Minimizing Buffer Requirements for Throughput Constrained Parallel Execution of Synchronous Dataflow Graph

Seoul National University, CAPLABT

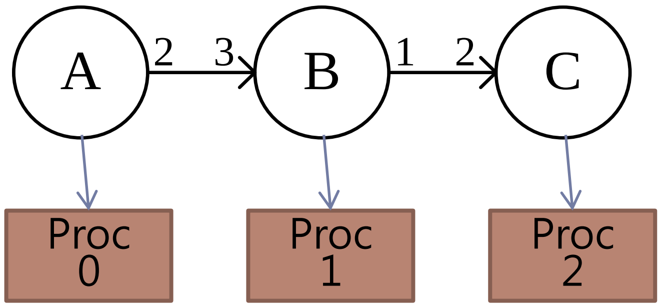
Tae-ho Shin[1], Hyunok Oh[2] and **Soonhoi Ha[1]**

1: Seoul National University
2: Hanyang University

Contents

Motivational Example

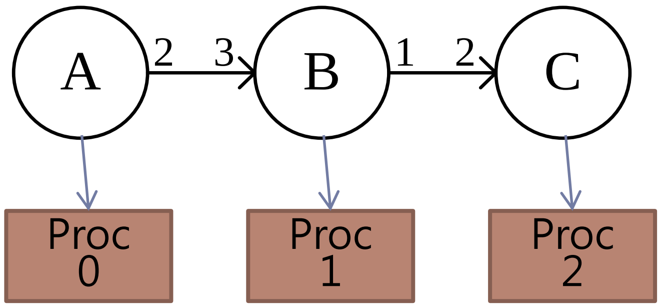
- A (Simple) SDF Graph
  - node: computation block
  - arc: FIFO queue
  - Sample rate: number of samples consumed or produced per node firing
  - A node is fireable only after it has enough number of samples on all input arcs



- A mapping instance (nodes to processors)

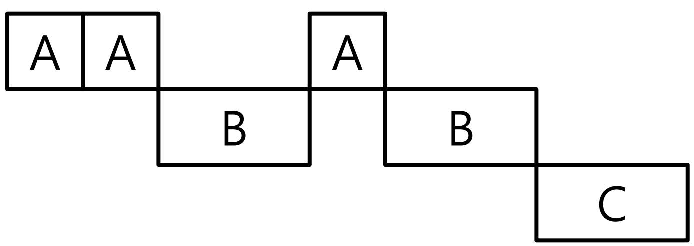
Node	A	B	C
Mapped Processor	1	2	3
Execution Time	1	2	2

Arc buffer size affects the throughput!

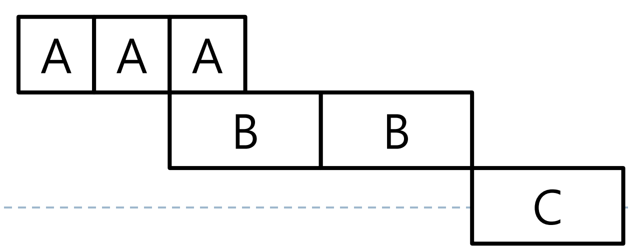


Node	A	B	C
Mapped Processor	1	2	3
Execution Time	1	2	2

- Scheduling result when the buffer size of arc AB is 4

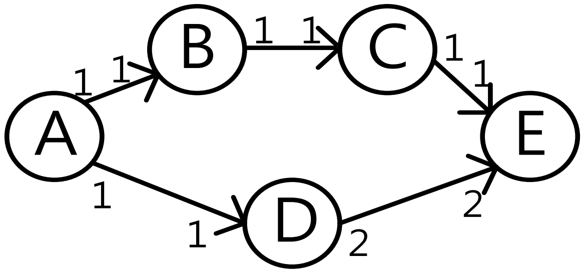


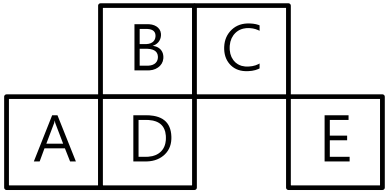
- Scheduling result when the buffer size of arc AB is 6



Unfolding affects the throughput!

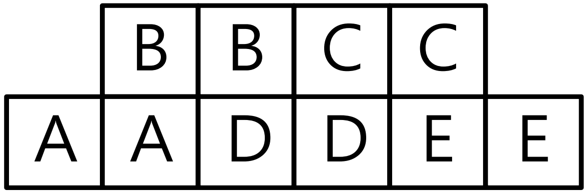
- Motivational Example 2





<Scheduling result without unfolding>

Node	A	B	C	D	E
Mapped Processor	2	1	1	2	2
Execution Time	1	1	1	1	1



<Scheduling result with 2-unfolding>

Related Work

- Related Work

<table>
<tr><th>Scheduling Policy</th><th>Fixed Number of Processors</th><th>Unlimited Number of Processors</th></tr>
<tr><td rowspan="2">Static scheduling</td><td rowspan="2">Pipeline, max-plus, model checking, scenario based, etc.</td><td>Without unfolding</td></tr>
<tr><td>With unfolding</td></tr>
<tr><td>Dynamic Scheduling</td><td>Proposed Method</td><td>N/A</td></tr>
</table>

- All previous work assumed “static scheduling”
- The optimization problem is NP-hard
- Extensive work has been performed recently – prove that the problem becomes practically important

Dynamic vs Static scheduling

- Pros of dynamic scheduling over static scheduling
  - Can get the effect of unfolding naturally
  - Easy to represent of schedule and uses less memory space
  - May improve system performance when the execution times are vary at run-time

- But we need
  - Run-time system to schedule the nodes dynamically
  - Priority assignment to the mapped nodes

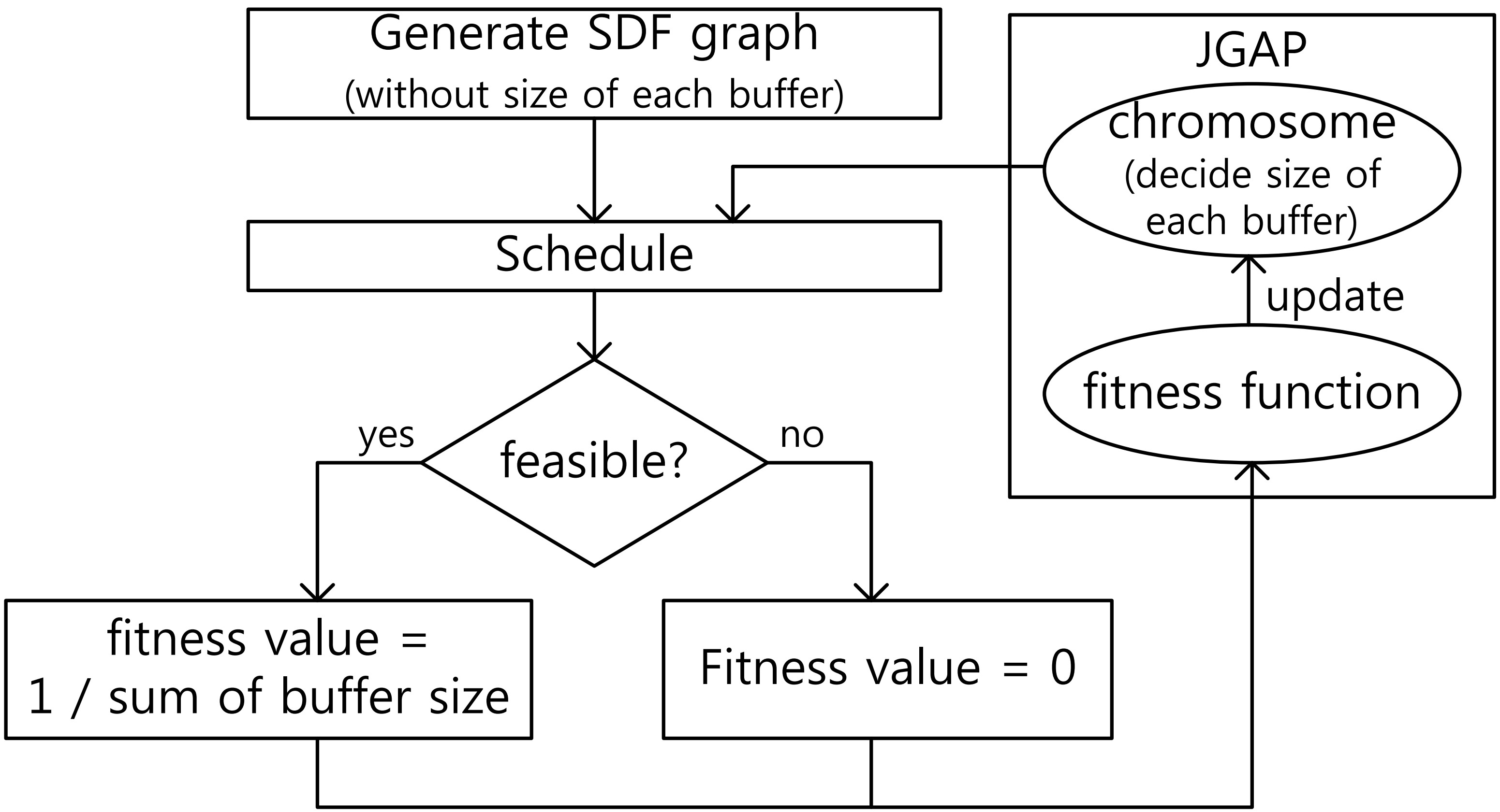
Problem Definition

- **Input**
  - Target Architecture: A heterogeneous MPSoC
  - Input Information
    - An SDF graph with given execution time of nodes
    - A given static mapping of nodes to processors
    - A known dynamic scheduling policy on each processor
  - Constraints: Throughput

- **Problem**
  - Minimize the total buffer requirement and determine the buffer size of all arcs
  - (Determine the priority of the mapped nodes)

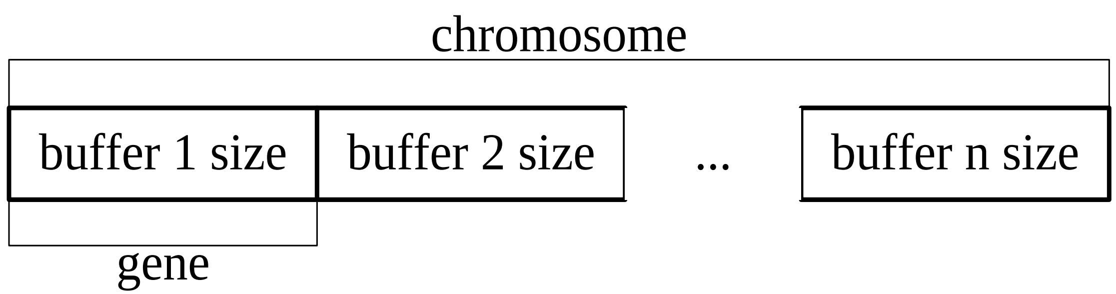
Proposed Solution

- Overall Optimization Flow



GA-based Heuristic

- JGAP package is used for current implementation
  - The size of each buffer size is encoded into chromosome and GA evaluate chromosome by scheduling dynamically with encoded buffer size information



  - Fitness value of chromosome is determined by feasibility of scheduling result based on given throughput constraint
  - Optimization process is repeated until fitness value converges or pre-defined upper bound of generation steps

Feasibility Analysis

- Simulate the system in which each processor performs dynamic scheduling of the mapped nodes for each candidate solution (given buffer sizes of all arcs)
  - All mapped nodes are assigned priorities
  - We consider the communication overhead between processors as well as execution time variation of the nodes
  - We repeat the execution of the graph until we obtain the throughput

Throughput Computation

- Approximate throughput
  - Since there is no guarantee that the same scheduling pattern will be repeated in dynamic scheduling, the following equation is defined to calculate throughput in dynamic scheduling

$$T(G) = \lim_{n \to \infty} \frac{n}{\text{time to finish n interations}}$$

  - If the number of iterations are increased to infinite, the value of equation converges to specific value and it can be considered as throughput
  - In most case, **after 10 iterations** the value converges

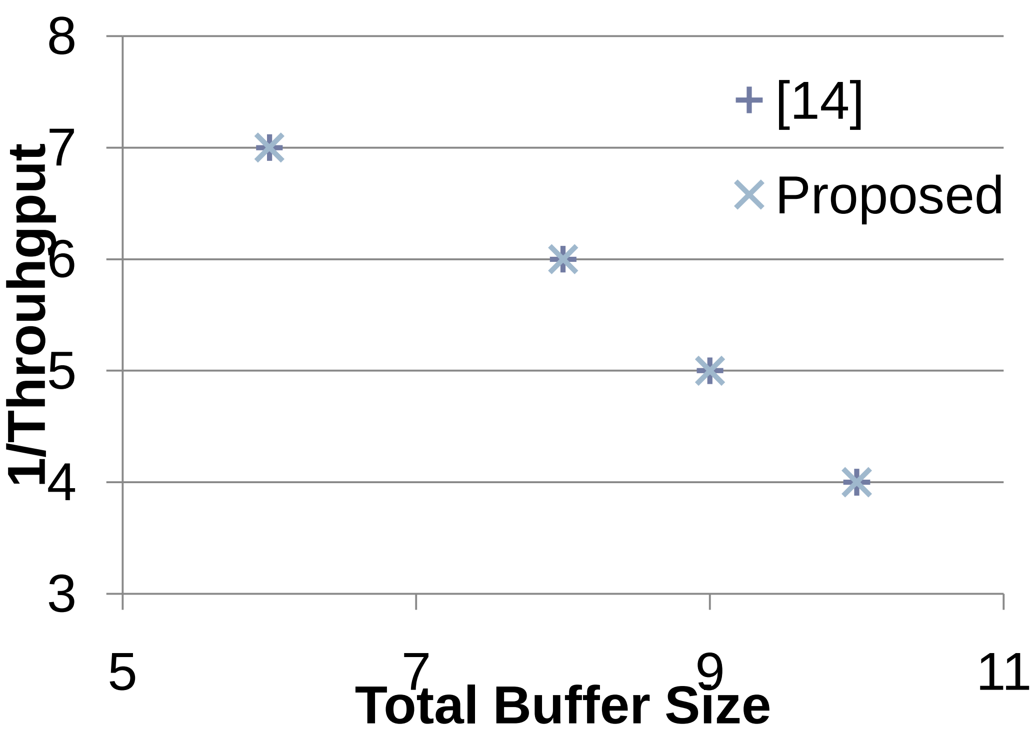
Priority Assignment

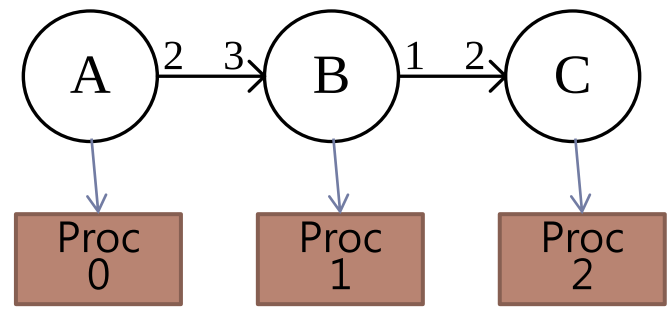
- Proposed heuristic
  - We assign a different priority of each invocation for a same node
  - To set priority to each node invocation, calculate "as late as possible(ALAP)" scheduling time to sink node as following
    - $P(N_{last}) = Ex(N) + \max\{P(K_1)\}$

      where node K is in {successors of node N}
    - $P(N_k) = P(N_{last}) + (rep(N) - k) * Ex(N)$
  - Optimal assignment is left as a future work

Experimental Results

- Comparison of total buffer size with an optimal solution in [14]





Node	A	B	C
Mapped Processor	1	2	3
Execution Time	1	2	2

Comparison with a pipelined method

- Pipelining is a popular way of throughput improvement
- But pipelining needs pipeline buffers.
- Paper [11] finds an sub-optimal pipelining for an SDF graph without considering unfolding

	Throughput	Total buffer size
[11]	1/3	8
Proposed Method	1/3	6

Scalability of the proposed technique

- Elapsed time with various input sets

# of instances	# of processors	# of edges	Throughput constraints	Elapsed time
30	3	5	1 / 100	190 s
			1 / 44	192 s
		32	1 / 100	134 s
			1 / 34	133 s
100	7	20	1 / 100	1052 s
			1 / 75	1059 s
		54	1 / 100	588 s
			1 / 79	665 s

Conclusion

- We propose a static mapping and dynamic scheduling method that has several benefits over static scheduling methods.
- The proposed GA_based algorithm minimizes the buffer requirement under the throughput constraints.
- A simple heuristic for priority assignment is also proposed – produces good results
- The proposed technique is scalable, while producing near-optimal results.

Future work

- Find an optimal mapping
- Find an optimal priority assignment scheme

Thank you!