A Shared-Memory Parallel Implementation of the RePIAce Global Cell Placer

Frédéric Gessler, EPFL  
Philip Brisk, UCR  
Mirjana Stojilović, EPFL

Photo source: https://commons.wikimedia.org. Author: Saad Faruque.
VLSI Placement Matters

- Important step, with impact on
  - Timing closure
  - Die utilization
  - Routability
  - Design turnaround time

- **ePlace** and **RePIAce**
  - Academic placers
  - Electrostatic-based global placers
  - Best half-perimeter wirelength

---


Our Contribution

- **Address runtime concerns** on generic and widely available CPU-based hardware platforms: **multicore CPUs**
  - First detailed exploration of strategies to accelerate RePIAce via shared memory parallelism and **OpenMP**
  - Accelerate global placement by reducing memory traffic
  - At 12 CPU threads, $\approx 3 \times$ speedup over RePIAce and $\approx 3.5 \times$ speedup over DREAMPlace

A Shared-Memory Parallel Implementation of the RePLIAce Global Cell Placer

- **Background**
- Global Placement with RePLIAce
- Runtime analysis
- Reducing memory traffic
- Parallel implementations
  - wirelength
  - bin density
  - cost gradient
- Results
Background

- **Placement instance**: a hyper-graph $G = (V, E, R)$
  - $V$: vertices (cells, macros)
  - $E$: edges (nets)
  - $R$: placement region, $m \times m$ rectangular grids (bins)

- **Placement solution**: a pair of integer sets
  \[ v = \{x_1, \ldots, x_n, y_1, \ldots, y_n\} \]
  - $(x_i, y_i)$: placed location of the origin of the cell $c_i$
  - $n$: number of cells
A Shared-Memory Parallel Implementation of the RePIAce Global Cell Placer

• Background
• **Global Placement with RePIAce**
• Runtime analysis
• Reducing memory traffic
• Parallel implementations
  • wirelength
  • bin density
  • cost gradient
• Results
Global Placement with RePLAce: HPWL
Global Placement with RePIAce: HPWL

- **Half-perimeter wirelength (HPWL)** of a net $e$, where $i$ and $j$ are net pins and $(x, y)$ pair are pin coordinates:

$$\text{HPWL}(e) = \max_{i,j \in e} |x_i - x_j| + \max_{i,j \in e} |y_i - y_j|$$
Global Placement with RePIAce: HPWL

- **First objective:** minimize the HPWL of all nets:

\[
\text{HPWL}(v) = \sum_{e \in E} \left( \max_{i,j \in e} |x_i - x_j| + \max_{i,j \in e} |y_i - y_j| \right)
\]

- **Issue:** HPWL is not differentiable
Global Placement with RePIAce: HPWL

- Weighted-average smoothing strategy:

\[
\tilde{W}_x(e) = \frac{\sum_{i \in e} x_i \exp(x_i/\gamma)}{\sum_{i \in e} \exp(x_i/\gamma)} - \frac{\sum_{i \in e} x_i \exp(-x_i/\gamma)}{\sum_{i \in e} \exp(-x_i/\gamma)}
\]

- Smooth and differentiable total wirelength:

\[
\tilde{W}(v) = \tilde{W}_x(e) + \tilde{W}_y(e)
\]
Global Placement with RePIAc: Bin Density

- Bin density of a placement solution $v$:

$$\rho_b(v) = \sum_{c_i \in V} \text{overlap}_b(c_i)$$
Global Placement with RePIAce: Bin Density

- Bin density of a placement solution $v$:

$$\rho_b(v) = \sum_{c_i \in V} \text{overlap}_b(c_i)$$

- Second objective: bin density below a target value
Global Placement with RePIAce: Objective

- **Global objective**: minimizing the cost function $f$

$$\min_v f(v) = \tilde{W}(v) + \lambda N(v)$$

Balancing influence of wirelength and density

Density penalty
Global Placement with RePIAce

- RePIAce uses electrostatic analogy:
  - **Cells** are positively **charged particles**
  - Cell electric **charge** is the cell **area**
  - System **potential energy** (density penalty) and **electric field** (density gradient):
    - solving Poisson’s equation
    - using an **FFT** library
  - Cost optimization:
    - using **Nesterov’s method**

Reaching placement objective equivalent to attaining electrostatic equilibrium

Figure taken from J. Lu et al., “ePlace: Electrostatics based Placement using FFT and Nesterov’s Method”, TODAES, Vol. 20, No. 2, Feb. 2015
A Shared-Memory Parallel Implementation of the RePIAce Global Cell Placer

- Background
- Global Placement with RePIAce
- **Runtime analysis**
  - Reducing memory traffic
  - Parallel implementations
    - wirelength
    - bin density
    - cost gradient
- Results
**Runtime Analysis**

- **ISPD-2005/2006** benchmarks
- Global placement is the bottleneck (~81.7%)
Runtime Analysis

- **ISPD-2005/2006** benchmarks
- Global placement is the bottleneck (~81.7%)

We accelerate these three critical spots

Global placement breakdown, averaged over ISPD-2005/2006 benchmark circuits
Runtime Analysis

- **ISPD-2005/2006** benchmarks
- Global placement is the bottleneck (~81.7%)

Global placement breakdown, averaged over ISPD-2005/2006 benchmark circuits

We accelerate these three critical spots

Max expected speedup
2-12 threads: 1.49–2.51×
Runtime Analysis

- **ISPD-2005/2006 benchmarks**
- Global placement is the bottleneck (~81.7%)
Our Acceleration Strategies

1. Reduce memory traffic
2. Design efficient parallel implementation
A Shared-Memory Parallel Implementation of the RePIAce Global Cell Placer

- Background
- Global Placement with RePIAce
- Runtime analysis
- Reducing memory traffic
- Parallel implementations
  - wirelength
  - bin density
  - cost gradient
- Results
Reducing Memory Traffic

- ... and increase data locality
## Reducing Memory Traffic

<table>
<thead>
<tr>
<th>Pins in V</th>
<th>Nets in V</th>
<th>Cells in V</th>
<th>Bins in V</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pin_t</code></td>
<td><code>net_t</code></td>
<td><code>cell_phy_t</code></td>
<td><code>area_t</code></td>
</tr>
<tr>
<td>fpos2_t expL</td>
<td>pin_t <code>pinArray[]</code></td>
<td>pin_t** <code>pinArrayPtr</code></td>
<td>pos2_t <code>binStart</code></td>
</tr>
<tr>
<td>fpos2_t expR</td>
<td>pin_t</td>
<td>int <code>pinCNT</code></td>
<td>long int <code>terminArea</code></td>
</tr>
<tr>
<td>fpos2_t coord</td>
<td>sumNumL</td>
<td>char <code>type</code></td>
<td>float <code>virtArea</code></td>
</tr>
<tr>
<td>int <code>pinID</code></td>
<td>sumNumR</td>
<td>Size: 16 bytes</td>
<td>float <code>binDensity</code></td>
</tr>
<tr>
<td>int <code>moduleID</code></td>
<td>sumDenL</td>
<td></td>
<td>float <code>cellArea</code></td>
</tr>
<tr>
<td>fpos2_t <code>min</code></td>
<td>sumDenR</td>
<td></td>
<td>float <code>fillerArea</code></td>
</tr>
<tr>
<td>fpos2_t <code>max</code></td>
<td>int <code>pinCNT</code></td>
<td></td>
<td>float <code>potential</code></td>
</tr>
<tr>
<td>int <code>netID</code></td>
<td></td>
<td></td>
<td>Size: 32 bytes</td>
</tr>
<tr>
<td>char <code>metaData1</code></td>
<td></td>
<td></td>
<td>Size: 36 bytes</td>
</tr>
<tr>
<td>char <code>metaData2</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Size: 40 bytes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data for wirelength
### Reducing Memory Traffic

<table>
<thead>
<tr>
<th>Pins in V</th>
<th>Nets in V</th>
<th>Cells in V</th>
<th>Bins in V</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pin_t</code></td>
<td><code>net_t</code></td>
<td><code>cell_phy_t</code></td>
<td><code>area_t</code></td>
</tr>
<tr>
<td><code>fpos2_t expL</code></td>
<td><code>pin_t pinArray[]</code></td>
<td><code>pin_t** pinArrayPtr</code></td>
<td><code>pos2_t coord</code></td>
</tr>
<tr>
<td><code>fpos2_t expR</code></td>
<td><code>fpos2_t sumNumL</code></td>
<td><code>int pinCNT</code></td>
<td><code>long int terminArea</code></td>
</tr>
<tr>
<td><code>fpos2_t coord</code></td>
<td><code>fpos2_t sumNumR</code></td>
<td><code>char type</code></td>
<td><code>float virtArea</code></td>
</tr>
<tr>
<td><code>int pinID</code></td>
<td><code>fpos2_t sumDenL</code></td>
<td><code>Size: 16 bytes</code></td>
<td><code>float binDensity</code></td>
</tr>
<tr>
<td><code>int moduleID</code></td>
<td><code>fpos2_t sumDenR</code></td>
<td></td>
<td><code>float cellArea</code></td>
</tr>
<tr>
<td><code>fpos2_t min</code></td>
<td><code>fpos2_t min</code></td>
<td></td>
<td><code>float fillerDensity</code></td>
</tr>
<tr>
<td><code>fpos2_t max</code></td>
<td><code>fpos2_t max</code></td>
<td></td>
<td><code>float fillerArea</code></td>
</tr>
<tr>
<td><code>int pinCNT</code></td>
<td><code>fpos2_t size</code></td>
<td></td>
<td><code>float potential</code></td>
</tr>
<tr>
<td><code>Size: 40 bytes</code></td>
<td><code>float scale</code></td>
<td></td>
<td><code>Size: 36 bytes</code></td>
</tr>
</tbody>
</table>

- **Data for wirelength gradient**
- **Bins that overlap with the cell**
# Reducing Memory Traffic

## Pins in V

<table>
<thead>
<tr>
<th>pin_t</th>
<th>fpos2_t expL</th>
<th>fpos2_t expR</th>
<th>fpos2_t coord</th>
<th>int pinID</th>
<th>int moduleID</th>
<th>int netID</th>
<th>char metaData1</th>
<th>char metaData2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size: 40 bytes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Nets in V

<table>
<thead>
<tr>
<th>net_t</th>
<th>pin_t pinArray[]</th>
<th>fpos2_t sumNumL</th>
<th>fpos2_t sumNumR</th>
<th>fpos2_t sumDenL</th>
<th>fpos2_t sumDenR</th>
<th>fpos2_t min</th>
<th>fpos2_t max</th>
<th>int pinCNT</th>
<th>Size: 64 bytes</th>
</tr>
</thead>
</table>

## Cells in V

<table>
<thead>
<tr>
<th>cell_phy_t</th>
<th>cell_den_t</th>
<th>area_t</th>
<th>bin_t</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos2_t pinArrayPtr</td>
<td>pos2_t min</td>
<td>pos2_t coord</td>
<td>fpos2_t min</td>
</tr>
<tr>
<td>int pinCNT</td>
<td>pos2_t max</td>
<td>long int terminArea</td>
<td>fpos2_t max</td>
</tr>
<tr>
<td>char type</td>
<td>float virtArea</td>
<td>float binDensity</td>
<td>fpos2_t size</td>
</tr>
<tr>
<td>Size: 16 bytes</td>
<td>float fillerDensity</td>
<td>float scale</td>
<td>fpos2_t size</td>
</tr>
<tr>
<td></td>
<td>char type</td>
<td>float cellArea</td>
<td>fpos2_t size</td>
</tr>
<tr>
<td></td>
<td>Size: 48 bytes</td>
<td>float fillerArea</td>
<td>fpos2_t size</td>
</tr>
</tbody>
</table>

## Bins in V

<table>
<thead>
<tr>
<th>bin_t</th>
</tr>
</thead>
<tbody>
<tr>
<td>fpos2_t min</td>
</tr>
<tr>
<td>float cellArea</td>
</tr>
<tr>
<td>Size: 36 bytes</td>
</tr>
</tbody>
</table>

---

Loop-invariant info consumed by FFT solver

Bin data close to its electrical properties
A Shared-Memory Parallel Implementation of the RePIAce Global Cell Placer

- Background
- Global Placement with RePIAce
- Runtime analysis
- Reducing memory traffic
- **Parallel implementations**
  - wirelength
  - bin density
  - cost gradient
- Results
## Parallel Wirelength Computation

1: #pragma omp parallel num_threads(T) {
2: 
3: 
4: 
5:   for all nets $e \in E$ do
6:     numL, numR, denL, denR = 0
7:     for all pins $p \in e$ do
8:       Compute and update $p.expL$
9:       numL += $p.coord \cdot p.expL$
10:      denL += $p.expL$
11:     Compute and update $p.expR$
12:      numR += $p.coord \cdot p.expR$
13:     denR += $p.expR$
14:     end for
15:     $e.sumNumL = numL$, $e.sumNumR = numR$
16:     $e.sumDenL = denL$, $e.sumDenR = denR$
17:   end for
18: }
Parallel Wirelength Computation

1: #pragma omp parallel num_threads(T) {
2:     numL, numR, denL, denR = 0
3:     for all nets e ∈ E do
4:         for all pins p ∈ e do
5:             Compute and update p.expL
6:                 numL += p.coord · p.expL
7:                 denL += p.expL
8:             end for
9:             Compute and update p.expR
10:                numR += p.coord · p.expR
11:                denR += p.expR
12:         end for
13:     e.sumNumL = numL, e.sumNumR = numR
14:     e.sumDenL = denL, e.sumDenR = denR
15: }
Parallel Wirelength Computation

\[
\tilde{W}_x(e) = \frac{\sum_{i \in e} x_i \exp(x_i / \gamma)}{\sum_{i \in e} \exp(x_i / \gamma)} - \frac{\sum_{i \in e} x_i \exp(-x_i / \gamma)}{\sum_{i \in e} \exp(-x_i / \gamma)}
\]

1: #pragma omp parallel num_threads(T) {
2:
3:
4:
5: for all nets \( e \in E \) do
6: \hspace{1em} numL, numR, denL, denR = 0
7: for all pins \( p \in e \) do
8: \hspace{2em} Compute and update \( p.expL \)
9: \hspace{2em} numL += p.coord \cdot p.expL
10: \hspace{2em} denL += p.expL
11: \hspace{2em} Compute and update \( p.expR \)
12: \hspace{2em} numR += p.coord \cdot p.expR
13: \hspace{2em} denR += p.expR
14: end for
15: e.sumNumL = numL, e.sumNumR = numR
16: e.sumDenL = denL, e.sumDenR = denR
17: end for
18: }

Our slim data structure \texttt{pin_t}
Parallel Wirelength Computation

1: #pragma omp parallel num_threads(T) {
2:  
4:  
5:  for all nets $e \in E$ do
6:    numL, numR, denL, denR = 0
7:    for all pins $p \in e$ do
8:      Compute and update $p.expL$
9:      numL += $p.coord \cdot p.expL$
10:     denL += $p.expL$
11:    Compute and update $p.expR$
12:     numR += $p.coord \cdot p.expR$
13:     denR += $p.expR$
14:   end for
15:   e.sumNumL = numL, e.sumNumR = numR
16:   e.sumDenL = denL, e.sumDenR = denR
17:  end for
18: }

\[
\tilde{W}_x(e) = \sum_{i \in e} x_i \exp(x_i / \gamma) - \sum_{i \in e} x_i \exp(-x_i / \gamma)
\]
Parallel Wirelength Computation

1: #pragma omp parallel num_threads(T) {
2:   t = omp get thread num()
3:   start = (t ≥ 1) ? w[t − 1] : 0
4:   end = (t < T − 1) ? w[t] : netCNT
5:   for all nets e ∈ E, start ≤ i < end do
6:     numL, numR, denL, denR = 0
7:     for all pins p ∈ e do
8:       Compute and update p.expL
9:       numL += p.coord · p.expL
10:      denL += p.expL
11:     Compute and update p.expR
12:     numR += p.coord · p.expR
13:      denR += p.expR
14:    end for
15:   e.sumNumL = numL, e.sumNumR = numR
16:   e.sumDenL = denL, e.sumDenR = denR
17: end for
18: }

Workload balancing: $\frac{1}{T} \sum_{e \in E} e.pinCNT$
Parallel Wirelength Computation: Results

- Workload balance overview for **ISPD-2005** benchmarks
- On average, **0.4%** away from the mean
Parallel Wirelength Computation: Results

- Impact of scheduling on runtime, example **BIGBLUE4**
A Shared-Memory Parallel Implementation of the RePIAce Global Cell Placer

• Background
• Global Placement with RePIAce
• Runtime analysis
• Reducing memory traffic
• Parallel implementations
  • wirelength
  • bin density
  • cost gradient
• Results
Parallel Bin Density Computation

- Sum of overlap surface areas between bins and placed cells
- Memory bound
- Naïve strategy (allocating subset of cells per thread) prone to data races.

Options:
- **Iterate over bins instead**
  - High synchronization overhead, cells >> bins
- **std-C++11 atomic variables, compare-and-swap**
  - Scales poorly, high likelihood of collisions
- **Privatization** of overlap sums
#pragma omp parallel num_threads(T) {
    t = omp get thread num()

Allocate and clear two per-thread matrices of partial bin-overlap sums: binOverFiller, binOverCell = 0

for all cells c ∈ V do
    c.binStart = (c.min - ORIGIN)/STEP
    c.binEnd = (c.max - ORIGIN)/STEP

for all bins bᵢ ∈ B, c.binStart ≤ i ≤ c.binEnd do
    minCorner = max(bᵢ.min, c.min),
    maxCorner = min(bᵢ.max, c.max)
    overlap = RectArea(minCorner, maxCorner)

    if c ∈ FillerCells, FillerCells ⊂ V then
        binOverFiller[t][i] += overlap · c.scale
    else
        binOverCell[t][i] += overlap · c.scale
end if
end for
end for
}
Loop over all cells

1: `#pragma omp parallel num_threads(T) {`
2: `t = omp get thread num()`
3: `Allocate and clear two per-thread matrices of partial bin-overlap sums: binOverFiller, binOverCell = 0`
4: `for all cells c ∈ V do`
5: `c.binStart = (c.min - ORIGIN)/STEP`
6: `c.binEnd = (c.max - ORIGIN)/STEP`
7: `for all bins b_i ∈ B, c.binStart ≤ i ≤ c.binEnd do`
8: `minCorner = max(b_i.min, c.min),`
9: `maxCorner = min(b_i.max, c.max)`
10: `overlap = RectArea(minCorner, maxCorner)`
11: `if c ∈ FillerCells, FillerCells ⊂ V then`
12: `binOverFiller[t][i] += overlap · c.scale`
13: `else`
14: `binOverCell[t][i] += overlap · c.scale`
15: `end if`
16: `end for`
17: `end for`
18: `}`
Parallel Bin Density Computation

1: #pragma omp parallel num_threads(T) {
2:   \textcolor{magenta}{t = omp get thread num()}
3:   \textcolor{magenta}{Allocate and clear two per-thread matrices of partial bin-overlap sums: binOverlapFiller, binOverCell = 0}
4:   \textcolor{magenta}{for all cells} \textcolor{blue}{c \in V} \textcolor{magenta}{do}
5:     c.binStart = (c.min - ORIGIN)/\text{STEP}
6:     c.binEnd = (c.max - ORIGIN)/\text{STEP}
7:     \textcolor{magenta}{for all bins} \textcolor{blue}{b_i \in B, c.binStart \leq i \leq c.binEnd} \textcolor{magenta}{do}
8:       \textcolor{magenta}{minCorner} = \text{max}(b_i.min, c.min),
9:       \textcolor{magenta}{maxCorner} = \text{min}(b_i.max, c.max)
10:      overlap = RectArea(minCorner, maxCorner)
11:      \textcolor{magenta}{if} \textcolor{blue}{c \in FillerCells, FillerCells \subset V} \textcolor{magenta}{then}
12:        binOverlapFiller[t][i] += overlap \cdot c.scale
13:      \textcolor{magenta}{else}
14:        binOverCell[t][i] += overlap \cdot c.scale
15:      \textcolor{magenta}{end if}
16:   \textcolor{magenta}{end for}
17: \textcolor{magenta}{end for}
18: }
Parallel Bin Density Profiling

- Runtime of **ADAPTEC1** benchmark, atomic variables vs. private overflow sums

![Chart showing runtime comparison between atomic variables and private overflow sums for different number of threads.]

- Execution time (s)
  - 1 thread: Atomic variables = 10.64 s, Private overflow sums = 7.81 s
  - 2 threads: Atomic variables = 8.53 s, Private overflow sums = 6.57 s
  - 4 threads: Atomic variables = 12.11 s, Private overflow sums = 4.79 s
  - 8 threads: Atomic variables = 7.80 s, Private overflow sums = 3.78 s
  - 12 threads: Atomic variables = 6.37 s, Private overflow sums = 3.80 s
A Shared-Memory Parallel Implementation of the RePIAce Global Cell Placer

- Background
- Global Placement with RePIAce
- Runtime analysis
- Reducing memory traffic
- Parallel implementations
  - wirelength
  - bin density
  - cost gradient
- Results
Parallel Cost Gradient

\[
\min_{v} f(v) = \tilde{W}(v) + \lambda N(v)
\]

- Three subroutines, carried out in sequence for all cells
  - Wirelength gradient
  - Density penalty
  - Cost preconditioning (only 5% of the runtime, thus ignored)

- Loop fission
  - Three loops instead of (originally) one
  - Independent parallelization strategies per loop
Parallel Cost Gradient

- Cost gradient computation, example **ADAPTEC1**
- Three independent loops scale considerably better
Parallel Cost Gradient

- Workload-balanced static scheduling vs naïve static scheduling, example **ADAPTEC1**
A Shared-Memory Parallel Implementation of the RePIAce Global Cell Placer

- Background
- Global Placement with RePIAce
- Runtime analysis
- Reducing memory traffic
- Parallel implementations
  - wirelength
  - bin density
  - cost gradient
- Results
Experimental Setup

- **C++**, g++ 7.3.0, optimization –O3
- **SimPL** for initial, **NTUplace3** for detailed placement
- **ISPD-2005 benchmarks**
- RePlAce could not place circuits with movable macros

**Hardware:**
- AMD Ryzen Threadripper 1920X
- 12 cores, 64GB DRAM@2134MHz, 32MB LL cache
# Global Placement ISPD-2005

<table>
<thead>
<tr>
<th></th>
<th>RePlAce</th>
<th>Sequential</th>
<th>T=1</th>
<th>T=2</th>
<th>T=4</th>
<th>T=8</th>
<th>T=12</th>
<th>RePlAce (T=12)</th>
<th>DREAMPlace (T=12)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GP TOTAL</td>
<td>GP TOTAL</td>
<td>GP TOTAL</td>
<td>GP TOTAL</td>
<td>GP TOTAL</td>
<td>GP TOTAL</td>
<td>GP TOTAL</td>
<td>GP TOTAL</td>
<td>GP TOTAL</td>
</tr>
<tr>
<td>ADAPTEC1</td>
<td>108.68</td>
<td>189.30</td>
<td>59.60</td>
<td>140.21</td>
<td>58.68</td>
<td>137.97</td>
<td>47.57</td>
<td>117.91</td>
<td>39.43</td>
</tr>
<tr>
<td>ADAPTEC2</td>
<td>210.07</td>
<td>309.29</td>
<td>134.40</td>
<td>234.44</td>
<td>134.16</td>
<td>232.65</td>
<td>113.82</td>
<td>202.51</td>
<td>93.54</td>
</tr>
<tr>
<td>ADAPTEC3</td>
<td>443.67</td>
<td>628.37</td>
<td>294.82</td>
<td>481.08</td>
<td>298.33</td>
<td>484.75</td>
<td>231.02</td>
<td>395.15</td>
<td>179.49</td>
</tr>
<tr>
<td>ADAPTEC4</td>
<td>497.98</td>
<td>691.84</td>
<td>346.26</td>
<td>537.55</td>
<td>350.52</td>
<td>539.44</td>
<td>268.59</td>
<td>441.89</td>
<td>202.76</td>
</tr>
<tr>
<td>BIGBLUE1</td>
<td>205.09</td>
<td>308.17</td>
<td>106.40</td>
<td>209.47</td>
<td>107.72</td>
<td>209.15</td>
<td>84.17</td>
<td>174.03</td>
<td>69.56</td>
</tr>
<tr>
<td>BIGBLUE2</td>
<td>376.93</td>
<td>607.11</td>
<td>257.62</td>
<td>490.28</td>
<td>258.59</td>
<td>487.54</td>
<td>202.43</td>
<td>409.36</td>
<td>154.68</td>
</tr>
<tr>
<td>BIGBLUE3</td>
<td>N/A N/A</td>
<td>N/A N/A</td>
<td>N/A N/A</td>
<td>N/A N/A</td>
<td>N/A N/A</td>
<td>N/A N/A</td>
<td>N/A N/A</td>
<td>N/A N/A</td>
<td>N/A N/A</td>
</tr>
<tr>
<td>BIGBLUE4</td>
<td>2461.81</td>
<td>3478.51</td>
<td>1571.31</td>
<td>2580.64</td>
<td>1675.59</td>
<td>2678.92</td>
<td>1335.57</td>
<td>2212.92</td>
<td>1003.35</td>
</tr>
<tr>
<td>geomean</td>
<td>1.00</td>
<td>1.00</td>
<td>1.60</td>
<td>1.33</td>
<td>1.58</td>
<td>1.33</td>
<td>1.99</td>
<td>1.59</td>
<td>2.53</td>
</tr>
<tr>
<td>HPWL(%)</td>
<td>0.00%</td>
<td>-0.08%</td>
<td>-0.09%</td>
<td>-0.10%</td>
<td>-0.10%</td>
<td>-0.10%</td>
<td>-0.09%</td>
<td>-0.09%</td>
<td>-0.32%</td>
</tr>
</tbody>
</table>
### Global Placement ISPD-2005

#### RePlAce vs. Sequential

<table>
<thead>
<tr>
<th></th>
<th>RePlAce</th>
<th>Sequential</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GP</td>
<td>TOTAL</td>
</tr>
<tr>
<td>ADAPTEC1</td>
<td>108.68</td>
<td>189.30</td>
</tr>
<tr>
<td>ADAPTEC2</td>
<td>210.07</td>
<td>309.29</td>
</tr>
<tr>
<td>ADAPTEC3</td>
<td>443.67</td>
<td>628.37</td>
</tr>
<tr>
<td>ADAPTEC4</td>
<td>497.98</td>
<td>691.84</td>
</tr>
<tr>
<td>BIGBLUE1</td>
<td>205.09</td>
<td>308.17</td>
</tr>
<tr>
<td>BIGBLUE2</td>
<td>376.93</td>
<td>607.11</td>
</tr>
<tr>
<td>BIGBLUE3</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>BIGBLUE4</td>
<td>2461.31</td>
<td>3478.51</td>
</tr>
<tr>
<td>geomean</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>△HPWL(%)</td>
<td>0.00%</td>
<td>-0.08%</td>
</tr>
</tbody>
</table>

**RePlAce + slim datastructures**

**Parallel RePlAce + slim datastructures**

**DREAMPlace on CPU**

**Multithreaded RePlAce**

**ΔHPWL within 0.1–0.3%**
### Individual Speedups ISPD-2005

<table>
<thead>
<tr>
<th></th>
<th>Global placement</th>
<th>Wirelength</th>
<th>Bin density</th>
<th>Cost gradient</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SEQ T=1 T=2 T=4 T=8 T=12</td>
<td>SEQ T=1 T=2 T=4 T=8 T=12</td>
<td>SEQ T=1 T=2 T=4 T=8 T=12</td>
<td>SEQ T=1 T=2 T=4 T=8 T=12</td>
</tr>
<tr>
<td>ADAPTEC1</td>
<td>1.82 1.85 2.28 2.76 3.05 3.09</td>
<td>1.76 1.87 2.53 3.01 3.26 3.34</td>
<td>5.39 4.48 5.33 7.31 9.27 9.21</td>
<td>2.22 2.39 3.58 5.10 6.58 7.17</td>
</tr>
<tr>
<td>ADAPTEC2</td>
<td>1.56 1.57 1.85 2.25 2.51 2.54</td>
<td>1.67 1.78 2.41 2.78 3.14 3.13</td>
<td>4.36 3.56 4.29 6.65 8.67 8.41</td>
<td>1.96 2.11 2.97 4.53 6.11 6.73</td>
</tr>
<tr>
<td>ADAPTEC3</td>
<td>1.50 1.49 1.92 2.47 2.93 3.07</td>
<td>1.63 1.74 2.27 2.60 2.83 2.86</td>
<td>3.19 2.74 3.88 6.63 9.45 9.99</td>
<td>1.73 1.85 2.63 3.85 5.46 6.44</td>
</tr>
<tr>
<td>ADAPTEC4</td>
<td>1.44 1.42 1.85 2.46 3.01 3.19</td>
<td>1.51 1.61 1.95 2.23 2.40 2.44</td>
<td>2.88 2.50 3.75 6.50 9.73 10.68</td>
<td>1.64 1.76 2.51 3.70 5.52 6.53</td>
</tr>
<tr>
<td>BIGBLUE1</td>
<td>1.93 1.90 2.44 2.95 3.26 3.37</td>
<td>1.64 1.72 2.27 2.63 2.92 3.06</td>
<td>5.56 4.48 5.93 9.42 11.50 11.99</td>
<td>2.37 2.50 3.76 5.35 6.61 7.08</td>
</tr>
<tr>
<td>BIGBLUE2</td>
<td>1.46 1.46 1.86 2.44 2.85 3.00</td>
<td>1.54 1.64 2.17 2.52 2.70 2.75</td>
<td>3.01 2.63 3.60 6.16 9.07 9.76</td>
<td>1.73 1.85 2.61 3.91 5.18 6.08</td>
</tr>
<tr>
<td>BIGBLUE4</td>
<td>1.57 1.47 1.80 2.45 3.00 3.28</td>
<td>1.81 1.79 2.29 3.07 3.41 3.56</td>
<td>3.16 2.53 3.20 5.71 8.72 9.26</td>
<td>1.81 1.82 2.51 3.61 5.26 5.82</td>
</tr>
<tr>
<td>geomean</td>
<td>1.60 1.58 1.99 2.53 2.94 3.06</td>
<td>1.65 1.73 2.29 2.68 2.93 2.99</td>
<td>3.80 3.18 4.20 6.83 9.45 9.95</td>
<td>1.91 2.02 2.89 4.25 5.79 6.63</td>
</tr>
</tbody>
</table>

- **Table**: The table compares the speedups of different placement tools and methods for ISPD-2005. The speedups are given in terms of multipliers for wirelength, bin density, and cost gradient improvements.

- **Geomean**: The geometric mean of the speedups is also provided, indicating the overall improvement across different metrics.
Conclusions

- **Careful data design** resulted in considerable reduction in memory traffic congestion

- Carefully chosen openMP **scheduling strategies** and **code refactoring** resulted in good speedup

- Not all possibilities have been explored, there is still space for improvement
  - Although, GPU-placers (such as **DREAMPlace**) are likely gaining this speed challenge
Thank you!

mirjana.stojilovic@epfl.ch