2

TERMINOLOGY AND NOTATIONS

In this chapter we introduce terminology and definitions used in the remainder of the thesis. We also formalize the scheduling concepts that were presented intuitively in the previous chapter.

2.1 HSDF graphs and associated graph theoretic notation

We represent an HSDFG by an ordered pair (V, E), where V is the set of vertices (actors) and E is the set of edges. We refer to the source and sink vertices of a graph edge e by $src(e)$ and $snk(e)$, and we denote the delay (or the number of initial tokens) on e by $delay(e)$. We say that e is an **output edge** of $src(e)$, and that e is an **input edge** of $snk(e)$. We will also use the notation (v_i, v_j), $v_i, v_j \in V$, for an edge directed from v_i to v_j.

A **path** in (V, E) is a finite, non-empty sequence $(e_1, e_2, \ldots, e_n)$, where each e_i is a member of E, and $snk(e_1) = src(e_2)$, $snk(e_2) = src(e_3)$, …, $snk(e_{n-1}) = src(e_n)$. We say that the path $p = (e_1, e_2, \ldots, e_n)$ **contains** each e_i and each subsequence of $(e_1, e_2, \ldots, e_n)$; p is **directed from** $src(e_1)$ **to** $snk(e_n)$; and each member of $\{src(e_1), src(e_2), \ldots, src(e_n), snk(e_n)\}$ is **on** p. A path that is directed from a vertex to itself is called a **cycle**, and a **fundamental cycle** is a cycle of which no proper subsequence is a cycle.

If $p = (e_1, e_2, \ldots, e_n)$ is a path in an HSDFG, then we define the **path delay** of p, denoted $Delay(p)$, by $Delay(p) = \sum_{i=1}^{n} delay(e_i)$. Since the delays on all HSDFG edges are restricted to be non-negative, it is easily seen that between any two vertices $x, y \in V$, either there is no path directed from x to y, or there exists a (not necessarily unique) **minimum-delay path** between x and y. Given an HSDFG G, and vertices x, y in G, we define $\rho_G(x, y)$ to be equal to the path delay of a minimum-delay path from x to y if there exist one or more paths from x to y, and equal to ∞ if there is no path from x to y. If G is understood, then we may drop the subscript and simply write "ρ" in place of "ρ_G".

By a **subgraph** of (V, E), we mean the directed graph formed by any $V' \subseteq V$ together with the set of edges $\{e \in E | src(e), snk(e) \in V'\}$. We denote the subgraph associated with the vertex-subset V' by $subgraph(V')$. We say that (V, E) is **strongly connected** if for each pair of distinct vertices x, y, there is a path directed from x to y and there is a path directed from y to x. We say that a subset $V' \subseteq V$ is strongly connected if $subgraph(V')$ is strongly connected. A **strongly connected component (SCC)** of (V, E) is a strongly connected subset $V' \subseteq V$ such that no strongly connected subset of V properly contains V'. If V' is an SCC, then when there is no ambiguity, we may also say that $subgraph(V')$ is an SCC. If C_1 and C_2 are distinct SCCs in (V, E), we say that C_1 is a **predecessor SCC** of C_2 if there is an edge directed from some vertex in C_1 to some vertex in C_2; C_1 is a **successor SCC** of C_2 if C_2 is a predecessor SCC of C_1. An SCC is a **source SCC** if it has no predecessor SCC; and an SCC is a **sink SCC** if it has no successor SCC. An edge e is a **feedforward** edge of (V, E) if it is not contained in an SCC, or equivalently, if it is not contained in a cycle; an edge that is contained in at least one cycle is called a **feedback** edge.

Given two arbitrary sets S_1 and S_2, we define the difference of these two sets by $S_1 - S_2 = \{s \in S_1 | s \notin S_2\}$, and we denote the number of elements in a finite set S by $|S|$. Also, if r is a real number, then we denote the smallest integer that is greater than or equal to r by $\lceil r \rceil$.

For elaboration on any of the graph-theoretic concepts presented in this section, we refer the reader to Cormen, Leiserson, and Rivest [Corm92].

2.2 Schedule notation

To model execution times of actors (and to perform static scheduling) we associate execution time $t(v) \in Z^+$ (non-negative integer) with each actor v in the HSDFG; $t(v)$ assigns execution time to each actor v (the actual execution time can be interpreted as $t(v)$ cycles of a base clock). Inter-processor communication costs are represented by assigning execution times to the *send* and *receive* actors. The values $t(v)$ may be set equal to execution time *estimates* when exact execution times are not available, in which case results of the computations that make use of these values (e.g. the iteration period T) are compile time estimates.

Recall that actors in an HSDFG are executed essentially infinitely. Each firing of an actor is called an **invocation** of that actor. An **iteration** of the HSDFG corresponds to one invocation of every actor in the HSDFG. A schedule specifies processor assignment, actor ordering and firing times of actors, and these may be done at compile time or at run time depending on the scheduling strategy being employed. To specify firing times, we let the function $start(v, k) \in Z^+$ represent the time at which the kth invocation of the actor v starts. Correspondingly, the function $end(v, k) \in Z^+$ represents the time at which the kth execution of the actor v completes, at which point v produces data tokens at its output edges. Since we are interested in the kth execution of each actor for $k = 0, 1, 2, 3, \ldots$, we set $start(v, k) = 0$ and $end(v, k) = 0$ for $k < 0$ as the "initial conditions". If the kth invocation of an actor v_j takes *exactly* $t(v_j)$, then we can claim:

$$end(v_j, k) = start(v_j, k) + t(v_j).$$

Recall that a fully-static schedule specifies a processor assignment, actor ordering on each processor, and also the precise firing times of actors. We use the following notation for a fully-static schedule:

Definition 2.1: A fully-static schedule S (for P processors) specifies a triple:

$$S = \{\sigma_p(v), \sigma_t(v), T_{FS}\},$$

where $\sigma_p(v) \rightarrow [1, 2, \ldots, P]$ is the processor assignment, and T_{FS} is the iteration period. An FS schedule specifies the firing times $start(v, k)$ of all actors, and since we want a finite representation for an infinite schedule, an FS schedule is constrained to be periodic:

$$start(v, k) = \sigma_t(v) + kT_{FS},$$

$\sigma_t(v)$ is thus the starting time of the first execution of actor v (i.e. $start(v, 0) = \sigma_t(v)$). Clearly, the throughput for such a schedule is T_{FS}^{-1}.

The $\sigma_p(v)$ function and the $\sigma_t(v)$ values are chosen so that all data precedence constraints and resource constraints are met. We define precedence constraints as follows:

Definition 2.2: An edge $(v_j, v_i) \in V$ in an HSDFG (V, E) represents the (data) **precedence constraint**:

$$start(v_i, k) \geq end(v_j, k - delay((v_j, v_i))), \forall k \geq delay(v_j, v_i).$$

The above definition arises because each actor consumes one token from each of its input edges when it fires. Since there are already $delay(e)$ tokens on each incoming edge e of actor v, another $k - delay(e) - 1$ tokens must be produced on e before the kth execution of v can begin. Thus the actor $src(e)$ must have completed its $(k - delay(e) - 1)$th execution before v can begin its kth execution. The "-1"s arise because we define $start(v, k)$ for $k \geq 0$ rather than $k > 0$.

This is done for notational convenience.

Any schedule that satisfies all the precedence constraints specified by edges in an HSDFG G is also called an **admissible schedule** for G [Reit68]. A **valid execution** of an HSDFG corresponds to a set of firing times $\{ start(v_i, k) \}$ that correspond to an admissible schedule, i.e. a valid execution respects all data precedences specified by the HSDFG.

For the purposes of the techniques presented in this thesis, we are only interested in the precedence relationships between actors in the HSDF graph. In a general HSDFG one or more pairs of vertices can have multiple edges connecting them in the same "direction." Such a situation often arises when a multirate SDF graph is converted into a homogeneous HSDFG. Multiple edges between the same pair of vertices in the same direction are redundant as far as precedence relationships are concerned. Suppose there are multiple edges from vertex v_i to v_j, and amongst these edges the edge that has minimum delay has delay equal to d_{min}. Then, if we replace all these edges by a single edge with delay equal to d_{min}, it is easy to verify that this single edge maintains the precedence constraints for *all* the edges that were directed from v_i to v_j. Thus a general HSDF graph may be preprocessed into a form where the source and sink vertices uniquely identify an edge in the graph, and we may represent an edge $e \in E$ by the ordered pair $(src(e), snk(e))$. The multiple edges are taken into account individually when buffers are assigned to the arcs in the graph.

As we discussed in section 1.2.1, in some cases it is advantageous to *unfold* a graph by a certain unfolding factor, say u, and schedule u iterations of the graph together in order to exploit inter-iteration parallelism more effectively. The unfolded graph contains u copies of each actor of the original graph. In this case σ_p and σ_t are defined for all the vertices of the *unfolded* graph (i.e. σ_p and σ_t are defined for u invocations of each actor); T_{FS} is the iteration period for the unfolded graph,

and the average iteration period for the original graph is then $\frac{T_{FS}}{u}$. In the remainder of this thesis, we assume we are dealing with the unfolded graph and we refer only to the iteration period and throughput of the unfolded graph, if unfolding is in fact employed, with the understanding that these quantities can be scaled by the unfolding factor to obtain the corresponding quantities for the original graph.

In a self-timed scheduling strategy, we determine a fully-static schedule, $\{\sigma_p(v), \sigma_t(v), T_{FS}\}$, using the execution time estimates, but we only retain the processor assignment σ_p and the ordering of actors on each processor as specified by σ_t, and discard the precise timing information specified in the fully-static schedule. Although we may start out with setting $start(v, 0) = \sigma_t(v)$, the subsequent $start(v, k)$ values are determined at runtime based on availability of data at the input of each actor; the average iteration period of a self-timed schedule is represented by T_{ST}. We analyze the evolution of a self-timed schedule further in Chapter 4.