Karthik Narayanan, Santosh Madiraju

EEL6935 - Embedded Systems Seminar

TOPIC: HARDWARE – SOFTWARE PARTITIONING AND CO DESIGN PRINCIPLES

26TH FEB, 2013.

Efficient Search Space Exploration for HW-SW Partitioning

Hardware/Software Codesign and System Synthesis, 2004. CODES + ISSS 2004. International Conference on Computing & Processing (Hardware/Software) 2004 Page(s): 122 - 127

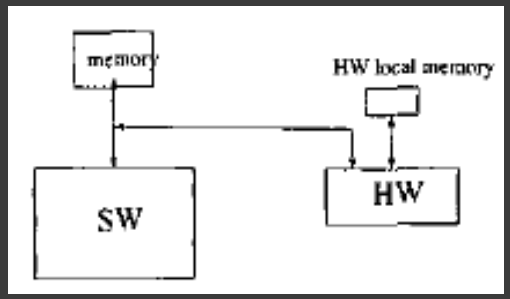
Introduction

- HW SW partitioning – key challenge in embedded systems.
- Issues addressed by this paper.
  - Large Design Space utilization
  - Scaling to Large Problem sizes.
- Minimizing the execution time of an application for a system with hard area constraints.

- Sequential application specified as a call graph DAG. (vertices, edges).
- Contributions made:
  - Updating the execution time change metric.
  - Cost function for Simulated Annealing (SA).
- Implementation compared with other similar algorithms.

Attributes and Assumptions

- Target Architecture – one SW processor and one HW unit connected by system bus.



- Assumptions:
  - mutually exclusive units
  - HW unit has no dynamic RTR capability

- Input – DAG
  - CG = (V,E)
  - Each partitioning object corresponds to a vertex (**vi € *V)***
  - Each edge (***eij € E)***represents a call or access to a callee vj from caller vi.

- Each edge eij has 2 weights (ccij, ctij) representing call count and HW-SW communication time.
- Each vertex vi has 3 weights (ti(s), ti(h), hi), representing execution time of a function on SW, on HW and area respectively.
- Partitioning attributes – (Tp, Hp) representing execution time and aggregate area mapped to HW under partitioning p.

Execution time change metric computation.

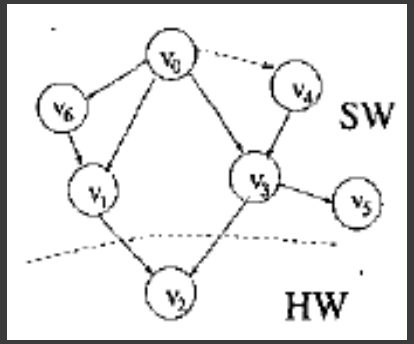
- Execution time of vertex vi – Ti(p)

$$T_i^P = t_i + \sum_{j=1}^{|C_i|} (cc_{ij} * T_j) + \sum_{j=1}^{|C_i^{diff}|} (cc_{ij} * ct_{ij})$$

  - Ci is set of all children of vi
  - Cdiff set of all children of vi mapped to a different partition.

- ~Pi represents the change in execution time when vi is moved to a different partition.

- A simple call graph is as shown.



- Earlier approach – when vi is moved, all ancestors need to be updated (all the way to the root).
- In figure, consider v2 to be initially in SW. Now v2 moved to HW.
  - Execution time changes due to HW-SW communication on edges (v3,v2) and (v1, v2).
  - It would appear that related metric for v0, v4 and v6 would need to be updated.
  - But proved that when vi is moved, ~Pj needs to be updated on if there is an edge for (vi,vj).

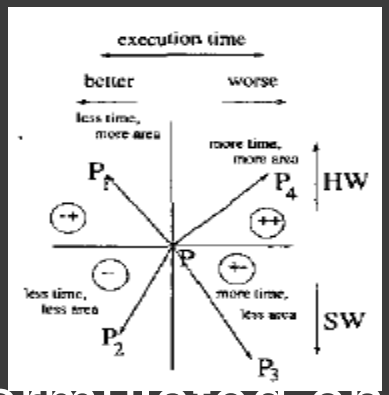
Simulated Annealing

- Move based algorithm.
- Essentially tries to find an optimal solution to a “hard” such as partitioning.
  - Systems with minimal energy is the optimal solution.
- Update the execution time for new partition by updating only the immediate neighbors of a vertex.
- SA algorithm – rapid evaluation of search space.
  - Indegree and outdegree of call graph is expected to be low and so average cost of a move is low.

Cost function of SA

- Force algorithm to accept bad moves when far away from objective
  - Guides it to potentially interesting design points.
- Force the algorithm to probabilistically reject some good moves
  - That would always be accepted by most heuristics.
- Cost function defined on parameters that change for a given move.
  - Execution time: same as execution time change metric for a moved vertex
  - HW area: (hi) for SW->HW and (–hi) for HW->SW

- Figure gives an idea of all the regions a partition can occupy.

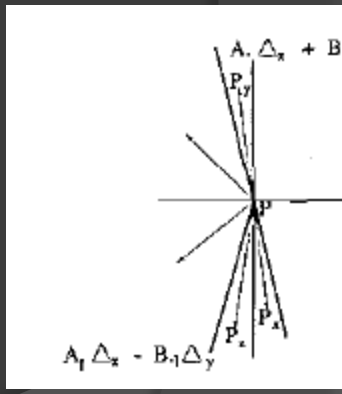


- A weighted cost function is formulated on which regions a partition is allowed to occupy and which regions it is rejected

- Dynamic Weighting factor for cost functions.
  - To better guide the search.
  - To avoid boundary violations

Example

- Partition P where few components are mapped to HW and execution time is expected to be closer to SW execution time. Cost function is biased as follows.
  - Provide additional weightage to moves like Px where execution time deteriotes slightly but frees up a large amount of HW area.
  - Reduce weightage on Py which improve execution time slightly but consume additional HW area
  - Reduce moves like Pz that improve execution time slightly but free up large HW area.



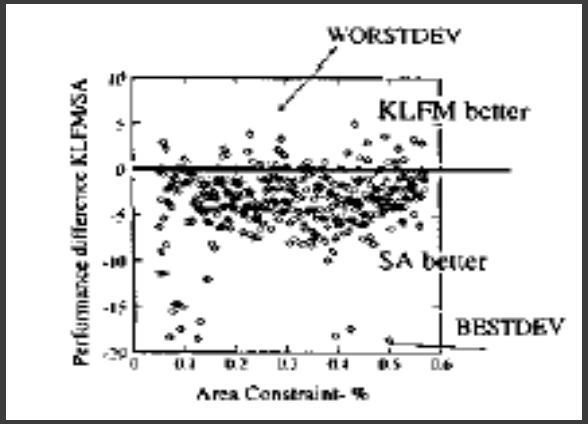
Experiment

- Comparison made between SA and KLFM algorithm
- Record program execution times of SA algorithm (with the new cost function) vs. KLFM algorithm.
- Graphs generated by
  - Varying indegree and outdegree
  - Varying number of vertices
  - Varying CCR(Communication to Computation Ratio)
  - Varying area

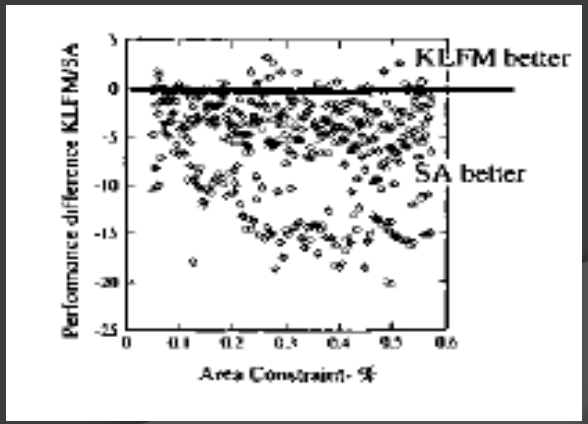
- Data was generated for over 12000 individual runs of SA with following configurations.
  - Max indegree and outdegree set to 4. Graph size (number of vertices) and CCR were selected accordingly.
  - Area constraint varied as a percentage of aggregate area needed to map all the vertices to HW.
  - Vary the max indegree and outdegree set earlier.
- Performance difference has been calculated by T(kl) – T(sa)/T(kl)*100

Results

- Fig 1: v=50, CCR=0.1



- Fig 2: v=50, CCR=0.3



Aggregated data

Graph type	BestDev (%)	WorstDev (%)	Avg (%)	SA rt.	KLFM rt.
v20	-24.9	12.3	-4.17	.07	.05
V50	-22.9	6.7	-5.75	.08	.05
V100	-18.2	5.7	-5.47	.1	.07
V200	-13.9	4.3	-3.74	.19	.11
V500	-16	6.8	-4.53	.25	.48
V1000	-13.7	6.4	-4.17	.36	1.6

Conclusion

- Two contributions made:
  - Updating the execution time metric
  - New cost function.
- Generate partitions with execution times which are often 10% better over KLFM.
- Quick processing of graphs with large vertices.

Limitations and Future work:

- Simple additive HW area estimation model – does not consider resource sharing.
- Can be extended to consider systems with concurrency, looking into scheduling issues during simulation.

Integrating Physical Constraints in HW-SW Partitioning for Architectures with Partial Dynamic Reconfiguration.

Sudarshan Banerjee, Elaheh Bozorgzadeh, Nikil Dutt

IEEE Transactions on VLSI systems, VOL.14, No. 11, Nov 2006.

.

INTRODUCTION

- Motivation: HW-SW Partioning for Partially Dynamic Reconfigurable Systems
- Major Challenges
  - Design Space Exploration
  - Placement
  - Scheduling
- Proposed Approach
  - Integer Linear Programming (ILP)
  - HW-SW Partioning Heuristic based on KLFM Algorithm

INTRODUCTION

- Dynamic Reconfiguration
  - Provides the ability to change the hardware configuration during application execution.
  - Also provides means to reduce reconfiguration overhead by enabling overlap of computation with reconfiguration.

- Generally HW-SW partioning optimizes design latency and is followed by physical design.
- Challenges
  - Placement Infeasibility
  - Heterogeneity

Challenges

- Placement Infeasibility
  - RTR capability imposes strict linear placement constraints
  - Schedule has to be aware of the exact physical location of the task
- Heterogeneity
  - FPGA consists of Heterogeneous modules .E.g.- DSP blocks, BRAM's etc..
  - Dedicated Resources lead to improved efficiency
  - Area –Execution time trade off

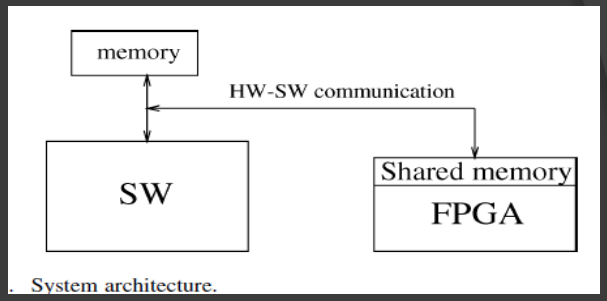
Heterogeneity Challenges

- Additional Challenges
  - Feasibility Issue, exact approach
    - ILP approach incorporates physical layout into HW-SW partitioning problem.
  - Heuristic Approach
    - KLFM based heuristic which considers detailed linear placement along with scheduling.
  - Heterogeneity
    - Arises due to considering placement and multiple task implementation

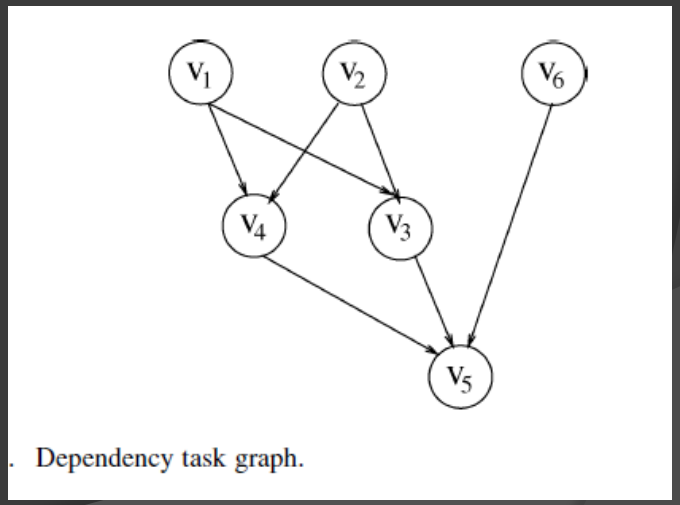
Problem Description & Target Architecture

- HW-SW partitioning of an application on the target architecture is considered
- Application is specified as a task dependency graph
  - Each vertex represents a task
  - Each edge represents data communicated

- Target Architecture
  - Software Processor
  - Dynamically Reconfigurable FPGA with PR
  - Processor and FPGA communicate via a system bus
  - Shared Memory



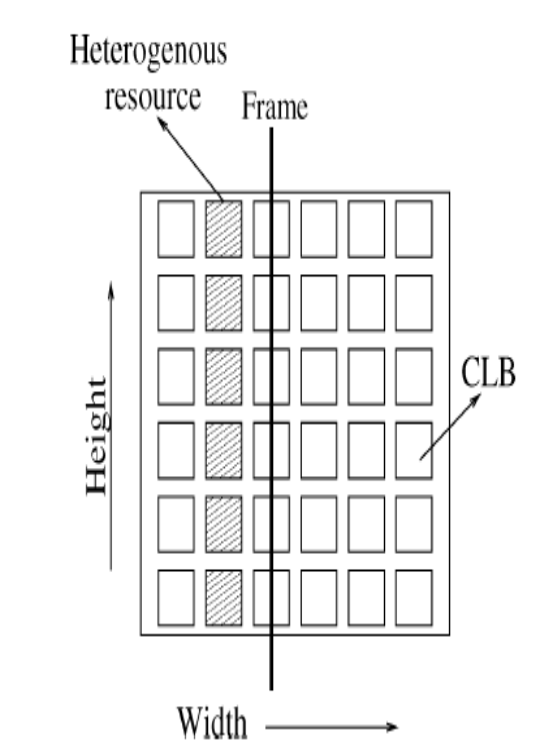
. System architecture.



. Dependency task graph.

Architecture

- Memory Accesses for tasks on processor restricted to local memory
- Communication overhead for transfer of data incurred
- HW-SW communication delay should be considered
- FPGA Hardware unit has a set of CLB's in a 2-D matrix
- Specialized resource columns are distributed between CLB
- Reconfiguration time of a task is proportional to the number of columns occupied by the task



Heterogenous FPGA with partial RTR.

Constraints

- Device Constraints
  - Columnar implementation of dynamic tasks
  - Single reconfiguration process
  - Location of specialized resource columns
- Each implementation of task has few parameters
  - Execution time
  - Area occupied in columns
  - Reconfiguration delay

Issues with Scheduling

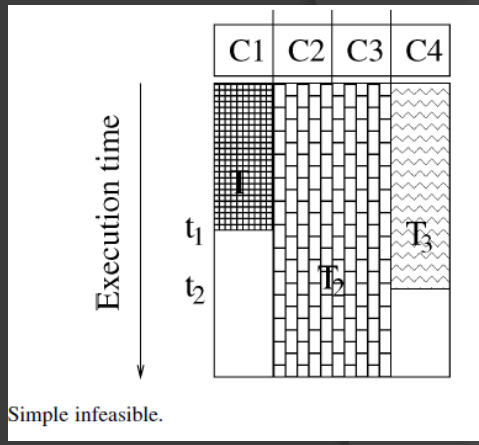
- Criticality of Linear Task Placement
  - Each task is implemented on adjacent columns
  - Linear Task Placement problem
    - Finding a feasible placement on the hardware for a scheduled task under resource constraint and size
    - Two Cases
      - Each task occupies an identical number of columns- solution is simple
      - Each task occupies different number of columns is – solution is complex and linear placement feasibility is not guaranteed even with an exact algorithm.

Issues with Scheduling

- There are schedules which cannot be placed by optimal placement tools

- Heterogeneity Considerations:
  - Resource columns are available at fixed locations
  - HW execution time and area vary with placement

- Scheduling for configuration Prefetch
  - Separating a task into reconfiguration and execution components
  - Reconfiguration component is not constrained by dependencies which poses a challenge.



Simple infeasible.

Approach

- Task Graph with “n” tasks and each task occupies certain number of columns
- 1 SW and Hw unit with m HW columns
- Each edge has a weight representing HW-SW comm’n time
- Each task corresponding to a vertex has 4 weights
- Objective is to obtain an optimal mapping with minimal latency when FPGA has most columns available.

ILP Formulation

$x_{i,j,k} = 1$, if task T_i starts execution on FPGA at time-step j, and k is leftmost column occupied by T_i;
$= 0$, otherwise.

$y_{i,j} = 1$, if T_i starts execution on processor in time-step j;
$= 0$, otherwise.

$r_{i,j,k} = 1$, if reconfiguration for task T_i starts at time-step j, and k is leftmost column occupied by T_i;
$= 0$, otherwise.

$in_{i_1,i_2} = 1$, if tasks T_{i_1} and T_{i_2} are mapped to different computing units and, thus, incur a HW-SW communication delay;
$= 0$, otherwise.

Constraints

- Uniqueness Constraint –Each task can start only once

$$\sum_j \left(y_{i,j} + \sum_k (x_{i,j,k}) \right) = 1$$

- Processor resource Constraint

$$\sum_i \sum_{m=j-t_i^s+1}^{j} (y_{i,m}) \leq 1$$

- Partial Dynamic Reconfiguration Constraints
  - Each task needs at most one recon

$$\sum_j \left(y_{i,j} + \sum_k (r_{i,j,k}) \right) \leq 1$$

  - Resource Constraints on FPGA
  - At every time step , at most single task is being reconfigured and mutual exclusion of execution and reconfiguration of every column

ILP Formulation

- If reconfiguration is needed for task , execution must start in the same column and only after the reconfiguration delay
- *A task can start execution* only if there are sufficient available columns to the right
- Interface Constraints
- Precedence constraints
- Tighter placement constraints
- Tighter timing constraints

Heuristic Approach

- KLFM based Heuristic
- Generic moves between tasks are defined instead of restricting to either HW or SW
- HW-HW and HW-SW are also taken into consideration
- Scheduling
  - The schedule quality depends on priority assignment of nodes
  - Scheduler is aware of communication costs
  - Simultaneous scheduling and Placement
    - For each schedulable task,

      compute (EST), earliest start time of computation

      (EFT), earliest finish time of computation

      Choose task that maximizes (EST, longest path, area, EFT)

Priority Function

- Key parameters of Priority Function are
  - Earliest Computation Start Time (EST)
  - Earliest Finish Time (EFT)
  - Task Area
  - Longest Path through the task

F(EST, longest path, area, EFT)

EST Computation

Code Segment 3: Compute EST for task bound to FPGA

```
find earliest time slot where task can be placed
reconfig start = earliest time instant space and reconfig
      controller are simultaneously available.
if ((reconfig start + reconfig time) < dependency time)
      // reconfiguration latency hidden completely: possibility
      // of timing gap between reconfig end and execution start
   EST = earliest time parent dependencies satisfied
else     // not possible to completely hide latency
   EST = end of reconfiguration
```

- The EST computation, embeds the placement issues and resource constraints related to reconfiguration

- Heterogeneity
  - A simple type descriptor is added to every column in resource description.
  - Resource queries check the type descriptor of a column while looking for available space.
  - Some initial preprocessing is done to make searches more efficient.
- Worst Case Complexity
  - Simplistic implementation of the EST computation has a worst case complexity O(n2*C)
  - Worst Case complexity of each list scheduler is O(n4*C)
  - The list scheduler is called O(n2) times
  - The overall worst case complexity is O(n6C)

Experimental Setup

- Area and timing data for key tasks like DCT and IDCT, was obtained by synthesizing tasks under columnar placement and routing constraints on the XC2V2000
- Tasks implemented on software are found to be 3-5 times slower than that on hardware

BASIS FOR NUMERICAL DATA

HW unit	similar to XC2V2000, organized as a CLB matrix of 56 rows and 48 columns
SW unit	PowerPC processor operating at 400 MHz
Communication bus	64-bit wide PLB operating at 133 MHz
Frames/CLB column	22 frames (total 1456 frames on the entire device)
Reconfiguration time	17.01 ms for full device (SelectMAP port@50 MHz)
Reconfig frequency	66 MHz (maximum suggested)
Reconfig delay/column	22/1456 * 17.01 * 50/66 = 0.19 ms

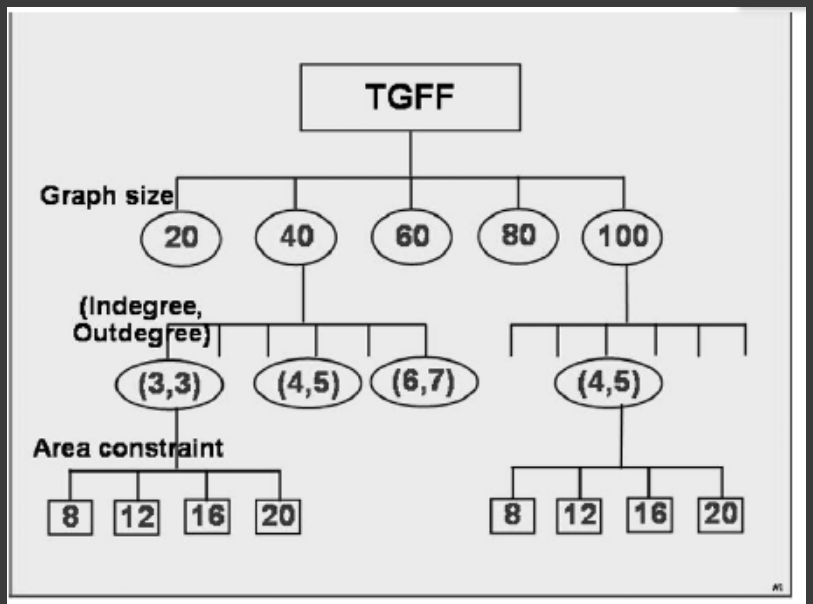
Experiments on Feasibility

- The Test cases are small graphs between 10-15 vertices
- Number of columns available is approx 20-30% of total area of all tasks
- One unit of time is reconfiguration time for a single column

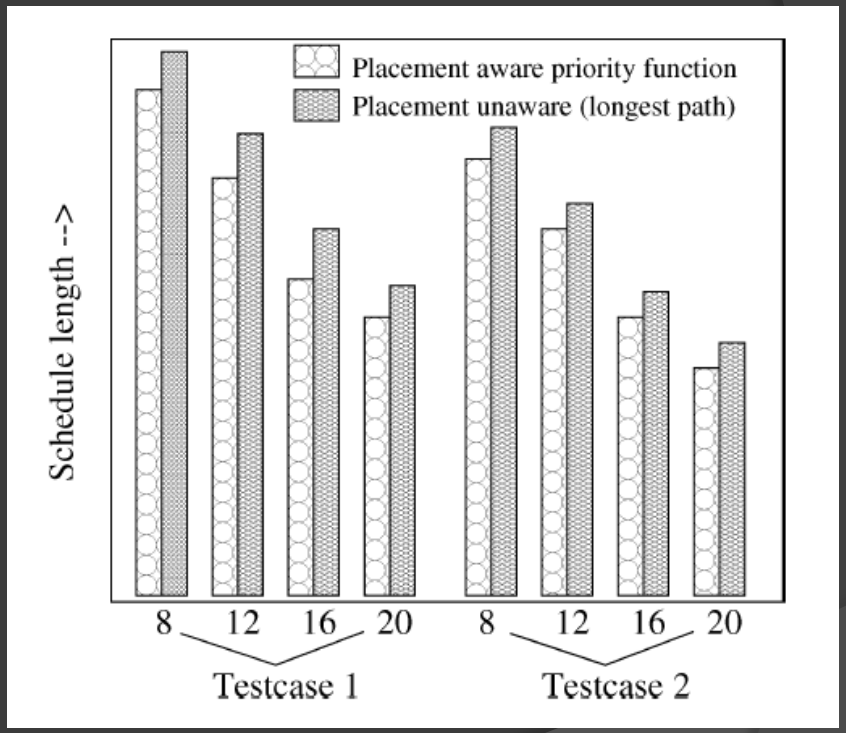
FEASIBILITY RESULTS AND HEURISTIC QUALITY FOR SMALL TESTS

Testcase	Placement-Unaware		Placement-Aware	
	T_{opt}^{area}	Feas.	T_{opt}	T_{heu}
tg1	10	Y	10	11
tg5	25	**NO**	26	26
Mean-value	21	Y	21	21
tg7	20	Y	20	20
tg10	27	**NO**	28	29
FFT	25	Y	25	25
tg11	36	**NO**	38	41
tg12	14	**NO**	15	18
4-band eq	27	Y	27	27

Experiments on Heuristic Quality



Synthetic experiments.



Results

AGGREGATE IMPROVEMENTS IN SCHEDULE LENGTH

Test group	Few cols (8,12)	More Cols (16,20)	Avg gain
v20	6.07%	6.79%	6.43%
v40	5.44%	10.64%	8.04%
v60	10.36%	10.56%	10.46%
v80	11.68%	13.64%	12.66%
v100	16.68%	19.09%	17.89%
Avg gain	10.05%	12.15%	11.09%

$$\text{Gain} = 100 * (T_{\text{longest\_path}} - T_{\text{heu}})/T_{\text{heu}}$$

RUNTIME OF PROPOSED APPROACH

Test group	Average run-time(s) 20 columns
v20	0.2
v40	2.0
v60	22
v80	90
v100	180

Conclusion

- Physical and architectural constraints imposed on dynamically reconfigurable architectures by PR was explained in detail.
- An exact approach based on ILP was formulated
- Ignoring linear task placement constraints can result in schedules which are optimal but are infeasible.
- Simultaneous placement of tasks along with scheduling
- Placement aware HW-SW approach based on KLFM heuristic was proposed
- Heuristic simultaneously partitions, schedules and performs a linear placement of tasks on the device.
- A wide range of experiments were conducted which validates the approach.

Improvements & Future Work

- An assumption is made that there is sufficient bandwidth available to perform task concurrently which may not be true always.
- Though the ILP takes into consideration of heterogeneous modules, the heuristic approach considers only homogeneous modules.
- Due to availability of sophisticated algorithms and data structures complexity of the algorithm can be reduced further.