Bound-Oriented Parallel Pruning Approaches for Efficient Resource Constrained Scheduling of High-Level Synthesis

Mingsong Chen, Lei Zhou, Geguang Pu and Jifeng HE

Shanghai Key Lab of Trustworthy Computing
East China Normal University, China

October 1, 2013
Outline

- Introduction
- RCS using Branch-and-Bound Approaches
  - Graph–based Notations
  - BULB Approach
- Our Parallel Pruning Approach
  - Search Task Decomposition
  - Parallel Search Task Cooperation
- Experiments
- Conclusion
Outline

- Introduction
- RCS using Branch-and-Bound Approaches
  - Graph–based Notations
  - BULB Approach
- Our Parallel Pruning Approach
  - Search Task Decomposition
  - Parallel Search Task Cooperation
- Experiments
- Conclusion
SoC Design Cost Model

Big Savings by using ESL Methodology

Rising cost of IC design and effect of CAD tools
(Courtesy: Andrew Kahng, UCSD and SRC)
High Level Synthesis

- Convert ESL specifications to RTL implementations, and satisfy the design constraints.

  - **Input:** Behavior specifications (C, SystemC, etc.), and design constraints (delay, power, area, etc.)
  
  - **Output:** RTL implementations (datapath, controller)

```c
int Sample(){
    int A,B,C,D,E,F,G;
    Read(A, B, C, D, E);
    F = E * (A + B);
    G = (A + B) * (C + D);
    ......}
```
Resource Constrained Scheduling

- Various resource constraints (e.g., functional units, power, ...).
- **Scheduling** is a mapping of operations to control steps
  - Given a DFG and a set of resource constraints, RCS tries to find a (optimal) schedule with minimum overall control steps.

![Control Step Diagram]

Constraints:  
Delay(+) = 1, Delay(*) = 2, functional units: 1+, 1*

RCS is NP-Complete. RCS should take care of  
1) Operation precedence. 2) Resource sharing constraints
Basic Solutions

- Non-optimal heuristics
  - Force Directed Scheduling
  - List scheduling
    - Pros: Fast to get near-optimal results
    - Cons: schedules may not be tight

- Optimal approaches
  - Integer linear programming (sequential, parallel)
    - Pros: easy modeling
    - Cons: scalability, cannot handle non-integer time
  - Branch-and-bound
    - Pros: can prune the fruitless search space efficiently
    - Cons: few of them support parallel HLS specifically
Outline

- Introduction
- RCS using Branch-and-Bound Approaches
  - Graph–based Notations
  - BULB Approach
- Our Parallel Pruning Approach
  - Search Task Decomposition
  - Parallel Search Task Task Cooperation
- Experiments
- Conclusion
Graph-Based Notations

- **[ASAP, ALAP]** intervals indicate the earliest and latest start time of operations
- ASAP assumes unlimited resources
  - \( \text{ASAP}(op_i) = \text{CP}(G_{\text{pre}}(op_i)) - \text{delay}(op_i) + 1 \)
- ALAP needs to find a feasible schedule \( S \) first
  - \( \text{ALAP}(op_i) = \text{length}(S) - \text{CP}(G(op_i)) + 1 \)
  - Update ALAP when obtaining a new better schedule

![Graph Notation Example](image-url)

- \( [1,6] \)
- \( [1,5] \)
- \( [1,5] \)
- \( [1,4] \)
- \( [1,4] \)

- \( [1,5] \)
- \( [1,4] \)
- \( [1,4] \)

- \( [1,5] \)
- \( [1,4] \)
- \( [1,4] \)

- \( [1,5] \)
- \( [1,4] \)
- \( [1,4] \)

- \( [1,5] \)
- \( [1,4] \)
- \( [1,4] \)

- \( [1,5] \)
- \( [1,4] \)
- \( [1,4] \)

- \( [1,5] \)
- \( [1,4] \)
- \( [1,4] \)

- \( [1,5] \)
- \( [1,4] \)
- \( [1,4] \)

- \( [1,5] \)
- \( [1,4] \)
- \( [1,4] \)

- \( [1,5] \)
- \( [1,4] \)
- \( [1,4] \)

- \( [1,5] \)
- \( [1,4] \)
- \( [1,4] \)

- \( [1,5] \)
- \( [1,4] \)
- \( [1,4] \)

- \( [1,5] \)
- \( [1,4] \)
- \( [1,4] \)

- \( [1,5] \)
- \( [1,4] \)
- \( [1,4] \)
A *schedule* is a binary relation of operations and corresponding dispatching control steps.

- E.g., \{(v1, 1), (v2, 1), (v3, 3), (v4, 5), (v5, 6)\}

Based on [ASAP, ALAP], naively enumerating all the possibilities can be extremely time consuming.

- The operations are enumerated in a specific order.
- Each operation is enumerated from ASAP to ALAP.
BULB tries to prune fruitless enumerations.

B&B approach keeps two data structure regarding bound information.

- $S_{bsf}$, best complete schedule searched so far
- $S$, current incomplete schedule
Pruning in BULB

- **Pruning**  \[\text{lower} > \omega\]
- **Termination**  \[\text{GL} == \omega \text{ or fully explored}\]
- **Substitution**  \[\text{if(upper} < \omega) \omega = \text{upper}\]

\(\omega\) plays an important role in B&B approaches. A wise use of \(\omega\) can
- enable the fast pruning of inferior schedules during RCS;
- tighten the [ASAP, ALAP] intervals, i.e., search space.
Outline

- Introduction
- RCS using Branch-and-Bound Approaches
  - Graph–based Notations
  - BULB Approach
- Our Parallel Pruning Approach
  - Search Task Decomposition
  - Parallel Search Task Task Cooperation
- Experiments
- Conclusion
The search space can be calculated using the cartesian product of [ASAP, ALAP] intervals.

If no better schedule is found, RCS can be easily stuck-at-local-search, i.e., be trapped in the deep recursive search.

a) Without partitioning

b) With 4 partitions
Search Space Partitioning

Termination condition:
1) $\omega = \text{GL}$; or 2) all the sub-search finishes.
In BULB approach, the tightest initial $\omega$ can achieve the best RCS time.

However, it is hard to achieve such a tightest estimation on a single-core platform.

If there are $k$ cores, the upper bound will be speculated with lengths $U_1, U_2, \ldots, U_k$ where $GL=U_1<U_2<\ldots<U_k=\omega$.
Static Upper Bound Speculation

Termination condition:
1) $\omega = GL$; or
2) Some sub-task finishes and finds one feasible schedule.
Dynamic Upper Bound Speculation

- Assume that $GL = U_1 < U_2 < \ldots < U_k = \omega$ are $k$ upper-bound speculations.

- When a sub-search task finds a new $\omega'$ such that $U_{i-1} < \omega' < U_i$ $(k > i > 1)$. The speculation on $U_j$ $(j > i)$ becomes useless.

- The speculation of the $j^{th}$ sub-task $(j > i)$ can be $U_j' = \max(\text{globalLow}, \omega' - |i-j|)$. 

![Diagram showing upper bound speculations and how they are updated](image-url)
Hybrid Approach

- Search space partitioning and static upper-bound speculation approaches can be combined to further reduce the searching time.

- Assume that $GL=U_1<U_2<...<U_k=\omega$ are $k$ upper-bound speculations. The hybrid approach has $k$ iterations with increasing upper-bound sizes.

**Termination condition: find a schedule in the $i$th iteration ($i \leq k$) and**

1) $\omega=\text{globalLow}$; or
2) all the sub-tasks in the iteration finish.
Minimum $\omega$ Synchronization

- During RCS, the search progress information (i.e., $\omega$) of each sub-task can be different.

- If one sub-task finds a new shorter schedule (i.e., shorter $\omega$) and such information can be propagated to other sub-tasks, the search space can be reduced drastically.

![Diagram showing cooperation among sub-tasks with minimum $\omega$.]
Cooperative Sub-task Implementation

- Each sub-task is modeled using an **EFSM**.
- Three states: *changed* means find a better schedule with length $\omega'$; *!changed* indicates no new better schedule since last update of $\omega$; *done* denotes the termination.

```
query() < $\Omega$
/ $\omega = \max(query() - spec, globalLow)$,
updateALAP()

$\omega' \leq query() / \omega = \omega'$, update(i, $\omega$)

$\omega > globalLow / \omega = max(\omega - 1, globalLow)$,

complete /

$\omega = globalLow /
```

![State Diagram](chart.png)
Introduction

RCS using Branch-and-Bound Approaches
- Graph–based Notations
- BULB Approach

Our Parallel Pruning Approach
- Search Task Decomposition
- Parallel Search Task Cooperation

Experiments

Conclusion
Benchmarks & Settings

- Using benchmarks from *MediaBench*.
- **BULB & our approach** are implemented using C++ and OpenMP.
- Experiments were conducted on a Linux server with 96 Intel Xeon 2.4GHz cores and 1T RAM.

Setting of functional units:

<table>
<thead>
<tr>
<th>Functional Unit</th>
<th>Operation class</th>
<th>Delay (unit)</th>
<th>Power (unit)</th>
<th>Energy (unit)</th>
<th>Area (unit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD/SUB</td>
<td>+/-</td>
<td>1</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>MUL/DIV</td>
<td>*/</td>
<td>2</td>
<td>20</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>MEM</td>
<td>LD/STR</td>
<td>1</td>
<td>15</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>Shift</td>
<td>&lt;&lt;/&gt;&gt;</td>
<td>1</td>
<td>10</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>Others</td>
<td>…</td>
<td>1</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>
### Results under Functional Constraints

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>CP (sec.)</th>
<th>BULB (sec.)</th>
<th>Spec. (sec.)</th>
<th>Partitioning (sec.)</th>
<th>Hybrid part.+sp.</th>
<th>Max speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>name</td>
<td># of a, m</td>
<td>ssp.+dsp.</td>
<td>w/o dspec. w/ dspec.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARFilter</td>
<td>1, 3</td>
<td>NA</td>
<td>0.31</td>
<td>0.02</td>
<td>0.40</td>
<td>0.39</td>
</tr>
<tr>
<td></td>
<td>1, 4</td>
<td>NA</td>
<td>0.78</td>
<td>0.06</td>
<td>1.00</td>
<td>0.97</td>
</tr>
<tr>
<td></td>
<td>1, 5</td>
<td>NA</td>
<td>0.77</td>
<td>0.07</td>
<td>0.98</td>
<td>0.98</td>
</tr>
<tr>
<td></td>
<td>2, 3</td>
<td>1.93</td>
<td>0.01</td>
<td>0.01</td>
<td>0.04</td>
<td>0.03</td>
</tr>
<tr>
<td>FDCT</td>
<td>1, 2</td>
<td>NA</td>
<td>36.91</td>
<td>43.89</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td></td>
<td>2, 2</td>
<td>NA</td>
<td>201.59</td>
<td>58.31</td>
<td>9.90</td>
<td>6.95</td>
</tr>
<tr>
<td></td>
<td>2, 3</td>
<td>NA</td>
<td>19.80</td>
<td>6.51</td>
<td>7.24</td>
<td>3.19</td>
</tr>
<tr>
<td></td>
<td>2, 4</td>
<td>NA</td>
<td>4.07</td>
<td>5.06</td>
<td>2.69</td>
<td>2.22</td>
</tr>
<tr>
<td></td>
<td>2, 5</td>
<td>NA</td>
<td>0.92</td>
<td>0.99</td>
<td>1.10</td>
<td>1.07</td>
</tr>
<tr>
<td></td>
<td>3, 4</td>
<td>NA</td>
<td>0.55</td>
<td>0.50</td>
<td>0.78</td>
<td>0.77</td>
</tr>
<tr>
<td></td>
<td>4, 4</td>
<td>NA</td>
<td>0.12</td>
<td>0.12</td>
<td>0.19</td>
<td>0.18</td>
</tr>
<tr>
<td>Feedback</td>
<td>4, 4</td>
<td>NA</td>
<td>154.18</td>
<td>176.43</td>
<td>2.92</td>
<td>2.88</td>
</tr>
<tr>
<td></td>
<td>4, 5</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>3.14</td>
<td>3.08</td>
</tr>
<tr>
<td></td>
<td>5, 5</td>
<td>NA</td>
<td>4.87</td>
<td>5.50</td>
<td>0.35</td>
<td>0.35</td>
</tr>
<tr>
<td>Cosine 1</td>
<td>1, 2</td>
<td>NA</td>
<td>107.43</td>
<td>137.36</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td></td>
<td>2, 2</td>
<td>NA</td>
<td>622.83</td>
<td>41.02</td>
<td>781.34</td>
<td>66.74</td>
</tr>
<tr>
<td></td>
<td>3, 3</td>
<td>NA</td>
<td>0.01</td>
<td>&lt;0.01</td>
<td>0.05</td>
<td>0.04</td>
</tr>
<tr>
<td>Collapse</td>
<td>2, 1</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>0.04</td>
<td>0.03</td>
</tr>
<tr>
<td></td>
<td>2, 2</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>&lt;0.01</td>
<td>0.02</td>
</tr>
<tr>
<td></td>
<td>2, 3</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td></td>
<td>2, 4</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
</tr>
</tbody>
</table>

RCS efforts are significantly improved with 8 cores:
- Our parallel approaches outperform both ILP and BULB approaches
- Hybrid approach can achieve the best overall performance
- Significant improvement using hybrid approach with 8 cores.
- When each core is assigned with ≥8 partitions, the performance will not change drastically.
- The search space is divided into 128 parts.
- When the number of cores is larger than 4, increasing the core number will not reduce the search time significantly.
- FDCT design with different power and area constraints
- The hybrid approach can achieve a speedup of several orders of magnitude.
Conclusions

- RCS is a major bottleneck in HLS
  - Branch-and-bound approaches are promising for optimal resource-constrained scheduling

- Proposed various parallel pruning heuristic
  - Search space partitioning approach
  - Static /dynamic upper bound speculation approaches
  - Parallel sub-task cooperation framework

- Successfully applied on various benchmark with different resource constraints
  - Significant reduction in overall RCS efforts
Thank you!