Evolving Architectures for Iterative Dataflow Graphs using Partial Schedules

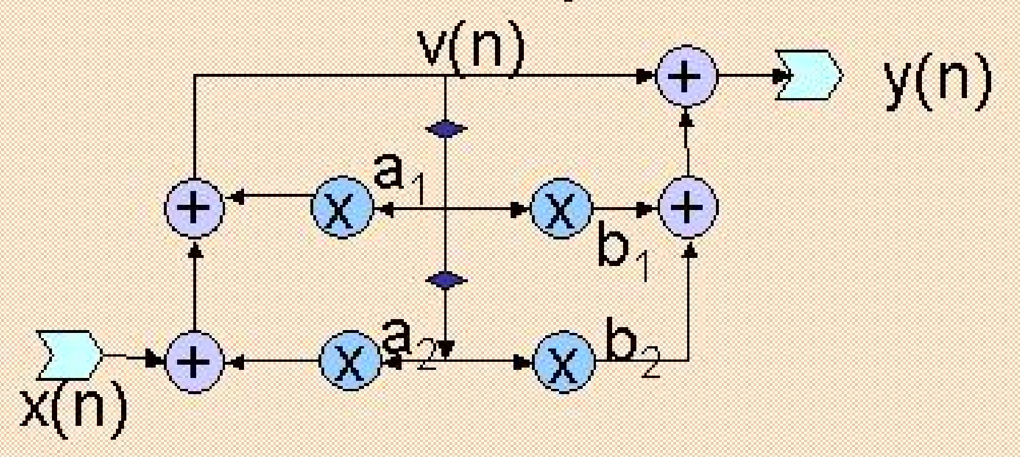
N. Chandrachoodan/S. S. Bhattacharyya, K. J. R. Liu

Synthesizing architectures for DSP Algorithms

- DSP Applications can be represented as dataflow graphs, nodes representing functions and edges representing communication or dependencies
- Resources capable of executing the operations are collected into libraries of useful elements, annotated with information about timing, power consumption, area etc.
- Synthesis problem involves solving the problems of
  - Allocation: Choosing appropriate numbers and types of units to execute all functions
  - Binding: Deciding the mapping of functions to actual physical resources
  - Scheduling: Specifying the order of execution, or time of execution, of each function

Example dataflow graph

Second order filter section: The graph consists of adders and multipliers. Delay elements, represented by blue diamonds, represent the "memory" in the system. Delay elements make the graph "Iterative", and cycles with delays determine the maximum performance achievable from the system.



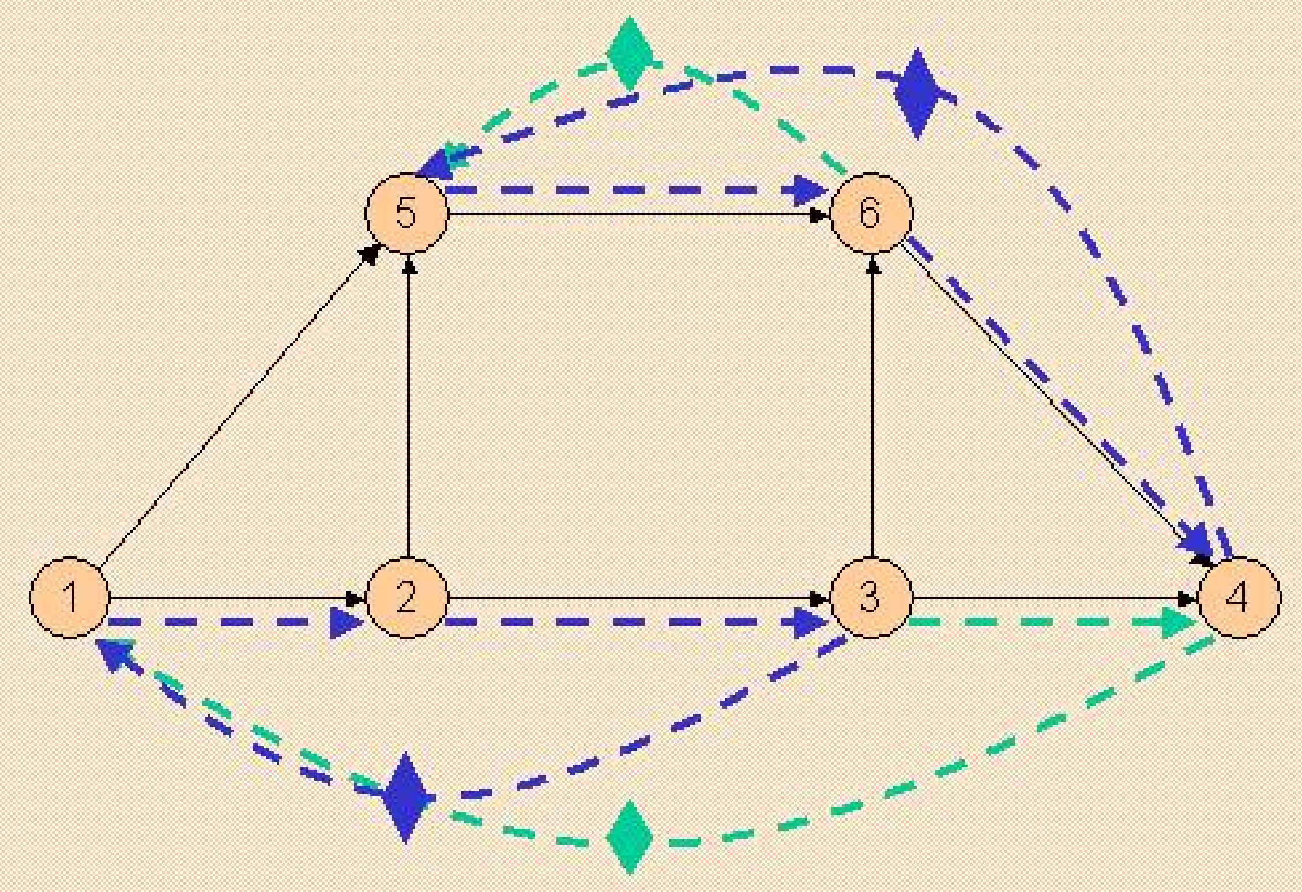
Scheduling Techniques

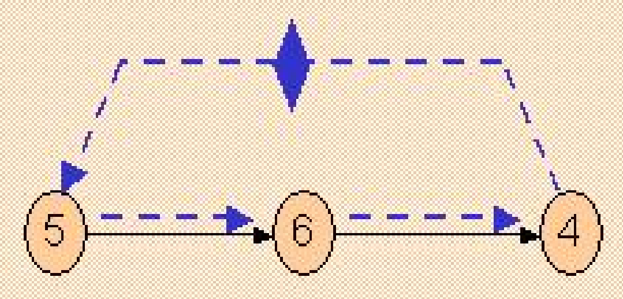
- NP-Complete problem
- Exponential growth of design space, full search generally not possible
- Priority list based methods, force-directed scheduling, Range-charts, other heuristics
- Non-iterative graphs (limited exploitation of inter-iteration parallelism)
- Deterministic *vs.* Probabilistic methods
- Simulated annealing, Tabu search, Evolutionary methods

Evolutionary approaches for Synthesis

- Need to evolve overall solution: architecture as well as scheduling order
- Evolving sequences not as intuitive as normal Gene-based evolution
- Graph dependencies often result in invalid solutions
- Chromosome repair techniques may be used:
  - Extra computational effort
  - Redundancy in representation
  - May introduce bias by increasing number of representations for some solutions
- Standard genetic crossover and mutation operators may not work: need to define custom techniques
- **Solution:** Use encoding that understands structure of dataflow graph – cannot result in invalid representations, more efficient, can use known techniques to construct good solutions

Partial Schedules





Partial schedule Specifies:
a) Processor type (adder, multiplier etc.)
b) Order of execution of nodes on processor

Search methods with partial schedules

- Exhaustive search over a finite population: does not cover entire search space, but can make good use of well-designed partial schedules
- Evolution of population of partial schedules possible

Results using Evolutionary approach

- Population size of 20 (initially randomly generated)
- 100 generations
- as population evolves, more valid solutions, tends to slow down
- evolution conditions:
  - best solution always retained
  - several new random partial schedules
  - mutation of existing schedules

Mutation rate	Average Iter. Period	Average Power	Runtime (s)
0.00	986.80	13493.53	24.33
0.05	963.08	13900.30	12.06
0.10	979.78	13993.30	11.20
0.25	911.55	13900.90	6.52
0.75	954.20	16149.40	3.05