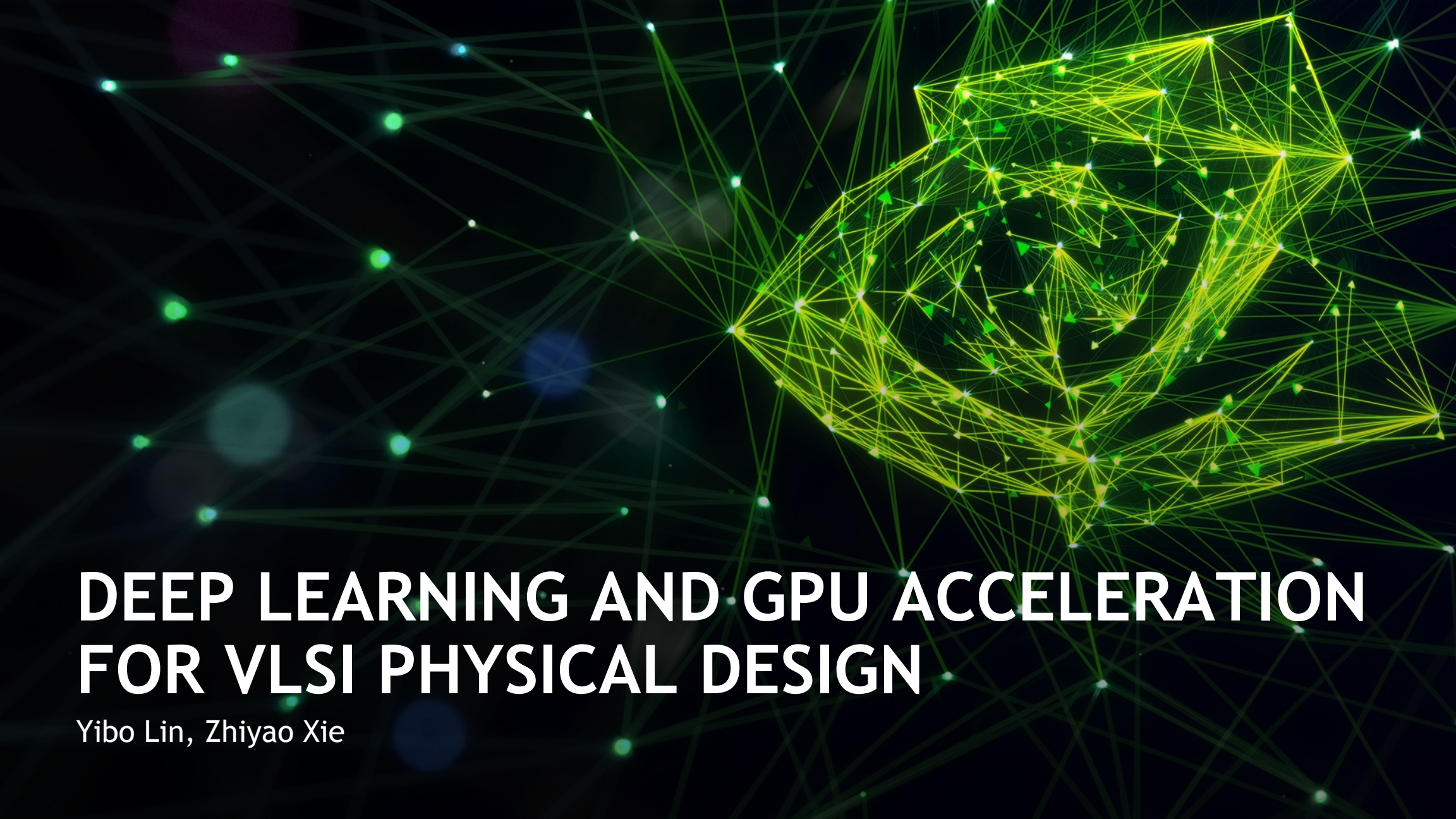


DEEP LEARNING AND GPU ACCELERATION FOR VLSI PHYSICAL DESIGN

Yibo Lin, Zhiyao Xie

PART 1

DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement

Yibo Lin, Shounak Dhar, Wuxi Li,
David Z. Pan

Mark Ren, Brucek Khailany

PART 2

RouteNet: Routability Prediction for Mixed-Size Designs Using Convolutional Neural Network

Zhiyao Xie, Yiran Chen

Duke UNIVERSITY

Yu-Hung Huang, Guan-Qi Fang, Shao-Yun Fang

TAIWAN TECH
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Mark Ren, Brucek Khailany

NVIDIA®

Jiang Hu

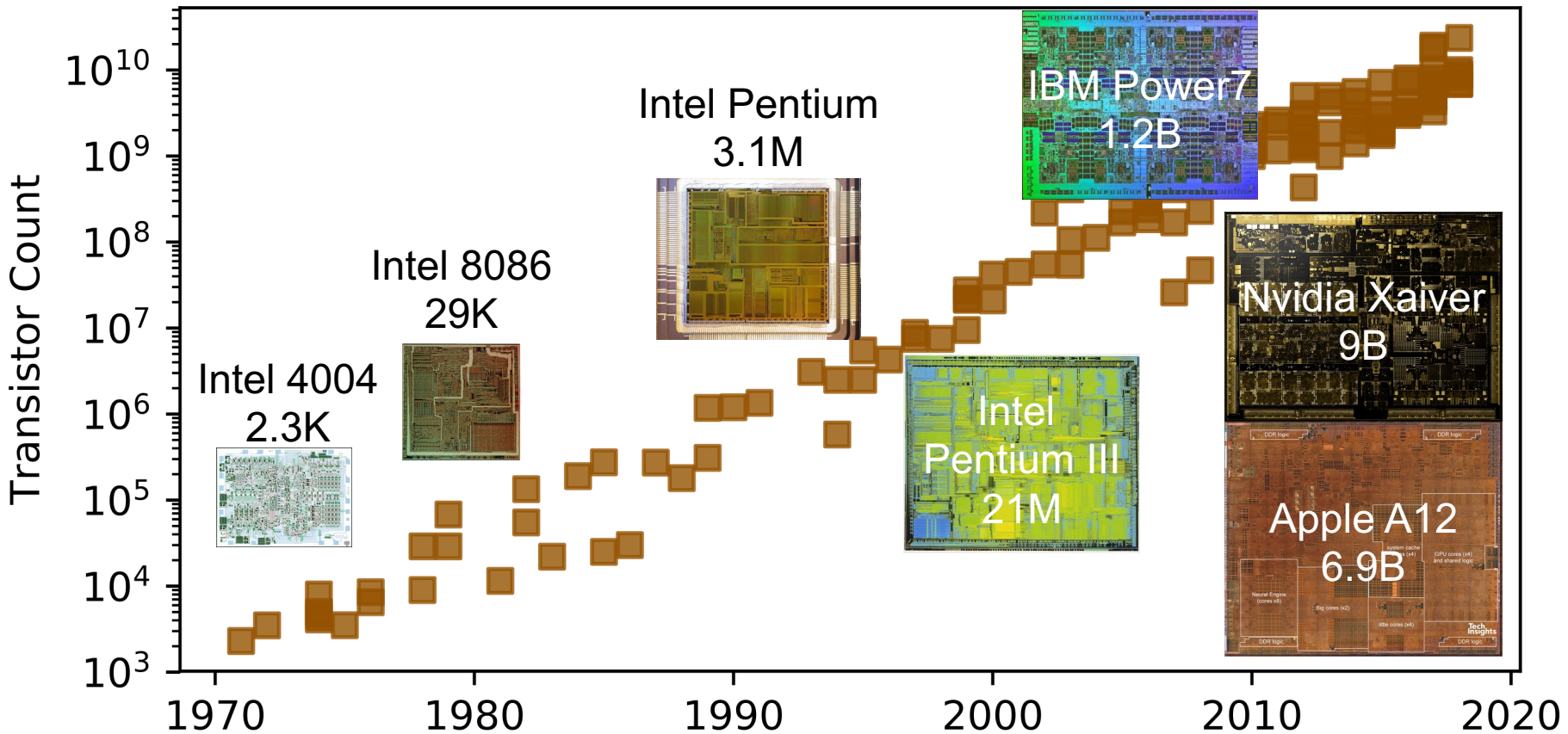
TEXAS A&M UNIVERSITY

DREAMPLACE

- Introduction
- Problem Formulation & Related Work
- DREAMPlace Framework
- Experiment Results
- Conclusion

INCREASING COMPLEXITY OF IC DESIGNS

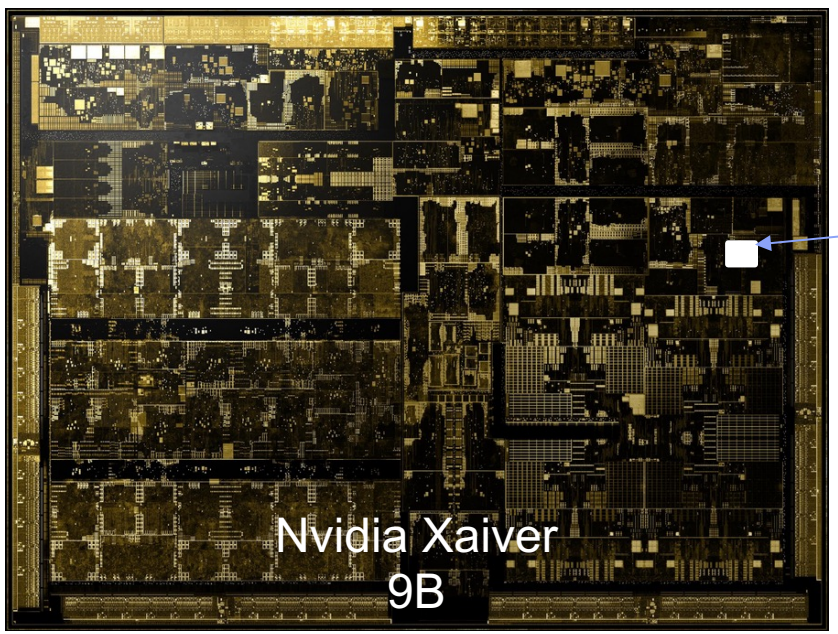
Design complexity ⇧ | Design cycles ⇧ | Design costs ⇧



NVIDIA

INCREASING COMPLEXITY OF IC DESIGNS

Design complexity ⇧ | Design cycles ⇧ | Design costs ⇧



Divide a chip into small partitions
for backend design
e.g., **1~2M** cells per partition

Turn-around time for 1 iteration of backend flow
3~6 days for one partition

Backend flow includes placement and routing optimization, clock tree synthesis, post routing optimization.

STATE-OF-THE-ART VLSI PLACEMENT

Placement is critical to VLSI design quality and design closure

Input

Gate-level netlist
Standard cell library

Output

Legal placement solution

Objective

Optimize wirelength, routability, etc.

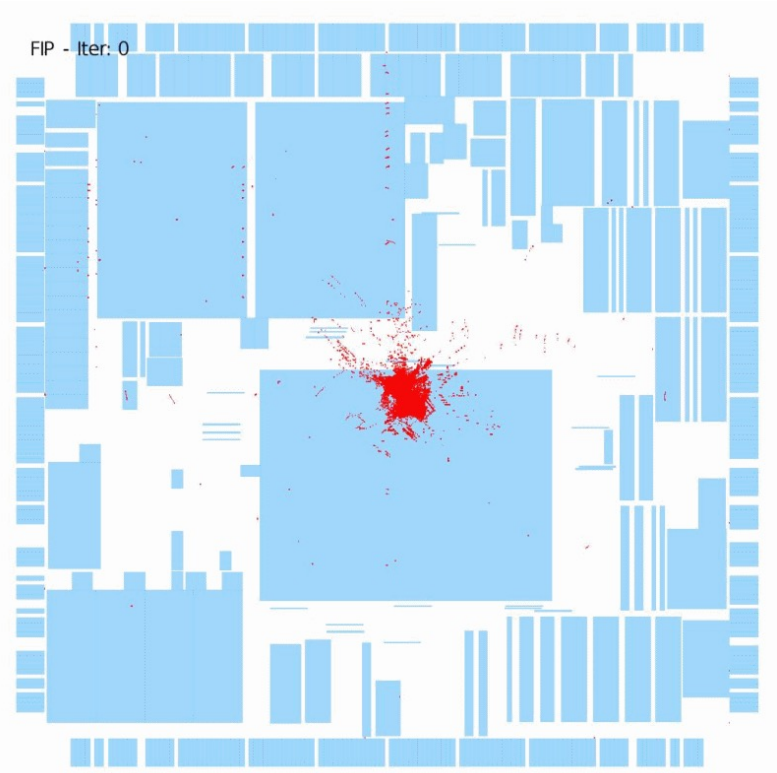
Typical approaches

Quadratic placement

- SimPL: [TCAD'12,Kim+]
- Ripple: [DAC'13'He+]
- POLAR: [TCAD'15,Lin+]

Nonlinear placement

- mPL: [MCP'07,Chan+]
- APlace: [ISPD'06,Kahng+]
- NTUplace: [TCAD'08,Chen+]
- ePlace: [TCAD'15,Lu+]
  RePlAce: [TCAD'18,Cheng+]

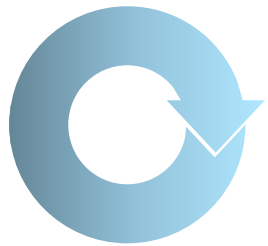


RePlAce

TYPICAL NONLINEAR PLACEMENT ALGORITHM

Objective of nonlinear placement

$$\min \quad \underbrace{\left(\sum_{e \in E} \mathrm{WL}(e; \mathbf{x}, \mathbf{y})\right)}_{\text{Wirelength}} + \lambda \underbrace{D(\mathbf{x}, \mathbf{y})}_{\text{Density}}$$



Gradient descent iterations

Ultrafast placement
No quality degradation
Low development overhead

What we desire?

Poor runtime-quality trade-off
E.g., >3h for a 10M-cell design
Huge development efforts

What we have?

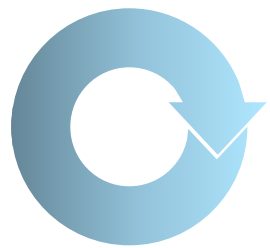
Drop to 5min

?

TYPICAL NONLINEAR PLACEMENT ALGORITHM

Objective of nonlinear placement

$$\min \quad \underbrace{\left(\sum_{e \in E} \mathrm{WL}(e; \mathbf{x}, \mathbf{y})\right)}_{\text{Wirelength}} + \lambda \underbrace{D(\mathbf{x}, \mathbf{y})}_{\text{Density}}$$



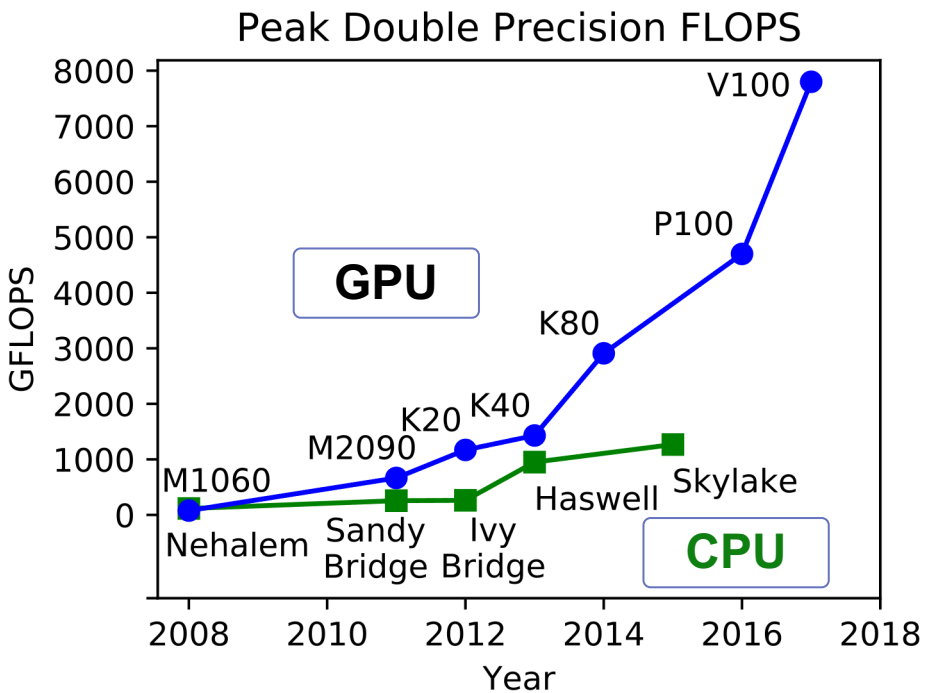
Gradient descent iterations

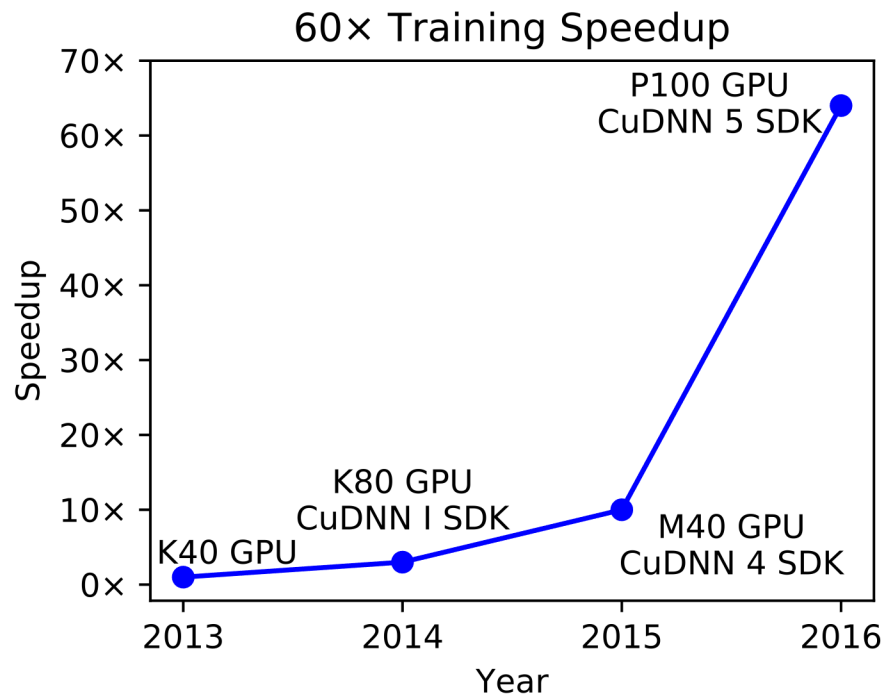
Previous work on acceleration

- POLAR 3.0 [ICCAD'15,Lin+]
  - Quadratic placement
  - Geometric partitioning
  - 4× speedup with 16 threads on CPU
  - 2-6% quality degradation
- mPL GPU acceleration [ICCAD'09,Cong+]
  - Clustering + Nonlinear placement
  - 15× speedup with < 1% quality degradation
  - CUDA development overhead

GPU EVOLUTION WITH DEEP LEARNING

Booming performance especially in deep learning

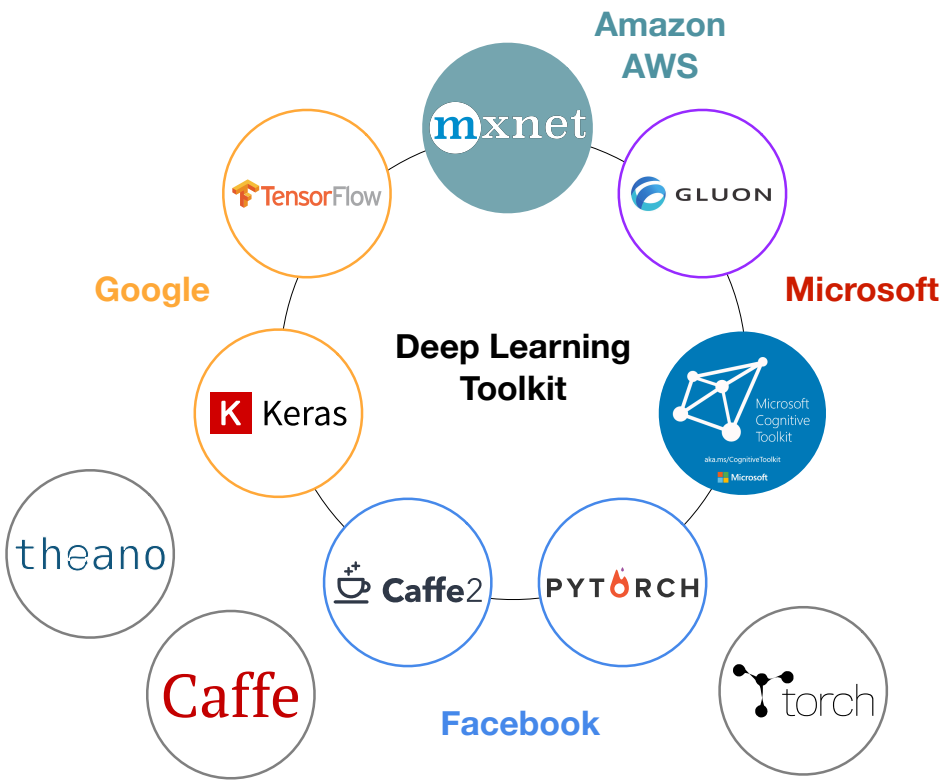


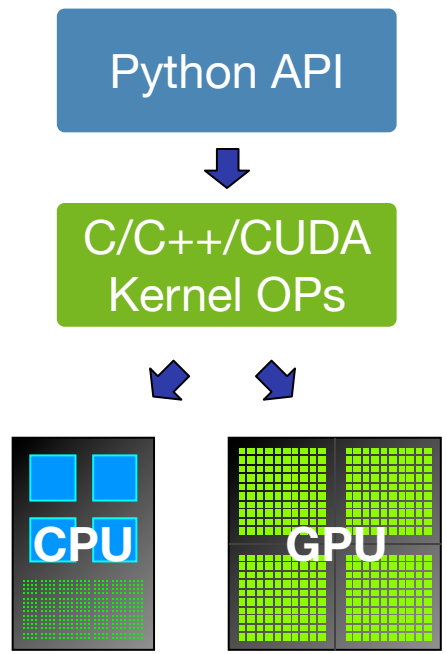


[Courtesy by Nvidia]

ADVANCES IN DEEP LEARNING TOOLKIT

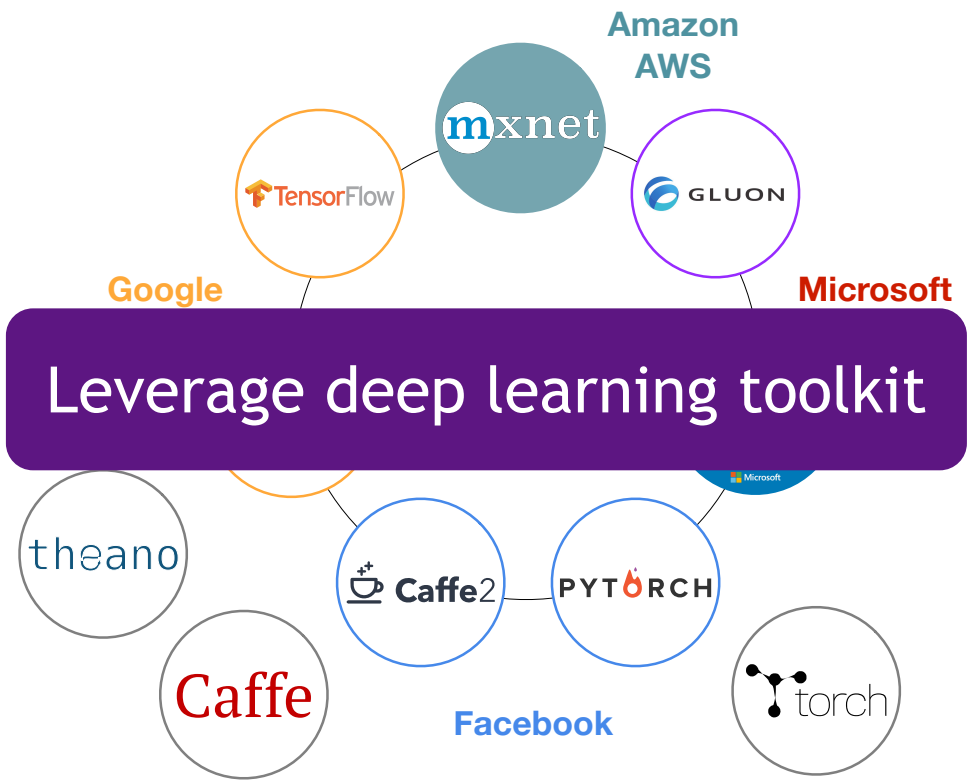
Easy-to-use API | Functionality | Multi-platform





ADVANCES IN DEEP LEARNING TOOLKIT

Easy-to-use API | Functionality | Multi-platform



Desired Placement Engine

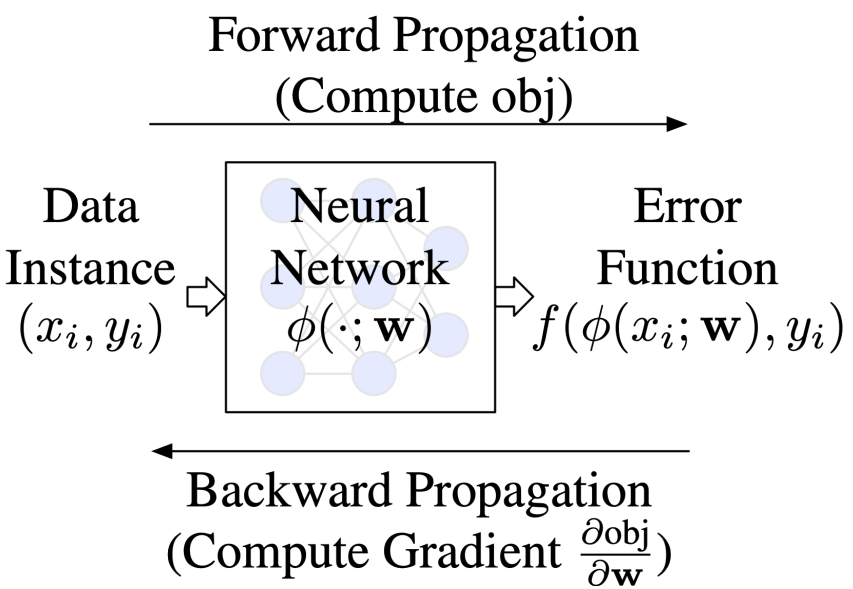
- Ultrafast
- High-quality
- Cheap development

10M-cell design in 5min ?

LEVERAGE DEEP LEARNING TOOLKIT

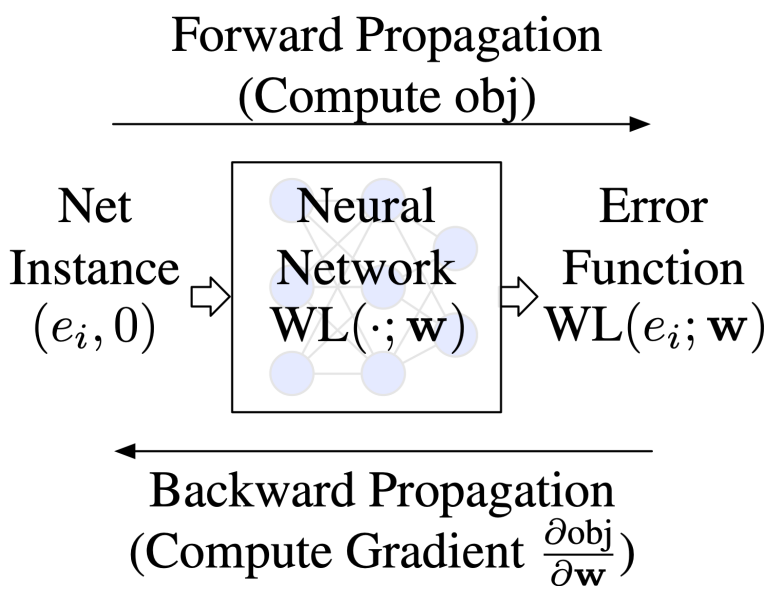
Accelerating placement using deep learning toolkit ??

$$\min_{\mathbf{w}} \sum_{i}^{n} f(\phi(x_i; \mathbf{w}), y_i) + \lambda R(\mathbf{w})$$



Train a neural network

$$\min_{\mathbf{w}} \sum_{i}^{n} \text{WL}(e_i; \mathbf{w}) + \lambda D(\mathbf{w})$$



Solve a placement

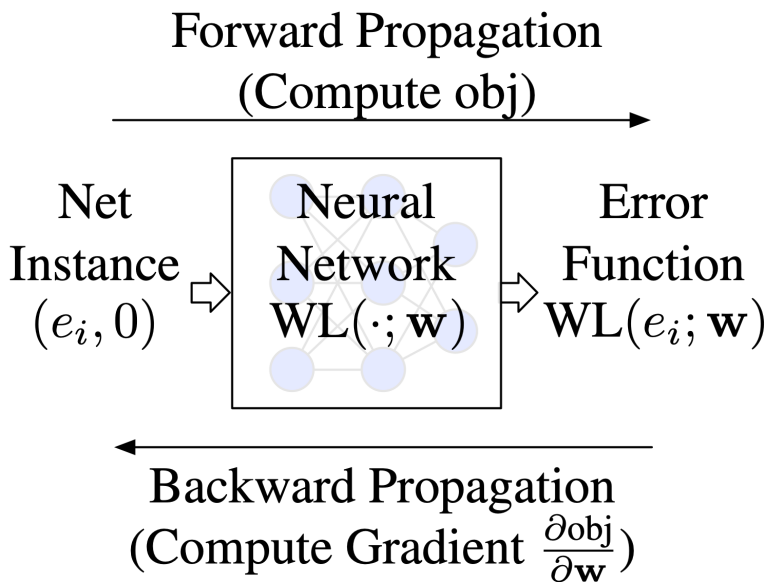
LEVERAGE DEEP LEARNING TOOLKIT

Accelerating placement using deep learning toolkit ??

```
# x: features, y: golden labels
# define optimizer
optimizer = SGD(model.parameters())
for step in range(500):
    # forward: predict labels
    y_pred = model(x)
    loss = loss_fn(y_pred, y)
    model.zero_grad()
    # backward: compute gradient
    loss.backward()
    # update parameters
    optimizer.step()
```

Train a neural network

$$\min_{\mathbf{w}} \sum_{i}^{n} \mathrm{WL}(e_i; \mathbf{w}) + \lambda D(\mathbf{w})$$



Solve a placement

LEVERAGE DEEP LEARNING TOOLKIT

Accelerating placement using deep learning toolkit ??

```
# x: features, y: golden labels
# define optimizer
optimizer = SGD(model.parameters())
for step in range(500):
    # forward: predict labels
    y_pred = model(x)
    loss = loss_fn(y_pred, y)
    model.zero_grad()
    # backward: compute gradient
    loss.backward()
    # update parameters
    optimizer.step()
```

Train a neural network

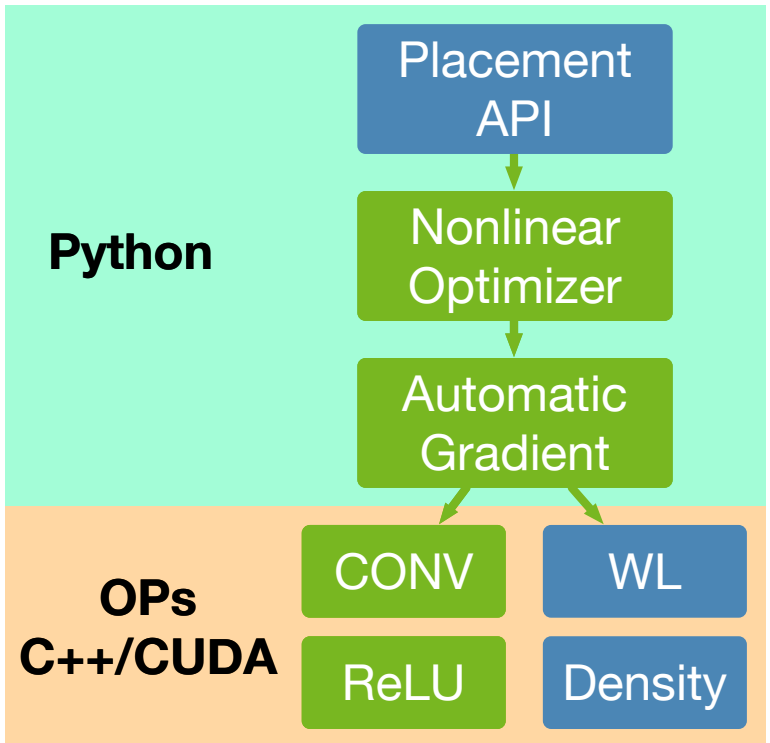
Solve a placement

Advantages from the Analogy

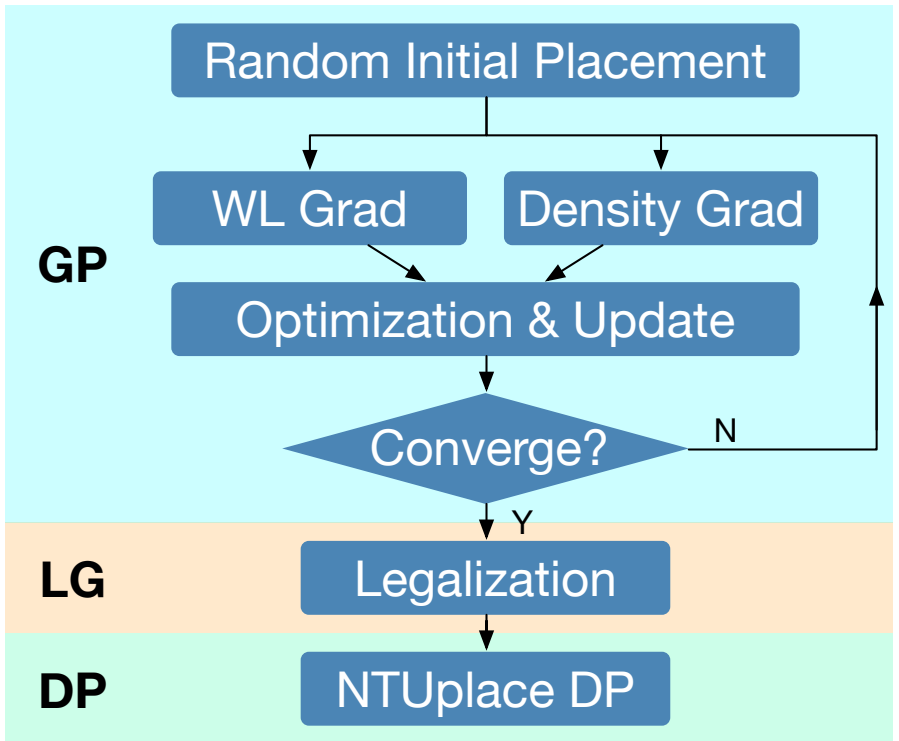
- Mature toolkit for placement
  - Well maintained
  - Highly optimized
- Multi-platform support
  - CPU/GPU
- Low development overhead
  - AutoGrad
  - Python API
- Optimization solver options
  - SGD, momentum, Adam, etc.

SOFTWARE ARCHITECTURE & OVERALL FLOW

Mature deep learning toolkit



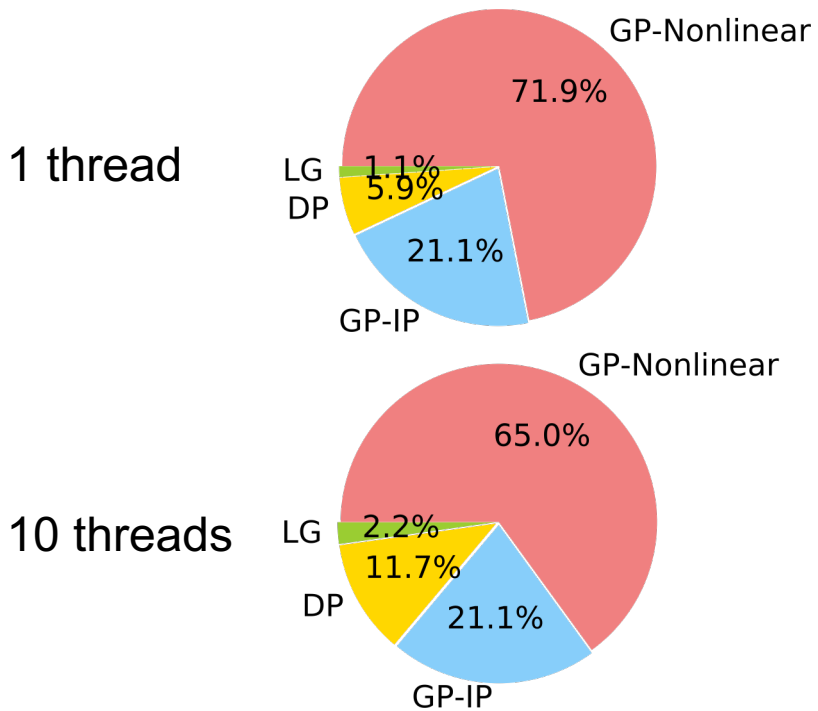
DREAMPlace architecture



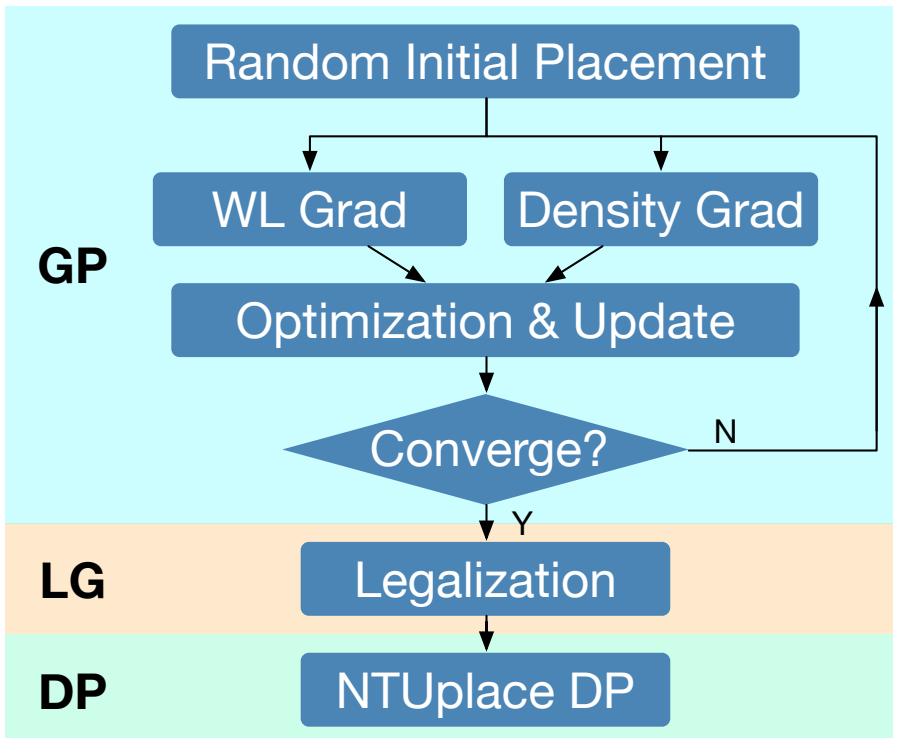
DREAMPlace flow

SOFTWARE ARCHITECTURE & OVERALL FLOW

RePlAce runtime breakdown



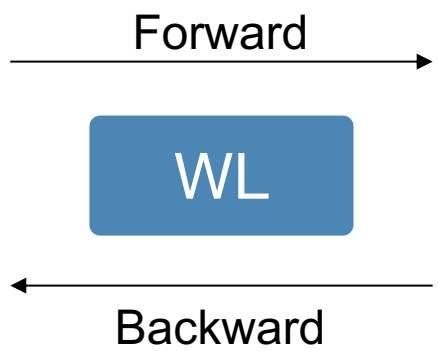
GP take >3h on 10M-cell designs



DREAMPlace flow

ACCELERATING KERNEL OPS

Wirelength OP

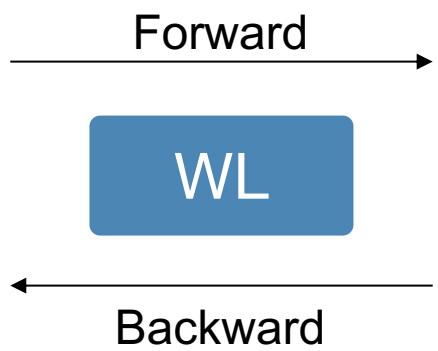


Smoothing HPWL by weighted average wirelength

$$\mathrm{WA}_e = \frac{\sum_{i \in e} x_i e^{\frac{x_i}{\gamma}}}{\sum_{i \in e} e^{\frac{x_i}{\gamma}}} - \frac{\sum_{i \in e} x_i e^{-\frac{x_i}{\gamma}}}{\sum_{i \in e} e^{-\frac{x_i}{\gamma}}}$$

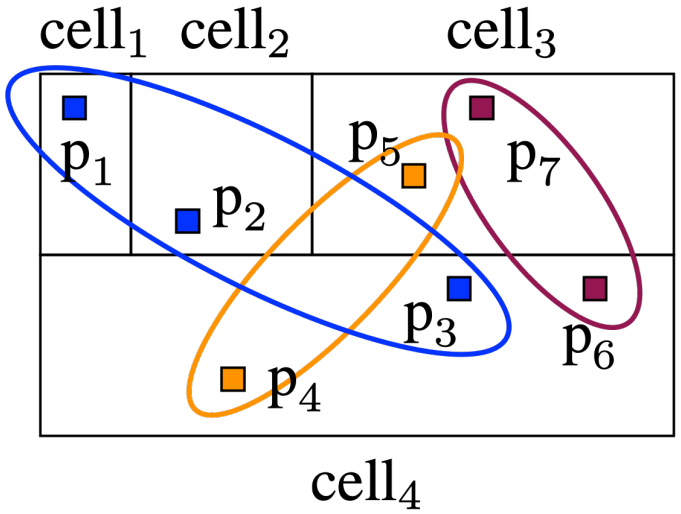
ACCELERATING KERNEL OPS

Wirelength OP



Smoothing HPWL by weighted average wirelength

$$\mathrm{WA}_e = \frac{\sum_{i \in e} x_i e^{\frac{x_i}{\gamma}}}{\sum_{i \in e} e^{\frac{x_i}{\gamma}}} - \frac{\sum_{i \in e} x_i e^{-\frac{x_i}{\gamma}}}{\sum_{i \in e} e^{-\frac{x_i}{\gamma}}}$$



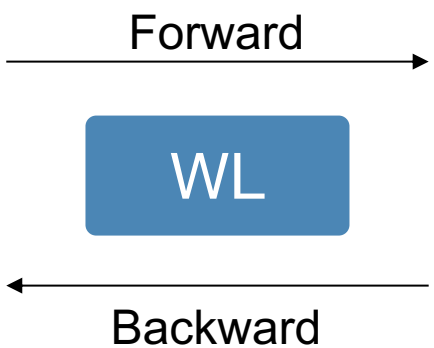
$\mathrm{net}_1 : \{p_1, p_2, p_3\}$

$\mathrm{net}_2 : \{p_4, p_5\}$

$\mathrm{net}_3 : \{p_6, p_7\}$

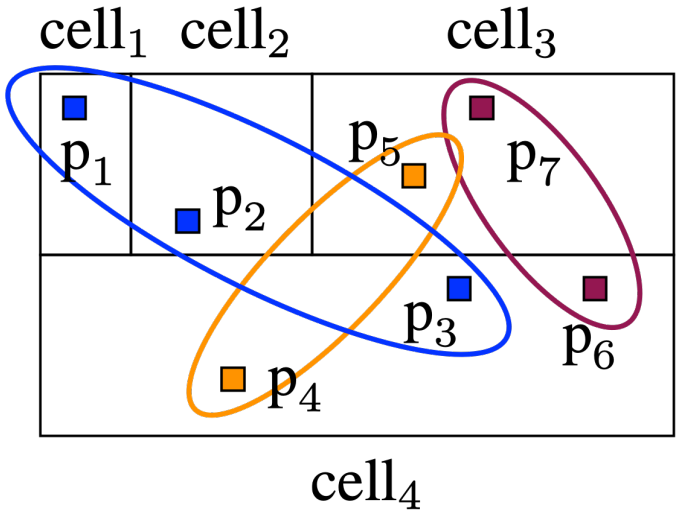
ACCELERATING KERNEL OPS

Wirelength OP



Smoothing HPWL by weighted average wirelength

$$\mathrm{WA}_e = \frac{\sum_{i\in e} x_i e^{\frac{x_i}{\gamma}}}{\sum_{i\in e} e^{\frac{x_i}{\gamma}}} - \frac{\sum_{i\in e} x_i e^{-\frac{x_i}{\gamma}}}{\sum_{i\in e} e^{-\frac{x_i}{\gamma}}}$$

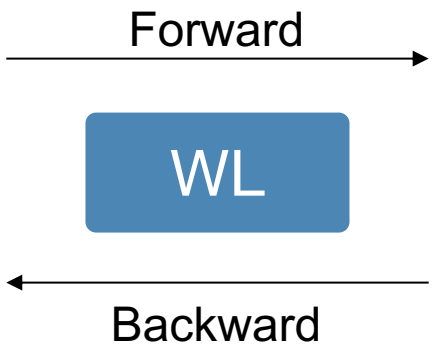


$$\mathrm{WA}_1 = \frac{x_1 e^{\frac{x_1}{\gamma}} + x_2 e^{\frac{x_2}{\gamma}} + x_3 e^{\frac{x_3}{\gamma}}}{e^{\frac{x_1}{\gamma}} + e^{\frac{x_2}{\gamma}} + e^{\frac{x_3}{\gamma}}} - \frac{x_1 e^{-\frac{x_1}{\gamma}} + x_2 e^{-\frac{x_2}{\gamma}} + x_3 e^{-\frac{x_3}{\gamma}}}{e^{-\frac{x_1}{\gamma}} + e^{-\frac{x_2}{\gamma}} + e^{-\frac{x_3}{\gamma}}}$$

$$\mathrm{WA}_2 = \frac{x_4 e^{\frac{x_4}{\gamma}} + x_5 e^{\frac{x_5}{\gamma}}}{e^{\frac{x_4}{\gamma}} + e^{\frac{x_5}{\gamma}}} - \frac{x_4 e^{-\frac{x_4}{\gamma}} + x_5 e^{-\frac{x_5}{\gamma}}}{e^{-\frac{x_4}{\gamma}} + e^{-\frac{x_5}{\gamma}}}$$

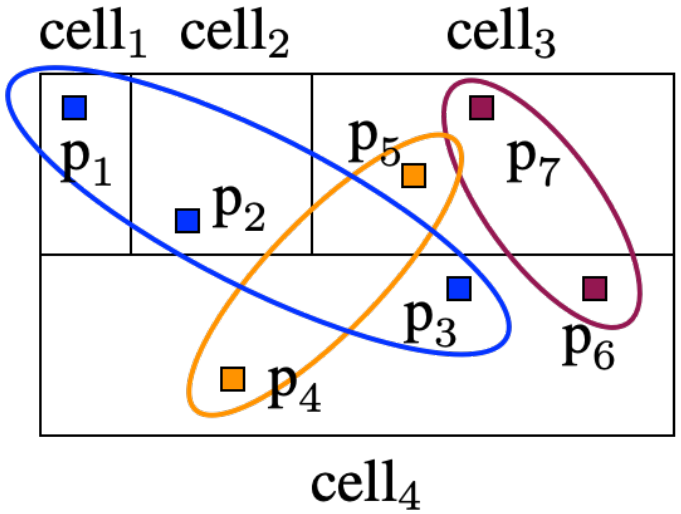
ACCELERATING KERNEL OPS

Wirelength OP



Smoothing HPWL by weighted average wirelength

$$\mathrm{WA}_e = \frac{\sum_{i \in e} x_i e^{\frac{x_i}{\gamma}}}{\sum_{i \in e} e^{\frac{x_i}{\gamma}}} - \frac{\sum_{i \in e} x_i e^{-\frac{x_i}{\gamma}}}{\sum_{i \in e} e^{-\frac{x_i}{\gamma}}}$$



$$\mathrm{WA}_1 = \frac{x_1 e^{\frac{x_1}{\gamma}} + x_2 e^{\frac{x_2}{\gamma}} + x_3 e^{\frac{x_3}{\gamma}}}{e^{\frac{x_1}{\gamma}} + e^{\frac{x_2}{\gamma}} + e^{\frac{x_3}{\gamma}}} + \frac{x_4 e^{\frac{x_4}{\gamma}} + x_5 e^{\frac{x_5}{\gamma}}}{e^{\frac{x_4}{\gamma}} + e^{\frac{x_5}{\gamma}}}$$

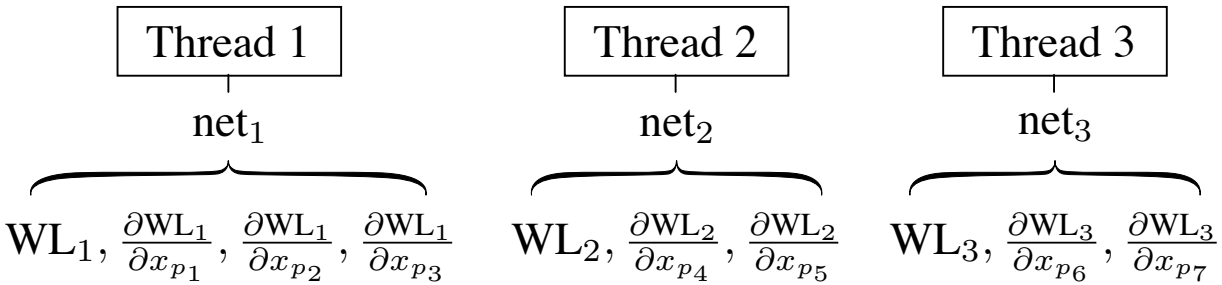
$$- \frac{x_1 e^{-\frac{x_1}{\gamma}} + x_2 e^{-\frac{x_2}{\gamma}} + x_3 e^{-\frac{x_3}{\gamma}}}{e^{-\frac{x_1}{\gamma}} + e^{-\frac{x_2}{\gamma}} + e^{-\frac{x_3}{\gamma}}} - \frac{x_4 e^{-\frac{x_4}{\gamma}} + x_5 e^{-\frac{x_5}{\gamma}}}{e^{-\frac{x_4}{\gamma}} + e^{-\frac{x_5}{\gamma}}}$$

Hard to implement with existing tensor OPs

MASSIVE PARALLELIZATION STRATEGIES

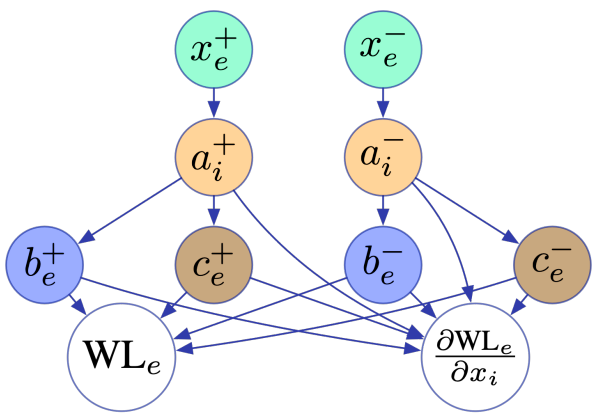
Strategy I: Net-by-Net

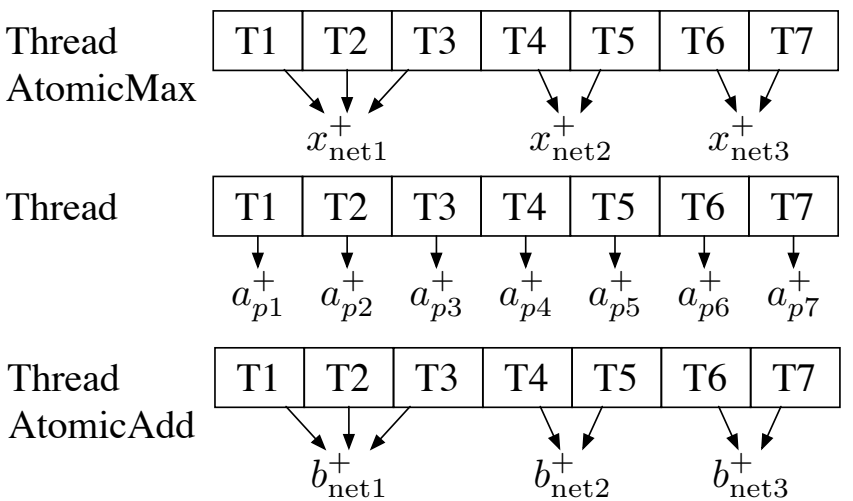
- Allocate one thread for each net
- No synchronization required
- Lack of parallelization for heterogenous net degrees



Strategy II: by-Pin

- Allocate one thread for each pin



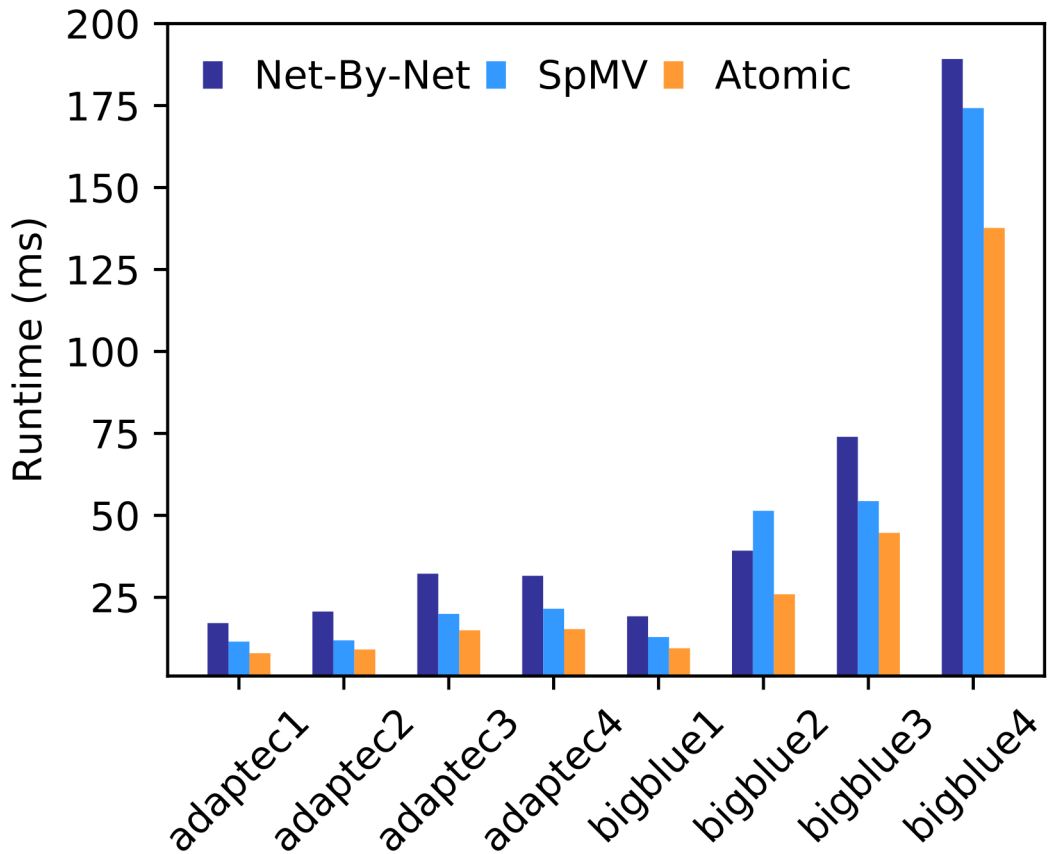


Use **SpMV**?
[DATE'18,Lin+]

MASSIVE PARALLELIZATION STRATEGIES

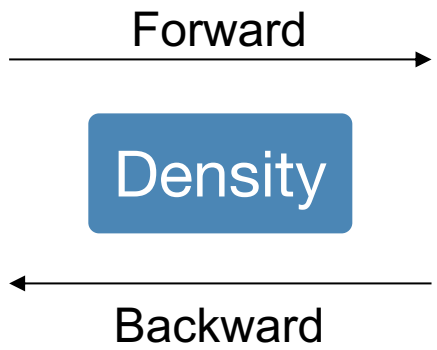
ISPD 2005 Contest Benchmarks

- *Atomic* is **1.9**× faster than *Net-by-Net*
- *Atomic* is **1.4**× faster than *SpMV*
- Speedup is very consistent



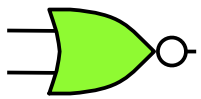
ACCELERATING KERNEL OPS

Density OP

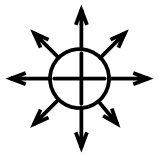


Electrostatic system analogy

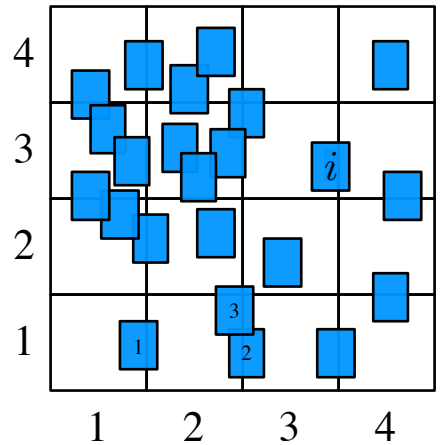
Cell instance



Electric particle



ρ

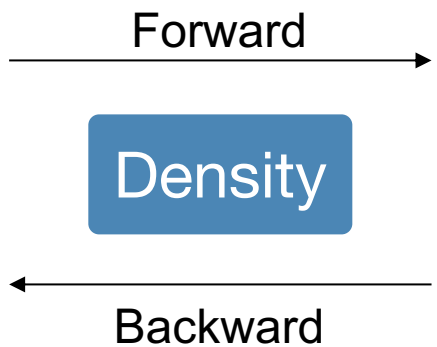


Poisson's Equation

$$\begin{cases} \nabla \cdot \nabla \psi(x,y) = -\rho(x,y), \\ \hat{\mathbf{n}} \cdot \nabla \psi(x,y) = \mathbf{0}, (x,y) \in \partial R, \\ \iint_R \rho(x,y) = \iint_R \psi(x,y) = 0 \end{cases}$$

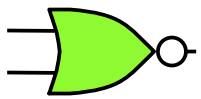
ACCELERATING KERNEL OPS

Density OP

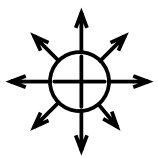


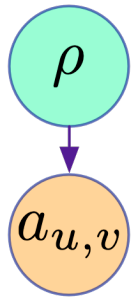
Electrostatic system analogy

Cell instance



Electric particle

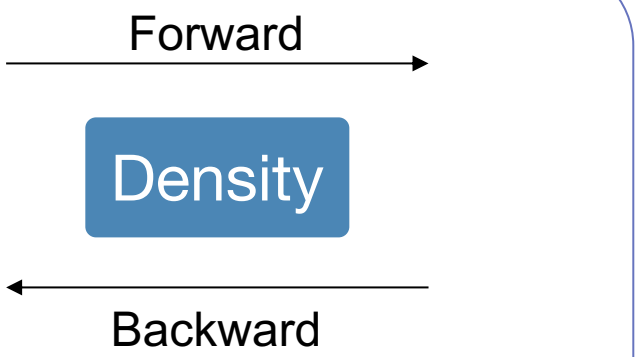




$$a_{u,v} = \mathrm{DCT}(\mathrm{DCT}(\rho)^T)^T$$

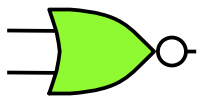
ACCELERATING KERNEL OPS

Density OP

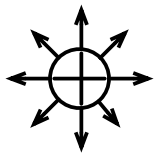


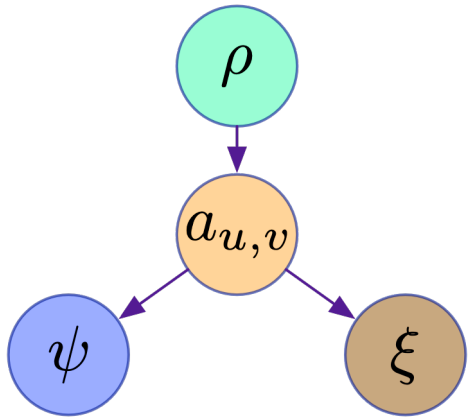
Electrostatic system analogy

Cell instance



Electric particle





$$a_{u,v} = \mathrm{DCT}(\mathrm{DCT}(\rho)^T)^T$$

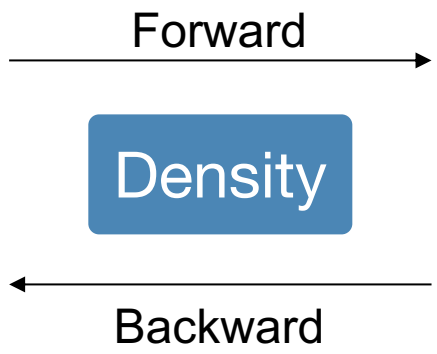
$$\psi_{\mathrm{DCT}} = \mathrm{IDCT}(\mathrm{IDCT}(\{a_{u,v}\})^T)^T$$

$$\xi^X_{\mathrm{DSCT}} = \mathrm{IDXST}(\mathrm{IDCT}(\{\frac{a_{u,v}w_u}{w_u^2+w_v^2}\})^T)^T$$

$$\xi^Y_{\mathrm{DCST}} = \mathrm{IDCT}(\mathrm{IDXST}(\{\frac{a_{u,v}w_v}{w_u^2+w_v^2}\})^T)^T$$

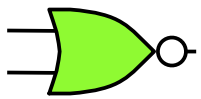
ACCELERATING KERNEL OPS

Density OP

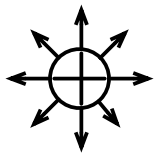


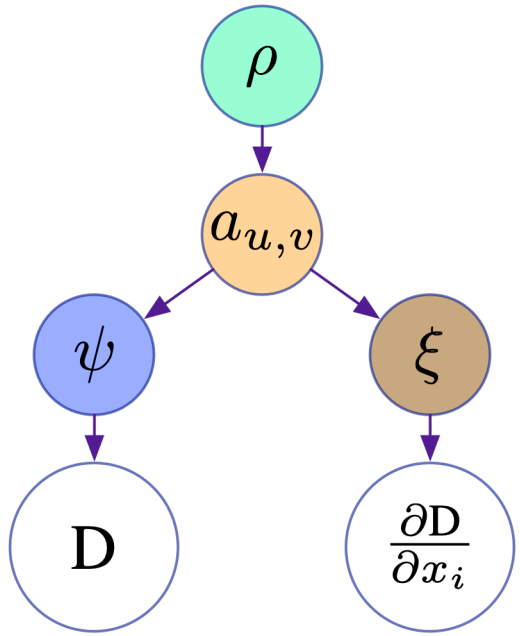
Electrostatic system analogy

Cell instance



Electric particle





$$a_{u,v} = \mathrm{DCT}(\mathrm{DCT}(\rho)^T)^T$$

$$\psi_{\mathrm{DCT}} = \mathrm{IDCT}(\mathrm{IDCT}(\{a_{u,v}\})^T)^T$$

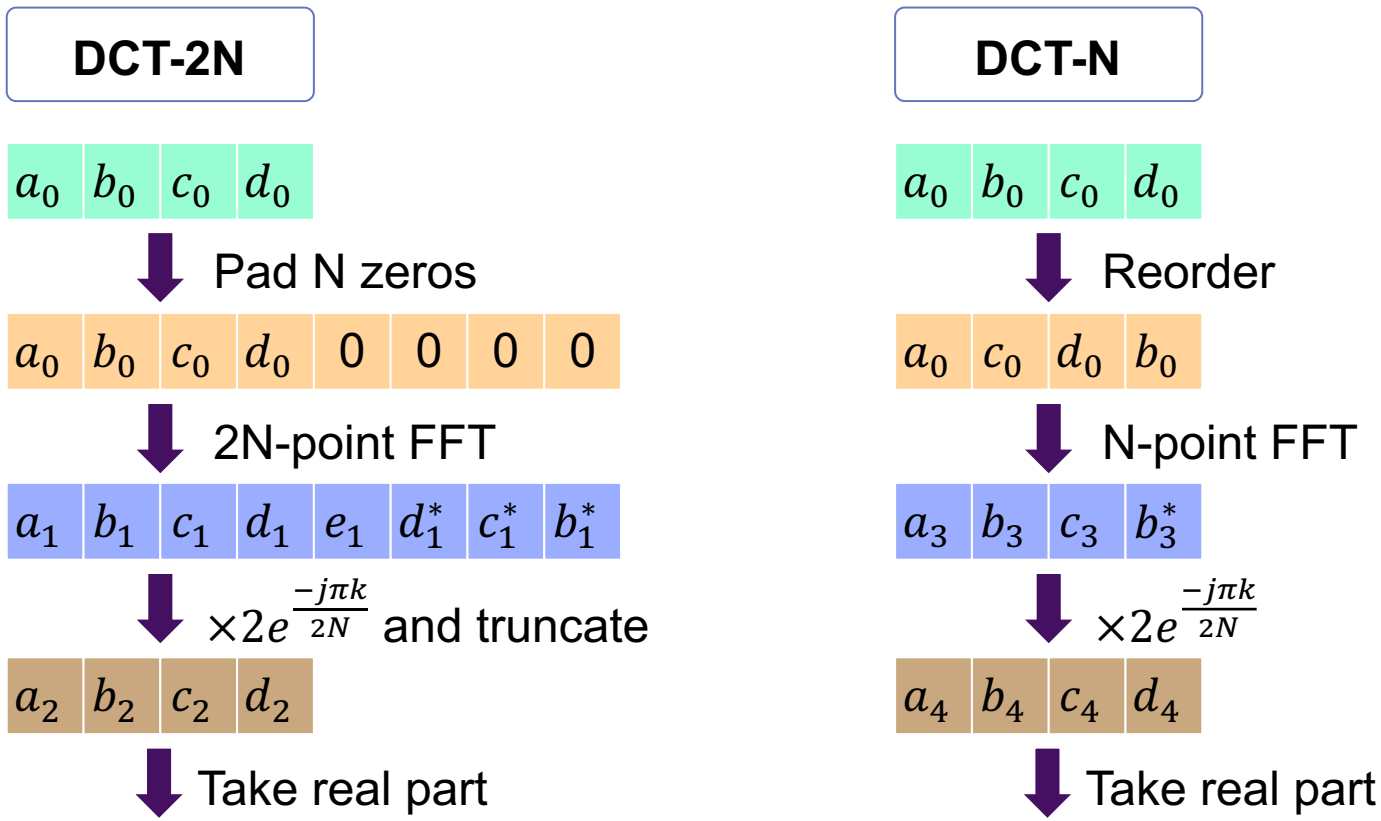
$$\xi^X_{\mathrm{DSCT}} = \mathrm{IDXST}(\mathrm{IDCT}(\{\frac{a_{u,v}w_u}{w_u^2+w_v^2}\})^T)^T$$

$$\xi^Y_{\mathrm{DCST}} = \mathrm{IDCT}(\mathrm{IDXST}(\{\frac{a_{u,v}w_v}{w_u^2+w_v^2}\})^T)^T$$

ACCELERATING DCT/IDCT

Leverage highly optimized FFT kernels in deep learning toolkit

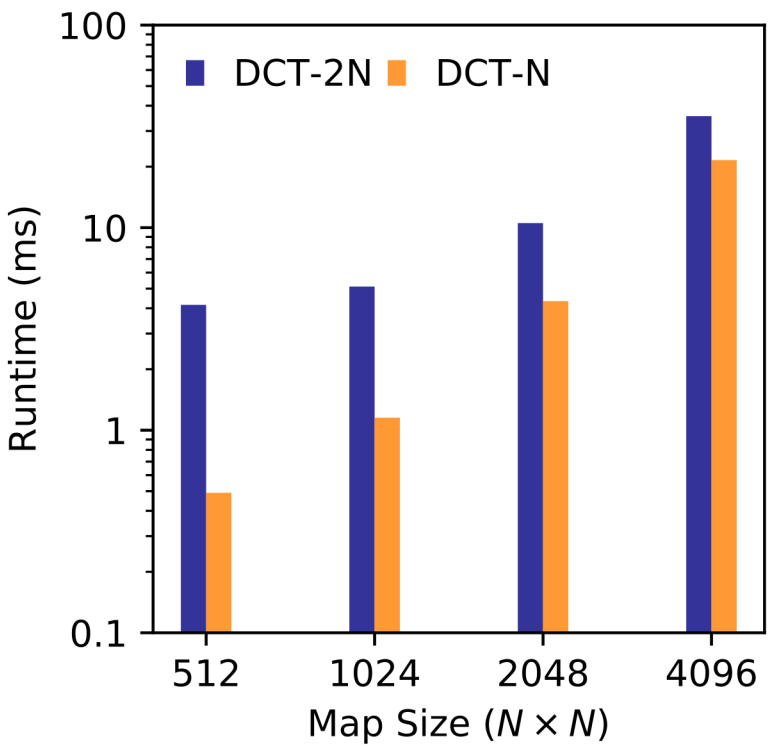
Tensorflow DCT



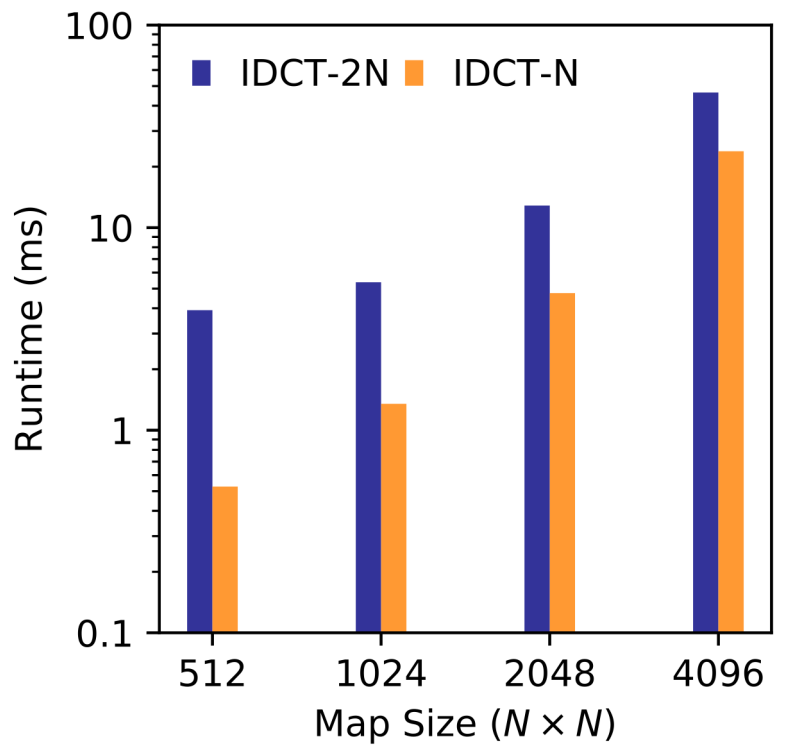
ACCELERATING DCT/IDCT

Leverage highly optimized FFT kernels in deep learning toolkit

DCT-N achieves **1.4**× speedup



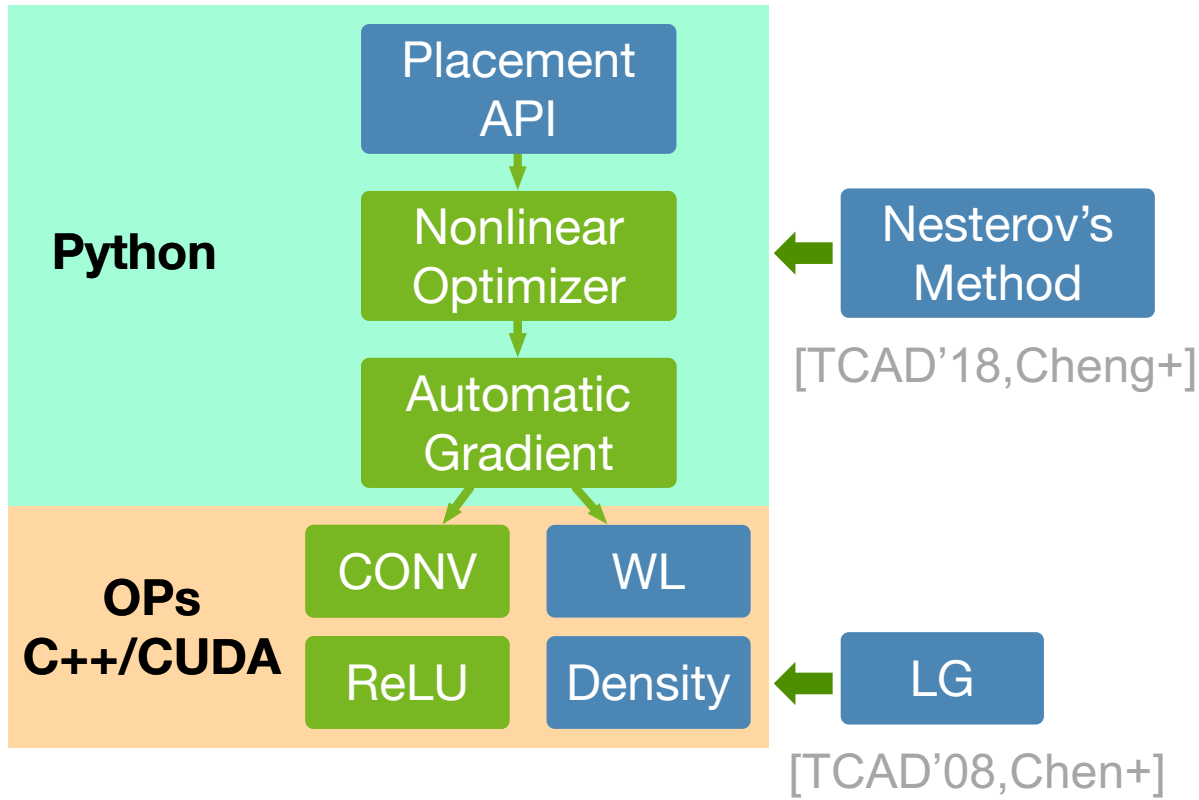
IDCT-N achieves **1.4**× speedup



Typical map sizes in placement

EXPERIMENTAL RESULTS

DREAMPlace framework



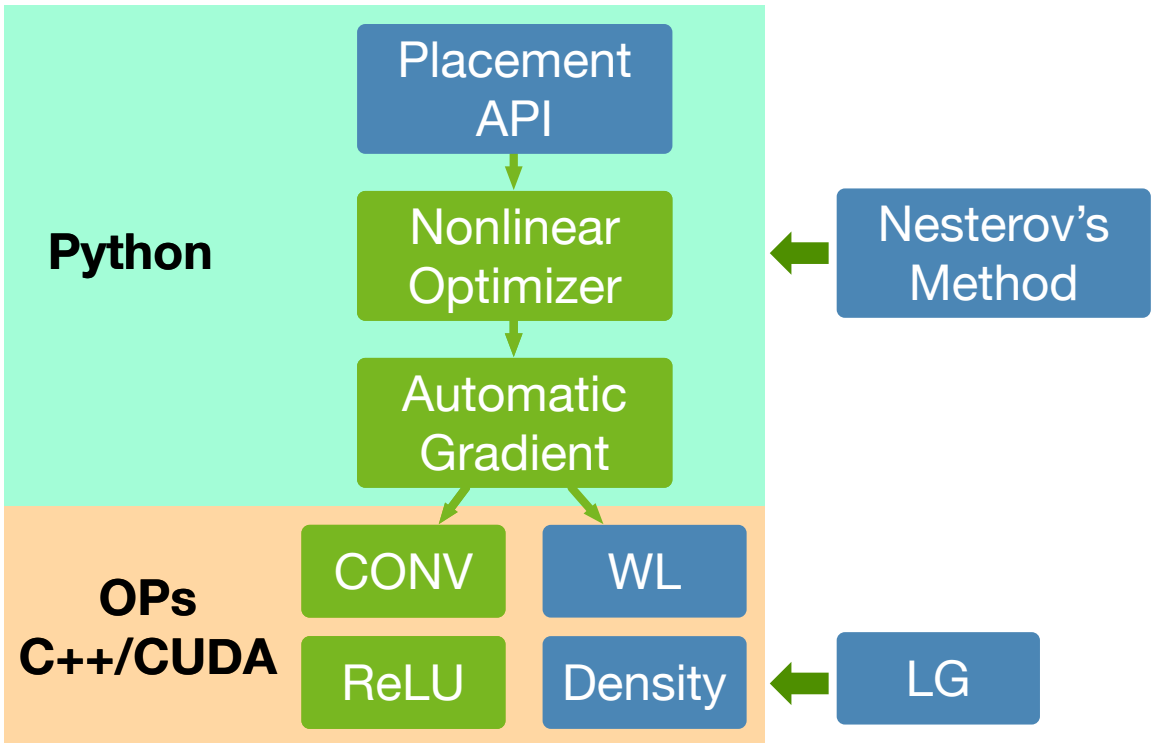
DP still relies on NTUPlace3

EXPERIMENTAL RESULTS

DREAMPlace framework

DREAMPlace
- CPU: Intel E5-2698 v4 @2.20GHz
- GPU: 1 NVIDIA Tesla V100
- Single CPU thread was used

RePlAce [TCAD'18,Cheng+]
- CPU: 24-core 3.0 GHz Intel Xeon
- 64GB memory allocated

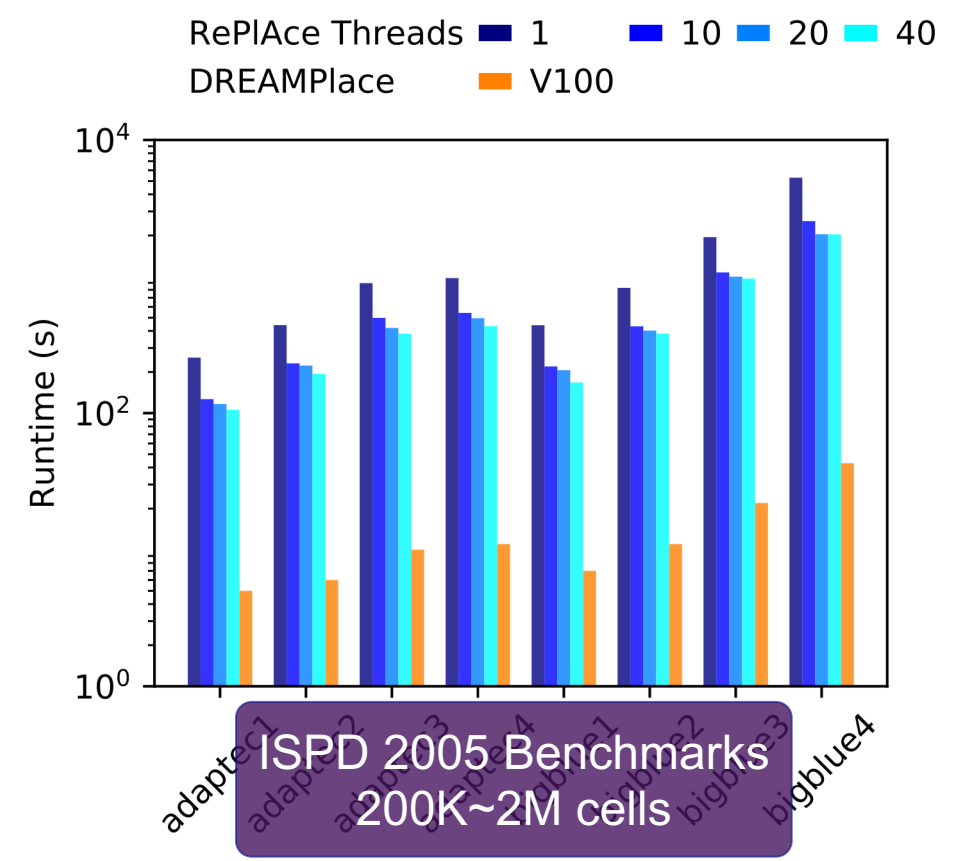


DP still relies on NTUPlace3

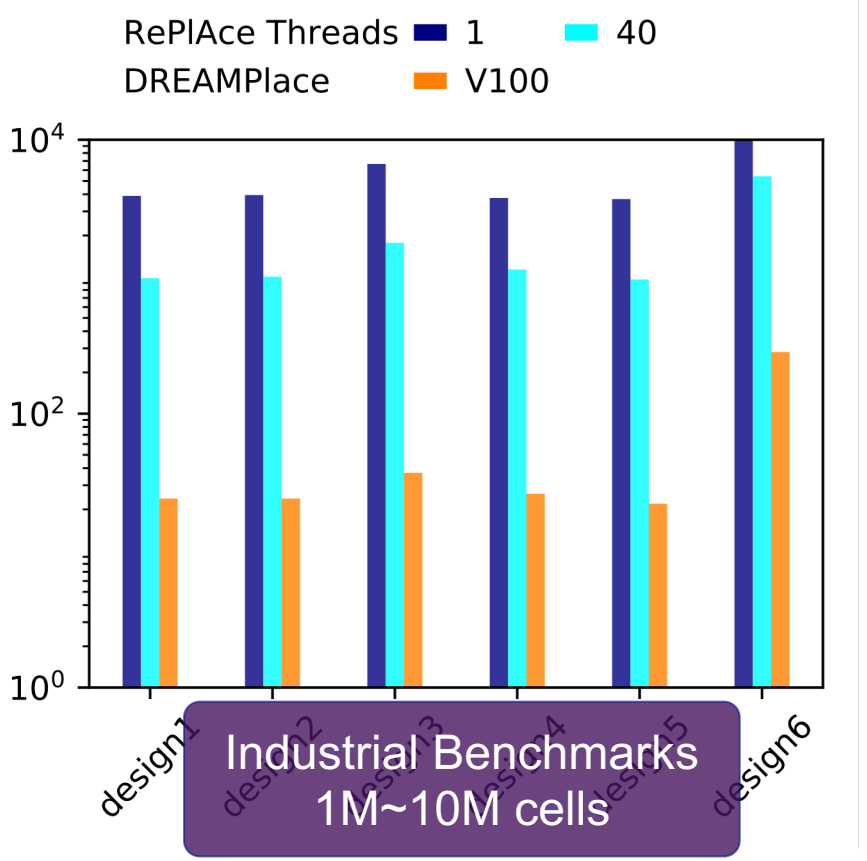
GP RUNTIME COMPARISON

No quality degradation

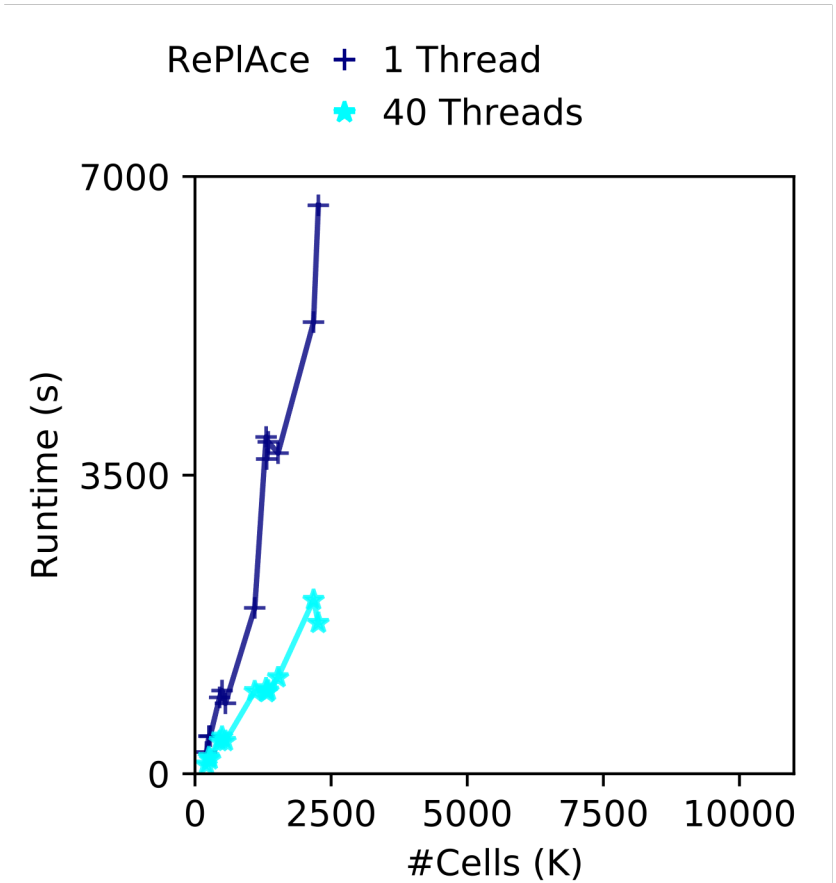
34× speedup by DREAMPlace

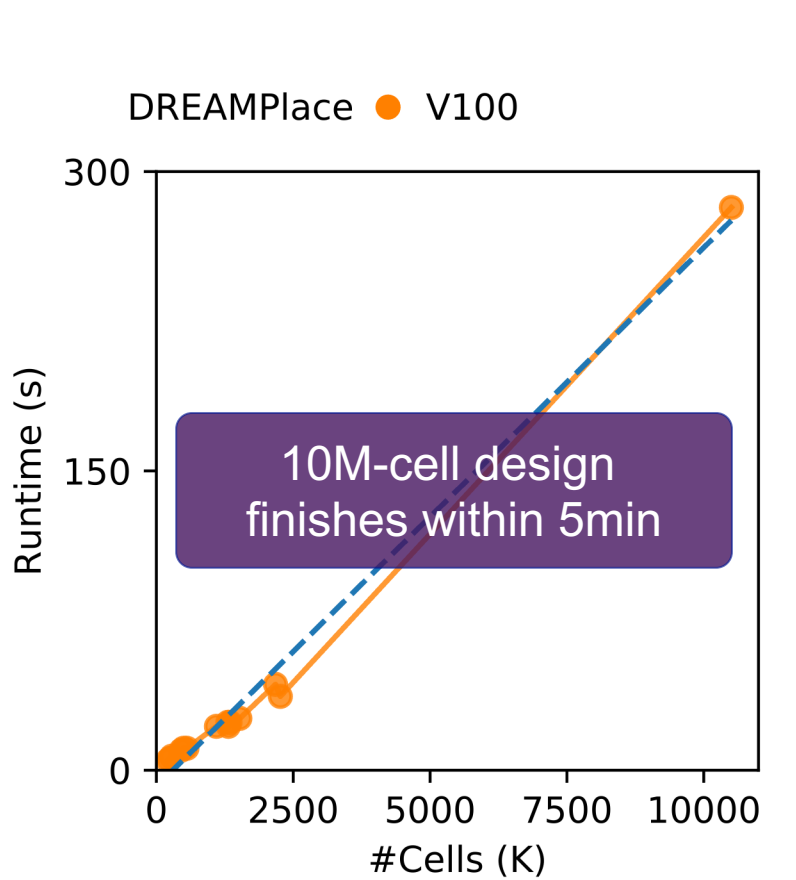


43× speedup by DREAMPlace



GP RUNTIME SCALING

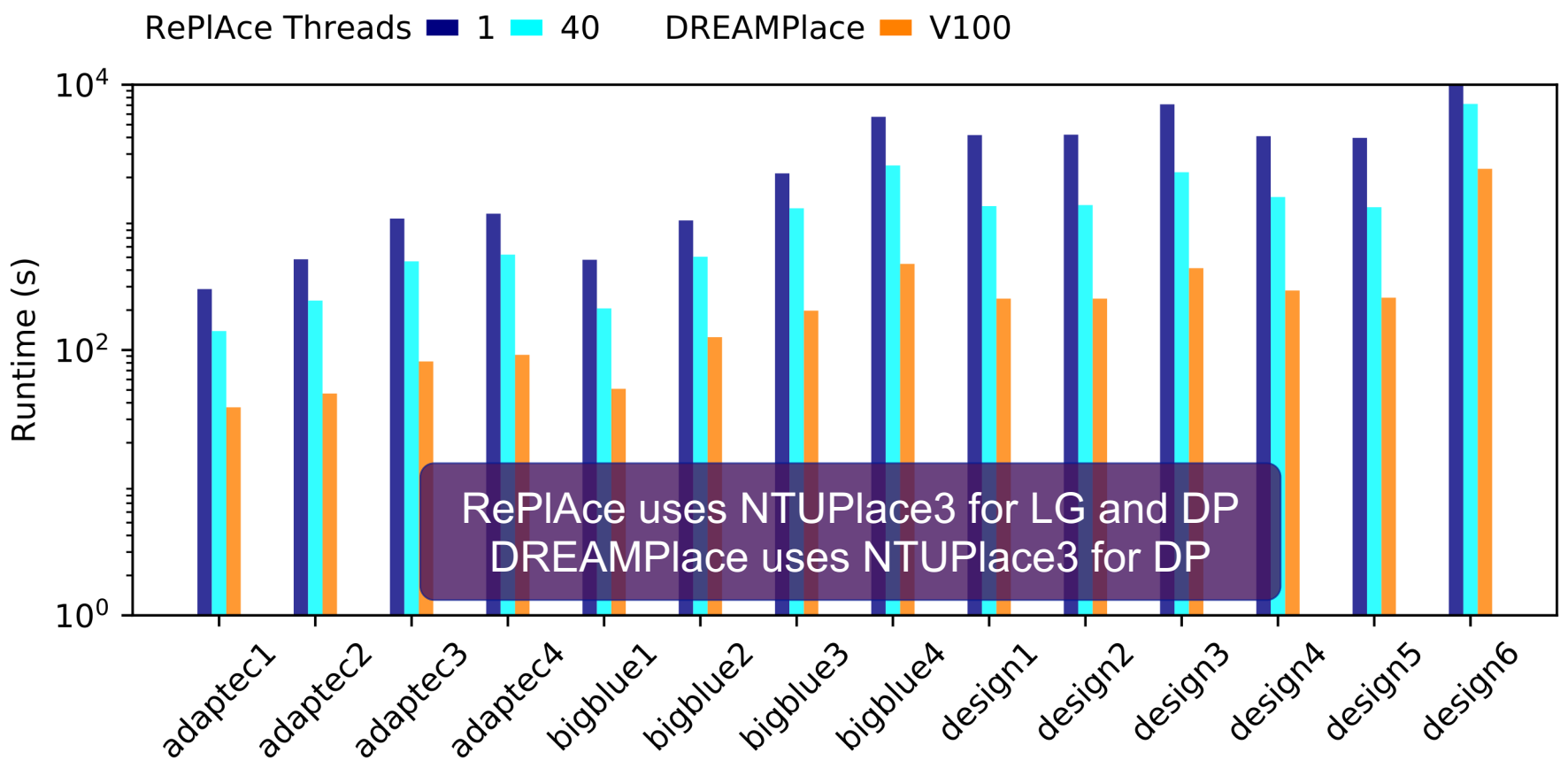




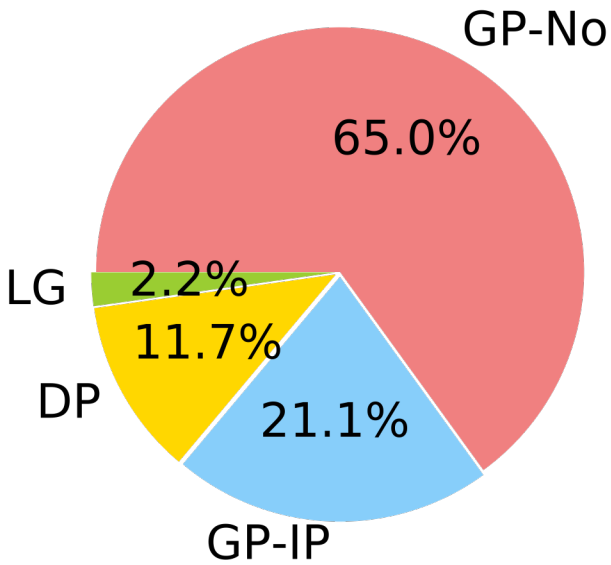
RePlAce crashed on the 10M-cell industry *design6* after using up all the memory allocated

OVERALL RUNTIME COMPARISON

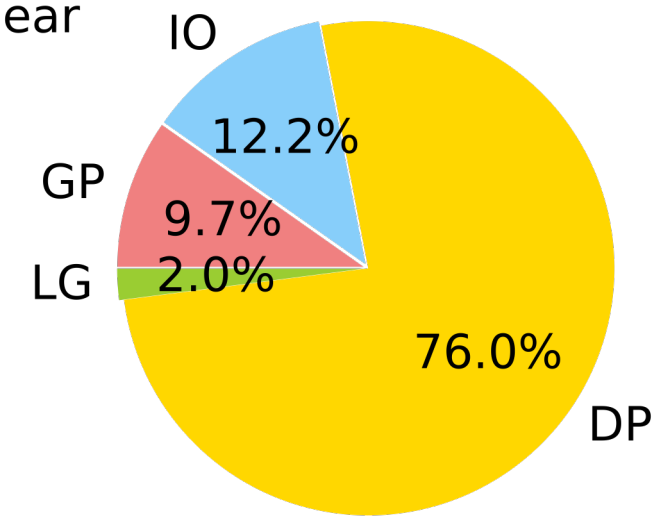
5× speedup in the overall GP+LG+DP flow



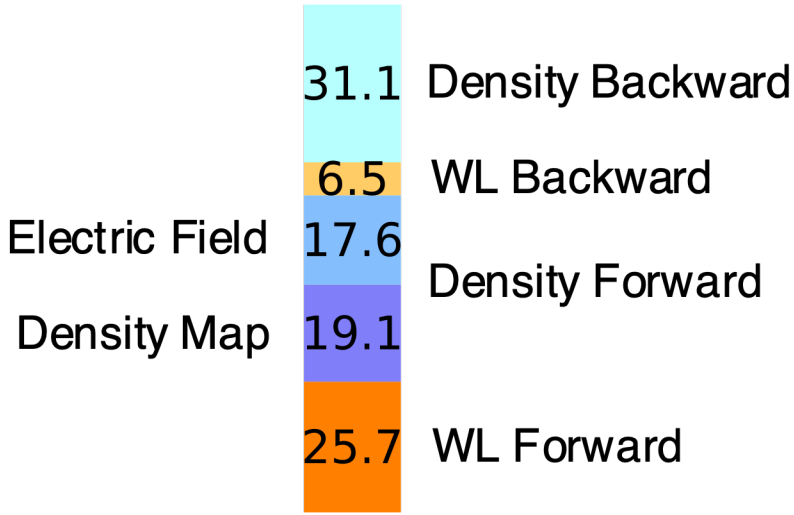
DREAMPLACE RUNTIME BREAKDOWN



RePlAce (10 threads)



DREAMPlace



Runtime breakdown of one forward/backward pass

DREAMPLACE - CONCLUSION

- New angle of viewing a placement problem as neural network training
  - Potentially applicable to other CAD problems, e.g., gate sizing, floorplanning
- GPU-accelerated analytical placer with deep learning toolkit *PyTorch*
- Efficient GPU implementations of key kernels, e.g., WL, density
- Over 30× speedup in global placement and legalization without quality degradation over multi-threaded CPU
  - 10M-cell design finishes in 5min on GPU implementation
  - While it takes >3h on CPU implementation

DREAMPLACE – FUTURE PLAN

- GPU-accelerated detailed placement
- Routability and timing considering in global placement
- Improve gradient descent strategies
  - Explore momentum approaches
  - E.g., ADAM, momentum SGD, provided by deep learning toolkit

Code release: https://github.com/limbo018/DREAMPlace

PART 2

RouteNet: Routability Prediction for Mixed-Size Designs Using Convolutional Neural Network

Zhiyao Xie, Yiran Chen

Duke UNIVERSITY

Yu-Hung Huang, Guan-Qi Fang, Shao-Yun Fang

TAIWAN TECH
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Mark Ren, Brucek Khailany

NVIDIA®

Jiang Hu

TEXAS A&M UNIVERSITY

ROUTENET

- Routability Prediction for Mixed-Size Designs
- After detailed routing: Design Rule Checking (DRC)
  - Minimum metal space & width
  - …
- Less DRC violation -> Higher routability

OUTLINE

- Background
  - Challenge in routability prediction & previous solutions
  - Influence of macros
  - Convolutional Neural Network
- RouteNet
  - Feature extraction
  - Proposed model
- Results
- Conclusion

OUTLINE

- **Background**
  - **Challenge in routability prediction & previous solutions**
  - Influence of macros
  - Convolutional Neural Network
- RouteNet
  - Feature extraction
  - Proposed model
- Results
- Conclusion

BACKGROUND

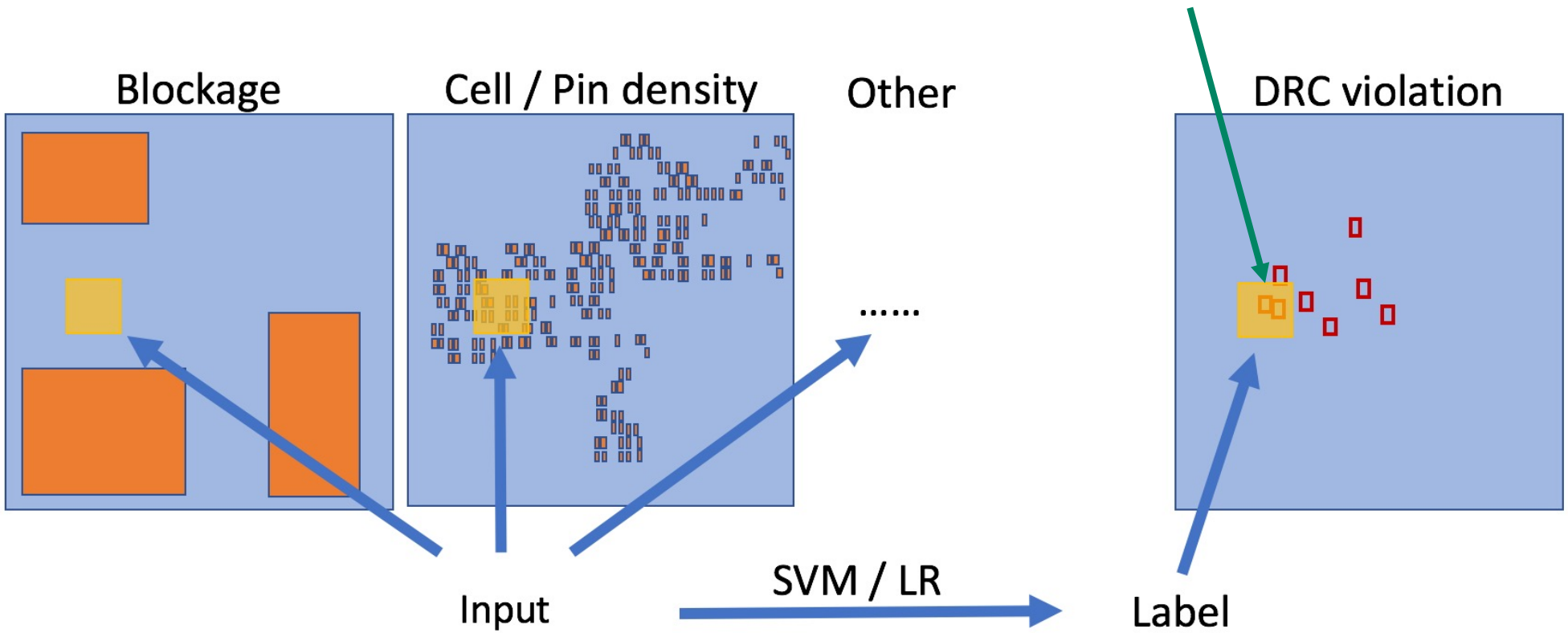
- Challenges in Routability Prediction:
  - Predict routability at placement stage
    - Predict by ‘fast trial global routing’
    - Not fast enough
  - Predict locations of Design Rule Checking (DRC) hotspots
    - Predict by global routing
    - Not accurate enough

BACKGROUND

- Our Attempt on Such Challenges:
  - Fast routability forecast for placement
    - In terms of number of Design Rule Violations (#DRV)
    - To identify more routable placements among many candidates
  - Prediction of DRC hotspot locations
    - In terms of DRC hotspot locations
    - To proactively modify solutions to prevent design rule violations

BACKGROUND

- Previous solutions:
  - Many apply machine learning on every ***small*** cropped region



BACKGROUND

- Previous solutions:
  - Many fail to consider macros
  - Some require Global Routing information for #DRV prediction

Methods	Use GR?	Predict #DRV?	Predict hotspot?	Handle macros?
[18] (Qi, et al., ICCD14)	Y	Y	N	Y
[26] (Zhou, et al., ASQED15)	Y	Y	N	N
[3] (Chan, et al., ICCD16)	N	Y	N	N
[4] (Chan, et al., ISPD17)	Y	N	Y	N
Our Method: RouteNet #DRV prediction	N	Y	N	Y
Our Method: RouteNet hotspot prediction	Y	N	Y	Y

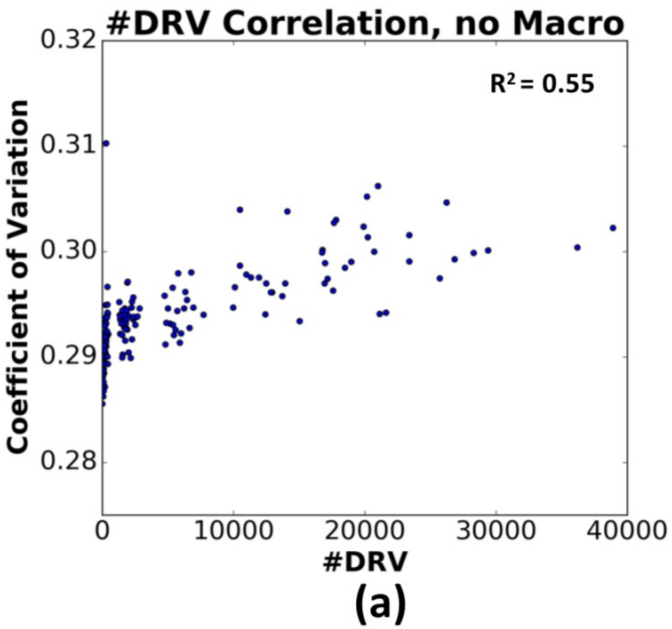
GR means global routing
#DRV means number of DRC violations

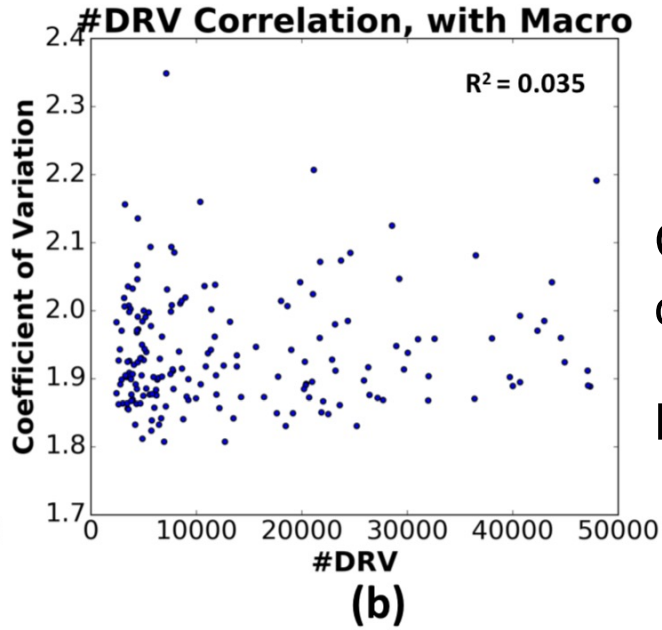
OUTLINE

- **Background**
  - Challenge in routability prediction & previous solutions
  - **Influence of macros**
  - Convolutional Neural Network
- RouteNet
  - Feature extraction
  - Proposed model
- Results
- Conclusion

BACKGROUND

- Challenges of Macros on #DRV prediction:
  - Correlation between pin density and #DRV largely disappears with macro



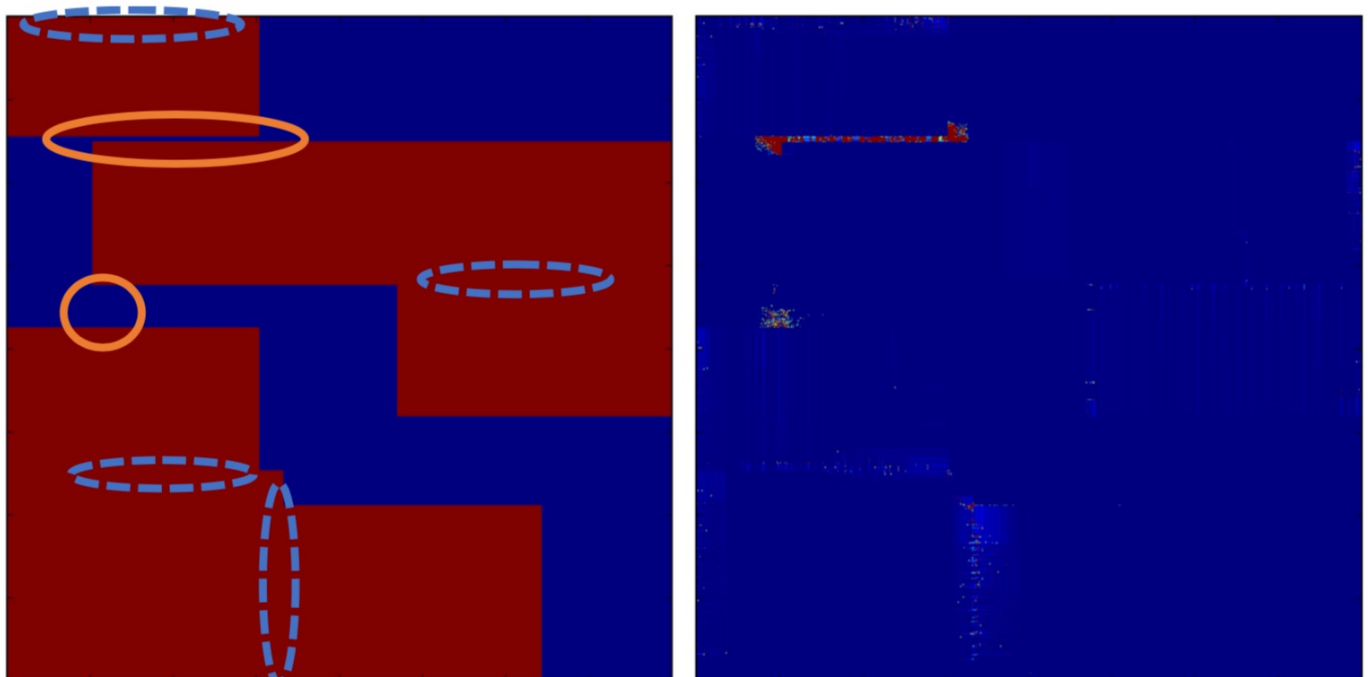


Correlation between #DRV and coefficient of variation ($\frac{\sigma}{\mu}$) of pin density.

Each point corresponds to one placement

BACKGROUND

- Challenges of Macros on hotspot detection:
  - Hotspots tend to aggregate at small gaps between neighboring macros



Macros DRC hotspot

BACKGROUND

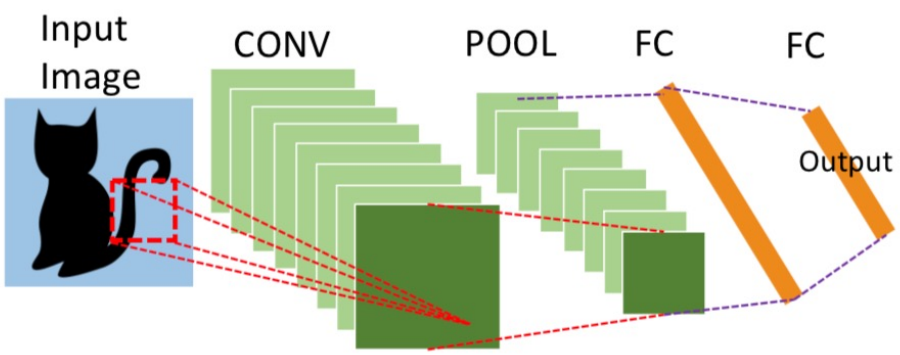
- Challenges of Macros:
  - A layout with macros is much less **homogeneous**
  - **Homogeneity** implies resemblance among different regions of layout
  - Need a larger region to capture global view
  - **Use deep neural network!**

OUTLINE

- **Background**
  - Challenge in routability prediction & previous solutions
  - Influence of macros
  - **Convolutional Neural Network**
- RouteNet
  - Feature extraction
  - Proposed model
- Results
- Conclusion

BACKGROUND

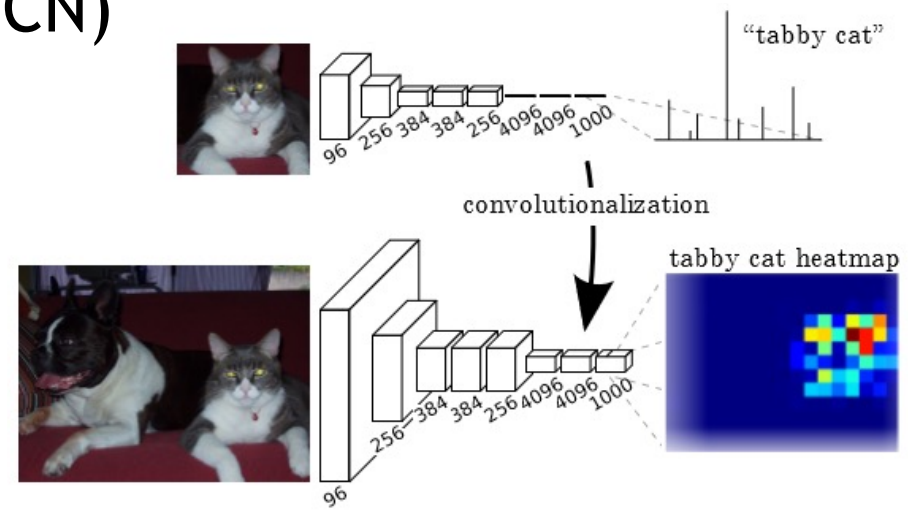
Convolutional Neural Network (CNN)



Convolutional (CONV), Pooling (POOL) and Fully Connected (FC) layers

Widely used in image classification

Fully Convolutional Network (FCN)



Eliminate FC layers

May use transposed-convolutional to up-sample

Used in image segmentation, object detection

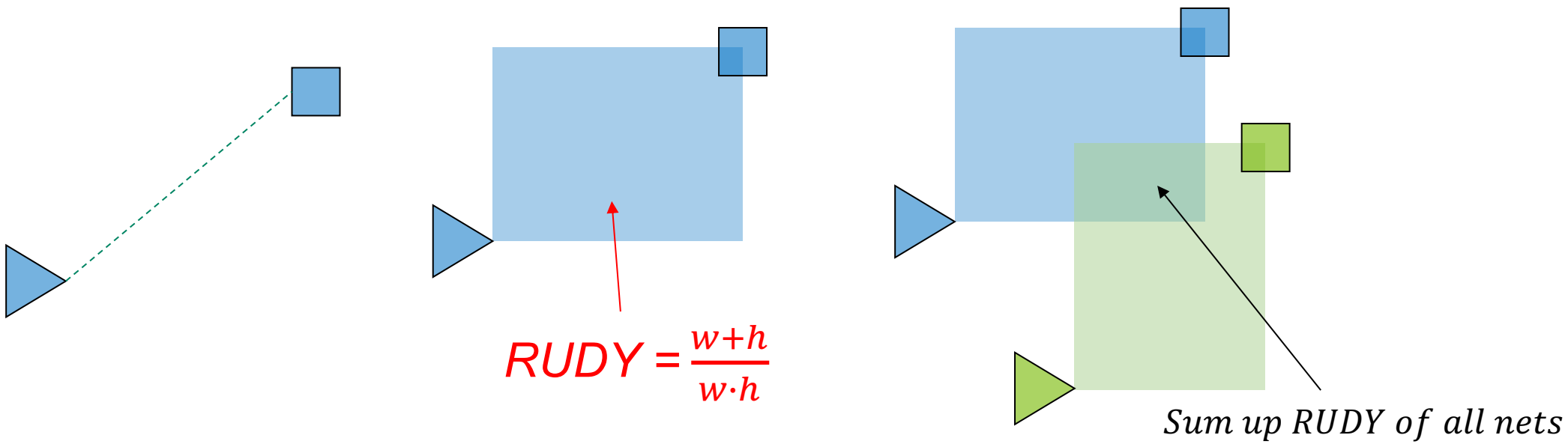
Image on right side from: Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2017. Fully Convolutional Networks for Semantic Segmentation. (TPAMI)

OUTLINE

- Background
  - Challenge in routability prediction & previous solutions
  - Influence of macros
  - Convolutional Neural Network
- **RouteNet**
  - **Feature extraction**
  - Proposed model
- Results
- Conclusion

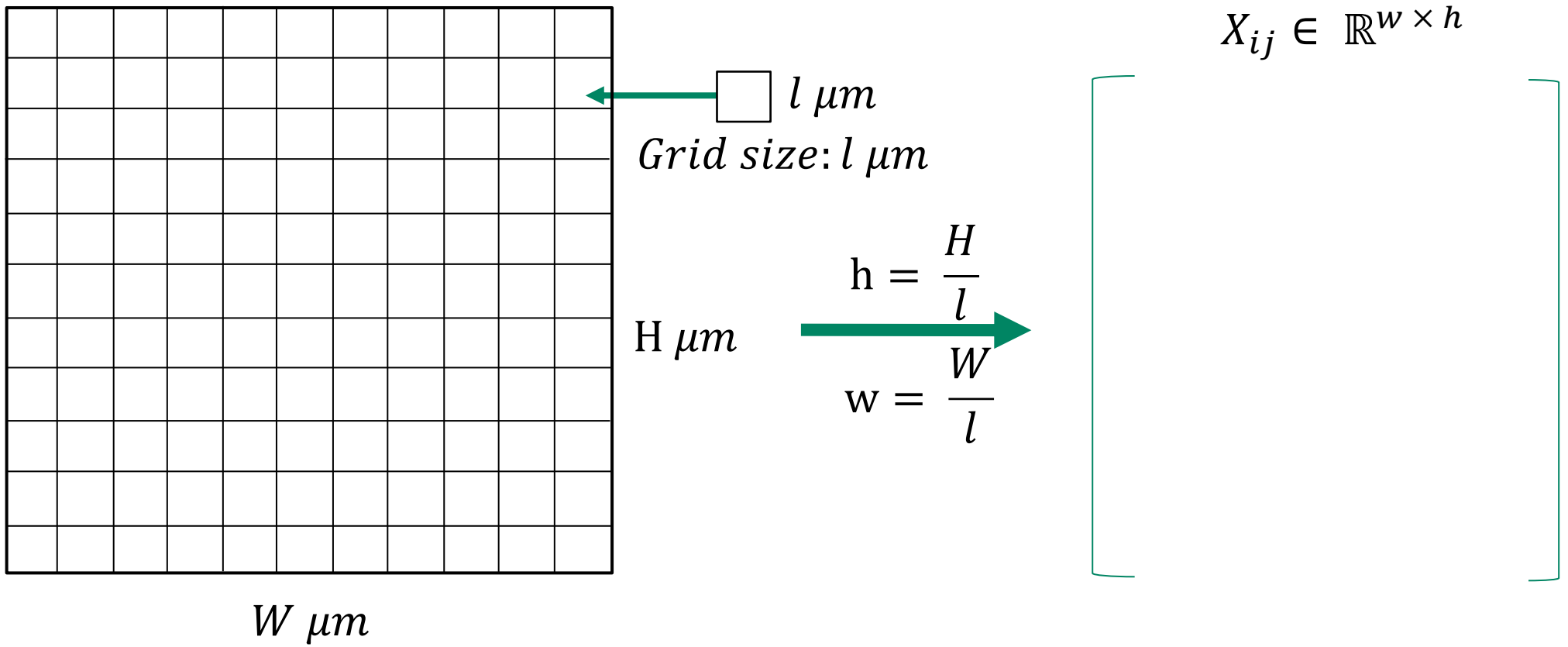
FEATURES EXTRACTION

- RUDY (Rectangular Uniform wire DensitY) (P. Spinder et al. 2007)
  - RUDY is a pre-routing congestion estimator
  - RUDY at a point is superposition of RUDYs of multiple nets



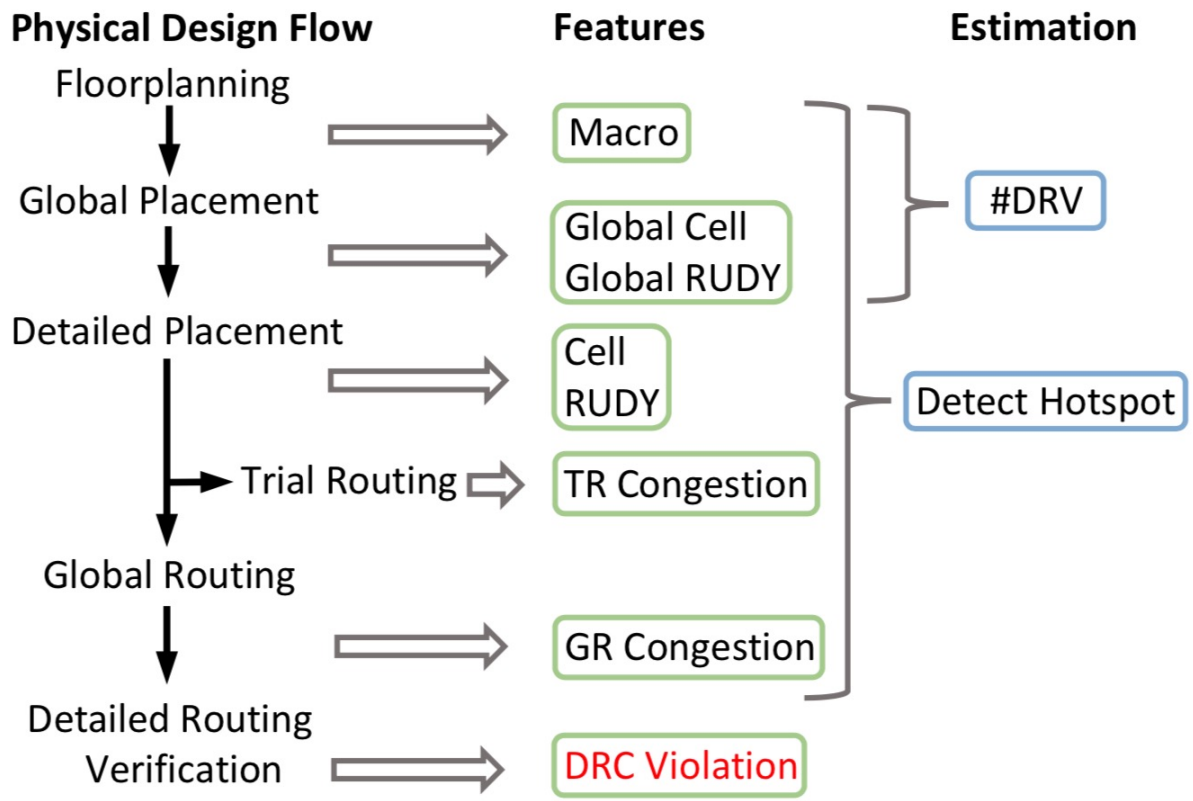
FEATURES EXTRACTION

$X_{ij} = j\ th$ feature in $i\ th$ placement



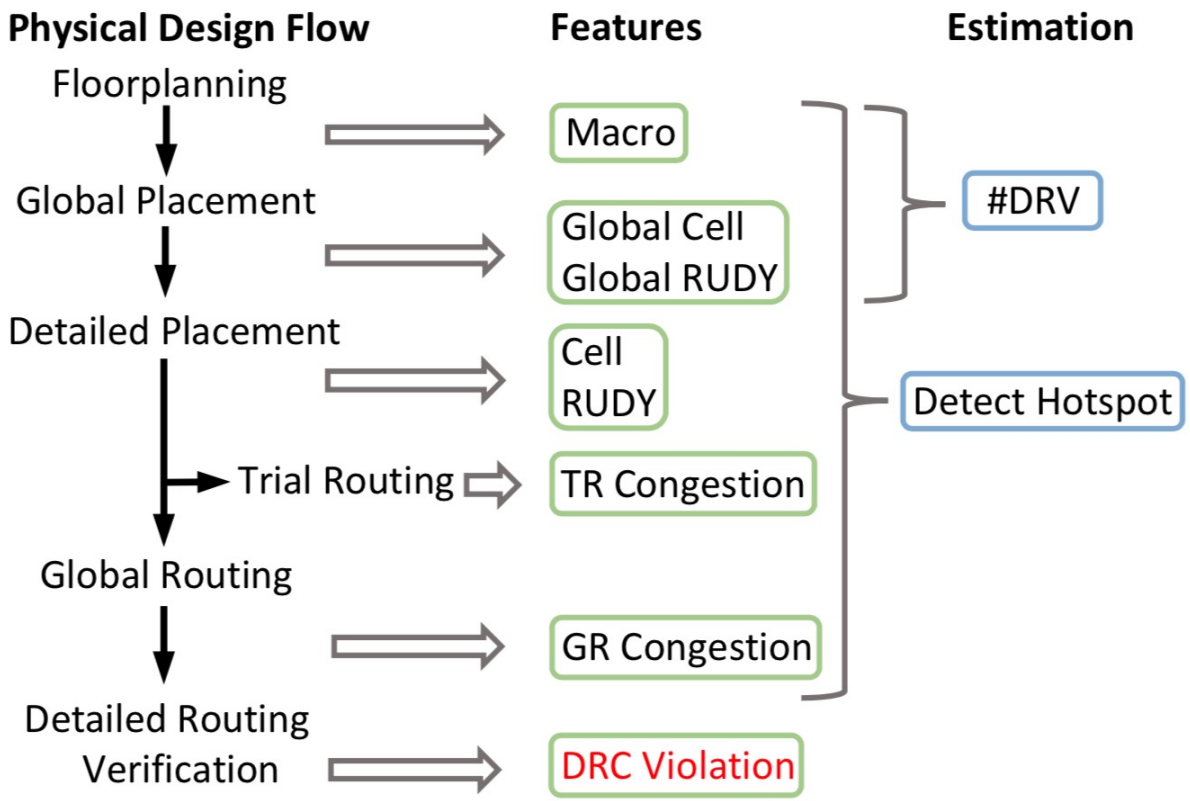
FEATURES EXTRACTION

- Macro:
  - region occupied by macros
  - density of macro pins in each layer
- Cell:
  - density of cells
  - density of cell pins
- Global cell:
  - cell features at global placement
- Global RUDY:
  - RUDY features calculated by global placement results



FEATURES EXTRACTION

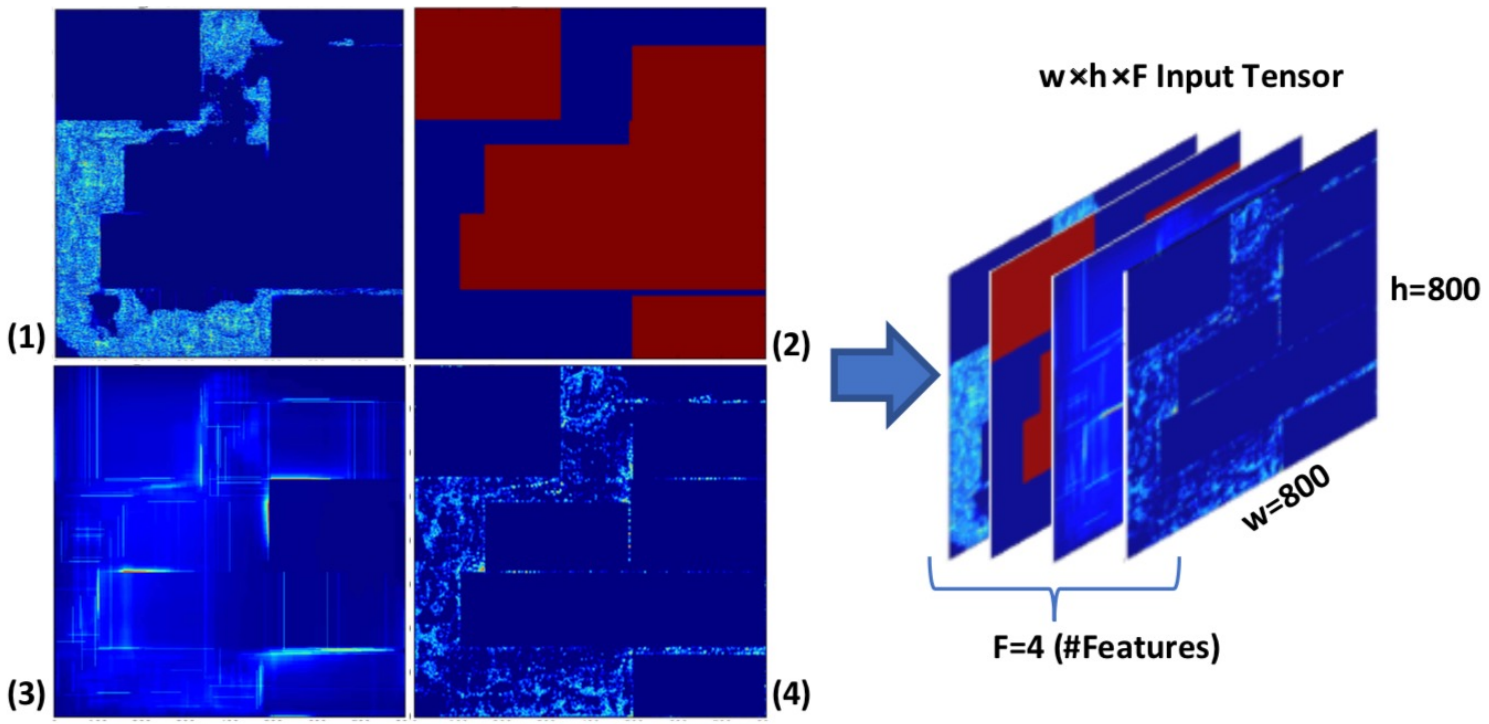
- RUDY
  - long-range RUDY
    - RUDY from long-range nets
  - short-range RUDY
    - DURY from short-range nets
  - RUDY pins
    - pins with density value equal to the RUDY value of its net
- Congestion
  - trial global routing congestion
  - global routing congestion
- DRC violation
  - prediction target / label



For i^{th} placement with size $w \times h$ and F features

$$X_i \in \mathbb{R}^{w \times h \times F}$$

FEATURES EXTRACTION



Input tensor constructed by stacking 2D features:

(1) Pin density, (2) macro (3) long-range RUDY, (4) RUDY pins

OUTLINE

- Background
  - Challenge in routability prediction & previous solutions
  - Influence of macros
  - Convolutional Neural Network
- **RouteNet**
  - Feature extraction
  - **Proposed model**
- Results
- Conclusion

PROPOSED MODEL

Problem 1 (#DRV prediction). *Find an estimator* $f^*_{\#DRV}$ *of DRV count in a placement:*

$$f_{\#DRV} : X_i^{(\#DRV)} \in \mathbb{R}^{w \times h \times F_1} \rightarrow y_i \in \mathbb{N}$$

$$f^*_{\#DRV} = \arg\min_f Loss(f(X_i^{(\#DRV)}),\ y_i)$$

Convolutional Neural Network (CNN)

Problem 2 (Hotspot prediction). *Find a detector* $f^*_{hotspot}$ *of hotspots. It reports locations of all DRC hotspots in a placement.*

$$f_{hotspot} : X_i^{(hotspot)} \in \mathbb{R}^{w \times h \times F_2} \rightarrow V_i \in \{0,1\}^{w \times h}$$

$$f^*_{hotspot} = \arg\min_f Loss(f(X_i^{(hotspot)}),\ V_i)$$

$$Y_i \in \mathbb{R}^{w \times h} \qquad V_{i_{mn}} = \mathbb{1}(Y_{i_{mn}} > \epsilon)$$

Fully Convolutional Network (FCN)

PROPOSED MODEL - #DRV PREDICTION

Algorithm 1 Algorithm of RouteNet for #DRV Prediction

Input: Number of training placements: N, Features: $\{X_i \in \mathbb{R}^{w \times h \times 3} \mid i \in [1, N]\}$, Targets: $\{y_i \in \mathbb{R} \mid i \in [1, N]\}$

Preprocess:

1: **for** each int $i \in [1, N]$ **do**
2: Resize $X_i \in \mathbb{R}^{w \times h \times 3}$ into $X_i^{\#DRV} \in \mathbb{R}^{224 \times 224 \times 3}$
3: Find 25%, 50%, 75% quantizes of y_i: q_1, q_2, q_3
4: **for** each int $i \in [1, N]$ **do**
5: $C_i \leftarrow 0$
6: **for** each int $t \in [1, 3]$ **do**
7: **if** $y_i > q_t$ **then**
8: $C_i \leftarrow t$, **break**
9: Form dataset $\{(X_i^{\#DRV}, C_i) | i \in [1, N]\}$
10: Training set $\{(X_i^{\#DRV}, C_i) \mid C_i = 0 \text{ or } C_i = 3\}$

Training:

1: Get pretrained ResNet18 $f_{Res} : \mathbb{R}^{224 \times 224 \times 3} \rightarrow \mathbb{R}^{1000}$
2: Replace output layer, s.t. $f_{\#DRV} : \mathbb{R}^{224 \times 224 \times 3} \rightarrow \mathbb{R}$
3: Choose MSE as loss function, SGD for optimization
4: Train $f_{\#DRV}$ with preprocessed dataset for ~30 epoches

Output: $f_{\#DRV}$ estimating #DRV level

Resize input to 224*224, to utilize models pre-trained on images with size 224*224

PROPOSED MODEL - #DRV PREDICTION

Algorithm 1 Algorithm of RouteNet for #DRV Prediction

Input: Number of training placements: N, Features: $\{X_i \in \mathbb{R}^{w \times h \times 3} \mid i \in [1, N]\}$, Targets: $\{y_i \in \mathbb{R} \mid i \in [1, N]\}$

Preprocess:

1: **for** each int $i \in [1, N]$ **do**
2: Resize $X_i \in \mathbb{R}^{w \times h \times 3}$ into $X_i^{\#DRV} \in \mathbb{R}^{224 \times 224 \times 3}$
3: Find 25%, 50%, 75% quantizes of y_i: q_1, q_2, q_3
4: **for** each int $i \in [1, N]$ **do**
5: $C_i \leftarrow 0$
6: **for** each int $t \in [1, 3]$ **do**
7: **if** $y_i > q_t$ **then**
8: $C_i \leftarrow t$, **break**
9: Form dataset $\{(X_i^{\#DRV}, C_i) | i \in [1, N]\}$
10: Training set $\{(X_i^{\#DRV}, C_i) \mid C_i = 0 \text{ or } C_i = 3\}$

Training:

1: Get pretrained ResNet18 $f_{Res} : \mathbb{R}^{224 \times 224 \times 3} \rightarrow \mathbb{R}^{1000}$
2: Replace output layer, s.t. $f_{\#DRV} : \mathbb{R}^{224 \times 224 \times 3} \rightarrow \mathbb{R}$
3: Choose MSE as loss function, SGD for optimization
4: Train $f_{\#DRV}$ with preprocessed dataset for ~30 epoches

Output: $f_{\#DRV}$ estimating #DRV level

Assign placements to 4 different classes (c_0, c_1, c_2, c_3) based on their level of violations (#DRV)

c_0 represents least #DRV, while c_3 represents most

PROPOSED MODEL - #DRV PREDICTION

Algorithm 1 Algorithm of RouteNet for #DRV Prediction

Input: Number of training placements: N, Features: $\{X_i \in \mathbb{R}^{w \times h \times 3} \mid i \in [1, N]\}$, Targets: $\{y_i \in \mathbb{R} \mid i \in [1, N]\}$

Preprocess:

for each int $i \in [1, N]$ **do**
    Resize $X_i \in \mathbb{R}^{w \times h \times 3}$ into $X_i^{\#DRV} \in \mathbb{R}^{224 \times 224 \times 3}$
Find 25%, 50%, 75% quantizes of y_i: q_1, q_2, q_3
for each int $i \in [1, N]$ **do**
    $C_i \leftarrow 0$
    for each int $t \in [1, 3]$ **do**
        if $y_i > q_t$ **then**
            $C_i \leftarrow t$, **break**
Form dataset $\{(X_i^{\#DRV}, C_i) | i \in [1, N]\}$
Training set $\{(X_i^{\#DRV}, C_i) \mid C_i = 0 \text{ or } C_i = 3\}$

Training:

Get pretrained ResNet18 $f_{Res} : \mathbb{R}^{224 \times 224 \times 3} \rightarrow \mathbb{R}^{1000}$
Replace output layer, s.t. $f_{\#DRV} : \mathbb{R}^{224 \times 224 \times 3} \rightarrow \mathbb{R}$
Choose MSE as loss function, SGD for optimization
Train $f_{\#DRV}$ with preprocessed dataset for ~30 epoches

Output: $f_{\#DRV}$ estimating #DRV level

Download a pre-trained CNN model named ResNet18

PROPOSED MODEL - #DRV PREDICTION

Algorithm 1 Algorithm of RouteNet for #DRV Prediction

Input: Number of training placements: N, Features: $\{X_i \in \mathbb{R}^{w \times h \times 3} \mid i \in [1, N]\}$, Targets: $\{y_i \in \mathbb{R} \mid i \in [1, N]\}$

Preprocess:

for each int $i \in [1, N]$ **do**
    Resize $X_i \in \mathbb{R}^{w \times h \times 3}$ into $X_i^{\#DRV} \in \mathbb{R}^{224 \times 224 \times 3}$
Find 25%, 50%, 75% quantizes of y_i: q_1, q_2, q_3
for each int $i \in [1, N]$ **do**
    $C_i \leftarrow 0$
    for each int $t \in [1, 3]$ **do**
        if $y_i > q_t$ **then**
            $C_i \leftarrow t$, **break**
Form dataset $\{(X_i^{\#DRV}, C_i) | i \in [1, N]\}$
Training set $\{(X_i^{\#DRV}, C_i) \mid C_i = 0 \text{ or } C_i = 3\}$

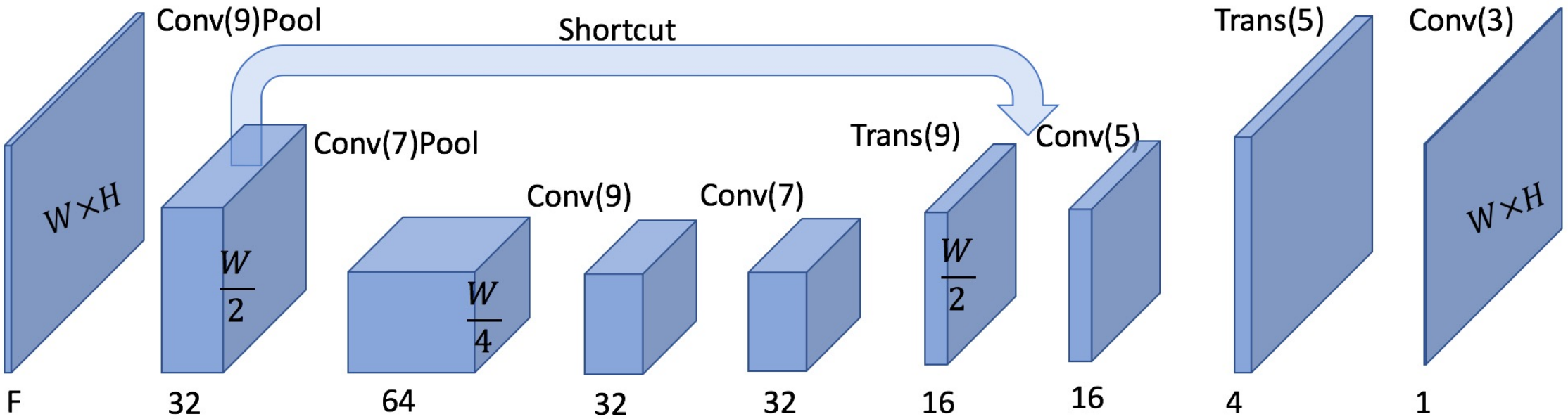
Training:

Get pretrained ResNet18 $f_{Res} : \mathbb{R}^{224 \times 224 \times 3} \rightarrow \mathbb{R}^{1000}$
Replace output layer, s.t. $f_{\#DRV} : \mathbb{R}^{224 \times 224 \times 3} \rightarrow \mathbb{R}$
Choose MSE as loss function, SGD for optimization
Train $f_{\#DRV}$ with preprocessed dataset for ~30 epoches

Output: $f_{\#DRV}$ estimating #DRV level

Fine-tune CNN with preprocessed data

PROPOSED MODEL - HOTSPOT DETECTION



$$Y_{i_{mn}}^{clip} = min(Y_{i_{mn}}, c)$$

$$Loss = \sum_{i=1}^{N} \sum_{m=1}^{w} \sum_{n=1}^{h} ||f_{hotspot}(X_{i_{mn}}) - Y_{i_{mn}}^{clip}||_2 + \lambda||W||_2$$

← Pixel-wise loss function

OUTLINE

- Background
  - Challenge in routability prediction & previous solutions
  - Influence of macros
  - Convolutional Neural Network
- RouteNet
  - Feature extraction
  - Proposed model
- **Results**
- Conclusion

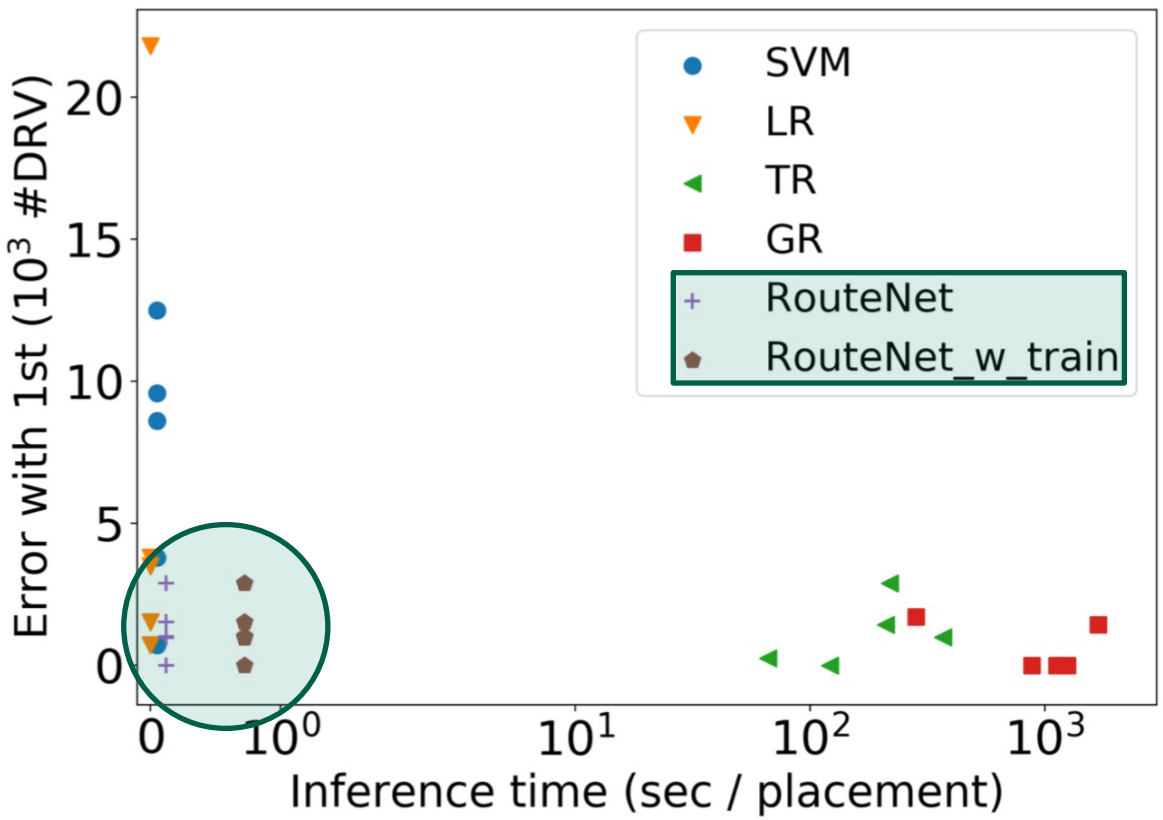
DATA

- Five designs from ISPD 2015
- ~300 different placements by placing macros in different way
- When each design tested, model trained only on four other designs
- SVM and Logistic Regression (LR) methods for comparison

Circuit Name	#Macros	#Cells	#Nets	Width (μm)	#Placements
des_perf	4	108288	110283	900	600
edit_dist	6	127413	131134	800	300
fft	6	30625	32088	800	300
matrix_mult_a	5	149650	154284	1500	300
matrix_mult_b	7	146435	151614	1500	300

#DRV PREDICTION EVALUATION

- Y: gap between the ‘best in 10’ and the actually 1st-ranked placement with least #DRV
- X: inference time taken for each method
- RouteNet achieves low inference time and high accuracy at the same time



DRC HOTSPOT DETECTION EVALUATION

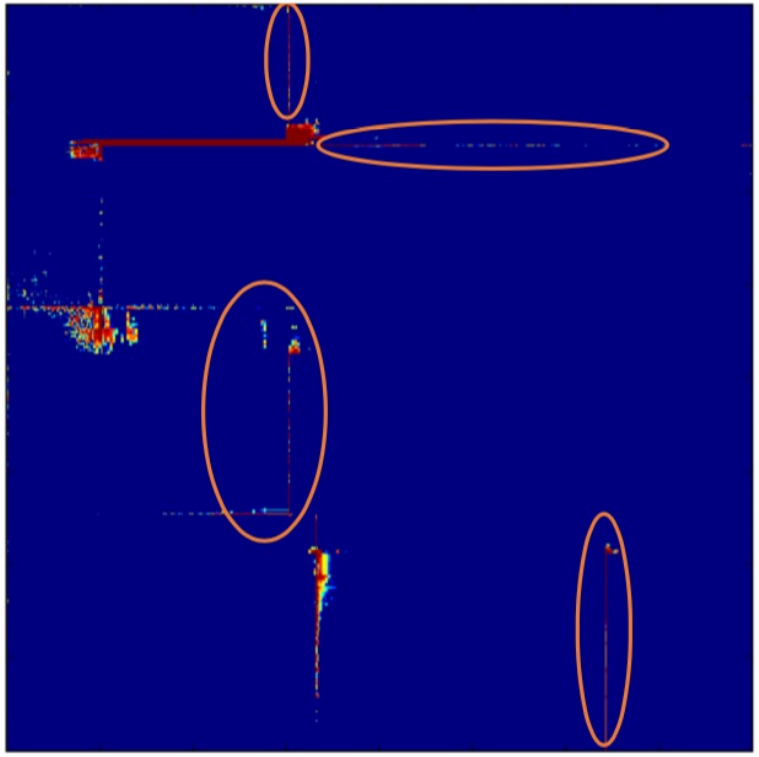
- Same decision threshold is used for all designs
- Slight different FPR, but all under 1%
- RouteNet is superior to all methods and improves global routing accuracy by 50%

Circuit Name	FPR (%)	TPR (%)				
		TR	GR	LR	SVM	RouteNet
des_perf	0.54	17	56	54	42	74
edit_dist	1.00	25	36	38	28	64
fft	0.30	21	45	54	31	71
matrix_mult_a	0.21	13	30	34	12	49
matrix_mult_b	0.24	13	37	41	20	53
Average	0.46	18	41	44	27	62

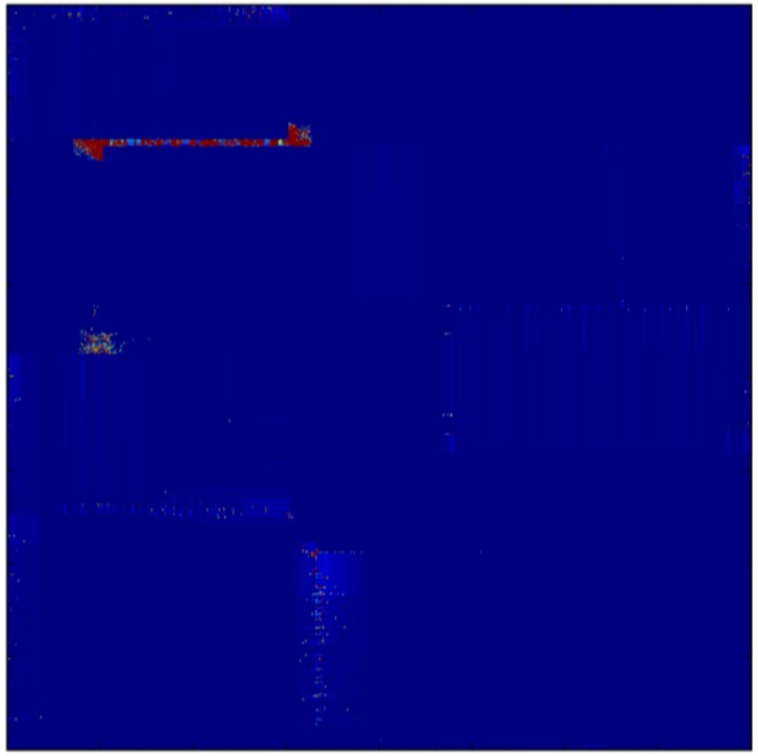
Label \ Prediction Result	Positive	Negative	Evaluation
Positive	TP	FN	$TPR = \frac{TP}{TP + FN}$
Negative	FP	TN	$FPR = \frac{FP}{FP + TN}$

TPR (True Positive Rate)
FPR (False Positive Rate)

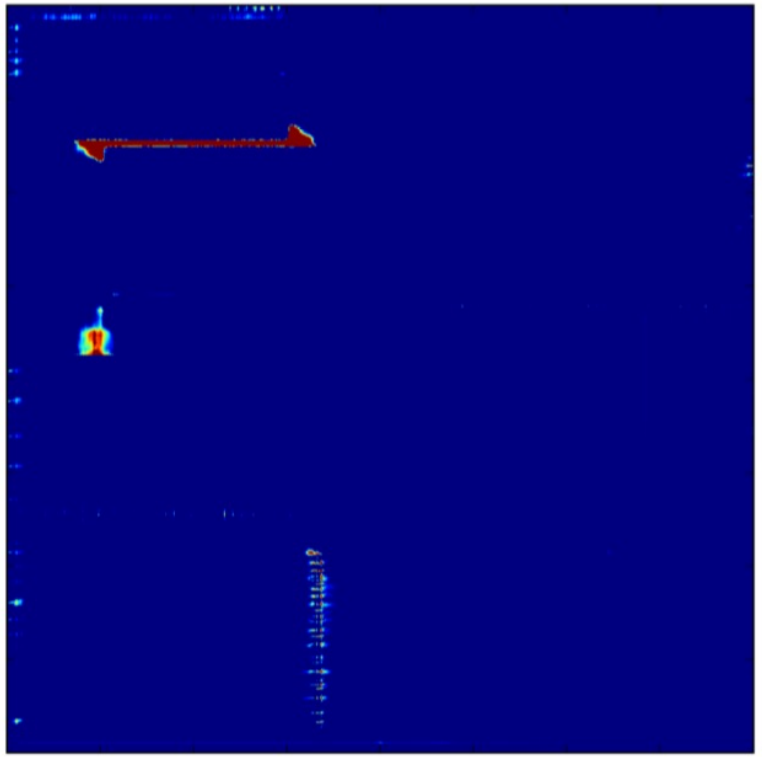
DRC HOTSPOT DETECTION EVALUATION



LR



Ground Truth



RouteNet

DRC HOTSPOT DETECTION EVALUATION

- Variations of FCN
  - No short: Shortcut structure is removed
  - Less conv: Three convolutional layers are removed
  - No pool: Pooling layers are removed

Circuit Name	FPR (%)	TPR (%)					
		Infer seen	Less data	No short	Less conv	No pool	Route Net
des_perf	0.54	77	71	71	73	68	74
edit_dist	1.00	68	61	63	62	55	64
fft	0.30	74	70	68	68	69	71
matrix_mult_a	0.21	51	46	45	45	45	49
matrix_mult_b	0.24	58	50	51	50	50	53
Average	0.46	66	60	60	60	57	62

Importance of large receptive region and global information

SUMMARY OF ROUTENET

- We propose RouteNet:
  - Enables a global view for less homogeneous layout
  - Faster overall routability forecast at placement
  - More accurate hotspot detection at global routing

BACKUP