

# Parallel Multi-level Analytical Global Placement on Graphics Processing Units

Jason Cong and Yi Zou

Computer Science Department

University of California, Los Angeles

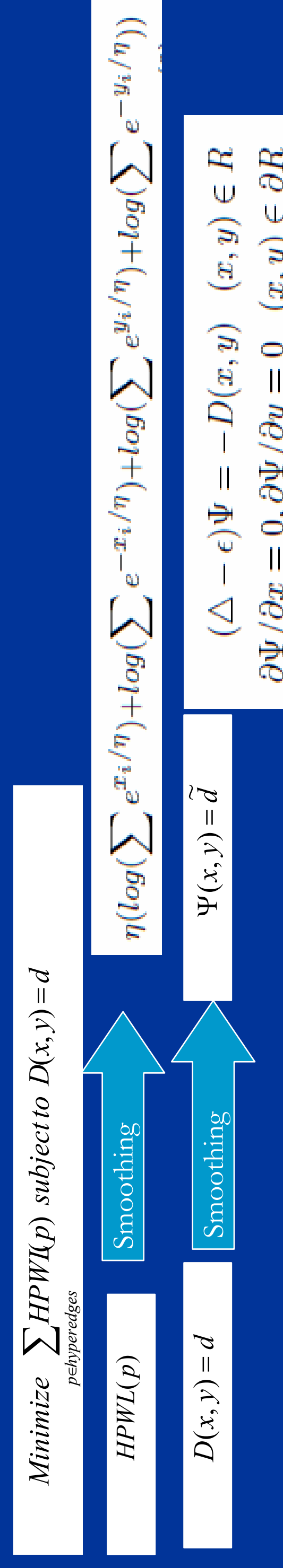
## Abstract

GPU platforms are becoming increasingly attractive for implementing accelerators because they feature a larger number of cores with improved programmability. In this paper, we describe our implementation of a state-of-the-art academic multi-level analytical placer mPL on Nvidia's massively parallel GT200 series platforms. We detail our efforts on performance tuning and optimizations. When compared to software implementation on Intel's recent generation Xeon CPU, the speed of the global placement part of mPL is 15X faster on average using a Tesla C1060 card, with comparable WL. (less than 1% WL degradation on average)

## Introduction

### Multi-level Analytical Global Placement

Circuit placement determines the location of cells and macros of physical chip layout. mPL is one award-winning academic placer developed by UCLA. It serves as the software reference in this work.



The smoothed formulation is solved through Augmented Lagrangian and Gradient descent.

Four key functions/components:

- 1) Native force computation (the gradient of smoothed HPWL)
- 2) Bin density computation
- 3) Density smoothing
- 4) Spreading force computation (the gradient of penalty terms)

## Implementation

### Native force computation

The x-component of native force by one hyperedge:

$$e^{x_i/\eta} / (\sum e^{x_i/\eta}) - e^{-x_i/\eta} / (\sum e^{-x_i/\eta})$$

### Paralleling outerloop or innerloop?

Alleviate load balancing issues through sorting of loop bounds

Inefficiency of small reductions

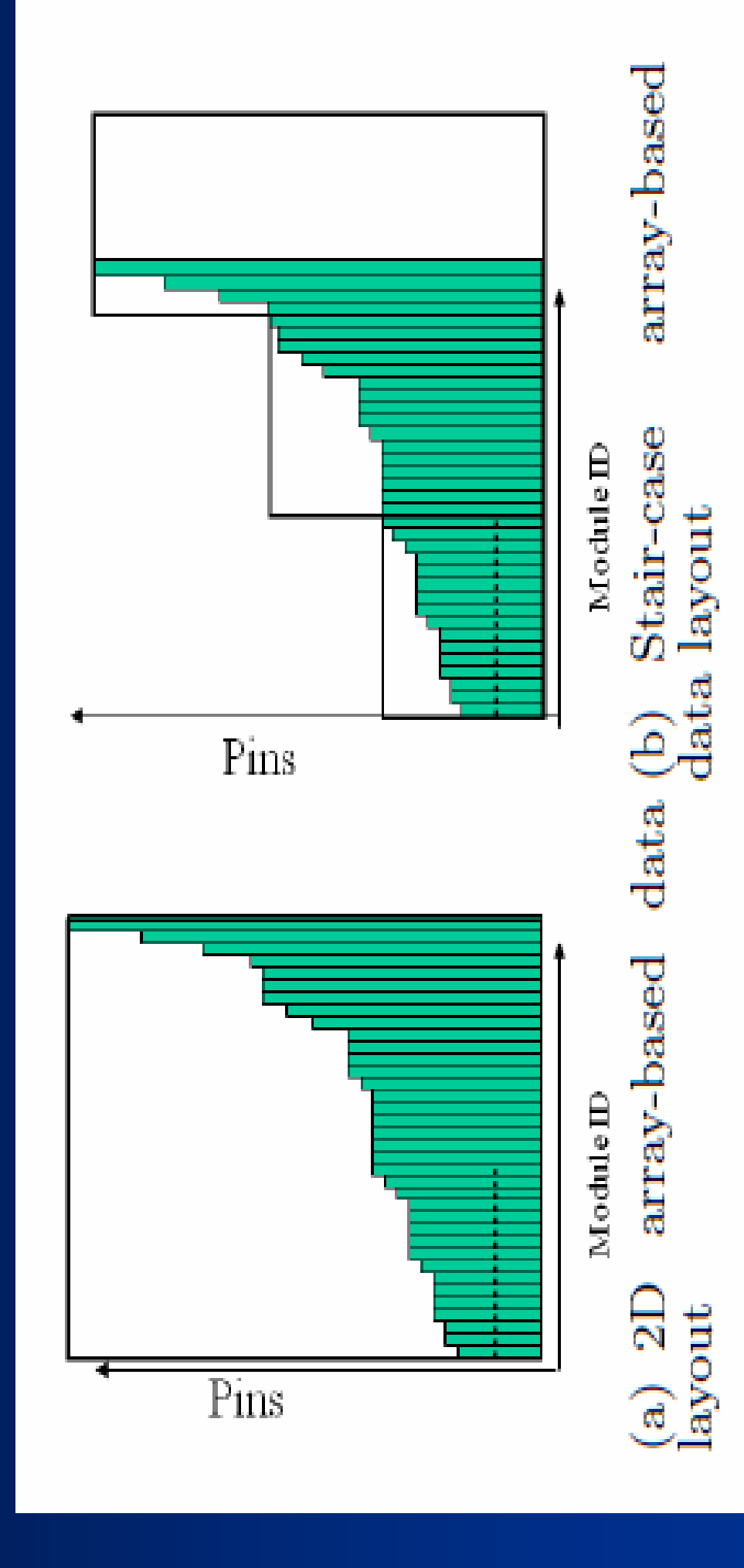
Inefficiency of atomic operations

### Data access optimization

- Removing atomic operations
- Reducing indirect memory access
- Coalesced access

Data layout for vector of vectors

Group uncoalesced access though wider access



### Bin density computation

Paralleling outerloop or innerloop?

Atomic operations

### Density smoothing

$$\text{SmoothedDensity} = \text{IDCT}_{2D}(\text{DCT}_{2D}(\text{BinDensity}) * \text{Weight})$$

Density smoothing is performed through 2D DCT/IDCT

Mainly use CUFFT to compute 2D DCT and 2D ICDDT.

For a very small matrix, direct matrix multiplication is even faster.

### Spreading force computation

Linear interpolation

Texture fetch

Fast reduction

```

1 Err_x = Err_y = 0;
2 ForAll module
3   F_spreading_x = \int_{y_{bottom}}^{y_{top}} \Psi(x_{left}, y) dy - \int_{y_{bottom}}^{y_{top}} \Psi(x_{right}, y) dy
4   F_spreading_y = \int_{x_{left}}^{x_{right}} \Psi(x, y_{bottom}) dx - \int_{x_{left}}^{x_{right}} \Psi(x, y_{top}) dx
5   Err_x += F_spreading_x; Err_y += F_spreading_y;
6 End ForAll

```

## Experiments

7~24X (avg. 15X) speedup over

Xeon E5405 (2.0GHZ) using

Tesla C1060

1% WL degradation after GP and

0.3% WL degradation after DP

## Discussions

Accuracy issues in the computation. Will double precision help?

Amdahl's law on the overall runtime.

Table 3: WL and run-time comparison

	WL(GP) (Original)	WL(DP) (Original)	WL(DP) (CUDA)	WL(DP) (CUDA) diff%	Time(GP) (Original)	Time(GP) (CUDA)	Time(GP) (CUDA) diff%	Speedup (GP)
IBM01	2.02E+06	1.96E+06	2.62	2.17E+06	2.16E+06	0.46	9.4	67.03
IBM05	9.25E+06	9.29E+06	-0.47	9.27E+06	9.33E+06	-0.64	16.73	184.06
IBM10	2.83E+07	2.80E+07	1.26	2.83E+07	2.80E+07	1.43	30.08	373.94
IBM15	4.87E+07	4.92E+07	-1.15	4.87E+07	4.92E+07	-0.33	53.43	1273.91
IBM20	7.87E+05	7.90E+05	-0.35	7.87E+05	7.90E+05	-0.34	60.99	17.46
PEK001	7.87E+05	7.90E+05	-0.35	7.87E+05	7.90E+05	-0.34	60.99	17.46
PEK005	2.01E+06	2.01E+06	0.06	2.01E+06	2.03E+06	0.43	10.38	146.38
PEK015	4.75E+06	4.72E+06	0.67	4.75E+06	4.78E+06	-0.34	19.63	388.48
PEK015	1.20E+07	1.33E+07	-10.28	1.42E+07	1.55E+07	-8.39	45.33	1113.48
PEK018	1.38E+07	1.31E+07	5.72	1.69E+07	1.62E+07	4.32	53.4	1197.52
PEK001	1.44E+06	1.32E+06	8.84	1.60E+06	1.51E+06	5.96	6.75	48.41
PEK005	3.74E+06	3.85E+06	-2.73	3.97E+06	4.08E+06	-2.70	11.24	151.31
PEK010	1.29E+07	1.31E+07	-1.69	1.33E+07	1.36E+07	-2.21	22.73	401.65
PEK015	4.60E+07	4.35E+07	5.40	4.71E+07	4.33E+07	0.92	52.9	1106.28
PEK018	5.35E+07	5.62E+07	-1.20	5.31E+07	5.48E+07	-1.28	62.58	1341.86
adaptes	3.31E+06	3.36E+06	-1.25	3.31E+06	3.36E+06	-1.30	62.58	692.57
adaptes	3.31E+06	3.36E+06	-1.25	3.31E+06	3.36E+06	-1.30	62.58	692.57
newblb2	2.07E+08	2.02E+08	2.46	2.02E+08	2.00E+08	1.00	209.36	3052.68
newblb3	2.67E+08	3.00E+08	1.43	2.93E+08	2.91E+08	0.69	331.48	3402.31
newblb4	2.67E+08	2.57E+08	3.66	2.69E+08	2.44E+08	2.05	296.74	4306.08
newblb5	4.72E+08	4.68E+08	0.89	4.72E+08	4.36E+08	0.23	725.03	9395.27
newblb6	5.30E+08	5.20E+08	1.97	4.93E+08	4.90E+08	1.02	736.51	9652.55
newblb7	1.17E+09	1.16E+09	1.52	1.09E+09	1.08E+09	0.93	1173.5	17228.2
Average								14.82