Cpr E 583 Adaptive Computing Systems	Fall Semester 1998
Lecture 17, 18: October 20 & October 22, 1998	
Instructor: Arun Somani, Akhilesh Tyagi	Scribe: Murari Sridharan

1 Signal Flow Graphs - An Introduction

The Signal Flow Graph (SFG) offers a powerful abstraction and graphical representation for problems in scientific and signal processing computations. The SFG expression consists of

- processing *nodes*,
- communicating *edges*,
- *delays*

In general a *node* is denoted by a circle representing an arithmatic or logic function performed with *zero delay*, such as multiply or add. An *edge*, on the other hand, denotes either a dependence relation or delay. As compared with the DG, which we studied in the previous lectures, the SFG has the following properties,

- The SFG is a more concise representation than the DG. (DG is not optimized for communication and PE use)
- The SFG is more closer to hardware level design
- SFG can have loops, as long as there is one delay D on each loop

More formally, the structural part of an SFG can be represented by a directed graph,$\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{D}(\mathcal{E}))$. THe vertices $\mathcal{V}$ represent the nodes, the directed edges$\mathcal{E}$ represent the interconnection between the nodes. Each edge $\rceil$ of $\mathcal{E}$ connects a *source* to a *sink* and is a weighted edge with a delay count D(e). Existence of delay-free edges means that the data will be propogated in 'zero time'. The computation in each node is also assumed to be done in 'zero time'.

2 Mapping DG to SFG

2.1 Processor Assignment and Scheduling

There are two basic considerations for mapping from an DG to an SFG,

- *Processor Assignment*
- *Scheduling*

The first step is the *processor assignment*. Once the processor assignment is fixed, the second step is *scheduling*. Eventhough the allowable processor and schedule assignments can be very general, the regularity of the DG can be exploited to derive a regular systolic array. Before we see how this is done, we have to understand the terms *Projection*, *Equitemporal Hyperplane*. It is common to use *linear projection* for processor assignment, in which nodes of the DG in a certain straight

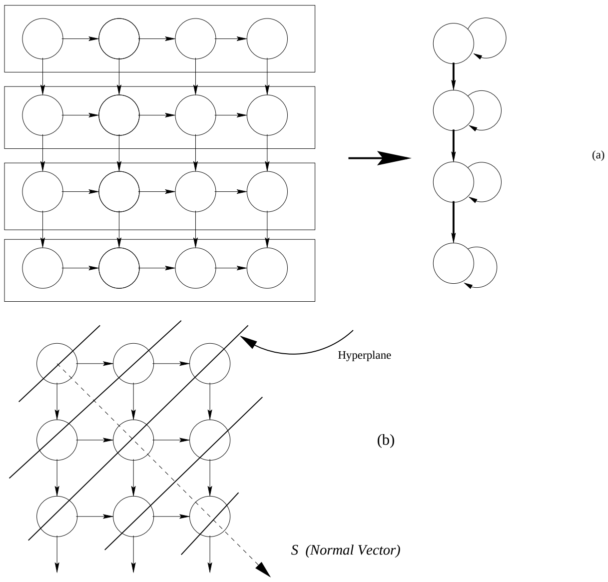

Figure 1: (a) Linear Projection (b) a linear schedule and its hyperplane

line are projected (or assigned) to a PE in the processor array (as shown in Figure. 2(a)) and a *linear scheduling* for schedule assignment, in which nodes in a parallel hyperplane in the DG are scheduled to be processed at the same time step (Equitemporal) as shown in Figure. 2(b).

Processor Assignment

One simple projection is that the nodes of the DG along a straight line are assigned to a common PE. Since the DG of a locally recursive algorithm is very regular, the projection maps the DG onto a lower dimensional lattice of points known as the *processor space*. Mathematically, a linear projection is represented by a *projection vector* $\vec{d}$. The results of this projection are represented by an SFG. In Figure. 2(a), the 2-D index space may be decomposed into a 1-D processor space.

Scheduling

The projection should be accompanied by a *scheduling* scheme which specifies the sequence of operations in all PEs. A schedule function represents a mapping from the N-dimensional index space of the DG onto a 1-D schedule space. A linear schedule is based on a set of parallel and uniformly spaced hyperplanes in the DG. These hyperplanes are called *equitemporal hyperplanes*. Mathematically, the schedule can be represented by a column schedule vector $\vec{s}$, pointing to the normal direction of the hyperplanes. A set of linear schedule hyperplanes and the schedule vector are shown in Figure. 2(b). The set of equitemporal hyperplanes define a partial order among computations, inherent in the algorithm. This means that if there is a directed path from node x to node y, then computation represented by node y must be executed after the computation executed by node x is completed. The feasibility of a schedule is determined by the partial ordering and the processor assignment scheme. We will see the necessary and sufficient conditions now.

Permissible Linear Schedules

Given a DG and a projection direction $\vec{d}$, not all the hyperplanes define a valid schedule for the

DG. Some of them violate the precedence relation of computation. The allowable directions define the permissible linear schedules. The necessary and sufficient conditions that the normal vector $\vec{s}$ satisfies in order for the given hyperplanes to represent a permissible linear schedule are as follows:

1. $\vec{s}^T\vec{e} \geq 0$, for any dependence arc $\vec{e}$
2. $\vec{s}^T\vec{d} > 0$

In other words, the schedule is permissible iff (1) all the dependency arcs flow in the same direction across the hyperplanes; and (2) the hyperplanes are not parallel with projection vector $\vec{d}$. The first condition enforces the fact that if node p depends on node q, then the time step assigned for p cannot be less than the time step assigned for q. The second condition implies that the nodes on an equitemporal hyperplane should not be projected on the same PE. This can be seen from the equation the dot product of $\vec{s}.\vec{d}$ is given by $\vec{s}.\vec{d}cos\theta$. If $\theta = 90$, the dot product is zero. This means that $\vec{d}$ is perpendicular to $\vec{s}$ which means that it is parallel to the hyperplane.

Types of Schedules
Given a DG and a projection vector, $\vec{d}$, the most likely used schedules for the SFG projection are the following,

- Default Schedule
- Recursion Schedule
- Systlolic Schedule
- Optimized Schedule

Time Invariance
A time invariant (TI) SFG is one whose node functions as well as the edge structure remain time invariant all the time.

2.2 SFG Projection Procedure

In general the projection procedure invlolves the following steps,

1. The processor space is orthogonal to the projection direction. A processor array is obtained by projecting the index points to the processor space. The default schedule is fixed for the projection direction. If the default schedule violates the permissible schedule conditions, then a recursion schedule may be used.
2. The arcs in DG are replaced by zero or nonzero delay edges in the corresponding processors.
3. Assign the input and output data to their corresponding processors.

2.2.1 Projection Algebra

The following mappings

1. *Node Mapping*
   Nodes in the DG are mapped to processors. The index space of DG is mapped to Indes space of SFG. This is represented by the mapping,

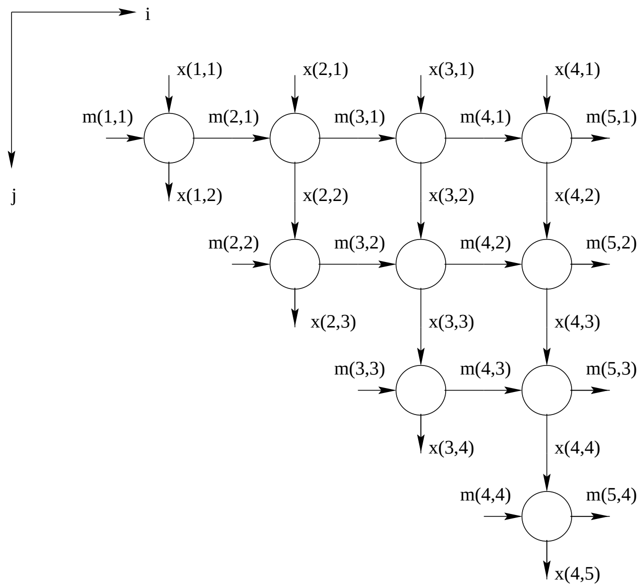

Figure 2: DG for sorting

$$\mathcal{P} : \underbrace{\mathcal{R}}_{IndexSpaceofDG} \rightarrow \underbrace{\mathcal{I}^{n-1}}_{IndexSpaceofSFG}$$

where $\mathcal{R}$ is the index set of nodes in DG and $\mathcal{I}^{n-1}$ is the cartesian product of (n-1) integers. Mapping a computation c in the DG onto a node n in the SFG is found by:

$$n = \mathcal{P}^T c$$

where $\mathcal{P}$, denoted by an n * (n-1) matrix, is orthogonal to $\vec{d}$. Mathematically,

$$\mathcal{P}^T \vec{d} = 0$$

2. *Arc Mapping*
   This mapping maps the arcs in DG to edges in SFG.

$$\vec{e} = \mathcal{P}^T \vec{b}$$

   where $\vec{e}$ is an edge in SFG and $\vec{b}$ is an edge in DG. The delay on each edge $\vec{e}$ is given by

$$D(\vec{e}) = \vec{s}^T \vec{b}$$

3. *I/O Mapping*
   The SFG node position, $n_{i/o}$ and time, t(c $\rightarrow n_{i/o}$) are given by

$$n_{i/o} = \mathcal{P}^T c$$

$$\text{t}(\text{c} \rightarrow n_{i/o}) = \vec{s}^T c$$

Consider the DG for sorting example shown in Figure 2. Let us consider applying different projection direction and permissible schedules to obtain SFGs corresponding to different sorters.

2.2.2 Insertion Sorter SFG

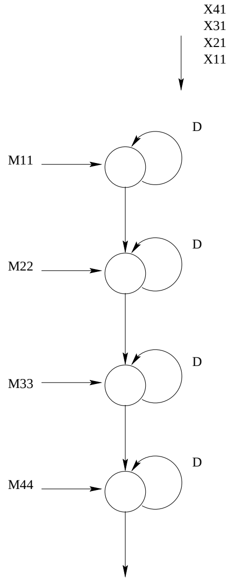

Figure 3: SFG for insertion sorting

$$\vec{d} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \vec{s} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \vec{p} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

1. *Node Mapping :*
$$\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = j$$

2. *Arc Mapping :*
$$\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 0 \quad \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 1$$

3. *Delay $D(\vec{e})$*
$$\begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 1 \quad \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 0$$

4. *Input Mapping for x*
$$\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ 1 \end{bmatrix} = 1$$

   Input Mapping for m
$$\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = j$$

5. *Output Mapping*

$$\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} N \\ j \end{bmatrix} = j \quad \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} N \\ j \end{bmatrix} = N$$

The resulting SFG is shown in Figure 3. It corresponds to Insertion Sorter. The input data is sequentially read at the first node. The output is obtained at different nodes. They may be parallely or sequentially obtained through an output bus.

2.2.3 Selection Sorter SFG

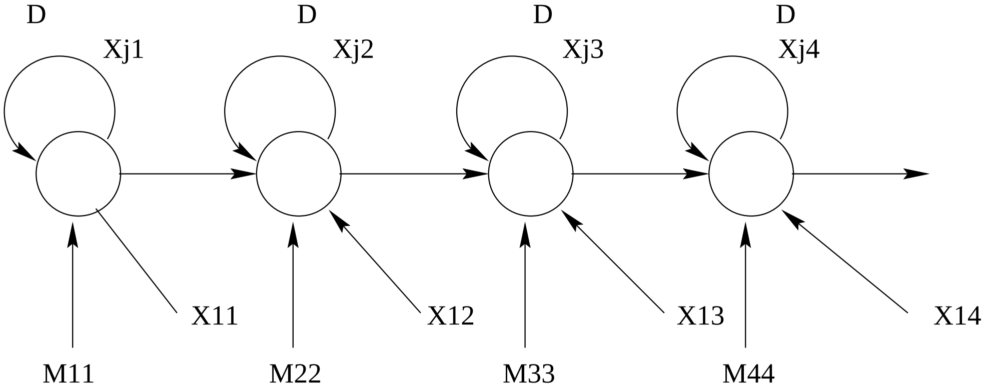

Figure 4: SFG for Selection sorting

$$\vec{d} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \vec{s} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \vec{p} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

1. *Node Mapping :*

$$\begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = i$$

2. *Arc Mapping :*

$$\begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 0 \quad \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 1$$

3. *Delay $D(\vec{e})$*

$$\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 1 \quad \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 0$$

4. *Input Mapping for x*

$$\begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} i \\ 1 \end{bmatrix} = i$$

Input delay for x

$$\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ 1 \end{bmatrix} = 1$$

Input Mapping for m

$$\begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = i$$

Input delay for m

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} i \\ i \end{bmatrix} = i$$

5. *Output Mapping*

$$\begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} N \\ j \end{bmatrix} = N$$

Output Delay

$$\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} N \\ j \end{bmatrix} = j$$

The resulting SFG is shown in Figure 4. It corresponds to Selection Sorter.

2.2.4 Bubble Sorter SFG

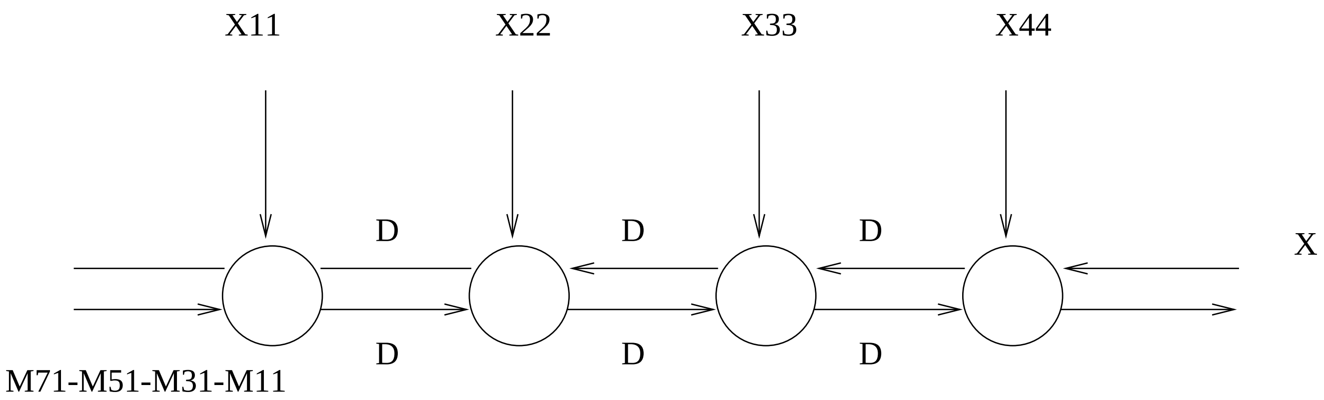

Figure 5: SFG for Bubble sorting

$$\vec{d} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \vec{s} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \vec{p} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

1. *Node Mapping :*

$$\begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = i - j$$

2. *Arc Mapping :*

$$\begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = -1 \quad \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}] = 1$$

3. *Delay* $D(\vec{e})$

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 1 \quad \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 1$$

4. *Input Mapping for* x

$$\begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} i \\ 1 \end{bmatrix} = i - 1$$

Input delay for x

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} i \\ 1 \end{bmatrix} = i + 1$$

Input Mapping for m

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

Input delay for m

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} i \\ i \end{bmatrix} = 2i$$

5. *Output Mapping*

$$\begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} N \\ j \end{bmatrix} = N - j$$

Output delay

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} N \\ j \end{bmatrix} = N + j$$

The resulting SFG is shown in Figure 5. It corresponds to Bubble Sorter.