**PROGRAMMING ASSIGNMENT – 1:**

**SCHEDULING OF A DATA FLOW GRAPH**

The process of Scheduling is associated with allocation of control steps to various operations subjected to different design constraints. Since, scheduling determines the concurrency of the resulting operations therefore it affects performance. The scheduling problem with respect to High level synthesis can be of various types namely – unconstrained, timing constrained, resource constrained and resource-timing constrained.

The Algorithms associated with Scheduling problem can be primarily categorised into two types – Heuristics and Exact. Exact algorithms such as Linear Programming approaches tend to provide the optimal solution but consumes a lot of time which may not be feasible for practical scenarios. So, a second class of algorithms namely heuristics exists which take a 'guess' approach to problem solving, yielding a 'good enough' answer, rather than finding a 'best possible' solution in a reasonable time frame.

1. **RESOURCE UNCONSTRAINED SCHEDULING**

* **As Soon As Possible (ASAP) Scheduling**

As-Soon-As-Possible (ASAP) scheduling is an unconstrained scheduling algorithm (no constraint on resources) that is used in HLS to calculate the earliest start time of operations. ASAP is a minimum latency schedule. The algorithm schedules each operation, one at a time, into the earliest possible control step subjected to satisfying the partial order, i.e., an operation is scheduled if and only if all its predecessors are scheduled in earlier control steps i.e. satisfying the precedence constraints.

Ti >= Tj + dj, (i,j) ϵ E

Ti = maxj {Tj + dj, (j,i) ϵ E}

ASAP can be solved easily in polynomial time by topologically sorting vertices of the graph.

* **As Late As Possible (ALAP) Scheduling**

As-Late-As-Possible (ALAP) scheduling is similar to ASAP, but instead of scheduling the operations in early control steps, it schedules them as late as possible such that the upper bound on latency constraint is not violated. In other words, ALAP algorithm schedules operations in the latest possible control step, subjected to satisfying the (reverse) partial order, i.e., an operation is scheduled if and only if all its successors are scheduled in latter control steps.

Ti = minj {Tj – di, (j,i) ϵ E}

If ALAP algorithm can schedule all the operations within 1st control step then the scheduling is successful. ALAP finds a schedule that satisfies an upper bound on the latency. It can also be stated as minimum latency schedule with maximum start times.

1. **LATENCY / RESOURCE CONSTRAINED SCHEDULING**

* **Integer Linear Programming based Scheduling**

The ILP based approach provides an exact solution for scheduling the operations in a data flow graph. For calculating the latency / resource constrained scheduling, we perform ASAP and ALAP to calculate the mobility range of each operation. The mobility (ALAP - ASAP) also known as slack gives the flexibility to the operations to get scheduled while minimising the desired parameter. Following this, the Start Time, Precedence and Resource constraints of the data flow graph are formulated. The resource constraints for resource constrained scheduling has the number of instances of resources as unknowns.

The objective function for **Minimum Latency Resource Constrained Scheduling** is to minimize the start times of each operation subjected to an upper bound on the number of resources for each operation. While for **Minimum Resources Latency Constrained Scheduling**, the emphasis of the objective function is on the total resource cost.

Although the ILP approach provides an optimal solution and works well for a few thousand variables, it may not feasible for practical scenarios since it is a NP-hard problem. So, the execution time increases rapidly with increase in decision variables. Also, other constraints can be easily incorporated into ILP.

* **List Based Scheduling**

List Scheduling is a Heuristic based approach since it handles each control step individually in increasing order of time step. It tries to schedule “maximum” number of operations in one control step, subjected to resource constraints and data dependency. The List scheduling process makes use of a ready list that has the unscheduled operations which can be scheduled in the current control step without violating the precedence constraint i.e. the predecessors must have got scheduled in the prior steps.

For **Minimum Latency Resource Constrained Scheduling**, the operations are chosen from the current ready list such that resource constraint is never violated. The choice of operation is based on a priority function that takes into consideration the mobility. Operations with lower mobility are given the highest priority since they are less flexible to get scheduled in future control steps. Delaying them would result in increasing the latency.

For **Minimum Resources Latency Constrained Scheduling**, the computation starts with the minimum number of resources. The slack of an operation is used to rank the operations, where the slack is the difference between the latest possible start time (computed by an ALAP schedule) and the index of the schedule step under consideration. The lower the slack, the higher the urgency in the list is. Operations with zero slack are always scheduled; otherwise the latency bound would be violated. Scheduling such operations may require additional resources, thus the number of resources gets updated only at the time of need. The remaining operations are scheduled only if they do not require additional resources.

1. **ASSUMPTIONS**

* The assumptions made for implementing MLRC and MRLC List based approach is that the delays associated with each operations are bounded and there are no multicycle paths.
* Also, there is only one distinct resource for a given operation, e.g. there is only one unit that performs multiplication, one that performs addition, one that performs subtraction, etc.

1. **APPLICATIONS OF MLRC and MRLC**

MLRC based algorithms are useful for resource dominated circuits where the primarily goal is to achieve the functionality in the least possible time i.e. minimizing latency without violating any resource constraints or ignoring the data dependencies that exist between the operations.

MRLC based algorithms are useful for applications which focuses on minimizing the cost of the resources under a given latency.

**PSEUDO CODE**

Here, we discuss the Pseudo codes of the various algorithms that we have used in our program.

**ASAP:**

ASAP (G(V, E)) {

Schedule by setting : = 1;

repeat {

Select a vertex , whose predecessors are all scheduled:

Schedule , by setting = ;

}

Until ( is scheduled);

Return

}

**ALAP:**

ALAP (G(V, E), λu) {

Schedule by setting : = λu + 1;

repeat {

Select a vertex , whose successors are all scheduled:

Schedule , by setting = ;

}

Until ( is scheduled);

Return

}

**MLRC using ILP:**

1. V is set of vertices corresponding to different operations = {0 (source),1,2,3,…n (sink)}
2. Calculate the earliest start-time tis using ASAP
3. The upper bound on latency is λu which is taken form ASAP λu = λmin
4. Calculate the latest start-time using ALAP (λmin)
5. A decision variable xil = 1, indicates that operation “i” has started in cycle l, and 0 otherwise.
6. We find a solution to the objective function i.e. determination of the start times of the operations subjected to the start-time, precedence and resource constraints so as to minimize the latency of the schedule.

Objective Function:

This is subjected to the below constraints:

Start-time constraints:

Sequencing/Data Dependency constraints:

Resource Constraints:

**MRLC using ILP:**

1. V is set of vertices corresponding to different operations = {0 (source),1,2,3,…n (sink)}
2. Calculate the earliest start-time tis using ASAP
3. The upper bound on latency is λu , as provided
4. Calculate the latest start-time using ALAP (λu)
5. A decision variable xil = 1, indicates operation “i” started in cycle l, and 0 otherwise
6. We find a solution to the objective function, which is a determination of the optimum number of resources for each operation subjected to the start-time, precedence and resource constraints (with number of resources unknown) and latency constraints.

Objective Function:, where ai is the number of resources of each type

This is subjected to the below constraints:

Start-time constraints:

Sequencing/Data Dependency constraints:

Resource Constraints:

Latency Constraints:

**MLRC using List Scheduling:**

Here, we continue with the algorithm as long as all the nodes are scheduled. We have a ready set that indicates the operations that can be scheduled in the current time step and a busy set which indicates operations which are still running in the current step. We select a subset of operations based on the mobility. The nodes with lower mobility have less flexibility so they get higher priority over others. The set of operations selected in the current time and the busy operations must be less than equal to the total number of resources available in the current time step.

LIST\_L (G(V, E), a) {

*l* = 1

repeat {

for each resource type k {

*Ul,k* = available vertices in V.

*Tl,k* = operations in progress.

Select *Sk* ⊆ *Ul,k* such that |*Sk*| + |*Tl,k*| ≤ *ak*

Schedule the *Sk* operations at step *l*

}

*l = l +* 1

} until *vn* is scheduled.

}

**MRLC using List Scheduling:**

We follow a similar approach as used in MLRC. Since, we have to minimize the number of resources we start with the number of resources set to 1. We select nodes from the ready set whose slack is 0 (slack = latest possible start time (computed by an ALAP schedule) and the index of the schedule step under consideration.), as they fall in the critical path, and so they must be scheduled in the current time step Scheduling such operations may require additional resources, thus the number of resources gets updated only at the time of need. The remaining operations are scheduled only if they do not require additional resources.

LIST\_R (G(V, E), λ’) {

*a* = 1, *l =* 1

Compute the ALAP times *tL*.

if *t0l < 0*

return (not feasible)

repeat {

for each resource type k {

*Ul,k* = available vertices in V.

Compute the slacks *{ si= ti*

*L - l,* ∀ *vi*∈ *Ul,k }*.

Schedule operations with zero slack, update *a*

Schedule additional *Sk* ⊆ *Ul,k* under *a* constraints

}

*l = l +* 1

} until *vn* is scheduled.

}

**PLOTS FOR MLRC using LIST SCHEDULING:**

Here, we have executed the program against different benchmarks to obtain the variation of latency with different resource combinations. We observe that with the increase in the number of resources, the latency reduces but it eventually saturates to a minimum value. Further increase in the number of resources has no effect in the latency value.

Benchmark 1: hal.dot

Benchmark 2: fir1.dot

Benchmark 3: cosine1.dot

Benchmark 4: cosine2.dot

**PLOTS FOR MRLC using LIST SCHEDULING:**

Here, we have executed the program against different benchmarks to obtain the variation of resources with different values of latency. We observe that with the decrease in the number of time steps, the algorithm can make use of more number of resources to schedule the operations. So, we observe a downward trend for the latency curve against the increase in number of resources.

Benchmark 1: hal.dot

Benchmark 2: fir1.dot

Benchmark 3: cosine1.dot

Benchmark 4: cosine2.dot

**Comparison of execution time of ILP vs List Scheduling**

* Gave all the resources as 10 for MLRC\_ILP and MLRC\_List
* Gave minimum latency as the maximum latency for MRLC\_ILP and MRLC\_List
* The following resource delays are used for rdf.txt (to reduce the computational time)
* Mul 🡪 3
* Add 🡪 2
* Sub 🡪 1
* les 🡪 2
* exp 🡪 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | MLRC ILP | MLRC\_List | MRLC\_ILP | MRLC\_List |
| hal.dot | **0.14sec** | **0.11sec** | **0.31 sec** | **0.11sec** |
| fir1.dot | **0.28sec** | **0.23sec** | **0.65sec** | **0.19sec** |
| cosine1.dot | **0.51sec** | **0.36sec** | **2.11 sec** | **0.43sec** |
| cosine2.dot | **0.51sec** | **0.49sec** | **16.54sec** | **0.45sec** |

**Comparison of ILP vs List Scheduling in terms of Number of resources (cost)**

The ILP based approach compared to List scheduling gives much more accurate results in terms of the number pf resources for some of the benchmarks.

We have used the following resource definition file to get the cost function.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Operation | Latency | DII | Area | Area |
| Mutliplier | mul | 30 | 15 | 29196 | 17 |
| Adder | add | 20 | 10 | 3077 | 2 |
| Subtractor | sub | 11 | 6 | 3077 | 2 |
| Comparator | les | 20 | 20 | 1771 | 1 |
| Exponent | exp | 20 | 20 | 3077 | 2 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | MRLC\_ILP | | | | |  | MRLC\_List | | | | | Results |
|  | Latency Input | Mul | Add | Sub | Les | Exp |  | Mul | Add | Sub | Les | Exp |  |
| hal.dot | 103 | 3 | 1 | 1 | 1 | 2 |  | 3 | 1 | 1 | 1 | 2 | Same |
| fir1.dot | 210 | 3 | **2** | 0 | 0 | 1 |  | 3 | **3** | 0 | 0 | 1 | ILP more accurate since adders needed is only 2 compared to 3 in List |
| cosine1.dot | 160 | - | - | - | - | - |  | 12 | 4 | 4 | 0 | 8 | ILP solver timed out |
| cosine2.dot | 160 | - | - | - | - | - |  | 12 | 4 | 4 | 0 | 8 | ILP solver timed out |

**CONCLUSIONS:**

We can conclude the following from the scheduling of data flow graph:

1. The ILP based approach consumes more time to calculate the results but the results are more accurate compared to List based approach.
2. With the increase in the number of resources, latency values goes down but it saturates to minimum value below which the increase in the number of resources has no effect in the latency.
3. With the increase in latency, the number of resources needed to compute goes down. We can schedule the operations with even one number of resources
4. Minimizing Latency under Resource constraints approach is effective for application when we have resource dominated circuits.