Yale University
Department of Computer Science

Time-optimal Synthesis of Systolic Arrays with Pipelined Cells

Björn Lisper

YALEU/DCS/TR-560
September 1987

This work has been supported in part by the Office of Naval Research under contract N00014-86-K-0564 and in part by The Swedish Board for Technical Development (STU).

Time-optimal Synthesis of Systolic Arrays with Pipelined Cells

Björn Lisper

Department of Computer Science, Yale University
New Haven, CT 06520-2158
lisper@cs.yale.edu

Abstract

We consider synthesis of systolic arrays where the cells in the network themselves are pipelined. Synthesis is done by mapping the set of steps in an algorithm to a *space-time* that consists of events taking place in different cells in the resulting array. Thus every step will be performed at a given time in a given cell. We narrow the scope to the case where each step in the algorithm is assigned an *index vector* and the resulting set of index vectors is linearly transformed into a discrete space-time. In this special case the pipeline constraints are easily expressible as linear constraints on the linear mapping. Some examples are shown where we derive, when possible, optimized versions of well-known systolic arrays where the pipelining capability of the cells is utilized. We also develop a technique to automatically find time-optimal linear mappings when the set of index vectors is a convex polyhedron and the desired constraints on the resulting design are expressible as linear inequalities.

1. Introduction

The concept of systolic arrays has been around for a few years now. Ever since it got well-known, [KLe80], its conceptual simplicity, suitability for direct hardware implementation and the potential large-scale parallelism possible for systolic implementation of certain algorithms have given rise to a considerable interest. In the theoretical field efforts have been made to find methods for correctness verification and synthesis. Especially in the field of formal synthesis progress has been made that seems to be able to have an impact on the real-world design of systolic arrays.

One approach to the synthesis problem that has gained popularity is to consider the algorithm to be implemented as consisting of a number of atomic steps, each step being suitable to perform by a cell in a systolic array in one time unit. The steps are partially dependent on each other; certain steps produce inputs to other steps and must therefore precede these steps. In other words: there is a *data dependence relation* between the steps. Steps with no input/output data relationship, direct or indirect, are independent and can be performed in parallel if suitable.

The steps constituting an algorithm can be defined in different ways. For instance they can be seen as executions of FORTRAN statements, [Mo82], [MiWi84]. Another approach is to define the steps as the different argument configurations possible in a uniform recurrence equation (URE), [KMW69], [Q83], or extensions thereof, [C86], [RPF86], [HDI87]. These argument configurations are tuples of integers. Other authors define the steps as nodes in a graph describing the data dependencies, [RFS82A], [Le83]. The approach taken here was outlined in [Li85] and presented in full detail in [Li87]. There a step in the algorithm is defined as either a block in a partition of a *set of computational events* or as a *multiple-assignment*. Here the latter definition will be used.

In a systolic array every cell is usually considered to be able to perform a computation every time step. While this restriction certainly enhances the clarity of the operation of the array, it can also force the systolic solutions for a given algorithm to be unnecessarily inefficient. The reason is that the kind of operations a systolic cell performs (say, floating point arithmetic) often have a particularly efficient *pipelined* hardware implementation. But since the systolic paradigm does not bother with the inner structure of the cells, an orthodox systolic array must treat the whole time for one computation as one "systolic" time step and thus the next computation can start only when the previous one is totally completed. Therefore the pipelining capability of the cell is not utilized. This problem was recognized in [JKNM86].

Here we will use the mathematical model given in [Li87] to express the basic pipelining constraint that a result of a computation cannot be used until p time steps have passed from the initialization of the computation. (We take the cycle time in the pipeline to be the unit of time and p is the number of pipeline stages for the operation in question.) We will show how this constraint sometimes can be expressed in a particularly simple way. Finally we use the developed formalism to synthesize systolic arrays that utilize the possible pipelining within the cells fully when possible.

The arrays have been synthesized using a technique to find a time-optimal mapping that under certain circumstances is possible to automate fully. This technique is closely related to the one used in SAGA, [HDI87]. Our underlying mathematical model is somewhat different, however, and it enables us to express a greater variety of constraints in the form required by the technique.

2. Preliminaries

In this paper we will rely on the concept of an *algebra*, which is a pair $\langle \mathcal{S}, F \rangle$ where $\mathcal{S}$ is a nonempty set of sets called *sorts* and F is a set of finitary operators. The sorts could be for instance the set of real numbers R and the set of boolean values B. Possible operators would then be the usual arithmetic ones ($+$, $\cdot$, $/$,...), boolean ones ($\wedge$, $\vee$,...) and mixed ones, like the *if*-operator: $B \times R \times R \to R$ that returns its second argument as its value if the first argument is *true* and returns its third argument otherwise.

We will also need *formal expressions.* These are objects built up recursively from variables and constants using the operators of an algebra, that is: all variables and constants are formal expressions. If $\mathbf{p}_1, \ldots, \mathbf{p}_n$ are formal expressions of the "right sorts" and if f is a n-ary operator then $f(\mathbf{p}_1, \ldots, \mathbf{p}_n)$ is a formal expression. Formal expressions are equal if and only if they are built up exactly the same way; thus $x_1 \cdot (x_2 + x_3) \neq x_1 \cdot x_2 + x_1 \cdot x_3$ when regarded as formal expressions, even in an algebra where "$\cdot$" distributes over "$+$", and even $x_1 \cdot (x_2 + x_3) \neq x_1 \cdot (x_2 + x_4)$ since variables with different names are considered not to be equal. *varset*$(\mathbf{p})$ is the set of variables in the expression $\mathbf{p}$.

For every formal expression $\mathbf{p}$ there is a corresponding function $\phi(\mathbf{p})$, a so-called *polynomial.* They are defined recursively in the following way: If $\mathbf{p}$ is a constant, then $\phi(\mathbf{p})$ is the corresponding constant function. If $\mathbf{p}$ is a variable x, then $\phi(\mathbf{p})$ is the *projection* $e^X_x \colon \prod(A_{x'} \mid x' \in X) \to A_x$ that for every tuple $\langle a_{x'} \mid x' \in X \rangle$ returns a_x. If $\mathbf{p}$ is a compound expression $f(\mathbf{p}_1, \ldots, \mathbf{p}_{n(f)})$ then $\phi(\mathbf{p})$ is $f(\phi(\mathbf{p}_1), \ldots, \phi(\mathbf{p}_{n(f)}))$. Note that, to the contrary of formal expressions, $\phi(x_1 \cdot (x_2 + x_3)) = \phi(x_1 \cdot x_2 + x_1 \cdot x_3)$ when "$\cdot$" distributes over "$+$".

A *substitution* σ is a partial function from variables to expressions such that the sort of $\sigma(x)$ and x always is the same. σ *applied to* the expression $\mathbf{p}$, $\mathbf{p}/\sigma$, is the expression obtained when all occurrences of variables x for which $\sigma(x)$ is defined are replaced with $\sigma(x)$. The *domain* of σ, *dom*(σ), is the set of all variables x for which $\sigma(x)$ is defined. The *range* of σ, *range*(σ), is the set of all variables that occurs in any $\sigma(x)$. A *finite substitution* is a substitution defined for only a finite number of variables. A finite substitution σ defined for $x_1, \ldots, x_n$ is often denoted

$$\begin{gathered} x_1 \leftarrow \sigma(x_1) \\ \vdots \\ x_n \leftarrow \sigma(x_n). \end{gathered}$$

For more detail, see [Li87] or a textbook on universal algebra such as [G79].

3. The mathematical model

For more detail and sometimes more general formulations of the concepts defined in this chapter, see [Li87].

3.1 The steps of an algorithm

Definition 3.1: A *multiple-assignment* is a finite substitution.

A multiple-assignment m (or m-assignment, for short) is a partial function and therefore it is a set of pairs. Every pair $\langle x, m(x)\rangle$ can be interpreted as an *assignment* where x is assigned the value of $m(x)$. $m(x)$ is evaluated bottom up, beginning with the constants and the variables, and then proceeding to the top while the operators in the expression are applied to the values of the previously evaluated subexpressions. An example of a m-assignment is

$$\begin{aligned} x_1 &\leftarrow x_0 + y_0 \cdot a_0 \\ y_1 &\leftarrow y_0. \end{aligned}$$

Here the values of x_0 and y_0 are considered to be known in advance. y_1 is assigned the value of y_0. $x_0 + y_0$ is evaluated and the result is assigned to x_1.

An algorithm, in our notion, is simply a countable *set* of m-assignments. If it is to make sense, however, it must have some additional properties:

Definition 3.2: A set of m-assignments M has the *single-assignment property* iff $m \neq m' \implies dom(m) \cap dom(m') = \emptyset$ for all m, m' in M.

This property means that a variable represents a *value*, not a location in a memory that can be reassigned new values. In short, a variable can only be assigned once. Cf. *single-assignment languages*, [A82].

If M has the single-assignment property, then $\bigcup(\, m \mid m \in M\,)$ is a partial function from variables to expressions as well as every m. For any such M $\bigcup(\, m \mid m \in M\,)$ is called the *recursion scheme* of M and is denoted by $\mathbf{M}$. Variables that belong to the domain of a recursion scheme are called *assigned variables*. Those that are not assigned are called *free variables*. The interpretation is that free variables provide inputs to the system from somewhere outside. Assigned variables, on the other hand, are computed according to the rules sketched above. They can be seen as carriers of data between m-assignments and we use them to formally define a *data dependence relation*:

Definition 3.3: Let M be a set of m-assignments that has the single-assignment property. The relation $\prec_M$ on M is defined by

$$m \prec_M m' \iff dom(m) \cap range(m') \neq \emptyset$$

for all m, m' in M.

Of course, if this is to make any sense there must be no cycles in the dependency graph implied by the interpretation of assigned variables as carriers of data. Nor should there be any infinite chains of dependencies without a "beginning":

Definition 3.4: A set of m-assignments M with the single-assignment property is *causal* iff $\prec_M^+$ is well-founded (a strict partial order and no infinite decreasing chains exist).

In the following we will consider only such sets of m-assignments that have the single-assignment property and are causal. We can give very simple *semantics* to such sets of m-assignments by giving semantics to the corresponding recursion schemes. These schemes can be considered as equation systems that defines the assigned variables.

Definition 3.5: Let $\mathbf{M}$ be a recursion scheme. For any x in $dom(\mathbf{M}) \cup range(\mathbf{M})$, the *output expression of x under* $\mathbf{M}$, $\psi_{\mathbf{M}}(x)$, is defined by:

$$\psi_{\mathbf{M}}(x) = \begin{cases} x, & x \text{ is a free variable} \\ \mathbf{M}(x)/\psi_{\mathbf{M}}, & x \text{ is an assigned variable.} \end{cases}$$

The *output function of x under* $\mathbf{M}$ is $\phi(\psi_{\mathbf{M}}(x))$.

The meaning of every variable is thus a function, a polynomial. $\psi_{\mathbf{M}}$ is a partial function from variables to expressions and thus formally a substitution. The recursive definition makes sense since the well-foundedness of $\prec_M$ guarantees that the recursion will eventually come to an end.

Sometimes we will be interested in which *variables* that are dependent of each other, rather than dependencies between m-assignments. Variable y is considered dependent of x if the value of x must be present when computing y, or formally:

Definition 3.6: Let M be a set of m-assignments. For any assigned variable y the m-assignment m_y is defined by $y \in dom(m_y)$. For any variables x, y

$$x \rightarrow_M y \iff x \in varset(m_y(y)).$$

So if the m-assignment

$$\begin{aligned} x_1 &\leftarrow x_0 + y_0 \cdot a_0 \\ y_1 &\leftarrow y_0 \end{aligned}$$

belongs to the set of m-assignments M we find that for instance $y_0 \rightarrow_M x_1$ and $y_0 \rightarrow_M y_1$ but $x_0 \not\rightarrow_M y_1$.

Proposition 3.1: *For any set of m-assignments M and any m, m' in M,*

$$m \prec_M m' \iff \exists x, y[m = m_x \wedge m' = m_y \wedge x \rightarrow_M y].$$

Proof. From the definitions of $\prec_M$ and $\rightarrow_M$,

$$\begin{aligned} m \prec_M m' &\iff dom(m) \cap range(m') \neq \emptyset \\ &\iff \exists x[x \in dom(m) \cap range(m')] \\ &\iff \exists x[x \in dom(m) \wedge \exists y[y \in dom(m') \wedge x \in varset(m'(y))]] \\ &\iff \exists x, y[m = m_x \wedge m' = m_y \wedge x \in varset(m_y(y))] \\ &\iff \exists x, y[m = m_x \wedge m' = m_y \wedge x \rightarrow_M y]. \end{aligned}$$

■

3.2 Space-time and communication orderings

Space-time is a model of the events in a network of cells; every space point is a possible location for a cell and at every time something (for instance a m-assignment) may take place at a cell.

Definition 3.7: Any countable nonempty set R is a *space*. Any set $T = N + t_0 = \{ n + t_0 \mid n \in N \}$, where $t_0 \in Z$ is a *set of times*. $T \times R$ is a *space-time*.

Z is the set of integers and N is the set of the natural numbers. Every time in T represents a discrete time. t_0 is the least time of T. The simple construction above of a space-time as the cartesian product of a set of discrete times and a space is a model of the events in a synchronous, clocked system.

Definition 3.8: The binary relation $\prec_s$ on the space-time S is a *communication ordering* on S if and only if:

co1. For all $\langle t,r\rangle, \langle t',r'\rangle$ in S, $\langle t,r\rangle \prec_s \langle t',r'\rangle \implies t \leq t'$.
co2. $\prec_s^+$ is well-founded.

$\langle S, \prec_s\rangle$ is called a *communication structure.*

Communication orderings describe *communication constraints* on synchronous systems. $s \prec_s s'$ means that communication may take place between the events s and s'. If $s \not\prec_s s'$ it cannot.

co1 is a time causality criterion; communication cannot take place backwards in time. It is however possible for communicating events to take place at the same time. The interpretation of this in a clocked system is that the signal carrying the information does not pass any latch on its way between the points in space where they take place. Thus communication orderings can model systems with *ripple*, where cells may be connected with no intervening latches. co2 prohibits cycles with zero delay. It also prohibits infinite chains of zero-delay connections without any beginning.

An interesting special case is when ripple is not allowed:

Definition 3.9: The relation $\prec$ on the space-time S is *ripple-free* if and only if for all $\langle t,r\rangle$, $\langle t',r'\rangle$ in S $\langle t,r\rangle \prec \langle t',r'\rangle \implies t < t'$.

Theorem 3.1: *All ripple-free relations are communication orderings.*

Proof. co1 follows directly. co2 also follows directly from definition 3.9 since the relation "<" is well-founded on any set of times T (See for instance [MaWa85]). ■

The communication constraints of a systolic system is typically described by a ripple-free communication ordering.

3.3 Connecting algorithm and hardware

The basic idea is to *map* the steps in an algorithm to distinct events in a space-time. Thus every step will have a space-time "tag" telling where and when it is to be performed. This can be done either in accordance with a predefined communication ordering specifying the communication constraints of an existing system, or a communication ordering in space-time is given by the mapping itself together with the communication requirements between the steps of the algorithm. The latter case includes the step of deriving a hardware structure that can support this ordering.

Definition 3.10: Let $\langle M, \prec_M\rangle$ be a m-assignment structure and let $\langle S, \prec_s\rangle$ be a communication structure. The function $F\colon M \to S$ is a *correct mapping* $\langle M, \prec_M\rangle \to \langle S, \prec_s\rangle$ if and only if:

cm1. For all m, m' in M, $m \prec_M m' \implies F(m) \prec_s F(m')$.
cm2. F is 1-1.

A function F to a space-time $T \times R$ will sometimes be seen as two functions; F_t to T and F_r to R.

Definition 3.11: For any m-assignment structure $\langle M, \prec_M\rangle$, space-time S and function $F\colon M \to S$ that is 1-1 the *mapped precedence relation* $\prec_{MF}$ on S is defined by

$$F(m) \prec_{MF} F(m') \iff m \prec_M m'$$

for all m, m' in M.

Thus we can define the concept of *free synthesis*: Given a m-assignment structure $\langle M, \prec_M\rangle$ and a space-time S, find an 1-1 mapping $F\colon M \to S$ such that $\prec_{MF}$ is a communication ordering on S.

Theorem 3.2: *Let $\langle M, \prec_M \rangle$ be a m-assignment structure and let S be a space-time. The function $F: M \to S$ is a correct mapping $\langle M, \prec_M \rangle \to \langle S, \prec_{MF} \rangle$ if and only if:*
1. For all m, m' in M, $m \prec_M m' \implies F_t(m) \leq F_t(m')$.
2. F is 1-1.

Proof.
$\Longrightarrow$: If F is a correct mapping it must be 1-1. Also $\prec_{MF}$ must be a communication ordering. Thus

$$m \prec_M m' \implies F(m) \prec_{MF} F(m') \implies F_t(m) \leq F_t(m').$$

$\Longleftarrow$: F is 1-1, so cm1 holds. Furthermore $m \prec_M m' \implies F(m) \prec_{MF} F(m')$. It remains to show that $\prec_{MF}$ is a communication ordering. co1 follows immediately from 1 above. co2 follows since F is 1-1, $\prec_M$ is well-founded and $F(m) \prec_{MF} F(m') \iff m \prec_M m'$. ■

Theorem 3.2 gives a more familiar condition for correctness of free synthesis. Cf. [Q83], [MiWi84]. Note, however, that we, to the contrary of other authors allow $F_t(m) = F_t(m')$ when $m \prec_M m'$, thus adding the capability to synthesize systems with rippling signals.

An issue of importance is now: How do we organize our hardware so it will support a given communication ordering? One possible way to do it is to provide a line with integer delay δ between the points r, r' in space whenever the communication ordering, for some t, says that communication is possible (or required) between $\langle t, r \rangle$ and $\langle t + \delta, r' \rangle$. One advantage of this organization is that the communication paths will be hardwired into the system, thus eliminating the need for routing. On the other hand the approach might lead to a large number of latches that are rarely used since certain communication paths may be used sparsely. In such a case it may be better to support the communication structure with local memory in the cells that is possible to reuse for different purposes at different times. Clearly the approach is best suited for synthesis of systolic and semisystolic systems.

Definition 3.12: Let R be a space. Then $\Delta \subseteq R \times R \times N$ is a *fixed hardware structure on R*.

Cf. *computation structures*, [CF84]. This model is also close to the delay operator formalism used in for instance [JWCD81], [WD81] or [KLi83], or the labelled graph approach in [LS81].

Definition 3.13: For any communication ordering $\prec_s$ on the space-time $N \times R$, the *fixed hardware projection* $\Delta(\prec_s)$ *of* $\prec_s$ *on* R is given by: for all r, r' in R and δ in N,

$$\exists t \in N[\langle t, r \rangle \prec_s \langle t + \delta, r' \rangle] \iff \langle r, r', \delta \rangle \in \Delta(\prec_s).$$

Note that for any finite communication ordering the fixed hardware projection can be automatically derived. Note also that for any finite set of m-assignments every mapped precedence ordering is finite. We can now formulate the process of free synthesis using the concepts defined so far. Given a set of m-assignments, perform the following steps:
1. Find a suitable space-time that reflects the desired organization of the cells (1-D or 2-D systolic arrays, n-cube networks etc.).
2. Find a 1-1 mapping from m-assignments to space-time such that the mapped precedence ordering is a communication ordering.
3. Derive the fixed hardware projection from the mapped precedence ordering.

3.4 Index vectors and linear mappings

Sets of m-assignments suitable to execute on systolic arrays will typically exhibit a high degree of uniformity. This can be used to simplify the mapping to space-time. The uniformity can often be captured by a suitable, sometimes multidimensional *enumeration* of the m-assignments.

Definition 3.14: For every $n > 0$, Z^n is an *index space.* Points in an index space are called *index vectors.* A subset of an index space is called a *set of index vectors.* A function from a set of m-assignments to an index space that is 1-1 is an *index function.*

Given a set of m-assignments M, an index space Z^n and an index function $G: M \to Z^n$ we can construct a correct mapping from M to any space-time S by finding a function $F: G(M) \to S$ such that $F \circ G$ is a correct mapping. Thus, given an indexing of a set of m-assignments, we can concentrate our efforts on finding a mapping from the set of index vectors.

A space-time of the form $T \times Z^n$ where T is a set of times and $n \geq 0$ is called a *discrete* space-time. Linear systolic arrays are naturally described by a space-time $T \times Z$ and two-dimensional systolic arrays by $T \times Z^2$. Many algorithms in, say, linear algebra have very natural indexings in two or three dimensions. Thus the effort of finding a suitable general mapping can be reduced to the task of finding a suitable mapping between two well-known vector spaces. In particular we can, if we wish, reduce our search to *linear* mappings. This approach is especially interesting for algorithms with a high degree of uniformity and will usually yield systolic array implementations of the algorithms.

A linear mapping $Z^k \to Z^l$ can be represented by a $l \times k$-matrix. Sometimes we will denote the first row, giving the mapping to time, of a linear mapping L by L_t and the rest of L, giving the mapping to space, by L_r.

Definition 3.15: Let $\langle M, \prec_M \rangle$ be a m-assignment structure and let G be an index function. Then $\mathcal{D}_G(\prec_M) = \{\, G(m') - G(m) \mid m \prec_M m' \,\}$ is *the set of data dependence vectors associated with M by G.* For any two assigned variables x and y such that $x \to_M y$, $d^G_{xy} = G(m_y) - G(m_x)$ is *the data dependence vector from x to y.*

When no ambiguity can arise about the index function G we will write $\mathcal{D}(\prec_M)$ for $\mathcal{D}_G(\prec_M)$ and d_{xy} for d^G_{xy}. From proposition 3.1 it is easily seen that $\mathcal{D}(\prec_M) = \bigcup(\, d_{xy} \mid x \to_M y\,)$. Data dependence vectors represent the "abstract" directions of communication in index space. A linear mapping L will transform these directions to physical directions in space-time. This gives an easy check to see if the composed mapping $L \circ G$ is correct or not:

Theorem 3.3: *Let $\langle M, \prec_M \rangle$ be a m-assignment structure and let G be an index function to an index space Z^n. Let T be a set of times. Let L be a linear mapping $Z^n \to Z^k$ for some $k > 0$ such that $L(G(M)) \subseteq T \times Z^{k-1}$ and L restricted to $G(M)$ is 1-1. Then $L \circ G$ is a correct mapping $\langle M, \prec_M \rangle \to \langle T \times Z^{k-1}, \prec_{M\,L\circ G} \rangle$ if and only if for every $d \in \mathcal{D}(\prec_M)$ holds that $L_t d \geq 0$.*

Proof.

$\Longrightarrow$: For every $d \in \mathcal{D}(\prec_M)$ holds that $d = G(m') - G(m)$ for some m, m' in M such that $m \prec_M m'$. Now

$$\begin{aligned} m \prec_M m' &\implies L \circ G(m) \prec_{M\,L\circ G} L \circ G(m') \implies (L \circ G)_t(m) \leq (L \circ G)_t(m') \\ &\implies L_t(G(m)) \leq L_t(G(m')) \implies L_t(G(m') - G(m)) = L_t d \geq 0 \end{aligned}$$

where we have used theorem 3.2.

$\Longleftarrow$: For every m, m' such that $m \prec_M m'$ holds that $G(m') - G(m) \in \mathcal{D}(\prec_M)$. Thus

$$\begin{aligned} m \prec_M m' &\implies L_t(G(m') - G(m)) \geq 0 \implies L_t(G(m)) \leq L_t(G(m')) \\ &\implies (L \circ G)_t(m) \leq (L \circ G)_t(m'). \end{aligned}$$

That $L \circ G$ is correct now follows from theorem 3.2 and the assumption that L restricted to $G(M)$ is 1-1. ■

Thus we can check the time component of every mapped data dependence vector to see if it is greater than zero or not. Note the condition that $L(G(M))$ must be a subset of the space-time $T \times Z^{k-1}$. If this does not hold the composed mapping $L \circ G$ would not map all m-assignments to the space-time! It should be pointed out, however, that if there is a time t such that $L_t i \geq t$ for all i in $G(M)$, then $L \circ G$ will be a correct mapping to $\langle N + t \times Z^{n-1}, \prec_{M\,L\circ G} \rangle$ if the condition on the data dependence vectors holds.

4. How to find optimal solutions

4.1 Separating time and space functions, general case

Consider the problem of finding an *optimal* correct mapping F from a set of m-assignments M to a space-time $T \times R$. The first question one should pose when speaking about optimality is: optimality with respect to *what*? When M is finite then time optimality, to find a solution to

$$\min_{F_t \in M \to N} \left(\max_{m, m' \in M} (F_t(m') - F_t(m)) \right)$$

is clearly of great interest, but spatial issues (locality of communication etc.) should not be neglected. A question of great interest is now: since F can be split into mappings F_t to time and F_r to space, under what circumstances can optimization of F_t with respect to time and of F_r with respect to space be performed independently of each other, and the results be merged to an optimal mapping $\langle F_t, F_r \rangle$ to space-time? In the general case there are clearly some problems:

1. *$\langle F_t, F_r \rangle$ might not be 1-1.* To decide whether this is the case or not information is in general needed from both F_t and F_r.
2. *Locality of communications* is a property dependending on both F_t and F_r. If $m \prec_M m'$, $F_t(m') = F_t(m) + 1$ and if $F_r(m)$, $F_r(m')$ are not "close" non-local communication is needed.

Locality of communications can, however, be assured by F_r if it fulfils a certain condition. Let $\leftrightarrow$ be a *neighborhood relation* on the space. $r \leftrightarrow r'$ means that r and r' are adjacent. Locality of communications now follows if for all m and m' $m \prec_M m' \implies F_r(m) \leftrightarrow F_r(m')$.

4.2 The linear case

Let F be composed by an index function $G: M \to Z^n$ and a linear mapping $L: Z^n \to Z^k$. In this case the requirements of 1 and 2 above can be met as follows:

1. If $k = n$ then $L = \begin{pmatrix} L_t \\ L_r \end{pmatrix}$ will be 1-1 if $\det(L) \neq 0$.
2. On Z^k a suitable neighborhood relation $\leftrightarrow$ is given by $r \leftrightarrow r' \iff \max_{i=1}^{k} |r_i - r'_i| \leq 1$. A linear mapping L_r to Z^k will yield locality of communications with respect to $\leftrightarrow$ if for all data dependence vectors d holds that $\max_{i=1}^{k} |L_r d| \leq 1$.

4.3 Finding optimal time functions with linear index constraints

When an index function G is given the time optimization problem obtains the following form: find a solution to

$$\min_{L_t \in G(M) \to T} \left(\max_{i, i' \in G(M)} (L_t(i') - L_t(i)) \right)$$

For some set of times T. Linear functions $G(M) \to T$, where $G(M) \subseteq Z^n$, can be represented by vectors in Q^n (Q is the set of the rational numbers). Note that there might be functions represented by vectors not in Z^n. Whether this is the case or not is dependent on the shape of $G(M)$. When

$L_t \in Q^n$ we can sometimes restrict the search space of the maximization. Consider the more general maximization problem

$$\max_{i \in I}(L_t i).$$

Assume that there is a subset I_C of I such that every i in I can be written as a convex combination $\alpha i' + (1-\alpha)i''$ of elements i', i'' in I_C, where $0 \leq \alpha \leq 1$. Then the optimal value will be obtained for some element of I_C and the search can be restricted to that set. If I is a convex *polyhedron* the search can be further reduced to the *corners* of I.

In the following we will consider time optimization when the time function L_t is linear and $G(M)$ is given by a system of linear inequalities:

$$i \in G(M) \iff Ai \geq b.$$

That is, $G(M)$ is a convex polyhedron in Z^n. We furthermore assume that all corners given by the inequalities are in $G(M)$ and that $G(M)$ is finite. Quinton [Q83] gave a method to find a feasible linear mapping when $G(M)$ is a possibly infinite polyhedron. The restriction here to finite polyhedra enables us to find a time-optimal linear mapping instead.

We further assume that all other constraints on the mapping regarding causality, injectivity etc. can be expressed as a system

$$L_t A' \geq b'$$

of linear inequalities. These constraints will be referred to as *external constraints.*

We thus have the following problem: given

$$i \in G(M) \iff Ai \geq b, \text{ all corners are in } G(M),$$

find a solution to

$$\min_{L_t}(\max_{i,i' \in G(M)}(L_t i' - L_t i)) = \min_{L_t}(\max_{i,i' \in G(M)}(L_t(i'-i))) = \min_{L_t}(\max_{j \in \{\, i'-i | i,i' \in G(M)\,\}}(L_t j))$$

or, if we define $C = \{\, i' - i \mid i, i' \text{ are corners of } G(M)\,\}$,

$$\min_{L_t}(\max_{j \in C}(L_t j)) \tag{1}$$

when

$$L_t A' \geq b'.$$

Now, for every j in C, let us find the time functions L_t for which $L_t j \geq L_t j'$ for all j' in $C \setminus \{j\}$. They are defined by the system of inequalities

$$L_t(j - j') \geq 0, \quad j' \in C \setminus \{j\}.$$

Furthermore, for each j in C we add the external constraints $L_t A' \geq b'$ on L_t. So we end up with the following system of inequalities for each j:

$$\begin{aligned} L_t(j-j') &\geq 0, \quad j' \in C \setminus \{j\} \\ L_t A' &\geq b' \end{aligned}$$

This is a polyhedron in Q^n. Let us denote it by $P(j)$. $L_t j$ will obtain its smallest value in $P(j)$ at a corner L_t^j. This corner can be found either through linear programming or, if the number of corners is small, by an exhaustive search. *So for every j in C L_t^j minimizes $L_t j$ in the subset*

$P(j)$ where it is greater than every L_tj'. Thus, for every j in C we have a candidate L_t^j to being an optimal solution globally to (1), which can be found by simply comparing all different L_t^j and select one giving the smallest L_t^jj. In summary we have the following steps:

1. Form the external constraints $L_tA' \geq b'$.
2. Find the corners of $G(M)$.
3. Form $C = \{\, i' - i \mid i, i' \text{ are corners of } G(M)\,\}$.
4. For every j in C, form $P(j)$ defined by

$$L_t \in P(j) \iff \begin{array}{c} L_t(j - j') \geq 0, \quad j' \in C \setminus \{j\} \\ L_tA' \geq b' \end{array}$$

5. For every j in C, find the corner L_t^j of $P(j)$ minimizing L_tj.
6. Find a j in C such that $L_t^jj \leq L_t^{j'}j'$ for all j' in C.

There is a small complication in connection with step 5 above. The corner L_t^j of $P(j)$ might not be a valid time function, that is, there might be some i in $G(M)$ such that L_t^ji is not an integer. It is, however, still possible to come up with a candidate of $P(j)$ by searching in the direction of increasing L_tj until we find a hyperplane $L_tj = c$ that contains a valid time function in $P(j)$. We can note that any integer-valued L_t maps index vectors to integer times and thus is a valid time function in the sense above.

How can the restriction of the external constraints to linear inequalities be justified? First we can note that *causality constraints* can be expressed as linear inequalities; if the conditions of theorem 3.3 are met $\prec_{M\,LoG}$ is a communication ordering if for all data depedence vectors d holds that

$$L_td \geq 0.$$

This is a linear inequality of the assumed form. If we want $\prec_{M\,LoG}$ to be ripple-free then it must hold that

$$L_td > 0$$

or equivalently

$$L_td \geq 1$$

since L_td must be an integer. This is also a valid external constraint.

Other frequently encountered constraints are of the form

$$L_tA \neq b$$

where A is an integer $n \times k$-matrix and b is integer-valued. This can be logically converted as follows:

$$L_tA \neq b \iff \bigwedge_{i=1}^{k} L_tA_i \neq b_i \iff \bigwedge_{i=1}^{k}(L_tA_i > b_i \vee L_tA_i < b_i).$$

This conjunction is true if and only if exactly one of the two inequalities is true for each i. (Both cannot be true at once.) Thus we will have 2^k different systems of inequalities, one for each possible choice of inequalities for every i. For every choice s of inequalities and for every j in C we will obtain a subset $P_s(j)$ of Z^n defined by the system of inequalities obtained when combining s with the original inequalities defining $P(j)$. So in this case we will get 2^k candidates of optimal time functions for each j instead of one if all $P_s(j)$ are nonempty.

A constraint of this form assures *injectivity* if we consider linear mappings $L = \begin{pmatrix} L_t \\ L_r \end{pmatrix} : Z^n \rightarrow Z^n$ and L_r is given. L is 1-1 if $\det(L) \neq 0$. But

$$\det(L) = \sum_{i=1}^{n} (L_t)_i (-1)^{i+1} \det(L_{ri}) = \sum_{i=1}^{n} (L_t)_i a_i$$

where L_{ri} is the $n-1 \times n-1$-matrix obtained by removing column i from L_r and $a_i = (-1)^{i+1} \det(L_{ri})$. Thus we obtain a constraint of the form $L_t a \neq 0$.

Broadcast of a variable x occurs if it is used by several m-assignments mapped to the same time by the time function F_t. Define $M(x) = \{ m \mid x \in \mathit{varset}(m) \}$. Broadcast of x is *avoided* iff for all m, m' in $M(x)$

$$F_t(m) \neq F_t(m')$$

or, in the linear case

$$L_t(G(m)) \neq L_t(G(m'))$$

or

$$L_t(i' - i) \neq 0$$

for all i, i' in $G(M(x))$.

A subset X of the free variables might be a *time series.* This means that the variables become available in a certain order. So there is an *enumeration* $(x_i \mid 0 \leq i \leq n)$ of the variables in X such that if $i < j$ then x_i becomes available before x_j. An execution pattern that fulfils the following condition will be suitable for on-line processing of the time series: *if $i < j$ then the first m-assignment using x_i should be executed before the first using x_j.* Let $M_i = M(x_i)$ for $0 \leq i \leq n$. A m in M_i that is executed first under the time mapping L_t satisfies the following condition:

$$\forall m' \in M_i [L_t(G(m') - G(m)) \geq 0].$$

Let us denote this condition $\mathit{first}(m, i, L_t)$. This is a system of linear inequalities. Using this predicate we can now write the predicate on L_t expressing that L_t yields an execution pattern where the first usage of x_i precedes the first usage of x_j: for all i, j such that $i < j$,

$$\exists m \in M_i, m' \in M_j [\mathit{first}(m, i, L_t) \wedge \mathit{first}(m', j, L_t) \wedge L_t(G(m') - G(m)) > 0].$$

This is a split into different cases: for every choice of m from M_i and m' from M_j there is a system of linear inequalities expressing that m, m' are executed first in M_i, M_j respectively and that m is executed before m', defining a convex polyhedron $P(i, j, m, m')$ where L_t resides if it satisfies the system. For every i, j there will thus be a number of such polyhedra, and the union $P(i, j) = \bigcup(P(i, j, m, m') \mid m \in M_i, m' \in M_j)$ gives the subspace where L_t must be if the first m-assignment in M_i is executed before the first of M_j. Finally this must hold for all i, j such that $i < j$, thus $\bigcap(P(i, j) \mid i < j)$ gives the subspace where L_t must be. Since the intersection can be distributed into every $P(i, j)$ this is a union of convex polyhedra, each being defined by a system of inequalities.

The number of inequalities can be greatly reduced if every point in $G(M_i)$ can be written as a convex combination of points from a set $C_i \subseteq G(M_i)$. Then the first m-assignment in M_i must be indexed in C_i and it is thus sufficient to form the differences between these index vectors. If every M_i contains only *one* m-assignment m_i, then the condition above reduces further to: for all i, j such that $i < j$

$$L_t(G(m_j) - G(m_i)) > 0$$

which is a single system of inequalities.

5. Expressing pipelining as time constraints

5.1 The general case

The basic property of a pipeline we will use as a starting point is the following: *in a p-stage pipeline the results are available p time units after the inputs have been provided.* At every time we can initialize a computation at a pipelined cell but we must wait p time steps until the result is ready.

A cell in a systolic array will be considered as possibly being composed out of more than one pipelined component. An intermediate result, internal to the cell, may for instance be computed in one pipeline and then fed in together with another input to a second pipeline. This calls for the possibility in the mathematical model to allow inputs to a m-assignment m to arrive later than the scheduled time for m. If all inputs were required to be present at this time unnecessary delays could be introduced.

A m-assignment being mapped to a space-time point $\langle t, r\rangle$ has the following interpretation if the cell at r is pipelined: at time t the computation is initialized. Some inputs may have to be present at r at this time, but possibly not all. As the computation goes on the remaining inputs must be provided at certain times relative to the time of initialization. Outputs will also become available at possibly different times, since the total length of the pipelines may be different for different outputs. Directly when an output is ready it is available to use for other m-assignments. We choose the integer time step to be equivalent to the clock cycle in the pipeline and not, as usual, to the time required to complete a computation.

Definition 5.1: Let M be a set of m-assignments. A *pipeline description of M* is a pair $\langle p, i\rangle$ where p is a function $dom(\mathrm{M}) \rightarrow N$, i is a function $range(\mathrm{M}) \times dom(\mathrm{M}) \rightarrow N$ and for all x, y such that $y \rightarrow_M x$ holds that $i(y,x) \leq p(x)$.

For every assigned variable x $p(x)$ gives the total time necessary to compute it from the inputs of the m-assignment computing x, and $i(y,x)$ gives the time when y has to be provided to the computation of x. Note the special case when $i(y,x) = 0$ for all y and either $p(x) = 1$ (x can be computed in one time step, no pipelining) or $p(x) = 0$ (x is instantly computed and the result is rippling through to other computations without passing any latches).

Example 5.1: Consider the m-assignment m:

$$\begin{aligned} y' &\leftarrow y \\ a' &\leftarrow ay \\ x' &\leftarrow x + (a + ay). \end{aligned}$$

Figure 5.1 shows a cell that can execute m. It is built out of one multiplier and two adders. $p_\times$ is the number of pipeline stages for the multiplier and p_+ the number of stages for each of the adders. If we assume that m is to be executed by such a cell we obtain the following pipeline description:

$$\begin{array}{ccc} i(y,y') = 0 & i(y,a') = 0 & i(y,x') = 0 \\ & i(a,a') = 0 & i(a,x') = 0 \\ & & i(x,x') = p_\times + p_+ \\ p(y') = 0 & p(a') = p_\times & p(x') = p_\times + 2p_+ \end{array}$$

■

Definition 5.2: Let M be a set of m-assignments, let S be a space-time and let F be a function $M \rightarrow S$. Let $\langle p, i\rangle$ be a pipeline description of M. F is *correct with respect to* $\langle p, i\rangle$ iff it is 1-1 and for all assigned variables x and y

$$x \rightarrow_M y \implies F_t(m_x) + p(x) \leq F_t(m_y) + i(x,y).$$

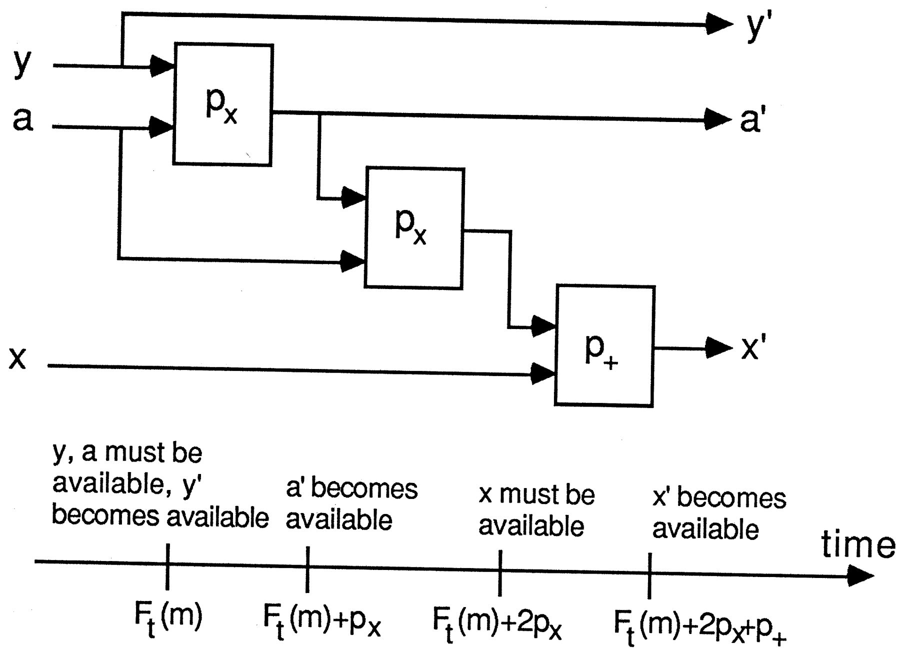

Figure 5.1: Cell executing m

5.2 Pipeline constraints and linear mappings

In the special case when the m-assignments are labelled with index vectors and the index vectors are mapped linearly to a discrete space-time the correctness criterion with respect to pipelining functions has a particularly neat formulation:

Theorem 5.1: *Let M be a set of m-assignments and let G be an index function to an index space Z^n. Let T be a set of times. Let L be a linear mapping $Z^n \to Z^k$ for some $k > 0$ such that $L(G(M)) \subseteq T \times Z^{k-1}$. Let $\langle p, i\rangle$ be a pipeline description of M. F is correct with respect to $\langle p, i\rangle$ iff for all assigned variables x and y such that $x \to_M y$ holds that $L_t d_{xy} \geq p(x) - i(x,y)$.*

Proof.

$\Longrightarrow$: We prove the negated implication going in the other direction. Assume that there are some x and some y such that $L_t d_{xy} < p(x) - i(x,y)$. Then $d_{xy} = G(m_y) - G(m_x)$, and

$$\begin{aligned} L_t(G(m_y) - G(m_x)) &= L_t(G(m_y)) - L_t(G(m_x)) \\ &= (L \circ G)_t(m_y) - (L \circ G)_t(m_x) < p(x) - i(x,y), \end{aligned}$$

that is:

$$(L \circ G)_t(m_x) + p(x) > (L \circ G)_t(m_y) + i(x,y).$$

$\Longleftarrow$: Assume that for all x and y such that $x \to_M y$ holds that $L_t d_{xy} \geq p(x) - i(x,y)$. Then $d_{xy} = G(m_y) - G(m_x)$, and

$$p(x) - i(x,y) \leq L_t(G(m_y) - G(m_x)) = L_t(G(m_y)) - L_t(G(m_x))$$

$$= (L \circ G)_t(m_y) - (L \circ G)_t(m_x)$$

that is:

$$(L \circ G)_t(m_x) + p(x) \leq (L \circ G)_t(m_y) + i(x, y).$$

■

Cf. [QG85], where a similar condition is formulated. Note that the pipeline constraint in theorem 5.1 is a linear inequality and therefore qualifies as an external constraint. Thus the method of section 4.3 can be used to find time-optimal systolic arrays with pipelined cells.

6. Applications

In this chapter we will show some simple examples of synthesis of systolic arrays with pipelined cells. For a given indexing and for given pipeline constraints we will derive optimal solutions.

6.1 FIR filtering

FIR filtering can be seen as a special case of matrix-vector multiplication $y = Ax$ where A is banded and all elements below the main diagonal are zero. We will treat this more general operation instead, and the following set of m-assignments describes an algorithm to perform it:

$1 \leq i \leq n$:

$$y_{ii} \leftarrow 0 + a_{ii}x_i \qquad (\,i \quad i\,)$$

$i < j < i + b$:

$$y_{ij} \leftarrow y_{ij-1} + a_{ij}x_j \qquad (\,i \quad j\,)$$

n values are filtered. The bandwidth of the system matrix is b. The input values x_j and the matrix elements a_{ij} are free variables. In the case of FIR filtering $a_{ij} = w_{j-i}$ for all i, j of interest. The outputs are y_{ii+b-1}, $1 \leq i \leq n$. We have also provided an index function, the index vector of each m-assignment is shown to the right of it. The set of index vectors is a polyhedron defined by the following inequalities:

$$\begin{gathered} 1 \leq i \\ i \leq n \\ i \leq j \\ j < i + b \end{gathered}$$

or, rewritten to our standard form:

$$\begin{pmatrix} 1 & 0 \\ -1 & 0 \\ -1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} \geq \begin{pmatrix} 1 \\ n \\ 0 \\ 1-b \end{pmatrix}$$

This polyhedron has four corners: $(\,1 \quad 1\,)$, $(\,1 \quad b\,)$, $(\,n \quad n\,)$ and $(\,n \quad n+b-1\,)$, and we define C as the set of all differences between these corners.

The assigned variables above are the different y_{ij}. They all yield the same data depedence vector $d = (\,0 \quad 1\,)$. Note that all x_i are free variables whenever they are used. The interpretation is that they are provided from outside the system every time they are needed.

Let us now find a time-optimal linear mapping L to some $T \times Z$, that is: we consider linear systolic array implementations. Before optimizing the time function L_t we must select a suitable space mapping L_r. Assume that the number of input values n is much greater than the matrix bandwidth b. If we want to achieve a high *cell utilization* this implies that $L_r = (\, l_{21} \quad l_{22} \,)$ should be chosen so that each diagonal of A always is mapped to the same space point. This gives the condition $l_{21} + l_{22} = 0$, [Li87]. For *nearest neighbor communication* L_r should be chosen so that $|L_r d| \leq 1$ which, if L is to be nonsingular, implies $l_{22} = \pm 1$, $l_{21} = \mp 1$. If (arbitrarily) the flow of y-operands is chosen to go to increasing points in Z we end up with $L_r = (\, -1 \quad 1 \,)$.

Let us now consider the external constraints under which the optimization will be performed. When $b > 1$ the polyhedron of index vectors has such shape that only integer-valued time functions are valid. This can be used to simplify the constraints.

1. *Injectivity of L* follows if $\det(L) \neq 0$. With $L_t = (\, l_{11} \quad l_{12} \,)$ and L_r as above $\det(L) = l_{11} + l_{12}$ which implies
$$l_{11} + l_{12} \geq 1 \qquad \text{or} \qquad l_{11} + l_{12} \leq -1.$$

2. *Pipeline constraints*: Here we must make some assumptions about the internal structure of the cells that will execute the m-assignments in our algorithm. We assume that the m-assignments of form $y' \leftarrow y + ax$ are executed in the following way: first a and x are multiplied in a unit with $p_\times$ pipeline stages. Then the result of this is added to y in a unit with p_+ stages. We end up with the following pipeline description:
$$\begin{array}{ll} i(x, y') = 0 & \\ i(a, y') = 0 & \qquad p(y') = p_\times + p_+ \\ i(y, y') = p_\times & \end{array}$$

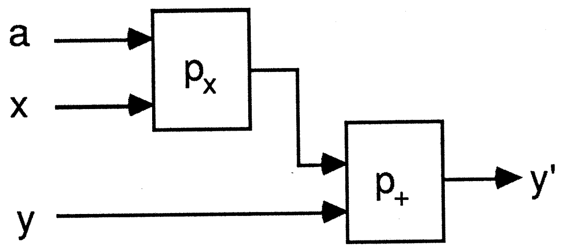

Figure 6.1: Inner structure of cell

This translates to the following pipeline constraint for the data depedence vector $d = (\, 0 \quad 1 \,)$:
$$L_t d \geq p_\times + p_+ - p_\times, \text{ or } l_{12} \geq p_+$$

3. *Broadcast avoidance*: Consider again the free variables x_j. Each x_j is used by the m-assignments indexed with $(\, i \quad j \,)$. To avoid broadcast of x_j these index vectors must be mapped to different times. This is accomplished if for all $i \neq i'$
$$L_t \begin{pmatrix} i \\ j \end{pmatrix} \neq L_t \begin{pmatrix} i' \\ j \end{pmatrix}$$
or
$$L_t \begin{pmatrix} i - i' \\ 0 \end{pmatrix} \neq 0$$
or

$$l_{11} \neq 0$$

or

$$l_{11} \geq 1 \qquad \text{or} \qquad l_{11} \leq -1.$$

Summing up, we have the following external constraints:

$$l_{11} + l_{12} \geq 1 \qquad \text{or} \qquad l_{11} + l_{12} \leq -1$$
$$l_{12} \geq p_+$$
$$l_{11} \geq 1 qquad \text{or} \qquad l_{11} \leq -1.$$

Next step is to find which elements j of C that will yield maximal $L_t j$ and where. It turns out that there are only two possible candidates in the area defined by the external constraints. These are $(\, n-1 \quad n+b-2 \,)$ which is maximal when $l_{11} \geq -l_{12}$ and $(\, 1-n \quad b-n \,)$ which is maximal when $l_{11} \leq -l_{12}$.

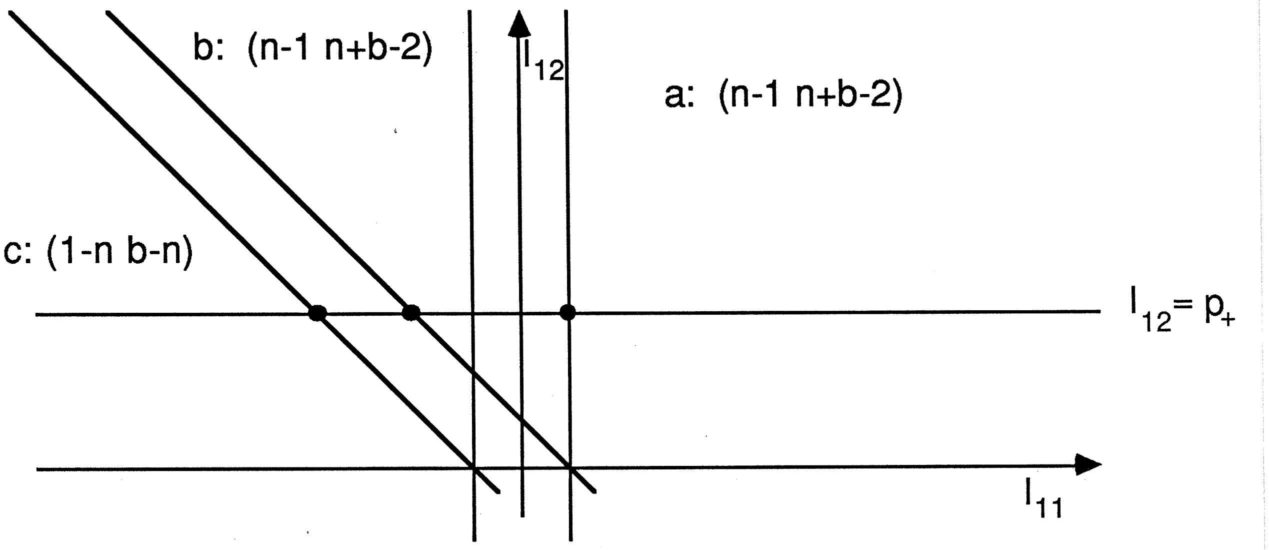

Figure 6.2: External constraints, convex polyhedra.

Together with the external constraints these two areas form three convex polyhedra a, b and c as shown in figure 6.2. Dependent on the value of p_+ the shape of b will be slightly different leading to different optimal corners:

polyhedron	element j of C	optimal corner L_t^j	value of $L_t^j j$
a	$(\, n-1 \quad n+b-2 \,)$	$(\, 1 \quad p_+ \,)$	$(p_+ + 1)n + p_+(b-2) - 1$
$b, \quad p_+ < 2$	$(\, n-1 \quad n+b-2 \,)$	$(\, -1 \quad 2 \,)$	$n + 2b - 1$
$b, \quad p_+ \geq 2$	$(\, n-1 \quad n+b-2 \,)$	$(\, 1-p_+ \quad p_+ \,)$	$n - 1 + p_+(b-1)$
c	$(\, 1-n \quad b-n \,)$	$(\, -p_+ - 1 \quad p_+ \,)$	$n - 1 + p_+(b-1)$

A winner in all cases is $L_t = (\, -p_+ - 1 \quad p_+ \,)$, with execution time $n - 1 + p_+(b-1)$. With $L_r = (\, -1 \quad 1 \,)$ the following time-optimal mapping L from index space to space-time is obtained:

$$\begin{pmatrix} t \\ r \end{pmatrix} = \begin{pmatrix} -p_+ - 1 & p_+ \\ -1 & 1 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$

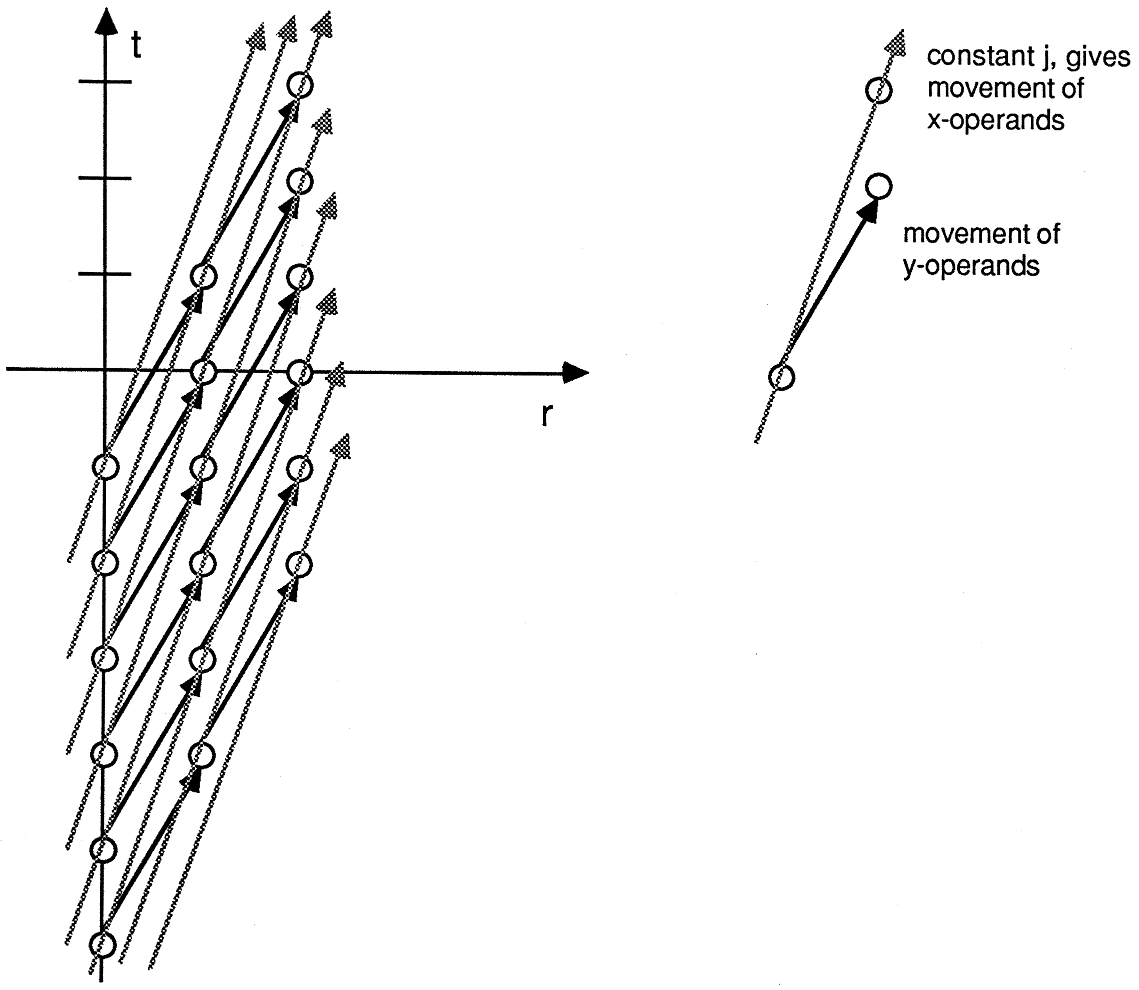

Figure 6.3: Space-time diagram of L, $p_+ = 2$.

Since we selected the external constraints to exclude broadcast of the variables x_i each x_i can be input once and reused instead of being input every time it is needed. This behavior can be derived formally, [Li87].

An array that supports the schedule given by L is shown in figure 6.4. Cf. Quinton and Gachet, [QG85], where similar arrays for FIR filtering are derived. Note, however, that the arrays of Quinton and Gachet do not implement the same recurrence. They implement a similar set of m-assignments where the summation is done *backwards*. If we had started with this recurrence rather than the forward summation recurrence we would have obtained this family of arrays instead.

If we set $p_\times = 0$ and $p_+ = 1$ we will obtain systolic solutions where the cells are not pipelined. The optimal corner $(1 \quad 1)$ of a gives the classical solution of Kung and Leiserson, [KLe80]. This solution has asymptotic cell utilization 1/2 and no extra delays between the cells, which is reflected in its execution time being $2n + b - 3$. The optimal corner $(-1 \quad 2)$ of b gives a solution with full cell utilization after startup but where every intermediate result y_{ij} is delayed an extra time step. This is in accordance with its execution time $n + 2b - 1$. The globally optimal corner $(-2 \quad 1)$ of d, finally, gives an array with full cell utilization where the x-operands are delayed an extra time step. Since there are no extra delays for the y-operands this solution gives the superior execution time $n + b - 2$. As a matter of fact this solution is implemented by exactly the same array as

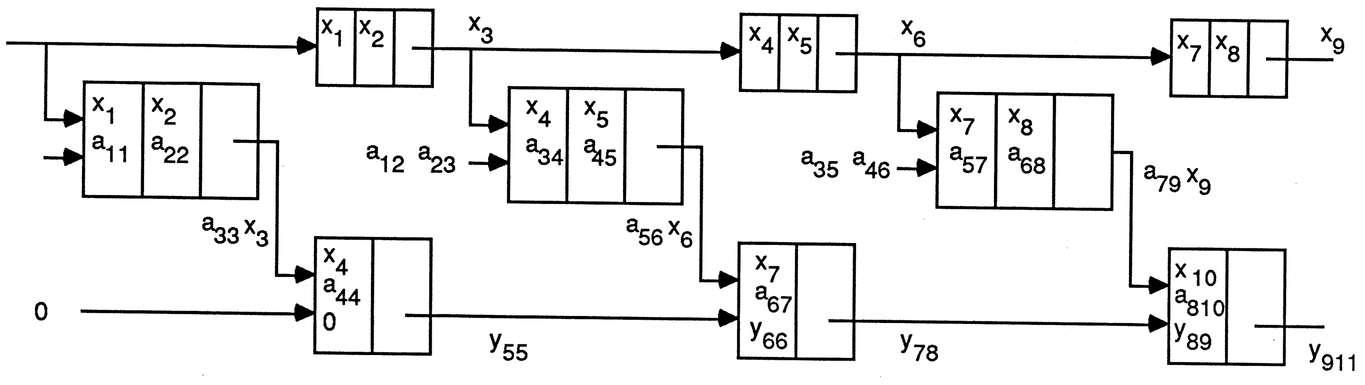

Figure 6.4: Array supporting the schedule defined by L. $p_\times = 3$, $p_+ = 2$.

the one proposed by Kung, [Ku82]. But the solution of Kung implements the same backwards recurrence as the arrays of Quinton and Gachet and can in fact be seen as a special case of these arrays. The difference to our solution lies entirely in the input pattern of the x-operands and the matrix elements a_{ij}. In our solution the x-operands are input "backwards", with x_j input *before* x_i if $j > i$. In Kung's solution it is the other way around.

This input pattern of the x-operands unfortunately makes our time-optimal solution not suitable when the x-operands form a time series. In order to find an optimal solution suitable for on-line execution we form the linear constraints expressing this. Every x_j is used by the m-assignments indexed by $\{(i \quad j) \mid l_j \leq i \leq u_j\}$, where $l_j = \max(1, j-b+1)$ and $u_j = \min(j, n)$. Every such index vector can be written as a convex combination of $(l_j \quad j)$ and $(u_j \quad j)$. For every j these two are possible candidates for being first. $(l_j \quad j)$ is first if

$$L_t(\begin{pmatrix} l_j \\ j \end{pmatrix} - \begin{pmatrix} u_j \\ j \end{pmatrix}) \geq 0, \text{ or } l_{11}(\min(n,j) - \max(1, j-b+1)) \geq 0, \text{ or } l_{11} \geq 0.$$

Similarly $(u_j \quad j)$ is first if $l_{11} \leq 0$. Thus there are four possible pairs of first vectors to compare for every $j < k$, giving four polyhedra for L_t:

$$\begin{pmatrix} l_k \\ k \end{pmatrix}, \begin{pmatrix} l_j \\ j \end{pmatrix}: \qquad l_{11} \geq 0 \wedge l_{11} \geq 0 \wedge L_t(\begin{pmatrix} l_k \\ k \end{pmatrix} - \begin{pmatrix} l_j \\ j \end{pmatrix}) > 0$$

$$\begin{pmatrix} u_k \\ k \end{pmatrix}, \begin{pmatrix} l_j \\ j \end{pmatrix}: \qquad l_{11} \geq 0 \wedge l_{11} \leq 0 \wedge L_t(\begin{pmatrix} u_k \\ k \end{pmatrix} - \begin{pmatrix} l_j \\ j \end{pmatrix}) > 0$$

$$\begin{pmatrix} l_k \\ k \end{pmatrix}, \begin{pmatrix} u_j \\ j \end{pmatrix}: \qquad l_{11} \leq 0 \wedge l_{11} \geq 0 \wedge L_t(\begin{pmatrix} l_k \\ k \end{pmatrix} - \begin{pmatrix} u_j \\ j \end{pmatrix}) > 0$$

$$\begin{pmatrix} u_k \\ k \end{pmatrix}, \begin{pmatrix} u_j \\ j \end{pmatrix}: \qquad l_{11} \leq 0 \wedge l_{11} \leq 0 \wedge L_t(\begin{pmatrix} u_k \\ k \end{pmatrix} - \begin{pmatrix} u_j \\ j \end{pmatrix}) > 0$$

The two middle polyhedra can be discarded since they do not comply with the no-broadcast condition $l_{11} \neq 0$. Left are the two polyhedra $P_l(j,k)$ and $P_u(j,k)$ defined by

$$P_l(j,k): \begin{cases} l_{11} \geq 0 \\ l_{12} > 0 \\ l_{11}(\max(1, k-b+1) - \max(1, j-b+1)) + l_{12}(k-j) > 0 \end{cases}$$

$$P_u(j,k): \begin{cases} l_{11} \leq 0 \\ l_{12} > 0 \\ l_{11}(\min(k,n) - \min(j,n)) + l_{12}(k-j) > 0 \end{cases}$$

Since either of these conditions is to hold for all $k > j$ the allowed subspace for L_t is

$$\bigcap_{j<k}(P_l(j,k) \cup P_u(j,k)) = \bigcap_{j<k} P_l(j,k) \cup \bigcap_{j<k} P_u(j,k) = P_l \cup P_u$$

since $P_l(j,k) \cup P_u(i,i') = \emptyset$ when $l_{11} \neq 0$. P_l is given by

$$\begin{cases} l_{11} \geq 0 \\ l_{12} > 0 \\ l_{11}(k-b) + l_{12}(k-j) > 0 & , j \leq b < k \\ l_{11}(k-j) + l_{12}(k-j) > 0 & , b < j < k \end{cases} \iff \begin{cases} l_{11} \geq 0 \\ l_{12} > 0 \end{cases}$$

and P_u is given by

$$\begin{cases} l_{11} \leq 0 \\ l_{12} > 0 \\ l_{11}(n-j) + l_{12}(k-j) > 0 & , j < n \leq k \\ l_{11}(k-j) + l_{12}(k-j) > 0 & , n < j < k \end{cases} \iff \begin{cases} l_{11} \leq 0 \\ l_{12} > 0 \\ l_{11} + l_{12} > 0. \end{cases}$$

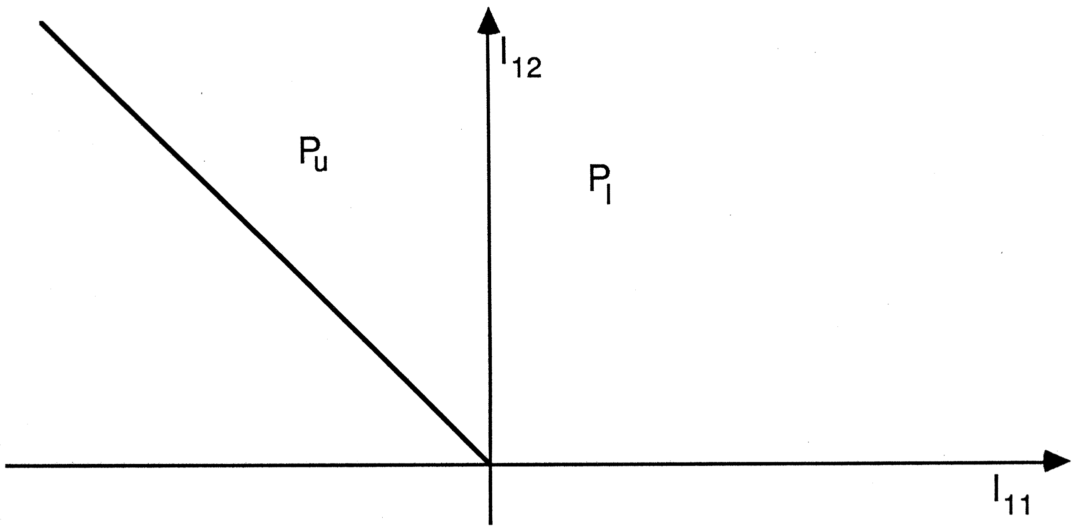

Figure 6.5: $P_l \cup P_u$

This condition will, as seen from figure 6.5, rule out our previous solution $(\,-p_+ - 1 \quad p_+\,)$. The new optimal solution becomes $(\,1 - p_+ \quad p_+\,)$ when $p_+ \geq 2$ and $(\,-1 \quad 2\,)$ when $p_+ < 2$. The execution time for $(\,1 - p_+ \quad p_+\,)$ is actually equal to the one for $(\,-p_+ - 1 \quad p_+\,)$. It furthermore has the advantage of using fewer delays for the x-operands.

6.2 Matrix multiplication

Our method of finding time-optimal systolic arrays with pipelined cells is by no means restricted to synthesis of one-dimensional arrays. To exemplify that the method works also in higher dimensions we will synthesize some time-optimal arrays for matrix multiplication. First we consider multiplication $C = AB$ where A is a dense $m \times q$-matrix and B is a dense $q \times n$-matrix. The following set of m-assignments computes the elements of C:

$1 \leq i \leq m, 1 \leq j \leq n$:

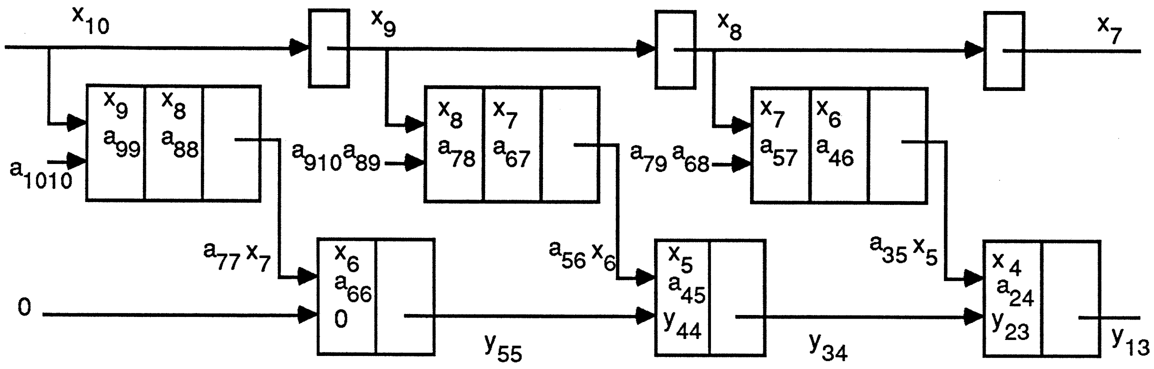

Figure 6.6: Array supporting the schedule of $L_t = (\,1 - p_+ \quad p_+\,)$ when $p_\times = 3$ and $p_+ = 2$.

$$c_{ij2} \leftarrow 0 + a_{i1}b_{1j} \qquad (\,i \quad j \quad 1\,)$$

$2 \leq k \leq q$:

$$c_{ijk+1} \leftarrow c_{ijk} + a_{ik}b_{kj} \qquad (\,i \quad j \quad k\,)$$

The results are $c_{ij} = c_{ijq+1}$. With the indexing as above the index vectors form a polyhedron with eight corners:

$$\begin{array}{cccc} (\,1 \quad 1 \quad 1\,) & (\,1 \quad n \quad 1\,) & (\,1 \quad 1 \quad q\,) & (\,1 \quad n \quad q\,) \\ (\,m \quad 1 \quad 1\,) & (\,m \quad n \quad 1\,) & (\,m \quad 1 \quad q\,) & (\,m \quad n \quad q\,) \end{array}$$

With the indexing above the c-operands yields a single data dependence vector $d = (\,0 \quad 0 \quad 1\,)$. There are no other data dependence vectors. We assume that the m-assignments are carried out by the same kind of cells as in the FIR example. Then we have the pipeline constraint $L_t d \geq p_+$, or $l_{13} \geq p_+$. Further we want designs without any signals rippling through. This gives the constraints $l_{11} \neq 0$, $l_{12} \neq 0$ and $l_{13} \neq 0$. Minimizing the maximal corner differences yields the following expression for execution time as a function of L_t:

$$|l_{11}|\,(m-1) + |l_{12}|\,(n-1) + l_{13}(q-1)$$

This function obtains its minimum $p_+(q-1) + m + n - 2$ for $l_{11} = \pm 1$, $l_{12} = \pm 1$ and $l_{13} = p_+$.

Dependent on the relation between the sizes of m, n and q different mappings to space will give the best cell utilization. The following are of interest:

$$L_{r1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \quad L_{r2} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad L_{r3} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

L_{r1} uses mn cells and is thus suitable when $q >> m, n$. In the resulting array the elements of C will be updated in place. L_{r2} uses mq cells and lets the elements of A stay in place. L_{r3} uses nq cells and lets the elements of B stay in place. For a closer description of the properties of these space mappings, see [Li87].

Interestingly these arrays show different behavior with respect to pipeline utilization and skewing. $L_t = (\,1 \quad 1 \quad p_+\,)$ and $L_r = L_{r1}$ gives skewing proportional to $m + n$, which is minimal. On the other hand the pipelines of the cells will not be filled. A new computation will be initiated at each cell every p_+ time. Selecting $L_r = L_{r2}$ or L_{r3} will give arrays with cells that have completely filled pipelines, but where the skewing is proportional to $p_+(m+q)$ and $p_+(n+q)$, repectively. Thus it is no contradiction that they all use the same time.

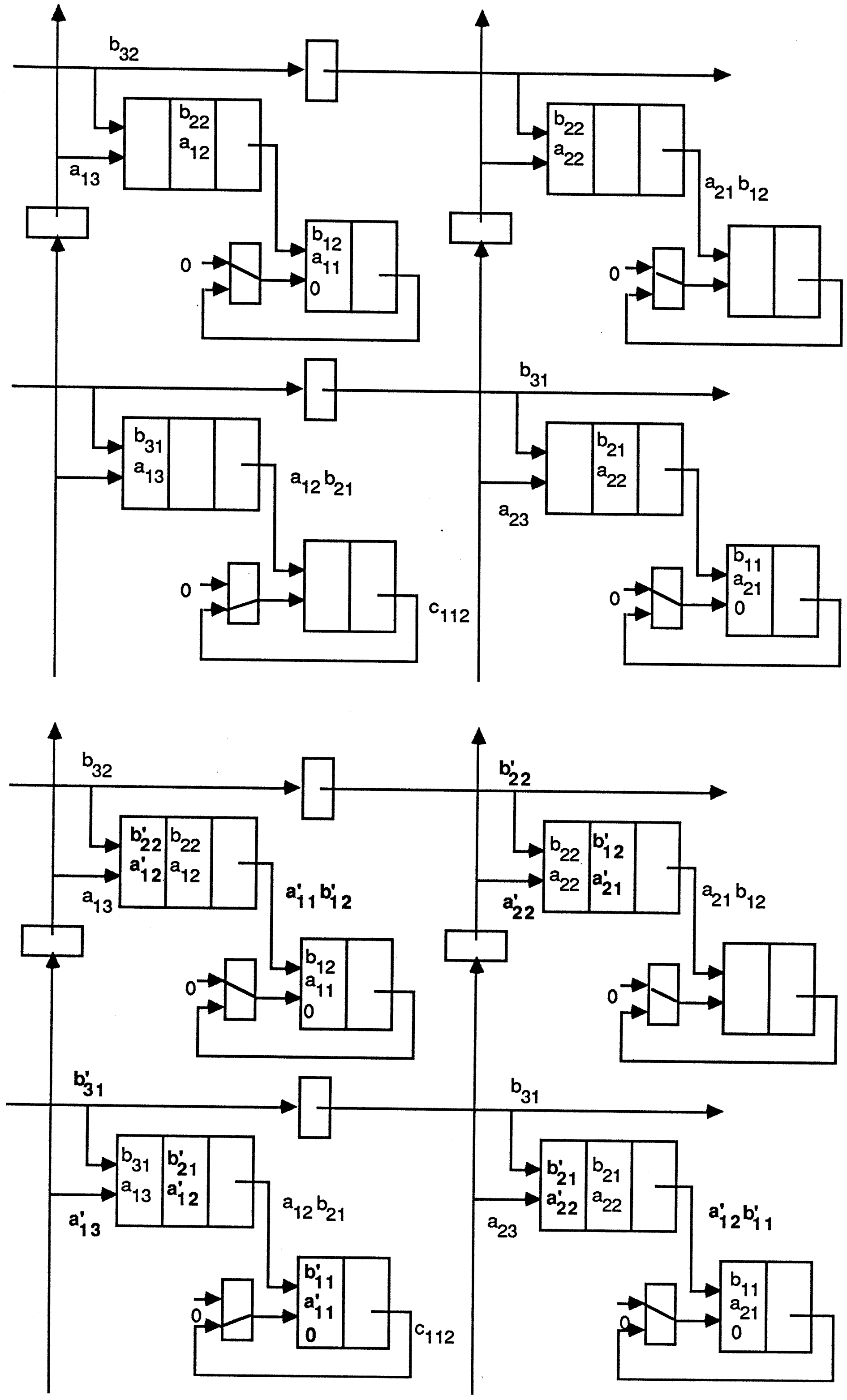

Figure 6.7: Array with pipelined cells for matrix multiplication. C is updated in place. $p_\times = 3$, $p_+ = 2$. Non-interleaved and interleaved execution.

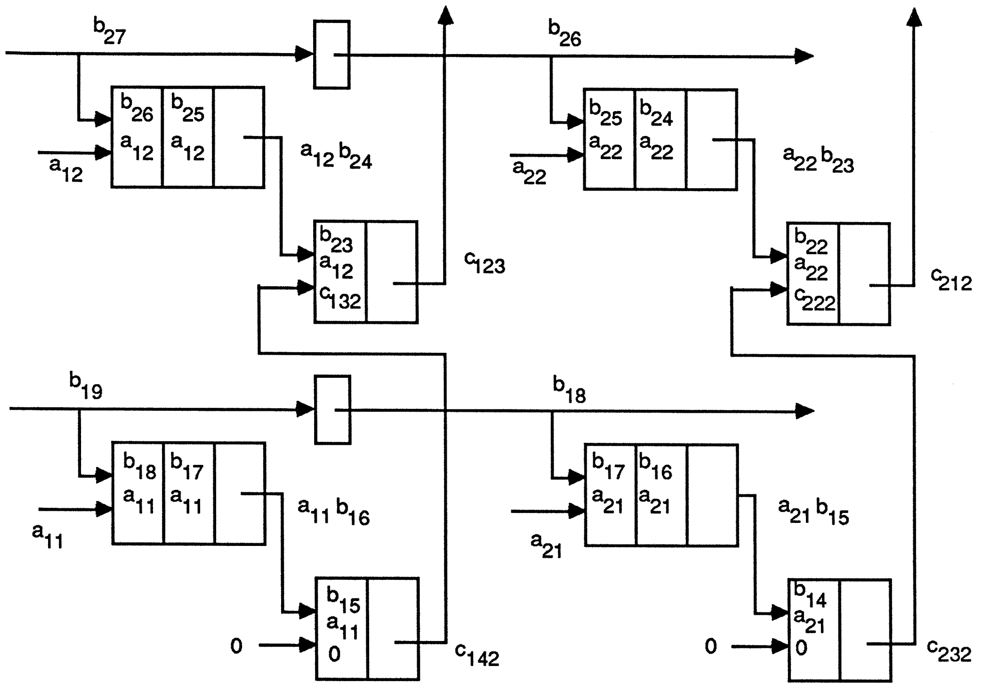

Figure 6.8: Array with pipelined cells for matrix multiplication. A stays in place. $p_\times = 3$, $p_+ = 2$.

The pipeline utilization of the L_{r1}-array can be increased by *interleaving.* p_+ independent matrix multiplications of the same size can be performed on the array, each offset one time step with respect to the previous one. The correctness of this can be formally verified by extending $\begin{pmatrix} L_t & L_{r1} \end{pmatrix}$ to a correct mapping $Z^4 \to Z^3$, where the new index enumerates the different multiplications. Cf. [Li87]. Thus p_+ independent matrix multiplications, each consisting of mnq steps, can be performed on the array in time $p_+q + m + n - 2$. Interleaving is not possible on the other two arrays.

As our last example we will consider matrix multiplication $C = AB$ where A and B are banded $n \times n$-matrices. Assume that A has u_A upper and l_A lower diagonals that are non-zero, and similarly for B. Then the m-assignments involving zero operands can be pruned away, and we obtain the following additional linear constraints on the set of index vectors:

$$\begin{array}{ll} i - k \leq l_A & k - j \leq l_B \\ k - i \leq u_A & j - k \leq u_B \end{array}$$

This will cause a number of new corners to appear, and some old will be cut away.

A space mapping that gives a good cell utilization for multiplication of banded matrices must fulfil the following condition, [Li83], [Li87]:

$$l_{21} + l_{22} + l_{23} = 0$$
$$l_{31} + l_{32} + l_{33} = 0$$

If we want nearest-neighbor communication the matrix elements should be 0, 1 or -1. If the matrix of the total mapping to space-time is to be non-singular two of the columns in the space mapping must be linearly independent. A solution that fulfils this is

$$L_r = \begin{pmatrix} -1 & 0 & 1 \\ 0 & -1 & 1 \end{pmatrix}.$$

Nonsingularity of the total space-time mapping L is now assured if $\det(L) \neq 0$, or

$$l_{11} + l_{12} + l_{13} \neq 0.$$

If we in order to minimize the number of cases assume that $l_A = u_A = l_B = u_B = b$ we obtain the following time-optimal solutions for L_t:

$p_+ > 2:$ $\quad L_{t1} = (\,-1 \quad 2 - p_+ \quad p_+\,)$

$p_+ \leq 2:$ $\quad L_{t2} = (\,-1 \quad -p_+ \quad p_+\,)$

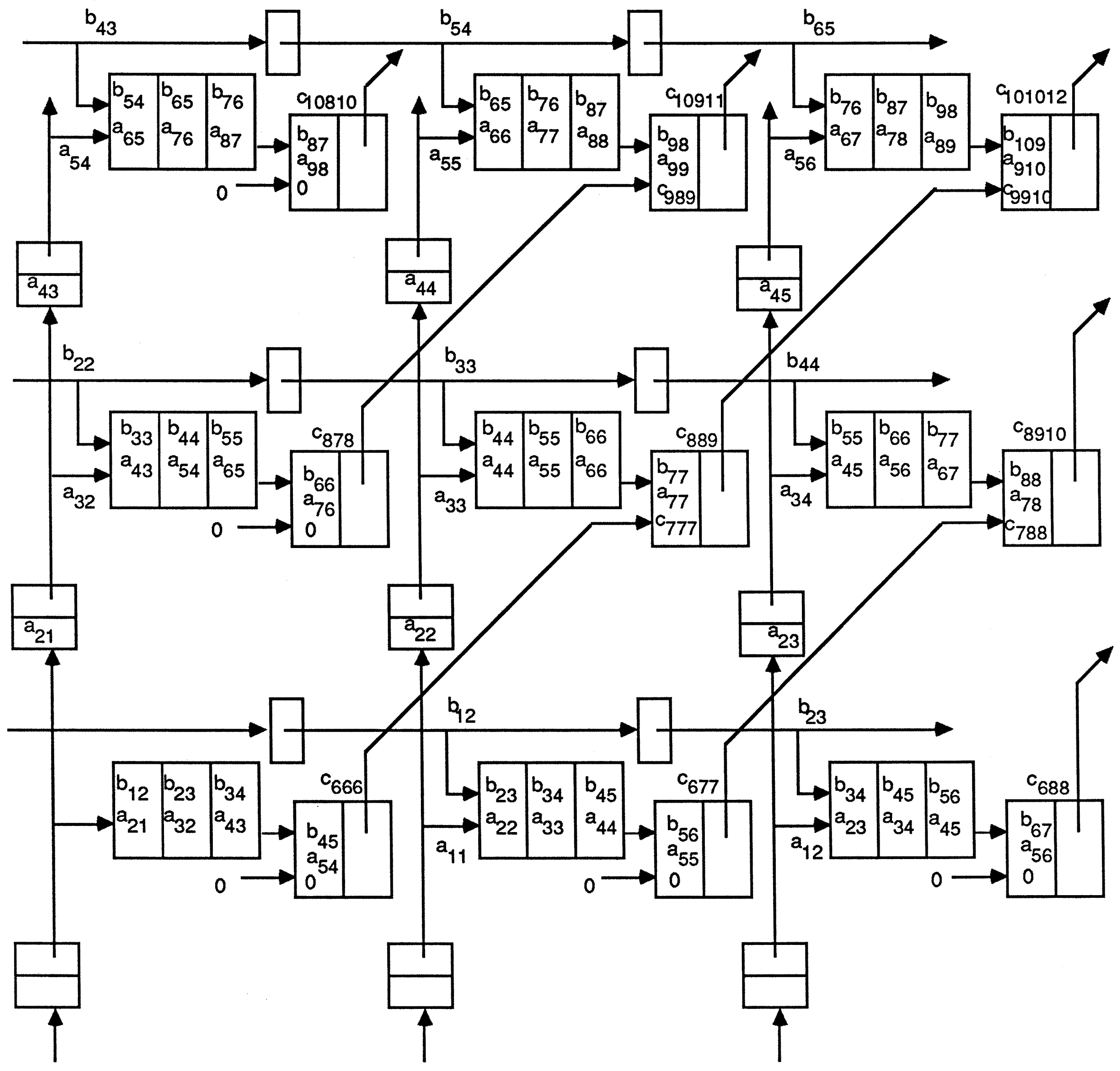

Figure 6.9: Time-optimal array for banded matrix multiplication, $p_\times = 3$, $p_+ = 2$.

As a matter of fact any L_t for which $2 - p_+ \leq l_{11}, l_{12} \leq -1$, $l_{13} = p_+$ and $l_{11} + l_{12} + l_{13} = 1$ is optimal when $p_+ > 2$. When $p_+ \leq 2$ any L_t for which $-p_+ \leq l_{11}, l_{12} \leq -1$, $l_{13} = p_+$ and $l_{11} + l_{12} + l_{13} = -1$ is optimal. These solutions have execution time $2(b-1)(p_+ - 1) + n - 1$ and $2(b-1)p_+ + n - 1$, respectively. They both have skewing proportional to $2bp_+$ and utilize the pipelines of the cells fully.

7. Conclusions

We have presented a powerful method to optimize the execution time for indexed sets of m-assignments when the time function is linear and the constraints on the design can be expressed as systems of linear inequalities. Constraints that can be expressed in this way include causality, nonsingularity of the total transformation matrix, broadcast avoidance and suitability for processing time series. Constraints arising from the inner structure of pipelined cells used to build the resulting array can also be expressed as linear inequalities. Thus the optimization method enables us to find solutions superior to pure systolic arrays when the cells are built of pipelined functional elements. Once the constraints are defined the method can automatically find a time-optimal array. The reason why so many constraints fit into this scheme is that every time we demand that an execution time must differ from or be greater than another execution time by a certain amount a new linear constraint of the form $L_t(i' - i) \geq c$ or $L_t(i' - i) \neq c$ appears.

A crucial point is that the number of inequalities must be kept within a reasonable limit. Some of the constraints described here can potientially generate very many inequalities. If the set of m-assignments and the indexing shows a certain degree of regularity, however, it is very likely that many of the inequalities will either be the same as other or dominated. A typical example is when potentially very many causality constraints reduces to constraints regarding a few data dependence vectors. Also the number of possible maximal corner differences will usually be reduced when combined with the external constraints. In the FIR example the number of non-trivial corner differences boils down from 12 to 2. A system that implements the method presented here must however be capable to detect that inequalities are dominated by others and that systems of inequalities may have no solutions and take advantage of this to keep the number of systems and inequalities down. If systems of inequalities are naïvely generated a combinatorial explosion is likely to take place. If some care on the other hand is taken it seems certain that the method can be applied to synthesis of systolic arrays for more non-trivial algorithms than the simple examples presented here.

8. Acknowledgements

I would like to thank Lennart Johnsson for his helpful comments at an early stage of the creation of this paper. This work has been supported in part by the Office of Naval Research under contract N00014-86-K-0564. Part of this work was done when the author was with the Royal Institute of Technology, Stockholm, and then it was in part supported by The Swedish Board for Technical Development (STU).

References

[A82] W. B. Ackerman, "Data Flow Languages", *Computer*, Vol. 15 (Feb. 1982), pp. 15–25.

[C86] M. C. Chen, "Transformation of Parallel Programs in Crystal", in Kugler, H.-J. ed. INFORMATION PROCESSING 86, Elsevier Publishers B.V. (North-Holland), 1986, pp. 455–462

[CF84] K. Culik II, I. Fris, *Topological Transformations As a Tool in The Design of Systolic Networks*, Report CS-84-11, Dept. Comput. Sci., University of Waterloo, April 1984

[G79] G. Grätzer, *Universal Algebra*, Springer–Verlag, New York, 1979

[HDI87] P. Hudak, J-M. Delosme, I. C. F. Ipsen, *ParLance: A Para-Functional Programming Environment for Parallel and Distributed Computing*, Research Report YALEU/DCS/RR-524, Dept. Comput. Sci., Yale University, March 1987

[JKNM86] H. V. Jagadish, R. G. Mathews, T. Kailath, J. A. Newkirk, "A Study of Pipelining in Computing Arrays", *IEEE Trans. Comput.*, Vol. C-35 (May 1986), pp. 431–440

[JWCD81] L. Johnsson, U. Weiser, D. Cohen, A. L. Davis, "Towards a Formal Treatment of VLSI Arrays", *Proc. Second Caltech Conf. on VLSI*, Jan. 1981, pp. 378–398

[KMW69] R. M. Karp, R. E. Miller, S. Winograd, "The Organization of Computations for Uniform Recurrence Equations", *JACM*, Vol. 14, No. 3 (July 1969), pp. 563–590

[KLe80] H. T. Kung, C. E. Leiserson, "Algorithms for VLSI Processor Arrays", Ch. 8.3 in C. Mead, L. Conway, *Introduction to VLSI systems*, Addison-Wesley, Reading, MA, 1980

[Ku82] H. T. Kung, "Why Systolic Architectures?", *Computer*, Vol. 15 (Jan. 1982), pp. 37–46.

[KLi83] H. T. Kung, W. T. Lin, *An Algebra for VLSI Algorithm Design*, Technical Report, Dept. Comput. Sci., Carnegie-Mellon Univ., PA, 1983

[LS81] C. E. Leiserson, J. B. Saxe, "Optimizing Synchronous Systems", *Proc. of 22nd Ann. FOCA Symposium*, 1981, pp. 23–36

[Le83] H. Lev-Ari, *Modular Computing Networks: a New Methodology for Analysis and Design of Parallel Algorithms/Architectures*, ISI Report 29, Integrated Systems Inc., Dec. 1983

[Li83] B. Lisper, *Description and Synthesis of Systolic Arrays*, Technical Report TRITA-NA-8318, NADA, KTH, Stockholm, 1983

[Li85] B. Lisper, *Hardware Synthesis from Specification with Polynomials*, Technical Report TRITA-NA-8506, NADA, KTH, Stockholm, 1985

[Li87] B. Lisper, *Synthesis of Synchronous Systems by Static Scheduling in Space-time*, Ph. D. dissertation TRITA-NA-8701, NADA, KTH, Stockholm, 1987

[MaWa85] Z. Manna, R. Waldinger, *The Logical Basis for Computer Programming, Volume I: Deductive Reasoning*, Addison-Wesley, Reading, MA, 1985

[MiWi84] W. L. Miranker, A. Winkler, "Spacetime Representations of Computational Structures", *Computing*, Vol. 32 (1984), pp. 93–114

[Mo82] D. I. Moldovan, "On the Analysis and Synthesis of VLSI algorithms", *IEEE Trans. Comput.*, Vol. C-31 (Oct. 1982), pp. 1121–1126

[Q83] P. Quinton, *The Systematic Design of Systolic Arrays*, Research Report RR 216, INRIA, Rennes, July 1983.

[QG85] P. Quinton, P. Gachet, *Automatic Design of Systolic Chips*, Research Report RR 450, INRIA, Rennes, Oct. 1985.

[RPF86] S. V. Rajopadye, S. Purushotaman, R. Fujimoto, *On Synthesizing Systolic Arrays from Recurrence Relations with Linear Dependencies*, Detailed Summary, Dept. Comput. Sci., University of Utah, 1986

[RFS82A] I. V. Ramakrishnan, D. S. Fussell, A. Silberschatz, *Towards a Characterization of Programs for a Model of VLSI Array-Processors*, Technical Report TR-202, Dept. Comput. Sci., University of Texas at Austin, July 1982

[WD81] U. Weiser, A. L. Davis, “A Wavefront Tool for VLSI Design”, in H. T. Kung, B. Sproull and G. Steele eds. *VLSI Systems and Computations*, Springer-Verlag, Berlin, 1981, pp. 226–234