ECE382M.20: System-on-Chip (SoC) Design

Lecture 11 – Advanced HLS Techniques

Source: Z. Zhang, Cornell Univ.

Andreas Gerstlauer
Electrical and Computer Engineering
The University of Texas at Austin
gerstl@ece.utexas.edu

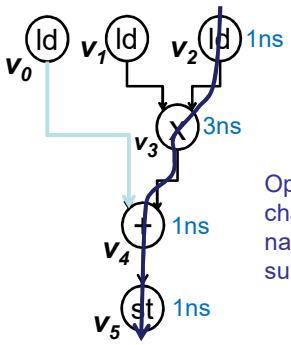
The University of Texas at Austin
Chandra Department of Electrical and Computer Engineering
Cockrell School of Engineering

Lecture 11: Outline

- **Advanced scheduling approaches**
  - SDC-based scheduling
  - Modulo scheduling
- **HLS design space exploration (DSE)**
  - Multi-objective optimization
- **HLS outlook**
  - Machine learning for HLS

SDC-Based Scheduling

- **SDC = System of difference constraints**



Operation chaining is naturally supported

s_i : schedule variable for operation i

- Dependence constraints
  - $<v_0, v_4> : s_0 - s_4 \le 0$
  - $<v_1, v_3> : s_1 - s_3 \le 0$
  - $<v_2, v_3> : s_2 - s_3 \le 0$
  - $<v_3, v_4> : s_3 - s_4 \le 0$
  - $<v_4, v_5> : s_4 - s_5 \le 0$
- Cycle time constraints
  - $v_1 \rightarrow v_5 : s_1 - s_5 \le -1$
  - $v_2 \rightarrow v_5 : s_2 - s_5 \le -1$

Timing constraints

- Target cycle time: 5ns
- Delay estimates
  - Mul (x): 3ns
  - Add (+): 1ns
  - Load/Store (ld/st): 1ns

To meet the cycle time, v_2 and v_5 should have a minimum separation of one cycle

[J. Cong & Z. Zhang, DAC, 2006] [Z. Zhang & B. Liu, ICCAD, 2013]

Difference Constraints

- **A difference constraint is a formula in the form of $x - y \le b$ or $x - y < b$ for numeric variables x and y, and constant b**
- **With scheduling variables, we use integer difference constraints to model a variety of scheduling constraints**
  - x and y must have integral values
    - Thus b only needs to be an integer => form $x - y < b$ is redundant

SDC Constraint Matrix

- **Constraint matrix of SDC(*X*, *C*) is a totally unimodular matrix (TUM)**
  - Every nonsingular square submatrix has a determinant of -1/+1

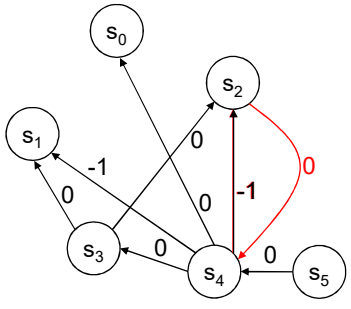
$$\begin{pmatrix} 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & \\ 0 & 0 & 1 & -1 & 0 & 0 & \\ 0 & 0 & 0 & 1 & -1 & -1 & \\ 0 & 0 & 0 & 0 & 1 & -1 & \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & & & \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \end{pmatrix} \leq \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \\ -1 \end{pmatrix}$$

A x b

  - Theorem (Hoffman & Kruskal, 1956): If *A* is totally unimodular and *b* is a vector of integers, every extreme point of polyhedron $\{x : Ax \leq b\}$ is integral.
  - ➢ **Solving linear programming (LP) relaxation leads to integral solutions**

SDC Constraint Graph

- **Difference constraints can be conveniently represented using constraint graph**
  - Each vertex represents a variable, and each weighted edge corresponds to a different constraint
  - Detect infeasibility by the presence of negative cycle (by solving single-source shortest path)

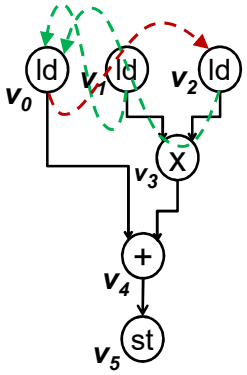


$s_2 - s_4 \leq -1$
$s_4 - s_2 \leq 0$
$0 \leq -1$

$s_0 - s_4 \leq 0$
$s_1 - s_3 \leq 0$
$s_2 - s_3 \leq 0$
$s_3 - s_4 \leq 0$
$s_4 - s_5 \leq 0$
$s_2 - s_4 \leq -1$
$s_1 - s_4 \leq -1$
$s_4 - s_2 \leq 0$

Handling Resource Constraints

- **Resource constraints cannot be represented exactly in integer difference form**



- Resource constraint
  - Two read ports

- Resource constraints
  - ➔ Heuristic partial orderings

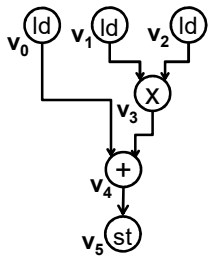
$v_0 \rightarrow v_2 : s_0 - s_2 \leq -1$ 3 cycle latency

OR

$v_1 \rightarrow v_0 : s_1 - s_0 \leq -1$
$v_2 \rightarrow v_0 : s_2 - s_0 \leq -1$ 2 cycle latency

Linear Objectives

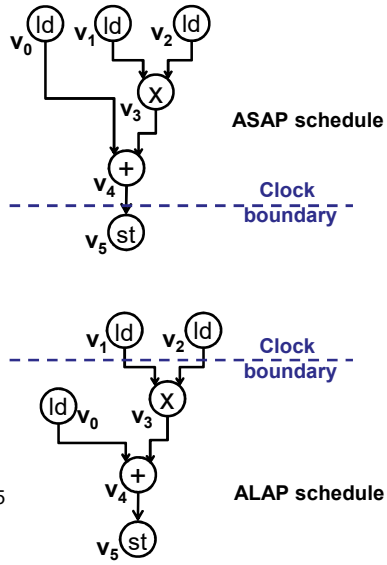
- **ASAP: min $\sum_{i \in V} s_i$**
- **ALAP: max $\sum_{i \in V} s_i$**
- **Minimum latency: min $\max_{i \in V} \{s_i\}$**
- **Minimum average case latency (control-intensive design)**
- **Many other …**



- Target cycle time: 5ns
- Delay estimates
  - Mul (x): 3ns
  - Add (+): 1ns
  - Load/Store (ld/st): 1ns

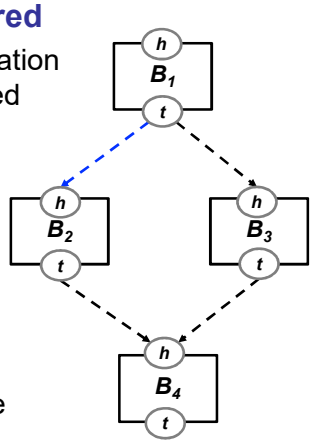
min s_0 + … + s_5

max s_0 + … + s_5



Control Flow Graphs

- **Control dependencies can also be honored**
  - If bb_2 is control dependent on bb_1, the operation nodes of bb_2 are not allowed to be scheduled before those of bb_1
  - Polarize each basic block bb_i with two scheduling variables (head and tail)
    - $\forall v \in bb_i$, $s_h(bb_i) - s_h(v) \leq 0$
    - $\forall v \in bb_i$, $s_t(v) - s_t(bb_i) \leq 0$
  - If $e_c(bb_i, bb_j) \in E_c$ and e_c is not a back edge
    - $s_t(bb_i) - s_h(bb_j) \leq 0$



$s_t(B_1) - s_h(B_2) \leq 0$

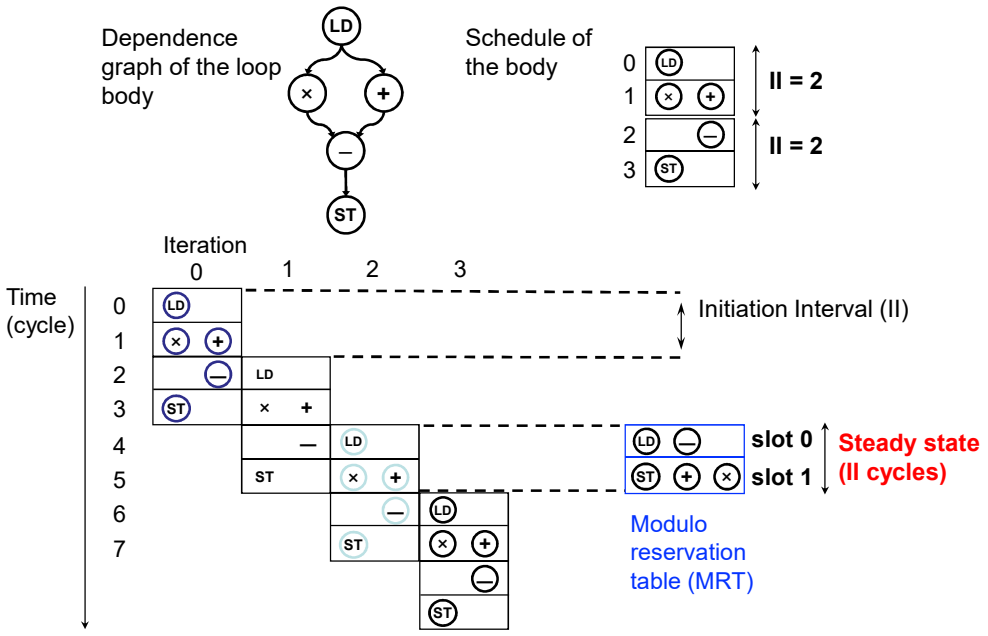
Lecture 11: Outline

- **Advanced scheduling approaches**
  - ✓ SDC-based scheduling
  - Modulo scheduling
- **HLS design space exploration (DSE)**
  - Multi-objective optimization
- **HLS outlook**
  - Machine learning for HLS

Modulo Scheduling

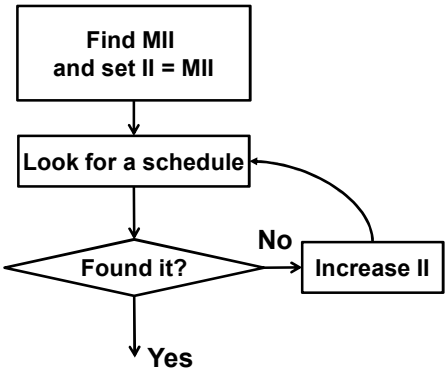
- **A regular form of loop (or function) pipelining technique**
  - Also applies to software pipelining in compiler optimization
  - Loop iterations use the same schedule, which are initiated at a constant rate
  - Typical objective: minimize II under resource constraints
    - NP-hard in general
    - Optimal polynomial time solution exists without recurrences or resource constraints
- **Advantages of modulo scheduling**
  - Cost efficient: No code or hardware replication
  - Easy to analyze
    - Steady state determines performance & resource

Modulo Scheduling Example



Algorithm for Modulo Scheduling

- **Heuristic algorithm**
  - Find a lower bound on II
  - Look for a schedule with the given II
  - If a feasible schedule not found, increase II and try again

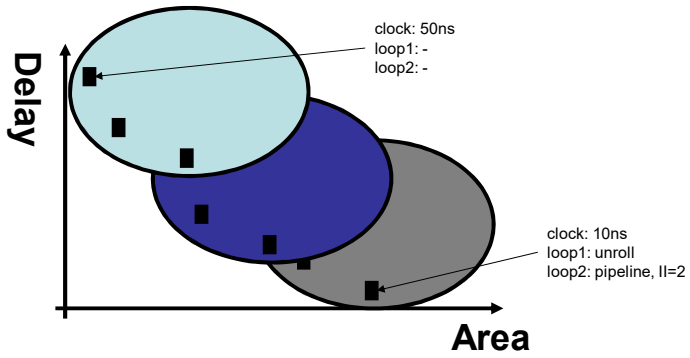


Lecture 11: Outline

- **Advanced scheduling approaches**
  - SDC-based scheduling
  - Modulo scheduling
- **HLS design space exploration (DSE)**
  - Multi-objective optimization
- **HLS outlook**
  - Machine learning for HLS

Multi-Objective Design Space

- **Design space enabled by manual synthesis directives**
  - Local pragma settings, global constraints & options
  - Automatic exploration not handled by traditional algorithms

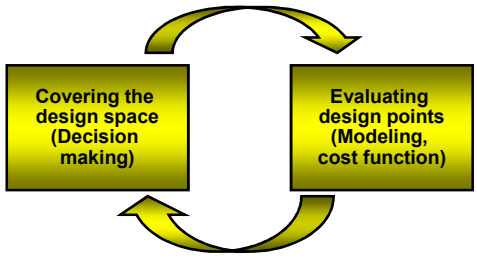


- **Multi-objective optimization (MOO)**
  - Pareto optimality, find Pareto front

Source: J. Abraham

Design Space Exploration (DSE)

- **Design space exploration**



- **Automatic DSE (Auto-DSE)**
  - Automatic pragma, option & configuration setting
  - Synthesis area & delay result estimation

Source: C. Haubelt

Automatic Decision Making & Exploration

- **Brute-force methods**
  - Exhaustive search
  - Random search
- **Meta-heuristics**
  - Simulated annealing
  - Genetic algorithms
  - …
- **Dedicated heuristics**
  - Divide & conquer
  - Clustering
  - …

Source: B. Carrion-Schaefer

Synthesis Result Estimation

- **Synthesis-based**
  - Run design through HLS tool
  - Use estimated HLS result metrics (synthesis report)
- **Model-based**
  - Sample the design space
  - Learn a model that predicts synthesis results
  - Potentially even predict (parts of) the design space

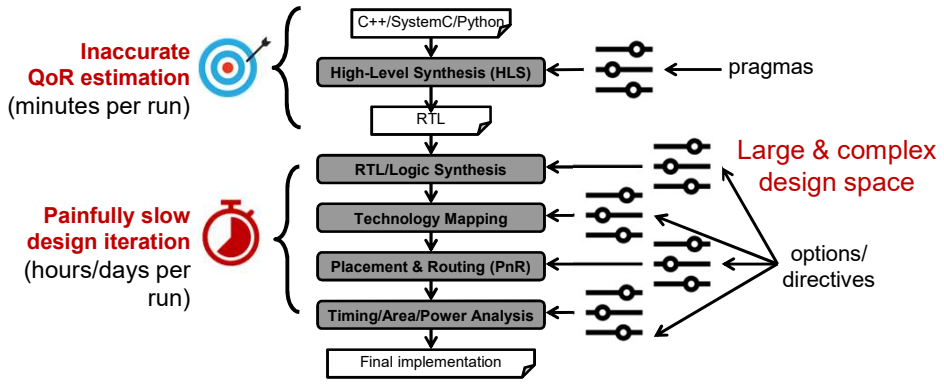
Source: B. Carrion-Schaefer

Lecture 11: Outline

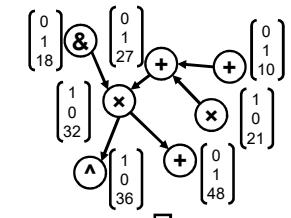
- ✓ Advanced scheduling approaches
  - ✓ SDC-based scheduling
  - ✓ Modulo scheduling
- ✓ HLS design space exploration (DSE)
  - ✓ Multi-objective optimization
- **HLS outlook**
  - Machine learning for HLS

Hurdles in Agile Hardware Design

- **Tension between speed and fidelity w/ HLS-based design**



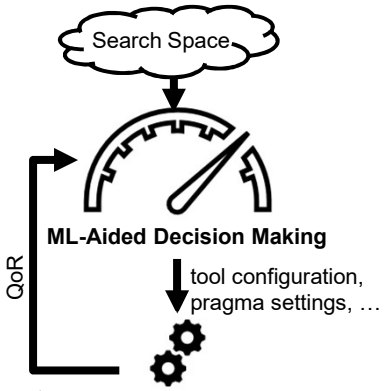
Machine Learning (ML) in HLS



ML-Based Function Approximation

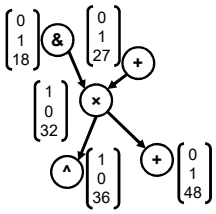
Predicted area, delay, power

① **Faster & More Accurate QoR Inference**



② **More Intelligent Design Space Exploration (DSE)**

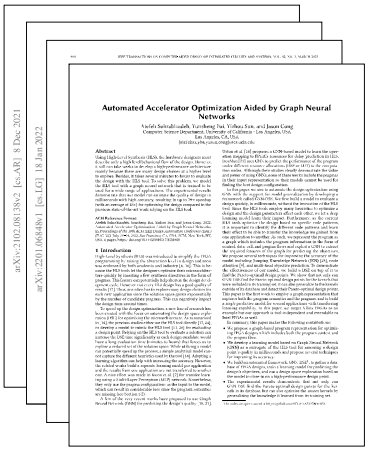
ML for HLS QoR Inference



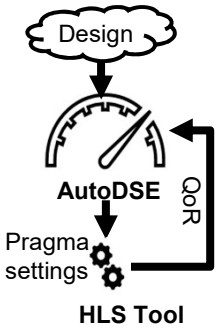
ML-Based Function Approximation

Predicted QoR

ML Model	Estimation Targets	Reference
GNN	Post-HLS Resource & Latency	GNN4HLS [Ferretti et al. TODAES'22]
GNN	Post-HLS Resource & Latency	GNN-DSE [Sohrabizadeh et al. DAC'22]
GNN	Post-PnR Resource & Timing	IronMan [Wu et al. GLSVLSI'21]
GNN	Post-PnR Resource & Timing	– [Wu et al. DAC'22]
GNN	Post-PnR Resource & Timing	IronMan-Pro [Wu et al. TCAD'23]
GNN	Post-PnR Power	PowerGear [Lin et al. DATE'22]
CNN	Post-PnR Power	HL-Pow [Lin et al. TCAD'23]



ML for HLS DSE



ML Model	Model Inputs	Reference
Gaussian Processes	HLS Pragmas	Prospector [Mehrabi et al. DATE'22]
Gaussian Processes	HLS Pragmas	– [Sun et al. TODAES'22]
CNN	HLS Pragmas	HL-Pow [Lin et al. TCAD'23]
GNN	HLS Pragmas + CDFGs	GNN4HLS [Ferretti et al. TODAES'22]
GNN	HLS Pragmas + CDFGs	GNN-DSE [Sohrabizadeh et al. DAC'22]
GNN + RL	HLS Pragmas + DFGs	IronMan [Wu et al. GLSVLSI'21]
GNN + RL	HLS Pragmas + DFGs	IronMan-Pro [Wu et al. TCAD'23]
LLM + GNN	HLS Source + Pragmas + CDFG	ProgSG [Bai et al. ArXiv'23]


IRONMAN-PRO: Multiobjective Design Space Exploration in HLS via Reinforcement Learning and Graph Neural Network-Based Modeling

HLS Estimation Accuracy

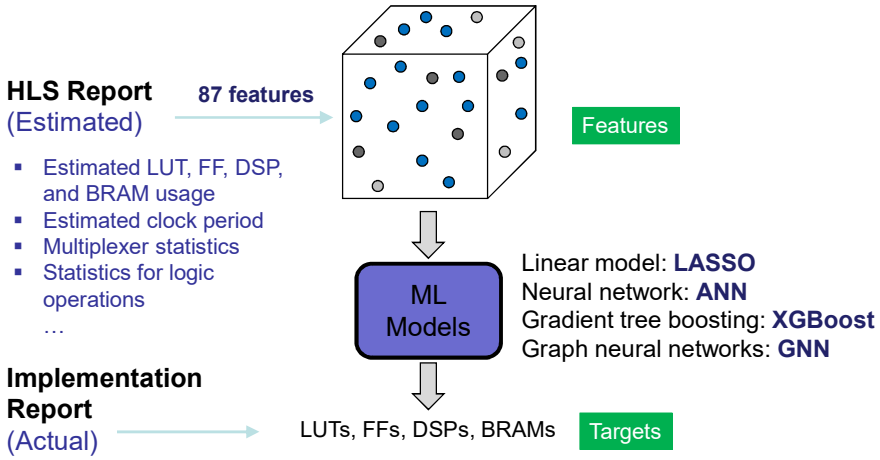
- **65 individual designs**
  - CHStone [Hara et.al, JIP'08]
  - Machsuite [Reagen et.al, IISWC'14]
  - S2CBench [Schafer & Mahapatra, ESL'14]
  - Rosetta [Zhou et.al, FPGA'18]
- **1300 data samples**
  - Clock periods: 1, 2, 3, 5, 10ns
  - Zynq, Artix, Kintex, and Virtex devices

Commercial HLS
(Post-HLS estimates)
- Resource usage
- Timing
- Performance
- HDL details, etc.

Implementation
(Post-PnR results)
- Resource usage
- Timing, etc.

Estimation Error
LUT: mean 2~4x
FF: mean 2~3x

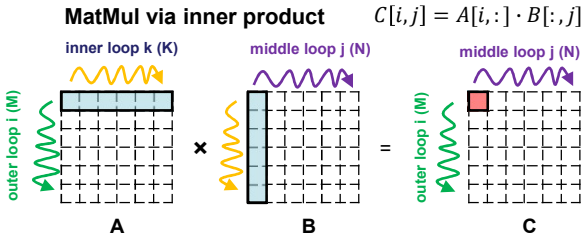
ML-Based Resource Estimation in HLS



[1] S. Dai, Y. Zhou, H. Zhang, E. Ustun, E. F.Y. Young, and Z. Zhang, Fast and Accurate Estimation of Quality of Results in High-Level Synthesis with Machine Learning, FCCM, May 2018. (Best Paper Award, Short Paper Category)

Challenges to Effective DSE

- **Creating a high-performance HLS design require nontrivial *manual* source-level transformations that alters the loop structures and data layout**
- **Example: Inefficient matrix multiplication using inner product**
  - Slow pipeline due to carried dependency from floating-point accumulation

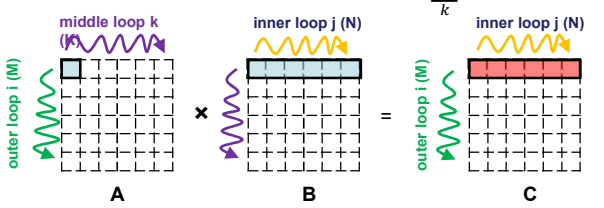



```
for (int i = 0; i < M; i++)
  for (int j = 0; j < N; j++)
    C[i, j] = 0;
    for (int k = 0; k < K; k++)
      C[i, j] += A[i, k] * B[k, j]
```

Challenges to Effective DSE

- **A faster approach using row-wise product**
  - Different loop orders & additional on-chip storage
  - ➢ Existing HLS AutoDSE methods cannot achieve this solution through permutations of the pragma settings

MatMul via row-wise product $C[i,:] = \sum_k A[i,k] \cdot B[k,:]$




```
for (int i = 0; i < M; i++) {
  float C_vec[N];
  for (int j = 0; j < N; j++)
    C_vec[j] = 0.0;

  for (int k = 0; k < K; k++)
    for (int j = 0; j < N; j++)
      #pragma pipeline II=1
      C_vec[j] += A[i, k] * B[k, j];

  for (int j = 0; j < N; j++)
    C[i, j] = C_vec[j];
}
```

New opportunities in co-designing HLS programming models and ML-based DSE to enable exploration of high-level (initially algebraic) transformations

Lecture 11: Summary

- **HLS is (again) an active research area**
  - Golden age for hardware specialization
    - End of traditional semiconductor scaling
  - Application demands
    - E.g. machine learning
  - "Programmable" hardware substrates
    - FPGAs
- **Advanced HLS techniques**
  - Scheduling generalizations and optimality
  - Automatic design space exploration (Auto-DSE)
  - Machine learning for HLS