Force Directed Scheduling for Behavioral Synthesis

Paulin & Knight, 1989

IEEE Transactions of Computer Aided Design

Itai M. Pines

EE298-2: Embedded Systems

10/22/98

Behavioral Synthesis & Scheduling

- Automatic synthesis requires several steps:
  - Definition of Circuit Function (HDL)
  - Translation to Graph Based Representation(CDFG)
  - *Operation Scheduling & Hardware Allocation*
    - Partitioning of operations into specific Control-steps
    - Allocation of Functional Units (FU's) to nodes of CDFG
  - Data Path Synthesis
    - Binding operations to resources
  - Gate-Level Structural Synthesis
    - Using predesigned templates or module generators
  - State Graph and Controller

Scheduling Approaches for Synthesis

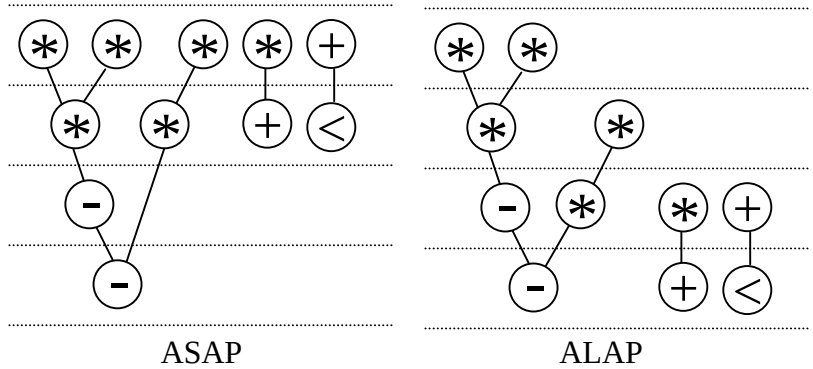
- Independent Scheduling & Allocation
- Interdependent Scheduling
  - Concurrent Scheduling & Allocation
- Scheduling/Allocation by Stepwise Refinement
  - Operations are clustered for max parallelism & unit sharing
  - Scheduling is done to minimize necessary control steps
  - Scheduler continually re-invoked till optimal solution reached
  - Unscheduled operation included in calculations

Force Driven Scheduling

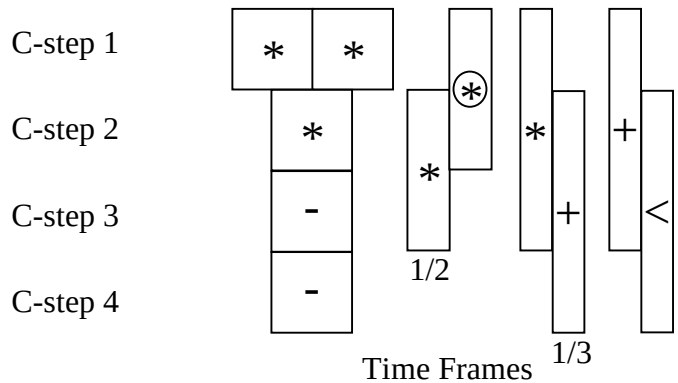
- Goal is to reduce hardware by balancing concurrency
- Iterative algorithm, one operation scheduled per iteration
- Information (i.e. speed & area) fed back into scheduler

Force Driven Scheduling Algorithm

- Determine *ASAP* & *ALAP* Schedules

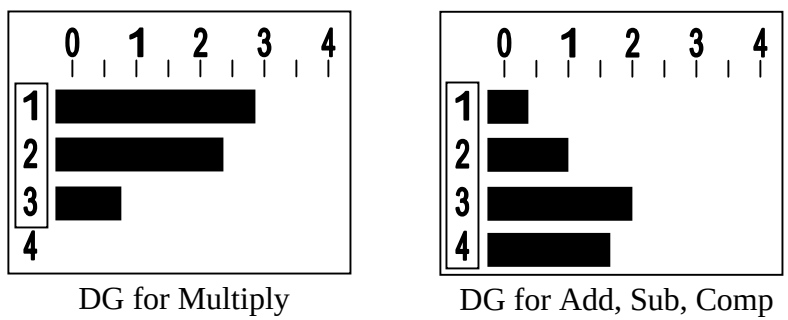


- Determine *Time Frame* of each Op
  - Length of box ~ Possible execution cycles
  - Width of box ~ Probability of assignment
  - Uniform distribution, Area assigned = 1



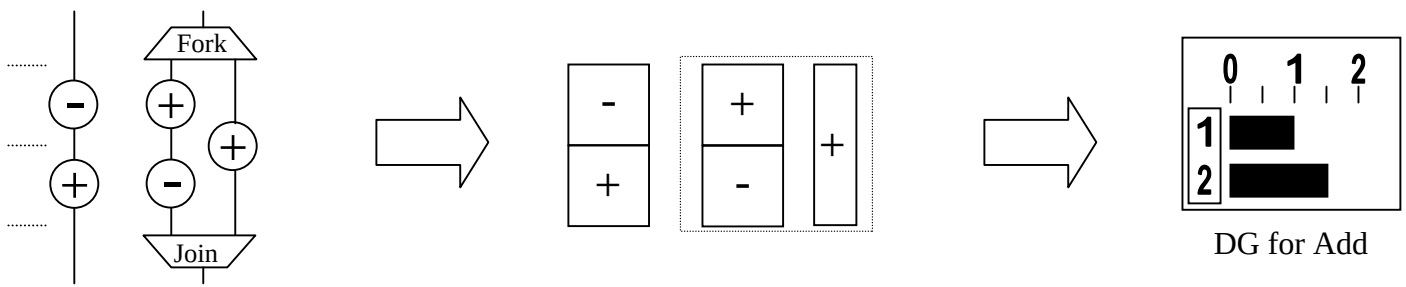
- Create *Distribution Graphs*
  - Sum of probabilities of each Op type
  - Indicates concurrency of similar Ops

$$DG(i) = \Sigma\ Prob(Op, i)$$



Conditional Statements

- Operations in different branches are mutually exclusive
- Operations of same type can be overlapped onto DG
- Probability of most likely operation is added to DG



DG for Add

Self Forces

- Scheduling an operation will effect overall concurrency
- Every operation has “self force” for every C-step of its time frame
- Analogous to the effect of a spring: f = Kx

$$\text{Force}(i) = DG(i) * x(i)$$

DG(i) ~ Current Distribution Graph value

x(i) ~ Change in operation’s probability

$$\text{Self Force}(j) = \sum_{i=t}^{b} [\text{Force}(i)]$$

- Desirable scheduling will have negative self force
  - Will achieve better concurrency (lower potential energy)

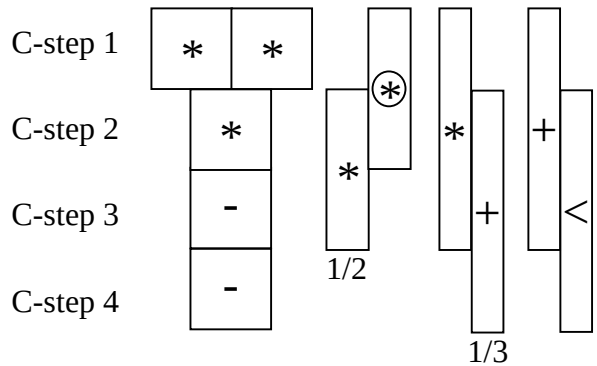
Example

- Attempt to schedule multiply in C-step 1

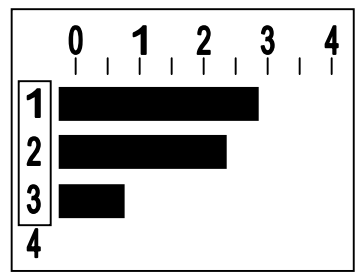
Self Force(1) = Force(1) + Force(2)

= (DG(1) * X(1)) + (DG(2) * X(2))

= [2.833*(0.5) + 2.333 * (-0.5)] = +0.25



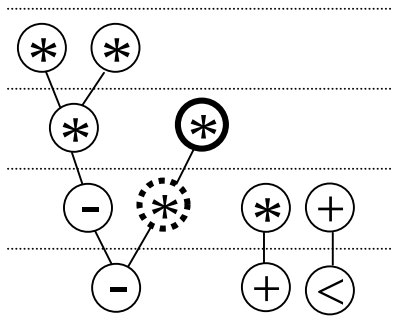
- This is positive, scheduling the multiply in the first C-step would be bad



DG for Multiply

Predecessor & Successor Forces

- Scheduling an operation may affect the time frames of other linked operations
- This may negate the benefits of the desired assignment
- Predecessor/Successor Forces = Sum of Self Forces of any implicitly scheduled operations



Lookahead

•Temporarily modify the constant DG(i) to include the effect of the iteration being considered

$$\text{Force}(i) = \text{temp\_DG}(i) * x(i)$$

$$\text{temp\_DG}(i) = DG(i) + x(i)/3$$

•Consider previous example:

$$\text{Self Force}(1) = (DG(1) + x(1)/3)x(1) + (DG(2) + x(2)/3)x(2)$$

$$= .5(2.833 + .5/3) - .5(2.333 - .5/3) = +.41667$$

•This is even worse than before

Minimization of Bus Costs

- Basic algorithm suitable for narrow class of problems
- Algorithm can be refined to consider “cost” factors
- Number of buses ~ number of concurrent data transfers
- Number of buses = maximum transfers in any C-step
- Create modified DG to include transfers: *Transfer DG*

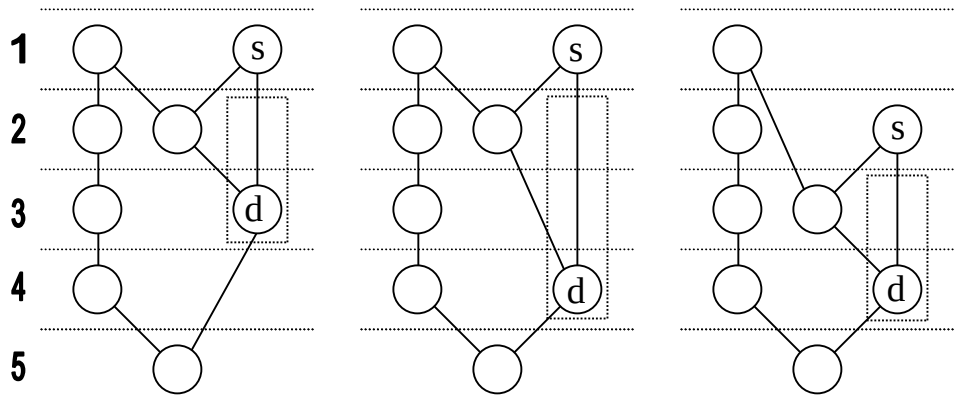
$$\text{Trans DG}(i) = \Sigma\,[\text{Prob}(op,i) * \text{Opn\_No\_InOuts}]$$

Opn_No_InOuts ~ combined distinct in/outputs for Op

- Calculate Force with this DG and add to Self Force

Minimization of Register Costs

- Minimum registers required is given by the largest number of data arcs crossing a C-step boundary
- Create *Storage Operations*, at output of any operation that transfers a value to a destination in a later C-step
- Generate *Storage DG* for these “operations”
- Length of storage operation depends on final schedule



ASAP Lifetime MAX Lifetime ALAP Lifetime

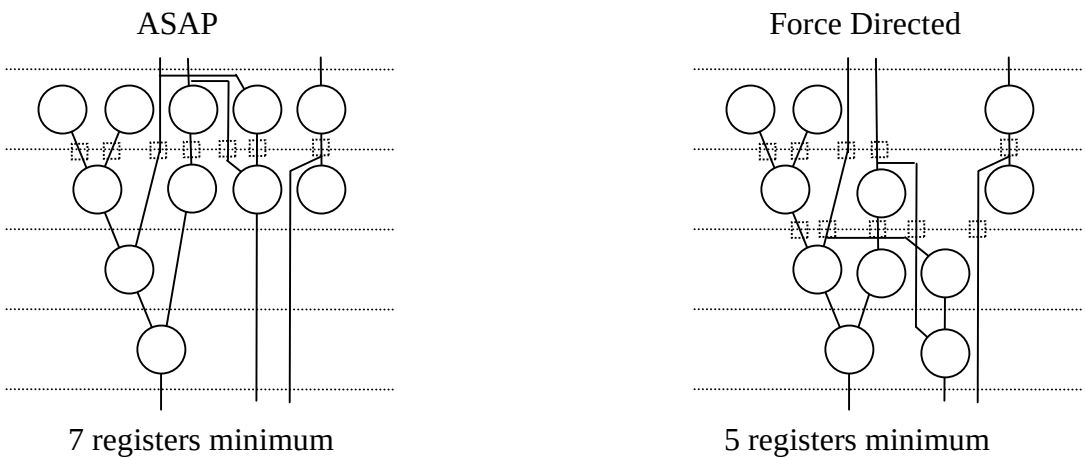
Storage distribution for S

- $[\text{avg life}] = \dfrac{[\text{ASAP life}] + [\text{ALAP life}] + [\text{MAX life}]}{3}$

$$\text{storage DG(i)} = \frac{[\text{avg life}]}{[\text{max life}]}$$ (no overlap between ASAP & ALAP)

$$\text{storage DG(i)} = \frac{[\text{avg life}] - [\text{overlap}]}{[\text{max life}] - [\text{overlap}]}$$ (if overlap)

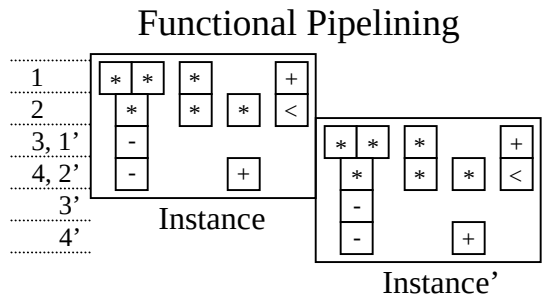
- Calculate and add “Storage” Force to Self Force

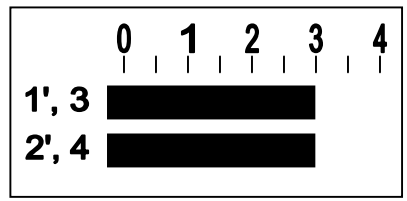


Pipelining

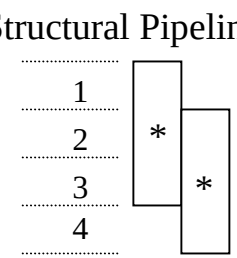
- Functional Pipelining
  - Pipelining across multiple operations
  - Must balance distribution across groups of concurrent C-steps
  - Cut DG horizontally and superimpose
  - Finally perform regular Force Directed Scheduling

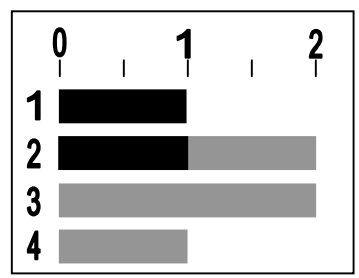
- Structural Pipelining
  - Pipelining within an operation
  - For non data-dependant operations, only the first C-step need be considered





DG for Multiply





Other Optimizations

- Local timing constraints
  - Insert dummy timing operations -> Restricted time frames
- Multiclass FU's
  - Create multiclass DG by summing probabilities of relevant ops
- Multistep/Chained operations.
  - Carry propagation delay information with operation
  - Extend time frames into other C-steps as required
- Hardware constraints
  - Use Force as priority function in list scheduling algorithms