DATA PATH SYNTHESIS WITH GLOBAL TIME CONSTRAINT

Prashant P. Nedungadi  M. Balakrishnan  Anshul Kumar
CSE Department, IIT Delhi
New Delhi, 110016, India

Abstract

This paper presents an algorithm and its implementation for performing scheduling and operator allocation for data path synthesis. The main advantage of this approach is that it is capable of handling global time constraint as compared to earlier systems[1] which expected the designer to artificially impose a time constraint on the individual blocks. This is achieved by judiciously distributing the time available over various blocks to reduce the global resource requirements. Another feature of our approach is that it is capable of handling a library which may have operators with different speeds for the same operation. It is proposed to integrate this scheme in the IDEAS system[2].

1 Global Time Constraint Scheduling

The input to the algorithm is a behavioral description in terms of the Control Data Flow Graph (CDFG), a global time constraint and an operator library. The algorithm proceeds iteratively and at each step time constrained scheduling and allocation is performed for individual blocks (using FDS[3]). The algorithm has two major steps:

• Initial Allocation: From the CDFG basic blocks (ones that do not contain any control nodes) are extracted. An ASAP schedule is done for each of the blocks to calculate the total minimum time. Slack time is the difference between the specified global time constraint input by the designer and the calculated total minimum time. For each block the Force Directed Scheduling algorithm is performed with fast operator set and ASAP time to find the Optimal operator allocation and schedule.

• Allocation Refinement : First a set of candidate operators are identified which can be dropped individually without exceeding the global time constraint. For an operator p, drop time is estimated by computing the total number of time steps in which it is used. Two important points to be noted are:

  a. The estimated additional time is an upper bound on the extra time required by dropping the operator.
  b. If there is no other operator in the allocated set that is able to perform the same operation, then the drop time of p is infinity i.e p cannot be dropped.

For all operators in the drop candidate set, a benefit coefficient is computed. It is inversely proportional to the drop time (additional steps are required by dropping the operator) and directly proportional to the operator cost (large decrease in cost by dropping a 'costly* operator). The procedure is terminated when no operator can be dropped while meeting the global time constraint. Among the operators which can be dropped, the one with the maximum benefit coefficient is dropped. For each of the blocks which used the dropped operator in the preceding schedule, we do a resource constrained scheduling. This is presently implemented by incrementally increasing the allocated time until the operator gets deallocated. Total time and slack time are updated based on the new schedule and the procedure is repeated.

2 Implementation

The extracted blocks are classified as Basic block, Sequential block, Conditional forks and Conditional loops. We define the utility of any operator j of type k in a basic block Bj, \( u_k(B_j) \), as

\[
\text{No of steps the operator } j \text{ of type } k \text{ is used in } B_j \quad \frac{\text{cost}(k)}{\text{cost}(k)}
\]

Let \( t_{Bj} = \text{Total number of steps in block } Bj \)
The utilities and delays for a block $B$ with sub-blocks $B_1, B_2, \ldots, B_n$ are calculated as follows.

\[ u_k(B_j) = \sum_{i=1}^{n} u_i^B_j, \quad t_B = \sum_{i=1}^{n} t_i B_i \quad \text{(Sequential)} \]

\[ u_k(B_j) = \max_{i=1}^{n} u^B_{j i}, \quad t_B = \max_{i=1}^{n} t_i B_i \quad \text{(Conditional)} \]

\[ u_k(B_j) = m * u^j, \quad t_B = m * t_B \quad \text{(loop)} \]

Where $m$ = average number of times (expected) the block $B_j$ occurs.

The benefit coefficient for each operator as used in the scheduling algorithm is just the inverse of its global utility. Based on these benefit coefficients, the algorithm described in the previous section performs scheduling and allocation under global time constraint.

### 3 Examples and Results

The behavior given in fig.1 was tested to illustrate the algorithm. It is assumed that the While construct

\begin{verbatim}
BEHAVIOR testbeh OF test IS
    ...;
    BODY
    WHILE (x < a) DO
        x1 := x + dx;
        u1 := u - (3 * x * u * dx) - (3 * y * dx);
        y1 := y + (u * dx);
        x := x1; u := u1; y := y1;
    ENDWHILE;
    IF (x + y > a) THEN
        x1 := u * dx;
        y1 := x + y;
        u1 := u + y * (dy - 1);
        x := x1; u := u1; y := y1;
    ENDIF;
END test beh
\end{verbatim}

Figure 1 IDEAL Behavioral Body

iterates an average of 20 times in a single run. The successive iterations with the total schedule time and allocated operator set is shown in fig. 2. Some iterations result in reducing the total cost without increase in total time. In this example, this is achieved by allocating more time to the assumed operator costs and the result of synthesis for different values of global time constraint are shown in else branch. This branch consumes more resources, but because it has shorter ASAP time schedule than the 'then' branch, additional allocated time does not change the total time. This illustrates the power of Global time constrained scheduling and allocation.

### Acknowledgements

This work was partially supported by Dept. of Electronics, Govt. of India under IDEAS project.

### References


Cost of Adder, Subtractor, Comparator = 1 unit each
Cost of Multiplier = 16 units

<table>
<thead>
<tr>
<th>Global Time Constraint</th>
<th>Iteration No</th>
<th>Operator Set &amp; Cost</th>
<th>Total Sdxxti.-Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>[3,2,1,3] 54</td>
<td>111</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>[2,2,1,2] 37</td>
<td>111</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>[1,2,1,2] 36</td>
<td>113</td>
</tr>
<tr>
<td>final</td>
<td></td>
<td>[1,1,1,2] 35</td>
<td>114</td>
</tr>
</tbody>
</table>

| 120                    | 1            | [3,2,1,3] 54        | 111                 |
|                        | 2            | [2,2,1,2] 37        | 111                 |
|                        | 3            | [1,2,1,2] 36        | 113                 |
| final                  |              | [1,1,1,1] 19        | 154                 |

Figure 2 Synthesis Results