IEEE/ACM
2024 INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN
ICCAD

GPU Acceleration for Versatile Buffer Insertion

Yuan Pu*[1,3], **Yuhao Ji***[1], Siying Yu[1], Zuodong Zhang[2], Zizheng Guo[2], Zhuolun He[1,3], Yibo Lin[2], David Pan[4], Bei Yu[1]

[1] The Chinese University of Hong Kong
[2] Peking University
[3] ChatEDA Tech
[4] University of Texas at Austin

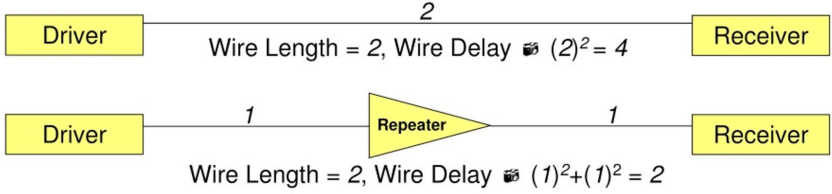
October 27, 2025




Outline

Introduction

Buffer Insertion: Introduction



- Wire RC delay is quadratic function of wire length: l: $\tau_{\text{int}} = (rl) \cdot (cl) = rcl^2 \propto l^2$
- Roles of buffer insertion:
  - Segment wires to decrease delay $\rightarrow$ setup violation.
  - modify downstream load capacitance $\rightarrow$ max transition/capacitance violation.
  - Split net $\rightarrow$ max fanout violation.

Van Ginneken's Algorithm

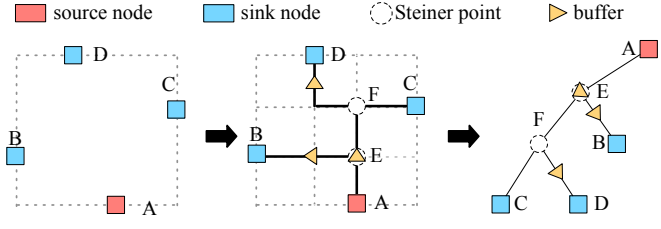
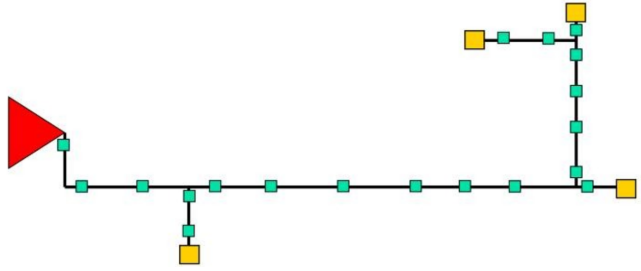


Illustration of the Van Ginneken's Algorithm. (a) a net with 4 pins, (b) the buffered Steiner tree of the 4-pin net and (c) the buffered binary routing tree of the 4-pin net.

- Algorithm: dynamic programming.

Van Ginneken's Algorithm: Limitations

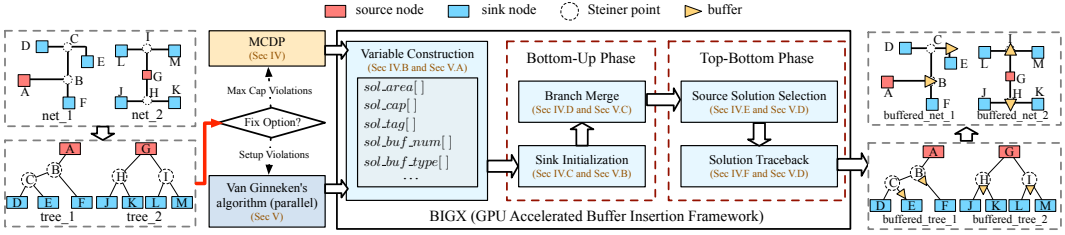


Limitations:

- Pre-allocate candidate buffer locations on the binary routing tree before DP process:
  - Trade-off between efficiency and effectiveness.
  - Hard to allocate buffer candidate locations to efficiently eliminate max capacitance violations $\rightarrow$ different buffer types $\rightarrow$ different driving strength and thus interval on the wire.
- CPU implementation $\rightarrow$ hard to parallelize for big design with many nets, not scalable.

Proposed Algorithm

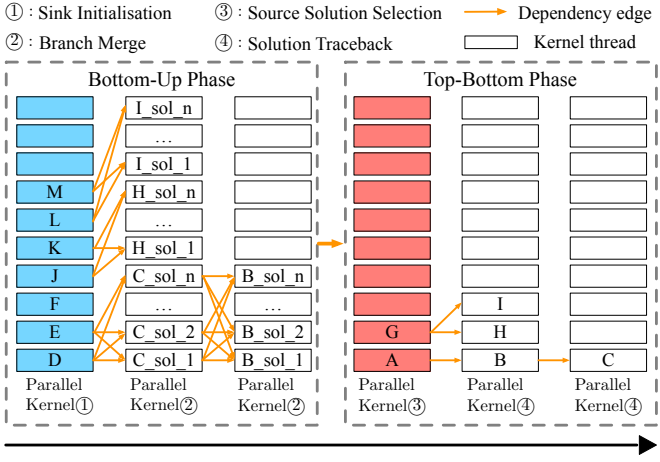
Overall Flow of BIGX



Summarization:

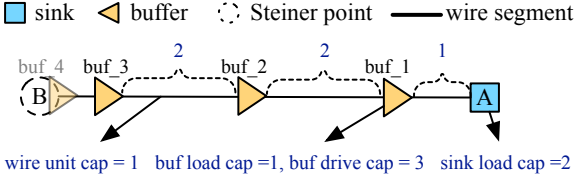
- MCDP algorithm: parallel DP algorithm customized for max capacitance violation.
- Parallel Van Ginneken's algorithm: parallel version.
- BIGX: versatile GPU acceleration framework for accelerating MCDP and Parallel Van Ginneken's algorithm

BIGX Parallelism Philosophy: Topology-Level Parallelism



Parallelly deal with nodes (sinks or Steiner points) at the same topological level.

MCDP Algorithm: Overview



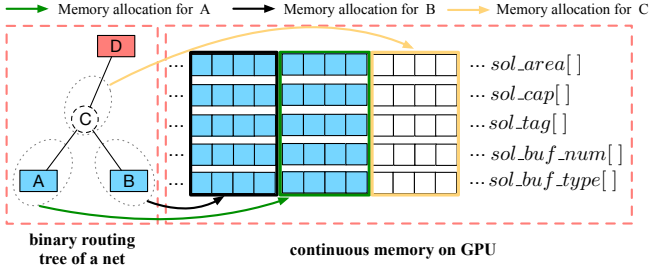
- Buffering for resolve max cap violation: buffers are along long wire segments → ensure each driver operates within its driving capability.
- Limitations of Van Ginneken's algorithm:
  - Allocating fewer candidate locations → leaving violations unresolved when the capacitance of long wire segments exceed the buffer's driving capacity
  - Allocating more locations → significantly increases computational overhead

MCDP Algorithm: Motivation

- Motivation:
  - MCDP dynamically determines optimal buffering locations along wire segments during the DP process.
  - A a buffer is placed $\rightarrow$ ensures its upstream driver has exact driving strength to drive both the wire segment's capacitance and the buffer's input load capacitance $\rightarrow$ **ensure all max cap violations are eliminated**.
  - Assumption: On each wire segment, only one buffer type support $\rightarrow$ for simplicity.
  - For buffer type k, the fixed interval between each buffer candidate location:

$$\text{num\_buf} = \texttt{floor}\left(\frac{\frac{C_{\text{wire}}}{C_{\text{unit}}} - d^k_{\text{init}}}{d^k_{\text{buf}}}\right) + 1 = \texttt{floor}\left(\frac{C_{\text{wire}} + C_{\text{sink}} - C^k_{\text{load}}}{C^k_{\text{drive}} - C^k_{\text{load}}}\right), \tag{1}$$

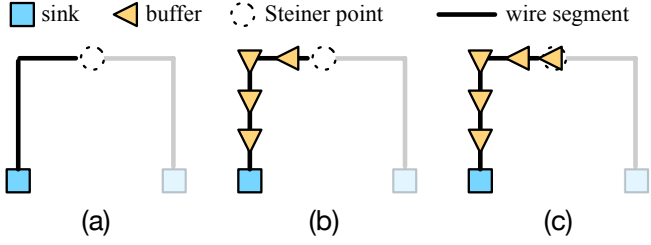
MCDP Algorithm: GPU Variable Design



- *sol_area*: the total buffer area inserted along the node's upstream and downstream branches.
- *sol_cap*: the load capacitance perceived on the upstream Steiner point of the branch.
- *sol_tag*: a boolean variable indicating whether to insert a buffer at the upstream Steiner point.
- *sol_buf_num*: the number of buffers inserted along the upstream branch.
- *sol_buf_type*: the type of buffers used.

MCDP Algorithm Step 1: Sink Initialization

- Phase: Bottom-up (DP table calculation).
- Purpose: initialize the buffer solutions for each sink (one net may have multiple sinks).
- Parallelism: one GPU thread handling one sink node.



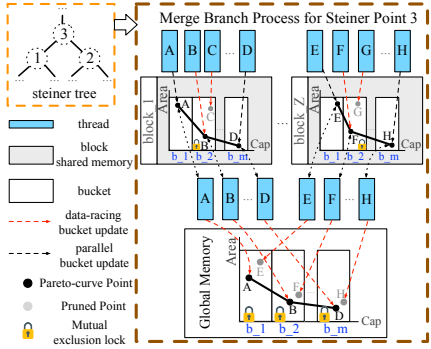
Three buffer insertion candidate solutions generated for one sink (and its upstream branch) during the Sink Initialization Stage.

MCDP Algorithm Step 2: Branch Merge

- Phase: Bottom-Up (DP table calculation)
- Purpose:
  - Merge the solutions of the left and right branches at each Steiner point.
  - Insert buffers into the upstream branch based on the merged results.
  - Maintain the Pareto front for all merged and buffered solutions.
- Runtime bottleneck: $O(N^2)$ for each Steiner point, where N is the number of solutions per branch.
- Solution: **A distributed, GPU-accelerated Branch Merge algorithm.**

MCDP Algorithm Step 2: Branch Merge (con't)

- Each Steiner point → assigned Z blocks, each block with a shared memory for maintaining Pareto Optimality curve.
- Solutions on Pareto Curve: bucket-sorted into specific bucket.
- 2-step algorithm:
  - Step 1: parallelly maintaining PO curve in each block shared memory → **each thread handles one solution point**.
  - Step 2: parallelly merge PO curves in each block shared memory into the global memory → **each thread handles one bucket**.



MCDP Algorithm: Top-Bottom Phase

Step 3: Source Solution Selection

- Purpose: Determine the starting optimal solution for each **source**.
- Parallel algorithm:
  - Each thread $\rightarrow$ one source.
  - Choose the solution with minimal area (all solution can guarantee max cap violation resolved).

Step 4: Solution Traceback

- Purpose: Determine the solution of each internal node following **reverse topological order.**
- Parallelism: Each thread $\rightarrow$ one node; nodes in the same topological order $\rightarrow$ handled parallelly.

GPU implementation of parallel Van Ginneken's algorithm: similar to MCDP's.

Experimental Results

Experimental Setting

- BIGX implementation: C++, CUDA, Flute[1] for Steiner tree generation.
- Benchmark: industrial designs from CircuiNet[2], Chipyard[3] and Xiangshan[4].

[1] **Chris Chu and Yiu-Chung Wong (2007). "FLUTE: Fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design".** In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27.1, pp. 70–83.

[2] **Zhuomin Chai et al. (2023). "Circuitnet: An open-source dataset for machine learning in vlsi cad applications with improved domain-specific evaluation metric and learning strategies".** In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42.12, pp. 5034–5047; **Xun Jiang et al. (2023). "Circuitnet 2.0: An advanced dataset for promoting machine learning innovations in realistic chip design environment".** In: *The Twelfth International Conference on Learning Representations*.

[3] **Alon Amid et al. (2020). "Chipyard: Integrated design, simulation, and implementation framework for custom socs".** In: *IEEE Micro* 40.4, pp. 10–21.

[4] **The XiangShan Project (2025). *High-performance RISC-V processor*.** URL: https://github.com/OpenXiangShan/XiangShan.

Benchmark

Table: Benchmark statistics.

Benchmark	#Macros	#Cells	#Nets	Period (ns)	Tech node (nm)
Vortex	43	113175	122705	2.0	14
Gemmini	737	926064	980634	2.0	14
Vortex-Large	376	1020677	1098249	3.0	14
Nvdla-Large	80	1539105	1695098	5.0	14
Nvdla-Small	108	269289	288877	2.0	14
LargeBoom	636	736553	772344	2.0	14
SmallBoom	314	296339	314287	2.0	14
NVDLA-Small	108	330761	352942	3.0	28
Pulpino	3	18583	19457	3.0	28
XScore	176	4295515	4530869	5.0	28

Experimental Results: MCDP-BIGX

Table: Performance of buffering methods for repairing the maximum capacitance violations.

Benchmark	Post-Placement				OpenROAD: `repair_design`					BIGX-MCDP (ours)				
	WNS (ns)	TNS (ns)	#max cap	power (mW)	WNS (ns)	TNS (ns)	#max cap	power (mW)	runtime (s)	WNS (ns)	TNS (ns)	#max cap	power (mW)	runtime (s)
Vortex	-1.636	-1255.0	1682	**2.003**	**-0.898**	-108.2	80	2.085	22	-0.904	**-100.0**	**1**	2.055	**14**
Gemmini	-4.432	-4235.1	4192	**16.241**	**0.221**	**0**	2959	16.666	212	-4.044	-3463.0	**44**	16.326	**35**
Vortex-Large	-4.686	-5585.5	13411	**16.471**	-1.134	**-150.1**	724	17.053	234	**-1.116**	-160.0	**0**	16.811	**115**
Nvdla-Large	-27.532	-276000.0	45651	**28.663**	-4.832	**-4752.8**	849	32.418	404	**-3.964**	-5589.5	**521**	31.037	**197**
Nvdla-Small (14nm)	-7.202	-34490.1	6961	**6.013**	-1.166	-1263.6	239	6.307	51	**-0.895**	**-621.8**	**0**	6.180	**36**
LargeBoom	-0.579	-850.6	1452	**13.323**	**0.024**	**0**	1287	13.346	123	-0.586	-638.4	**158**	13.348	**26**
SmallBoom	0.002	0	453	**5.904**	0.554	0	900	5.993	59	0.001	0	**19**	5.913	**11**
NVDLA-Small (28nm)	**-0.628**	-89.5	2643	**6.948**	-0.879	-516.8	2189	6.993	51	-0.633	**-57.2**	**319**	6.981	**20**
Pulpino	-0.588	-152.5	108	**1.671**	-1.232	-430.1	28	1.673	3	**-0.569**	**-22.2**	**1**	1.673	**1**
XScore	-7.013	-22172.6	49863	**119.434**	-6.551	-154000.0	68488	119.813	1251	**-1.700**	**-9409.7**	**1970**	119.680	**252**
Normalize	1.000	1.000	1.000	**1.000**	0.656	1.671	0.619	1.031	3.374	**0.620**	**0.392**	**0.034**	1.017	**1.000**

- Baseline: `repair_design` of OpenROAD[5]
- Compared with `repair_design`:
  - Repairs 2.54x more max cap violations.
  - 3.37x Speedup.

[5] **Tutu Ajayi et al. (2019). "Toward an open-source digital flow: First learnings from the openroad project".** In: *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–4.

Experimental Results: Parallel Van Ginnenken's Algorithm

Table: Performance comparison of buffering methods for repairing setup violations based on post-DRV results.

Benchmark	Post-DRV			Van Ginneken's algorithm-CPU				Van Ginneken's algorithm-BIGX (ours)			
	WNS (ns)	TNS (ns)	power (mW)	WNS (ns)	TNS (ns)	power (mW)	runtime (s)	WNS (ns)	TNS (ns)	power (mW)	runtime (s)
Vortex	-0.904	-100.0	**2.055**	**-0.641**	-55.0	2.131	216	-0.658	**-49.6**	2.224	**14**
Gemmini	-4.044	-3463.0	**16.326**	**-0.736**	-512.1	16.595	214	-0.767	**-444.0**	16.666	**45**
Vortex-Large	-1.116	-160.0	**16.811**	-0.498	-21.645	17.826	1855	**-0.423**	**-15.9**	18.855	**134**
Nvdla-Large	-3.964	-5589.5	**31.037**	-3.935	-5142.1	31.598	1802	**-3.885**	**-5130.8**	32.307	**150**
Nvdla-Small (14nm)	-0.895	-621.8	**6.180**	-0.809	-333.447	6.309	346	**-0.793**	**-308.2**	6.434	**27**
LargeBoom	-0.586	-638.4	**13.348**	-0.156	-247.0	13.625	714	**-0.128**	**-190.573**	13.860	**51**
SmallBoom[6]	0.001	0	5.913	0.001	0	5.913	N/A	0.001	0	5.913	N/A
Nvdla-Small (14nm)	-0.633	-57.2	**6.981**	**-0.427**	-26.7	7.025	310	-0.495	**-19.0**	7.205	**23**
Pulpino	-0.569	-22.2	**1.673**	-0.450	-9.0	1.680	31	**-0.449**	**-6.6**	1.710	**2**
XScore	-1.700	-9409.7	**119.680**	-0.965	-8592.8	119.874	563	**-0.878**	**-8118.3**	119.983	**173**
Normalize	1.000	1.000	**1.000**	0.615	0.496	1.021	11.676	**0.608**	**0.436**	1.045	**1.000**

- Baseline: CPU implementation of Van Ginneken's algorithm
- Speedup: 11.68x

THANK YOU!