\}, u_i = \alpha\}|$.*

Definition 11 (well-balanced) *Let $u \in \sum^*$. u is said to be well-balanced if for any pair of sub-strings of u, (v, v'),*

$$|v| = |v'| \Longrightarrow ||v|_1 - |v'|_1| \leq 1$$

Lemma 5 *The infinite string $\lambda = \lambda_0\lambda_1\lambda_2.....$ is well-balanced.*

Proof Refer [18].

Now we can easily prove *theorem 3*,

Proof: First, note that $\lambda_0 = 1$. We have, $|X_0| = |X_0'| = |\{\alpha \in I\!N, 0 \leq \alpha l < n\}| = p+1$. Also $\lambda_{l-1} = 0$.

$$\begin{aligned} |X_{l-1}| &= |\{\alpha \in I\!N, (l-1)n \leq \alpha l < ln\}| \\ &= |\{\alpha \in I\!N, (l-1)n \leq \alpha l \leq ln\}| - 1 \\ &\leq (p+1) - 1 \end{aligned}$$

1. Case $k = 1$:
   - As $k = |\lambda_0\lambda_1....\lambda_{l-1}|_1 = 1$ & $\lambda_0 = 1$ therefore, $\Lambda \leq \max_{1 \leq p, p+1 \leq l-1}(\lambda_p + \lambda_{p+1}) = 0$
   - Consequently, $\Lambda = 0$.
2. Case $1 < k \leq \lceil \frac{l}{2} \rceil$:
   - $\forall i, |\lambda_i\lambda_{i+1}....\lambda_{i+l-1}| = k \geq 2$, therefore, $\forall i, \exists p \in \{i, ..., l+i-2\} \mid \lambda_p = 1$ Hence, $\Lambda \geq 1$.
   - From the proof of *lemma 4*, we have
     $$\Lambda \leq \min_{1 \leq p, p+1 \leq l-1} \lambda_p + \lambda_{p+1} \leq 1$$
   - Consequently, $\Lambda = 1$.
3. $l > k > \lceil \frac{l}{2} \rceil$:
   - We have $2n = 2pl + 2k \geq (2p+1)l + 1$. Therefore,
     $$\begin{aligned} |X_0'| + |X_1'| &= |\{\alpha \in I\!N, 0 \leq \alpha l < 2n\}| \\ &= 2p+2 \end{aligned}$$
     Hence, $\lambda_0\lambda_1 = 11$
   - Suppose that *there exists* i *in* $\{2, 3, ..., l-1\}$ such that $\lambda_i\lambda_{i+1} = 11$. Since $i \neq l-1$ this will lead to $\Lambda = 2$

- Suppose that *there does not exist* $i \in \{2,3,....,l-1\}$ such that $\lambda_i\lambda_{i+1} = 11$.
  Then $\lambda_{l-1}\lambda_0\lambda_1\lambda_2 = 0111$ ($\lambda_2 = 0$ violates $k > \lceil \frac{l}{2} \rceil$).
  Hence, $l \geq 4$. Let $l > 4$.
  From *lemma 5* we have, $\lambda_0\lambda_1.....\lambda_{l-1}$ does not contain the substring 00.
  So, l is necessarily even ($l \geq 6$)

$$\begin{aligned} \lambda_0\lambda_1....\lambda_{l-1} &= 111(01)^{\frac{l-4}{2}}0 \\ &= 111010(10)^{\frac{l-6}{2}} \end{aligned}$$

  But this violates *lemma 5*, because of the sub-strings 111 & 010.
  Finally, $l = 4$ & $k = 3$ gives
  $\lambda_0\lambda_1\lambda_2\lambda_3\lambda_4\lambda_5\lambda_6\lambda_7 = 11101110$,
  and $\Lambda = \max_{2 \leq p,p+1 \leq 4} \lambda_p + \lambda_{p+1} = 1$

□

5.2 Lower Bound for the Period of Ordering

We now prove the lower bound on the period of ordering using the properties of the equivalence classes.

Theorem 4 *For all correct ordering and* $l > 2$, $T \geq 2p + \Lambda$.

□

We develop a framework that will indeed allow us to prove the lower bound by contradiction.

Consider a permutation that gives a legal ordering of tasks so that the dependences are maintained. Let s_i be the time when the first element of the equivalence class $X_i (x \in \{0,1,....,l-1\} \cap X_i)$ is executed. Let f be an integer such that $\forall i \in [0, l-1] \bullet s_i \leq s_f$.

Let us rename the equivalence classes $X_0, X_1, X_2, ..., X_{l-1}$ to $A_1, A_2, A_3, ...A_l$ such that $X_f = A_l$ and the relation $A_l \rightarrow A_{l-1} \rightarrow .... \rightarrow A_1$ holds. Now, let b_i be the time when the first element of A_i is executed and let e_i be the time when the last element of A_i ($x \in \{n-l, n-l+1, ..., n-1\} \cap A_i$) is executed. Let $\lambda_i = |A_i| - p$.

In order to explain the results of theorem, we introduce the following notation.

$$b_i \rightsquigarrow b_k \Longleftrightarrow (b_i < b_k) \wedge \neg\ (\exists j \mid b_i < b_j < b_k)$$

The set $B = \{b_1, b_2, ..., b_l\}$, which is a subset of $\mathbb{R}$ is totally ordered with respect to the relation $<$. $b_i \rightsquigarrow b_k$ means that b_i and b_k are consecutive elements of B.
The following notation corresponds to the recurrence hypothesis of the theorem.

$$\boxed{b_i} \Longleftrightarrow \forall j\ ,\ j < i \Rightarrow b_j < b_i$$

Since, A_l is the last equivalence class whose tasks begin executing, $\exists i \mid \boxed{b_i}$

During the transition from a tile to the next, there is a dependence from the last point of A_1 to the first point of A_l, from the last point A_l to the first point of A_{l-1}, from the last point of A_{i+1} to the first point of A_i,...., from the last point of A_2 to the first point A_1.

Hence, we have the following inequalities between the beginning and finishing time instants of the equivalence sets,

$$1 \leq i \leq l-1,\ e_{i+1} \leq b_i + T - 1$$

$$1 \leq i \leq l,\ e_i \geq b_i + p + \lambda_i - 1$$

Now lets consider $i \in \{3,4,.....,l\}$ such that $\boxed{b_i}$, then we either have,
1. $b_{i-2} < b_{i-1} < b_i$ or,
2. $b_{i-1} < b_{i-2} < b_i$.

The following two lemmas allow us to inductively apply the hypothesis $\boxed{b_i}$ in the two above cases to determine the lower bound on the period of ordering.

Lemma 6 *Let* $i \in \{3,4,....l\}$ *such that* $b_{i-2} < b_{i-1} < b_i$. *Then,*
1. $T \geq 2p + \lambda_{i-1} + \lambda_i$
2. If $T = 2p + \lambda_{i-1} + \lambda_i$ *then* $b_{i-1} \rightsquigarrow b_i$.

□
Proof Refer [18].

Lemma 7 *Let* $i \in \{3,4,...,l\}$ *such that* $b_{i-1} < b_{i-2} < b_i$, *and that* $\boxed{b_i}$. *Then,*

1. $T \geq 2p + \lambda_{i-1} + \lambda_i$

2. If $T = 2p + \lambda_{i-1} + \lambda_i$ *then*

$b_{i-1} \rightsquigarrow b_{i-2} \rightsquigarrow b_i$ *and* $\left| \begin{array}{l} p = 1 \ \& \ \lambda_{i-2} = 0 \\ \textit{or } i = 3 \end{array} \right.$

□

Proof Refer [18].

Proof of theorem 4

Now, let us use the following notation,

$$i = max\{i \mid 3 \leq i \leq l \cdot \lambda_{i-1} + \lambda_i = \max_{2 \leq p, p+1 \leq l} \lambda_p + \lambda_{p+1}\}$$

$$\Lambda = \lambda_{i-1} + \lambda_i$$

Let us assume that $T = 2p + \Lambda - 1$

Now, let's consider the smallest integer $i' \geq i$ such that $\boxed{b_{i'}}$. Since, A_l is the last equivalence class to start execution i' exists. Suppose that $i' \neq i$. This gives rise to two cases,

1. $b_{i'-2} < b_{i'-1} < b_{i'}$. Then according to *lemma 6*, if $T = 2p + \lambda_{i'} + \lambda_{i'-1}$ then, $b_{i'-1} \rightsquigarrow b_{i'}$. Thus, we have $\boxed{b_{i'-1}}$, which is absurd because i' is minimum.

2. $b_{i'-1} < b_{i'-2} < b_{i'}$. Since $i' \neq i \Rightarrow i' \in [4, l]$.
   According to *lemma 7*, we have, $\lambda_{i'-2} = 0$
   Hence, $\lambda_{i'-2} + \lambda_{i'-1} \leq \lambda_{i'-1} + \lambda_{i'} < \Lambda$
   Therefore, $i' - 1 \neq i$
   Finally, $i' - 2 \geq i$

   Also, if $T = 2p + \lambda_{i'} + \lambda_{i'-1} = 2p + \Lambda - 1$ then $b_{i'-1} \rightsquigarrow b_{i'-2} \rightsquigarrow b_{i'}$.
   Hence, $\boxed{b_{i'-2}}$ which is also absurd because i' is minimum.

Hence, $i = i'$ and according to *lemmas 6 & 7*,
$T \geq 2p + \lambda_i + \lambda_{i-1} \geq 2p + \Lambda$
which contradicts our previous assumption ($T = 2p + \Lambda - 1$).
Hence, $T \geq 2p + \Lambda$.

□

5.3 Constant Permutation Algorithm

Using the framework developed in the previous sections we are now able to devise an algorithm that will compute the optimal ordering of tasks under the assumption that the ordering is the same in every tile. *Algorithm 1* gives us a correct permutation of tasks with period of ordering $(T) = 2p + \Lambda$. Recall that $l \geq 3$ and $n \wedge l = 1$.

Algorithm 1

procedure *ComputeOrderingCst* (n, l)
Input: n (tile size)
l (dependence distance)
Precondition: $l \geq 3$
$n \wedge l = 1$
Output: $permutation[0 : n-1]$ (task ordering)
{ Initialize variables p and k }
  $p := \lfloor \frac{n}{l} \rfloor$
  $k := n \bmod l$
{ Assign the first task f to be executed }
  if ($l = 4$ **and** $k = 3$) **then**
    $f := l - k$ { because $0 = (f + n)[l]$ }
  else $f := 0$
{ Execute the first p tasks of X_f }
  $t := f$
  for $i := 0$ **to** $p - 1$ **do**
    $permutation[i] := t$
    $t := t + l$
  endfor
{ Equivalence classes are executed in the opposite order to ($\longrightarrow$) }
  $t := (t + n) \bmod l$
  while $t \neq f$
    repeat
      $permutation[i] := t$
      $t := t + l$
      $i := i + 1$
    until $t \geq n$
    $t := (t + n) \bmod l$
  endwhile
{ Execute the last task of X_f }
  $permutation[i] := f + p \times l$
end *ComputeOrderingCst*

The permutation given by this algorithm clearly obeys the constraints of dependences in the tile. To prove that the algorithm is optimal we have to show that the permutation given by the algorithm gives us the minimum period of ordering, $T = 2p + \Lambda$.

We use notations from section 5.2. We have,

$b_0 = 0$

$b_1 = p$ $\quad e_1 = p + |Y_1| - 1$

$b_2 = p + |Y_1|$ $\quad e_2 = p + |Y_1| + |Y_2| - 1$

$\vdots$ $\quad \vdots$

$b_j = p + \sum_{i=1}^{j-1} |Y_i|$ $\quad e_j = p + \sum_{i=1}^{j} |Y_i| - 1$

$\vdots$ $\quad \vdots$

$b_{l-1} = p + \sum_{i=1}^{l-2} |Y_i|$ $\quad e_{l-1} = n - 2$

$e_0 = e_l = n - 1$

Moreover,
$\forall j \in [0, l-2], Y_{j+1} \rightarrow Y_j$.
A necessary and sufficient condition for T to be a correct period of ordering is,
$\forall j \in [0, l-1],\ e_{j+1} < b_j + T$ i.e.,
$\forall j \in [0, l-1],\ T \geq e_{j+1} - b_j + 1$
Now, $\forall j \in \{1, \cdots, l-2\}$

$$\begin{aligned} e_{j+1} - b_j + 1 &= p + \sum_{i=1}^{j+1} |Y_i| - 1 - \\ &\quad (p + \sum_{i=1}^{j-1} |Y_i| + 1) \\ &= |Y_j| + |Y_{j+1}| \\ e_1 - b_0 + 1 &= p + |Y_1| \\ e_l - b_{l-1} + 1 &= n - 1 - (p + \sum_{i=1}^{l-2} |Y_i|) + 1 \\ &= 1 + |Y_{l-1}| \end{aligned}$$

Finally,

- If $l = 4$ and $k = 3$, we have,

$$\begin{aligned} (|Y_0|, |Y_1|, |Y_2|, |Y_3|) &= (|X_1|, |X_0|, |X_3|, |X_2|) \\ &= (\lambda_5 + p, \lambda_4 + p, \\ &\quad \lambda_3 + p, \lambda_2 + p) \end{aligned}$$

  Hence, $T = 2p + \max_{2 \leq q, q+1 \leq 4} \lambda_q + \lambda_{q+1}$ is correct.

- Else $(|Y_0|, |Y_1|, \cdots, |Y_{l-1}|) = (|X_0|, |X_{l-1}|, \cdots, |X_1|) = (\lambda_l + p, \lambda_{l-1} + p, \cdots, \lambda_1 + p)$
  Hence, $T = 2p + \max_{1 \leq q, q+1 \leq l-1} \lambda_q + \lambda_{q+1}$ is correct.

6 General Case with Constant Permutation

6.1 Optimal Permutation with One Dependence Vector

Suppose that $n \wedge l = d$. We illustrate how our approach finds a better permutation than M. Dion's approach.

Let us consider a correct ordering with T as the period. Consider the sub-set of tasks $(t'_i)_{i \in [0, n-1]}$ such that $\forall i, t'_i = t_{id}$. Let us call the tile consisting of the tasks (t'_i) a derived tile. The length of the derived tile is $n' = \frac{n}{d}$. The derived dependence distance is $l' = \frac{l}{d}$. We assume that the derived permutations in each tile are constant *i.e.,* they are the same for all processors.

Hence, according to the last part $T \geq T_{min}(n', l')$.

Now, let us show that $T_{min}(n, l) = 2\lfloor \frac{n'}{l'} \rfloor + \Lambda(n', l')$

Let $\sigma' : \{0, \cdots, n'-1\} \rightarrow \{0, \cdots, n'-1\}$ be the permutation of tasks that gives the period T'_{min}. Then, consider $\forall v \in \{0, \cdots, d-1\}, \forall i \in \{0, \cdots, n'-1\}$ $\sigma(v + id) = v + \sigma'(i)d$
Clearly, this permutation permits to reach the period $T = T'_{min}$. Thus, *Algorithm 1* generates the optimal permutation for this case also simply by using n' and l' as inputs instead of n and l.

The task execution times given by our algorithm for $n = 14$ and $l = 6$ is shown in Table 1. Since $n \wedge l = 2$, tile 0 has two components *viz.,* {0,2,4,6,8,10,12} and {1,3,5,7,9,11,13}. Each of them form sub-tiles with $n' = 7$ and $l' = 3$. The first component has equivalence classes $X_0 = \{0, 6, 12\}, X_1 = \{4, 10\}, X_2 = \{2, 8\}$. First, p tasks of X_0 are executed followed by all the tasks of X_2 and X_1 in that order. Finally the last task of X_0 is executed. The second component is then executed with a similar task ordering.

The period of ordering reached by our algorithm is 4 while that reached by Dion's algorithm is 6.

Case $n = 14$ & $l = 6$															
Tile 0	Tasks	0	1	2	3	4	5	6	7	8	9	10	11	12	13
	Times	0	7	2	9	4	11	1	8	3	10	5	12	6	13
Tile 1	Tasks	14	15	16	17	18	19	20	21	22	23	24	25	26	27
	Times	4	11	6	13	8	15	5	12	7	14	9	16	10	17
Tile 2	Tasks	28	29	30	31	32	33	34	35	36	37	38	39	40	41
	Times	8	15	10	17	12	19	9	16	11	18	13	20	14	21

Table 1: Optimal ordering with constant permutation in each tile ($n = 14, l = 6$)

6.2 Several Dependence Vectors

The problem with several constant dependence vectors is complicated and we do not propose an optimal algorithm. Let v be the number of dependence vectors with $l_1 < l_2 < \cdots < l_v$ being the lengths of the dependence vectors. Let us denote $l = l_1 \wedge l_2 \wedge \cdots \wedge l_v$. Dion has shown that for large n, $T_{min}(n, (l_1, l_2, \cdots, l_v)) \approx T_{min}(n, l)$

Hence, the heuristic proposed is to simply use the algorithm for the single dependence vector problem *ComputeOrderingCst(n, l)* with n and l as defined above.

7 Optimal Solution with Non-constant Permutation

We can further optimize the solution if we relax the constraint of maintaining a constant permutation in every tile. We also show that computing the optimal permutation in every tile does not result in any overhead because the optimal permutation in each tile is a simple shift of the permutation in the previous tile. By removing the constraint, we can reach the optimal period of ordering which is half smaller than that in the case of constant permutation.

7.1 Case $n \wedge l = 1$

Recall that from section 4, we have $X_0 \rightarrow X_1 \rightarrow \cdots \rightarrow X_{l-1} \rightarrow X_0$. In fact, the optimal algorithm is the most natural algorithm. The first processor executes tasks belonging to X_0 then X_1, X_2 and finally X_{l-1}. The second processor starts working $|X_0|\tau_{calc} + \tau_{comm}$ units of time after the first one, and executes the tasks of X_1 then $X_2, X_3 \cdots X_{l-1}$ and finally X_0. The third processor starts working $|X_1|\tau_{calc} + \tau_{comm}$ units of time after the second one, and executes the tasks of X_2 then $X_3, X_4 \cdots X_0$ and finally X_1. This leads to the following algorithm.

procedure *ComputeOrderingNoncst* (n, l, i)
Input: n (tile size)
l (dependence distance)
i (tile number)
Precondition: $n \wedge l = 1$
Output: $permutation[0 : n-1]$ (task ordering)
{ Initialize variables m and f }
$m := \lceil \frac{ni}{l} \rceil$
$f := (ml) \bmod n$
{ Assign the first task d to be executed }
$x := 0$
$permutation[x] := f$
{ Execute each equivalence class in the order of $(\longrightarrow)$}
$j := (f + l) \bmod n$
while $j \neq f$ **do**
$x := x + 1$
$permutation[x] := j$
$j := (j + l) \bmod n$
endwhile
end *ComputeOrderingNoncst*

In the above algorithm we have,

- i is the subscript of the tile being executed.
- m is the smallest integer such that $ni \leq ml < n(i+1)$.
- f is the first task to be executed by processor i.

The time offset O_i for each tile is the number of time units between the start of execution of tile i and tile $i + 1$. The offset O_i as opposed to the period of ordering need not be the same for every tile. The offset O_i (in computational-time units) corresponds to the size of the following set,

$$X_i = \{m \in N, in \leq ml < (i+1)n\}$$

Case $n = 5$ & $l = 3$						
Tile 0	Order	0	3	1	4	2
	Offset	2				
Tile 1	Order	1	4	2	0	3
	Offset	2				
Tile 2	Order	2	0	3	1	4
	Offset	1				

Table 2: Optimal ordering with non-constant permutation in each tile ($n = 5, l = 3$)

Hence,

$$O_i = \lceil \frac{n(i+1) - \lceil \frac{ni}{l} \rceil l}{l} \rceil \tau_{calc} + \tau_{comm}$$

Consider the example $n = 5$ and $l = 3$. We have,

0	2	4	1	3	5	2	4	6	3	5	7	4	6	8 $\cdots$
•	∘	×	•	∘	×	•	∘	×	•	∘	×	•	∘	× $\cdots$

Table 2[1] illustrates how the algorithm *ComputeOrderingNoncst* reaches the above ordering.

7.2 General Case

In the general case *i.e.,* with $n \wedge l = d$ and with several dependence vectors, the treatment is similar to the method used for the constant permutation case. Let v be the number of constant dependence vectors with $l_1 < l_2 < \cdots < l_v$ denoting their lengths. Let $l = l_1 \wedge l_2 \wedge \cdots \wedge l_v$. *Algorithm 2* is presented below.

Algorithm 2

procedure *ComputeOrderingNoncstGen* (n, l, i)
Input: n (tile size)
l (dependence distance)
i (tile number)
Output: *permutation*$[0 : n-1]$ (task ordering)
{ Initialize variables d, n_d and l_d }
$d := \gcd(n, l)$
$n_d := \frac{n}{d}$
$l_d := \frac{l}{d}$
{ Initialize variables m and f }
$m := \lceil \frac{n_d i}{l_d} \rceil$
$f := (m l_d) \bmod n_d$
for $i_d := 0$ **to** $d - 1$ **do**
{ Assign the first task f to be executed }
$x := 0$
$permutation[x] := fd + i_d$
{ Execute each equivalence class in the order of $(\longrightarrow)$}
$j := (f + l_d) \bmod n_d$
while $j \neq f$ **do**
$x := x + 1$
$permutation[x] := jd + i_d$
$j := (j + l_d) \bmod n_d$
endwhile
endfor
end *ComputeOrderingNoncstGen*

The offset (in computation time units) is given by the following function,

function *Offset* (n, l, i)
Input: n (tile size)
l (dependence distance)
i (tile number)
Output: O_i(Offset for tile i)
{Initialize d, n_d and l_d}
$d := \gcd(n, l)$
$n_d := \frac{n}{d}$
$l_d := \frac{l}{d}$
{ Initialize variables m and f }
$m := \lceil \frac{n_d i}{l_d} \rceil$
$f := (m l_d) \bmod n_d$
{ Compute Offset }
$O_i := \lceil \frac{n_d - f}{l_d} \rceil$
return(O_i)
end *Offset*

Table 3 presents an example ($n = 15$ & $l = 9$) As stated earlier, when $n \wedge l = d \neq 1$, the problem is reduced to a smaller problem with tile size $n' = \frac{n}{d}$ and dependence distance $l' = \frac{l}{d}$. Consider the equivalence classes $X_0 \rightarrow X_1 \rightarrow \cdots \rightarrow X_{l'-1} \rightarrow X_0$ as defined in section 4. Consider the equivalence class X_0. It starts executing on tile T_0 at $t = 0$ and completes execution at $t = |X_0|$.

[1]The subscripts of tasks are modulo n.

Case $n = 15$ & $l = 9$	
Tile 0	Order 0 9 3 12 6 1 10 4 13 7 2 11 5 14 8
	Offset 2
Tile 1	Order 3 12 6 0 9 4 13 7 1 10 5 14 8 2 11
	Offset 2
Tile 2	Order 6 0 9 3 12 7 1 10 4 13 8 2 11 5 14
	Offset 1

Table 3: Optimal ordering with non-constant permutation in each tile ($n = 15, l = 9$)

Now, at time $t = |X_0|$, equivalence class X_1 starts executing on tile T_1. Note that simultaneously, X_1 starts executing on tile T_0. In general, tasks corresponding to X_i and tile T_j are being executed by processor P_j at the same time as tasks corresponding to X_i and tile T_{j+1} are being executed by processor P_{j+1}. Clearly, we can see that all dependences are satisfied.

It is easy to show that the period of ordering reached by *Algorithm 2* is optimal. Consider two tiles T_j and T_{j+1}. Suppose that equivalence class X_i starts executing on T_j at $t = t_0$. Since $X_i \rightarrow X_{i+1}$ the earliest time at which X_{i+1} can start execution on T_{j+1} is $t = t_0 + |X_i|$. This is precisely the time offset between two consecutive tiles obtained through *Algorithm 2*.

8 Results

8.1 Performance Evaluation

The performance evaluation of the proposed methods was carried out using several signal processing applications consisting of matrix transformations. We tested our proposed algorithms using a sample test routine shown below,

```
Do i: 0 -> N
  c[i] = compute_intensive_func(i)
  If (i .geq. 1)
     A[i] = c[i]*A[i - 1] + 3
  Endif
Enddo
```

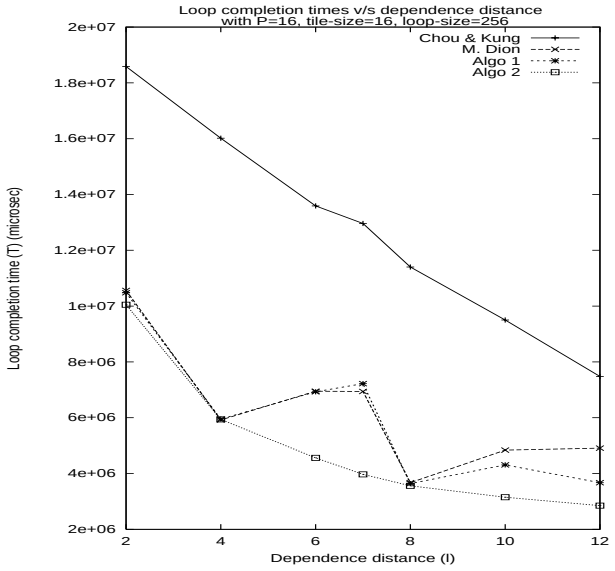
We tiled the above uniform loop using the tiling transformation provided by the *SUIF* [19] compiler varying the tile size (n) and the dependence distance (l). Since the dependence distance is a compile-time unknown in the above loop, the code which computes the permutation is generated at the entry point of the tiled loop as shown below. At run time this code will thus generate an appropriate permutation in each tile for ordering tasks within the tiled loop.

```
Do i_tile: 0 -> N by n  <tiled loop>
<code for generating permutation>
 Do i: i_tile -> min(N,i_tile+n-1)
   c[i] = compute_intensive_func(i)
   If (i .geq. 1)
      A[i] = c[i]*A[i - 1] + 3
   Endif
 Enddo
Enddo
```

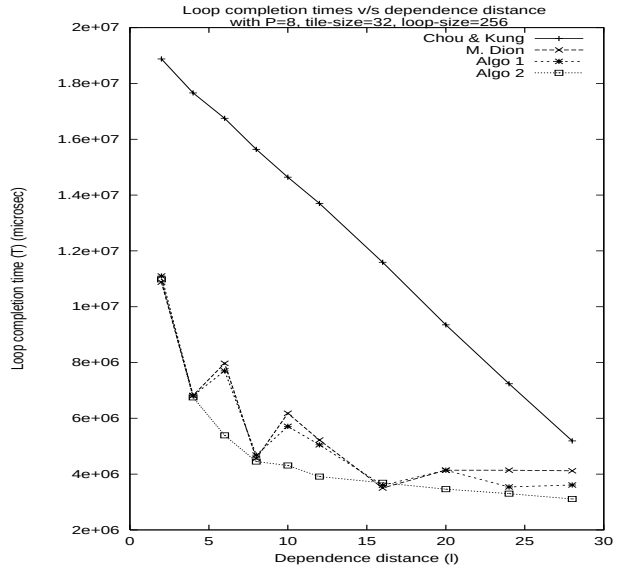
We then carried out a parallel execution by inserting *MPI* [15] calls in order to achieve the synchronization imposed by the dependency constraints. The complete framework has been incorporated in the *SUIF* compiler as an optimization pass. The final transformed code was targeted on

- SGI Power Challenge : A shared memory multiprocessor system consisting of sixteen 90 MHz IP21 processors and 2GB of main memory. Each processor has a MIPS R8000 CPU and a MIPS R8010 floating point unit.
- CRAY T3D : A parallel processing system consisting of 128 DEC Alpha 21064 RISC processors linked via a high bandwidth, low-latency torus interconnection network. T3D memory is globally distributed but logically shared among the processors.

The metric used to evaluate the performance of our algorithms in comparison to previous algorithms was the total loop completion time. Figure 7 presents results obtained using the two proposed algorithms (Algorithm 1 and 2 presented in sections 5.3 and 7.2) in comparison to previous approaches on the SGI Power Challenge. Figure 8 presents results obtained on the CRAY T3D.

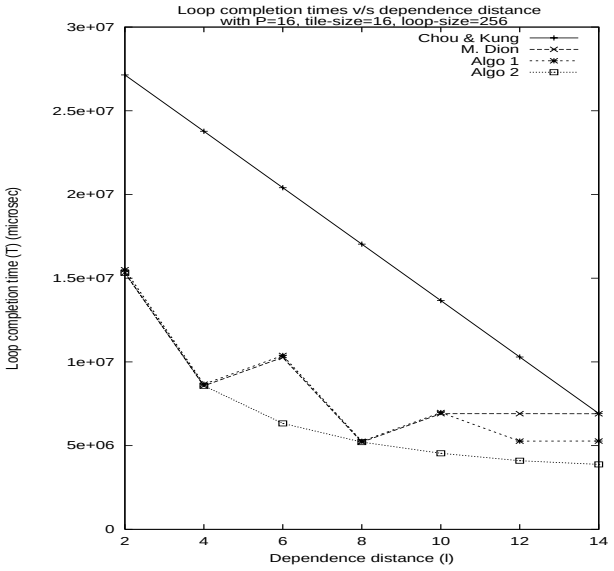


(a)

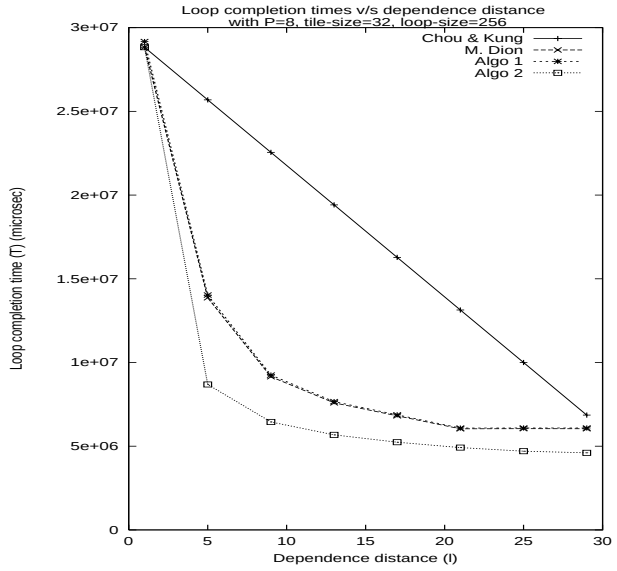


(b)

Figure 7: Loop completion time v/s dependence distance on SGI PC for (a) P = 16, n = 16 and N = 256 and (b) P = 8, n = 32 and N = 256



(a)



(b)

Figure 8: Loop completion time v/s dependence distance on Cray T3D for (a) P = 16, n = 16 and N = 256 and (b) P = 8, n = 32 and N = 256

As seen in Figures 7 and 8[2] the performance obtained by using the two proposed algorithms is superior to that obtained by using algorithms proposed by Dion and Chou & Kung. This is true in the case of shared memory systems[3] as well as distributed memory systems.

From the implementation for the single dimensional tiling problem we observed that for small tile sizes we need long or compute intensive tasks so that the ratio $\frac{\tau_{comm}}{\tau_{calc}} < 1$. The results indicate the following performance hierarchy of the algorithms proposed in this paper. For small tile sizes,
Algorithm 2 > Algorithm 1 > Dion's algorithm

Algorithm 2 is also the most natural and efficient algorithm. For large tile sizes the results indicate that Algorithm 2 yields the best solution which is superior to solutions obtained by all constant permutation algorithms. The performance hierarchy of the algorithms for large tile sizes is shown below,
Algorithm 2 > Algorithm1 $\approx$ Dion's algorithm

8.2 Effect of tile size

An interesting issue is to study the effect of variation of tile size on performance. As the dimension of the problem increases the tile size is limited by the cache size. In case of the multi-dimensional tiling problem $n_1 \times n_2 \times \cdots \times n_d$, the gain of our method using Algorithm 1 over Dion's algorithm will be linear in $\prod_{i=1}^{d-1} n_i$.

For the 1-dimensional problem the cache size is usually not limited and $n = \frac{N}{P}$ or n is chosen as discussed below.[4]

Desprez et al [6] have addressed the issue of finding the optimal grain size that minimizes the execution time by improving pipeline communications on parallel computers. Our goal is to determine the optimal tile size for minimal loop completion time given our framework. The following discussion presents the effect of tile size (n) on the total loop completion time (T_{tot}).

Using Algorithm 2, we obtain the following period of ordering,

$$T \approx \frac{n}{l}\tau_{calc} + \tau_{comm}$$

The total loop completion time is given by,

$$\begin{aligned} T_{tot} &\approx \frac{N}{n}T + n\tau_{calc} \\ &= \frac{N\tau_{calc}}{l} + n\tau_{calc} + \frac{N\tau_{comm}}{n} \\ &= A + Bn + \frac{C}{n} \end{aligned}$$

Minimizing the above expression we get,

$$n_{opt} = \sqrt{\frac{C}{B}} = \sqrt{\frac{N\tau_{comm}}{\tau_{calc}}}$$

Let $\frac{\tau_{comm}}{\tau_{calc}} = c$. Therefore,

$$n_{opt} \approx \sqrt{cN}$$

Also we have,

$$\frac{n}{l}\tau_{calc} > \tau_{comm}$$

This leads to,

$$n > lc$$

In order to compare the above analytical solution with experimental results we observed the total loop completion time (T_{tot}) varying the tile size (n) keeping P fixed. Figure 9(a) presents this comparison for $N = 2097152$ and $l = 7$. Figure 9(b) presents the comparison for $N = 524288$ and $l = 7$. Figure 9 shows that the analytical expression derived for the loop completion time closely matches the experimental results. The knee of the analytical solution curve corresponds to the optimum tile size that yields the minimum loop completion time. The deviation of the analytical solution from the experimental results for

[2] The legend Algo1 denotes results for Algorithm 1 and Algo 2 denotes the results for Algorithm 2.

[3] The minor performance gain of M. Dion's algorithm over algorithm 1 in figure 7(a) can be attributed to the fact that the optimal permutation generation of algorithm 1 is more complex than that of M. Dion's algorithm

[4] For the multi-dimensional problem cache size becomes a limiting factor and we have Algorithm 1 > Dion's algorithm. Whether a constant permutation algorithm like Algorithm 1 yields a better solution than a non-constant permutation algorithm like Algorithm 2 is an open issue.

small tile sizes is due to the fact that for small tile sizes $n < lc$ ($c = 2048$) indicating that the communication time exceeds the total computation time.

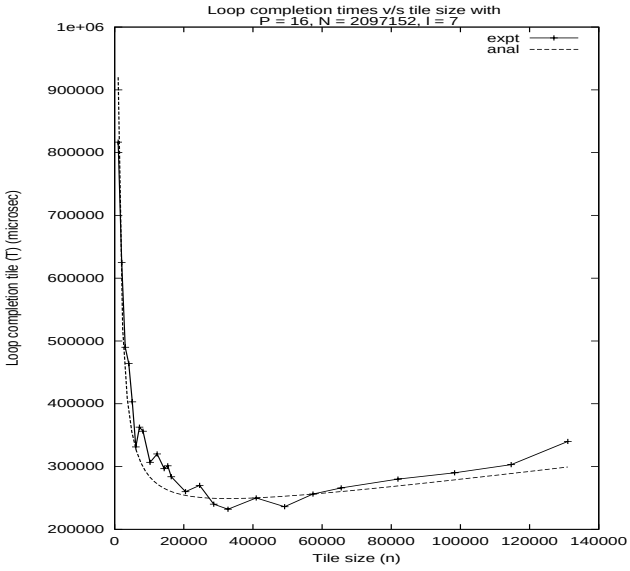
On conventional parallel architectures, it is clear that the non-constant permutation algorithm yields the least loop completion time for the single dimensional tiling problem.

9 Conclusions

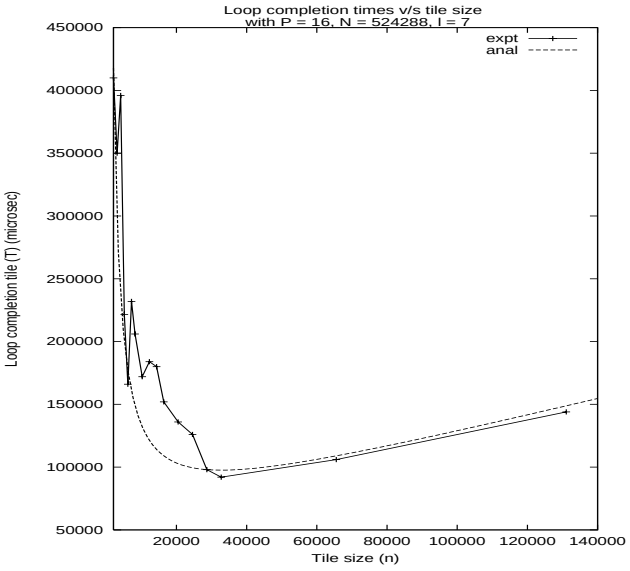
Tiling is typically carried out to increase granularity of computations, locality of data references and data reuse on cache-based multicomputers. A significant amount of work has been done to solve the important problem of deriving the optimal shape and size of the tile to minimize communication. Once the parameters of the tile have been determined, the effectiveness of tiling is critically dependent on the execution order of tasks within a tile. In this work, we have addressed the problem of finding an optimal ordering of tasks within tiles executed on multicomputers for constant but compile-time unknown dependences. We remove the restriction of atomicity on tiles and exploit the internal parallelism within each tile by overlapping computation with communication. We have formulated the problem and developed a new framework based on equivalence classes in order to prove optimality results for single dimensional tiles with single constant dependences. Using the framework we have also developed two efficient algorithms that provide the optimal solution in both cases,

1. Same (constant) task ordering in every tile,
2. Different (non-constant) task ordering in each tile.

We have shown that the two proposed algorithms yield superior results to the previous approaches when tested on distributed and shared memory systems. We also show that the non-constant permutations in our approach significantly reduce the loop completion time unlike the constant permutations in previous approaches. Finally, we



(a)



(b)

Figure 9: Loop completion time v/s tile size on SGI PC for (a) P = 16, l = 7 and N = 2097152 and (b) P = 16, l = 7 and N = 524288

have investigated the relationship between tile size and the loop completion time and developed a methodology to obtain optimal tile size given our framework.

On conventional parallel architectures, it is clear that the non-constant permutation algorithm yields the least loop completion time. It is an open question whether the non-constant permutations yield better results in the multi-dimensional case as well[5] A performance comparison of these algorithms on a VLSI array processor based architecture [13] (for which Chou and Kung had originally proposed their algorithm) could be interesting. A comparison of approaches based on constant and non-constant permutations using the architectural characteristics (such as specialized hardware for generating and transforming permutations) could also be interesting.

References

[1] Anant Agarwal, David Kranz, and Venkat Natrajan. Automatic Partitioning of Parallel Loops and Data Arrays for Distributed Shared-Memory Multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 6(9):943–962, September 1995.

[2] Jennifer M. Anderson and Monica S. Lam. Global Optmizations for Parallelism and Locality on Scalable Parallel Machines. In *Proceedings of the ACM SIGPLAN '91 Conference on Programming Language Design and Implementation*, pages 112–125, June 1993.

[3] C. Calvin and F. Desprez. Minimizing Communication Overhead Using Pipelining for Multi-Dimensional FFT on Distributed Memory Machines. In D.J. Evans, H. Liddell J.R. Joubert, and D. Trystram, editors, *Parallel Computing'93*, pages 65–72. Elsevier Science Publishers B.V. (North-Holland), 1993.

[4] W. H. Chou and S. Y. Kung. Scheduling Partitioned Algorithms on Processor Arrays with Limited Communication Supports. In *Proceedings of the International Conference on Application Specific Array Processors (ASAP)*, pages 53–64, 1993.

[5] Stephanie Coleman and Kathryn Mckinley. Tile Size Selection using Cache Organization and Data Layout. In *Proceedings of the ACM SIGPLAN '95 Conference on Programming Language Design and Implementation*, volume 30(6), pages 279–290, June 1995.

[6] F. Desprez, P. Ramet, and J. Roman. Optimal Grain Size Computation for Pipelined Algorithms. In *Europar'96 Parallel Processing*, volume 1123 of *Lecture Notes in Computer Science*, pages 165–172. Springer Verlag, August 1996.

[7] Frédéric Desprez, Jack Dongarra, Fabrice Rastello, and Yves Robert. Determining the Idle Time of a Tiling: New Results. In *Proceedings of the Conference on Parallel Architectures and Compilation Techniques (PACT '97)*, pages 307–321. IEEE/ACM, November 1997.

[8] Erik H. D'Hollander. Partitioning and Labeling of Loops by Unimodular Transformations. *IEEE Transactions on Parallel and Distributed Systems*, 3(4):465–476, July 1992.

[9] Michèle Dion. *Alignement et Distribution en Parallélisation Automatique*. PhD thesis, Ecole Normale Supérieure de Lyon, January 1996.

[10] Michèle Dion, Tanguy Risset, and Yves Robert. Resource-constrained Scheduling of Partitioned Algorithms on Processor Arrays. In *Proceedings of Euromicro Workshop on Parallel and Distributed Processing*, pages 571–580. IEEE Computer Society Press, January 1995.

[5] The general problem of determining optimal permutations in the multi-dimensional case is very hard.

[11] F. Irigoin and R. Triolet. Supernode Partitioning. In *15th Symposium on Principles of Programming Languages (POPL XV)*, pages 319–329, January 1988.

[12] Wesley K. Kaplow and Boleslaw K. Szymanski. Tiling for Parallel Execution - Optimizing Node Cache Performance. In *Workshop on Challenges in Compiling for Scaleable Parallel Systems, Eigth IEEE Symposium on Parallel and Distributed Processing*, 1996.

[13] S. Y. Kung. *VLSI Array Processors*. Prentice Hall, 1988.

[14] Wei Li. Compiler Cache Optimizations for Banded Matrix Problems. In *Conference proceedings of the 1995 International Conference on Supercomputing*, pages 21–30, July 1995.

[15] MPI Forum. *MPI : A Message Passing Interface Standard*, June 1995. Version 1.1, `http://www.mcs.anl.gov/mpi/`.

[16] Hiroshi Ohta, Yasuhiko Saito, Masahiro Kainaga, and Hiroyuki Ona. Optimal Tile Size Adjustment in Compiling General DOACROSS Loop Nests. In *Conference proceedings of the 1995 International Conference on Supercomputing*, pages 270–279, July 1995.

[17] J. Ramanujam and P. Sadayappan. Tiling Multidimensional Iteration Spaces for Multicomputers. *Journal of Parallel and Distributed Computing*, 16:108–120, 1992.

[18] Fabrice Rastello, Amit Rao, and Santosh Pande. Optimal Task Ordering in Linear Tiles for Minimizing Loop Completion Time. Technical report, University of Cincinnati, January 1998. Tech. Report for NSF sponsored project no. CCR-996129.

[19] Stanford University. *The SUIF Library*, 1994. This manual is a part of the SUIF compiler documentation set, `http://suif.stanford.edu/`.

[20] Peiyi Tang and John N. Zigman. Reducing Data Communication Overhead for DOACROSS Loop Nests. In *Conference proceedings of the 1994 International Conference on Supercomputing*, pages 44–53, July 1994.

[21] Michael E. Wolf and Monica S. Lam. A Data Locality Optimizing Algorithm. In *Proceedings of the ACM SIGPLAN '91 Conference on Programming Language Design and Implementation*, pages 30–44, June 1991.

[22] M. J. Wolfe. More Iteration Space Tiling. In *Proceedings of Supercomputing '89*, pages 655–664, November 1989.

[23] Jingling Xue. On Tiling as a Loop Transformation. *Parallel Processing Letters*, 7(4):409–424, 1997.