ELSEVIER

Theoretical Computer Science 255 (2001) 449–481

Theoretical Computer Science

www.elsevier.com/locate/tcs

A theoretical foundation for program transformations to reduce cache thrashing due to true data sharing

Guohua Jin[a], Zhiyuan Li[b,*], Fujie Chen[c]

[a] *Department of Computer Science, Rice University, 6100 Main Street, Houston, TX 77005-1892, USA*
[b] *Department of Computer Science, Purdue University, 1398 University Street, West Lafayette, IN 47907, USA*
[c] *Department of Computer Science, Changsha Institute of Technology, Changsha, Hunan 410073, People's Republic of China*

Received January 1999; revised May 1999
Communicated by D.-Z. Du

Abstract

Cache thrashing due to true data sharing can degrade the performance of parallel programs significantly. Our previous work showed that parallel task alignment via program transformations can be quite effective for the reduction of such cache thrashing. In this paper, we present a theoretical foundation for such program transformations. Based on linear algebra and the theory of numbers, our work analyzes the data dependences among the tasks created by a fork-join parallel program and determines at compile time how these tasks should be assigned to processors in order to reduce cache thrashing due to true data sharing. Our analysis and program transformations can be easily performed by compilers for parallel computers.

Keywords: Complier algorithms; Linear algebra; Theory of numbers; Multiprocessors; Caches

1. Introduction

Private caches are commonly used in tightly coupled multiprocessors to reduce the average latency of memory references. Unfortunately, if parallel tasks are not aligned well, a processor may repeatedly find its data missing in the cache because its cached copy was invalidated by other processors which overwrote the data. This phenomenon, called *cache thrashing*, may be due to either *false sharing* or *true sharing* of data among different processors. For caches whose line size is greater than one word, different processors may write and read different data words which happen to be in the same cache line, causing false sharing. Several algorithms have been proposed to reduce false sharing [10, 17, 19, 36].

* Corresponding author. Tel.: 001-765-494-7822; fax: 001-765-494-0739.
E-mail address: li@cs.purdue.edu (Z. Li).

0304-3975/01/$ - see front matter
PII: S0304-3975(99)00313-8

In contrast, *true sharing* occurs when different processors modify and read the same data word. True sharing causes cache thrashing when a DOALL loop is nested within a sequential loop and different processors modify and read the same data in *different* instances of the same DOALL loop. If tasks are assigned to processors without care, true sharing can quite frequently cause cache thrashing, because DOALL loops are commonly found to be embedded in sequential loops [11].

Recently, we proposed a compiler technique which transforms parallel loop nests in order to reduce cache thrashing due to true sharing [21]. Our experimental results showed that this technique can increase the parallel execution speed of certain Linpack benchmark programs by as high as 105% on a Silicon Graphics Challenge multiprocessor, compared with the best task assignment scheme provided by the machine vendor. Our technique is based on an analysis of data dependences between the tasks created in the different instances of a given DOALL loop. In this paper, we present a number of theorems and their proofs which form the theoretical foundation for the data dependence analysis used in our technique.

Next, in Section 2, we first describe the category of parallel program constructs which are handled by this paper and we make a few assumptions about the target tightly coupled multiprocessors. In Section 3, after a brief presentation of the main concepts and the main algorithms of our program transformation, we prove a number of theorems based on which our algorithms are designed. The related works are discussed in Section 4. We conclude in Section 5.

2. Programming model and machine model

In this paper, we consider fork-join parallel programs whose parallelism is expressed as DOALL loops. No data dependences may exist between different iterations of a DOALL loop [6, 27, 4, 28]. In particular, we consider a program construct which consists of a sequential loop which contains one or several single-level DOALL loops. If there exist multilevel DOALL loops, only one level is parallelized, as on most commercial shared-memory multiprocessor systems. Hence, as shown below, a loop nest in our model has three levels: a parallel loop, its immediately embedded sequential loop, and its immediately enclosing sequential loop. The loops are not necessarily perfectly nested:

DO $I = 1, N_1$
  .....
  DOALL $J = 1, N_2$
    ......
    DO $K = 1, N_3$
      LOOP BODY $\{(A_q(\vec{h}_\gamma(I,J,K)), A_q(\vec{h}'_\gamma(I,J,K))) \mid 1 \leqslant q \leqslant n\}$
    ENDDO
    ......
  ENDDOALL
  ......
ENDDO

where A_q $(q = 1, 2, \ldots, n)$ is an array name appearing in the loop body, $\vec{h}_\gamma, \vec{h}'_\gamma$ $(1 \leqslant \gamma \leqslant m)$ are linear mappings from the iteration space to the domain space of A_q, $(A_q(\vec{h}_\gamma(I,J,K))$, $A_q(\vec{h}'_\gamma(I,J,K)))$ $(1 \leqslant q \leqslant n)$ are potentially dependent reference pairs, and n is the number of such pairs. Multiple array variables and multiple linear subscript functions may exist in the nested loop. Note that we set the lower loop limits to 1 only to simplify the notations throughout the rest of the paper. It is a trivial matter to adjust the theorems for non-unitary lower limits, using more complex notations.

Fang and Lu [11] reported that arrays involved in nested loops are usually two-dimensional or three-dimensional with a small-sized third dimension. The latter can be treated as a small number of two-dimensional arrays. Nested loops with the parallel loop at the innermost level are degenerate cases of the above loop nest. Therefore, our loop nest model seems quite general. Our method can also be applied to a loop nest which contains several separate parallel loops at the middle level. Each of these parallel loops may be restructured according to its own reference patterns, such that the tasks in different instances of the same parallel loop are aligned. We currently do not align the tasks created by different parallel inner loops. For programs with more complicated loop nests, pattern-matching techniques can be used to identify a loop subnest that matches the nest shown above. Other outer- or inner- loops that are not a part of the subnest can be ignored, as long as their loop indices do not appear in the array subscripts.

Our compiler analysis uses a simple multiprocessor model in which each processor has a private cache with copy-back snoopy coherence hardware.

3. Program transformations and a supporting theory

In this section, we first give an example to illustrate our program transformations and to motivate several key definitions. We then present the main theoretical results which serve as the foundation of the proposed transformations.

3.1. Loop staggering and aligned processor folding

We use the following example to explain the main concepts and to show how our techniques can be applied to align the tasks to avoid true data sharing.

Example 1.

```
DO I = 1, N1
  ......
  DOALL J = 1, N2
    .....
    DO K = 1, N3
      ......
        A(J + K, I + J) = A(J + K, I + J) + ...
```

```
            B(J + 2 * K, I + J) = B(J + 2 * K, I + J) + ...
          ......
        ENDDO
        ......
      ENDDOALL
      ......
    ENDDO
```

In the example above, three loops are imperfectly nested, of which the two outer loops form an iteration subspace $N_1 \times N_2$. Loop I is a sequential loop because of the *loop-carried* data dependences involving two linear subscript functions:

$$A(f_1, g_1): \quad f_1(i,j,k) = j + k, \quad g_1(i,j,k) = i + j,$$

$$B(f_2, g_2): \quad f_2(i,j,k) = j + 2k, \quad g_2(i,j,k) = i + j.$$

Each time the DOALL loop J is executed, it generates N_2 parallel iterations, or *tasks*. Since loop J will be executed N_1 times, it generates $N_1 \times N_2$ tasks overall. We let $T_{i,j}$ denote the task corresponding to loop index values $I = i$ and $J = j$. Each task accesses N_3 elements of each array. Many array elements are shared by multiple tasks which are created in different I iterations. One can partition the tasks into subsets, called *equivalence classes* in this paper, such that the tasks in the same equivalence class may modify and reuse certain common array elements but tasks in different equivalence classes do not. $(T_{1,2}, T_{2,1})$, $(T_{1,3}, T_{2,2}\ T_{3,1})$, $(T_{1,4}, T_{2,3}, T_{3,2}, T_{4,1}), \ldots$, are some of the equivalence classes in our example. If the tasks in the same equivalence class are not executed by the same processor, array data being shared will unnecessarily move back and force between different caches in the system, causing a cache thrashing problem due to true data sharing [12]. Unfortunately, the run-time systems on most commercial multiprocessors will assign the tasks which have the same J index to the same processor regardless of the I index. This will distribute the tasks in the same equivalence class to different processors.

To avoid true date sharing, we transform the loop nest in Example 1 such that the tasks in the same equivalence class are guaranteed to be assigned to the same processor. The desired transformation in this case is a linear transform $JJ = I + J$, which results in the following new loop nest:

```
DO I = 1, N1
  ......
  DOALL JJ = 2, N1 + N2
    if (1 ⩽ JJ − I ⩽ N2) then
      ......
      DO K = 1, N3
        ......
```

```
        A(JJ − I + K, JJ) = A(JJ − I + K, JJ) + ...
        ......
        B(JJ − I + 2 * K, JJ) = B(JJ − I + 2 * K, JJ) + ...
        ......
      ENDDO
      ......
    endif
    ......
  ENDDOALL
  ......
ENDDO
```

After the program transformation, different processors will no longer access common array elements, provided that the run-time system follows the common practice and assigns the tasks which have the same DOALL loop index, JJ, to the same processor. In certain cases, true data sharing may not be eliminated by applying the linear transform to the individual tasks, as in the above example. Instead, as shown in the following subsection, such a transform must be applied to sets of tasks. We use the term *loop staggering* to refer to this variant of linear transforms. A straightforward linear transform is a special case of loop staggering. In the next subsection, we will prove a condition under which true sharing can be eliminated by loop staggering alone. In more complex cases, true data sharing may not be eliminated even by loop staggering. For example, if we change the subscripts in Example 1 to $(J+3*K, J-2*I)$ and $(J+3*K+1, I+J)$, respectively, then staggering alone cannot align the tasks and eliminate true sharing. However, the degree of true sharing, and hence cache thrashing, will be reduced even in such cases. Moreover, we will prove that, in such cases, true sharing can be eliminated by *aligned processor folding*, or *aligned folding* in short, in addition to loop staggering.

The idea of aligned folding and linear transforms can be illustrated by an example derived by changing the subscripts in Example 1 to $(J+3*K, J-2*I)$ and $(J+3*K+1, I+J)$, respectively. Using the theorems in the next subsection, we will find that there exist nine disjoint equivalence classes of tasks. Hence, we create a DOALL loop, indexed by JJ, which has nine iterations (to be executed by nine processors). We say that this example has a *folding factor*[1] of 9. We then embed the original DOALL loop J (now serialized and strip-mined with a factor of 9) within loop JJ. The nine equivalence classes are mapped to the 9 JJ loop iterations by dynamically *left-shifting* the lower loop limit of J by three iterations after every three I-iterations. We say that the loop nest is staggered by a pair of *staggering parameters* $(3, -3)$. The transformed program is as follows, in which a pair of local variable, *PSI* and *PSJ* are used to implement the dynamic left-shifting on the lower limit of the J loop, namely

[1] We used the term *compacting factor* in our previous work [21].

$OFFSET + 1$, where $OFFSET$ is also a local variable.

```
DO I = 1, N1
  .....
  DOALL JJ = 1, 9
    if I − PSI = 3 then
      PSI = I
      PSJ = (PSJ − 3) mod 9
    endif
    OFFSET = PSJ − 1
    if OFFSET < 0 then
      OFFSET = OFFSET + 9
    endif
    DO J = OFFSET + 1, N2, 9
      .....
      DO K = 1, N3
        .....
        A(J + 3 * K, J − 2 * I) = A(J + 3 * K, J − 2 * I) + ...
        B(J + 3 * K + 1, I + J) = B(J + 3 * K + 1, I + J) + ...
        .....
      ENDDO
      .....
    ENDDO
    .....
  ENDDOALL
  .....
ENDDO
```

Note that aligned folding may potentially generate a new loop nest which does not utilize all available processors. However, it is quite well known in practice that utilizing all available processors does not always lead to the shortest parallel execution time, due to synchronization overhead, among other factors. Cache thrashing due to true data sharing is yet another factor that must be considered.

The main body of this paper is a theory about how to determine the staggering parameters and the folding factor. For clarity of presentation, we decompose our problem into two parts. In Section 3.2, we first reduce the problem to a basic model and, in Section 3.3, we generalize the model by applying simple affine transformations to the basic model.

3.2. Theorems and proofs for the basic model

In our *basic model*, for each pair of dependent references, the same subscript function is used in both references. (However, different dependent pairs can use different

subscript functions.) The examples in Section 3.1 conform to this basic model. If we extract the subscript function $\vec{h}_\gamma(I,J,K)$ from each pair of dependent references, then the considered loop nest which has m pairs of dependent references can be illustrated by the following code segment.

```
DO I = 1, N1
  .....
  DOALL J = 1, N2
    ......
    DO K = 1, N3
      LOOP BODY {h_γ(I,J,K) | 1 ≤ γ ≤ m}
    ENDDO
    ......
  ENDDOALL
  ......
ENDDO
```

Without loss of generality, suppose that all m linear subscript functions above are different. We assume that each function $\vec{h}_\gamma$, $1 \leqslant \gamma \leqslant m$, is of rank 2 and is in the form of $\vec{h}_\gamma(i,j,k) = (f_\gamma(i,j,k),\ g_\gamma(i,j,k))$, where

$$\begin{bmatrix} f_\gamma(i,j,k) \\ g_\gamma(i,j,k) \end{bmatrix} = \begin{bmatrix} a_{\gamma,1} & b_{\gamma,1} & c_{\gamma,1} & d_{\gamma,1} \\ a_{\gamma,2} & b_{\gamma,2} & c_{\gamma,2} & d_{\gamma,2} \end{bmatrix} \begin{bmatrix} i \\ j \\ k \\ 1 \end{bmatrix}.$$

We denote

$$\begin{bmatrix} a_{\gamma,1} & b_{\gamma,1} & c_{\gamma,1} \\ a_{\gamma,2} & b_{\gamma,2} & c_{\gamma,2} \end{bmatrix}$$

as H_γ and

$$\begin{bmatrix} x_{\gamma,1} & y_{\gamma,1} \\ x_{\gamma,2} & y_{\gamma,2} \end{bmatrix}$$

as $H_\gamma^{x,y}$, with $x, y \in \{a,b,c\}$.

The iteration subspace $N_1 \times N_2$, shown in Fig. 1, is called the *reduced* iteration space, because it omits the K loop. In order to find the iterations in the reduced iteration space which may access common memory locations within the corresponding tasks, we define a set of elements of array A_γ which are accessed within task T_{i_0,j_0} by using subscript function $\vec{h}_\gamma$.

Definition 1. Given iteration (i_0, j_0) in the reduced iteration space, the elements $A_\gamma(f_\gamma(i_0,j_0,k),\ g_\gamma(i_0,j_0,k))$ of array A_γ, where $1 \leqslant k \leqslant N_3$, $1 \leqslant \gamma \leqslant m$, are accessed within task T_{i_0,j_0}. They are denoted by $\mathscr{A}_\gamma^{i_0,j_0} \equiv \{A_\gamma(f_\gamma(i_0,j_0,k), g_\gamma(i_0,j_0,k)) \mid \text{where } 1 \leqslant k \leqslant N_3\}$.

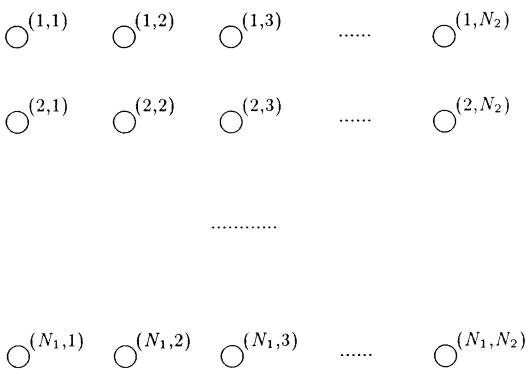

Fig. 1. Reduced iteration space.

Definition 2. If we suppose that $T_{i,j}$ and $T_{i',j'}$ are two tasks corresponding to $(I,J)=(i,j)$ and $(I,J)=(i',j')$ in the reduced iteration space of the given loop nest such that $\mathscr{A}_\gamma^{i,j} \cap \mathscr{A}_\gamma^{i',j'} \neq \emptyset$ $(1 \leqslant \gamma \leqslant m)$, we say $T_{i,j}$ and $T_{i',j'}$ has a dependence because of $\vec{h}_\gamma$, denoted by $T_{i,j}\delta_\gamma T_{i',j'}$.

Since our objective is to eliminate true data sharing by assigning dependent tasks to the same processor, we make the dependence (between tasks due to data sharing) transitive, which leads to the following definition.

Definition 3. If there exists γ, $1 \leqslant \gamma \leqslant m$, such that $T_{i,j}\ \delta_\gamma\ T_{i',j'}$, then we write $T_{i,j}\delta T_{i',j'}$. If $T_{i,j}\delta T_{i',j'}$ and $T_{i',j'}\delta T_{i'',j''}$, then we also have $T_{i,j}\delta T_{i'',j''}$.

The following lemma and two theorems establish the relationship between the loop indexes corresponding to two inter-dependent tasks. We will use this index relationship to *stagger* the loop iteration space such that inter-dependent tasks can be assigned to the same processors.

Lemma 1. *Suppose $T_{i,j}$, $T_{i',j'}$ are two tasks, $i \neq i'$. $T_{i,j}\delta_\gamma T_{i',j'}$ iff there exist k, k', $1 \leqslant k,k' \leqslant N_3$, such that $H_\gamma[i\ j\ k]^{\mathrm{T}} = H_\gamma[i'\ j'\ k']^{\mathrm{T}}$.*

Theorem 1. *Suppose $det\, H_\gamma^{b,a} \neq 0$. $T_{i,j}\delta_\gamma T_{i',j'}$ iff there exist k, k' $(1 \leqslant k,k' \leqslant N_3)$ such that*

$$i' - i = \frac{det\, H_\gamma^{b,c}}{det\, H_\gamma^{b,a}}(k-k'), \quad j' - j = \frac{det\, H_\gamma^{c,a}}{det\, H_\gamma^{b,a}}(k-k').$$

Both Lemma 1 *and Theorem* 1 *are obvious, because both f_γ and g_γ are linear in terms of i,j and k.*

Table 1
All patterns of subscript coefficients when $det\, H_\gamma^{b,a} = det\, H_\gamma^{c,b} = det\, H_\gamma^{c,a} = 0$

$det\, H_\gamma^{b,a} =$ $det\, H_\gamma^{c,b} =$ $det\, H_\gamma^{c,a} = 0$	$a_{\gamma,1} \neq 0,\ b_{\gamma,2} \neq 0$ $\Rightarrow a_{\gamma,2} \neq 0,\ b_{\gamma,1} \neq 0$	$c_{\gamma,1} = c_{\gamma,2} = 0$			1
		$c_{\gamma,1} \neq 0,\ c_{\gamma,2} \neq 0$		$*$	2
	$a_{\gamma,1} = 0,\ b_{\gamma,2} \neq 0$		$c_{\gamma,1} = c_{\gamma,2} = 0$		3
		$a_{\gamma,2} = 0,\ b_{\gamma,1} \neq 0$	$c_{\gamma,1} \neq 0,\ c_{\gamma,2} \neq 0$	$*$	4
		$a_{\gamma,2} \neq 0,\ b_{\gamma,1} = 0$	$c_{\gamma,2} = 0$		5
		$\Rightarrow c_{\gamma,1} = 0$	$c_{\gamma,2} \neq 0$	$\star$	6
		$a_{\gamma,2} = b_{\gamma,1} = 0$	$c_{\gamma,2} = 0$		7
		$\Rightarrow c_{\gamma,1} = 0$	$c_{\gamma,2} \neq 0$	$\star$	8
	$a_{\gamma,1} \neq 0,\ b_{\gamma,2} = 0$	$a_{\gamma,2} = 0,\ b_{\gamma,1} \neq 0$	$c_{\gamma,1} = 0$		9
		$\Rightarrow c_{\gamma,2} = 0$	$c_{\gamma,1} \neq 0$	$*$	10
			$c_{\gamma,1} = c_{\gamma,2} = 0$	$\times$	11
		$a_{\gamma,2} \neq 0,\ b_{\gamma,1} = 0$	$c_{\gamma,1} \neq 0,\ c_{\gamma,2} \neq 0$	$\times$	12
		$a_{\gamma,2} = b_{\gamma,1} = 0$	$c_{\gamma,1} = 0$	$\times$	13
		$\Rightarrow c_{\gamma,2} = 0$	$c_{\gamma,1} \neq 0$	$\times$	14
	$a_{\gamma,1} = b_{\gamma,2} = 0$	$a_{\gamma,2} = 0,\ b_{\gamma,1} \neq 0$	$c_{\gamma,1} = 0$		15
		$\Rightarrow c_{\gamma,2} = 0$	$c_{\gamma,1} \neq 0$	$*$	16
		$a_{\gamma,2} \neq 0,\ b_{\gamma,1} = 0$	$c_{\gamma,2} = 0$	$\times$	17
		$\Rightarrow c_{\gamma,1} = 0$	$c_{\gamma,2} \neq 0$	$\times$	18
			$c_{\gamma,1} = 0,\ c_{\gamma,2} \neq 0$	$\times$	19
			$c_{\gamma,1} \neq 0,\ c_{\gamma,2} = 0$	$\times$	20
		$a_{\gamma,2} = b_{\gamma,1} = 0$	$c_{\gamma,1} = c_{\gamma,2} = 0$	$\times$	21
			$c_{\gamma,1} \neq 0,\ c_{\gamma,2} \neq 0$	$\times$	22

We now consider the case of $det\, H_\gamma^{b,a} = 0$, assuming that the loop bounds, N_2 and N_3, are large enough to satisfy the following:

$$N_2 > \max\left(\left|\frac{c_{\gamma,1}}{\mathrm{GCD}(b_{\gamma,1}, c_{\gamma,1})}\right|, \left|\frac{c_{\gamma,2}}{\mathrm{GCD}(b_{\gamma,2}, c_{\gamma,2})}\right|\right),$$

$$N_3 > \max\left(\left|\frac{b_{\gamma,1}}{\mathrm{GCD}(b_{\gamma,1}, c_{\gamma,1})}\right|, \left|\frac{b_{\gamma,2}}{\mathrm{GCD}(b_{\gamma,2}, c_{\gamma,2})}\right|\right).$$

These assumptions are almost always true in practice [34]. When they are not true, the parallel loops will be too small to be important. With these assumptions, we have the following theorem.

Theorem 2. *Suppose* $det\, H_\gamma^{b,a} = 0$. *The fact that the J loop at the middle level is a DOALL loop guarantees that* $T_{i,j}\delta_\gamma T_{i',j'}$ *iff we have*
(1) $a_{\gamma,1}(i'-i) + b_{\gamma,1}(j'-j) = 0$, *for* $a_{\gamma,1} \neq 0$,
(2) $a_{\gamma,2}(i'-i) + b_{\gamma,2}(j'-j) = 0$, *for* $a_{\gamma,2} \neq 0$, *and*
(3) $j'-j = 0$, *for* $a_{\gamma,1} = a_{\gamma,2} = 0$.

Proof. The case of $det\, H_\gamma^{b,a} = 0$ can be divided into two subcases: (1) $det\, H_\gamma^{c,b} = det\, H_\gamma^{c,a} = 0$; and (2) one of $det\, H_\gamma^{c,b}$ and $det\, H_\gamma^{c,a}$ is non-zero.

In the first subcase, $det\, H_\gamma^{b,a} = det\, H_\gamma^{c,b} = det\, H_\gamma^{c,a} = 0$, there are 22 different patterns of coefficients as shown in Table 1. Within these, 10 patterns labeled as "×" (11–14

Table 2
All patterns of subscript coefficients when $det\, H_\gamma^{b,a} = 0$ and at least one of $det\, H_\gamma^{c,b}$ and $det\, H_\gamma^{c,a}$ being non-zero

$(det\, H_\gamma^{b,a} = 0) \wedge$ $((det\, H_\gamma^{c,b} \neq 0) \vee$ $(det\, H_\gamma^{c,a} \neq 0))$	$a_{\gamma,1} = a_{\gamma,2} = b_{\gamma,1} = 0,\ b_{\gamma,2} \neq 0 \Rightarrow c_{\gamma,1} \neq 0$		1
	$a_{\gamma,1} = a_{\gamma,2} = b_{\gamma,2} = 0,\ b_{\gamma,1} \neq 0 \Rightarrow c_{\gamma,2} \neq 0$		2
	$a_{\gamma,1} = a_{\gamma,2} = 0,\ b_{\gamma,1} \neq 0,\ b_{\gamma,2} \neq 0 \Rightarrow det\, H_\gamma^{c,b} \neq 0$		3
	$a_{\gamma,1} = b_{\gamma,1} = b_{\gamma,2} = 0,\ a_{\gamma,2} \neq 0 \Rightarrow c_{\gamma,1} \neq 0$	×	4
	$a_{\gamma,1} = b_{\gamma,1} = 0,\ a_{\gamma,2} \neq 0,\ b_{\gamma,2} \neq 0 \Rightarrow c_{\gamma,1} \neq 0$		5
	$a_{\gamma,1} \neq 0,\ a_{\gamma,2} = b_{\gamma,1} = b_{\gamma,2} = 0 \Rightarrow c_{\gamma,2} \neq 0$	×	6
	$a_{\gamma,1} \neq 0,\ a_{\gamma,2} = b_{\gamma,2} = 0,\ b_{\gamma,1} \neq 0 \Rightarrow c_{\gamma,2} \neq 0$		7
	$a_{\gamma,1} \neq 0,\ a_{\gamma,2} \neq 0,\ b_{\gamma,1} = b_{\gamma,2} = 0 \Rightarrow det\, H_\gamma^{c,a} \neq 0$	×	8
	$a_{\gamma,1} \neq 0,\ a_{\gamma,2} \neq 0,\ b_{\gamma,1} \neq 0,\ b_{\gamma,2} \neq 0$		9

and 17–22) are impossible because, in one instance of the execution of the DOALL loop J in the given loop nest, no two iterations are not allowed to write to the same array elements. The other six patterns labeled as "$*$" and "$\star$" (2, 4, 6, 8, 10, and 16) are impossible because of the assumptions of

$$N_2 > \max\left(\left|\frac{c_{\gamma,1}}{\mathrm{GCD}(b_{\gamma,1}, c_{\gamma,1})}\right|, \left|\frac{c_{\gamma,2}}{\mathrm{GCD}(b_{\gamma,2}, c_{\gamma,2})}\right|\right),$$

$$N_3 > \max\left(\left|\frac{b_{\gamma,1}}{\mathrm{GCD}(b_{\gamma,1}, c_{\gamma,1})}\right|, \left|\frac{b_{\gamma,2}}{\mathrm{GCD}(b_{\gamma,2}, c_{\gamma,2})}\right|\right).$$

So we only need to consider the remaining six possible patterns of coefficients (1, 3, 5, 7, 9, and 15).

Similarly, when $det\, H_\gamma^{b,a} = 0$ and at least one of $det\, H_\gamma^{c,b}$ and $det\, H_\gamma^{c,a}$ is non-zero, all patterns of coefficients are listed in Table 2. The three patterns labeled as "$\times$" (4, 6, and 8) are impossible because of the DOALL loop J in our model. The rest to be considered are the other six patterns (1–3, 5, 7, and 9).

Sufficiency: For the six remaining patterns of coefficients in Table 1, we have $c_{\gamma,1} = c_{\gamma,2} = 0$, and at least one of $b_{\gamma,1}$ and $b_{\gamma,2}$ being non-zero. If there are iterations (i, j) and (i', j') in the reduced iteration space, which satisfy $T_{i,j}\ \delta\ T_{i',j'}$, then according to Lemma 2, we have

$$H_\gamma \begin{bmatrix} i' - i \\ j' - j \\ k' - k \end{bmatrix} = \mathbf{0}.$$

Therefore,

$$H_\gamma^{a,b} \begin{bmatrix} i' - i \\ j' - j \end{bmatrix} = \mathbf{0},$$

because $c_{\gamma,1} = c_{\gamma,2} = 0$. On the other hand, if $a_{\gamma,1} = a_{\gamma,2} = 0$, then from the above equations we have $j' - j = 0$.

For the remaining six possible patterns of coefficients in Table 2, we have at least one of $b_{\gamma,1}$ and $b_{\gamma,2}$ being non-zero. It is clear that $k'-k=0$. So, we have

$$H_{\gamma}^{a,b}\begin{bmatrix} i'-i \\ j'-j \end{bmatrix}=\mathbf{0}.$$

Again, we have $j'-j=0$ if $a_{\gamma,1}=a_{\gamma,2}=0$.

Necessity: If $a_{\gamma,1}\neq 0$, then the possible patterns of coefficients are 1 and 9 in Table 1 and 7 and 9 in Table 2. For Pattern 1 in Table 1 and Pattern 9 in Table 2, we have $a_{\gamma,2}(i'-i)+b_{\gamma,2}(j'-j)=0$ because $a_{\gamma,1}(i'-i)+b_{\gamma,1}(j'-j)=0$ and $det\, H_{\gamma}^{b,a}=0$. Also, for Pattern 9 in Table 1 and Pattern 7 in Table 2, we have $a_{\gamma,2}(i'-i)+b_{\gamma,2}(j'-j)=0$ because $a_{\gamma,2}=b_{\gamma,2}=0$. So, there exist $k=k'$, such that

$$H_{\gamma}\begin{bmatrix} i'-i \\ j'-j \\ k'-k \end{bmatrix}=\mathbf{0}.$$

So, we have $T_{i,j}\ \delta\ T_{i',j'}$.

Similarly, we can prove the other two cases, one for $a_{\gamma,2}\neq 0$ and the other for $a_{\gamma,1}=a_{\gamma,2}=0$. □

We now define staggering parameters whose linear combinations define a set of (i,j) iterations, i.e. tasks.

Definition 4. Given iteration (i,j) in the reduced iteration space, we let $S_{i,j}$ denote the following set of iterations in the space:

$$S_{i,j}=\left\{\left(i+\sum_{\gamma=1}^{m} r_{\gamma}L_{\gamma,1},\, j+\sum_{\gamma=1}^{m} r_{\gamma}L_{\gamma,2}\right)\,\middle|\, r_{\gamma}\in Z, 1\leqslant\gamma\leqslant m\right\},$$

where $L_{\gamma,1}(L_{\gamma,1}\neq 0)$ and $L_{\gamma,2}(1\leqslant\gamma\leqslant m)$ are defined as:

(1) $L_{\gamma,1}=det\, H_{\gamma}^{b,c}/\mathrm{GCD}_{\gamma}$ and $L_{\gamma,2}=det\, H_{\gamma}^{c,a}/\mathrm{GCD}_{\gamma}$, if $det\, H_{\gamma}^{b,a}\neq 0$, with GCD_{γ} equal to GCD $(det\, H_{\gamma}^{b,c}, det\, H_{\gamma}^{c,a}, det\, H_{\gamma}^{b,a})$ or equal to $-$GCD $(det\, H_{\gamma}^{b,c}, det\, H_{\gamma}^{c,a}, det\, H_{\gamma}^{b,a})$ to guarantee $L_{\gamma,1}>0$;

(2) $L_{\gamma,1}=b_{\gamma,1}/\mathrm{GCD}_{\gamma}$ and $L_{\gamma,2}=-a_{\gamma,1}/\mathrm{GCD}_{\gamma}$, if $det\, H_{\gamma}^{b,a}=0$ and $a_{\gamma,1}\neq 0$, with GCD_{γ} equal to GCD $(a_{\gamma,1},b_{\gamma,1})$ or equal to $-$GCD $(a_{\gamma,1},b_{\gamma,1})$ to guarantee $L_{\gamma,1}>0$;

(3) $L_{\gamma,1}=b_{\gamma,2}/\mathrm{GCD}_{\gamma}$ and $L_{\gamma,2}=-a_{\gamma,2}/\mathrm{GCD}_{\gamma}$, if $det\, H_{\gamma}^{b,a}=0$ and $a_{\gamma,2}\neq 0$, with GCD_{γ} equal to GCD $(a_{\gamma,2},b_{\gamma,2})$ or equal to $-$GCD $(a_{\gamma,2},b_{\gamma,2})$ to guarantee $L_{\gamma,1}>0$;

(4) $L_{\gamma,1}=1$ and $L_{\gamma,2}=0$, if $a_{\gamma,1}=a_{\gamma,2}=0$.

We call $(L_{\gamma,1},L_{\gamma,2})$ the pair of *staggering parameters* corresponding to linear function $\vec{h}_{\gamma}$. If there exist no data dependences between the given pair of references, we define $(L_{\gamma,1},L_{\gamma,2})$ as $(0,0)$.

The following theorem, derived from Theorems 1 and 2 and Definition 4, states that we can use the staggering parameters to uniquely partition the tasks into independent sets.

Theorem 3. *$S_{i,j}$ as defined above satisfies*:
(1) *if* $(i,j) \in S_{i',j'}$ *and* $(i,j) \in S_{i'',j''}$ *then* $S_{i',j'} = S_{i'',j''}$,
(2) *if* $(i',j') \in S_{i,j}$ *then* $S_{i,j} = S_{i',j'}$, *and*
(3) *if* $(i',j') \in S_{i,j}$ *and* $T_{i',j'}\ \delta\ T_{i'',j''}$ *then* $(i'',j'') \in S_{i,j}$.

The theorem above indicates that $S_{i,j}$ includes all the iterations whose corresponding tasks have a data dependence relation with $T_{i,j}$. We call $S_{i,j}$ an *equivalence class* of the reduced iteration space. In order to eliminate true data sharing, tasks in the same equivalence class should be assigned to the same processor. We want to restructure the reduced iteration space such that tasks in the same equivalence class will appear in the same column. Each staggering parameter (L_1, L_2) computed for a dependent reference pair ensures that if we stagger the $(i+L_1)$th row in the reduced iteration space by $|L_2|$ columns to the right if $L_2 < 0$, or to the left if $L_2 > 0$, relative to the ith row, then the tasks involved in the dependence pair will be aligned in the same column. Different staggering parameters computed for different reference pairs may require different ways to stagger the iteration space. However, if these staggering parameters are in proportion, then staggering by the *unified staggering parameter* defined below will satisfy all the requirements simultaneously.

Definition 5. Given pairs of staggering parameters $(L_{1,1}, L_{1,2}), (L_{2,1}, L_{2,2}), \ldots, (L_{m,1}, L_{m,2})$ and $g = \mathrm{GCD}\ (L_{1,1},\ L_{2,1}, \ldots, L_{m,1})$, if we have $L_{k,1}/L_{k,2} = L_{j,1}/L_{j,2}$ for all (j,k) such that $1 \leqslant j,\ k \leqslant m$, then we call $(g, L_{1,2}/L_{1,1}g)$ the *unified staggering parameter*.

Lemma 2. *If the condition* $L_{k,1}/L_{k,2} = L_{j,1}/L_{j,2}$ $(1 \leqslant j,k \leqslant m)$ *in Definition* 5 *is true, then* (a) *the iterations* (i,j) *and* (i,k), *where* $j \neq k$, *belong to two different equivalence classes*; *and* (b) *the iterations* (i,j) *and* (i',k), *where* $0 < i' - i < g$, *belong to two different equivalence classes.*

Proof. (a) If (i,j) and (i,k) belong to the same equivalence class, then we have $(i,k) \in S_{i,j}$ because $(i,j) \in S_{i,j}$. We then have

$$i = i + \sum_{\gamma=1}^{m} r_\gamma L_{\gamma,1},$$

$$k = j + \sum_{\gamma=1}^{m} r_\gamma L_{\gamma,2}$$

and hence

$$\sum_{\gamma=1}^{m} r_\gamma L_{\gamma,1} = 0.$$

Since we assume

$$\frac{L_{k,1}}{L_{k,2}} = \frac{L_{j,1}}{L_{j,2}}$$

for any j and k $(1 \leqslant j,k \leqslant m)$, we have

$$\frac{L_{1,1}}{L_{1,2}} \sum_{\gamma=1}^{m} r_\gamma L_{\gamma,2} = \sum_{\gamma=1}^{m} \frac{L_{\gamma,1}}{L_{\gamma,2}} r_\gamma L_{\gamma,2} = \sum_{\gamma=1}^{m} r_\gamma L_{\gamma,1} = 0$$

and hence

$$\sum_{\gamma=1}^{m} r_\gamma L_{\gamma,2} = 0,$$

which leads to $k = j$, contradicting the assumption.

(b) If (i,j) and (i',k) belong to the same equivalence class then we have (i,j) and (i',k) both belonging to $S_{i,j}$, and thus

$$i' = i + \sum_{\gamma=1}^{m} r_\gamma L_{\gamma,1}.$$

Since $g = \mathrm{GCD}\ (L_{1,1}, L_{2,1}, \ldots, L_{m,1})$, g must divide $i' - i$, contradicting the premise that $i' - i < g$. The iterations (i,j) and (i',k), where $0 < i' - i < g$, belong to different equivalence classes. □

Theorem 4. *If the condition* $L_{k,1}/L_{k,2} = L_{j,1}/L_{j,2}$ $(1 \leqslant j,k \leqslant m)$ *in Definition* 5 *is true, then the reduced iteration space must be staggered according to the unified staggering parameter* $(g, (L_{1,2}/L_{1,1})g)$ *in order to reduce or eliminate data sharing among the tasks, i.e. the* $(i+g)$*th row in the reduced iteration space must be staggered by* $|(L_{1,2}/L_{1,1})g|$ *columns to the right if* $L_{1,2} < 0$, *or to the left if* $L_{1,2} > 0$, *relative to the* i*th row.*

Proof. Suppose $g = \mathrm{GCD}\ (L_{1,1}, L_{2,1}, \ldots, L_{m,1})$. According to Lemma 2, we only need to stagger the $(i+g)$th row related to the ith row in the reduced iteration space. There exists a linear combination of $L_{1,1}, L_{2,1}, \ldots, L_{m,1}$ such that

$$g = \sum_{\gamma=1}^{m} k_\gamma L_{\gamma,1}.$$

According to Definition 4, iteration $(i+g, j+\Delta j)$ belongs to $S_{i,j}$, where

$$\Delta j = \sum_{\gamma=1}^{m} k_\gamma L_{\gamma,2} = \sum_{\gamma=1}^{m} k_\gamma \frac{L_{\gamma,2}}{L_{\gamma,1}} L_{\gamma,1},$$

provided that $1 < i+g \leqslant N_1, 1 < j+\Delta j \leqslant N_2$. Since for any j and k $(1 \leqslant j,k \leqslant m)$

$$\frac{L_{k,1}}{L_{k,2}} = \frac{L_{j,1}}{L_{j,2}},$$

we have

$$\Delta j = \frac{L_{1,2}}{L_{1,1}} \sum_{\gamma=1}^{m} k_\gamma L_{\gamma,1} = \frac{L_{1,2}}{L_{1,1}} g,$$

which means that iteration (i,j) and $(i+g, j+(L_{1,2}/L_{1,1})g)$ are in the same equivalence class. By Lemma 2(a), if there exists any $(i+g, j')$ that belongs to $S_{i,j}$ then $j' = \Delta j$. Hence, the $(i+g)$th row must be staggered by $|\Delta j|$ columns to the right if $L_{1,2} < 0$, or to the left if $L_{1,2} > 0$, relative to the ith row, to guarantee that there will be no data sharing between different columns. □

If a given loop nest satisfies the condition $L_{k,1}/L_{k,2} = L_{j,1}/L_{j,2}$ $(1 \leqslant j,k \leqslant m)$ in Definition 5, then, according to Theorem 4 above, the reduced iteration space can be transformed into a *staggered and reduced iteration space* (*SRIS*) by leaving the first g rows unchanged, $g = \text{GCD}\ (L_{1,1}, L_{2,1}, \ldots, L_{m,1})$, and by staggering each of the remaining rows using the unified staggering parameter. There will be no data dependences between different columns in the SRIS.

However, if the staggering parameters are not in proportion, i.e, if there exist (j,k) such that $1 \leqslant j,k \leqslant m$ and $L_{k,1}/L_{k,2} \neq L_{j,1}/L_{j,2}$, then we can no longer obtain a unique unified staggering parameter. Moreover, staggering alone is no longer sufficient for eliminating data dependences between the different columns in the restructured iteration space. This is because some tasks in the same equivalence class are still in different columns. We perform a procedure called *aligned processor folding*, or *aligned folding* in short, which stacks these columns onto each other. We will discuss staggering first.

Definition 6. Given pairs of staggering parameters $(L_{1,1}, L_{1,2}), (L_{2,1}, L_{2,2}), \ldots, (L_{m,1}, L_{m,2})$ and $g = \text{GCD}\ (L_{1,1}, L_{2,1}, \ldots, L_{m,1})$, suppose there exists (j,k) such that $1 \leqslant j, k \leqslant m$ and $L_{k,1}/L_{k,2} \neq L_{j,1}/L_{j,2}$. According to the theory of numbers [30], there exist integers $a_1, a_2, \ldots, a_m$ that satisfy

$$g = \sum_{\gamma=1}^{m} a_\gamma L_{\gamma,1}.$$

Let $g' = \sum_{\gamma=1}^{m} a_\gamma L_{\gamma,2}$. We call (g, g') a pair of *unified staggering parameters*.

Note that since the m-tuple $(a_1, a_2, \ldots, a_m)$ is not necessarily unique, the (g, g') pair may not be unique either.

In the introduction, we showed an example of using aligned folding to eliminate true data sharing. The following two algorithms compute the folding factor, d.

Algorithm 1 (*Jin et al.* [21])**:**

Input: Pairs of staggering parameters $(L_{1,1}, L_{1,2}), (L_{2,1}, L_{2,2}), \ldots, (L_{m,1}, L_{m,2})$.

Output: The folding factor d.

Step 1: For each 2-element subset, $\{L_{i,1}, L_{j,1}\}$, of $\{L_{1,1}, L_{2,1}, \ldots, L_{m,1}\}$, compute

$$d_2\langle L_{i,1}, L_{j,1}\rangle = \frac{L_{j,1}L_{i,2} - L_{i,1}L_{j,2}}{\text{GCD}\ (L_{i,1}, L_{j,1})}.$$

Let $d_2 = \text{GCD}$ of all such $d_2\langle L_{i,1}, L_{j,1}\rangle$.

Step 2: For each j-element subset, $\{L_{i_1,1}, L_{i_2,1}, \ldots, L_{i_j,1}\}$, where $3 \leqslant j \leqslant m$, pick any element, say $L_{i_1,1}$, and compute

$$r_1 = \frac{\text{LCD}(\text{GCD } (L_{i_2,1}, \ldots, L_{i_j,1}), L_{i_1,1})}{L_{i_1,1}}.$$

Using the Euclidean Algorithm, compute integers $b_2, b_3, \ldots, b_j$ such that

$$\text{GCD } (L_{i_2,1}, \ldots, L_{i_j,1}) = \sum_{\gamma=2}^{j} b_\gamma L_{i_\gamma,1}.$$

Apply Algorithm 2 below to find nonzero integers $r_2, \ldots, r_j$ such that

$$r_1 L_{i_1,1} = \sum_{\gamma=2}^{j} r_\gamma L_{i_\gamma,1}.$$

Let

$$d_j\langle L_{i_1,1}, L_{i_2,1}, \ldots, L_{i_j,1}\rangle = r_1 L_{i_1,2} - \sum_{\gamma=2}^{j} r_\gamma L_{i_\gamma,2}.$$

Step 3: For j from 3 to m, compute

$$d_j = \text{GCD } (d_j\langle L_{i_1,1}, L_{i_2,1}, \ldots, L_{i_j,1}\rangle \mid \text{for all distinct } \langle L_{i_1,1}, L_{i_2,1}, \ldots, L_{i_j,1}\rangle).$$

Step 4: $d = \text{GCD } (d_2, d_3, \ldots, d_m)$.

As will be established later, d is unique regardless of the choice of $L_{i_1,1}$ in Step 2.

To calculate the folding factor d, non-zero integers $r_2, \ldots, r_j$ need to be found in Algorithm 1 from the integer coefficients $b_2, b_3, \ldots, b_j$ computed by the Euclidean Algorithm. Algorithm 2 is therefore invoked to derive a group of non-zero integer coefficients from a group of any integer coefficients of a linear expression.

Algorithm 2 (*Jin et al.* [21])**:**

Input: Non-zero positive integers $p, L_1, L_2, \ldots, L_p$, integer $x \geqslant 0$, and integers $a_1, a_2, \ldots, a_p$ such that $x = \sum_{i=1}^{p} a_i L_i$.

Output: non-zero integers $b_1, b_2, \ldots, b_p$ such that $x = \sum_{i=1}^{p} b_i L_i$.

Step 1: If there are an even number of zero coefficients $a_{i_1}, a_{i_2}, \ldots, a_{i_{2k}}$ $(0 \leqslant 2k \leqslant p)$ among $a_1, a_2, \ldots, a_p$, then let

$$b_j = a_j \ (a_j \neq 0),$$

$$b_{i_{2l-1}} = L_{i_{2l}} \ (1 \leqslant l \leqslant k),$$

$$b_{i_{2l}} = -L_{i_{2l-1}} \ (1 \leqslant l \leqslant k).$$

Step 2: If there are an odd number of zero coefficients $a_{i_1}, a_{i_2}, \ldots, a_{i_{2k+1}}$ $(0 \leqslant 2k+1 \leqslant p)$ among $a_1, a_2, \ldots, a_p$, then let

$$b_j = a_j \ (a_j \neq 0),$$
$$b_{i_{2l-1}} = L_{i_{2l}} \ (1 \leqslant l \leqslant k-1),$$
$$b_{i_{2l}} = -L_{i_{2l-1}} \ (1 \leqslant l \leqslant k-1),$$
$$b_{i_{2k-1}} = L_{i_{2k}},$$
$$b_{i_{2k}} = -(L_{i_{2k-1}} + L_{i_{2k+1}}),$$
$$b_{i_{2k+1}} = L_{i_{2k}}.$$

Obviously, the non-zero integers $b_1, b_2, \ldots, b_p$ computed by Algorithm 2 satisfy $x = \sum_{i=1}^{p} b_i L_i$.

After d is computed, we partition the SRIS into n chunks, where

$$n = \left\lceil \frac{N_2 + (\lceil N_1/g \rceil - 1)g'}{d} \right\rceil,$$

which is the total number of columns in the SRIS divided by the folding factor d. These d-wide chunks are stacked onto each other to form a folded iteration space of width d. As we will explain later, the tasks in different columns after aligned folding the SRIS with d are independent. Moreover, the product of d and g equals the number of equivalence classes.

We need to establish two important facts. First, after folding with d, the tasks in different columns are independent. Second, the folding factor d computed by Algorithm 1 is the largest number of independent columns possible as the result of folding the SRIS with a constant value. The first fact is established by a number of theorems which are based on the following definition.

Definition 7. Given an iteration (i, j) in the reduced iteration space, pairs of staggering parameters $(L_{1,1}, L_{1,2}), (L_{2,1}, L_{2,2}), \ldots, (L_{m,1}, L_{m,2})$, and the pair of unified staggering parameters (g, g'), and suppose $a_1, a_2, \ldots, a_m$ are integers that satisfy

$$g = \sum_{\gamma=1}^{m} a_\gamma L_{\gamma,1},$$

a set of iterations $S'_{i,j}$ is constructed as follows:

(1) For any integer r, iteration $(i + rg, j + rg')$ in the space belongs to $S'_{i,j}$, where $g' = \sum_{\gamma=1}^{m} a_\gamma L_{\gamma,2}$;

(2) If there exist integers $r_1, r_2, \ldots, r_m$, not all zero, integer r, and iterations $(i', j'), (i', k')$ $(k' \neq j')$ in the space, such that

$$k' = j' + \sum_{\gamma=1}^{m} r_\gamma L_{\gamma,2}, \quad \sum_{\gamma=1}^{m} r_\gamma L_{\gamma,1} = 0$$

and

$$i = i' + rg, \quad j = j' + rg' \left(g' = \sum_{\gamma=1}^{m} a_\gamma L_{\gamma,2} \right),$$

then $S'_{i,j} = S'_{i,j} \cup S'_{i',k'}$.

The following three lemmas and Theorem 5 show that $S'_{i,j}$ is the same as the equivalence class $S_{i,j}$. From the process of constructing $S'_{i,j}$, we immediately have the following lemma.

Lemma 3. *Given iterations (i,j) and (i',j') in the reduced iteration space, and a pair of unified staggering parameters (g,g'), if there exists integer r such that*

$$i = i' + rg,$$

$$j = j' + rg',$$

then $S'_{i,j} = S'_{i',j'}$.

Lemma 4. *Given iterations (i',j') and (i',k') $(j' \neq k')$ in the reduced iteration space, if there exist integers $r_1, r_2, \ldots, r_m$, not all zero, such that*

$$k' = j' + \sum_{\gamma=1}^{m} r_\gamma L_{\gamma,2},$$

$$\sum_{\gamma=1}^{m} r_\gamma L_{\gamma,1} = 0,$$

then $S'_{i',j'} = S'_{i',k'}$.

Proof. Since there exist integers $r_1, r_2, \ldots, r_m$, not all zero, such that

$$k' = j' + \sum_{\gamma=1}^{m} r_\gamma L_{\gamma,2}, \quad \sum_{\gamma=1}^{m} r_\gamma L_{\gamma,1} = 0$$

and

$$i' = i' + 0 \times g, \quad j' = j' + 0 \times g',$$

we have $S'_{i',j'} = S'_{i',j'} \cup S'_{i',k'}$ according to Definition 7. Hence, we have $S'_{i',j'} \supseteq S'_{i',k'}$. Similarly, we can derive $S'_{i',k'} \supseteq S'_{i',j'}$ because

$$j' = k' - \sum_{\gamma=1}^{m} r_\gamma L_{\gamma,2} = k' + \sum_{\gamma=1}^{m} (-r_\gamma) L_{\gamma,2}, \quad \sum_{\gamma=1}^{m} (-r_\gamma) L_{\gamma,1} = -\sum_{\gamma=1}^{m} r_\gamma L_{\gamma,1} = 0$$

and

$$i' = i' + 0 \times g, \quad k' = k' + 0 \times g'.$$

Therefore $S'_{i',j'} = S'_{i',k'}$. □

Lemma 5. *Given iterations* (i,j) *and* (i',j') *in the reduced iteration space, if* $(i',j') \in S'_{i,j}$ *then* $S'_{i,j} = S'_{i',j'}$.

Proof. According to the constructing process of $S'_{i,j}$ in Definition 7, there exists an integer $n \leqslant 0$ such that $S'_{i,j}$ is the union of the following $n+1$ sets of iterations in the reduced iteration space:

(1) $\{(i+rg, j+rg') \mid r \in Z\}$,

(2) $\{(i_1 + r'_2 g,\ k_1 + r'_2 g') \mid i_1 = i + r'_1 g,\ j_1 = j + r'_1 g',\ \sum_{\gamma=1}^{m} r_{1,\gamma} L_{\gamma,1} = 0,\ k_1 = j_1 + \sum_{\gamma=1}^{m} r_{1,\gamma} L_{\gamma,2},\ r'_1, r'_2, r_{1,1}, \ldots, r_{1,m} \in Z\}$,

$\ldots$

(n) $\{(i_n + r'_{n+1} g,\ k_n + r'_{n+1} g') \mid i_1 = i + r'_1 g,\ j_1 = j + r'_1 g',\ \sum_{\gamma=1}^{m} r_{1,\gamma} L_{\gamma,1} = 0,\ k_1 = j_1 + \sum_{\gamma=1}^{m} r_{1,\gamma} L_{\gamma,2},\ i_2 = i_1 + r'_2 g,\ j_2 = k_1 + r'_2 g',\ \sum_{\gamma=1}^{m} r_{2,\gamma} L_{\gamma,1} = 0,\ k_2 = j_2 + \sum_{\gamma=1}^{m} r_{2,\gamma} L_{\gamma,2},$

$\ldots$

$i_n = i_{n-1} + r'_n g,\ j_n = k_{n-1} + r'_n g',\ \sum_{\gamma=1}^{m} r_{n,\gamma} L_{\gamma,1} = 0,\ k_n = j_n + \sum_{\gamma=1}^{m} r_{n,\gamma} L_{\gamma,2},$

$r'_1, r'_2, \ldots, r'_{n+1}, r_{1,1}, r_{1,2}, \ldots, r_{1,m}, \ldots, r_{n,1}, r_{n,2}, \ldots, r_{n,m} \in Z\}$.

If there exists $r' \in Z$ such that

$$i' = i + r'g,$$

$$j' = j + r'g',$$

then $S'_{i,j} = S'_{i',j'}$ according to Lemma 3. Otherwise, if there exist integers $r''_1, \ldots, r''_{p+1}$, $r'_{1,1}, \ldots, r'_{1,m}, \ldots, r'_{p,1}, \ldots, r'_{p,m}$ $(1 \leqslant p \leqslant n)$ such that

$$i_1 = i + r''_1 g, \quad j_1 = j + r''_1 g', \quad \sum_{\gamma=1}^{m} r'_{1,\gamma} L_{\gamma,1} = 0, \quad k_1 = j_1 + \sum_{\gamma=1}^{m} r'_{1,\gamma} L_{\gamma,2},$$

$\ldots$

$$i_p = i_{p-1} + r''_p g, \quad j_p = k_{p-1} + r''_p g', \quad \sum_{\gamma=1}^{m} r'_{p,\gamma} L_{\gamma,1} = 0, \quad k_p = j_p + \sum_{\gamma=1}^{m} r'_{p,\gamma} L_{\gamma,2},$$

$$i' = i_p + r''_{p+1} g,$$

$$j' = k_p + r''_{p+1} g',$$

we have

$$S'_{ij} = S'_{ij} \cup S'_{i_1,k_1},$$

$\ldots$

$$S'_{i_{p-1},k_{p-1}} = S'_{i_{p-1},k_{p-1}} \cup S'_{i_p,k_p},$$

according to Definition 7. Therefore, we have

$$S'_{i,j} \supseteq S'_{i_1,k_1} \supseteq \cdots \supseteq S'_{i_{p-1},k_{p-1}} \supseteq S'_{i_p,k_p}$$

and similarly,

$$S'_{i',j'} = S'_{i',j'} \cup S'_{i_p,j_p},$$

$$\cdots$$

$$S'_{i_2,j_2} = S'_{i_2,j_2} \cup S'_{i_1,j_1}.$$

Thus,

$$S'_{i_1,j_1} \subseteq S'_{i_2,j_2} \subseteq \cdots \subseteq S'_{i_p,j_p} \subseteq S'_{i',j'}.$$

Since $S'_{i,j} = S'_{i_1,j_1}$ and $S'_{i',j'} = S'_{i_p,k_p}$, according to Lemma 3, we have $S'_{i,j} \subseteq S'_{i',j'}$ and $S'_{i,j} \supseteq S'_{i',j'}$. Therefore, we have $S'_{i,j} = S'_{i',j'}$. □

Theorem 5. *Given pairs of staggering parameters* $(L_{1,1}, L_{1,2}), (L_{2,1}, L_{2,2}), \ldots, (L_{m,1}, L_{m,2})$, *we have*

$$S_{i,j} = S'_{i,j}$$

for any iteration (i,j) *in the reduced iteration space.*

Proof. We first prove that $S_{i,j} \subseteq S'_{i,j}$.

Since for any iteration $(i',j') \in S_{i,j}$, there exist integers $r_1, r_2, \ldots, r_m$ such that

$$i' = i + \sum_{\gamma=1}^{m} r_\gamma L_{\gamma,1},$$

$$j' = j + \sum_{\gamma=1}^{m} r_\gamma L_{\gamma,2},$$

according to Definition 4, we only need to prove

$$\left(i + \sum_{\gamma=1}^{m} r_\gamma L_{\gamma,1}, j + \sum_{\gamma=1}^{m} r_\gamma L_{\gamma,2} \right) \in S'_{i,j}.$$

Given $(i'', j'') \in S'_{i,j}$, let us prove

$$(i'' + r_{\gamma'} L_{\gamma',1}, j'' + r_{\gamma'} L_{\gamma',2}) \in S'_{i,j} (1 \leqslant \gamma' \leqslant m).$$

If $r_{\gamma'} = 0$, then we have

$$(i'' + r_{\gamma'} L_{\gamma',1}, j'' + r_{\gamma'} L_{\gamma',2}) = (i'', j'') \in S'_{i,j}.$$

Otherwise we have $r_{\gamma'} \neq 0$ and

$$i'' + r_{\gamma'} L_{\gamma',1} = i'' + r_{\gamma'} \frac{L_{\gamma',1}}{g} g + \left(r_{\gamma'} L_{\gamma',1} - r_{\gamma'} \frac{L_{\gamma',1}}{g} \sum_{\gamma=1}^{m} a_\gamma L_{\gamma,1} \right),$$

$$j'' + r_{\gamma'} L_{\gamma',2} = j'' + r_{\gamma'} \frac{L_{\gamma',1}}{g} g + \left(r_{\gamma'} L_{\gamma',2} - r_{\gamma'} \frac{L_{\gamma',1}}{g} \sum_{\gamma=1}^{m} a_\gamma L_{\gamma,2} \right).$$

We have $(i'' + (r_{\gamma'} L_{\gamma',1}/g)g,\ j'' + (r_{\gamma'} L_{\gamma',1}/g)g') \in S'_{i'',j''}$ according to Definition 7 and have $S'_{i,j} = S'_{i'',j''}$ according to the assumption $(i'', j'') \in S'_{i,j}$ and Lemma 5. Hence, we have

$$\left(i'' + r_{\gamma'} \frac{L_{\gamma',1}}{g} g, j'' + r_{\gamma'} \frac{L_{\gamma',1}}{g} g' \right) \in S'_{i,j}.$$

Furthermore, according to Lemmas 4 and 5, we have

$$\begin{aligned} &\left(i'' + \left(r_{\gamma'} \frac{L_{\gamma',1}}{g} \right) g, j'' + \left(r_{\gamma'} \frac{L_{\gamma',1}}{g} \right) g' - \left(r_{\gamma'} \frac{L_{\gamma',1}}{g} a_1 \right) L_{1,2} - \left(r_{\gamma'} \frac{L_{\gamma',1}}{g} a_2 \right) L_{2,2} \right. \\ &\quad - \cdots - \left(r_{\gamma'} \frac{L_{\gamma',1}}{g} a_{\gamma'-1} \right) L_{\gamma'-1,2} + \left(r_{\gamma'} - r_{\gamma'} \frac{L_{\gamma',1}}{g} a_{\gamma'} \right) L_{\gamma',2} \\ &\quad \left. - \left(r_{\gamma'} \frac{L_{\gamma',1}}{g} a_{\gamma'+1} \right) L_{\gamma'+1,2} - \cdots - \left(r_{\gamma'} \frac{L_{\gamma',1}}{g} a_m \right) L_{m,2} \right) \in S'_{i,j}, \end{aligned}$$

because

$$\begin{aligned} &- \left(r_{\gamma'} \frac{L_{\gamma',1}}{g} a_1 \right) L_{1,1} - \left(r_{\gamma'} \frac{L_{\gamma',1}}{g} a_2 \right) L_{2,1} - \cdots - \left(r_{\gamma'} \frac{L_{\gamma',1}}{g} a_{\gamma'-1} \right) L_{\gamma'-1,1} \\ &\quad + \left(r_{\gamma'} - r_{\gamma'} \frac{L_{\gamma',1}}{g} a_{\gamma'} \right) L_{\gamma',1} \left(r_{\gamma'} \frac{L_{\gamma',1}}{g} a_{\gamma'+1} \right) L_{\gamma'+1,1} - \cdots - \left(r_{\gamma'} \frac{L_{\gamma',1}}{g} a_m \right) L_{m,1} \\ &\quad = r_{\gamma'} L_{\gamma',1} - r_{\gamma'} \frac{L_{\gamma',1}}{g} \sum_{\gamma=1}^{m} a_\gamma L_{\gamma,1} = 0, \end{aligned}$$

that is,

$$\left(i'' + \left(r_{\gamma'} \frac{L_{\gamma',1}}{g} \right) g, j'' + \left(r_{\gamma'} \frac{L_{\gamma',1}}{g} \right) g' - \left(r_{\gamma'} \frac{L_{\gamma',1}}{g} \right) \sum_{\gamma=1}^{m} a_\gamma L_{\gamma,2} + r_{\gamma'} L_{\gamma',2} \right) \in S'_{i,j}.$$

Thus,

$$(i'' + r_{\gamma'} L_{\gamma',1}, j'' + r_{\gamma'} L_{\gamma',2}) \in S'_{i,j}.$$

Therefore, we have proved: if $(i'', j'') \in S'_{i,j}$ then $(i'' + r_{\gamma'} L_{\gamma',1}, j'' + r_{\gamma'} L_{\gamma',2}) \in S'_{i,j}$ $(1 \leqslant \gamma' \leqslant m)$.

Because $(i,j)\in S'_{i,j}$ (according to the definition of $S'_{i,j}$), we have

$$(i+r_1L_{1,1}, j+r_1L_{1,2})\in S'_{i,j},$$

$$(i+r_1L_{1,1}+r_2L_{2,1}, j+r_1L_{1,2}+r_2L_{2,2})\in S'_{i,j},$$

$$\cdots$$

$$\left(i+\sum_{\gamma=1}^{m} r_\gamma L_{\gamma,1}, j+\sum_{\gamma=1}^{m} r_\gamma L_{\gamma,2}\right)\in S'_{i,j}.$$

Thus, we have $(i',j')\in S'_{i,j}$ and consequently $S_{i,j}\subseteq S'_{i,j}$.

On the other hand, for any iteration $(i',j')\in S'_{i,j}$, following the Proof of Lemma 5, either there exists $r'\in Z$ such that

$$i'=i+r'g,$$

$$j'=j+r'g'$$

or there exist integers $r_1,\ldots,r_{p+1},r_{1,1},\ldots,r_{1,m},\ldots,r_{p,1},\ldots,r_{p,m}$ $(1\leqslant p\leqslant n)$ such that

$$i_1=i+r_1g,\quad j_1=j+r_1g',\quad \sum_{\gamma=1}^{m} r_{1\gamma}L_{\gamma,1}=0,\quad k_1=j_1+\sum_{\gamma=1}^{m} r_{1\gamma}L_{\gamma,2},$$

$$\cdots$$

$$i_p=i_{p-1}+r_pg,\quad j_p=k_{p-1}+r_pg',\quad \sum_{\gamma=1}^{m} r_{p,\gamma}L_{\gamma,1}=0,\quad k_p=j_p+\sum_{\gamma=1}^{m} r_{p,\gamma}L_{\gamma,2},$$

$$i'=i_p+r_{p+1}g,$$

$$j'=k_p+r_{p+1}g'.$$

Hence, we have

$$i'=i+\sum_{\gamma=1}^{m}(r'a_\gamma)L_{\gamma,1},$$

$$j'=j+\sum_{\gamma=1}^{m}(r'a_\gamma)L_{\gamma,2}$$

or

$$\begin{aligned} i' &= i_p+r_{p+1}g=i_{p-1}+r_pg+r_{p+1}g=\cdots=i+(r_1+\cdots+r_{p+1})g\\ &= i+(r_1+\cdots+r_{p+1})g+\sum_{\gamma=1}^{m} r_{1,\gamma}L_{\gamma,1}+\cdots+\sum_{\gamma=1}^{m} r_{p,\gamma}L_{\gamma,1}\\ &= i+\sum_{\gamma=1}^{m}((r_1+\cdots+r_{p+1})a_\gamma+r_{1,\gamma}+\cdots+r_{p,\gamma})L_{\gamma,1}, \end{aligned}$$

$$
\begin{aligned}
j' &= k_p + r_{p+1}g' = j_p + \sum_{\gamma=1}^{m} r_{p,\gamma}L_{\gamma,2} + r_{p+1}g' = k_{p-1} + \sum_{\gamma=1}^{m} r_{p,\gamma}L_{\gamma,2} + (r_p + r_{p+1})g' \\
&= j_{p-1} + \sum_{\gamma=1}^{m} r_{p-1,\gamma}L_{\gamma,2} + \sum_{\gamma=1}^{m} r_{p,\gamma}L_{\gamma,2} + (r_p + r_{p+1})g' = \cdots \\
&= j + \sum_{\gamma=1}^{m} r_{1,\gamma}L_{\gamma,2} + \cdots + \sum_{\gamma=1}^{m} r_{p,\gamma}L_{\gamma,2} + (r_1 + \cdots + r_{p+1})g' \\
&= j + \sum_{\gamma=1}^{m} ((r_1 + \cdots + r_{p+1})a_\gamma + r_{1,\gamma} + \cdots + r_{p,\gamma})L_{\gamma,2}.
\end{aligned}
$$

Therefore we have $(i', j') \in S_{i,j}$ according to Definition 4. Hence $S'_{i,j} \subseteq S_{i,j}$ and finally $S_{i,j} = S'_{i,j}$. □

Next, we establish that $S'_{i,j}$ is the result of staggering with (g, g') followed by folding with d. This is stated by Corollary 1 below.

Lemma 6. *Given pairs of staggering parameters* $(L_{i_1,1}, L_{i_1,2}), (L_{i_2,1}, L_{i_2,2}), \ldots, (L_{i_j,1}, L_{i_j,2})$, *suppose that* $r_1, r_2, \ldots, r_j$ *are integers that satisfy*

(1) $\sum_{\gamma=1}^{j} r_\gamma L_{i_\gamma,1} = 0$ $(r_1 > 0)$; *and*

(2) *if there exist integers* $r'_1, r'_2, \ldots, r'_j$ *satisfying* $\sum_{\gamma=1}^{j} r'_\gamma L_{i_\gamma,1} = 0$ $(r'_1 > 0)$, *then* $r_1 \leqslant r'_1$.

For any integers $r''_1, r''_2, \ldots, r''_j$ *satisfying*

$$\sum_{\gamma=1}^{j} r''_\gamma L_{i_\gamma,1} = 0 \quad (r''_1 > 0),$$

there exists an integer $k \geqslant 1$ *such that* $r''_1 = k r_1$.

Proof. If there exist integers $r''_1, r''_2, \ldots, r''_j$ satisfying

$$\sum_{\gamma=1}^{j} r''_\gamma L_{i_\gamma,1} = 0 \quad (r''_1 > 0)$$

and there is no integer $k \leqslant 1$ such that $r''_1 = k r_1$, then there exists integer k_1 such that

$$r''_1 = k_1 r_1 + q_1 \quad (0 < q_1 < r_1).$$

Because

$$\sum_{\gamma=1}^{j} r_\gamma L_{i_\gamma,1} = \sum_{\gamma=1}^{j} r''_\gamma L_{i_\gamma,1} = 0 \quad (r_1 > 0,\ r''_1 > 0),$$

we have

$$\sum_{\gamma=1}^{j} (r''_\gamma - k_1 r_\gamma)L_{i_\gamma,1} = \sum_{\gamma=1}^{j} r''_\gamma L_{i_\gamma,1} - k_1 \sum_{\gamma=1}^{j} r_\gamma L_{i_\gamma,1} = 0,$$

that is,

$$q_1 L_{i_1,1} + \sum_{\gamma=2}^{j} (r''_\gamma - k_1 r_\gamma) L_{i_\gamma,1} = 0$$

contrary to the fact that r_1 is the smallest integer satisfying $\sum_{\gamma=1}^{j} r_\gamma L_{i_\gamma,1} = 0$ $(r_1 > 0)$. □

Theorem 6. *If d is the folding factor determined by Algorithm 1, and $d' = \sum_{\gamma=1}^{m} r_\gamma L_{\gamma,2}$, where $r_1, r_2, \ldots, r_m$ are integers, not all zero, which satisfy*

$$\sum_{\gamma=1}^{m} r_\gamma L_{\gamma,1} = 0$$

then there exists an integer k such that $d' = kd$.

Proof. We proceed by induction on q, the number of non-zero integers in $r_1, r_2, \ldots, r_m$. Since $L_{\gamma,1} > 0$ $(\gamma = 1, 2, \ldots, m)$, we only need to consider the case: $q \leqslant 2$.

(1) If $q = 2$, then we have

$$d' = r_{i_1} L_{i_1,2} + r_{i_2} L_{i_2,2},$$

where integers r_{i_1} and r_{i_2} satisfy

$$r_{i_1} L_{i_1,1} + r_{i_2} L_{i_2,1} = 0 \quad \text{where } r_{i_1} \neq 0 \text{ and } r_{i_2} \neq 0.$$

According to Lemma 6, there exists an integer $k_1 \leqslant 1$ such that

$$r_{i_1} = k_1 \frac{L_{i_2,1}}{\mathrm{GCD}(L_{i_1,1}, L_{i_2,1})},$$

$$r_{i_2} = -k_1 \frac{L_{i_1,1}}{\mathrm{GCD}(L_{i_1,1}, L_{i_2,1})}.$$

So, we have

$$\begin{aligned} d' &= r_{i_1} L_{i_1,2} + r_{i_2} L_{i_2,2} = k_1 \frac{L_{i_2,1}}{\mathrm{GCD}(L_{i_1,1}, L_{i_2,1})} L_{i_1,2} - k_1 \frac{L_{i_1,1}}{\mathrm{GCD}(L_{i_1,1}, L_{i_2,1})} L_{i_2,2} \\ &= k_1 (d_2 \langle L_{i_1,1}, L_{i_2,1} \rangle). \end{aligned}$$

According to Algorithm 1, one further derives that there must exist integers k_2 and k_3 such that

$$d' = k_1 (d_2 \langle L_{i_1,1}, L_{i_2,1} \rangle) = k_1 k_2 d_2 = k_1 k_2 k_3 d.$$

Let $k = k_1 k_2 k_3$. We have $d' = kd$. Hence, our assertion is true for $q = 2$.

(2) Suppose that the assertion is true for all q $(2 \leqslant q < n,\ 2 < n \leqslant m)$, that is, if $d' = \sum_{\gamma=1}^{q} r_{i_\gamma} L_{i_\gamma,2}$ and non-zero coefficients $r_{i_1}, r_{i_2}, \ldots, r_{i_q}$ satisfy

$$\sum_{\gamma=1}^{q} r_{i_\gamma} L_{i_\gamma,1} = 0,$$

then there exists an integer k such that $d' = kd$.

(3) Consider the case of $q = n$. We assume that non-zero integers $r'_{i_1}, r'_{i_2}, \ldots, r'_{i_n}$ satisfy

$$\sum_{\gamma=1}^{n} r'_{i_\gamma} L_{i_\gamma,1} = 0 \quad (r'_{i_1} > 0)$$

and if there exist non-zero integers $r''_{i_1}, r''_{i_2}, \ldots, r''_{i_n}$ such that

$$\sum_{\gamma=1}^{n} r''_{i_\gamma} L_{i_\gamma,1} = 0 \quad (r''_{i_1} > 0),$$

then $r'_{i_\gamma} \leqslant r''_{i_\gamma}$. According to Lemma 7, if $d' = \sum_{\gamma=1}^{n} r_{i_\gamma} L_{i_\gamma,2}$ and non-zero coefficients $r_{i_1}, r_{i_2}, \ldots, r_{i_n}$ satisfy

$$\sum_{\gamma=1}^{n} r_{i_\gamma} L_{i_\gamma,1} = 0,$$

then there exists an integer k_4 (if $r_{i_1} > 0$ then $k_4 \geqslant 1$, else $k_4 \leqslant -1$) such that

$$r_{i_1} = k_4 r'_{i_1}.$$

So, we have

$$r_{i_1} L_{i_1,1} = k_4 r'_{i_1} L_{i_1,1} = -\sum_{\gamma=2}^{n} r_{i_\gamma} L_{i_\gamma,1} = -k_4 \sum_{\gamma=2}^{n} r'_{i_\gamma} L_{i_\gamma,1} - \sum_{\gamma=2}^{n} (r_{i_\gamma} - k_4 r'_{i_\gamma}) L_{i_\gamma,1},$$

where

$$\sum_{\gamma=2}^{n} (r_{i_\gamma} - k_4 r'_{i_\gamma}) L_{i_\gamma,1} = 0,$$

because

$$\sum_{\gamma=1}^{n} r'_{i_\gamma} L_{i_\gamma,1} = 0.$$

Correspondingly, d' can be expressed as

$$d' = \sum_{\gamma=1}^{n} r_{i_\gamma} L_{i_\gamma,2} = k_4 r'_{i_1} L_{i_1,2} + \sum_{\gamma=2}^{n} r_{i_\gamma} L_{i_\gamma,2} = k_4 \sum_{\gamma=1}^{n} r'_{i_\gamma} L_{i_\gamma,2} + \sum_{\gamma=2}^{n} (r_{i_\gamma} - k_4 r'_{i_\gamma}) L_{i_\gamma,2}.$$

If all the coefficients $(r_{i_\gamma} - k_4 r'_{i_\gamma})$ are equal to zero in

$$\sum_{\gamma=2}^{n} (r_{i_\gamma} - k_4 r'_{i_\gamma}) L_{i_\gamma,2},$$

then

$$d' = k_4 \sum_{\gamma=1}^{n} r'_{i_\gamma} L_{i_\gamma,2} = k_4 d_n \langle L_{i_1,1}, L_{i_2,1}, \ldots, L_{i_n,1} \rangle.$$

Hence, there exist integers k_5, k_6 such that $d' = k_4 k_5 d_n = k_4 k_5 k_6 d$. Let $k = k_4 k_5 k_6$, we have $d' = kd$. If the coefficients $(r_{i_\gamma} - k_4 r'_{i_\gamma})$ are not all equal to zero, then the number of the non-zero coefficients is less than n but no less than 2. By our inductive hypothesis, there exists an integer k' such that

$$\sum_{\gamma=2}^{n} (r_{i_\gamma} - k_4 r'_{i_\gamma}) L_{i_\gamma,2} = k'd.$$

Let $k = k_4 k_5 k_6 + k'$, we have $d' = k_4 k_5 k_6 d + k'd = (k_4 k_5 k_6 + k')d = kd$. □

Corollary 1. *The set $S'_{i,j}$ in Definition 7 satisfies*

$$S'_{i,j} = \{(i', j') \mid i' = i + rg,\ j' = j + rg' + kd\}$$

where k and r are integers, (g, g') is the unified staggering parameter in Definition 6, and d is the folding factor computed by Algorithm 1.

From the above result, the tasks in different columns after folding the SRIS with d are independent. Next, we establish with Theorem 7 that d computed by Algorithm 1 is the largest possible number of independent columns, as the result of aligned folding the SRIS with a constant number.

Theorem 7. *Given (i, j), we have $(i, j + d) \in S'_{i,j}$.*

Proof. According to how d is computed in Algorithm 1, there exist integers $r_1, r_2, \ldots, r_m$ such that $\sum_{i=1}^{m} r_i L_{i,2} = d$ and $\sum_{i=1}^{m} r_i L_{i,1} = 0$. By the definition of $S'_{i,j}$, $(i, j + d) \in S'_{i,j}$. □

To further simplify the process of the staggering and the folding of the reduced iteration space, the following theorem can be used to replace multiple pairs of staggering parameters, which are in proportion, with a single pair of staggering parameters.

Theorem 8. *Given pairs of staggering parameters $(L_{1,1}, L_{1,2}), (L_{2,1}, L_{2,2}), \ldots, (L_{m,1}, L_{m,2})$, where $(L_{i_1,1}, L_{i_1,2}), (L_{i_2,1}, L_{i_2,2}), \ldots, (L_{i_p,1}, L_{i_p,2})$ $(2 \leqslant p < m)$ are the pairs of staggering parameters satisfying*

$$\frac{L_{i_1,1}}{L_{i_1,2}} = \frac{L_{i_2,1}}{L_{i_2,2}} = \cdots = \frac{L_{i_p,1}}{L_{i_p,2}},$$

there exists an integer r satisfying

$$\sum_{\gamma=1}^{p} r_{i_\gamma} L_{i_\gamma,1} = rl,$$

$$\sum_{\gamma=1}^{p} r_{i_\gamma} L_{i_\gamma,2} = rl\frac{L_{i_1,2}}{L_{i_1,1}},$$

where $l = \mathrm{GCD}(L_{i_1,1}, L_{i_2,1}, \ldots, L_{i_p,1})$.

Proof. Since $l = \mathrm{GCD}(L_{i_1,1}, L_{i_2,1}, \ldots, L_{i_p,1})$ and

$$\frac{L_{i_1,1}}{L_{i_1,2}} = \frac{L_{i_2,1}}{L_{i_2,2}} = \cdots = \frac{L_{i_p,1}}{L_{i_p,2}},$$

we have

$$\sum_{\gamma=1}^{p} r_{i_\gamma} L_{i_\gamma,1} = \left(\sum_{\gamma=1}^{p} r_{i_\gamma}\frac{L_{i_\gamma,1}}{l}\right) l,$$

$$\sum_{\gamma=1}^{p} r_{i_\gamma} L_{i_\gamma,2} = \sum_{\gamma=1}^{p} r_{i_\gamma} L_{i_\gamma,1}\frac{L_{i_\gamma,2}}{L_{i_\gamma,1}} = \frac{L_{i_1,2}}{L_{i_1,1}}\sum_{\gamma=1}^{p} r_{i_\gamma} L_{i_\gamma,1} = \left(\sum_{\gamma=1}^{p} r_{i_\gamma}\frac{L_{i_\gamma,1}}{l}\right) l\frac{L_{i_1,2}}{L_{i_1,1}}.$$

Therefore, there exists an integer $r = \sum_{\gamma=1}^{p} r_{i_\gamma} L_{i_\gamma,1}/l$ satisfying $\sum_{\gamma=1}^{p} r_{i_\gamma} L_{i_\gamma,1} = rl$ and $\sum_{\gamma=1}^{p} r_{i_\gamma} L_{i_\gamma,2} = rlL_{i_1,2}/L_{i_1,1}$. □

We now estimate the time needed by the compiler to compute the pairs of staggering parameters, a pair of unified staggering parameters, and the folding factor. Suppose there are m reference pairs. The complexity of determining all the pairs of staggering parameters is $\mathrm{O}(m)$. A pair of unified staggering parameters (g, g') of these pairs of staggering parameters can be determined in $\mathrm{O}(m)$ with the Euclidean Algorithm. Let m' be the number of groups of staggering parameter pairs such that all pairs in the same group are in proportion (m' is very small in practice). According to Theorem 8, we only need to consider one representative from each group. The complexity of Algorithms 1 and 2 for computing the folding factor is $C_{m'}^2\mathrm{O}(2) + C_{m'}^3\mathrm{O}(3) + \cdots + C_{m'}^{m'}\mathrm{O}(m') = \mathrm{O}(m'2^{m'-1})$.

3.3. Theorems and proofs for an extended model

The theory we developed in the previous subsection can be extended to more general cases in which the subscript functions in two dependent references are not necessary the same. Suppose the following two linear functions

$$\vec{h}_1(i,j,k) = (f_1(i,j,k), g_1(i,j,k))$$

and

$$\vec{h}_2(i,j,k) = (f_2(i,j,k), g_2(i,j,k))$$

belong to the a pair of dependent references, where

$$\begin{bmatrix} f_t(i,j,k) \\ g_t(i,j,k) \end{bmatrix} = \begin{bmatrix} a_{t,1} & b_{t,1} & c_{t,1} & d_{t,1} \\ a_{t,2} & b_{t,2} & c_{t,2} & d_{t,2} \end{bmatrix} \begin{bmatrix} i \\ j \\ k \\ 1 \end{bmatrix} \quad (t=1,2).$$

Same as in Section 3.2, we denote

$$\begin{bmatrix} a_{t,1} & b_{t,1} & c_{t,1} \\ a_{t,2} & b_{t,2} & c_{t,2} \end{bmatrix} \text{ as } H_t \text{ and } \begin{bmatrix} x_{t,1} & y_{t,1} \\ x_{t,2} & y_{t,2} \end{bmatrix} \text{ as } H_t^{x,y},$$

with $x, y \in \{a,b,c\}$. In order to determine which iterations in the reduced iteration space are dependent due to this reference pair, we consider an affine transformation

$$\begin{bmatrix} i \\ j \\ k \end{bmatrix} = \begin{bmatrix} \alpha_1 & 0 & 0 \\ 0 & \alpha_2 & 0 \\ 0 & 0 & \alpha_3 \end{bmatrix} \begin{bmatrix} i' \\ j' \\ k' \end{bmatrix} + \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix},$$

such that the linear function $\vec{h}_2$ can be expressed as $\vec{h}_2(i,j,k) = (f_2(i,j,k), g_2(i,j,k))$, where

$$\begin{aligned} \begin{bmatrix} f_2(i,j,k) \\ g_2(i,j,k) \end{bmatrix} &= \begin{bmatrix} a_{2,1} & b_{2,1} & c_{2,1} & d_{2,1} \\ a_{2,2} & b_{2,2} & c_{2,2} & d_{2,2} \end{bmatrix} \\ &\quad \times \begin{bmatrix} \left(\begin{bmatrix} \alpha_1 & 0 & 0 \\ 0 & \alpha_2 & 0 \\ 0 & 0 & \alpha_3 \end{bmatrix} \begin{bmatrix} i' \\ j' \\ k' \end{bmatrix} + \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} \right)^{\mathrm{T}} & 1 \end{bmatrix}^{\mathrm{T}} \\ &= H_2 \begin{bmatrix} \begin{bmatrix} \alpha_1 & 0 & 0 \\ 0 & \alpha_2 & 0 \\ 0 & 0 & \alpha_3 \end{bmatrix} \begin{bmatrix} i' \\ j' \\ k' \end{bmatrix} + \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} \end{bmatrix} + \begin{bmatrix} d_{2,1} \\ d_{2,2} \end{bmatrix} \\ &= H_2 \begin{bmatrix} \alpha_1 & 0 & 0 \\ 0 & \alpha_2 & 0 \\ 0 & 0 & \alpha_3 \end{bmatrix} \begin{bmatrix} i' \\ j' \\ k' \end{bmatrix} + H_2 \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} + \begin{bmatrix} d_{2,1} \\ d_{2,2} \end{bmatrix}. \end{aligned}$$

We denote the last expression in the above equations as $\vec{h}_2'(i',j',k')$. In order to use the previous results from Section 3.2, we let $\vec{h}_2'(i',j',k')$ be identical to $\vec{h}_1(i',j',k')$, which implies

$$H_2 \begin{bmatrix} \alpha_1 & 0 & 0 \\ 0 & \alpha_2 & 0 \\ 0 & 0 & \alpha_3 \end{bmatrix} = H_1$$

and

$$H_2 \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} + \begin{bmatrix} d_{2,1} \\ d_{2,2} \end{bmatrix} = \begin{bmatrix} d_{1,1} \\ d_{1,2} \end{bmatrix}. \tag{1}$$

We can now apply the algorithms in Section 3.2 to $\vec{h}'_2$ and $\vec{h}_1$, which yield a pair of staggering parameters, say (L_1, L_2). For a given iteration (i', j'), let $(i'', j'') = (L_1, L_2) + (i', j')$. The iteration (i, j) must have a dependence with (i'', j'') before the affine transformation if and only if the iteration (i', j') has a dependence with (i'', j'') after the transformation. We denote the distance between (i, j) and (i'', j'') as (L'_1, L'_2), which can be calculated as

$$L'_1 = i'' - i = (1 - \alpha_1)i' + i'' - i' - \beta_1 = (1 - \alpha_1)i' + L_1 - \beta_1,$$

$$L'_2 = j'' - j = (1 - \alpha_2)j' + j'' - j' - \beta_2 = (1 - \alpha_2)j' + L_2 - \beta_2$$

or

$$L'_1 = i - i'' = (\alpha_1 - 1)i' + i' - i'' + \beta_1 = (\alpha_1 - 1)i' + \beta_1 - L_1,$$

$$L'_2 = j - j'' = (\alpha_2 - 1)j' + j' - j'' + \beta_2 = (\alpha_2 - 1)j' + \beta_2 - L_2.$$

such that $L'_1 > 0$.

If $\alpha_1 \neq 1$ or $\alpha_2 \neq 1$, then (L'_1, L'_2) will not be constant, meaning that the iterations cannot be aligned with a pair of constant staggering parameters. In common practice, since loop J is DOALL in our loop nest model, the two linear functions $\vec{h}_1$ and $\vec{h}_2$ will have the same coefficients for loop index variables I and J, which implies that $\alpha_1 = \alpha_2 = 1$. In this paper, we will consider the case of $\alpha_1 = \alpha_2 = 1$ only. We now have

$$L'_1 \overset{\alpha_1=1}{=} L_1 - \beta_1,$$

$$L'_2 \overset{\alpha_2=1}{=} L_2 - \beta_2$$

or

$$L'_1 \overset{\alpha_1=1}{=} \beta_1 - L_1,$$

$$L'_2 \overset{\alpha_2=1}{=} \beta_2 - L_2.$$

Given (β_1, β_2) fixed, L'_1 and L'_2 are two constants. We define (L'_1, L'_2) as a pair of staggering parameters in this case.

If Eq. (1) has a unique solution for (β_1, β_2), then we have a unique pair of staggering parameters (L'_1, L'_2). On the other hand, if there exist multiple solutions for (β_1, β_2), then the following theorem shows that under certain conditions, (L'_1, L'_2) determined by different (β_1, β_2) should be in proportion.

Theorem 9. *Assume $\alpha_1 = \alpha_2 = 1$. If the pair of staggering parameters (L_1, L_2) of the subscript function $\vec{h}_1$ after the affine transformation is a solution for (β_1, β_2) in Eq. (1), then (L'_1, L'_2), which is equal to $(\beta_1 - L_1, \beta_2 - L_2)$ if $\beta_1 - L_1 > 0$, or to $(L_1 - \beta_1, L_2 - \beta_2)$ if $L_1 - \beta_1 > 0$, is in proportion with (L_1, L_2), for any solution $(\beta_1, \beta_2, \beta_3)$ of Eq. (1).*

Proof. Since $(L_1-\beta_1, L_2-\beta_2)$ is in proportion with $(\beta_1-L_1, \beta_2-L_2)$, we only prove that $(\beta_1-L_1, \beta_2-L_2)$ is in proportion to (L_1, L_2), supposing that $\beta_1-L_1>0$. Every solution to Eq. (1) can be written as

$$\begin{bmatrix}\beta_1\\ \beta_2\\ \beta_3\end{bmatrix} = \begin{bmatrix}\xi_1\\ \xi_2\\ \xi_3\end{bmatrix} + \begin{bmatrix}L_1\\ L_2\\ \Delta k\end{bmatrix},$$

where (ξ_1, ξ_2, ξ_3) is a solution of the homogeneous system associated with Eq. (1), that is

$$H_2 \begin{bmatrix}\xi_1\\ \xi_2\\ \xi_3\end{bmatrix} = \mathbf{0}.$$

So, if $det\, H_1^{b,a} \neq 0$, we have

$$\xi_1 = \frac{det\, H_2^{c,b}}{det\, H_2^{b,a}}\xi_3 = \frac{1}{\alpha_3}\frac{det\, H_1^{c,b}}{det\, H_1^{b,a}}\xi_3,$$

$$\xi_2 = \frac{det\, H_2^{a,c}}{det\, H_2^{b,a}}\xi_3 = \frac{1}{\alpha_3}\frac{det\, H_1^{a,c}}{det\, H_1^{b,a}}\xi_3.$$

Suppose $L_2=L_1=0$. We have $det\, H_1^{a,c}=0$ and $det\, H_1^{c,b}=0$. Therefore $\beta_2-L_2=\xi_2=0$ and $\beta_1-L_1=\xi_1=0$. If $L_2=0$ and $L_1\neq 0$, then we have $det\, H_1^{a,c}=0$, and hence $\beta_2-L_2=\xi_2=0$. If $L_2\neq 0$, according to Definition 4, we have

$$\frac{\beta_1-L_1}{\beta_2-L_2} = \frac{\xi_1}{\xi_2} = \frac{det\, H_1^{c,b}}{det\, H_1^{a,c}} = \frac{L_1}{L_2}.$$

For the case of $det\, H_1^{b,a}=0$, as in Theorem 2, we have: (1) $a_{1,1}\xi_1 + b_{1,1}\xi_2 = 0$ for $a_{1,1}\neq 0$; (2) $a_{1,2}\xi_1 + b_{1,2}\xi_2 = 0$ for $a_{1,2}\neq 0$; (3) $\xi_2=0$ for $a_{1,1}=a_{1,2}=0$. Therefore, according to Definition 4, we also have (ξ_1, ξ_2) in proportion with (L_1, L_2). So $(\beta_1-L_1, \beta_2-L_2)$ and $(L_1-\beta_1, L_2-\beta_2)$ are in proportion with (L_1, L_2). Therefore (L_1', L_2') is in proportion with (L_1, L_2). □

If the condition in Theorem 9 is met, we choose $(L_1', L_2') = (L_1/GCD(L_1, L_2), L_2/GCD(L_1, L_2))$ as the pair of staggering parameters for the reference pair $\vec{h}_1$ and $\vec{h}_2$.

Table 3 shows examples of staggering parameters for different subscript functions appearing in the dependent reference pair, where the loop index variables are listed in the order from the outermost loop level to the innermost. If we simultaneously consider two reference pairs: $A(I,J)$ with $A(I-3, J-1)$, and $B(I,J)$ with $B(I-1, J-3)$, then the task $T_{i,j}$ will share the same array element $A(i,j)$ with task $T_{i+3,j+1}$ and the same array element $B(i,j)$ with task $T_{i+1,j+3}$. Using Theorem 9, the pairs of staggering parameters (L_1', L_2') for these two pairs of array references are (3,1) and (1,3), respectively. A unified pair of staggering parameters and the folding factor are calculated as $(g, g') = (1, 3)$ and $d = 8$.

Table 3
Examples of different functions in the same dependent reference pair

Loop nest	Dependent reference pair $(\vec{h}_1, \vec{h}_2)$	(β_1, β_2)	(L_1, L_2)	(L'_1, L'_2)
(I,J,K)	$A(I,J), A(I-3,J-1)$	(3,1)	(0,0)	(3,1)
(I,J,K)	$B(I,J), B(I-1,J-3)$	(1,3)	(0,0)	(1,3)
(I,J,K)	$A(J+K,I+J), A(J+K+c,I+J)$	$(d,-d), d \in I$	$(1,-1)$	$(1,-1)$

4. Related work

The work by Peir and Cytron [31], Shang and Fortes [33], and by D'Hollander [9] share the common goal of partitioning an index set into independent execution subsets such that the corresponding loop iterations can execute on different processors without interprocessor communication. Their methods apply to a specific type of loop nest called a *uniform recurrence* or a *uniform dependence algorithm*, in which the loops are perfectly nested, the loop bounds are constant, the *loop-carried dependences* have constant distances, and the array subscripts are of the form $i+c$, where i is a loop index and c an integer constant. Hudak and Abraham [1, 18] develop a static partitioning approach called *adaptive data partitioning* (ADP) to reduce interprocessor communication for iterative data-parallel loops. They also assume perfectly nested loops. The loop body is restricted to update a single data point $A(i,j)$ within a two-dimensional global matrix A. The subscript expressions of right-hand side array references are restricted to be the sum of a parallel loop index and a small constant, while the subscript expressions of left-hand array references are restricted to contain the parallel loop indices only. Tomko and Abraham [35] develop iteration partitioning techniques for data-parallel application programs. They assume that there is only one pair of data access functions and that each loop index variable can appear in only one dimension of each array subscript expression. Agarwal et al. [3] propose a framework for automatically partitioning parallel loops to minimize cache coherence traffic on shared-memory multiprocessors. They restrict their discussion to perfectly nested doall loops. They assume rectangular iteration spaces. Unlike these previous works, our work considers nested loops which are not necessarily perfectly nested. Loop bounds can be any variables, and array subscript expressions are much more general. Many researchers have studied the cache false sharing problem in which cache thrashing occurs when different processors share the same cache line of multiple words, although the processors do not share the same word [10, 17, 19, 36]. Many algorithms have been proposed to reduce false sharing by better memory allocation, better task scheduling, or by program transformations. Our work considers cache thrashing which is due to the *true sharing* of data words.

Kelly and Pugh presented a framework for unifying iteration reordering transformations [25]. Within their framework, transformations are represented as mappings from the original iteration space to a new iteration space. Based on data dependences in a loop nest, a search tree of legal mappings is defined [26]. Guided by performance

estimators, a mapping with minimum cost is then selected for each statement to achieve better performance [24]. Locality is quantified by the number of cache misses based on self reuse. However, in this framework, they do not consider data reuses between consecutive executions of the same parallel loop and hence they do not consider alignment of parallel tasks created by nested loops. In contrast, our work focuses on data reuses between consecutive executions of the same parallel loop and we align parallel tasks to reduce cache thrashing due to true data sharing. Therefore, our work is complementary to their work.

Our work is most closely related to the research done by Fang and Lu [11, 12, 29, 13]. In their work, the iteration space is partitioned into a set of equivalence classes, and each processor uses a formula to determine which iterations belong to the same equivalence class at execution time. Each processor then executes the corresponding iterations so as to reduce or eliminate cache thrashing. These iterations are the solution vectors of a linear integer system. In Fang and Lu's work, these vectors may either be computed at run time or may be precomputed and later retrieved at run time when loop bounds are known before execution. Both approaches require additional execution time when a processor fetches the next iteration. Unlike Fang and Lu's approaches, we solve the thrashing problem at compile time to reduce run-time overhead, while we achieve the same effect of reducing cache thrashing. Our new method restructures the loops at compile time and it is based on a thorough analysis of the relationship between the array element accesses and the loop indices in the nested loop. Previous experimental results conducted on a commercial multiprocessor, namely a Silicon Graphics Challenge Cluster, showed that our technique is quite effective for the reduction and elimination of cache thrashing due to true sharing.

5. Conclusions

This paper presents a method with which the reduced iteration space is rearranged by loop staggering and aligned processor folding. The nested loop (either perfectly nested or imperfectly nested) is restructured to reduce or even eliminate cache thrashing due to true data sharing. This method can be efficiently implemented in any parallel compiler. Although the analysis per se is based on a simple machine model, the resulting code executes correctly on more complex models. Our previous experimental results show that the transformed code can perform quite well on a real machine [21]. How to extend the techniques proposed in this paper to incorporate additional machine parameters is interesting future work.

References

[1] S. Abraham, D. Hudak, Compile-time partitioning of iterative parallel loops to reduce cache coherence traffic, IEEE Trans. on Parallel Distributed Systems 2 (3) (1991) 318–328.

[2] W. Abu-Sufah, D. Kuck, D. Lawrie, On the performance enhancement of paging systems through program analysis and transformations, IEEE Trans. Comput. C-30 (5) (1981) 341–356.

[3] A. Agarwal, D. Kranz, V. Natarajan, Automatic partitioning of parallel loops and data arrays for distributed shared-memory multiprocessors, IEEE Trans. Parallel Distributed Systems 6 (9) (1995) 943–962.
[4] J.R. Allen, K. Kennedy, Automatic loop interchange, Proc. SIGPLAN'84 Symp. on Compiler Construction, Montreal, Canada, June 1984.
[5] J. Baer, W. Wang, Multilevel cache hierarchies: organizations, protocols, and performance, J. Parallel Distributed Comput. 6 (1989) 451–476.
[6] U. Banerjee, Dependence Analysis for Supercomputing, Kluwer, Dordrecht, 1988.
[7] D. Callahan, S. Carr, K. Kennedy, Improving register allocation for subscripted variables, in Proc. ACM SIGPLAN'90 Conf. on Programming Languages Design and Implementation, June 1990.
[8] S. Carr, K. Kennedy, Compiling scientific code for complex memory hierarchies, Proc. Hawaii Internat. Conf. on System Sciences, 1991, pp. 536–544.
[9] E. D'Hollander, Partitioning and labeling of loops by unimodular transformations, IEEE Trans. Parallel Distributed Systems 3(4) (1992) 465–476.
[10] S.J. Eggers, T.E. Jeremiassen, Eliminating false sharing, Proc. 1991 Internat. Conf. on Parallel Processing, August 1991.
[11] J. Fang, M. Lu, A solution of cache ping-pong problem in RISC based parallel processing systems, Proc. 1991 Internat. Conf. on Parallel Processing, August 1991.
[12] Z. Fang, Cache or local memory thrashing and compiler strategy in parallel processing systems, Proc. 1990 Internat. Conf. on Parallel Processing, August 1990, pp. 271–275.
[13] J. Fang, M. Lu, An iteration partition approach for cache or local memory thrashing on parallel processing, IEEE Trans. Comput. C-42 (1993) 529–546.
[14] M. Galles, E. Williams, Performance optimizations, implementation, and verification of the SGI Challenge multiprocessor, Proc. 27th Ann. Hawaii Internat. Conf. on System Sciences, 1994.
[15] K. Gallivan, W. Jalby, D. Gannon, On the problem of optimizing data transfers for complex memory systems, Proc. Supercomputing '88, 1988, pp. 238–253.
[16] D. Gannon, W. Jalby, K. Gallivan, Strategies for cache and local memory management by global program transformation, J. Parallel Distributed Comput. 5 (1988) 587–616.
[17] M. Gupta, D. Padua, Effects of program parallelization and stripmining transformation on cache performance in a multiprocessor, Proc. 1991 Internat. Conf. on Parallel Processing, August 1991.
[18] D. Hudak, S. Abraham, Compiler techniques for data partitioning of sequentially iterated parallel loops, Proc. ACM Internat. Conf. on Supercomputing, 1990, pp. 187–200.
[19] T.E. Jeremiassen, S.J. Eggers, Reducing false sharing on shared memory multiprocessors through compile-time data transformations, Proc. 5th ACM SIGPLAN Symp. on Principals and Practice of Parallel Programming, 1995, pp. 179–188.
[20] G. Jin, F. Chen, The design and the implementation of a knowledge-based parallelizing tool, Proc. 2nd IES Inform. Technol. Conf., July 1991, Singapore.
[21] G. Jin, Z. Li, F. Chen, An efficient solution to the cache thrashing problem caused by true data sharing, IEEE Trans. Comput. 47 (5) (1998) 527–543.
[22] G. Jin, F. Chen. Loop restructuring techniques for the thrashing problem, Proc. 1992 Internat. Conf. on Parallel Architectures and Languages Europe, 1992.
[23] G. Jin, X. Yang, F. Chen, Loop staggering, loop staggering and loop compacting: restructuring techniques for the thrashing problem, Proc. 1991 Internat. Conf. on Parallel Processing, August 1991.
[24] W. Kelly, W. Pugh, Determining schedules based on performance estimation, Technical Report CS-TR-3108, University of Maryland, April 1993.
[25] W. Kelly, W. Pugh, A framework for unifying reordering transformations. Technical Report CS-TR-3193, University of Maryland, April 1993.
[26] W. Kelly, W. Pugh, Finding legal reordering transformations using mapping. Technical Report CS-TR-3297, University of Maryland, June 1994.
[27] D. Kuck. The Structure of Computers and Computations, vol. 1, Wiley, New York, 1978.
[28] D. Kuck, R. Kuhn, D. Padua, B. Leasure, M. Wolfe, Dependence graphs and compiler optimizations, Proc. 8th ACM Symp. on Principle of Programming Languages (POPL), 1981.
[29] M. Lu, J. Fang, A solution of the cache ping-pong problem in multiprocessor systems, J. Parallel Distributed Comput. 16 (1992) 158–171.
[30] I. Nivan et al. An Introduction to the Theory of Numbers, 4th ed., Wiley, New York, 1980.

[31] J. Peir, R. Cytron, Minimum distance: a method for partitioning recurrences for multiprocessors, IEEE Trans. Comput. C-38 (1989) 1203–1211.
[32] C.D. Polychronopoulos, D. Kuck, Guided self-scheduling: a practical scheduling scheme for parallel supercomputers, IEEE Trans. Comput. C-36 (1987) 1425–1439.
[33] W. Shang, J. Fortes, Time optimal linear schedules for algorithms with uniform dependencies, IEEE Trans. Comput. C-40 (1991) 723–742.
[34] Z. Shen, Z. Li, P.-C. Yew, An empirical study of Fortran programs for parallelizing compilers, IEEE Trans. Parallel Distributed Systems 1 (3) (1990) 356–364.
[35] K. Tomko, S. Abraham, Iteration partitioning for resolving stride conflicts on cache-coherent multiprocessors, Proc. 1993 Internat. Conf. on Parallel Processing, August 1993.
[36] J. Torrellas, M.S. Lam, J.L. Hennessy, False sharing and spatial locality in multiprocessor caches, IEEE Trans. Comput. C-43 (1994) 651–663.
[37] M. Wolf, M. Lam, A data locality optimizing algorithm, Proc. ACM SIGPLAN'91 Conf. on Program Language Design and Implementation, June 1991.
[38] M. Wolfe, More iteration space tiling, Proc. Supercomputing'89, 1989.