SYNTHESISING OPTIMAL LINEAR PROCESSOR ARRAYS

Jingling Xue
School of Electrical and Electronic Engineering
Nanyang Technological University
Nanyang Avenue, Singapore 2263

ABSTRACT

Elsewhere in 1, we presented the closed-form conditions for mapping three-dimensional algorithms to linear processor arrays. The mappings that satisfy these conditions are free of computation conflicts and data link collisions. In this paper, we show that optimal mappings in terms of these conditions can be found by a systematic enumeration over a polynomial search space. We illustrate our method with time-optimal linear processor arrays for matrix product.

Keywords: Processor array, space-time mapping, closed-form condition, time-optimal.

1. Introduction

The systolic array is a paradigm that exploits the strength of VLSI and is particularly suitable for implementing many locally recursive algorithms (e.g. DSP and scientific computation). A lot of research has been done on the systematic synthesis of systolic arrays from algorithmic descriptions. However, most existing methods are restricted to the case where \( n \)-dimensional algorithms are mapped to \((n-1)\)-dimensional arrays. This paper addresses the problem of designing optimal linear arrays for three-dimensional algorithms.

There have been several attempts on mapping \( n \)-dimensional algorithms into lower dimensional processor arrays \( 1,2,3,4,5,6,7 \). Lee and Kedem \( 8 \) gave a set of necessary and sufficient conditions for mappings of three-dimensional algorithms to linear processor arrays, without computation conflicts and data link collisions. A computation conflict occurs if two or more computations of an algorithm are mapped to the same processor and the same time step. A communication conflict (or data link collision) occurs if two data elements collide when travelling along a common data link. The mapping conditions presented by Lee and Kedem do not have closed-form expressions; their verification is done by a heuristic analysis of all computations of an algorithm. Shang and Fortes \( 9 \) presented a set of closed-form conditions to avoid computation conflicts. In their methodology, the
issue of avoiding communication conflicts was not addressed. Ganapathy and Wah 2 proposed a parameter-based method in which the operations of the array are captured by a set of parameters, and the constraints are derived to avoid computation and communication conflicts. The issue of deriving closed-form conditions to avoid communication conflicts was not adequately addressed, although it was discussed in the context of Warshall-Floyd path-finding algorithm. In 1, we presented a general method for the derivation of closed-form conditions for mappings of three-dimensional algorithms to linear processor arrays, without computation and communication conflicts. The basic idea is to classify the domains of input/output data of an algorithm according to their shapes and dimensions and then to devise the closed-form conditions accordingly.

In this paper, we use the closed-form conditions presented in 1 to find optimal linear processor arrays for three-dimensional algorithms. We show that optimal mappings in terms of these conditions can be found by a systematic enumeration over a polynomial search space. If space mappings are given, the optimal time mappings may be found by parametric integer programming. We illustrate our method with time-optimal linear processor arrays for matrix product.

2. Algorithm Model

Algorithms under consideration in this paper are loop nests with uniform dependence structures. Such algorithms can also be modeled by uniform recurrence equations 4,8. For the purpose of this paper, it suffices to characterize an algorithm by its index space, dependence vectors, and input and output spaces. Φ is the index space (or iteration space) of the algorithm; every element in Φ is an integer vector of length n (called a point or index vector). Without loss of generality, we assume that each variable of the algorithm is associated with one dependence vector 1. We therefore use Φv to represent the dependence vector with a variable V. We write V for the set of all variables of the algorithm. The set of all dependence vectors of the algorithm can therefore be written as {ΦV | V ∈ V}. The input space of a variable V is the set of vectors I such that every V(I) represents an input datum; it is denoted by inV. Similarly, the output space of a variable V is the set of vectors I such that every V(I) represents an output datum; it is denoted by outV. An input or output space is also referred to as an I/O space.

Matrix product is the multiplication of two square matrices A and B of size m×m to form a matrix C: (∀ i, j, 0 ≤ i, j ≤ m : cij = ( ∑ k : 0 < k < m : aikbkj )).

Matrix product can be specified in the following loops with uniform dependences:

for i from 0 to m do
  for j from 0 to m do
    for k from 0 to m do
      Ai( i, j, k) = Ai( i, j−1, k) 
      Bi( i, j, k) = Bi( i−1, j, k) 
      Ci( i, j, k) = Ci( i, j, k−1) + Ai( i, j−1, k)Bi( i−1, j, k) 

Where Ai(0, k) = aik, Bi(0, j, k) = bijk, Ci( 0, j, 0) = 0 and cij = Ci( i, j, m).

The index space is a cube with sides of length m: Φ = { (i, j, k) | 0 ≤ i, j, k ≤ m }.
There are three variables: \( V = \{A, B, C\} \). The dependence vectors are: \( \vartheta_A = (0, 1, 0) \), \( \vartheta_B = (1, 0, 0) \) and \( \vartheta_C = (0, 0, 1) \), as shown in the corresponding dependence graph.

The input spaces of \( A \), \( B \) and \( C \) are:

- \( \text{in}_A = \{(i, 0, k) | 0 < i, k \leq m\} \)
- \( \text{in}_B = \{(0, j, k) | 0 < j, k \leq m\} \)
- \( \text{in}_C = \{(i, j, 0) | 0 < i, j \leq m\} \)

The output space of \( C \) is:

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

The outputs of \( A \) and \( B \) are of no interest, the corresponding output spaces are omitted. Note that the index space does not intersect any of the three input spaces.

3. Array Model

Figure 1 depicts the linear array model, where the large boxes represent processors, the small boxes delay buffers and the lines connecting channels.

![Figure 1: The linear systolic array model.](image)

The linear systolic array model has the following properties:

- The execution of the array is synchronous, i.e., is governed by a global clock that ticks in unit time.
- Only the two border processors are connected to the host.
- Only neighbouring processors are connected directly with each other.
- A (data) link is a line of channels all of which have the same number of delay buffers. A buffer retains a value for one time step, i.e., one clock tick. (Hence, all the elements move along a common link with a constant velocity.)

4. Space-Time Mapping Model

We use the space-time mapping method to map algorithms to linear arrays. A space-time mapping, \( \Pi \), that describes a linear systolic array is a \( 2 \times n \) integer matrix:

\[
\Pi = \begin{bmatrix}
\lambda \\
\sigma
\end{bmatrix} = \begin{bmatrix}
\lambda_1 & \lambda_2 & \cdots & \lambda_n \\
\sigma_1 & \sigma_2 & \cdots & \sigma_n
\end{bmatrix}
\]

where \( \lambda \in \mathbb{Z} \) denotes the scheduling vector and \( \sigma \in \mathbb{Z} \) the allocation vector. A point \( I \) in the index space is computed at processor \( \sigma I \) at time step \( \lambda I \).

We write \( \text{flow}(V) \) for the velocity with which the elements of a variable \( V \) travel:

\[
\text{flow}(V) = \frac{\sigma \vartheta_V}{\lambda \vartheta_V}
\]

Thus, the number of buffers associated with a channel for variable \( V \) is \( \frac{1}{\text{flow}(V)} - 1 \), where the 1 accounts for the fact that evaluation of a point takes unit time.
variable $V$ is moving if $\text{flow}(V) \neq 0$ and is stationary if $\text{flow}(V) = 0$. We make the convention that a variable $V$ moves to the right if $\text{flow}(V) > 0$ and to the left if $\text{flow}(V) < 0$. Stationary variables do not require data links. Instead, local processor memories are needed to store the data elements of stationary variables.

The coordinates of the two border processors of the array are given by

\[ p_{\text{min}} = (\min I : I \in \Phi : \sigma I) \quad p_{\text{max}} = (\max I : I \in \Phi : \sigma I) \]

The time steps at which data are input and output are specified as follows.

- Function input specifies the input time steps of input data:

\[
\text{input}(V)(I) = \begin{cases} 
\text{if} & \text{flow}(V) > 0 \rightarrow \lambda I - (\sigma I - p_{\text{min}}) / \text{flow}(V) \\
\text{fi} & \\
\text{if} & \text{flow}(V) < 0 \rightarrow \lambda I - (\sigma I - p_{\text{max}}) / \text{flow}(V) \\
\end{cases}
\]

- Function output specifies the output time steps of output data:

\[
\text{output}(V)(I) = \begin{cases} 
\text{if} & \text{flow}(V) > 0 \rightarrow \lambda I - (\sigma I - p_{\text{max}}) / \text{flow}(V) \\
\text{fi} & \\
\text{if} & \text{flow}(V) < 0 \rightarrow \lambda I - (\sigma I - p_{\text{min}}) / \text{flow}(V) \\
\end{cases}
\]

The mapping conditions for the correctness of space-time mappings for linear systolic arrays are given below. The notation $f : D \mapsto R$ denotes that function $f$ is injective from domain $D$ to range $R$. For $x, y \in \mathbb{Z}$, we write $x | y$ iff $x$ divides $y$.

**Definition 1 (Mapping conditions)** A space-time mapping $\Pi$ is correct iff

- \((\forall V : V \in V : \lambda V > 0) \) (Precedence Constraint)
- \(\Pi : \Phi \leftarrow \mathbb{Z}^2 \) (Computation Constraint)
- \((\forall V : V \in V : \text{flow}(V) \neq 0 \Rightarrow \sigma V | \lambda V) \) (Delay Constraint)
- \((\forall V : V \in V : \text{flow}(V) \neq 0 \Rightarrow \text{input}(V) : \text{inv} \leftarrow \mathbb{Z}) \) (Communication Constraint)

The precedence constraint ensures that the dependences prescribed in the algorithm are respected. It remains the same as in the case where $n$-dimensional algorithms are mapped to $(n - 1)$-dimensional arrays. The computation constraint ensures that the mapping is computation conflict-free. The delay constraint states that the number of buffers on any channel for a moving variable is a non-negative integer. It serves to enforce the neighbouring communication required in the linear array model. The delay constraint is not needed if the restriction of neighbouring communication is relaxed. The communication constraint ensures that the mapping is communication conflict-free, i.e., free of data link collisions. More constraints on the mapping are possible for some implementation requirements.

5. Design Method

The mapping conditions of Definition 1 allow us to identify all feasible mappings that are conflict-free. In practice, it is interesting to search for optimal conflict-free mappings with respect to a prescribed objective function. The objective could be the latency of the array, the VLSI area taken to implement the algorithm, or a combination of the two. The search problem is an optimisation problem whose
constraints are the mapping conditions governing the correctness of the mapping. To solve efficiently the optimisation problem for applications of reasonable problem sizes, the following two issues must be adequately addressed.

The first issue is to make the mapping conditions of Definition 1 closed-form. The computation and communication constraints are not in closed-form. Detection of computation and communication conflicts based on them can only be done by a heuristic analysis of all computations of an algorithm and the optimality of the mapping cannot be guaranteed. The derivation of closed-form computation and communication constraints is the subject of Section 6. The second issue is to develop a method that searches the optimal mappings with respect to a certain objective function in a polynomial time complexity. This will be addressed in Section 7.

We shall drop the antecedent flow(V) in the delay and the communication constraints. It is understood that the two constraints are only applicable for moving variables. Similarly, whenever we apply functions input and output to a variable V or write σΦV|λΦV, we mean that V is a moving variable.

6. Closed-Form Computation and Communication Constraints

This section briefly reviews our method presented in \(^1\) for the derivation of closed-form computation and communication constraints. To enforce the computation constraint, we transform it into the communication constraint for a pseudo-variable. To make closed-form the communication constraint, we first classify the I/O spaces of an algorithm according to their shapes and dimensions and then devise closed-form constraints accordingly.

6.1. Transforming the Computation Constraint

Sometimes the absence of the communication conflict on a data link implies the absence of the computation conflict. This occurs if a certain relationship is satisfied between the index space and the I/O spaces of the variable for the link. Let X and Y be two sets and θ be a vector in \(\mathbb{Z}^n\). We write \(X \xrightarrow{θ} Y\) if, when projected along θ onto a fixed hyperplane, the projection of X is contained in the projection of Y.

**Theorem 1** Let \(\lambda \theta_V \neq 0\).

(1) If \(\Phi \xrightarrow{\theta_V} \text{in}_V\), then \((\text{input}(V) : \text{in}_V \xrightarrow{\theta} \mathbb{Z}) \implies (\Pi : \Phi \xrightarrow{\theta} \mathbb{Z}^2)\)

(2) If \(\Phi \xrightarrow{\theta_V} \text{out}_V\), then \((\text{output}(V) : \text{out}_V \xrightarrow{\theta} \mathbb{Z}) \implies (\Pi : \Phi \xrightarrow{\theta} \mathbb{Z}^2)\)

To replace the computation constraint by a communication constraint, we simply add a pseudo-variable, \(\Gamma\), into the variable set \(V\). The dependence vector \(\theta_V\) can be chosen as any integer vector. The I/O spaces of \(\Gamma\) are the projection of the index space along \(\theta_V\) onto a fixed hyperplane. By construction, \(\Phi \xrightarrow{\theta} \text{in}_V\) and \(\Phi \xrightarrow{\theta} \text{out}_V\).

6.2. Making the Communication Constraint Closed-Form

For reasons of symmetry, we restrict ourselves to the input spaces. Let \(\text{gcd}(x, y)\) be the greatest common divisor of two integers \(x\) and \(y\) and set \(\text{gcd}(0, 0) = 0\). Let \(PQ\) be the line connecting \(P\) and \(Q\), inclusive, where \(P\) and \(Q\) are two points in \(\mathbb{Z}^n\).

6.2.1. Parallelograms
Note that when we speak of an index point we mean a point that is contained in the index space. The input space $\text{inv}_V$ of a variable $V$ is a parallelepiped if it satisfies the following properties (Figure 2a):

$$\begin{align*}
\text{in}_V &= \{O + (i-1)e_V^x + (j-1)e_V^y | 0 < i \leq \text{num}_V^x \land 0 < j \leq \text{num}_V^y \land i, j \in \mathbb{Z}\}
\end{align*}$$

A parallelepiped as defined here is not necessarily a convex polyhedron; it is sometimes referred to as a linearly bounded lattice. Consider, for example, the parallelepiped defined as follows (Figure 2b): $O = (2, 2)$, $P = (8, 2)$, $Q = (0, 0)$, $R = (6, 0)$, $e_V^x = (2, 0)$, $e_V^y = (-1, -1)$, $\text{num}_V^x = 4$, and $\text{num}_V^y = 3$. This parallelepiped contains $\text{num}_V^x \times \text{num}_V^y = 12$ index points, as depicted by solid dots. However, the boundaries of the parallelepiped enclose a total of 21 integer points, of which nine, as depicted by circles, are not index points.

Let $\delta_V$ be the time difference between the input of two neighbouring data elements $V(I + e_V^x)$ and $V(I)$ along $e_V^x$. Let $\delta_V'$ be the time difference between the input of two neighbouring data elements $V(I + e_V^y)$ and $V(I)$ along $e_V^y$:

$$\delta_V = \text{input}(V)(I + e_V^x) - \text{input}(V)(I) = \lambda e_V^x - \sigma e_V^y \frac{\lambda \theta_y / \sigma \theta_y}{\text{num}_V^x} \quad \text{where } s \in \{x, y\} \quad (1)$$

**Theorem 2** Let $\text{inv}_V$ be a parallelepiped. The communication constraint for the input space of $V$ is satisfied iff

$$\delta_V \neq 0 \quad \text{and} \quad \max \left( \frac{|\delta_V^x|}{\text{num}_V^x}, \frac{|\delta_V^y|}{\text{num}_V^y} \right) \geq \text{gcd}(\delta_V^x, \delta_V^y)$$

In words, the closed-form communication constraint of Theorem 2 is satisfied if and only if all input data of $V$ are injected at different time steps.

6.2.2. Others

Trapezoidal I/O spaces are dealt with similarly as parallelepiped I/O spaces. Arbitrary two-dimensional I/O spaces are added with points in such a way that they become either parallelograms or trapezoids. Arbitrary three-dimensional I/O spaces are first projected into two-dimensional spaces and then adapted into either parallelograms or trapezoids. We refer to [1] for more details.
7. Optimisation Problem

This section illustrates our method with time-optimal linear arrays for matrix product. We show that optimal mappings can be found efficiently by a systematic enumeration of a polynomial search space.

7.1. Formulating the Optimisation Problem

We only consider the arrays each of which consists of at least one moving variable. For matrix product, we have $\Phi^{V\nu}$ in $V$ for every variable $V (V \in \{A, B, C\})$. Thus, the absence of communication conflicts on data link $V$ implies the absence of computation conflicts (Theorem 1). Therefore, the computation constraint can be disregarded. To avoid the communication conflict on data link $C$, it suffices to consider one of the two I/O spaces of $C$. Intuitively, if all input data are injected at different time steps, all output data will be ejected at different time steps. The converse is true, also. Let us disregard the communication constraint for the output space. See 1 for a discussion of how to eliminate the redundant communication constraints.

The input spaces $in_A$, $in_B$ and $in_C$ are parallelograms: $e_A = (1, 0, 0)$, $e_B = (0, 0, 1)$ and $num_A = num_B = m$; $e_B = (0, 1, 0)$, $e_B = (0, 0, 1)$ and $num_B = num_B = m$; $e_B = (1, 0, 0)$, $e_B = (0, 1, 0)$ and $num_B = num_B = m$. By (1), we obtain

$$
\delta_A^1 = \lambda_1 - \sigma_{12} \frac{\lambda_2}{\lambda_2}, \quad \delta_B^2 = \lambda_2 - \sigma_{23} \frac{\lambda_3}{\lambda_3}, \quad \delta_B^2 = \lambda_3 - \sigma_{31} \frac{\lambda_1}{\lambda_1}, \quad \delta_B^3 = \lambda_1 - \sigma_{12} \frac{\lambda_2}{\lambda_2}.
$$

The closed-form communication constraints for the input spaces $in_A$, $in_B$ and $in_C$ follow from Theorem 2. The problem of finding time-optimal linear arrays for matrix product can be formulated as the following optimisation problem:

Minimise \hspace{1cm} \text{latency}(\Pi)

Subject to \hspace{1cm} \begin{align*}
&\lambda_1 > 0, \lambda_2 > 0, \lambda_3 > 0 & \text{(Precedence Constraint)} \\
&\sigma_{12}, \sigma_{23}, \sigma_{31} & \text{(Delay Constraint)} \\
&\delta_A^1 \neq 0, \max(|\delta_A^1|, |\delta_A^2|) \geq m \gcd(\delta_A^1, \delta_A^2) & \\
&\delta_B^2 \neq 0, \max(|\delta_B^1|, |\delta_B^2|) \geq m \gcd(\delta_B^1, \delta_B^2) & \\
&\delta_B^3 \neq 0, \max(|\delta_B^3|, |\delta_B^2|) \geq m \gcd(\delta_B^3, \delta_B^2) & \\
&\delta_B^3 \neq 0, \max(|\delta_B^3|, |\delta_B^2|) \geq m \gcd(\delta_B^3, \delta_B^2) & \text{(Communication Constraint)}
\end{align*}

where latency(\Pi) is the latency of the array described by \Pi:

\text{latency}(\Pi) = \Pi_{\text{soak}} + \Pi_{\text{comp}} + \Pi_{\text{drain}}

\Pi_{\text{soak}}, \Pi_{\text{comp}} and \Pi_{\text{drain}} represent the soaking time for the input data, the computation time of the algorithm and the draining time for the output data, respectively. By analysing the dependences of the matrix product algorithm, we find that point $(1, 1, 1)$ must be computed at the first step and point $(m, m, m)$ at the last step. We are therefore able to define $\Pi_{\text{soak}}, \Pi_{\text{comp}} and \Pi_{\text{drain}}$ as follows:

$$
\Pi_{\text{soak}} = \lambda(1, 1, 1) - (\min V, I : V \in \{A, B, C\} \land \text{flow}(V) \neq 0 \land I \in \text{in}_V : \text{input}(V)(I))
$$

$$
\Pi_{\text{comp}} = \lambda(m, m, m) - \lambda(1, 1, 1) + 1
$$

$$
\Pi_{\text{drain}} = \begin{cases}
0 & \text{if flow}(C) \neq 0 \\
\lambda(m, m, m) - \lambda(1, 1, 1) + 1 & \text{otherwise}
\end{cases}
$$

\text{flow}(C) = 0

7.2. Solving the Optimisation Problem

The communication constraint is a non-linear expression of the scheduling and allocation vectors. The objective function is also non-linear: \( \Pi_{\text{comp}} \) is a linear function of the scheduling vector; but \( \Pi_{\text{soak}} \) and \( \Pi_{\text{drain}} \) are non-linear functions of the scheduling and the allocation vectors. We consider the four instances of the optimisation problem by the following case analysis.

Correctness-Problem: Given a pair of scheduling and allocation vectors, verify the correctness of the mapping.

\( \lambda \)-Problem: Given an allocation vector, find all scheduling vectors such that the resulting mappings are time-optimal. The allocation vector determines the processor layout of the array. This problem may occur when the optimisation of processor count is the primary objective, whereas the optimisation of latency is the secondary objective.

\( \sigma \)-Problem: Given a scheduling vector, find all allocation vectors such that the resulting mappings are time-optimal. This problem is impractical and is therefore excluded from further consideration.

\( \Pi \)-Problem: Given the directions in which all variables move, i.e., the signs of \( \text{flow}(V) \) for all variables \( V \), find all time-optimal mappings.

7.2.1. Correctness-Problem

It takes constant time to verify the correctness of a given mapping. Consider the following mapping

\[
\Pi = \begin{bmatrix}
m & 1 & 1 \\
0 & 0 & 1
\end{bmatrix}
\]

This mapping is correct. We can easily verify that the precedence and delay constraints are satisfied. The communication constraint for variable \( C \) is satisfied because \( \max(|\delta^C_1|, |\delta^C_2|) = \max(m, 1) \geq m \gcd(m, 1) = \gcd(\delta^C_1, \delta^C_2) \). The communication constraint does not apply for stationary variables \( A \) and \( B \). The array has \( m \) processors and zero buffers and runs in \( m^2 + m - 1 \) time steps.

7.2.2. \( \lambda \)-Problem

For a given allocation vector, the arguments of \( \max \) and \( \gcd \) in the communication constraint become linear expressions of the scheduling vector. So are \( \Pi_{\text{soak}} \) and \( \Pi_{\text{drain}} \). Thus, the objective function is a monotonic function of the scheduling vector, the optimal solution can be found by an enumeration from its smallest permissible value until a feasible solution is found. Since the objective function is monotonic, the first such solution obtained is the optimal solution.

An analytic and symbolic solution to the \( \lambda \)-problem is also possible. Letting \( \gcd(\delta^V_1, \delta^V_2) = 1 \) for all variables \( V \), we obtain a new optimisation problem. We find its optimal solutions by parametric integer programming. These solutions are the optimal solutions to the original \( \lambda \)-problem, if the communication constraint is satisfied. This is so if \( \gcd(\delta^V_1, \delta^V_2) = 1 \) for all variables \( V \). Otherwise, we perturb these solutions to obtain the optimal solutions to the original \( \lambda \)-problem.
**Example 1** Given $\sigma = (1, 1, 0)$, find all scheduling vectors $\lambda$ such that the resulting mappings are time-optimal.

For reasons of symmetry, we assume that $\lambda_1 > \lambda_2$. By a simple algebraic calculation, we obtain the objective function as

$$\text{latency}(\Pi) = (\lambda_1, \lambda_2, \lambda_3)(m, m, m) - \text{input}(B)(0, m, 1) + 1 = (m-1)(2\lambda_1 + \lambda_2 + 1)$$

The optimisation problem becomes

Minimise $2\lambda_1 + \lambda_3$

Subject to $\lambda_1 > 0, \lambda_2 > 0, \lambda_3 > 0$

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

$\lambda_1 > \lambda_2$

Letting $\gcd(\lambda_1 - \lambda_2, \lambda_3) = 1$, we obtain the optimal solution $(2, 1, m)$. Since $(2, 1, m)$ satisfies $\gcd(\lambda_1 - \lambda_2, \lambda_3) = 1$, it is also the optimal solution of the original $\lambda$-problem. Hence, the following mapping

$$\Pi = \begin{bmatrix} 2 & 1 & m \\ 1 & 1 & 0 \end{bmatrix}$$

is time-optimal with respect to the given allocation vector $\sigma = (1, 1, 0)$. This array has $2m-1$ processors and $2m-2$ buffers and runs in $m^2 + 3m - 3$ time steps.

**Example 2** Given $\sigma = (0, 1, -1)$, find all scheduling vectors $\lambda$ such that the resulting mappings are time-optimal.

The objective function follows from a simple algebraic calculation:

$$\text{latency}(\Pi) = \text{output}(C)(m, m, 0) - \min(\text{input}(A)(1, 0, 1), \text{input}(C)(1, 1, 0)) + 1$$

$$= (m-1) \max(\lambda_1 + 2\lambda_2 + 2\lambda_3, \lambda_1 + \lambda_2 + 3\lambda_3) + 1$$

The optimisation problem becomes

Minimise $\max(\lambda_1 + 2\lambda_2 + 2\lambda_3, \lambda_1 + \lambda_2 + 3\lambda_3)$

Subject to $\lambda_1 > 0, \lambda_2 > 0, \lambda_3 > 0$

$$\max(\lambda_1, \lambda_2 + \lambda_3) \geq m \gcd(\lambda_1, \lambda_2 + \lambda_3)$$

Letting $\gcd(\lambda_1, \lambda_2 + \lambda_3) = 1$, we obtain the optimal solution $(m, 1, 1)$ for $m > 2$. If $m$ is odd, $(m, 1, 1)$ satisfies $\gcd(\lambda_1, \lambda_2 + \lambda_3) = 1$. Hence, the following mapping

$$\Pi = \begin{bmatrix} m & 1 & 1 \\ 0 & 1 & -1 \end{bmatrix}$$

is time-optimal for the allocation vector $\sigma = (0, 1, -1)$. This array has $2m-1$ processors and zero buffers and runs in $m^2 + 3m - 3$ time steps. If $m$ is even, the communication constraint is violated. This is because $\max(\lambda_1, \lambda_2 + \lambda_3) = m + 1 \geq m \gcd(\lambda_1, \lambda_2 + \lambda_3) = 2m$ does not hold $(m > 2)$. A small perturbation of $(m, 1, 1)$ gives rise to the desired solution $(m+1, 1, 1)$. Hence, the following mapping

$$\Pi = \begin{bmatrix} m + 1 & 1 & 1 \\ 0 & 1 & -1 \end{bmatrix}$$

is time-optimal for the allocation vector $\sigma = (0, 1, -1)$. This array has $2m-1$ processors and zero buffers and runs in $m^2 + 4m - 4$ time steps.
7.2.3. II-Problem

We solve the II-problem by a systematic enumeration of a polynomial search space. Assume that the index space is a cube with sides of length $L$. The latency of a linear array is $O(L^{n-1})$. Hence, every component of the scheduling vector is bounded by $O(L^{n-3})$. Because of the existence of the delay constraint, every component of the allocation vector is also bounded by $O(L^{n-2})$. Hence, the search space is bounded by $O(L^{3(n-2)})$. Consider the matrix product example: $L = m$ and $n = 3$. Each $\lambda_i$ or $\sigma_i$ ($0 < i \leq 3$) is bounded by $O(m)$. Thus, the search space is bounded by $O(m^6)$.

**Example 3** Find the time-optimal mappings for the class of arrays for which $A$, $B$ and $C$ move to the left.

For reasons of symmetry, we assume that flow($A$) $>$ flow($B$). The time-optimal mappings for several values of $m$ are listed in Table 1.

<table>
<thead>
<tr>
<th>$m$</th>
<th>$\lambda$</th>
<th>$\sigma$</th>
<th>Buffer Count</th>
<th>Processor Count</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>99</td>
<td>(108, 11, 10)</td>
<td>(9, 1, 10)</td>
<td>41160</td>
<td>1961</td>
<td>23521</td>
</tr>
<tr>
<td>100</td>
<td>(120, 2, 9)</td>
<td>(10, 1, 9)</td>
<td>23520</td>
<td>1189</td>
<td>24949</td>
</tr>
<tr>
<td>200</td>
<td>(126, 2, 5)</td>
<td>(5, 1, 6)</td>
<td>46332</td>
<td>4379</td>
<td>91939</td>
</tr>
<tr>
<td>300</td>
<td>(210, 2, 10)</td>
<td>(6, 1, 5)</td>
<td>24948</td>
<td>9569</td>
<td>200929</td>
</tr>
</tbody>
</table>

**Example 4** Find the time-optimal mappings for the class of arrays for which $A$ and $C$ move to the left and $B$ to the right.

The time-optimal mappings for several values of $m$ are listed in Table 2.

<table>
<thead>
<tr>
<th>$m$</th>
<th>$\lambda$</th>
<th>$\sigma$</th>
<th>Buffer Count</th>
<th>Processor Count</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>99</td>
<td>(80, 34, 1)</td>
<td>(-20, 17, 1)</td>
<td>14896</td>
<td>3725</td>
<td>26287</td>
</tr>
<tr>
<td>100</td>
<td>(80, 34, 1)</td>
<td>(-20, 17, 1)</td>
<td>15048</td>
<td>3763</td>
<td>20494</td>
</tr>
<tr>
<td>200</td>
<td>(150, 82, 1)</td>
<td>(-50, 41, 1)</td>
<td>54924</td>
<td>18309</td>
<td>81392</td>
</tr>
<tr>
<td>300</td>
<td>(250, 86, 1)</td>
<td>(-50, 43, 1)</td>
<td>140530</td>
<td>28107</td>
<td>181494</td>
</tr>
</tbody>
</table>

Two observations are drawn based on Tables 1 and 2. First, the optimal mappings do not have closed-form expressions of the problem size. The closed-form mappings reported in [10,11,12] are not optimal. Second, the arrays with counter-flow variables run slightly faster but require far more processors than those with within-flow variables only. If the three factors – latency, processor count and buffer count – are taken into account, the arrays with within-flow variables may be preferable.

In their recent technical report [13], Ganapathy and Wah have extended their work of [2] by providing a more efficient search method. Their method would allow us to solve the II-problem in a more efficient manner.
Acknowledgements

I would like to thank the referees for their careful reading of the manuscript and many helpful comments.

References


