A Gradual Scheduling Framework for Problem Size Reduction and Cross Basic Block Parallelism Exploitation in High-level Synthesis

Hongbin Zheng
Q. Liu, J. Li, D. Chen, Z. Wang
School of Physics and Engineering
Sun Yat-sen University, P. R. China
High-level Synthesis

• From high-level language:
  – C, C++, C#, Java

• Scheduling and binding

• Generate hardware description

HLL
Scheduling
Binding
Hardware Description
Control-Data Flow-Graph

Nodes represent operations, e.g. addition, load/store, goto

Edges represent dependencies
- Control-flow dependencies
- Dataflow dependencies

Weights represent latency constraints between operations

Nodes: A, B, C, D, E, F

Weights: 5, 5, 2, 2, 8, 3
Scheduling in High-level Synthesis

C-step 1

C-step 2

C-step 3

C-step 4

ASP-DAC 2013, Pacífico Yokohama, Yokohama, Japan
Scheduling in High-level Synthesis

• Have big impact on synthesis quality
  – Speed performance
  – Resource usage
  – Energy consumption

• Time consuming!
Critical v.s. Non-critical

64-bit shifter (393 LEs)

8-bit bitwise-and (8 LEs)
Gradual Scheduling Framework

- Schedule Critical-Op
- Schedule Trivial-Op

- HLL
- Scheduling
- Binding
- Hardware Description
Gradual Scheduling Framework

Aggressive yet time-consuming algorithm

Schedule Critical-Op

Optimize for:
- Speed
- Area
Exploit parallelism

Schedule Trivial-Op
Gradual Scheduling Framework

Aggressive yet time-consuming algorithm

Simple and fast algorithm

Schedule Critical-Op

Optimize for:
- Speed
- Area
- Exploit parallelism

Schedule Trivial-Op

Only need valid schedules
Evaluation Metrics

• Scheduling problem size reduction
  – The size of the critical part and the noncritical part

• Latency reduction, exploiting global parallelism
  – In terms of number of cycles
BACKGROUND
Scheduling for the Best QoR

Maximize speed
Minimize energy consumption
Maximize FU sharing opportunities

Scheduling

CDFG

N Iteration
Related Work

• Force-directed scheduling
  – Balance resource usage

• Path-based scheduling, HCDG-based scheduling
  – Identify mutual exclusive operations for parallelism
    and FU sharing

• Global Code Motion, Hyper-block Formation
  – Exploit global parallelism

• SDC scheduling
  – Optimize the latency for the whole design
  – Use soft-constraints to model other design goals
Related Work

• Force-directed scheduling
  – Balance resource usage

Schedule the critical and non-critical operations with same effort

Global Parallelism Exploiting limited by conditional dependencies in the CDFG

– Optimize the latency for the whole design
– Use soft-constraints to model other design goals
Distribution of Scheduling Effort

- The scheduling effort is distributed equally

- But, schedule of Ops have different impact:
  - Sharing large FUs is more important than small FUs

- Can we do something to make the effort distribution match the importance?
Gradual Scheduling Definition

• Given:
  – Control-Data Flow-Graph to be scheduled
  – Criticality partitioning constraints
    • In terms of area of the functional unit

• Goal:
  – Schedule the critical-operations separately from noncritical-operations
Gradual Scheduling Framework

Aggressive yet time-consuming algorithm

Schedule Critical-Op

Optimize for:
• Speed
• Area
Exploit parallelism

Simple and fast algorithm

Schedule Trivial-Op

Only need valid results

CDFG
Overview

- Build new CDFG that only contains critical operations

Schedule the newly built CDFG

Schedule non-critical operations
CDFG Refining

• Given:
  – CDFG
  – Criticality partitioning constraints
    • In terms of area of the functional unit

• Goal:
  – Build a critical-operation-only CDFG
  – Preserve the constraints between critical operations
CDFG Refining Example

Refine
CDFG Refining Example

Longest-path distance

5 + 2 + 3
CDFG Refining Requirement

- The source of the noncritical chain should dominate the whole chain
CDFG Refining Requirement

• The source of the noncritical chain should dominate the whole chain

Illegal Chain

Not dominated by A
CDFG Refining Requirement

• The source of the noncritical chain should dominate the whole chain
PROBLEM SIZE REDUCTION
EXPERIMENT
Size of Refined CDFG vs. Partitions

• Size of Refined CDFG depends on the Partition

• Show the % of critical operations for:
  – **Chained**: load/store, gotos; Minimal Set.
  – **S16M16**: Including **Chained**, mults and shifts bigger than 16 bits
  – **All**: Including **Chained**, all arithmetic, shifts and comparisons; Maximal Set.
Experimental Setup

- LLVM-based HLS framework
- Targeting Altera Cyclone-II FPGA (available on DE2-70 board)
- Run on CHStone benchmarks

Diagram:

1. CDFG
2. Bit-level optimization
3. Gradual Scheduling
## Size of the Refined CDFG (Geomean)

<table>
<thead>
<tr>
<th></th>
<th>Refined CDFG</th>
<th>Partial-Scheduled CDFG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>N</td>
</tr>
<tr>
<td>Chained</td>
<td>12%</td>
<td>8%</td>
</tr>
<tr>
<td>S16M16</td>
<td>13%</td>
<td>9%</td>
</tr>
<tr>
<td>All</td>
<td>24%</td>
<td>16%</td>
</tr>
</tbody>
</table>
Problem Size Reduction

- CDFG
- Schedule Critical-Op
- Polynomial Time of (24% × 16%)
- Schedule Trivial-Op
- O(74% + 82%)

SDC
ALAP

ASP-DAC 2013, Pacifico Yokohama, Japan
EXPLOITING GLOBAL PARALLELISM
BY THE GRADUAL SCHEDULING FRAMEWORK
Parallelism Exploiting Techniques

- Hyper-block Formation
  - Build a bigger BB by if-conversion, may introduce lots of idle states

- (Traditional) Global Code Motion
  - Move the operations across BBs, but still restrict them inside a BB

- This work: No need to restrict non-critical operations inside a BB
“Implicit” Global Code Motion

Latency

Other BBs

BB1

BB2

Other BBs

Latency
Implicit Global Code Motion

- Execute in parallel with other BB
  - Not necessarily restricted in a specific BB
Implicit Global Code Motion

- Execute in parallel with other BB
- No need to duplicate the operations into BBs in each path
  - Confuse FU binding
  - Too many paths
Implicit Global Code Motion

- Execute in parallel with other BB
- No need to duplicate the operations into BBs in each path
- Completely integrated with scheduling algorithm
Implicit Global Code Motion

- **CDFG**
- **Schedule**
  - Critical-Op
  - Trivial-Op
- **Control-Deps Relaxing**
- **Refining**
- **Refined CDFG**
- **Scheduling**
  - (with Implicit Global Code Motion)
- **Wait States Insertion**
Why wait states?

- All cross BB chains are scheduled according to the longest-path in the CDFG
  - Without knowing the deps between BBs are conditional
Why wait states?

- But a shorter path maybe taken
- The latency of cross BB chains are NOT preserved
  \(- 4 + 2 + 3 \leq 5\)
Why wait states?

• But a shorter path maybe taken
• The latency of cross BB chains are NOT preserved
  \(-4 + 2 + 3 \leq 5 + [\text{wait states}]\)
Wait States Insertion

• Fix the cross BB constraints

• Based on Shortest Path Distance
  – #States = Expected SPD - Actual SPD

• Wait states are inserted as late as possible
Experimental Setup - Reminder

- Evaluate the latency reduction by Implicit Global Code Motion (IGCM)
- Run on CHStone benchmarks
Latency Reduction by IGCM

Latency normalized to default flow

Baseline

IGCM

0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

motion
dfadd
dfmul
dfdiv
aes
adpcm
blowfish
gsm
mips
jpeg
geomean
Latency Reduction by IGCM

Latency normalized to default flow

- IGCM
- C-Op
- Others

- motion
- dfadd
- dfmul
- dfdiv
- aes
- adpcm
- blowfish
- gsm
- mips
- jpeg
- geomean
Latency Reduction by IGCM

Latency normalized to default flow

- IGCM
- C-Op
- Others

1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

motion
dfadd
dfmul
dfdiv
aes
adpcm
blowfish
gsm
mips
jpeg
geomean
Latency Reduction by IGCM

Latency normalized to default flow

- IGCM
- C-Op
- Others

Values range from 0 to 1.
Summary

• Gradual scheduling framework
  – Schedule the critical/noncritical operations separately

• Reduced the problem size of scheduling
  – Size reduced to 24% of Nodes and 16% of Edges
  – Corresponds to 96.8% reduction in SDC scheduling

• Exploited cross-BB parallelism
  – Reduced run-time up to 37.7% and 15.5% on average
Thanks and Acknowledgement

• Thanks for the EDA group in SYSU for participating the project:
  – Q. Liu, J. Li, D. Chen, Z. Wang

• Thanks for helpful discussions with colleagues at ADSC:
  – K. Rupnow, S. Gurumani, T. Satria

• Thanks for listening!
Refining Time Less Than 0.05s!

![Graph showing the relationship between run time and normalized CDFG size for different conditions: Chained, S16M16, and All.](image)

- Chained
- S16M16
- All

**Run Time (s)**

0.05

0.04

0.03

0.02

0.01

0

0

0.5

1

Normalized CDFG Size