Avoiding Data Link and Computational Conflicts in Mapping Nested Loop Algorithms to Lower-Dimensional Processor Arrays

Jingling Xue and Patrick Lenders
Department of Mathematics, Statistics and Computing Science
University of New England
Armidale, NSW 2351, Australia

Abstract

This paper describes a unified approach to checking data link and computational conflicts in mapping algorithms to lower-dimensional processor arrays. Based primarily on the notion of Hermite normal form, we propose a range of necessary and sufficient conditions to identify mappings without data link and computational conflicts. These conditions are then used to find optimal time mappings of a transitive closure algorithm to linear processor arrays.

1 Introduction

This paper is concerned with how to avoid data link and computational conflicts in mapping \( n \)-dimensional algorithms to \( r \)-dimensional processor arrays, \( 0 < r < n - 2 \), known as the lower-dimensional processor arrays. A data link conflict (or link conflict for short) occurs if two data elements contend for a given link at the same time step. A computational conflict occurs if two computations of the algorithm are mapped to the same processor at the same time step. The problem of avoiding link and computational conflicts has been studied previously in [1, 2, 5, 6, 10, 11, 13, 17]. Lee and Kedem [5, 6] pioneered the research on checking link conflicts in the design of lower-dimensional arrays. Several conditions on conflict-free mappings were proposed. But, their verification is done by analysis of all computations in the iteration space, which is too expensive for applications of large problem sizes. To alleviate this problem, Shang and Fortes [13] discussed several necessary and sufficient conditions on computational conflict-free mappings and proposed procedures to find optimal time mappings (space mappings are assumed to be given). In their methodology, the issue of avoiding link conflicts was not discussed. Ganapathy and Wah [1, 2] proposed an elegant pruning algorithm for finding optimal lower-dimensional arrays from uniform recurrence equations in the sense of [3], but they did not pay adequate attention to the issue of checking conflicts. In our previous work [17], we studied the problem of mapping three-dimensional algorithms to linear arrays. Several closed-form conditions were derived to identify mappings without link and computational conflicts. For several commonly occurring special types of I/O spaces such as parallelograms and triangles, these conditions are both necessary and sufficient.

In this paper, we continue our previous work [17] by investigating conditions on link and computational conflict-free mappings in the design of lower-dimensional processor arrays. This problem is of practical interest, since many algorithms at bit level (matrix product and LU decomposition) are at least four-dimensional and most existing bit level processor arrays are two-dimensional. The main contributions are as follows. Based on the notion of Hermite normal form, several necessary and sufficient conditions on link conflict-free mappings are developed. In the case where \( n \)-dimensional algorithms are mapped to \((n-2)\)-dimensional arrays, closed-form conditions are also derived. To check computational conflicts, a theorem is first used to test its necessity. If the absence of link conflicts on some link always implies the absence of computational conflicts, then computational conflicts need not be checked explicitly. Otherwise, the problem of checking computational conflicts is solved by reducing it to a problem of checking link conflicts.

Some notations and terminologies used in this paper are introduced here. \( \mathbb{Z} \) and \( \mathbb{Q} \) denote the set of integers and rationals, respectively. The empty set is denoted \( \emptyset \). We do not distinguish whether a vector is a row or column vector and assume that this is deducible from the context. The rank and determinant of a matrix \( A \) are denoted \( \text{rank}(A) \) and \( \text{det}(A) \), respectively. Let \( x \) and \( y \) be two vectors. Then \( x \geq (>) y \) means that every entry of \( x \) is no less than (larger than) the corresponding entry of \( y \). If \( a \) and \( b \) are integers, not both zero, then their greatest common divisor, \( \text{gcd}(a, b) \), is the largest positive integer that divides both \( a \) and \( b \). We define \( \text{gcd}(0, 0) = 1 \).

The rest of the paper is organised as follows. Section 2 presents a few terminologies concerning uniform dependence algorithms. Section 3 introduces the space-time mapping model used for mapping algorithms to processor arrays. Section 4 discusses the problem of checking link conflicts. Section 5 discusses the problem of checking computational conflicts. Section 6 illustrates how to use the proposed conditions to find time-optimal mappings for a transitive closure algorithm. Section 7 contains the concluding remarks.

2 Algorithm Model

Algorithms under consideration in this paper are nested loops with uniform dependence structures [8,
9, 10, 13]. They have the characteristic that their dependencies can be represented by integer vectors. The set of all these index vectors is known as the iteration space and is denoted $\Phi$.

For notational convenience, we assume that each variable is associated with a single dependence vector. The results presented here work for the general case where a variable has several associated dependence vectors, as was explained in [17]. Let $V$ be the set of all variables of the algorithm. For a variable $V \in V$, we use $\varphi_V$ to identify its associated dependence vector. Thus, the set of dependence vectors of the algorithm is $\{ \varphi_V | V \in V \}$. For purposes of checking link conflicts, the concepts of input and output spaces are introduced next. The input (output) space of a variable $V$ is the set of all input (output) data of $V$ and is denoted by $\text{in}_V$ (out$_V$). Both input and output spaces are referred to as the I/O spaces. We shall write $\text{io}_V$ for an I/O space of variable $V$ if we are not concerned whether it is in$_V$ or out$_V$. The notation $\text{IO}$ denotes the set of all non-empty I/O spaces of the algorithm. The structural information of an algorithm can therefore be represented by a pair $(\Phi, \text{IO})$.

**Example 1** Consider the three-dimensional indexed transitive closure algorithm $(\Phi, \text{IO})$ [5]. The iteration space is a cube with lengths of 3: $\Phi = \{(i, j, k) | 0 \leq i, j, k \leq 3\}$. There is only one variable $C$ which has five associated dependence vectors: 

\[
\begin{align*}
(0, 1, 0), & (1, 0, 0), (0, 1, 0), (0, -1, 1), & \text{and} & (1, -1, 1).
\end{align*}
\]

No inputs and outputs are associated with the first four dependence vectors. So the I/O spaces of the algorithm are $\text{IO} = \{(\varphi_C, \text{in}_C), (\varphi_C, \text{out}_C)\}$, where

\[
\varphi_C = (-1, -1, 1), \quad \text{in}_C = \{(i, j, m) | 0 < i, j \leq m\},
\]

\[
\text{out}_C = \{(i, j, m) | 0 < i, j \leq m\}
\]

3 Space-Time Mapping

A space-time mapping $T$, that describes an r-dimensional processor array, $0 < r \leq n - 2$, is a linear index transformation [8]:

\[T : T^n \rightarrow T^{n+1}, \quad T = \left[ \begin{array}{c} \lambda \\ \sigma \end{array} \right] \text{ has full-row rank}\]

where $\lambda \in Z^m$ is the scheduling vector (or time mapping) and $\sigma \in Z^{n \times n}$ is the allocation matrix (or space mapping). The computation indexed by $I$ is executed at time step $\lambda I$ at processor $\sigma I$. The mapping matrix $T$ must have full-row rank. Otherwise, the algorithm is mapped to a processor array of less than r dimension.

Useful functions defined in terms of the space-time mapping are flow and pattern.

- flow : $V \rightarrow Q^r$, $\text{flow}(V) = \sigma \varphi_V / \lambda \varphi_V$ specifies the velocity with which elements of a variable travel. A variable $V$ is moving if $\text{flow}(V) \neq 0$ and is stationary if $\text{flow}(V) = 0$.

- pattern : $V \rightarrow \text{io}_V \rightarrow Q^r$, $\text{pattern}(V)(I) = \sigma I - (\lambda I - \text{no})\text{flow}(V)$ specifies the distribution of inputs/outputs in the processor space at the time step no, where no is the first (last) step number if $\text{io}_V$ is the input space $\text{in}_V$ (output space $\text{out}_V$).

A correct space-time mapping must satisfy the following necessary and sufficient conditions [5, 6]:

- **Precedence Constraint:** $\forall V \in V : \lambda \varphi_V > 0$. A computation is executed only after all the computations on which it depends have been executed.

- **Computation Constraint:** $\forall I, J \in \Phi : I \neq J \Rightarrow \lambda I \neq \lambda J \lor \sigma I \neq \sigma J$. No two computations are executed at the same processor at the same step.

- **Communication Constraint (for Moving Variables):** $\forall V \in V$, flow($V$) $\neq 0$, $I, J \in \text{io}_V : I \neq J \Rightarrow \text{pattern}(V)(I) \neq \text{pattern}(V)(J)$.

The conditions on link and computational conflicts given above [5, 6] are expensive to check. Computationally efficient conditions are proposed next.

4 Checking Data Link Conflicts

If the I/O space $\text{io}_V$ is a convex polyhedron, the communication constraint can be rewritten as follows.

**Theorem 1** (Necessary and sufficient condition 1 for $T \in Z^{(r+1) \times n}$.) Let $\text{io}_V$ be a convex polyhedron: $\text{io}_V = \{I | A I \leq b \}$. $T$ has no link conflicts on $\text{io}_V$ iff there are no integer solutions $(I, J)$ to

\[
\sigma(I - J) = \lambda(I - J)(\lambda \varphi_V / \lambda \varphi_V)
\]

\[
\lambda I \leq b, \quad \lambda J \leq b, \quad I \neq J \quad \text{where} \quad I > J
\]

where $>$ denotes the standard lexicographical "larger than" ordering on vectors.

This theorem sheds certain light on the inherent difficulty of checking link conflicts. The problem demands integer programming technique when $T$ is given in numbers, and is even harder to solve when $T$ is given as a variable. Fortunately, this problem becomes relatively easy if the I/O space is a parallelepiped.

We can represent a p-dimensional parallelepiped I/O space $\text{io}_V$, $0 < p < n$, in the form:

\[
\text{io}_V = \{O + \sum_{i=1}^{p} e_i \xi_i | \forall 0 \leq i \leq p : 0 \leq e_i < B_i\}
\]

where $O \in Z^p$ is a vertex of the parallelepiped, $e_i \in Z^p$ are the edge vectors of the parallelepiped and the integers $B_i \geq 2$ are the integer point counts along $e_i$.

We write $B = (B_1, \ldots, B_p)$. We use $d_i, i = 1, \ldots, p$ to represent the directional distance between two data $V(I + e_i)$ and $V(I)$ as they pass through the array:

\[
d_i = \text{pattern}(V)(I + e_i) - \text{pattern}(I) = \sigma e_i - \lambda e_i \varphi_V / \lambda \varphi_V
\]

Following [7], we call $d_i$ the displacement of $V$ along $e_i$. All displacements are rational vectors in $Q^r$ due to pipelining in systolic processing. The data distribution for $\text{io}_V$ can be precisely described by the following (rational) distribution matrix $P_V$, of size $r \times p$:

\[
P_V = [d_1 \ d_2 \ \cdots \ d_p]
\]

**Theorem 2** (Necessary and sufficient condition 2 for $T \in Z^{(r+1) \times n}$.) Let $\text{io}_V$ be a parallelepiped. $T$ has no link conflicts on $\text{io}_V$ iff there is only zero integer solution to

\[
-P_V X = 0
\]

\[
-B_V < X < B_V
\]
As a consequence of this theorem, all displacements must be non-zero if a mapping is link conflict-free.

**Theorem 3 (Necessary condition 3 for \( T \in \mathbb{Z}^{(r+1) \times n} \))** If \( T \) has no link conflict on \( \text{i/o} \), then \( d_i \neq 0, i = 1, \ldots, p \).

If, in addition, all displacements for variable \( Y \) are linearly independent, then \( T \) is link \( V \) conflict-free.

**Theorem 4 (Sufficient condition 4 for \( T \in \mathbb{Z}^{(r+1) \times n} \))** Let \( \text{i/o} \) be a parallelepiped. \( T \) has no link conflicts on \( \text{i/o} \) if \( \mathcal{P}_V \) has full-column rank.

**Example 2** Consider a three-dimensional I/O space of a four-dimensional algorithm defined as follows:

\[
i_{\text{o}} = \{(i, j, k, l) \mid 0 < i, j, k, l \leq m\}
\]

Where \( d_i = \left\{ \begin{array}{ll} 1 & \text{if} \ (0, i, j, k, l) \in \mathcal{T}_v \\text{and} \ (0, i, j, k, l) \notin \mathcal{T}_v \end{array} \right. \)

\[
\mathcal{T}_v = \{(i, j, k) \mid 0 \leq x_1, x_2, x_3 < m\}
\]

\[
\hat{\mathcal{T}}_v = (0, 0, 0, 1)
\]

Assume that the algorithm is mapped to a two-dimensional array. The three displacements are:

\[d_1 = [\frac{1}{2}, \frac{1}{2}], d_2 = [-\frac{1}{2}, \frac{1}{2}], d_3 = [\frac{1}{2}, -\frac{1}{2}]\]

Then the distribution matrix \( \mathcal{P}_V \) is:

\[
\mathcal{P}_V = \begin{bmatrix}
1 & -m & -1 & -1 \\
-m & 0 & 1 & 0
\end{bmatrix}
\]

So the mapping \( T \) has no link conflicts on \( \text{i/o} \) if there is only zero integer solution to

\[
(1 - m)x_1 - x_2 - x_3 = 0
\]

\[
-x_1 - x_3 = 0
\]

\[
-m < x_1, x_2, x_3 < m
\]

This example demonstrates that, if the distribution matrix \( \mathcal{P}_V \) is a matrix in numbers, we can use integer programming to check the existence of link conflicts. To avoid using integer programming approach, we note that the problem in Theorem 2 is a special form of integer programming. We can then develop some necessary and sufficient conditions to test whether the problem has only zero solution or not.

The basic idea is as follows. All integer vectors that satisfy \( \mathcal{P}_V X = 0 \) are in the nullspace of \( \mathcal{P}_V \). One possible way to avoid link conflicts on \( \text{i/o} \) is to find \( T \) such that all these integer vectors are not contained in the parallelepiped \(-B_V < X < B_V\). To this end, we need a good representation of the integer vectors in the nullspace of \( \mathcal{P}_V \). It is for this purpose that the notion of the Hermite normal form comes into play. The idea of using the Hermite normal form is borrowed from Shang and Forges' work of using the Hermite normal form of \( T \) to check computational conflicts [1]. How the two problems are related is discussed in Section 5. Before further discussions, the following definitions are first introduced, and the concepts about link conflict-free mappings are therefore made precise.

**Definition 1** (Link conflict vectors, and feasible and non-feasible link conflict vectors) An integer vector \( \eta_v = (\eta_1, \ldots, \eta_v) \) is a link \( V \) conflict vector of the distribution matrix \( \mathcal{P}_V \) iff \( \mathcal{P}_V \eta_v = 0 \) and \( \gcd(\eta_1, \ldots, \eta_v) = 1 \). A link \( V \) conflict vector \( \eta_v \) is non-feasible if it satisfies \(-B_V < \eta_v < B_V \) and is feasible otherwise. The mapping matrix \( T \) is link \( V \) conflict-free iff all link \( V \) conflict vectors are feasible, and is link \( V \) conflict-free iff it is link \( V \) conflict-free for every I/O space \( \text{i/o} \).

The (column) Hermite normal form of the distribution matrix \( \mathcal{P}_V \) is introduced next.

**Theorem 5** (Hermite normal form) Let \( \text{rank}(\mathcal{P}_V) = s \), where \( s \leq \min(r, p) \). Scale \( \mathcal{P}_V \) by \( \lambda \) such that \( (\lambda \mathcal{P}_V) \mathcal{P}_V \) is an integer matrix. There exists a unimodular matrix \( U_v = [u_1, \ldots, u_s] \in \mathbb{Z}^{s \times r} \) where \( u_1, u_2, \ldots, u_s \) are its column vectors, such that \( \mathcal{P}_V U_v = [H_v, 0] \) (0 is a zero-entry matrix), where \( H_v \in \mathbb{Z}^{s \times s} \) is a matrix of full-column rank. \( [H_v, 0] \) is called a Hermite normal form of \( \mathcal{P}_V \).

By the same theorem, the same matrix \( \mathcal{P}_V \) may have several different Hermite normal forms, and consequently, several associated unimodular matrices \( U_v \). This is not a problem since we are interested in the last \( p - s \) columns of \( U_v \). All these different unimodular matrices \( U_v \) are equivalent in the sense that their last \( p - s \) columns are different bases of the null space \( \mathcal{P}_V \). A good characterisation of the link \( V \) conflict vectors can be obtained based on any of these bases.

**Theorem 6** All link \( V \) conflict vectors of \( \mathcal{P}_V \) can be expressed as

\[
\eta_v = [u_{s+1}, \ldots, u_p] \begin{bmatrix} \beta_{s+1} & \cdots & 0 \\
0 & \ddots & 0 \\
0 & \cdots & 0
\end{bmatrix}
\]

where \( \beta_i, i = s + 1, \ldots, p \) are arbitrary integers which are not all zero and are relatively prime.

**Example 3** Consider a \( 4 \times 3 \) distribution matrix

\[
\mathcal{P}_V = \begin{bmatrix} 1 & 0 & 1 \\
0 & 1 & 1 \\
0 & 0 & 1 \\
0 & 0 & 0
\end{bmatrix}
\]

where \( \text{rank}(\mathcal{P}_V) = 2 \). Its Hermite normal form is \( \mathcal{P}_V U_v = [H_v, 0] \), where

\[
U_v = \begin{bmatrix} 1 & 0 & -1 \\
0 & 1 & -1 \\
0 & 0 & 1 \\
0 & 0 & 0
\end{bmatrix}, \quad H_v = \begin{bmatrix} 1 & 0 \\
0 & 1 \\
0 & 1 \\
0 & 0
\end{bmatrix}
\]

By Theorem 6, there are only two link \( V \) conflict vector: \( \pm(1, -1, 1) \), which cannot be feasible. So the corresponding mapping is not link \( V \) conflict-free.
This example shows that, in the special case when the nullspace of $P_V$ is one-dimensional, whether $T$ is link $V$ conflict-free or not can be trivially verified. In the general case, several necessary or sufficient conditions on link conflict-free mappings are developed based on the expression in Theorem 6. The key observation is to have matrix $U_V$ satisfy certain conditions. The problem of checking link conflicts formulated in Theorem 2 is similar to that of checking computational conflicts by Shang and Fortes in [13, Theorem 2.2]. Their results are relevant in this context. Theorems 7 – 8 correspond to Shang and Fortes’ Theorems 4.3 – 4.4. Theorem 9 improves Shang and Fortes’ Theorem 4.5 in the sense that the sufficient condition given here is weaker than theirs, and, in addition, includes Shang and Fortes’ Theorem 4.6 as a special case.

**Theorem 7** (Necessary condition for $T \in \mathbb{Z}^{(r+1) \times n}$)

If $T$ is link $V$ conflict-free, then no column vector of $U_V^{-1}$ can have all zero entries in its first $s$ positions.

**Theorem 8** (Necessary condition for $T \in \mathbb{Z}^{(r+1) \times n}$)

If $T$ is link $V$ conflict-free, then $u_{i_1, i} = s + 1, \cdots, p$ are feasible conflict vectors.

**Theorem 9** (Sufficient condition for $T \in \mathbb{Z}^{(r+1) \times n}$)

$T$ is link $V$ conflict-free if the following conditions are satisfied. There are $i_1, \cdots, i_{p-r} \in \{1, \cdots, p\}$ such that

1. $gcd(u_{i_1, i_1} + 1, u_{i_1, i_2} + 2, \cdots, u_{i_1, i_p}) \geq B_{i_1}$
2. $gcd(u_{i_2, i_1} + 1, u_{i_2, i_2} + 2, \cdots, u_{i_2, i_p}) \geq B_{i_2}$
3. $gcd(u_{i_{p-r}, i_1} + 1, u_{i_{p-r}, i_2} + 2, \cdots, u_{i_{p-r}, i_p}) \geq B_{i_{p-r}}$

has full-row rank.

3. There exists $j \in \{i_1, \cdots, i_{p-r} \}$ such that $|u_{j, i_1} + 1, u_{j, i_2} + 2, \cdots, u_{j, i_p}| \geq B_j$ where $x$ is a right null vector of the matrix in Condition (2) and satisfies $gcd(x_1, \cdots, x_p) = 1$.

There is an alternative way to check link conflicts, which might also provide insight why the general problem of checking link conflicts is difficult. Given a distribution matrix $P_V$, the set of all its non-feasible link conflict vectors is contained in a convex polyhedron:

$$C_V = \{ [\beta_{s+1}, \cdots, \beta_p], [u_{s+1}, \cdots, u_p] \}$$

where $gcd(\beta_{s+1}, \cdots, \beta_p) = 1$. By Theorem 2, the mapping $T$ is link $V$ conflict-free iff $C_V = \{0\}$. However, it is generally difficult to check whether or not an arbitrary convex polyhedron contains integer points or not. One possible approach is to construct a minimal bounding parallelepiped by for example the methods in [15, 19], denoted $R_V$, for $C_V$ and check the link conflicts according to the following theorem.

**Theorem 10** (Necessary and sufficient condition for $T \in \mathbb{Z}^{(r+1) \times n}$)

If $T$ is link $V$ conflict-free iff no non-zero integer vectors in $R_V$ are contained in $C_V$.

The conditions provided by the above theorems all involve the matrix $U_V$, which is computed from the distribution matrix $P_V$. When $T$ is known, $P_V$ is a matrix in numbers. There are polynomial algorithms to compute $U_V$ [12]. When $T$ is a variable, $P_V$ is a matrix with symbolic terms. Finding a general approach to computing $U_V$ is difficult. One way out of this dilemma is to find closed-form conditions on conflict-free mappings. This is possible in the case where $n$-dimensional algorithms are mapped to $(n-2)$-dimensional arrays, i.e., for mappings $T \in \mathbb{Z}^{(n-1) \times n}$.

The following lemma describes the ranks of distribution matrices for mappings $T \in \mathbb{Z}^{(n-1) \times n}$

**Lemma 1** Let $io_V$ be a $p$-dimensional parallelepiped, $0 < p < n$. Then $rank(P_{io_V}) \geq p - 1$.

$P_V$ may contain rational entries. Scaling by $\lambda io_V$ yields an integer matrix, $P'_{io_V}$:

$$P'_{io_V} = (\lambda io_V)P_V = [(\lambda io_V)d_1 \cdots (\lambda io_V)d_p] = [d_1 \cdots d_p]$$

**Theorem 11** (Closed-from necessary and sufficient condition for $T \in \mathbb{Z}^{(n-1) \times n}$)

Let $io_V$ be a $p$-dimensional parallelepiped, $0 < p < n$. $T$ is link $V$ conflict-free iff either of the following two conditions is satisfied.

1. $rank(P'_{io_V}) = p$ (i.e., $d_1, \cdots, d_p$ are linearly independent).
2. $rank(P'_{io_V}) = p - 1$. There exists a $(p - 1) \times p$ submatrix $(G, G_g)$ of $P'_{io_V}$ where

$$
G = \begin{bmatrix}
\xi_{i_1, j_1} & \xi_{i_1, j_2} & \cdots & \xi_{i_1, j_{p-1}} \\
\xi_{i_2, j_1} & \xi_{i_2, j_2} & \cdots & \xi_{i_2, j_{p-1}} \\
\vdots & \vdots & \ddots & \vdots \\
\xi_{i_{p-r}, j_1} & \xi_{i_{p-r}, j_2} & \cdots & \xi_{i_{p-r}, j_{p-1}}
\end{bmatrix}
$$

such that $det(G) \neq 0$ and $g \neq 0$ and

$$
\xi = (\xi_1, \cdots, \xi_p) = \begin{bmatrix}
-G'g \\
det(G)
\end{bmatrix}
$$

where $G'$ is the adjugate matrix of $G$ [14], and

$$
\max(\frac{\xi_1}{d_1}, \cdots, \frac{\xi_p}{d_p}) \geq gcd(\xi_1, \cdots, \xi_p)
$$

Frequently in practice, three-dimensional algorithms are mapped to linear processor arrays. The closed-form conditions for one-dimensional I/O spaces are a special case of Theorem 12 given below. The closed-form conditions for two-dimensional I/O spaces are stated below as a corollary of Theorem 11.

**Corollary 1** (Closed-from necessary and sufficient condition for $T \in \mathbb{Z}^{2 \times n}$)

Let $io_V$ be a two-dimensional parallelepiped. $T$ is link $V$ conflict-free iff

$$d_1 = \lambda io_V e_{s_1} - \lambda e_1 \neq 0$$

$$d_2 = \lambda io_V e_{s_2} - \lambda e_2 \neq 0$$

$$\max(\frac{d_1}{e_{s_1}}, \frac{d_2}{e_{s_2}}) \geq gcd(d_1, d_2)$$

570
In this corollary, the first two inequalities are implied by the last inequality and thus be disregarded. This is due to the fact that \( \gcd(0,0) = 1 \) and the assumptions that \( B_1 \geq 2 \) and \( B_2 \geq 2 \).

**Example 4** Consider the transitive closure algorithm again. By [17, Theorem 3], one of the two \( I/O \) spaces is redundant. It suffices to consider \( IO = \{ (\Phi_C, inC) \} \).

By Corollary 1, \( T \) is link \( C \) conflict-free if
\[
\max([\lambda + 2 - 3\sigma_1 - 3(\sigma_1 + \sigma_2 - 3)],
\lambda + 2 - 3\sigma_2 - 3(\sigma_1 + \sigma_2 - 3)),
\lambda + 2 - 3\sigma_2 - 3(\sigma_1 + \sigma_2 - 3)) \geq \gcd([\lambda + 2 - 3\sigma_1 - 3(\sigma_1 + \sigma_2 - 3)],
\lambda + 2 - 3\sigma_2 - 3(\sigma_1 + \sigma_2 - 3))
\]

Generalising Theorem 11 to the case where \( n \)-dimensional algorithms are mapped to \( r \)-dimensional processor arrays, the closed-form conditions for one- and two-dimensional \( I/O \) spaces are available.

**Theorem 12** (Closed-form necessary and sufficient condition 11 for \( T \in \mathbb{Z}^{(r+1) \times n} \)) Let \( IO \) be a one-dimensional parallelepiped. \( T \) is link \( V \) conflict-free iff \( d_{ij} \neq 0 \).

**Theorem 13** (Closed-form necessary and sufficient condition 12 for \( T \in \mathbb{Z}^{(r+1) \times n} \)) Let \( IO \) be a two-dimensional parallelepiped. \( T \) is link \( V \) conflict-free iff either of the two conditions is satisfied.

1. There exist two distinct \( i, j \in \{1, \ldots, r\} \) such that \( d_{ij}, d_{ij} \neq d_{ij} \) (i.e., \( d_{ij} \) and \( d_{ij} \) are linearly independent).
2. \( a \) \( d_{i+1,j+1} = d_{i+1,j+1} \) \( i = 1, \ldots, r \) (i.e., \( d_{ij} \) and \( d_{ij} \) are linearly dependent).

**Checking Computational Conflicts**

A computational conflict occurs iff \( TF \neq TF \) for two different index points \( I, J \in \Phi \). If the iteration space \( \Phi \) is a parallelepiped, the existence of computational conflicts can be checked by the following theorem.

**Theorem 14** (Necessary and sufficient condition 13 for \( T \in \mathbb{Z}^{(r+1) \times n} \)) Let \( \Phi \) be a parallelepiped \( \Phi = \{ I \mid 0 \leq I < C \} \). \( T \) is computational conflict-free iff there is only zero integer solution to \( TX = 0 \) and \( C < X < C \).

The problem of checking computational conflicts as formulated in this theorem is identical to the problem of checking link conflicts as formulated in Theorem 2. In certain cases, computational conflicts need not be checked explicitly.

**Theorem 15** Assume that (1) \( \lambda \Phi \neq 0 \) and (2) that, for every \( K \in \Phi \), there is a \( K' \in \Phi \) \( (K \text{ and } K' \text{ are not necessarily different}) \) such that \( K - K' = \Phi \) and \( \Phi' \) are co-linear. Then the absence of link conflicts on \( \Phi' \) implies the absence of computational conflicts.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Var</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix Product [17]</td>
<td>A, B, C</td>
</tr>
<tr>
<td>LU Decomposition [18]</td>
<td>A</td>
</tr>
<tr>
<td>Dynamic Programming [16]</td>
<td>C</td>
</tr>
<tr>
<td>Gauss-Jordan Elimination [9]</td>
<td>A</td>
</tr>
<tr>
<td>QR Decomposition [4]</td>
<td>nz</td>
</tr>
<tr>
<td>Transitive Closure Algorithm [5]</td>
<td>C</td>
</tr>
</tbody>
</table>

|            |            |
| 1 Calculate a minimal bounding parallelepiped for the iteration space \( \Phi \) using for example the method in [15] and denote it \( R \). |
| 2 Treat the mapping matrix \( T \) as a pseudo distribution matrix, say, \( Pr = T \). |
| 3 Treat the parallelepiped iteration space \( R \) as a pseudo I/O space, \( 0 = R \). |
| 4 Apply Theorems 2 – 11. In the case where \( n \)-dimensional algorithms are mapped to \( (n - 2) \)-dimensional arrays, the closed-form conditions on computational conflict-free mappings are provided by Theorem 11 and for convenience are stated here as a corollary of this theorem. |

**Corollary 2** (Closed-form necessary and sufficient condition 14 for \( T \in \mathbb{Z}^{(n-1) \times n} \)) Let \( \Phi = \{ I \mid 0 \leq I < C \} \). \( T \) is computational conflict-free iff \( \det(G) = 0 \), where \( G \) is the adjugate matrix of \( G \) [14].

|            |            |
| 1 Calculate a minimal bounding parallelepiped for the iteration space \( \Phi \) using for example the method in [15] and denote it \( R \). |
| 2 Treat the mapping matrix \( T \) as a pseudo distribution matrix, say, \( Pr = T \). |
| 3 Treat the parallelepiped iteration space \( R \) as a pseudo I/O space, \( 0 = R \). |
| 4 Apply Theorems 2 – 11. In the case where \( n \)-dimensional algorithms are mapped to \( (n - 2) \)-dimensional arrays, the closed-form conditions on computational conflict-free mappings are provided by Theorem 11 and for convenience are stated here as a corollary of this theorem. |

Most of the results described in this section for checking computational conflicts were originally described in Shang and Fortes’ work [13]. Our contributions are to provide a theorem to avoid checking computational conflicts for a class of algorithms and
to show that the problem of checking computational conflicts can be reduced to one of checking link conflicts. Therefore, in the designing of lower-dimensional processor arrays, more research should be devoted to tackling the problem of checking link conflicts.

6 An Example: Transitive Closure

This section illustrates how to use the conditions described previously to find time-optimal mappings of the transitive closure algorithm to linear arrays.

Example 5 Computational conflicts need not be checked explicitly for the transitive closure algorithm in Example 1. Let us consider the allocation matrix $\sigma = (0, 1, 0)$ used in [5, 13]. The problem of finding optimal scheduling vectors is as follows.

Minimise \( f_{\text{latency}} = (m - 1)(3\lambda_3 - \lambda_1 - \lambda_2) + 1 \)

or \( f_{\text{comp}} = (m - 1)(\lambda_1 + \lambda_2 + \lambda_3) + 1 \)

Subject to \( \lambda_1 > 0, \lambda_2 > 0, \lambda_3 - \lambda_1 - \lambda_2 > 0, \)

\( \max(\lambda_1, \lambda_3 - \lambda_1, \lambda_2 - \lambda_1, \lambda_3) \geq m \gcd(\lambda_1, \lambda_3 - \lambda_1) \)

where the last constraint is from Example 4, which implies rank(\( T \)) = 2; the others are the precedence constraints; \( f_{\text{comp}} \) is the computation time of the algorithm; \( f_{\text{latency}} \) is the latency of the array including \( f_{\text{comp}} \) and the loading time for the inputs and the draining time for the outputs. Here, \( f_{\text{latency}} \) is calculated based on the fact that \( c_{1,1} \) is the first datum to be loaded and \( c_{m,m} \) is the last datum to be drained.

If \( f_{\text{comp}} \) is used as the objective function, the optimal scheduling vector is \((1, 1, m + 1)\). This optimal solution was reported in [1]. The computation time of the algorithm is \( f_{\text{comp}} = m^2 + 2m - 2 \). Since the issue of avoiding link conflicts was not considered in [13], their optimal scheduling vector \((1, 1, m)\) is not link conflict-free. To see this, we note that

\[
T = \begin{bmatrix}
\lambda_1 & 0 & \lambda_2 & \lambda_3 \\
0 & 0 & 1 & 0
\end{bmatrix}
\]

The constraint \( \max(\lambda_1, \lambda_3) \geq m \gcd(\lambda_1, \lambda_3) \) on computational conflict-free derived by Corollary 2 is weaker than the last constraint in the formulation.

If \( f_{\text{latency}} \) is used as the objective function, the optimum \( f_{\text{latency}} = 2m^2 + 2m - 2 \) is attained when the optimal scheduling vector is \((1, m - 1, m + 1)\).

7 Conclusion

This paper focused on the derivation of conditions on conflict-free mapping algorithms to lower-dimensional processor arrays. Our primary contribution includes a range of necessary and sufficient conditions on link conflict-free mappings — some have closed-form expressions and the others were developed based on the Hermite normal form of the distribution matrix. A secondary contribution is the method of checking computational conflicts. A theorem was provided to check whether the absence of link conflicts on a certain link always implies the absence of computational conflicts. If one such a link is found, computational conflicts need not be checked explicitly. Otherwise, the problem of checking computational conflicts was reduced to a problem of checking link conflicts.

Our future work consists of investigating conflict-free conditions for other types of iteration and I/O spaces such as triangles and using the conditions described here to develop efficient procedures for finding optimal mappings.

References


