HIGH-LEVEL TRANSFORMATIONS

Topics:

* Data-flow graphs
* (Non)overlapped scheduling
* Minimal iteration period
* Transformations for speed-up
* Transformations for low power

Further reading:

☞ Parhi, K.K., "High-Level Algorithm and Architecture Transformations for DSP Synthesis", Journal of VLSI Signal Processing, Vol. 9, pp 121–143, (1995).

☞ Gerez, S.H., S.M. Heemstra de Groot, E.R. Bonsma and M.J.M. Heijligers, "Overlapped Scheduling Techniques for High-Level Synthesis and Multiprocessor Realizations of DSP Algorithms", In: J.C. Lopez, R. Hermida and W. Geisselhardt (Eds.), Advanced Techniques for Embedded System Design and Test, Kluwer Academic Publishers, Boston, pp 125–150, (1998).

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.

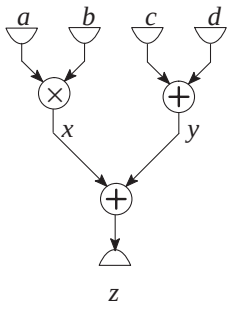
A data-flow graph is built from:

* *nodes (vertices):* representing computation, and
* *edges:* representing *precedence* relations.

DATA FLOW

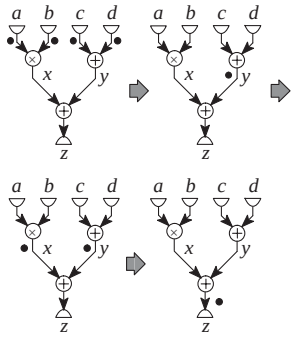
Example:

```
x := a * b;
y := c + d;
z := x + y;
```



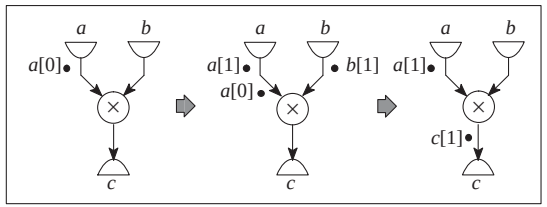
TOKEN FLOW IN A DFG

* 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.

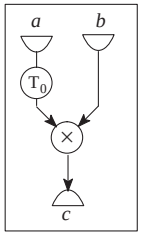


IMPLICIT ITERATIVE DATA FLOW (Ctd.)

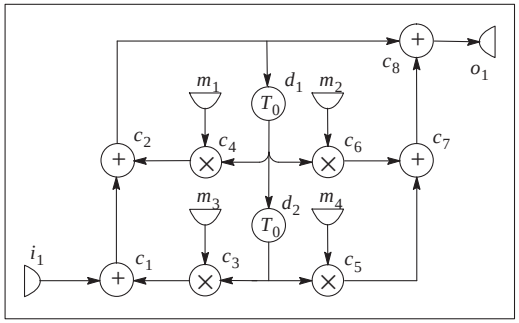
* *Delay elements* instead of initial tokens.

Two notations are encountered:

* explicit delay elements
* delay elements as an edge property



ITERATIVE DFG EXAMPLE



A second-order filter section.

IDFG NOTATION

IDFG(*V,E*) with:

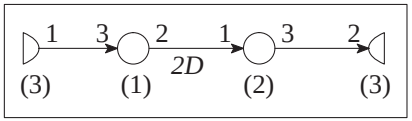
* V: the *vertex* set: $V = C \cup D \cup I \cup O$
* C: set of *computational* nodes
* D: set of *delay* nodes
* I: set of of *input* nodes
* O: set of *output* nodes
* E: the *edge* set
* $\delta(c), c \in C$ gives the duration of a computation (atomic, non-preemptive, restricted library)
* $\mu(d),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



☞ Lee, E.A. and D.G. Messerschmitt, "Synchronous Data Flow", Proceedings of the IEEE, Vol. 75(9), pp 1235–1245, (September 1987).

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.*

TERMINOLOGY

Subtasks:

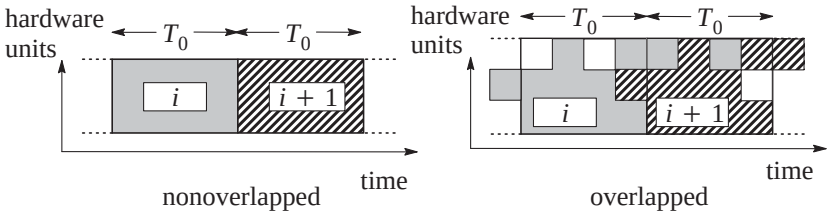
* *Scheduling:* determine for each operation the time at which it should be performed such that no precedence constraint is violated.
* *Allocation:* specify the hardware resources that will be necessary.
* *Assignment:* provide a mapping from each operation to a specific functional unit and from each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

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* (expoiting *interiteration* parallelism) or *nonoverlapped*.



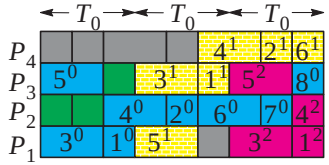
☞ Parhi, K.K. and D.G. Messerschmitt, "Static Rate-Optimal Scheduling of Iterative Data-Flow Programs via Optimum Unfolding", IEEE Transactions on Computers, Vol. 40(2), pp 178–195, (February 1991).

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* λ. In general $\lambda \neq T_0$.
* An ovelapped 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).



A cyclostatic schedule

THE MINIMAL ITERATION PERIOD T_0 ($T_{0_{min}}$)

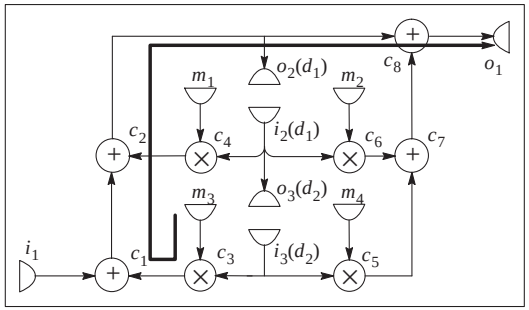
There are four cases:

* Acyclic DFG, nonoverlapped schedule;
* Acyclic DFG, overlapped schedule;
* Cyclic DFG, nonoverlapped schedule;
  + Replace all delay nodes by pairs of input and output nodes.
* Cyclic DFG, overlapped schedule.

For the nonoverlapped cases:

* Compute the *critical path*, longest path from any input to any output, in the acyclic graph. $T_{0_{min}}$ = "length of critical path".

EXAMPLE

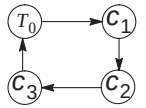


Critical path in a cyclic DFG for a nonoverlapped schedule.

OVERLAPPED SCHEDULING IN A CYCLIC DFG

* Minimal iteration period is given by *critical loop*.

  Example:



$$T_0 \geq \delta(c_1) + \delta(c_2) + \delta(c_3)$$

* 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).

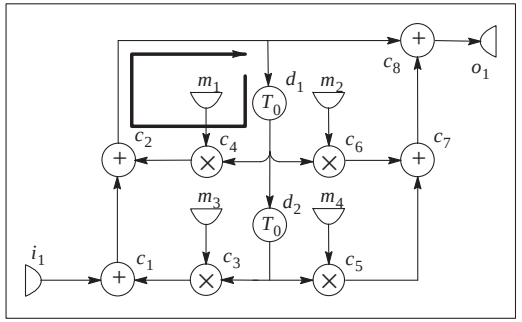
CRITICAL LOOP

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

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

- Reiter, R., "Scheduling Parallel Computations", Journal of the ACM, Vol. 15(4), pp 590–599, (October 1968).
- Renfors, M. and Y. Neuvo, "The Maximum Sampling Rate of Digital Filters Under Hardware Speed Constraints", IEEE Transactions on Circuits and Systems, Vol. CAS–28(3), pp 196–202, (March 1981).

EXAMPLE



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

ON COMPUTING $T_{0_{\min}}$

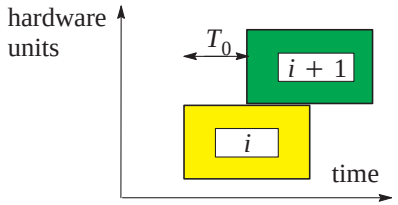
Remarks:

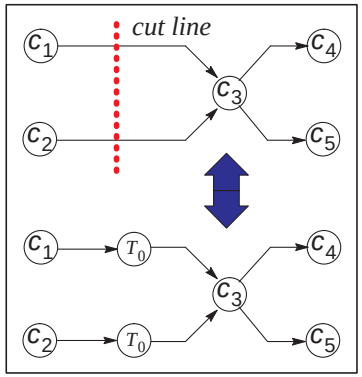
* Direct use of expression for $T_{0_{\min}}$ not efficient (number of loops in graph may grow exponentially with respect to number of nodes).
* Many polynomial-time algorithms have been published; survey in:
  - Dasdan, A., S.S. Irani and R.K. Gupta, "Efficient Algorithms for Optimum Cycle Mean and Optmimum Cost to Time Ratio Problems", 36th Design Automation Conference, (1999).
* An easy to understand but not very efficient method is based on "matrix multiplication".
  - Gerez, S.H., S.M. Heemstra de Groot and O.E. Herrmann, "A Polynomial-Time Algorithm for the Computation of the Iteration-Period Bound in Recursive Data-Flow Graphs", IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, Vol. 39(1), pp 49–52, (January 1992).

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 over-lapped.

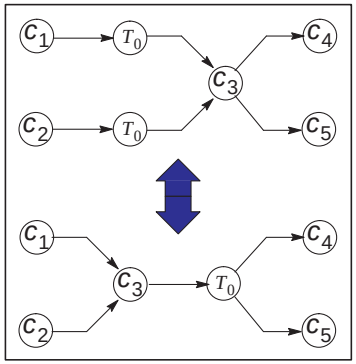




Example

SPEED-UP TECHNIQUES: RETIMING

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



Example

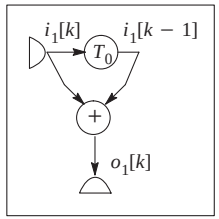
SOME REMARKS ON $T_{0_{\min}}$

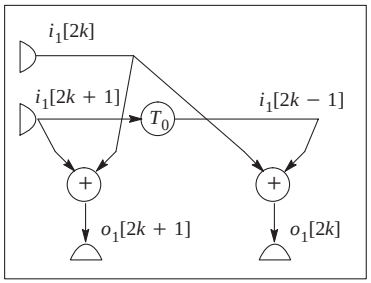
* Retiming does not affect $T_{0_{\min}}$ for overlapped scheduling of IDFG's.
* The T_0 for *nonoverlapped* scheduling obtained after optimal retiming may still be larger than $T_{0_{\min}}$.
* $T_{0_{\min}}$ has been defined as an integer; a fractional $T_{0_{\min}}$ makes sense when *unfolding* is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

☞ Chao, L.F. and E.H.M. Sha, "Rate-Optimal Static Scheduling for DSP Data-Flow Programs", 3rd Great Lakes Symposium on VLSI Design, Automation of High-Performance VLSI Systems, pp 80–84, (March 1993).

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.

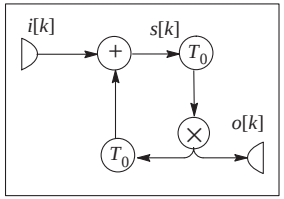




process 2 inputs at a time

UNFOLDING (1)

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



* If $\delta(+) = 1$ and $\delta(*) = 2$, $T_{0_{\min}} = \left\lceil \frac{3}{2} \right\rceil = 2$.
* 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:

$$s[k] = i[k] + o[k-1]$$
$$o[k] = as[k-1]$$

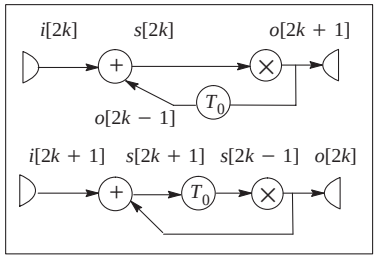
UNFOLDING (2)

* 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.
* The method will be illustrated using the example IDFG and unfolding factor of two, meaning that two inputs will be available per iteration and two outputs will be produced. The equations:

$$s[2k] = i[2k] + o[2k-1]$$
$$s[2k+1] = i[2k+1] + o[2k]$$
$$o[2k] = as[2k-1]$$
$$o[2k+1] = as[2k]$$

UNFOLDING (3)

* The example IDFG after unfolding:

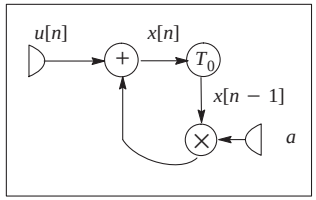


* Note that the unfolded IDFG has two loops with one delay element each and a computational duration of 3. Because a delay element creates an offset of two indices (2 inputs are processed in each iteration), the effective iteration period bound is equal to $T_{0_{\min}} = \frac{3}{2}$.

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 $\delta(*) = 2$, $T_{0_{\min}} = \left\lceil \frac{3}{1} \right\rceil = 3$.

LOOK-AHEAD TRANSFORMATION (2)

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

$$x[n] = a(ax[n-2] + u[n-1]) + u[n]$$
$$x[n] = a^2x[n-2] + au[n-1] + u[n]$$

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