Accelerating FPGA Routing Through Algorithmic Enhancements and Connection-aware Parallelization

https://core.ac.uk/download/pdf/333624203.pdf

Core algorithm is the pathfinder algorithm

Connection based router strategy

- rip up and reroute a net by rerouting congested connections instead of rerouting all of the nets (including uncongested connections)

Cost function

Path cost of node n

$$f(n) = c_{prev}(n) + c(n) + \alpha \times c_{exp}(n)$$

Previous cost

- sum of congestion cost (possibly delay) of all previous nodes upstream on this path

Current cost

- base cost
- present congestion
    - updated after every routed connection

$$p(n) = \begin{cases} 1, & cap(n) > occ(n), \\ 1 + p_f(occ(n) - cap(n) + 1), & \text{otherwise.} \end{cases}$$

- historical congestion
  - updated after every iteration

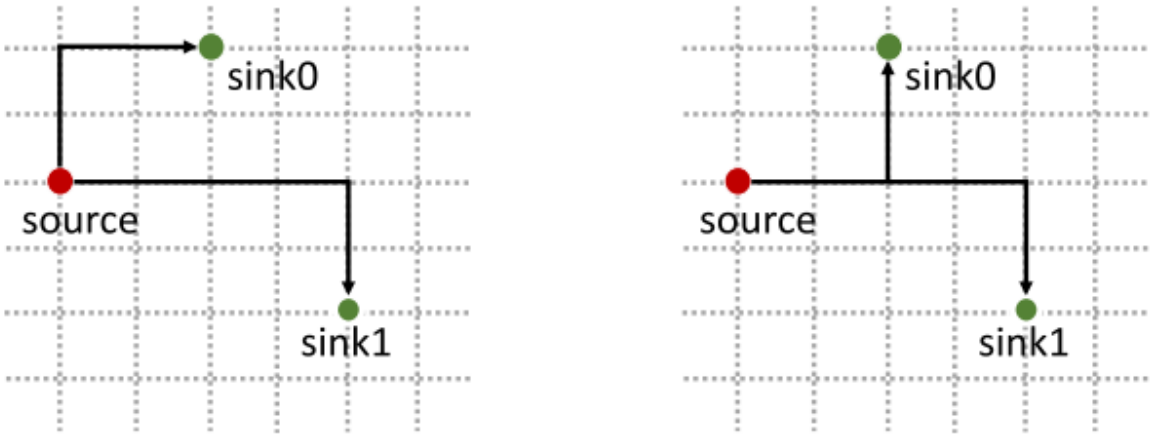
$$h^i(n) = \begin{cases} 1, & i = 1, \\ h^{i-1}(n), & cap(n) \geq occ(n), \\ h^{i-1}(n) + h_f(occ(n) - cap(n)), & \text{otherwise.} \end{cases}$$

- node sharing factor
  - equals to # of connections of the same net that uses this node
  - used to promote resource sharing

Expected cost

- based on estimated wire segment from node n to sink
  - include orthogonal path as well
  - divided by node sharing factor
- sink/ipin cost included

Resource Sharing



(a) After iteration 1 (b) After iteration 2

Non-sharing problem worsens as Manhattan distance of connection increases, and as number of equivalent shortest/min cost paths increases

Negotiated sharing

$$c(n) = \frac{b(n) \cdot p(n) \cdot h(n)}{1 + share(n)}.$$

- The cost of a node for a connection should be lower in case it is already being used by other connections of the same net
- Doesn't work well if we are routing the first connection of the net because it doesn't know how the other connections will be routed

Adapted wirelength driven cost

Adaptive base cost

- Base cost of of different wire types should be different. Short wires should have smaller cost than longer wires. This would give router priority in exploring short wires first, then long wires if necessary.

Expected distance to sink of connection

expected cost function:

$$c_{exp}(n) = \frac{\delta_{same} \cdot \bar{c}_{same}}{1 + share(n)} + \frac{\delta_{ortho} \cdot \bar{c}_{ortho}}{1 + share(n)} + b(ipin) + b(sink).$$

Based on Manhattan distance and cost per distance

- Hard to know exactly what set of wires it will use
- Cost per distance is average of a unit distance cost over all wire segment types, taking into account number of wire segments of each type (not sure exactly what

this means)

Bias Cost

Purpose is to help router choose a good initial path for its first connection by pulling a path towards the geometric center of the net. This cost should be small as it is only meant to be a tiebreaker

Modifies the current cost of node n

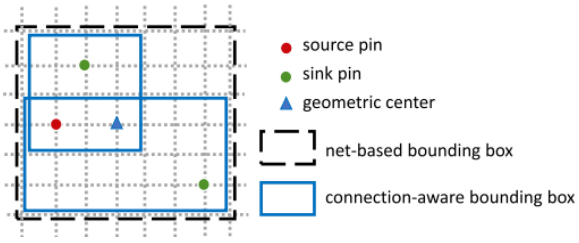
$$c(n) = \frac{b(n) \cdot p(n) \cdot h(n)}{1 + share(n)} + c_{bias}(n),$$
$$c_{bias}(n) = \frac{b(n)}{2 \cdot fanout} \cdot \frac{\delta_{m,c}}{HPWL}.$$

$\delta_{m,c}$ is the Manhattan distance to the geometric center of the net. This distance is then normalized by dividing the HPWL of the net.

$\frac{b(n)}{2fanout}$ is to scale the bias cost down.

NOTE: Should it scale up with fanout size, as resource sharing opportunities become greater

Geometric center calculation: (N is number of fanouts)



$$x = \frac{1}{1+N} \cdot \left(x_{source} + \sum_{i=0}^{N-1} x_{sink_i} \right),$$
$$y = \frac{1}{1+N} \cdot \left(y_{source} + \sum_{i=0}^{N-1} y_{sink_i} \right).$$

NOTE: single geometric center may not be suitable for high fanout nets

Timing driven

Need to compute connection criticality based on slack and maximum delay in the circuit. Critical connections will focus on optimizing for delay. Non-critical connections will optimize for minimum wire-length.

Buffered routing switches

For unbuffered switches (pass transistors), the capacitance and resistance of a wire is dependent on downstream capacitance and upstream resistance of routing resources along the connection. More difficult to model. For buffered switches, no dependency.

Cost function (timing driven)

Current node cost:

- Critical connections care more on node delay
- less critical connections care about congestion

$$c(n) = (1 - f_{crit}) \cdot c(n)_{wld} + f_{crit} \cdot T_{del},$$

Expected cost:

- non-critical connections, cost is wire-length driven
- critical connections, cost is delay driven
- Expected delay cost is based on Manhattan distance and average delay cost per distance
- β smaller ⇒ slower, better Fmax (optimal 0.7)
- α smaller ⇒ slower, better Fmax (optimal 1.4)

$$c_{exp}(n) = (1 - f_{crit}) \cdot \alpha \cdot c_{exp,wld}(n) + f_{crit} \cdot \beta \cdot c_{exp,td}(n),$$

$$c_{exp,td}(n) = \delta_{same} \cdot \bar{T}_{same} + \delta_{ortho} \cdot \bar{T}_{ortho}.$$

Initial connection criticality

Estimate initial criticality based on placer estimations so that only critical long paths are driven by delay cost.

Fix illegal routing tree

When net has high and low criticality connections, its possible to have illegal routing tree with nodes driven by multiple sources. This can be fixed by rerouting all illegal connections

Connection Aware BBox

Smaller compared to net-based BBox

- Beneficial for parallelization and run-time

Computed based on source, destination, and geometric center points

3 tile buffer is added to BBox to allow more resource sharing

- more search space → run-time/quality trade-off

Experimental result: enhanced CRoute vs VPR 7.0.7

- CRoute improves wirelength and critical path delay by 10% and 7% respectively
- CRoute improves run-time by 3.5 times
  - mostly due to reduced efforts in re-routing because CRoute performs partial rip-up
- CRoute uses 1% more short wires and significantly reduce long wire usage
  - due to adaptive wire segment base cost, long wires only used by critical connections

Parallelization

Motivation/goal:

- Take advantage of hardware capabilities (multi-cores)
- Take advantage of spatial separation of connection/net by BBox
  - exploration is done only within boundaries of BBox

- Produce deterministic results
- Minimal impact to other QoQ metrics

Previous works (comparisons are made against VPR 7.0 unless noted otherwise)

- Gort and Anderson → Recursive bipartitioning technique
- Shen and Luo → partitioning based parallel router
    - nets are partitioned into 3 subsets by cutline (2 sets are cleanly separated by cutline, 1 set crosses the cutline)
    - the set that crosses the cutline can be further partitioned
    - 7X speedup, 10% wirelength increase with 32 processes
- Hoo and Kumar → another partitioning based parallel router ParaDRo
    - 3 phases
        - lower BBox expansion factor to reduce overlaps
        - nets are assigned to nodes of the slicing tree
        - nets within a node are further decomposed, where high fanout nets are reduced to source-sink connections where the BBox is further reduced
            - essentially, multi-fanout nets are modelled as set of virtual single fanout nets
            - routing path is limited to perimeter of BBox
    - 5.4X speedup with 8 threads
- Wang et al → ParRA, hybrid partitioning + parallel routing process based on authors' serial router
    - nets are partitioned into two kinds of subsets
        - conflict free subsets
            - nets spanning different regions and independent of others in the same subset
        - local subsets

        - nets fitting entirely in one region
    - conflict free subsets are routed one by one, with nets in each subset routed in parallel
    - local subsets are routed after conflict free subsets
        - local subsets are routed in parallel, with nets in each subset routed in series
    - 1.6X, 2.7X, 3.9X, 5X speedup with 2,4,8,16 threads compared against authors' serial router

The novelty of this paper is on using connection based geometric center adjusted BBox while incorporating the ideas of above works mentioned.

Sample partition cutline & slicing tree:

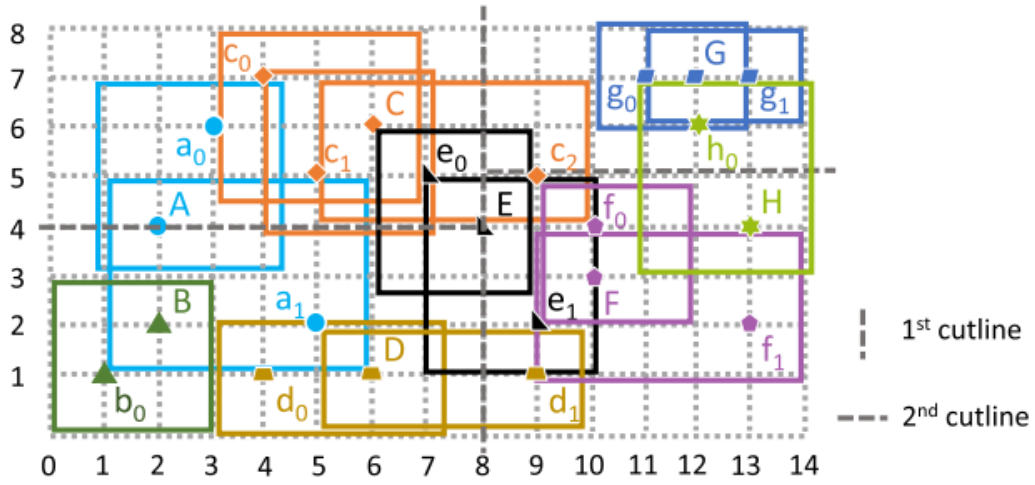


Fig. 6. An example of connection-aware recursive bi-partitioning with load balance in terms of the number of connections where: callouts in different shapes and colors represent different nets' terminals; sources and sinks of nets are named after capital letters and corresponding lower case letters, respectively; each solid line rectangle indicates the connection-aware routing bounding box of a source-sink connection.

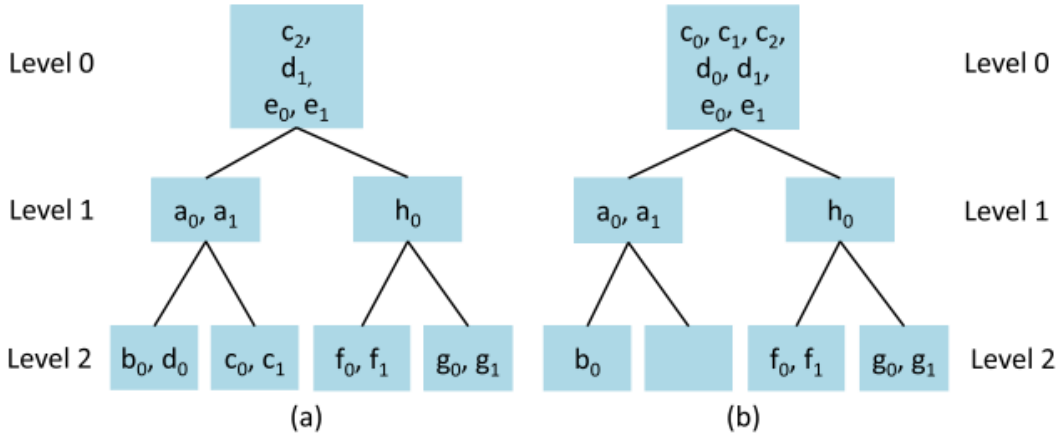


Fig. 7. Corresponding slicing trees of the (a) connection-aware and (b) net-based spatial partitioning for four threads.

Key points:

- Cutline strategy is to balance the left and right subtree load
- Number of leaf nodes in slicing tree is equal to the number of threads
- The slicing tree is processed in top-down fashion
  - Each node at every level is processed in parallel
  - Connections in each node are routed in series
  - we do not move the the next level unless current level is all processed

Terminology:

- Root node: level 0 node
- intermediate nodes: all nodes in the three excluding root and leaf nodes
- leaf nodes: nodes at bottom of tree

Ideally, the workload is distributed over leaf nodes, to maximize the parallelization speedup

Metrics to evaluate load balancing

Workload at each level is the maximum node load at the level

$$load_{level_i} = \max_{0 \le n < 2^i} \{load_n\},$$

With this definition, we can define the intermediate load and tree load

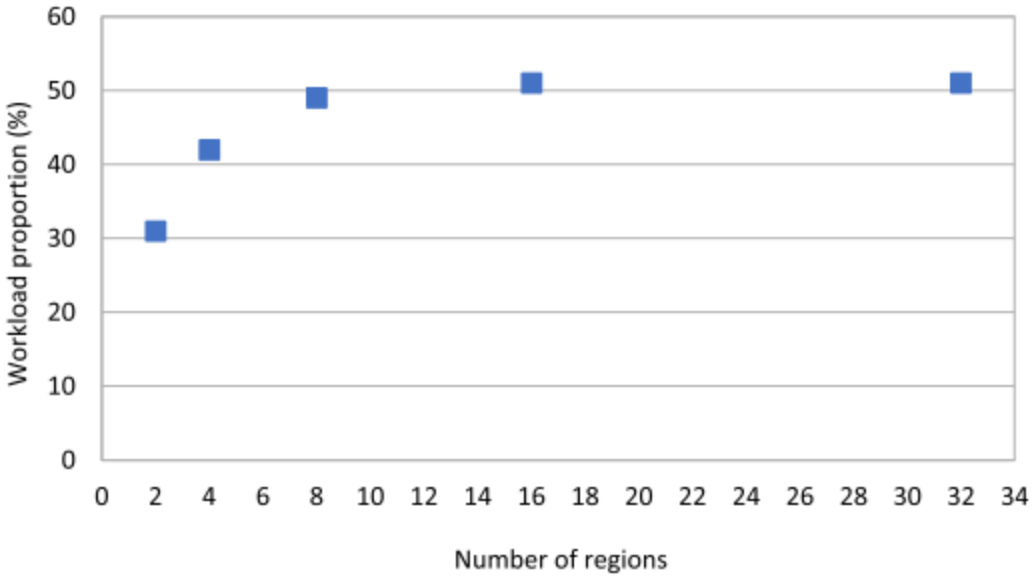
$$load_{inters} = \sum_{i=1}^{I-1} load_{level_i},$$

$$load_{tree} = \sum_{i=0}^{I} load_{level_i}.$$

High fanout nets have high chance of being placed in root node

- this can dominate total run-time

Based on paper's experimental data for net based BBox partitioning, as we increase the # of regions obtained via partitioning, the root work load increases and saturates ~50%. which is bottleneck for parallelization.



With connection based BBox partitioning (finer granularity BBox), paper achieves average 28% reduction in root work load. Intermediate work load also reduces while leaf workload increases.

Runtime comparison between net based BBox and connection based BBox

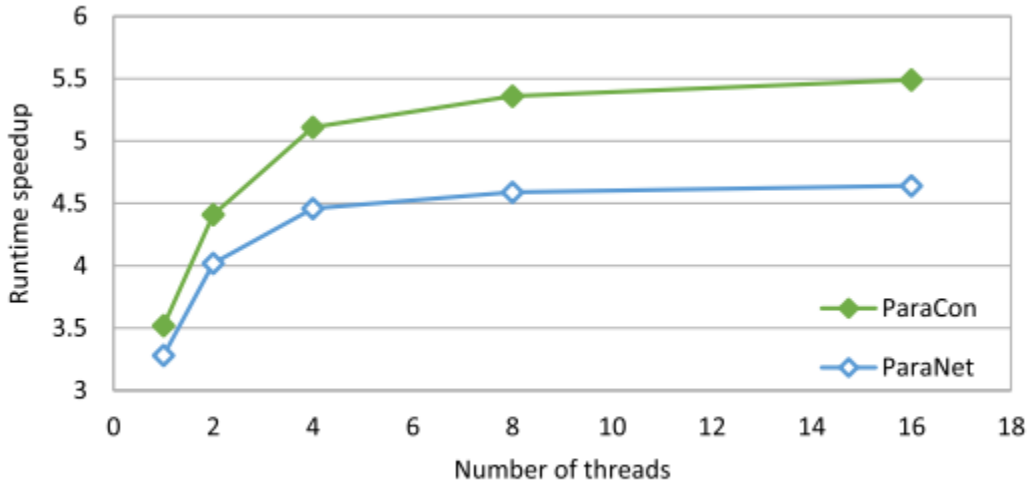
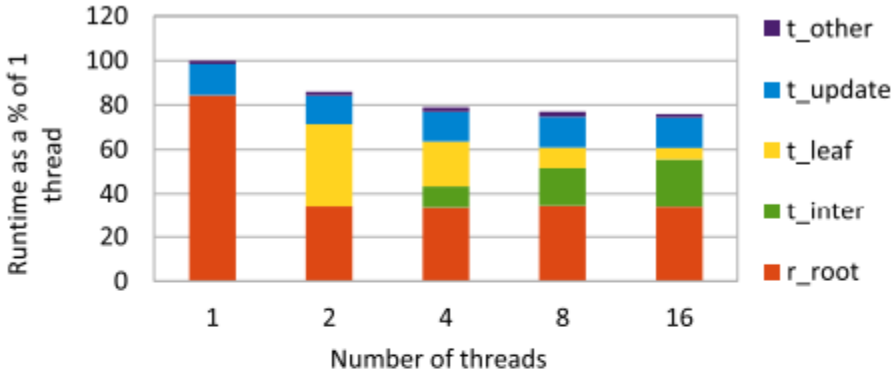
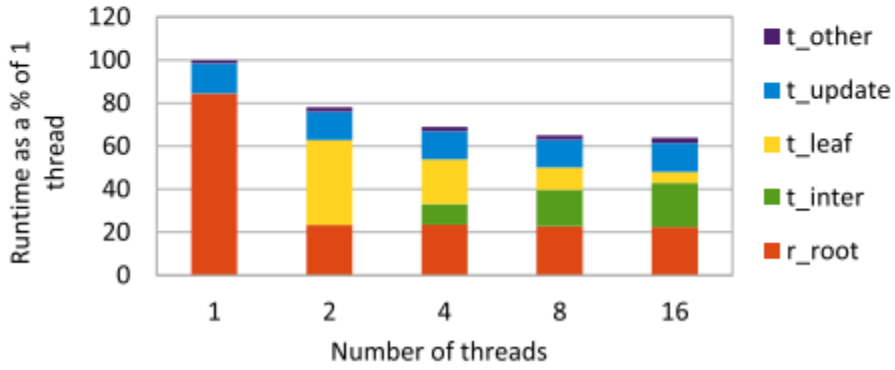


Fig. 11. Runtime speedups by ParaNet and ParaCon vs. VPR.

ParaCon vs ParaNet Run-time distribution across the slicing tree: Root/intermediate/leaf



(a) ParaNet



(b) ParaCon

key observations:

- number of intermediate workload increase as # threads increase
- ParaCon has less root workload

Improved Load Balancing Via Further Partitioning

Within the root node or any intermediate node, we can perform the same bipartitioning to further parallelize the node.

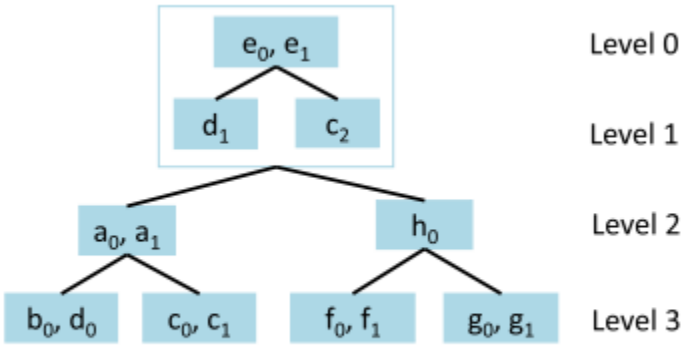


Fig. 14. An example of the slicing tree with further splitting.

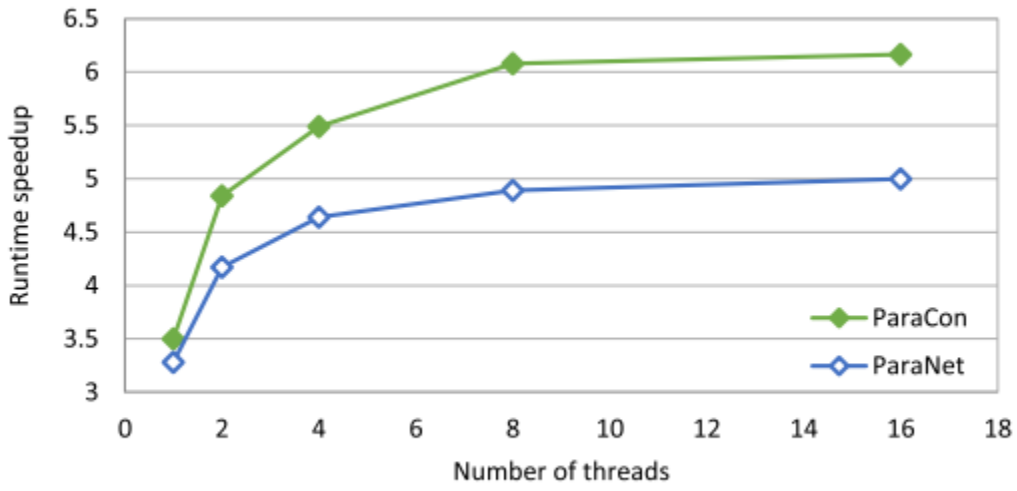


Fig. 16. Runtime speedups of ParaNet and ParaCon with further splitting vs. VPR.