HIGH-LEVEL TRANSFORMATIONS

Topics:
* Data-flow graphs
* (Non)overlapped scheduling
* Minimal iteration period

Transformations for speed-up
Transformations for low power

Further reading:

DATA-FLOW MODEL OF COMPUTATION

Data-flow graphs (DFGs) explicitly represent parallelism in computations. A DFG may or may not contain information on control flow.

A data-flow graph is built from:
* nodes (vertices): representing computation, and
* edges: representing precedence relations.

DATA FLOW

Example:
\[
x := a \times b; \\
y := c + d; \\
z := x + y;
\]

TOKEN FLOW IN A DFG

* A node in a DFG fires when tokens are present at its inputs.
* The input tokens are consumed and an output token is produced.
**IMPLICIT ITERATIVE DATA FLOW**

- Iteration implied by stream of input tokens arriving at regular instants in time. The computation of the DFG is repeated every $T_0$ time units.
- Initial tokens act as buffers.

![Diagram of iterative data flow](image)

**IMPLICIT ITERATIVE DATA FLOW (Ctd.)**

- *Delay elements* instead of initial tokens.

Two notations are encountered:

- explicit delay elements
- delay elements as an edge property

![Diagram of delay elements](image)

**ITERATIVE DFG EXAMPLE**

A second-order filter section.

**IDFG NOTATION**

$\text{IDFG}(V,E)$ with:

- $V$: the vertex set: $V = C \cup D \cup I \cup O$
- $C$: set of computational nodes
- $D$: set of delay nodes
- $I$: set of of input nodes
- $O$: set of output nodes
- $E$: the edge set
- $\delta(c), c \in C$ gives the duration of a computation (atomic, non-preemptive, restricted library)
- $\mu(d), d \in D$ gives the multiplicity of a delay node
SYNCHRONOUS DATA-FLOW

The iterative data-flow graph is a special case of a synchronous data-flow graph (introduced by Edward Lee).

Characteristics:
* no conditional nodes
* each edge has integer attributes for numbers of tokens produced at one side and consumed at the other: multirate system
* each edge has a delay attribute.


TERMINOLOGY

Subtasks:
* Scheduling: determine for each operation the time at which it should be performed such that no precedence constraint is violated.
* Allocation: specify the hardware resources that will be necessary.
* Assignment: provide a mapping from each operation to a specific functional unit and from each variable to a register.

Remarks:
* The subproblems are strongly interrelated; they are, however, often solved separately.
* Scheduling (except for a few versions) is NP-complete ⇒ heuristics have to be used.

OPTIMIZATION CRITERIA

Most commonly used:
* Time-constrained synthesis: given the iteration period $T_0$, use as few processors as possible or as little hardware as possible (typical for DSP).
* Resource-constrained synthesis: given a multiprocessor configuration or a set of hardware resources on chip, minimize $T_0$.

Another important issue:
* Minimization of power.

SCHEDULING TERMINOLOGY

* Static scheduling means: mapping to time and processor (functional unit, register, etc.) is identical in all iterations.
* A static schedule is either overlapped (exploiting interiteration parallelism) or nonoverlapped.

SCHEDULING TERMINOLOGY (Ctd.)

- Overlapped scheduling is also called: loop folding, software pipelining.
- The delay between consumption of input and production of output is called the latency \( \lambda \). In general \( \lambda \neq T_0 \).
- An overlapped schedule may allow shorter iteration period or hardware utilization, but:
  - the search space is larger and finding optimal solutions harder.

Not covered in this presentation:
- cyclostatic schedules
- dynamic schedules (requires a run-time scheduler).

\[
\begin{align*}
\text{P}_1 & : 5 \quad 4 \quad 3 \quad 1 \quad 0 \quad 6 \quad 5 \\
\text{P}_2 & : 3 \quad 1 \quad 5 \quad 4 \quad 3 \quad 0 \quad 6 \\
\text{P}_3 & : 3 \quad 1 \quad 5 \quad 4 \quad 3 \quad 0 \quad 6 \\
\text{P}_4 & : 3 \quad 1 \quad 5 \quad 4 \quad 3 \quad 0 \quad 6
\end{align*}
\]

\textit{A cyclostatic schedule}

The minimal iteration period \( T_0 \) (\( T_{0,\text{min}} \))

There are four cases:
- Acyclic DFG, nonoverlapped schedule;
- Acyclic DFG, overlapped schedule;
- Cyclic DFG, nonoverlapped schedule;
- Cyclic DFG, overlapped schedule.

For the nonoverlapped cases:
- Compute the critical path, longest path from any input to any output, in the acyclic graph. \( T_{0,\text{min}} = \text{“length of critical path”} \).

EXAMPLE

\[
\begin{align*}
&c_1 \\
&c_2 \\
&c_3 \\
&c_4 \\
&c_5 \\
&c_6 \\
&c_7
\end{align*}
\]

Critical path in a cyclic DFG for a nonoverlapped schedule.

OVERLAPPED SCHEDULING IN A CYCLIC DFG

- Minimal iteration period is given by critical loop.

\[
T_0 \geq \delta(c_1) + \delta(c_2) + \delta(c_3)
\]

* When the DFG is acyclic, arbitrarily small iteration periods are possible (just duplicate the hardware as often as necessary; each copy can start any time as there are no feedback loops in the DFG; see later on).
CRITICAL LOOP

For a general DFG, $T_{0\text{min}}$ is given by:

$$T_{0\text{min}} = \max_{\text{all loops } l} \left( \sum_{c \in l} \delta(c) \right) \left( \sum_{d \in l} \mu(d) \right)$$

Remarks:

* Direct use of expression for $T_{0\text{min}}$ not efficient (number of loops in graph may grow exponentially with respect to number of nodes).

* Many polynomial-time algorithms have been published; survey in:


EXAMPLE

$T_{0\text{min}} = 3$, when "\(\delta(+) = 1\)" and "\(\delta(\times) = 2\)".

SPEED-UP TECHNIQUES: PIPELINING

Insert delay elements on all edges that are cut by a cut line through an edge of the critical path in the DFG.

* Works for acyclic DFGs.

* Schedule becomes overlapped.
SPEED-UP TECHNIQUES: RETIMING

* Useful for obtaining the minimal $T_0$ for a non-overlapped schedule by reduction of critical-path length, both for cyclic and acyclic IDFGs.

Example

SOME REMARKS ON $T_{0\min}$

* Retiming does not affect $T_{0\min}$ for overlapped scheduling of IDFG's.
* The $T_0$ for non-overlapped scheduling obtained after optimal retiming may still be larger than $T_{0\min}$.
* $T_{0\min}$ has been defined as an integer; a fractional $T_{0\min}$ makes sense when unfolding is applied (unfolding creates a new DFG of multiple copies of the original one; see later).


SPEED-UP TECHNIQUES: PARALLEL PROCESSING

* Works for acyclic IDFGs.
* Duplicate the IDFG as often as desired speed-up factor.
* Allows any arbitrary speed-up, but is proportionally expensive.

process 2 inputs at a time

UNFOLDING (1)

* A technique for the duplication of cyclic IDFGs in combination with processing multiple inputs at a time.
* Consider the following IDFG:

\[ i[k] \rightarrow T_0 \rightarrow o[k] \]

* If $\delta(+) = 1$ and $\delta(*) = 2$, $T_{0\min} = \frac{3}{2}$.
* Using unfolding by 2, one can reach the value $T_{0\min} = \frac{3}{2}$.
* The graph computes the following difference equations, assuming that one multiplies by a factor $a$:
  \[
  i[k] = i[k] + o[k - 1] \\
  o[k] = a s[k - 1]
  \]
UNFOLDING (2)

* The precise unfolding algorithm will not be given here; it amounts to duplicating all vertices in the IDFG such that \( n \) copies of each vertex is created (\( n \) is the unfolding factor) and then to connecting these vertices with edges having an appropriate number of delay elements. The unfolded graph can also be reconstructed from the equations.

\[
x[2k] = i[2k] + o[2k - 1]
\]
\[
x[2k + 1] = i[2k + 1] + o[2k]
\]
\[
o[2k] = as[2k - 1]
\]
\[
o[2k + 1] = as[2k]
\]

LOOK-AHEAD TRANSFORMATION (1)

* Consider the following computation:

\[
x[n] = ax[n - 1] + u[n]
\]

* It has one multiplication and one addition in the critical loop with one delay element. If \( \delta(+) = 1 \) and \( \delta(\times) = 2 \), \( T_{0\text{min}} = \left\lceil \frac{3}{2} \right\rceil = 3 \).

UNFOLDING (3)

* Note that the unfolded IDFG has two loops with one delay element each and a computational duration of 3. Because a delay element creates an offset of two indices (2 inputs are processed in each iteration), the effective iteration period bound is equal to \( T_{0\text{min}} = \frac{3}{2} \).

LOOK-AHEAD TRANSFORMATION (2)

* Apply look-ahead transformation (think of the principle of look-ahead addition):

\[
x[n] = ax[n - 2] + u[n - 1]
\]
\[
x[n] = a^2x[n - 2] + au[n - 1] + u[n]
\]

* The new equation has one multiplication and one addition in the critical loop with two delays leading to \( T_{0\text{min}} = \left\lceil \frac{3}{2} \right\rceil = 2 \).
* The transformation can affect the original computation (final wordlength effects).