acceleration

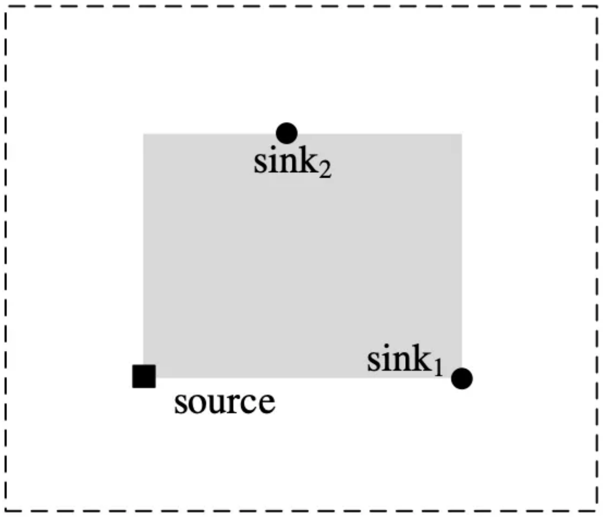
Coarse–grained Parallelization

Geometry–level Parallelism

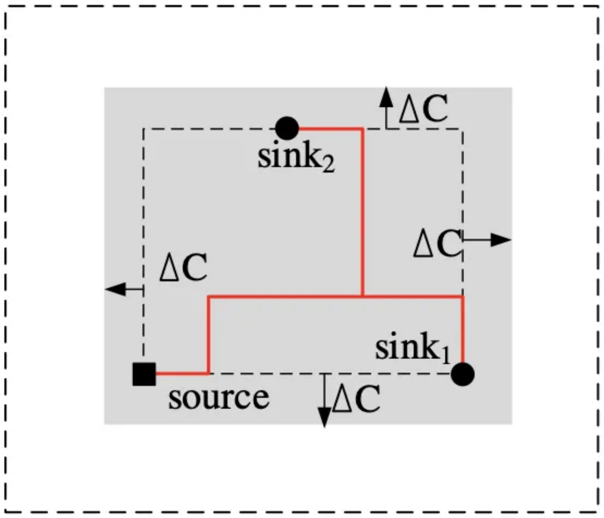
Connection–level Parallelism

Multi–core Accelerated

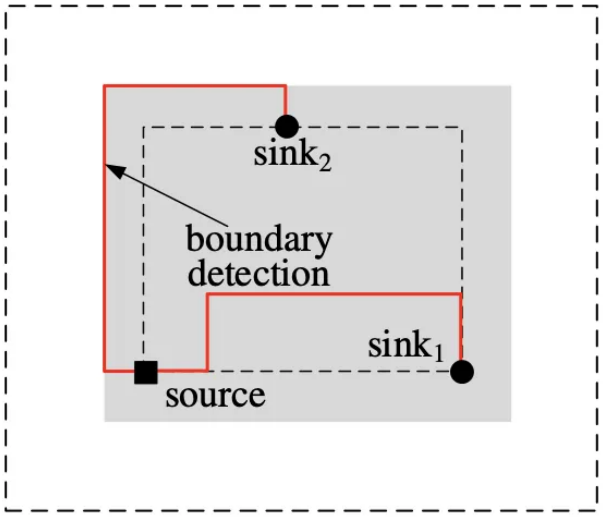
- Shen M, Luo G. "Accelerate FPGA routing with parallel recursive partitioning," ICCAD 2015.
    - Recursive partition using dynamic programming: horizontal and vertical
    - Coarese–garined parallel: Nets in different region can be routed simultaneously
    - Maze Routing
    - MPI
- Shen M, Luo G. "Corolla: GPU–Accelerated FPGA Routing Based on Subgraph Dynamic Expansion," FPGA 2017.
    - Obtain the net bounding box before routing
    - estimate the initial coverage that provides most nets a sufficiently large subgraph to route, and
    - perform boundary detection
    - dynamic expansion to guarantee that the routing subgraph is eventually large enough

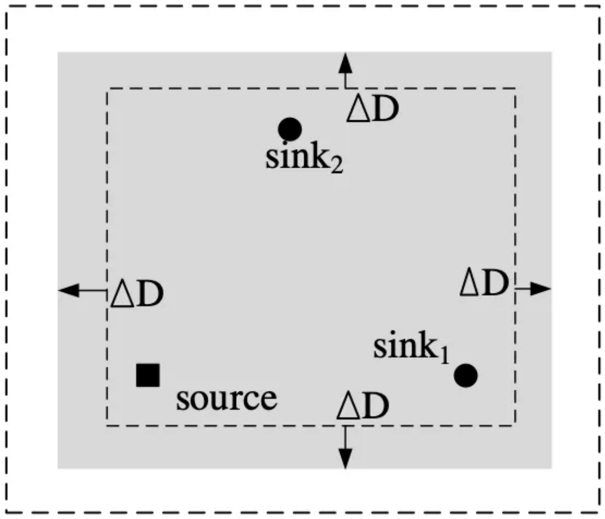


(a) Bounding box



(b) Initial coverage

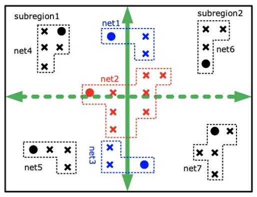




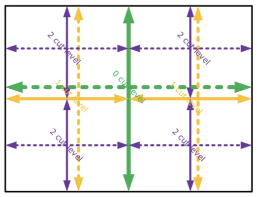
vpr/src/route/route_common.cpp C++

```
t_bb load_net_route_bb(const Netlist<>& net_list,
                       ParentNetId net_id,
                       int bb_factor) {
    /*
     * This routine loads the bounding box used to limit the space
     * searched by the maze router when routing a specific net. The search is
     * limited to channels contained with the net bounding box expanded
     * by bb_factor channels on each side.  For example, if bb_factor is
     * 0, the maze router must route each net within its bounding box.
     * If bb_factor = max(device_ctx.grid.width()-1, device_cts.grid.height() - 1),
     * the maze router will search every channel in
     * the FPGA if necessary.  The bounding box returned by this routine
     * are different from the ones used by the placer in that they are
     * clipped to lie within (0,0) and (device_ctx.grid.width()-1,device_ctx.grid.height()-1)
     * rather than (1,1) and (device_ctx.grid.width()-1,device_ctx.grid.height()-1).
     */
    ...
}
```

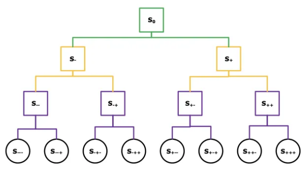
- Shen M, Luo G, Xiao N. "Combining static and dynamic load balance in parallel routing for FPGAs," TCAD 2020.
    - Transaction version of ICCAD15
    - load-balance partitioning
    - Recursive partition based on locations into 3 sets: S+, S−, S0
    -



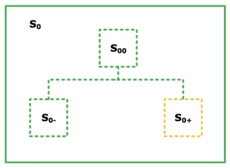
(a) Subregions and nets after partitioning



(b) Recursive partitioning



(c) A tree structure for task assignment



(d) Parallelize the nets across subregions

    - Each subregion can be searched in parallel coarse-grained parallel pathfinder

Python

```python
while True: # see "while (routeIteration < config.getMaxIterations()) {...}" in routeIndirectConnections()

    update_count = [0 for _ in len(nodes)]
    while !M.is_empty():
        read_lock()
        (s,t) = get_net(thread_id, M) # for (net in Nets)
        read_unlock();

        if !is_routed(s,t) || is_congested(s,t) || is_critical(s,t): # see shouldRoute(...)
            ripup(s,t) # see prepareRouteConnection(...)
        else:
            continue

        min_heap.push((null, s), 0)
        routes = ()

        while min_heap.isEmpty() == False: # see "while ((rnode = queue.poll()) != null) {...}" in routeConnection(...)
            (edge = (prev, curr), curr_dist) = min_heap.pop()
            if curr in routes: continue
            update_present_cost(prev)
            routes.insert(edge)

            update_count[curr]++
            if update_count > 3: continue
            write_lock();
            for _net in curr.nets:
                M.push_back(_net)
            write_unlock();

            if curr is t: break

            for next in fanout(curr): # see "for (RouteNode childRNode:rnode.getChildren()) {...}" in exploreAndExpand(...)
                if next in routes: continue
                next_dist = curr_dist + cost(curr) + (est(next, t) - est(curr, t))
                min_heap.push((curr, next), next_dist)

    #parallel for
    for node in routes: # see updateCostFactors()
        update_historical_cost(node)

        if conflicts.isEmpty(): break
```

Fine-grained Parallelization

Amorphous Data-Parallelism

- example: SSSP in LonestarGPU 6.0 (integrated with the Galois library)
- older version uses IrGL compiler for faster BFS and SSSP, but no longer maintained
- note: the same group has also worked on AIG rewriting and SPRoute

GPU-Accelerated

- Shen M, Luo G. "Corolla: GPU-accelerated FPGA routing based on subgraph dynamic expansion," FPGA 2017.
    - subgraph dynamic expansion + GPU-based SSSP (LonestarGPU)
    - SSSP: Bellman-Ford
    - Coarse-graind parallel: Parallel multi-net simultaneously
    - Fine-grained parallel: GPU-based SSSP
- Shen M, Xiao N, Luo G. "Dependency-aware parallel routing for large-scale FPGAs," ICCD 2018.
    - Net dependency-aware region partitioning (load-balance)

SIMD Vectorization

- Khorasani F, Gupta R, Bhuyan L N. "Scalable SIMD-efficient graph processing on GPUs," PACT 2015.
- ~~Jiang P, Agrawal G. "Conflict-free vectorization of associative irregular applications with recent SIMD architectural advances," CGO 2018.~~ (targeted at Xeon Phi)
- Zheng R, Pai S. "Efficient execution of graph algorithms on CPU with SIMD extensions," CGO 2021.
- https://github.com/muneeb706/Bellman-Ford-SSSP
    - Three versions Bellman Ford: serial, pthread, SIMD (intel processor)

Parallel A*

Bidirectional A*:

- Paper: https://homepages.dcc.ufmg.br/~chaimo/public/ENIA11.pdf
- Paper (serial): https://repub.eur.nl/pub/16100/ei2009-10.pdf
- Code: https://github.com/davidleston/Parallel-New-Bidirectional-A-Star

GPU accelerated A*:

- Paper: https://yichaozhou.com/publication/1501massive/paper.pdf
- Code: https://github.com/zhou13/uastar

Collections

- Code: https://github.com/lulufa390/Parallel-A-Star-Algorithm

Survey: https://arxiv.org/pdf/1708.05296.pdf

Parallel Rippel Search: https://graphics.tudelft.nl/~rafa/myPapers/bidarra.MIG2011.pdf