Retiming for Synchronous Data Flow Graphs

N. Liveris, C. Lin, J. Wang, H. Zhou, P. Banerjee*
Northwestern University, Evanston IL
*University of Illinois, Chicago IL

ASP-DAC 2007, Yokohama, Japan
January 25, 2007
Outline

• Intro to SDF and retiming
• Previous work
• First Algorithm
• Improved Algorithm
• Experimental Results
• Conclusion
Synchronous Dataflow Graphs

- Each node represents a computation process
  - constant production and consumption rate
  - executed a specific number of times during each complete cycle
- Edge represents a channel between two processes
  - FIFO protocol for tokens
  - initial number of tokens on edge (delays)
Blocking vs Non-blocking Schedule

Example of a blocking schedule

Example of a non-blocking schedule
Retiming SDF Graphs

- DSP applications with constant consumption and production data rates and predictable execution time are modeled by SDF graphs.

- Some applications whose behavior is determined at run-time or that share resources with high-priority tasks are normally executed on programmable cores.

- When data dependencies exist between SDF actors and tasks executed on programmable cores, a non-blocking schedule may not be feasible.
Example
Example – retimed

<table>
<thead>
<tr>
<th>d</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
</tr>
</tbody>
</table>

\[
p(e) \quad w(e) \quad c(e)
\]

\[
\begin{array}{c|cc|c|cc}
\text{CPU} & \text{C1} & \text{CPU} & \text{C1} & \text{CPU} & \text{C1} \\
\hline
\text{C2} & \text{A1} & \text{B1} & \text{C2} & \text{A1} & \text{B1} \\
\text{A2} & \text{B2} & \text{B2} & \text{B3} & \text{B3} \\
\text{B3} & \text{A2} & \text{B2} & \text{T=5} & \\
\end{array}
\]
Previous Approach


• Only check whether a given cycle time is feasible

• Computing the maximum path in the EHG (Equivalent Homogenous Graph)
  – a distinct node for each node instance
  – each token transferred on a separate edge
  – \( p(e) = c(e) = 1 \)
  – number of edges \( \sum_{(u,v) \in E} q(v) \ c(u,v) \)

• Selection of node \( v \), whose \( r(v) \) will be increased, is based on heuristic

• Termination criteria is not provable
Our Approach

- Computation of max length is done on the SDF graph
  - avoiding expensive generation of EHG
  - avoiding computation for nodes that cannot affect the max length path
- Selection of nodes is justified based on properties
- Algorithm reduces cycle time at each iteration or proves that the cycle time of the iteration is optimal
- Upon termination an optimal solution is generated
Dependence Walk

Execution of \((v_i, l_i)\) can start only after execution of \((v_{i-1}, l_{i-1})\) has been completed.

\[W = (A,1) \rightarrow (B,1) \rightarrow (C,1) \rightarrow (A,2) \rightarrow (B,2) \rightarrow (C,3)\]

(node name, instance number)
Critical Dependence Walk

\[ q_A = 3 \quad q_B = 2 \quad q_C = 3 \]

\[ W = (A,1) \rightarrow (B,1) \rightarrow (C,1) \rightarrow (A,2) \rightarrow (B,2) \rightarrow (C,3) \]

\[(A,1) = (v_0,l_0)\]

Execution of \((v_i,l_i)\) starts exactly when execution of \((v_{i-1},l_{i-1})\) completes and \((v_0,l_0)\) starts at the beginning of the period (time 0).
Node Selection

If \( W \) is a critical walk, with \( t(v_n,l_n)+d_n = T \), then the only way to obtain graph with \( T' < T \) is by increasing \( r(v_n) \).

\[ W = (A,1) \rightarrow (B,1) \rightarrow (C,1) \rightarrow (A,2) \rightarrow (B,2) \rightarrow (C,2) \rightarrow (A,3) \]

\[(A,3) = (v_n,l_n)\]
In this example the length of $W$ has been reduced after the retiming operation.

$W = (A, 2) \rightarrow (B, 1) \rightarrow (C, 1) \rightarrow (A, 3) \rightarrow (B, 2) \rightarrow (C, 3)$
Maximum Length Walk Computation

- Execution of \((v_i, l_i)\) cannot start before execution of \((v_{i-1}, l_{i-1})\) has finished
- Computing the arrival time of each walk starting from the last instance of each node
- Dynamic programming algorithm (memory function)

```plaintext
proc get_t(v, k, r)
    if (k < 1) then
        return -d(v);
    fi;
    if (t[v, k] ≠ -1) then
        return t[v, k];
    fi;
    max_t ← -1;
    for each \((u, v) ∈ E\)
        l ← \([\frac{k·c(u, v)−w(u, v)}{p(u, v)}]\);
        t_1 ← get_t(u, l) + d(u);
        if (max_t < t_1) then
            max_t ← t_1;
        fi;
    endfor;
    t[v, k] ← max_t;
    return t[v, k];
```
Termination Conditions

• It is proven that the algorithm will always find a basic optimal solution, i.e. in the solution there will exist \( v \) such that \( r(v) < q(v) \)

• Following from the above condition and from the conditions that can trigger an \( r \) change:

\[
(\forall v : r(v) \leq 2 \cdot q_v \cdot |V|)
\]

If any of these conditions are violated, the algorithm cannot improve the best solution found thus far.
First Version of the Algorithm

- Finds last node of a critical walk for which \( t(v_n, l_n) + d_n = T \)
- Increments \( r(v_n) \) (\( r'(v_n) = r(v_n) + 1 \))
- Recomputes arrival times for the nodes using the dynamic programming algorithm
- Stores solution if \( T' < T \)
- Continues this process until any of the termination conditions are satisfied
- Worst-case complexity \( O(|V|^3|E|q_{ave}^2) \)
Improved Version

• First version changes the $r(v)$ of one node by 1 and then tries to find critical walk again
  – guarantees that the edge weight will never become negative, but
  – for each $r$ change, arrival times have to be recomputed

• Improved version relaxes the non-negativity constraint for edges, and does more than one change in each iteration

• Mechanism can be used to validate additional constraints for edges
Improved Version

• Maintains two queues:
  • First queue holds the nodes, which require an r-value increase in order for a potential reduction of T to occur
  • Second queue holds edges with negative weights. The r-value of the head of each edge needs to be increased, so that the non-negativity constraint is satisfied
• Arrival times are recomputed only after queues are empty (all necessary r-value increases have occurred)
Execution Snapshot 1

\[ T_{\text{step}} = 3 \quad Q_1 = \{ B \} \quad Q_2 = \{ \} \]

\[ T_{\text{step}} = 4 \quad Q_1 = \{ E \} \quad Q_2 = \{ \} \]
Execution Snapshot 2

\[
\begin{align*}
\text{ro}(A) &= 0 & \text{ro}(B) &= 0 & \text{ro}(C) &= 1 & \text{ro}(D) &= 0 & \text{ro}(E) &= 4 & \text{ro}(F) &= 0 \\
\text{r}(A) &= 0 & \text{r}(B) &= 1 & \text{r}(C) &= 1 & \text{r}(D) &= 0 & \text{r}(E) &= 4 & \text{r}(F) &= 0 \\
\text{tf}(A,q_A) &= 2 & \text{tf}(B,q_B) &= 1 & \text{tf}(C,q_C) &= 3 & \text{tf}(D,q_D) &= 4 & \text{tf}(E,q_E) &= 1 & \text{tf}(F,q_F) &= 2 \end{align*}
\]

\[
T_{\text{step}} = 3 \quad Q_1 = \{C,D\} \quad Q_2 = \{\}
\]
### Experimental Results ($q_{\text{max}}=32$)

<table>
<thead>
<tr>
<th>Graph</th>
<th>$T$</th>
<th>Execution time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>O’Neil’s</td>
<td>First</td>
</tr>
<tr>
<td>$s27$</td>
<td>459</td>
<td>416</td>
</tr>
<tr>
<td>$s208.1$</td>
<td>834</td>
<td>834</td>
</tr>
<tr>
<td>$s298$</td>
<td>1083</td>
<td>1027</td>
</tr>
<tr>
<td>$s344$</td>
<td>2534</td>
<td>2468</td>
</tr>
<tr>
<td>$s349$</td>
<td>1503</td>
<td>1415</td>
</tr>
<tr>
<td>$s382$</td>
<td>1312</td>
<td>1273</td>
</tr>
<tr>
<td>$s386$</td>
<td>938</td>
<td>806</td>
</tr>
<tr>
<td>$s444$</td>
<td>1185</td>
<td>888</td>
</tr>
<tr>
<td>$s526$</td>
<td>2161</td>
<td>2007</td>
</tr>
<tr>
<td>$s641$</td>
<td>690</td>
<td>610</td>
</tr>
<tr>
<td>$s820$</td>
<td>1594</td>
<td>1573</td>
</tr>
<tr>
<td>$s953$</td>
<td>1776</td>
<td>1776</td>
</tr>
</tbody>
</table>
Modeling Environment

<table>
<thead>
<tr>
<th></th>
<th>d</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>I</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>O</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
## Experimental Results

<table>
<thead>
<tr>
<th>Graph</th>
<th>T</th>
<th>Execution Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Initial</td>
<td>Final</td>
</tr>
<tr>
<td>$s_{27}$</td>
<td>368</td>
<td>351</td>
</tr>
<tr>
<td>$s_{208.1}$</td>
<td>1035</td>
<td>852</td>
</tr>
<tr>
<td>$s_{298}$</td>
<td>1052</td>
<td>742</td>
</tr>
<tr>
<td>$s_{344}$</td>
<td>1062</td>
<td>928</td>
</tr>
<tr>
<td>$s_{349}$</td>
<td>933</td>
<td>833</td>
</tr>
<tr>
<td>$s_{382}$</td>
<td>951</td>
<td>908</td>
</tr>
<tr>
<td>$s_{386}$</td>
<td>745</td>
<td>650</td>
</tr>
<tr>
<td>$s_{444}$</td>
<td>902</td>
<td>882</td>
</tr>
<tr>
<td>$s_{526}$</td>
<td>1690</td>
<td>1690</td>
</tr>
<tr>
<td>$s_{641}$</td>
<td>694</td>
<td>665</td>
</tr>
<tr>
<td>$s_{820}$</td>
<td>1264</td>
<td>1219</td>
</tr>
<tr>
<td>$s_{953}$</td>
<td>1558</td>
<td>1558</td>
</tr>
</tbody>
</table>

* Indicates unusual execution times.
Summary

• Presented two new algorithms for retiming SDF graphs
• Algorithms aim at minimizing the cycle length of the SDF and are optimal
• Improved version is orders of magnitude faster than other approaches
Thank you