Graceful Register Clustering by Effective Mean Shift Algorithm for Power and Timing Balancing

Iris Hui-Ru Jiang

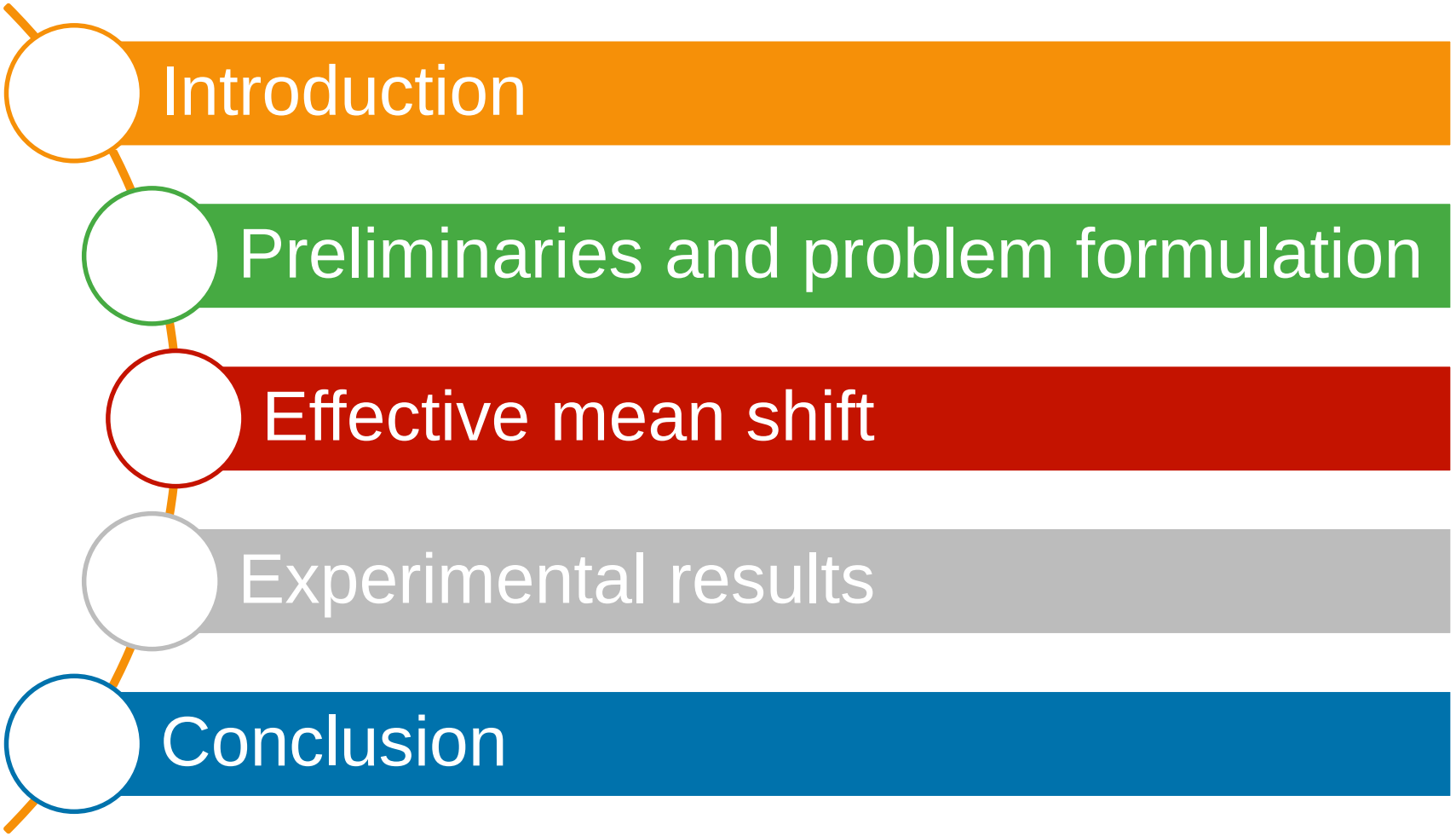
Ya-Chu Chang Tung-Wei Lin Gi-Joon Nam

國立交通大學 National Chiao Tung University

國立臺灣大學 National Taiwan University

IBM

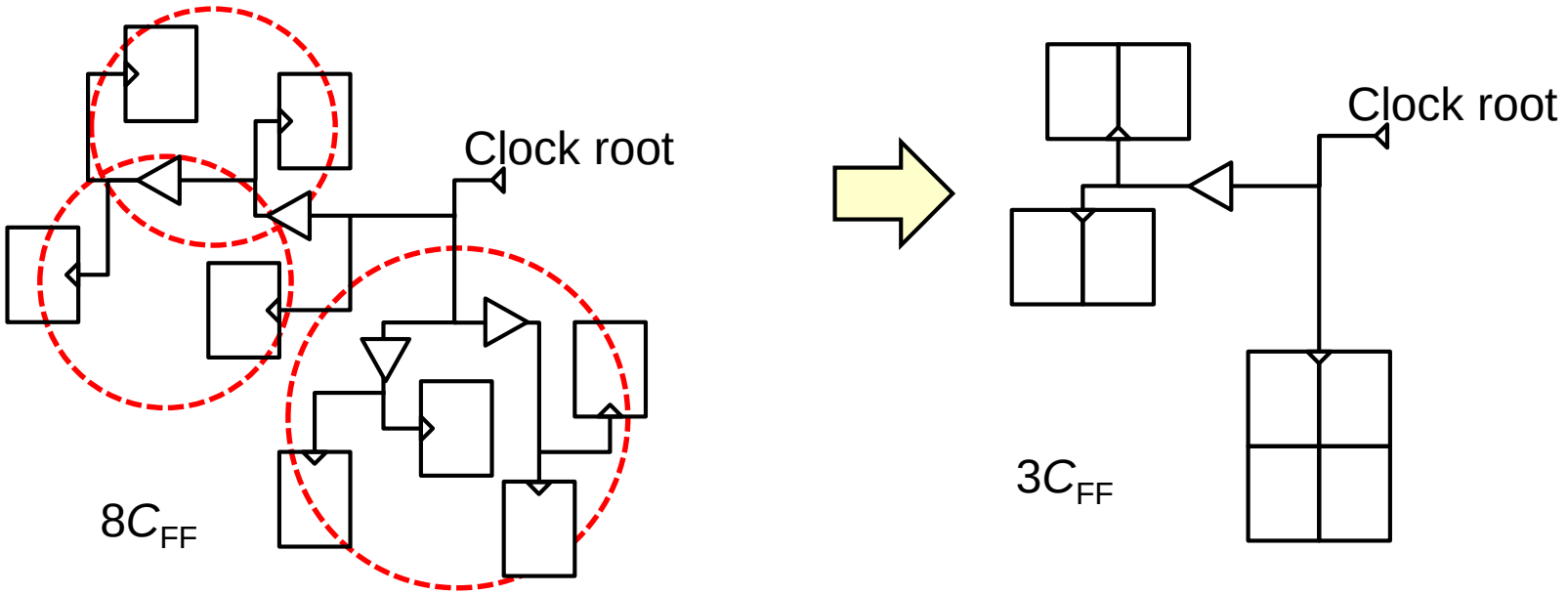
Outline


Introduction
Preliminaries and problem formulation
Effective mean shift
Experimental results
Conclusion

Why Register Clustering?

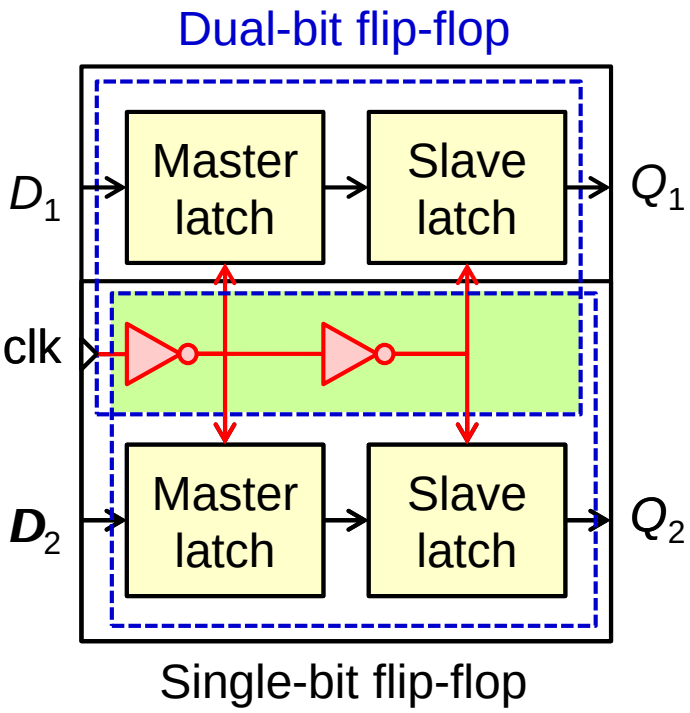
- Dynamic power!! Clock power dominates!!
- Reduce the switching capacitance in a clock network.

Switching capacitance	Clock power saving	Other benefits
Clock sinks (Register capacitance)	**Shared clocking circuitry; #leafs ↓**	**Smaller area**
Clock network (Wirelength, clock buffers)	**#leaf ↓ ⇒ depth ↓**	**Simpler topology and easier skew control**

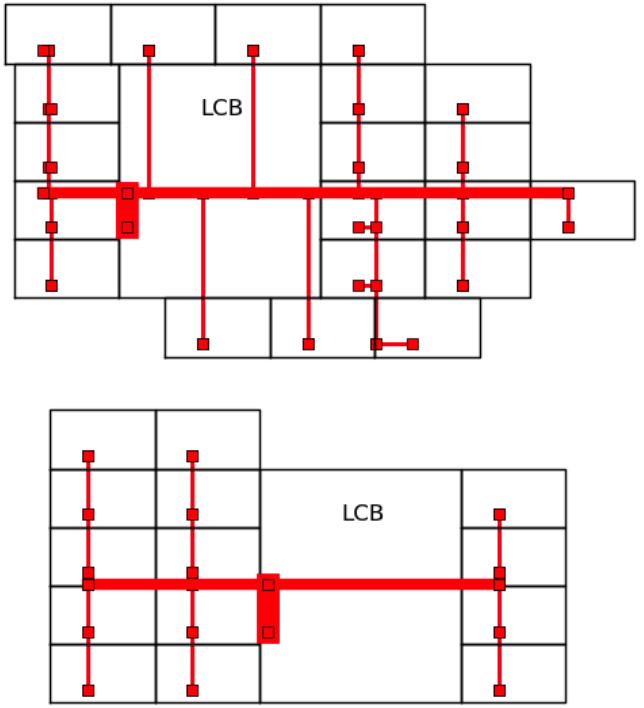


Two Register Cluster Designs

- **Rigid cell**
  - Discrete bits:
    1, 2, 4, 8, 16, 32, 64

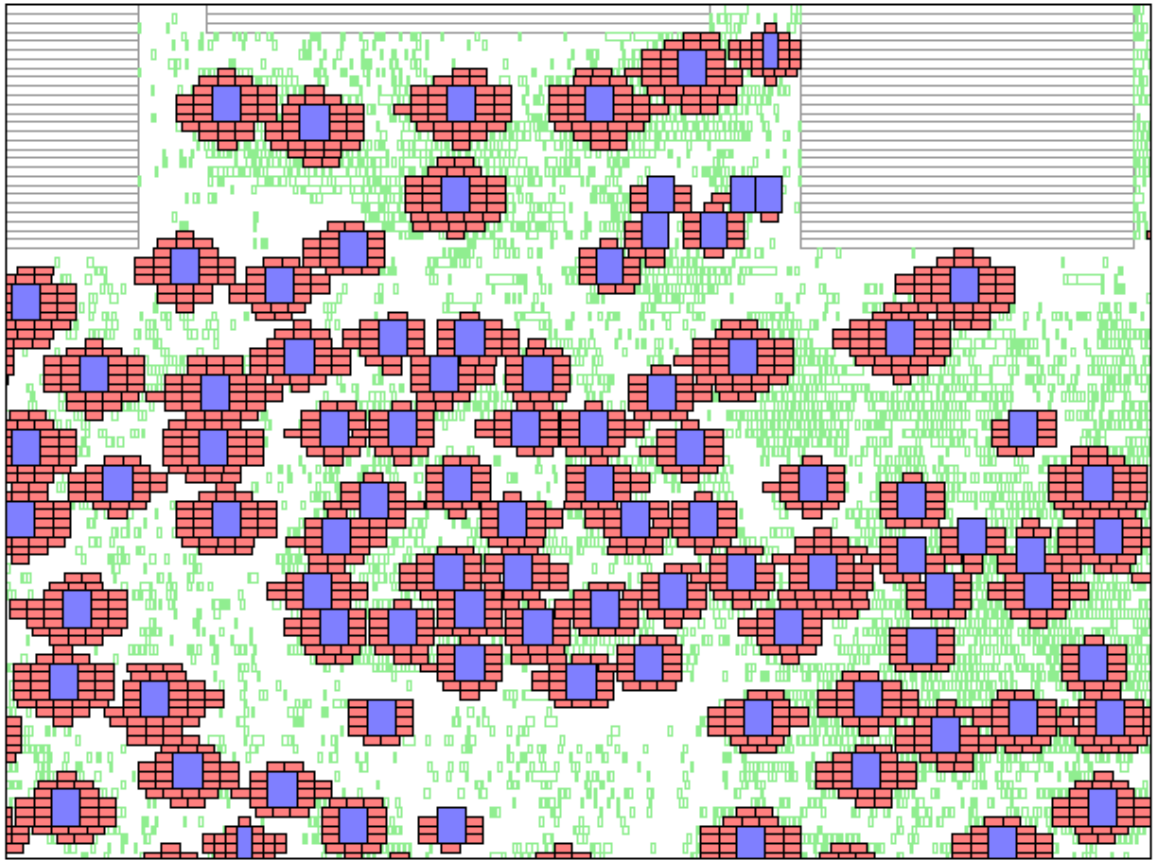


- **Flexible template**
  - Structured latch template:
  - **1**, **2**, 3~**4**, 5~**8**, 9~**16**, 17~**32**, 33~**64**

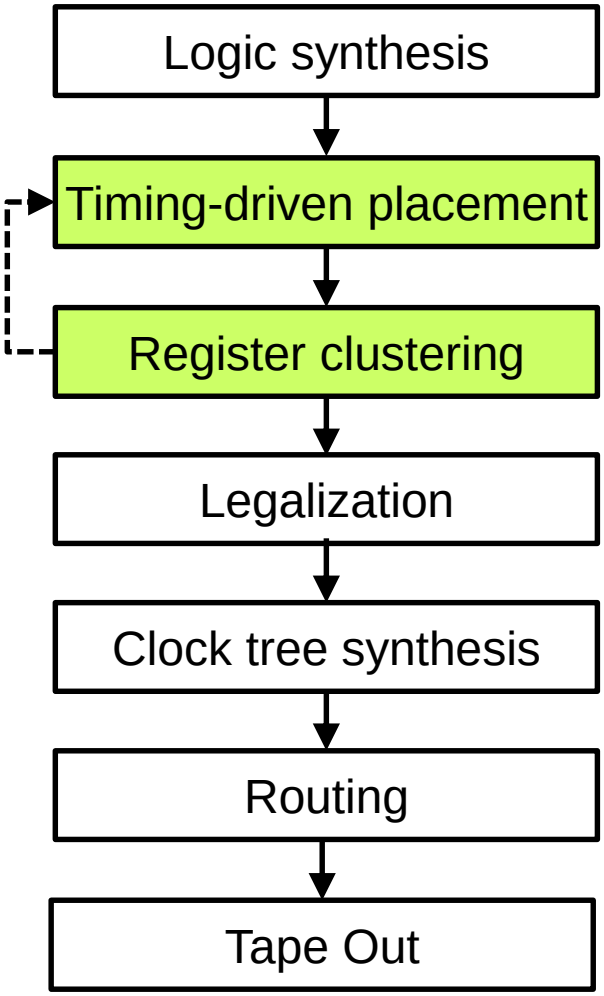


Prior Work (1/3)

- In-placement or post-placement



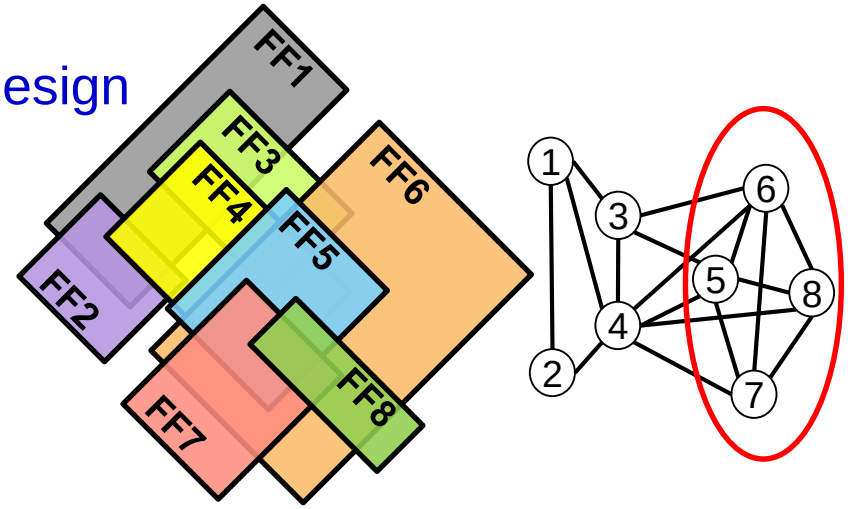
Source: IBM



Prior Work (2/3)

- Clique partitioning
  - Constructs a clustering compatibility graph based on timing feasible regions
  - Extracts maximal cliques to form multi-bit registers without timing degradation

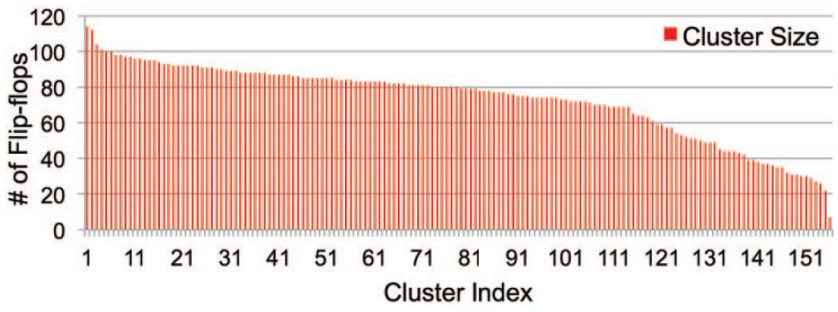
- Up-to-date: [Seitanidis+, DAC-17]
  - Clique enumeration + ILP
  - High complexities!
  - Scalability issue for large-scale design

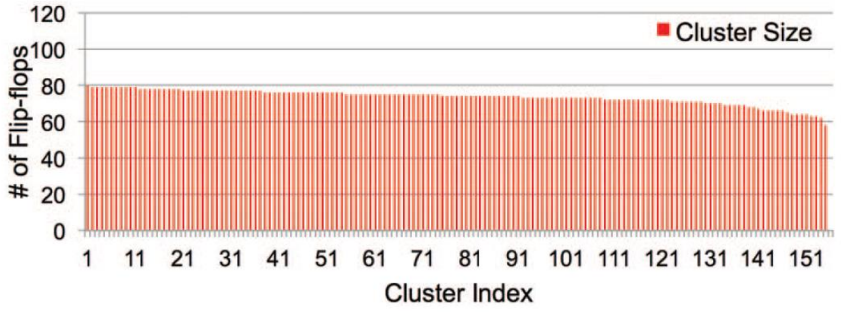


Prior Work (3/3)

- K-means
  - Relaxes timing constraints to maximum displacement constraints
  - Starts with a prespecified # of clusters and initial cluster centers
  - Assigns registers to nearest clusters iteratively until convergence

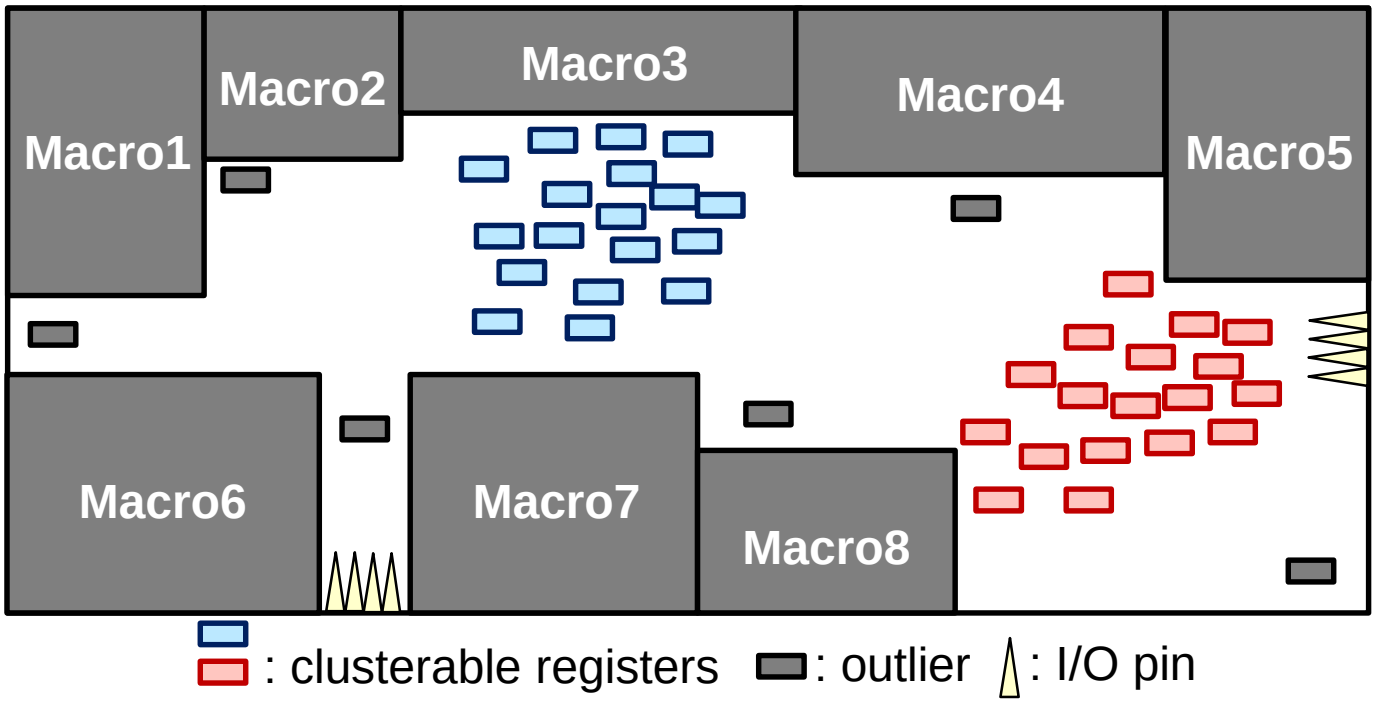
- State-of-the-art: Weighted K-means [Wu+, DAC-16]
  - Is sensitive to initializations and outliers (distant from others)
  - Intends to form large clusters (nearly max. allowable bits)
  - Possibly moves outliers far away
  - Needs extra processes to fix over-displacement & size overflow





Investigations

- Creating large clusters or dragging outliers far away causes large disruption to placement thus incurring significant timing degradation
  - The more timing degradations, the more ECO efforts.
- We can save power even few registers are clustered



What's a Good Register Clustering Algorithm?

- 1) Requires no prespecified number of clusters
- 2) Is insensitive to initializations
- 3) Is robust to outliers
- 4) Is tolerant of various register distributions
- 5) Is efficient and scalable
- 6) Balances power and timing

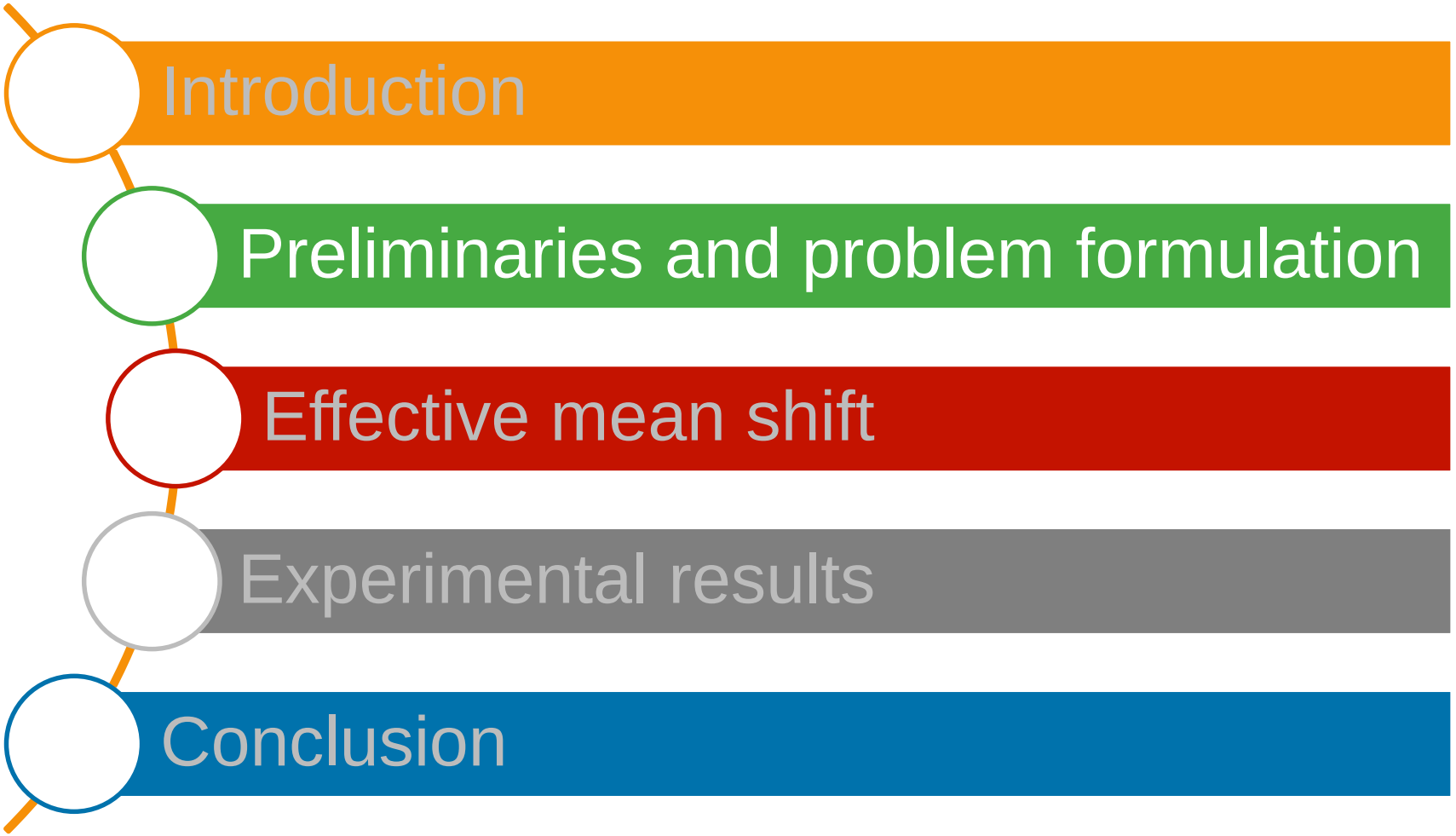
Our Contributions

- Propose **effective mean shift** to perform **graceful register clustering** for reducing clock power while minimizing timing degradation

- Augment classic mean shift with special treatments for register clustering to attain these goals

- Key idea: Conceptually, clusters are expected to reside in dense regions of registers. Our idea is to direct registers towards their nearest densest spots to form clusters naturally.

Outline

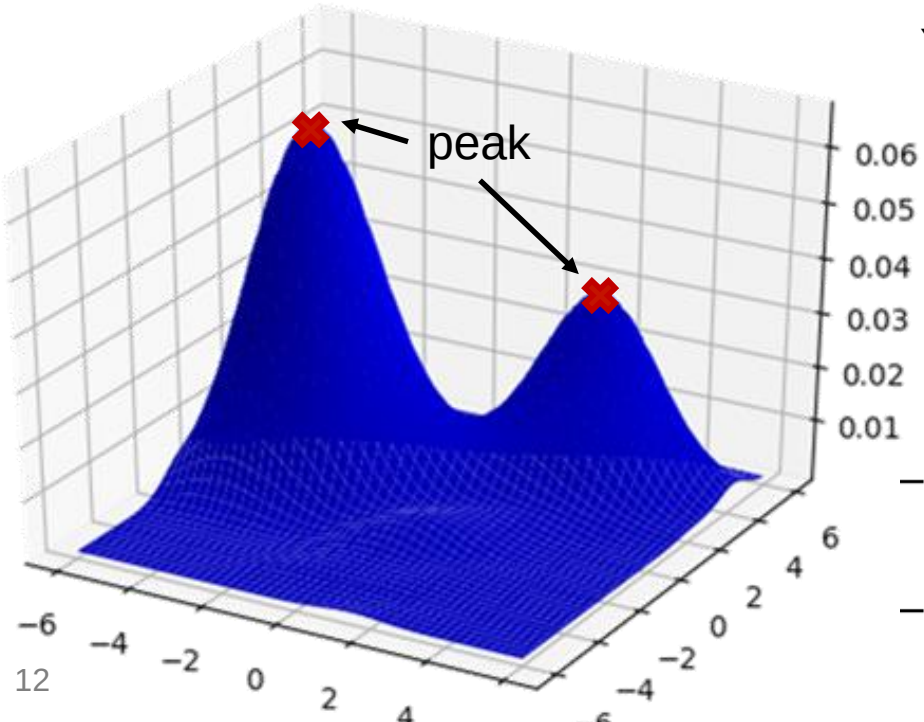

Introduction
Preliminaries and problem formulation
Effective mean shift
Experimental results
Conclusion

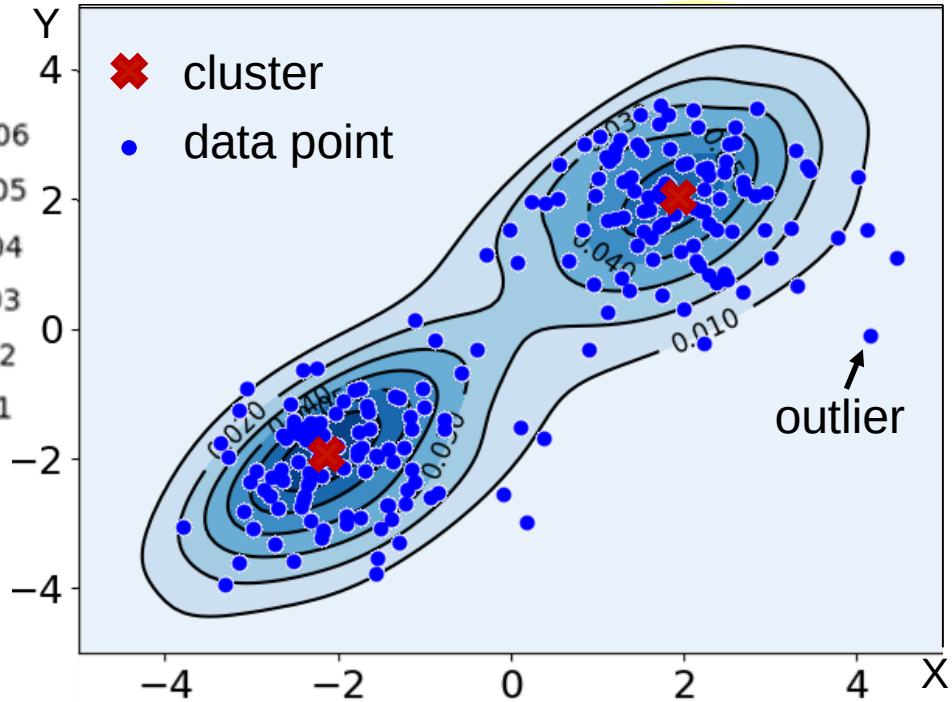
Classic Mean Shift

- Generate a density surface
- Iteratively shift each point uphill
- Time complexity is of $O(Tn^2)$: T iterations, n points

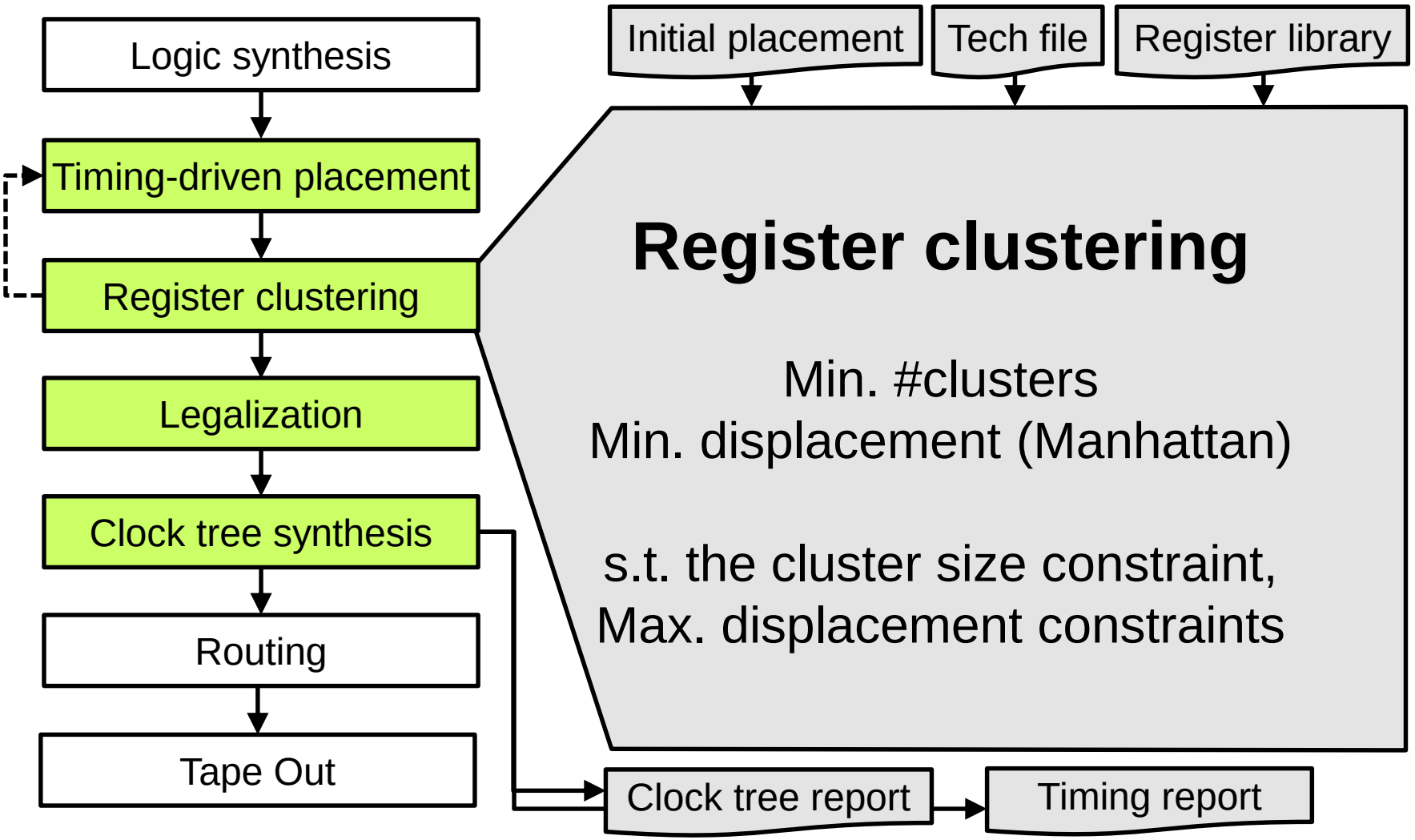
$$f(x) = \frac{1}{nh^d} \sum_{i=1}^{n} k\left(\frac{x - x_i}{h}\right)$$

$$m(x) = \frac{\sum_{i=1}^{n} x_i g\left(\left\|\frac{x - x_i}{h}\right\|^2\right)}{\sum_{i=1}^{n} g\left(\left\|\frac{x - x_i}{h}\right\|^2\right)} - x$$





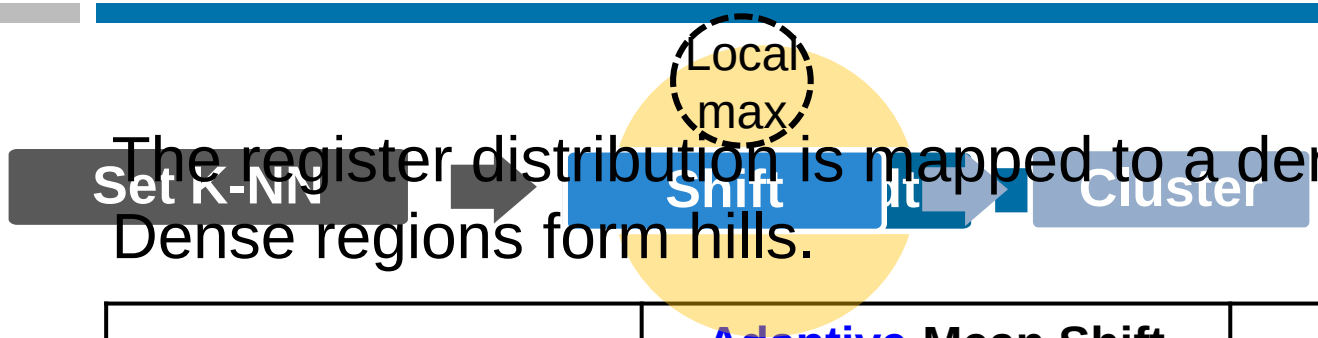
Problem Formulation



Outline


Introduction
Preliminaries and problem formulation
Effective mean shift
Experimental results
Conclusion

Classic vs. Adaptive vs. Effective

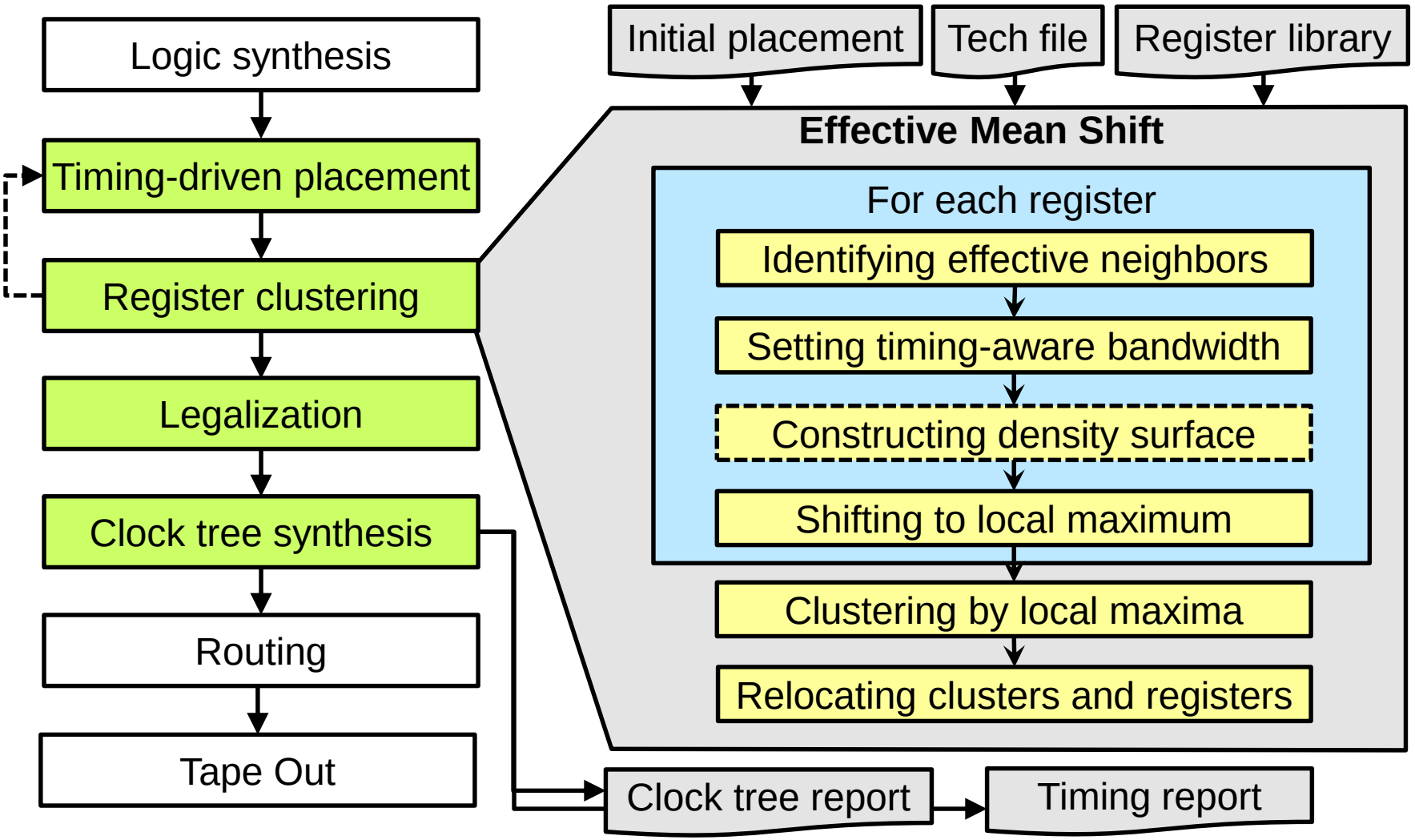


The register distribution is mapped to a density surface.
Dense regions form hills.

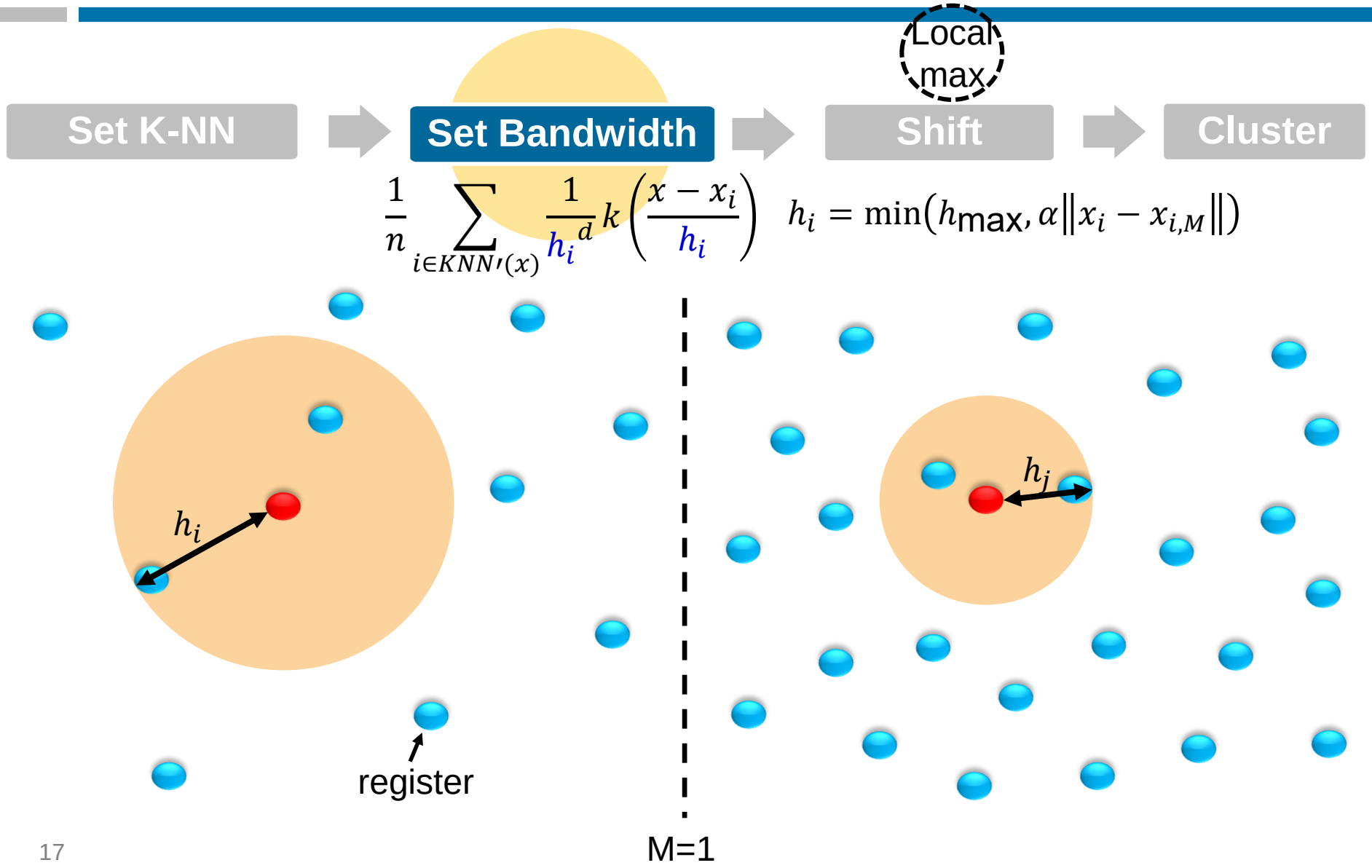
Classic Mean Shift	**Adaptive** Mean Shift (Variable Bandwidth)	**Effective** Mean Shift
Density estimator $\frac{1}{nh^d}\sum_{i=1}^{n} k\left(\frac{x-x_i}{h_i}\right)$	$\frac{1}{n}\sum_{i=1}^{n}\frac{1}{h_i{}^d} k\left(\frac{x-x_i}{h_i}\right)$	$\frac{1}{n}\sum_{i\in KNN'(x)}\frac{1}{h_i{}^d} k\left(\frac{x-x_i}{h_i}\right)$
Shift point $\frac{\sum_{i=1}^{n} x_i g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)}{\sum_{i=1}^{n} g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)}$	$\frac{\sum_{i=1}^{n}\frac{x_i}{h_i{}^{d+2}} g\left(\left\|\frac{x-x_i}{h_i}\right\|^2\right)}{\sum_{i=1}^{n}\frac{1}{h_i{}^{d+2}} g\left(\left\|\frac{x-x_i}{h_i}\right\|^2\right)}$	$\frac{\sum_{i\in KNN'(x)}\frac{x_i}{h_i{}^{d+2}} g\left(\left\|\frac{x-x_i}{h_i}\right\|^2\right)}{\sum_{i\in KNN'(x)}\frac{1}{h_i{}^{d+2}} g\left(\left\|\frac{x-x_i}{h_i}\right\|^2\right)}$

1. $k(x) = \kappa(\|x\|^2)$, Gaussian kernel 2. $g(x) = -\kappa'(x)$ 3. $d = 2$

Overview


Logic synthesis
Timing-driven placement
Register clustering
Legalization
Clock tree synthesis
Routing
Tape Out
Initial placement
Tech file
Register library
Effective Mean Shift
For each register
Identifying effective neighbors
Setting timing-aware bandwidth
Constructing density surface
Shifting to local maximum
Clustering by local maxima
Relocating clusters and registers
Clock tree report
Timing report

Variable Bandwidth Selection



Identifying Effective Neighbors

- Points that correspond to the tails of the underlying density function receive small weights, and thus they are almost automatically discarded.
- Consider only effective neighbors
- Iteratively updating effective neighbors may still be computation intensive
- Computing KNN only once
  - Neighbors barely change, effective neighbors can be identified only once (at the beginning)
  - Analysis of distinct neighbors (K=140)

Circuit	# of Iterations	# of Total Distinct Neighbors	# of Distinct Neighbors per Iteration
Superblue16	213	158.25	0.74
Superblue18	315	158.09	0.50
Superblue10	533	156.13	0.29

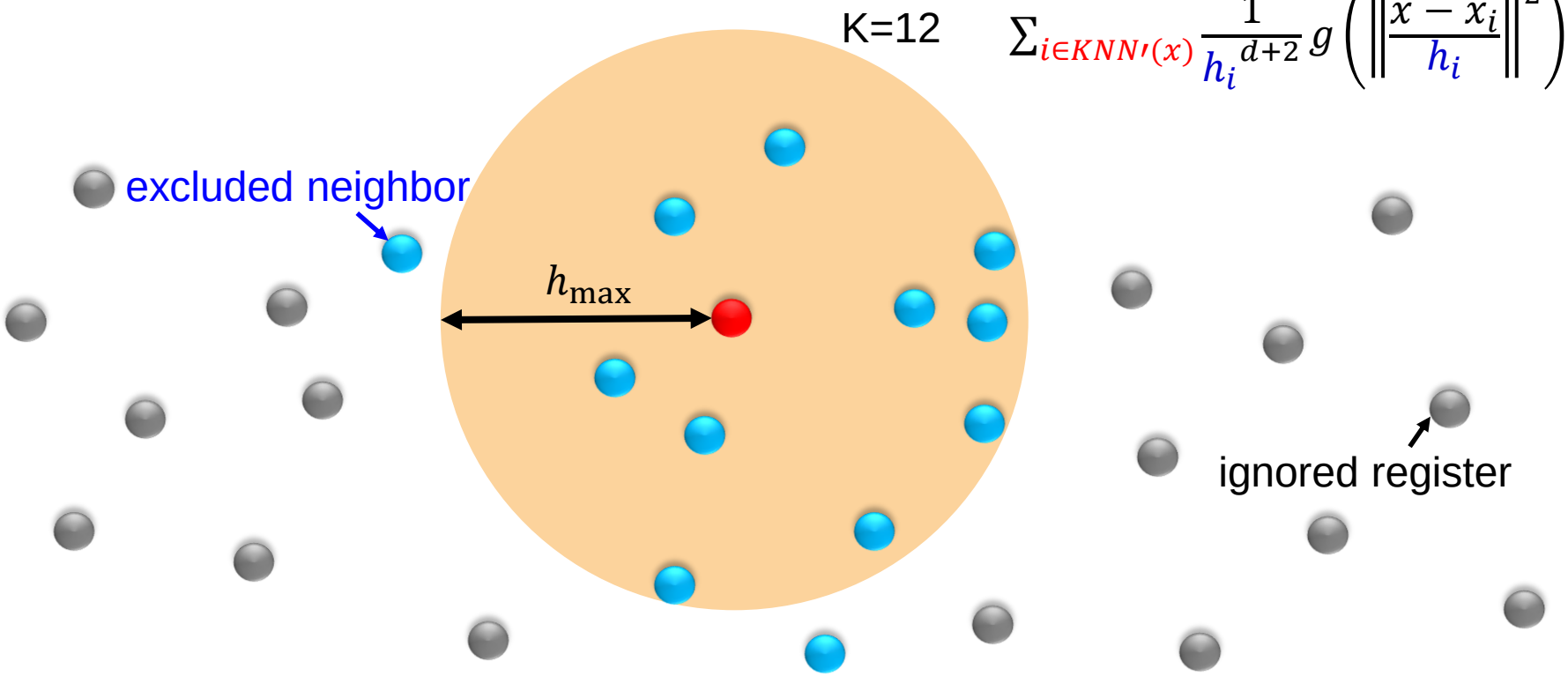
Setting K-Nearest Neighbors

Local max

Set K-NN → **Set Bandwidth** → **Shift** → **Cluster**

– Constraint: maximum displacement

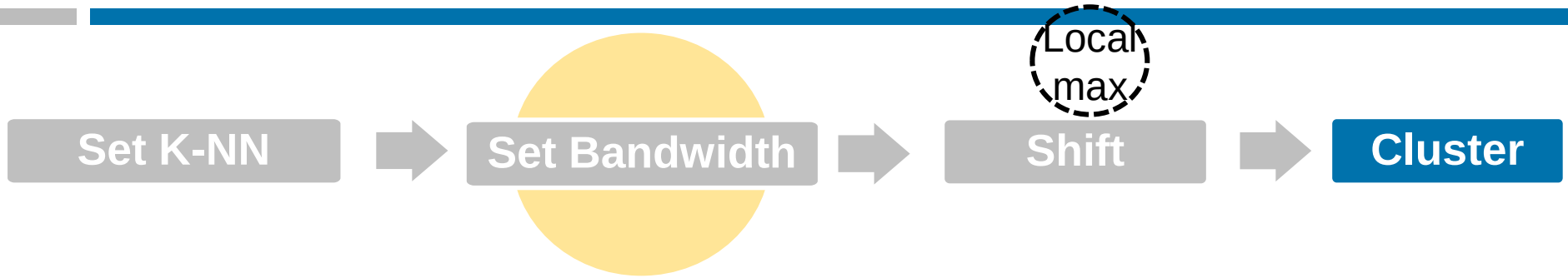
$$\frac{\sum_{i \in KNN'(x)} \frac{x_i}{h_i^{d+2}} g\left(\left\|\frac{x - x_i}{h_i}\right\|^2\right)}{\sum_{i \in KNN'(x)} \frac{1}{h_i^{d+2}} g\left(\left\|\frac{x - x_i}{h_i}\right\|^2\right)}$$



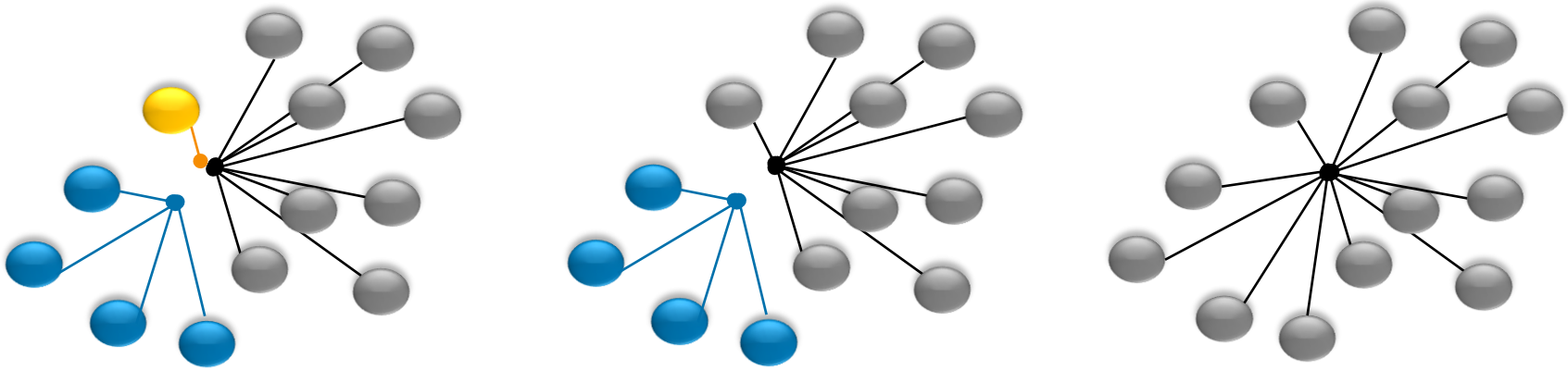
Shifting to Local Density Maxima

- Each register undergoes the following steps to seek the local density maximum
  1. Set the initial coordinates, $y_j^0 = x_j, j = 1..n$
  2. Identify effective neighbors, $KNN'(y_j^0)$; set bandwidth h_j
     - Then, the density surface is formed
  3. Compute the mean shift vector $m(y_j^t)$
  4. Shift each register, $y_j^{t+1} = y_j^t + m(y_j^t) = \dfrac{\sum_{i \in KNN'(y_j^0)} \frac{x_i}{h_i^{d+2}} g\left(\left\|\frac{y_j^t - x_i}{h_i}\right\|^2\right)}{\sum_{i \in KNN'(y_j^0)} \frac{1}{h_i^{d+2}} g\left(\left\|\frac{y_j^t - x_i}{h_i}\right\|^2\right)}$
  5. Iterate steps 3 and 4 until convergence, $\left|y_j^{t+1} - y_j^t\right| < \delta$

Clustering by Local Density Maxima



- Compensate the approximation error of KNN



(a) Small threshold (b) Medium threshold (c) Large threshold

Relocation for Timing and Displacement

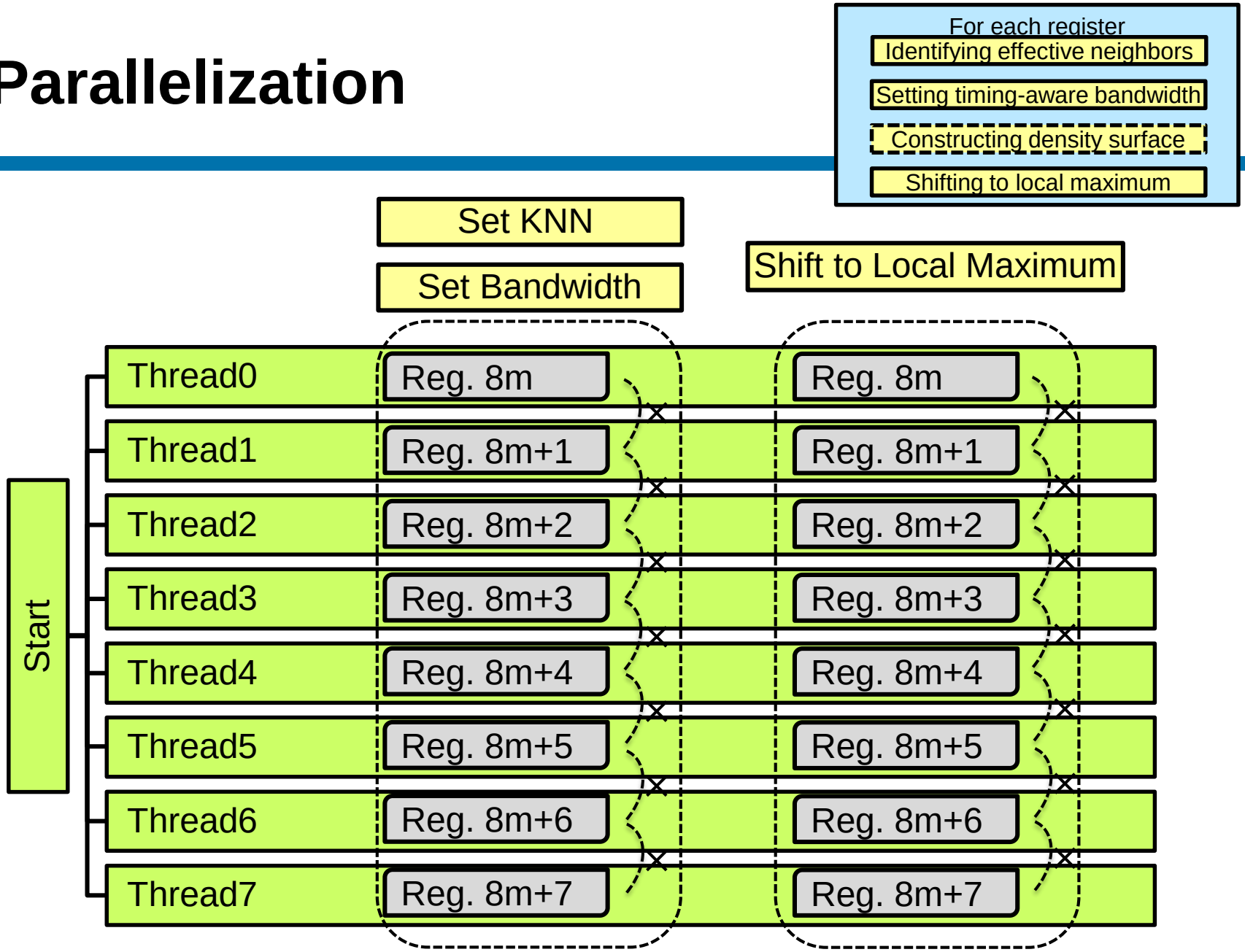
- The previous steps in effective mean shift can be viewed as seeking the locations of clusters
- Reassign registers and relocate clusters for improving timing and displacement
  - Manhattan distance
- Relocate each cluster to the median coordinate of its register members for minimizing displacement and reducing timing degradation

Complexity Analysis

- Classic mean shift: $O(Tn^2)$, T iterations, n registers

- Effective mean shift: $O(TKn + Cn)$, K effective neighbors, C clusters.
  - Shifting to local density maxima: $O(TKn)$ time, $K \ll n$
  - Register reassignment and cluster relocation: $O(Cn)$ time, $C \ll n$

Parallelization



Outline


Introduction
Preliminaries and problem formulation
Effective mean shift
Experimental results
Conclusion

Experimental Setting

- C++; Intel Xeon 2.6 GHz CPU and 256 GB memory
- 2015 CAD contest in incremental timing-driven placement

Circuit	# of Cells	# of Registers
superblue16	981,559	142,543
superblue18	768,068	101,758
superblue4	796,645	167,731
superblue5	1,086,888	110,941
superblue3	1,213,253	163,107
superblue1	1,209,716	137,560
superblue7	1,931,639	262,176
superblue10	1,876,130	231,747

- Pseudo power of multi-bit register library

# of Bits	Normalized Pseudo Power per Bit
1	1.000
2~3	0.860
4~7	0.790
8~15	0.755
16~31	0.738
32~63	0.729
64~80	0.724

- Cadence Innovus

Effective Mean Shift vs. Weighted K-Means

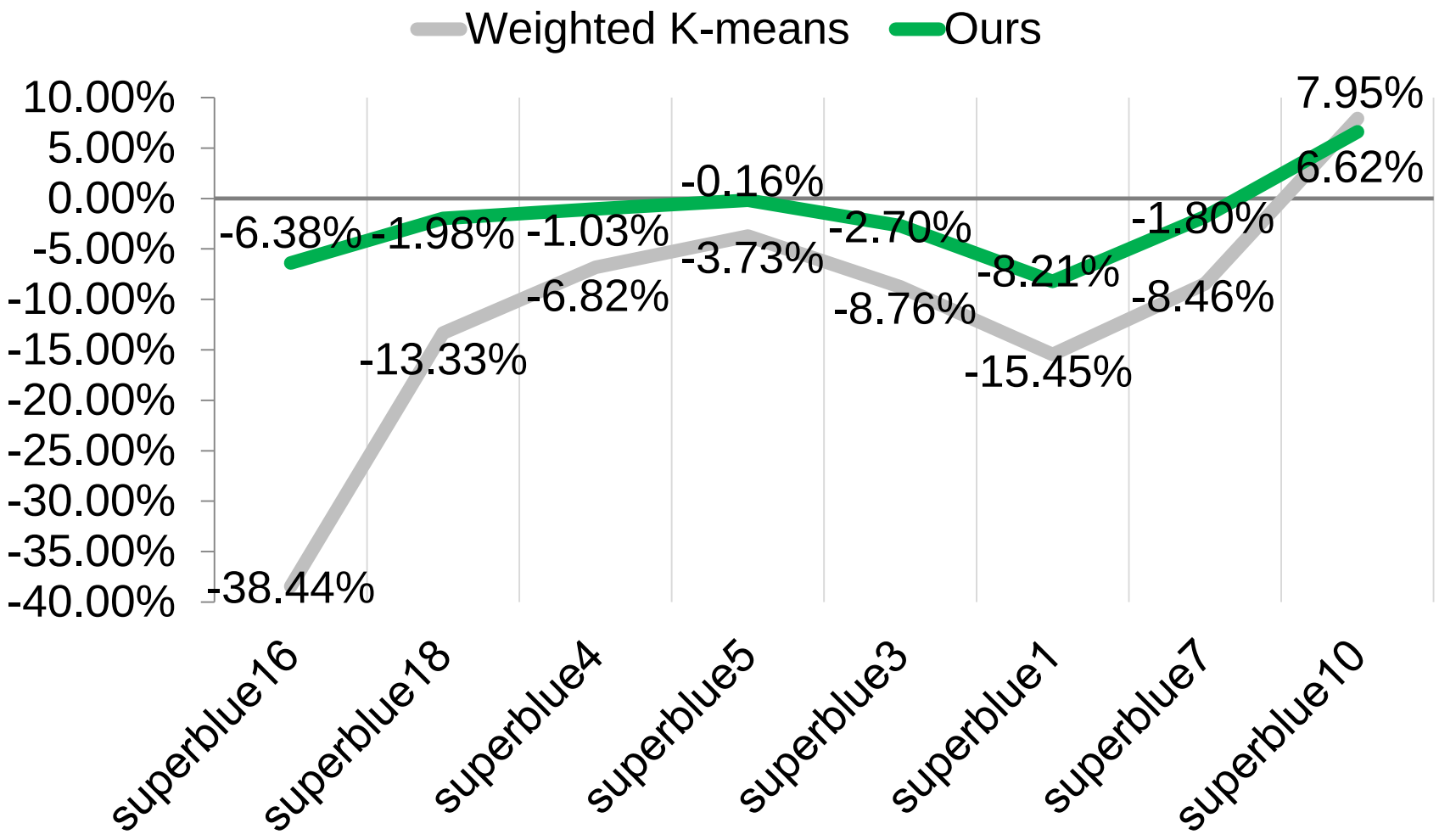
Circuit	Method	Cluster Size		Displacement	Runtime (s)	
		Min	Max	Average	Parallel	Sequential
superblue16	WK	34	80	56000.54	2370	
	Ours	1	55	22353.75	35	186
superblue18	WK	35	80	60843.50	6080	
	Ours	1	70	25792.54	25	138
superblue4	WK	34	80	48129.71	8470	
	Ours	1	56	19446.86	51	311
superblue5	WK	32	80	69453.46	3590	
	Ours	1	78	29747.90	28	131
superblue3	WK	28	80	54968.00	9098	
	Ours	1	79	25696.45	45	244
superblue1	WK	42	80	64158.15	5295	
	Ours	1	62	24456.03	40	200
superblue7	WK	39	80	54761.63	37692	
	Ours	1	79	26048.28	91	513
superblue10	WK	26	80	57643.75	27474	
	Ours	1	79	27914.53	75	412
Ratio	**WK/Ours**			**2.33**	**215.03**	**39.42**

The maximum allowable cluster size is 80 (same setting as weighted K-means (WK)
The maximum allowable displacement is 400 nm
For effective mean shift, K = 140 for KNN
Convergence threshold δ = 0.0001 units
Cluster merging threshold ε = 5000 units (2000 unit length = 1 nm)

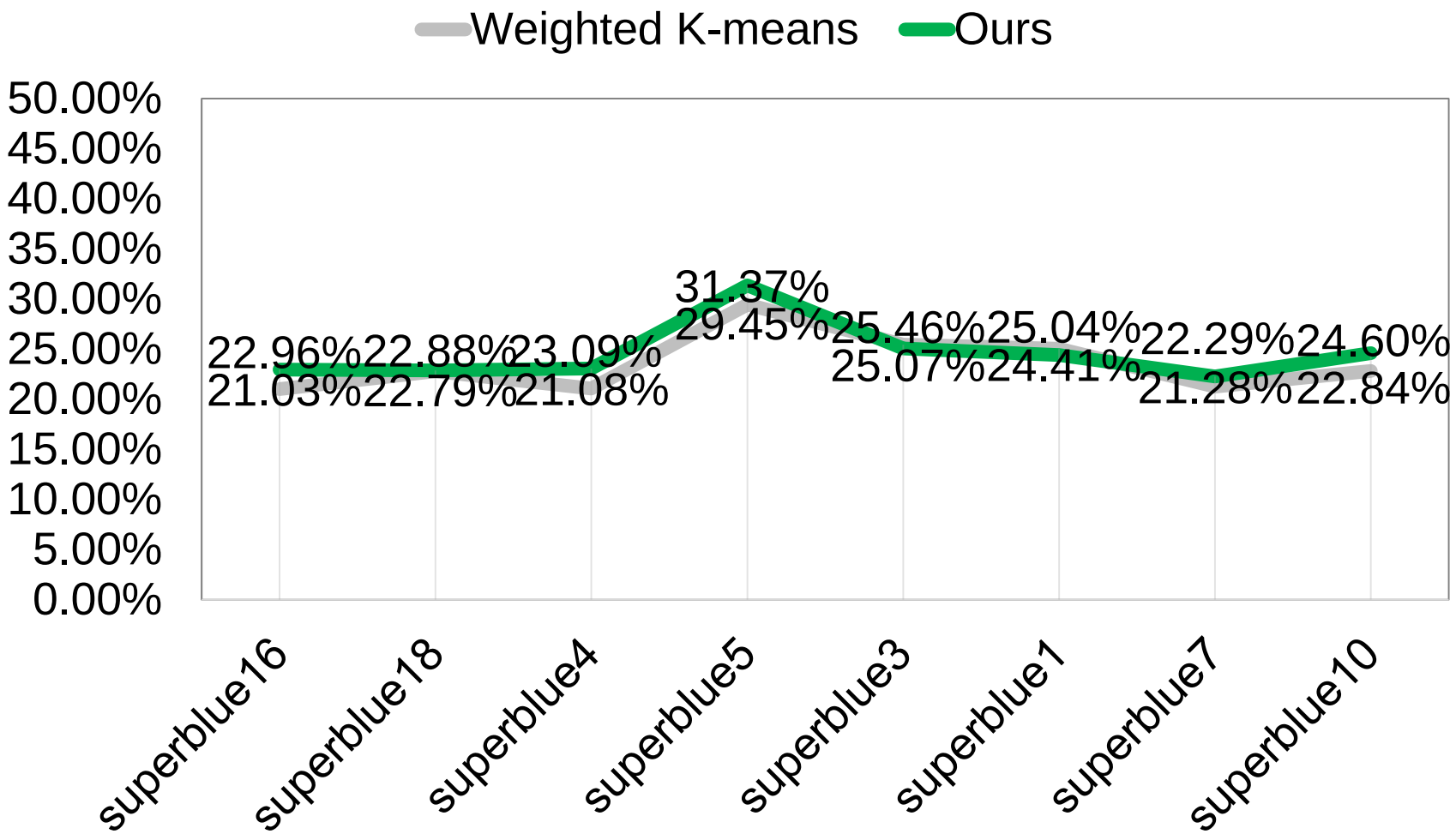
Timing and Power Comparison

Circuit	Method	Timing				Power			
		WNS	TNS (ns)	TNS Degradation Ratio	Clock Routed WL (um)	Ratio	#Buffers	Ratio	Clock Sink Power Ratio
superblue16	NC	-6.2	-1532.0	0.00%	934,654	100.00%	3,414	100.00%	100.00%
	WK	-6.6	-2120.9	-38.44%	196,543	21.03%	1,872	54.83%	72.47%
	Ours	-6.2	-1629.8	-6.38%	214,560	22.96%	1,873	54.86%	74.86%
superblue18	NC	-9.1	-5148.3	0.00%	629,463	100.00%	2,449	100.00%	100.00%
	WK	-9.4	-5834.8	-13.33%	143,471	22.79%	1,314	53.65%	72.47%
	Ours	-9.1	-5250.0	-1.98%	144,009	22.88%	1,228	50.14%	74.32%
superblue4	NC	-9.7	-15669.9	0.00%	1,017,709	100.00%	4,303	100.00%	100.00%
	WK	-10.1	-16738.6	-6.82%	214,560	21.08%	2,124	49.36%	72.47%
	Ours	-9.9	-15830.8	-1.03%	234,966	23.09%	2,072	48.15%	74.91%
superblue5	NC	-30.2	-19866.8	0.00%	928,619	100.00%	3,626	100.00%	100.00%
	WK	-32.3	-20607.3	-3.73%	273,496	29.45%	2,251	62.08%	72.51%
	Ours	-30.3	-19898.6	-0.16%	291,267	31.37%	2,355	64.95%	74.16%
superblue3	NC	-18.9	-7892.9	0.00%	1,047,502	100.00%	4,251	100.00%	100.00%
	WK	-19.7	-8584.5	-8.76%	266,706	25.46%	2,054	48.32%	72.48%
	Ours	-18.9	-8106.1	-2.70%	262,588	25.07%	2,133	50.18%	74.14%
superblue1	NC	-10.2	-6778.5	0.00%	1,047,502	100.00%	3,759	100.00%	100.00%
	WK	-10.5	-7825.5	-15.45%	262,261	25.04%	2,052	54.59%	72.47%
	Ours	-10.2	-7334.7	-8.21%	255,708	24.41%	2,104	55.97%	74.87%
superblue7	NC	-19.4	-12531.2	0.00%	1,702,650	100.00%	6,482	100.00%	100.00%
	WK	-20.9	-13591.3	-8.46%	362,256	21.28%	3,427	52.87%	72.48%
	Ours	-19.2	-12757.0	-1.80%	379,577	22.29%	3,341	51.54%	74.31%
superblue10	NC	-48.7	-151000.0	0.00%	1,660,396	100.00%	6,189	100.00%	100.00%
	WK	-42.7	-139000.0	7.95%	379,246	22.84%	3,210	51.87%	72.48%
	Ours	-42.3	-141000.0	6.62%	408,500	24.60%	3,495	56.47%	74.25%
Average	**NC**			**0.00%**		**100.00%**		**100.00%**	**100.00%**
	WK			**-10.88%**		**23.62%**		**53.45%**	**72.48%**
	Ours			**-1.95%**		**24.58%**		**54.03%**	**74.48%**

TNS Comparison

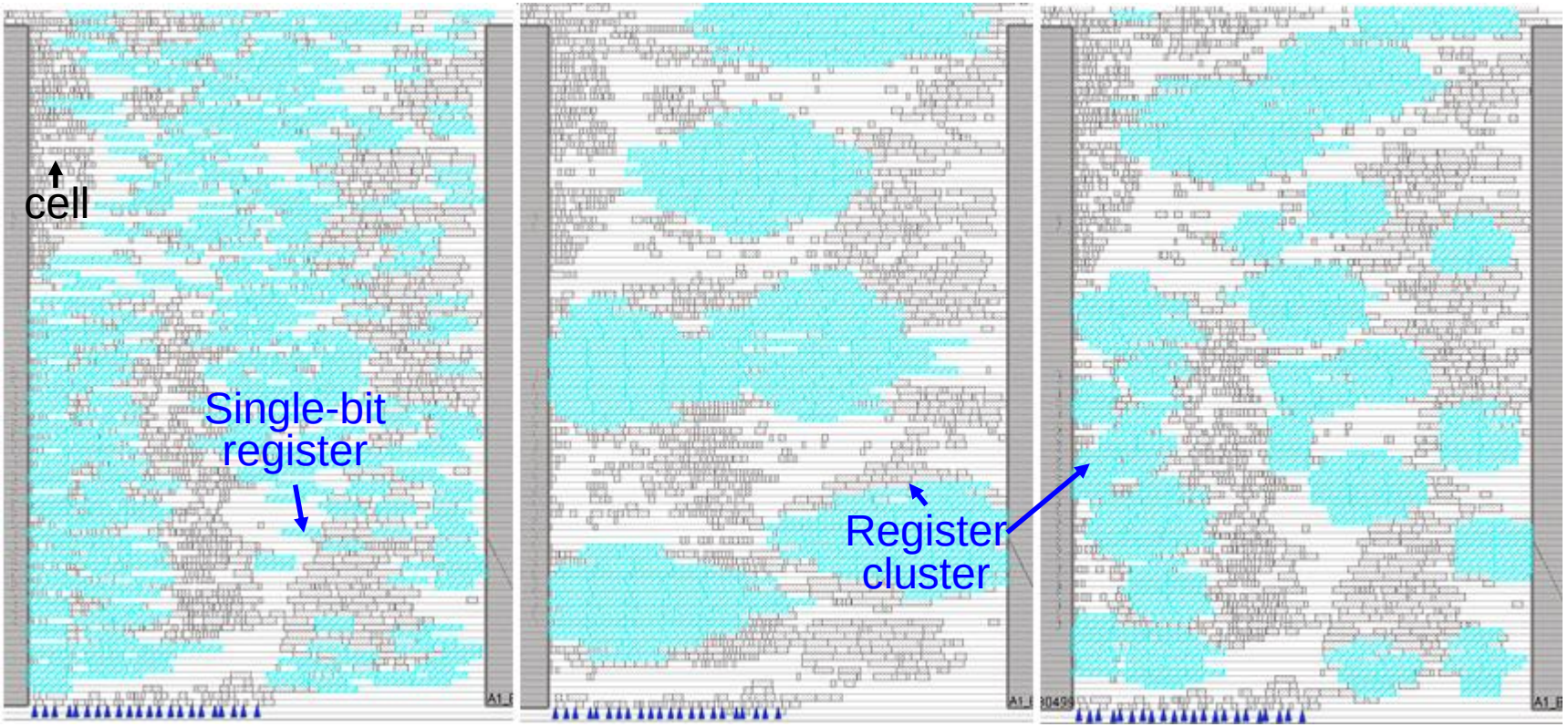


Clock Routed WL Comparison



Zooming In

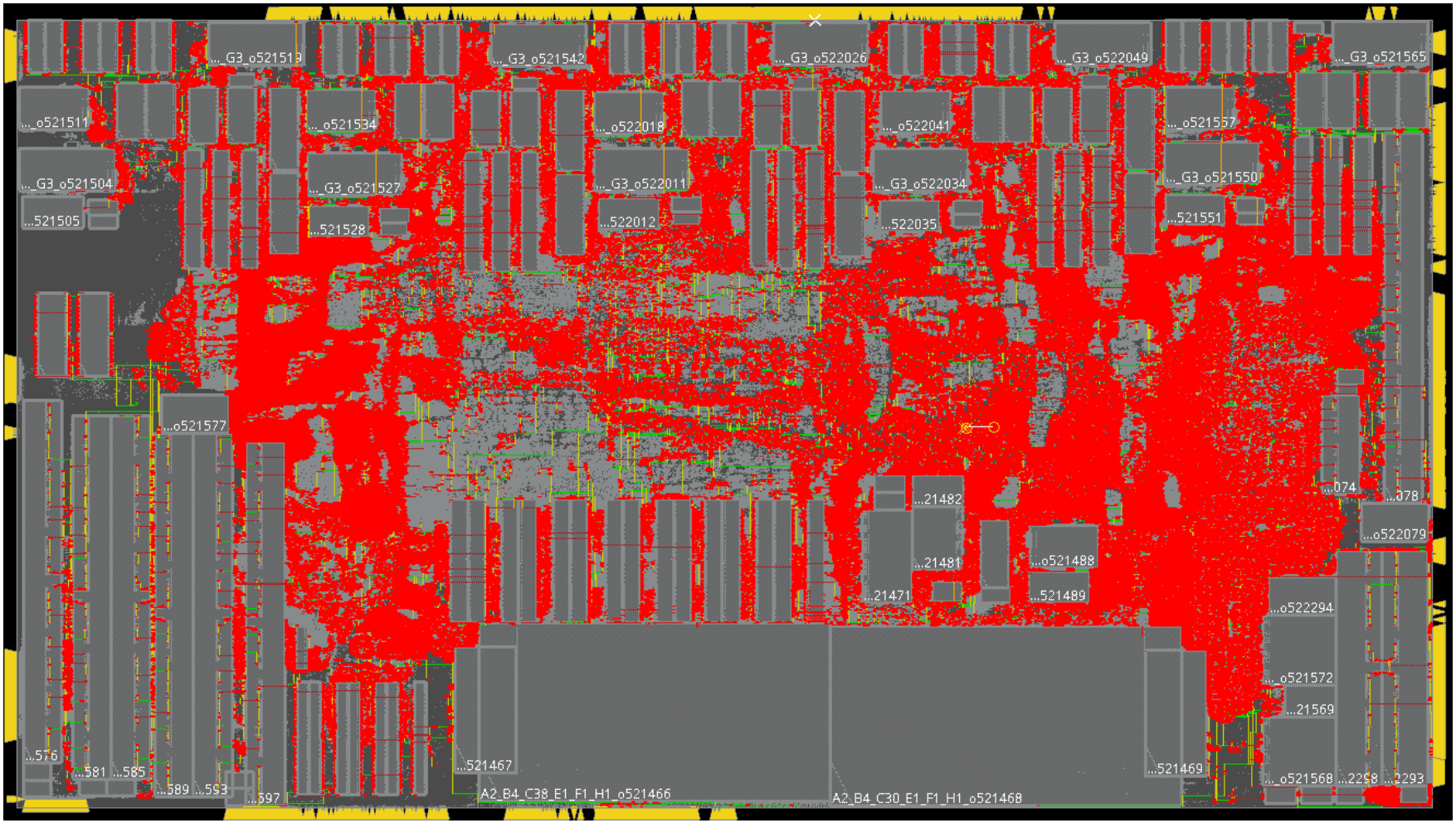
superblue16



(a) Non-clustered (b) Weighted K-means (c) Effective mean shift

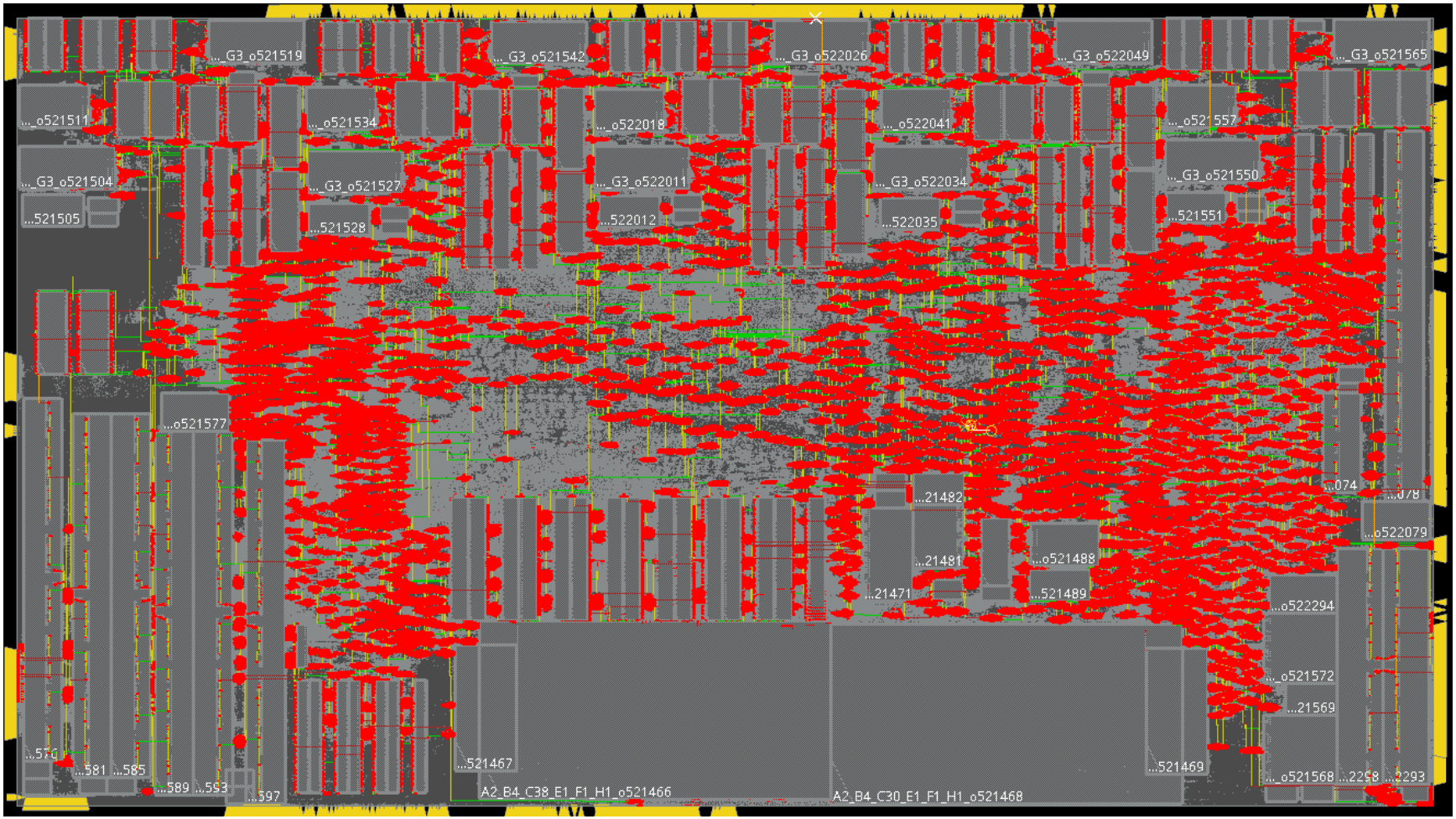
Non-Clustered

superblue4



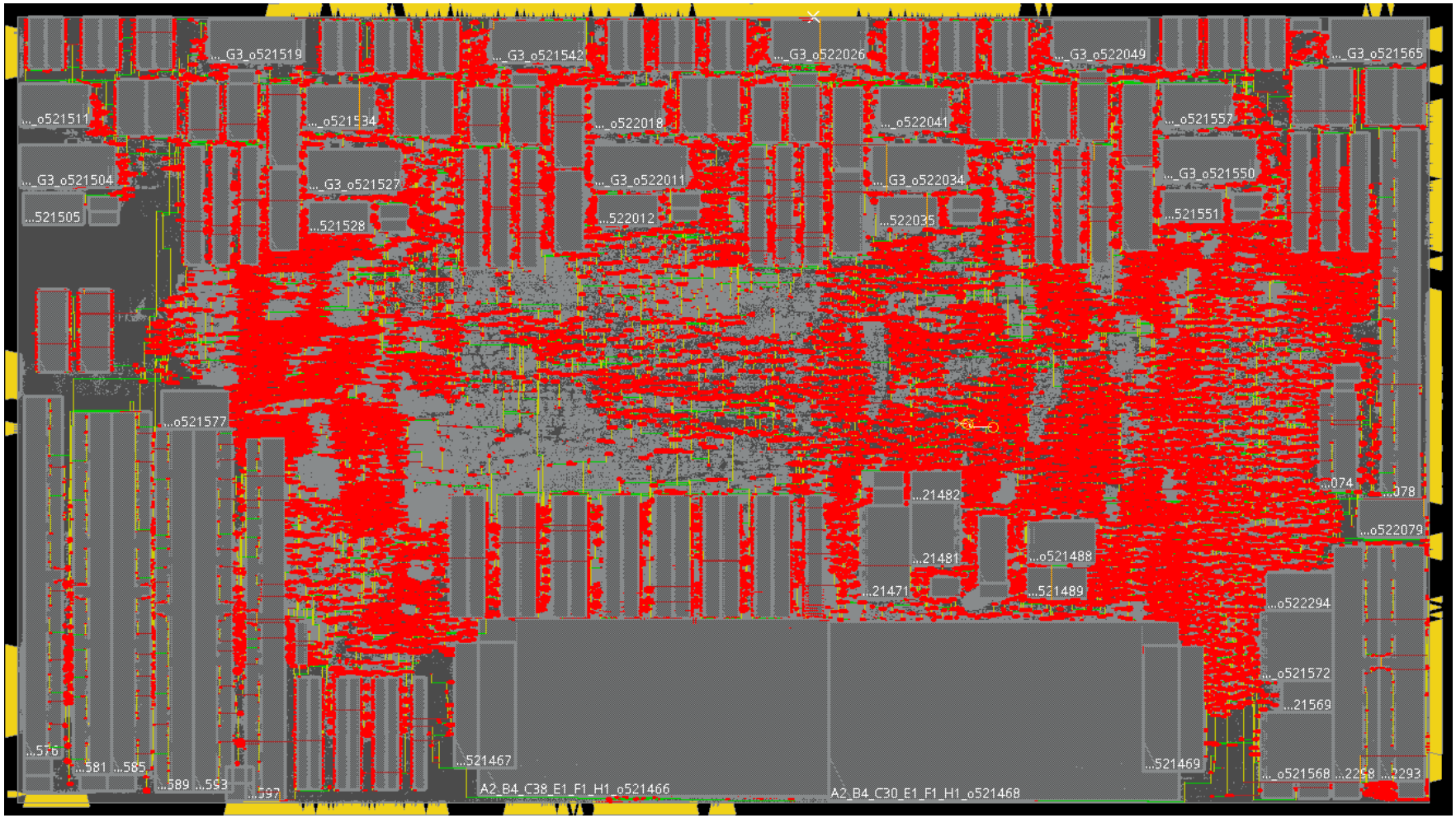
Weighted K-means

superblue4



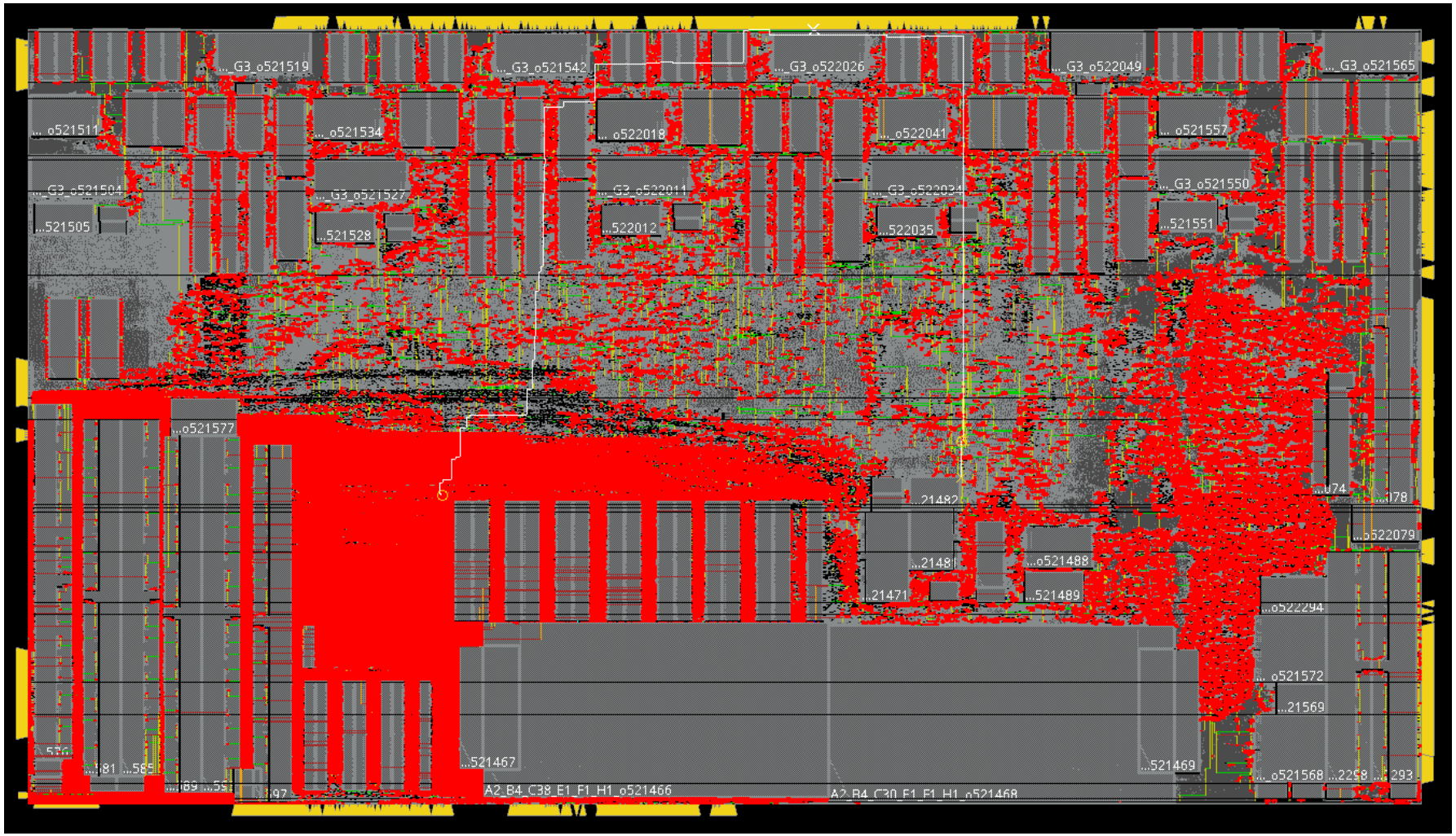
Ours: Effective Mean Shift

superblue4



Wirelength Optimum Sites

superblue4

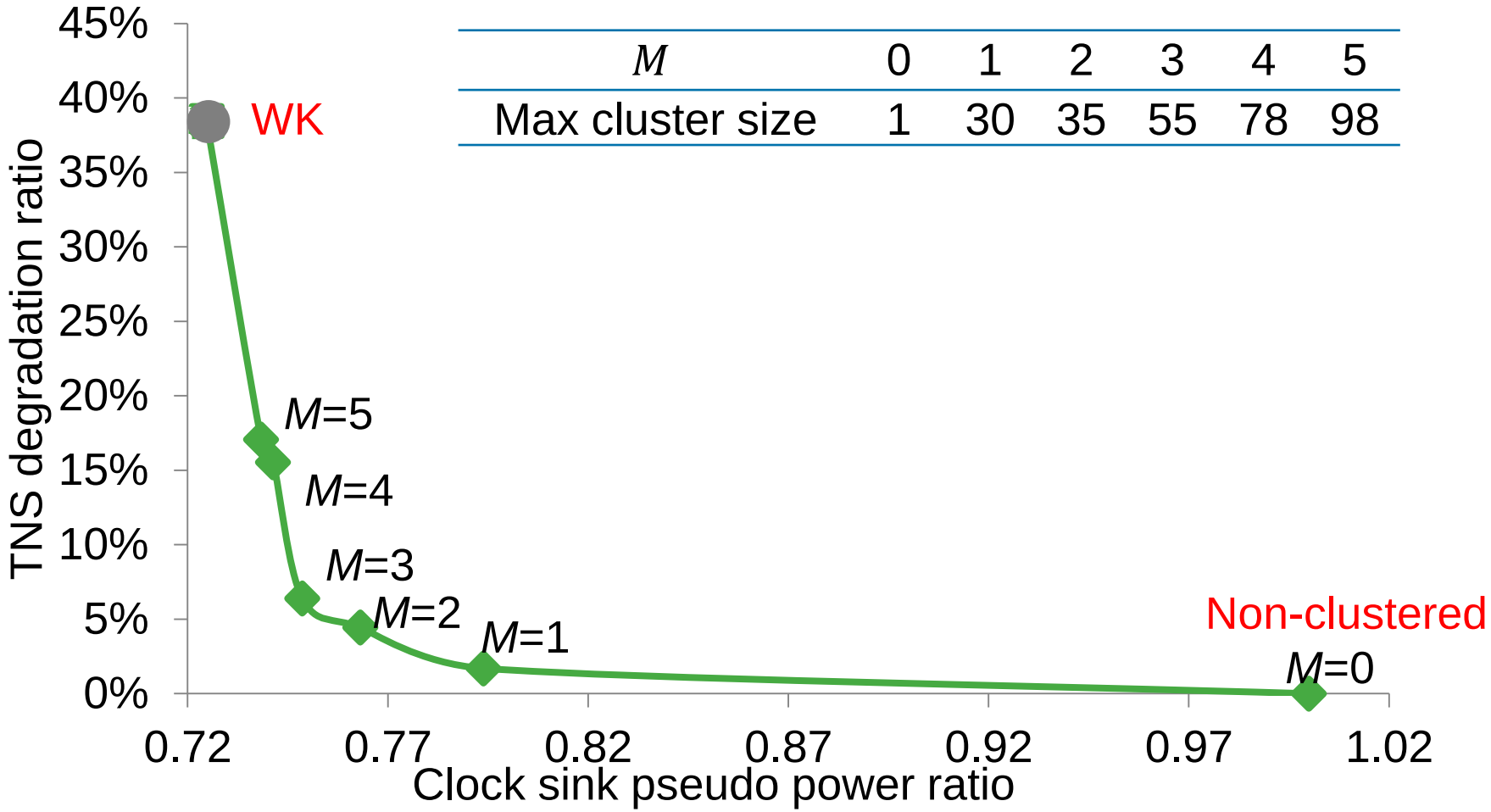


– Based on wirelength optimum site of each register

Clock Sink Power vs. TNS Degradation

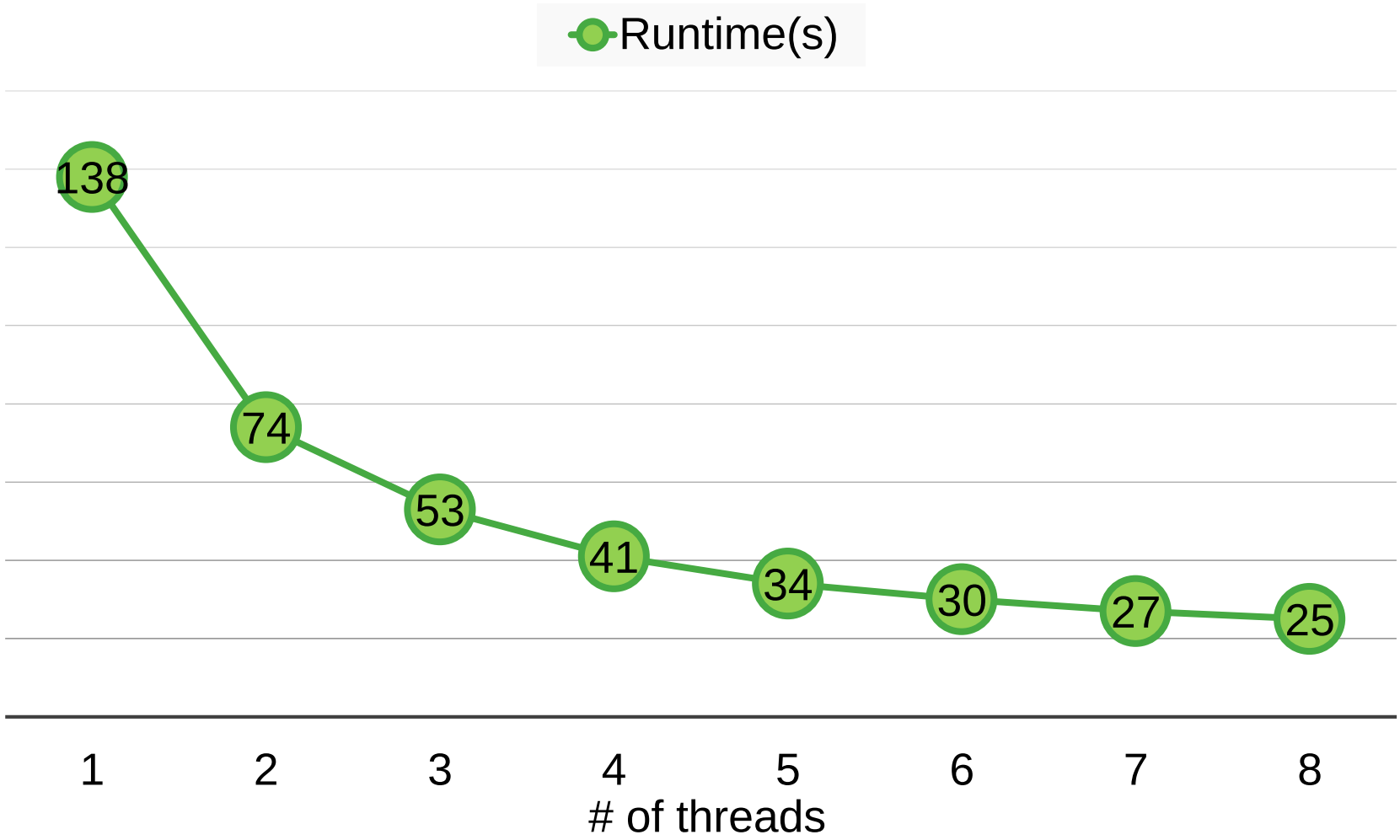
superblue16

M	0	1	2	3	4	5
Max cluster size	1	30	35	55	78	98



Speedups by Multithreading

superblue18



Conclusion

- 1) Requires no prespecified number of clusters
  - Exploits the density of registers to generate clusters naturally
- 2) Is insensitive to initializations
  - Actually, no initial seeds are needed
- 3) Is robust to outliers
  - Our effective neighbor consideration and bandwidth setting prevent outliers in sparse regions from over-displacement
- 4) Is tolerant of various register distributions
  - According to local density and sparsity, our clustering can tolerate uneven register distribution
- 5) Is efficient and scalable
  - Our KNN and bandwidth setting expedites shift vector computation for each register, and our algorithm is highly parallelizable
- 6) Balances power and timing
  - Graceful register clustering!

Thank you !

References

- S. I. Ward, N. Viswanathan, N. Y. Zhou, C. C. N. Sze, Z. Li, C. J. Alpert, and D. Z. Pan. 2013. Clock power minimization using structured latch templates and decision tree induction. *(ICCAD ’13)*. IEEE, Piscataway, NJ, USA, 599-606.
- D. A. Papa, C. J. Alpert, C. C. N. Sze, Z. Li, N. Viswanathan, G.-J. Nam, I. L. Markov. 2011. Physical synthesis with clock-network optimization for large systems on chips. *IEEE Micro* 31, 4 (July 2011), 51–62.
- M. P.-H. Lin, C. C. Hsu and Y.-T. Chang. 2011. Post-placement power optimization with multi-bit flip-flops. *IEEE Trans. on CAD of Integrated Circuits and Systems (TCAD)* 30, 12 (December 2011), 1870–1882.
- I. H.-R. Jiang, C.-L. Chang, and Y.-M. Yang. 2012. INTEGRA: Fast multibit flip-flop clustering for clock power saving. *IEEE Trans. on CAD of Integrated Circuits and Systems (TCAD)* 31, 2 (February 2012), 192–204.
- S.-H. Wang, Y.-Y. Liang, T.-Y. Kuo, and W.-K. Mak. 2012. Power-driven flip-flop merging and relocation. *IEEE Trans. on CAD of Integrated Circuits and Systems (TCAD)* 31, 2 (February 2012), 180–191.

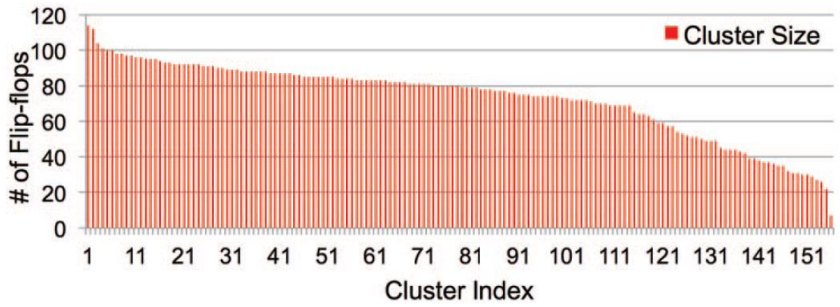
References

- S. S.-Y. Liu, W.-T. Lo, C.-J. Lee, and H.-M. Chen. 2013. Agglomerative-based flip-flop merging and relocation for signal wirelength and clock tree optimization. *ACM Trans. Design Automation Electronic Systems (TODAES)* 18, 3, Article 40 (July 2013), 20 pages.
- C.-C. Tsai, Y. Shi, G. Luo, and I. H.-R. Jiang. 2013. FF-Bond: Multi-bit flip-flop bonding at placement. In *Proc. Int'l Symp. on Physical Design (ISPD '13).* ACM, New York, NY, 147–153.
- G. Wu, Y. Xu, D. Wu, M. Ragupathy, Y.-Y. Mo, and C. Chu. 2016. Flip-flop clustering by weighted K-means algorithm. 2016. In *Proc. Design Automation Conf. (DAC '16)*. ACM, New York, NY, Article 82, 6 pages.
- A. B. Kahng, J. Li, and L. Wang. 2016. Improved flop tray-based design implementation for power reduction. In *Proc. Int'l Conf. on Computer-Aided Design (ICCAD '16)*. ACM, New York, NY, Article 20, 8 pages.
- I. Seitanidis, G. Dimitrakopoulos, P. M. Mattheakis, L. Masse-Navette, D. Chinnery. 2018. Timing-driven and placement-aware multi-bit register composition. *IEEE Trans. on CAD of Integrated Circuits and Systems (TCAD)*, early access.

Weighted K-Means

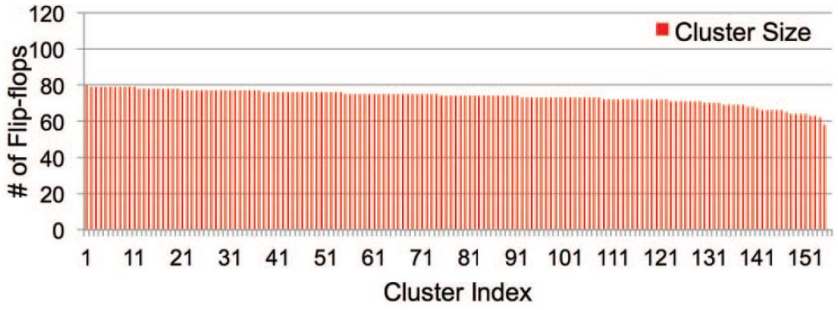
- **Vanilla K-Means**
  - Incur unbalanced cluster sizes

$$Cost = |x_i - \mu_x(C_k)| + |y_i - \mu_y(C_k)|$$



- **Weighted K-Means**
  - Try to generate even-sized clusters

$$Cost = (|x_i - \mu_x(C_k)| + |y_i - \mu_y(C_k)|) * max(\ (|C_k|/size\_limit),\ 1\ )$$



- Introduce a cluster size balancing weight into displacement cost
- Intend to form large cluster (nearly max. allowable bits)
- Need additional over-displacement and over-size fixing

Variable Bandwidth Selection

- Reflect timing criticality and local distribution
  - $f(x) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{{h_i}^d} k\left(\frac{x-x_i}{h_i}\right)$
  - $m(x) = \frac{\sum_{i=1}^{n} \frac{x_i}{{h_i}^{d+2}} g\left(\left\|\frac{x-x_i}{h_i}\right\|^2\right)}{\sum_{i=1}^{n} \frac{1}{{h_i}^{d+2}} g\left(\left\|\frac{x-x_i}{h_i}\right\|^2\right)} - x$
  - $h_i = \min\left(h_{\max}, \alpha\left\|x_i - x_{i,M}\right\|\right)$
  - $h_{\max}$: the maximum allowable displacement
  - $\left\|x_i - x_{i,M}\right\|$: the Euclidean distance between register i and its *M*-th nearest neighbor ($x_{i,0} = x_i$)
  - α: a timing criticality coefficient; $\alpha \to 0$ for the most critical register (i.e., a very tall and skinny kernel)

Identifying Effective Neighbors

- Classic mean shift considers all original data points during shift vector computation
- However, the points that correspond to the tails of the underlying density function receive small weights, and thus they are almost automatically discarded.
- Moreover, we do not expect registers to travel far away (for minimizing disturbance to timing and placement), and try to avoid oversized clusters.
- Thus, we can ignore distant registers
  - $f(x) = \frac{1}{n} \sum_{i \in KNN'(x)} \frac{1}{{h_i}^d} k\left(\frac{x - x_i}{h_i}\right)$
  - $m(x) = \frac{\sum_{i \in KNN'(x)} \frac{x_i}{{h_i}^{d+2}} g\left(\left\|\frac{x - x_i}{h_i}\right\|^2\right)}{\sum_{i \in KNN'(x)} \frac{1}{{h_i}^{d+2}} g\left(\left\|\frac{x - x_i}{h_i}\right\|^2\right)} - x$

Relocation for Timing and Displacement

- The previous steps in effective mean shift can be viewed as seeking the locations of clusters.
- Reassign registers and relocate clusters for improving timing and displacement.
  - Reduce to stable matching
  - The capacity of a cluster location equals the maximum allowable cluster size.
  - The preference is ranked in non-decreasing order of displacement (Manhattan distance)
- Relocate each cluster to the median coordinate of its register members for minimizing displacement and reducing timing degradation.

Experimental Setting

- C++ programming language and compiled by G++ 4.8.5
- Intel Xeon 2.6 GHz CPU and 256 GB memory
- ICCAD-2015 CAD contest in incremental timing-driven placement benchmark suite
- Cadence Innovus

Table 3. Benchmark Statistics.

Circuit	# of Cells	# of Registers
superblue16	981,559	142,543
superblue18	768,068	101,758
superblue4	796,645	167,731
superblue5	1,086,888	110,941
superblue3	1,213,253	163,107
superblue1	1,209,716	137,560
superblue7	1,931,639	262,176
superblue10	1,876,130	231,747

Table 4. Pseudo Power of Multi-bit Register Library.

# of Bits	Normalized Pseudo Power per Bit
1	1.000
2~3	0.860
4~7	0.790
8~15	0.755
16~31	0.738
32~63	0.729
64~80	0.724