Exploring ILP for VLIW architecture by Quantified Modeling and Dynamic Programming-based Instruction Scheduling

Author: Can Deng, Zhaoyun Chen, Yang Shi, Xichang Kong and Mei Wen*
Reporter: Can Deng
Outline

- PART 1: Introduction
- PART 2: Method
- PART 3: Results
- PART 4: Conclusion
PART 1

Introduction
1.1 Background

- VLIW architecture is widely adopted in dedicated processors
- The performance of VLIW processors is getting higher and higher
1.1 Background

- **Disadvantage:**
  - LS algorithms make a decision from the feasible solutions in a local view.
  - The efficiency of the final solution is unpredictable.

<table>
<thead>
<tr>
<th></th>
<th>LS (list scheduling)</th>
<th>DP (dynamic programming)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Searching space</td>
<td>Loccal view</td>
<td>Global view</td>
</tr>
<tr>
<td>Goal</td>
<td>Feasible solution</td>
<td>Optimal solution</td>
</tr>
<tr>
<td>Time overhead</td>
<td>Low</td>
<td>low</td>
</tr>
<tr>
<td>Space Complexity</td>
<td>Low</td>
<td>High</td>
</tr>
</tbody>
</table>
1.2 Motivation

- Propose a dynamic programming based strategy (DPS) to make a trade-off
- Achieve a high efficiency scheduling solution within acceptable time overhead
- Construct a quantifiable model for the instruction scheduling problem and get a theoretical upper bound of efficiency
PART 2

Method
### 2.1 Objective:

\[
min(T) = min(max(\sum_{q=0}^{T_0-1} (1 - \sum_{t=0}^{q} X_{i,t}^f) + C_i))
\]

### Constraints:

1. \[
\sum_{f=0}^{m-1} \sum_{t=0}^{T_0-1} X_{i,t}^f = 1 \quad (\forall I_i \in I)
\]

2. \[
\sum_{f=0}^{m-1} \sum_{t=0}^{T_0-1} Y_{i,f} \cdot X_{i,t}^f = 1 \quad (\forall I_i \in I)
\]

3. \[
\sum_{i=0}^{n-1} X_{i,t}^f \leq 1 \quad (\forall t \in T, \forall F_f \in F)
\]

4. \[
S_i + Edge_{i,l} \leq S_l \quad (I_i, I_l \in I)
\]

<table>
<thead>
<tr>
<th>(I)</th>
<th>Sequence of Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>(F)</td>
<td>Sequence of function unit</td>
</tr>
<tr>
<td>(T)</td>
<td>Sequence of cycle</td>
</tr>
<tr>
<td>(X_{i,t}^f)</td>
<td>Binary variable ({0,1})</td>
</tr>
<tr>
<td>(Y_{i,f})</td>
<td>Binary variable ({0,1})</td>
</tr>
</tbody>
</table>
2.2 Dynamic Programming-Based Strategy (DPS)
2.2.1 State Computation

**Algorithm 1: State Computation**

**Input:** Instructions  
**Output:** States

1. for $i \leftarrow 0$ to $(n - 1)$ do
2. // $n$ is the number of instructions;
3. if $chl = 0$ then
4. // $chl$ is the number of children;
5. $p[i] \leftarrow C_i$
6. else
7. for $j \leftarrow 0$ to $(chl - 1)$ do
8. $p[i] \leftarrow \max(p[j] + \text{Edge}_{i,j}, C_i)$

$p[i] = \max(p[j] + \text{Edge}_{i,j}, C_i)$
2.2.2 Instruction Assignment

Algorithm 2: Pseudocode for DPS Method

Input: A block
Output: A scheduled block

1. Call State Computation // Call for Algorithm 1;
2. for $i \leftarrow 0$ to $(n - 1)$ do
3.   Pop the highest state instruction $I_i$;
4.   for $t \leftarrow ES_i$ to $T_0 - 1$ do
5.     for $k \leftarrow 0$ to $(m - 1)$ do
6.       if function unit available then
7.         Allocate function unit $F_f$ and time slot $t$ for
8.         instruction $I_i$;
9.       else
10.      Shift to the next time slot;
11. Update $ES$ values for all the children;
2.3 Experiments

- **Platform:** FT-Matrix DSP
- **Benchmark:** Transcendental Functions
- **BaseLines:** Heterogeneous Earliest Finish Time (HEFT), Critical-Path-Node-Dominant (CPND), and Longest Job First (LJF)
PART 3

Results
3.1 Execution cycle of solutions

Instruction number of benchmarks

Execution cycle of the benchmarks
3.2 Efficiency

\[ \text{Efficiency} = \frac{\text{Execution cycle of GUROBI}}{\text{Execution cycle of the methods}} \]
3.2 Time Overhead
PART 4

Conclusion
4. Conclusion

- The DPS proposed in this work achieves a trade-off between execution and time overhead.
- Compared with the three LS algorithms, DPS shows a good scalability and efficiency improvement of up to 44% within acceptable time overhead.
- One future work is to explore the optimization space toward the optimal solutions.
Thank you!