Regularity-Constrained Floorplanning for Multi-Core Processors

Xi Chen Jiang Hu Ning Xu

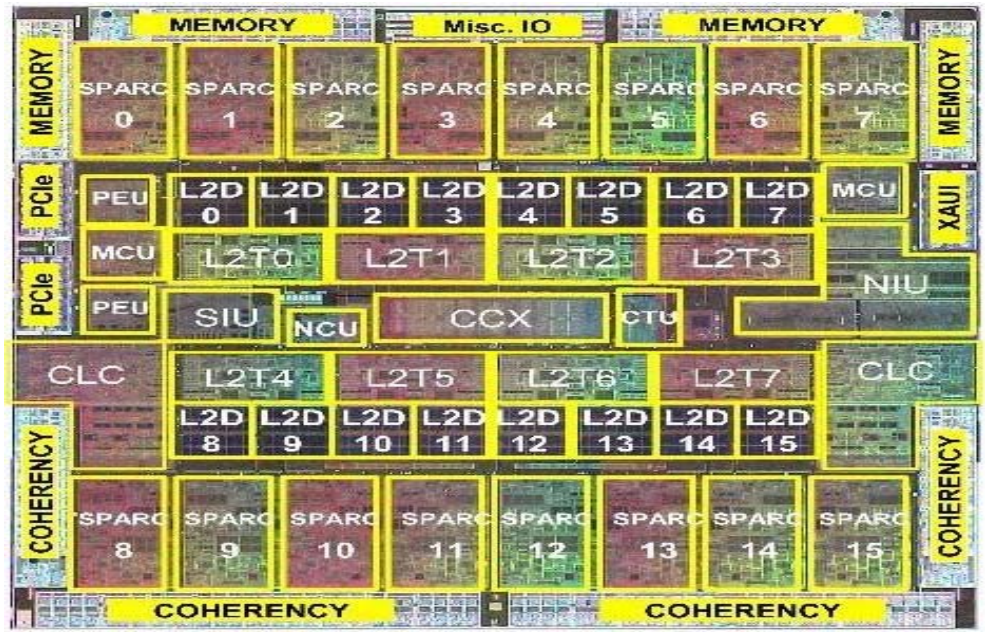
Department of ECE
Texas A&M University

College of CST
Wuhan University of Technology

Outline

- Introduction
- Floorplanning with Regularity Constraint
- Experimental Results
- Conclusions and Future Research

Floorplanning for Multi-core Processors



SUN Niagara-3 processor

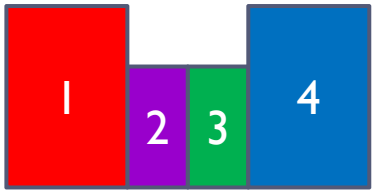
- Identical modules are placed in arrays
- One array can be embedded in another array
- Random blocks can be placed within an array

Symmetry Constraint in Analog Circuit Layout

- Similar to symmetry constraint in analog design
- For sequence-pair (α,β), block A and B is symmetry-feasible if for any block A and B

$$\alpha_A^{-1} < \alpha_B^{-1} \leftrightarrow \beta_{\delta(B)}^{-1} < \beta_{\delta(A)}^{-1}$$

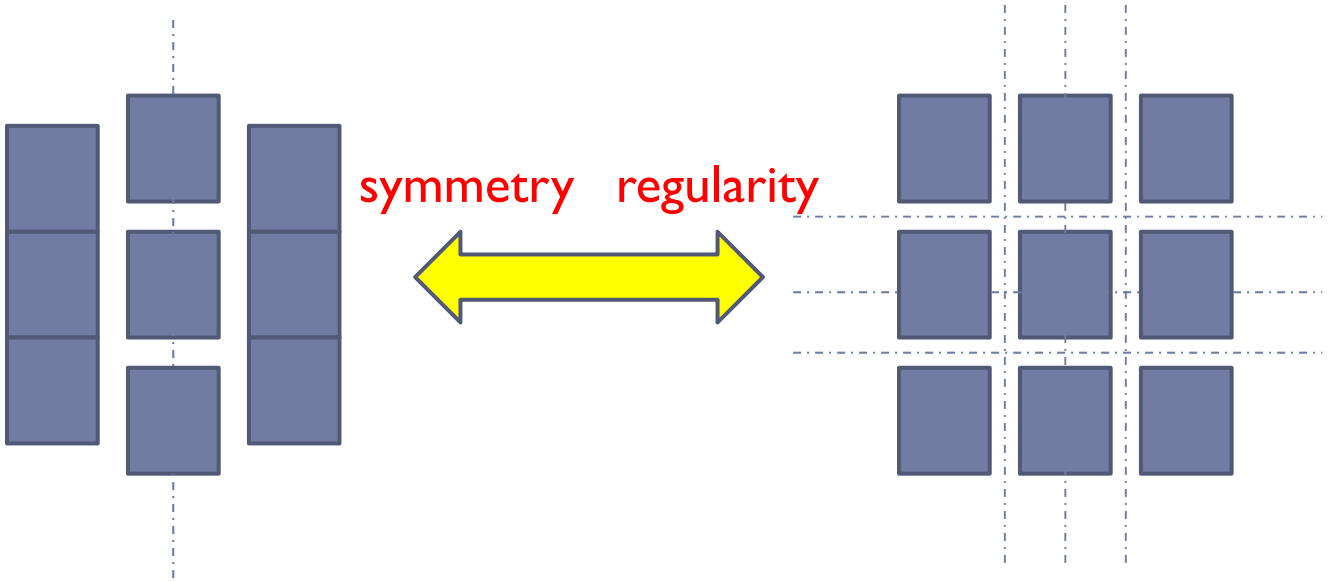
1. α_A^{-1} denotes the position of block A in sequence α
2. δ(A) is block symmetric to A



(1234,1234)

Regularity Constraint vs. Symmetry Constraint

- Regularity constraint can be treated as an extension to symmetry constraint
- However, the number of implicit symmetry constraints can be quite **large**

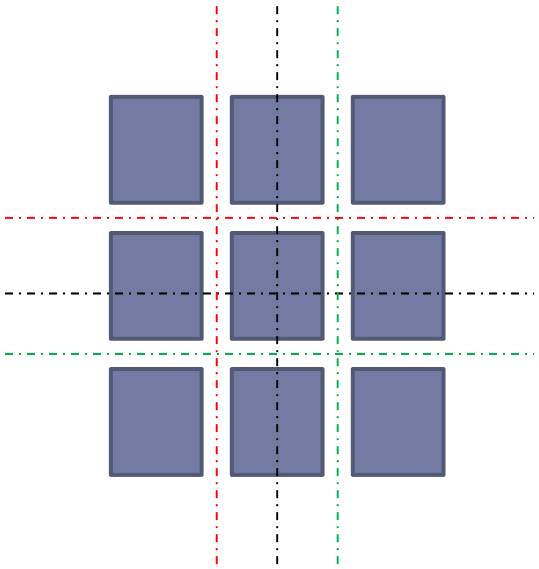


Regularity Constraint Factorization

- A chip with m cores can be placed in a $p \times q$ array:

  e.g. m=24=3×8=4×6=6×4=8×3

- For specific factorization, symmetries for different axes need to be maintained



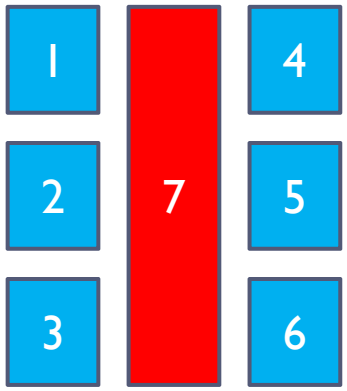
Outline

- Introduction
- Floorplanning with Regularity Constraint
- Experimental Results
- Conclusions and Future Research

Array and Non-array Blocks

- ***Array group*** is a subset of blocks that must be placed in a regular array
- If a block is in an array group, it is an ***array block***
- Otherwise called ***non-array block***

Array block: 1,2,3,4,5,6
Non-array block: 7



Problem Formulation

- Objective:

Minimize cost=(1-λ)×area + λ×wirelength

Constraints:

(1) Regularity Constraint

(2) Allow non-array block in the array group

λ is a weighting factor

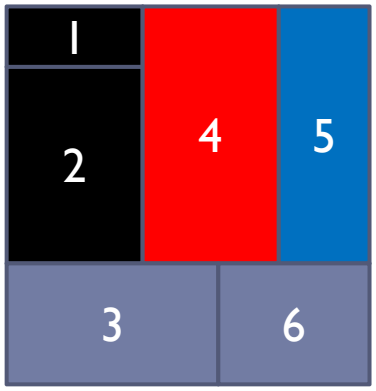
Algorithm Overview

- Using simulated annealing algorithm with sequence-pair representation
- Key contribution:

  1. How to encode the regularity constraint in sequence-pair
  2. How to achieve the regularity in packing procedure

Sequence Pair

- A sequence-pair like (<… *i* … *j* …>,<… *i* … *j* …>) implies that block *i* is to the left of block *j*
- A sequence-pair like (<… *i* … *j* …>,<… *j* … *i* …>) implies that block *i* is above block *j*



(<12<u>45</u>36>,<36<u>21</u>45>)

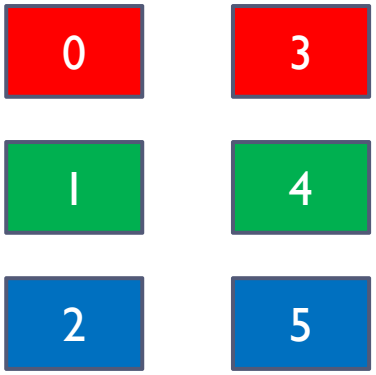
Common Subsequence

- Definition 1: Common Subsequence

  A set of q blocks $b_1, b_2 \ldots b_q$ form a common subsequence [Tang, Tian and Wong, DATE 2000] in a sequence-pair (α,β) if $\alpha_1^{-1} < \alpha_2^{-1} < \ldots < \alpha_q^{-1}$ and $\beta_1^{-1} < \beta_2^{-1} < \ldots < \beta_q^{-1}$

  where α_i^{-1} (β_i^{-1}) indicates the position of block b_i in sequence $\alpha(\beta)$

sequence pair (<0 3 1 4 2 5>,<2 5 1 4 0 3>)



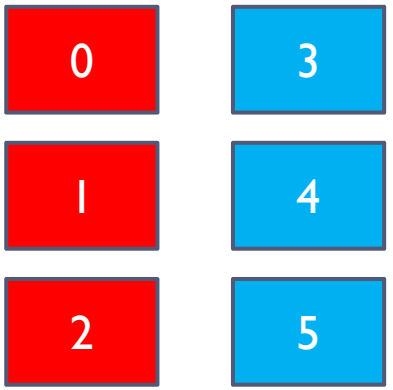
Reversely Common Subsequence

- Definition 2: Reversely Common Subsequence

  A set of q blocks $b_1, b_{2\ldots} b_q$ form a reversely common subsequence in a sequence-pair (α,β) if $\alpha_1^{-1} < \alpha_2^{-1} < \ldots < \alpha_q^{-1}$ and $\beta_1^{-1} > \beta_2^{-1} > \ldots > \beta_q^{-1}$

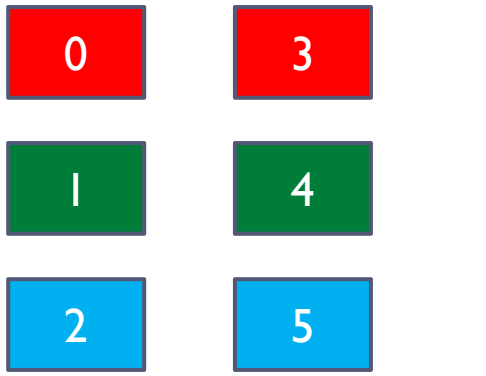
  where α_i^{-1} (β_i^{-1}) indicates the position of block b_i in sequence $\alpha(\beta)$

sequence pair (<0 1 2 3 4 5>,<2 1 0 5 4 3>)

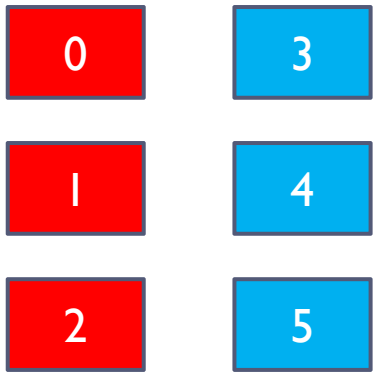


Necessary Condition

- Lemma 1 *The necessary condition that m blocks lead to a p×q array floorplan: the m blocks constitute p common subsequences of length q or vise versa*



(<0 3 1 4 2 5>,<2 5 1 4 0 3>)



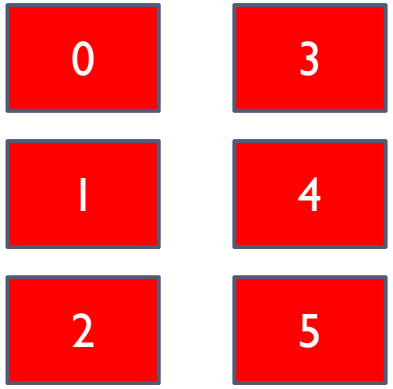
(<0 1 2 3 4 5>,<2 1 0 5 4 3>)

Regularity Subsequence-pair

- Definition 3: Regularity subsequence-pair(RSP)

  A contiguous subsequence of length m that satisfies Lemma 1 in a sequence-pair is called regularity subsequence-pair

The right figure can be represented as either
(<0 3 1 4 2 5>, <2 5 1 4 0 3>) or
(<0 1 2 3 4 5>, <2 1 0 5 4 3>)



Row (Column)-based Regularity Subsequence-pair

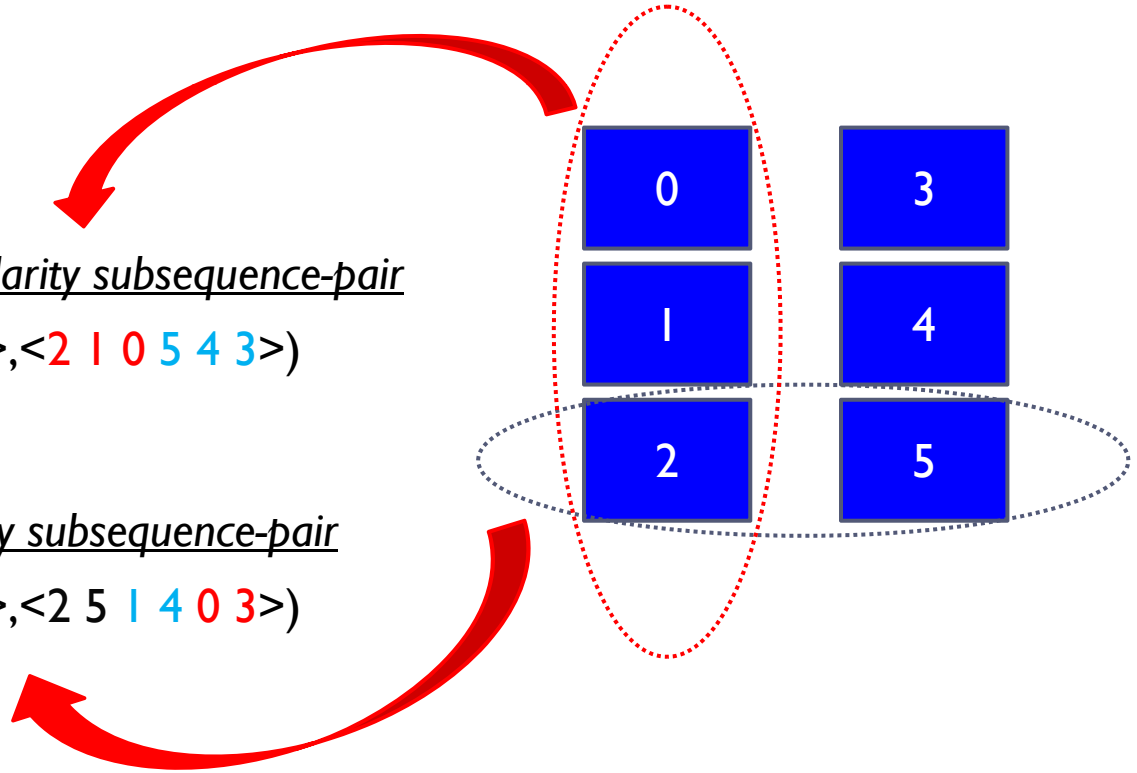
- Definition 4: *Row (column) based regularity subsequence-pair is a regularity subsequence-pair where each (inversely) common subsequence corresponding a row (column) is contiguous*

column based regularity subsequence-pair

(<0 1 2 3 4 5>,<2 1 0 5 4 3>)

row based regularity subsequence-pair

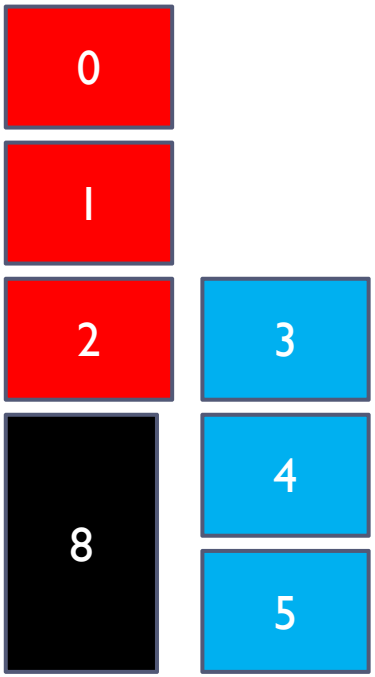
(<0 3 1 4 2 5>,<2 5 1 4 0 3>)



Non-array Block in Regularity Subsequence-pair

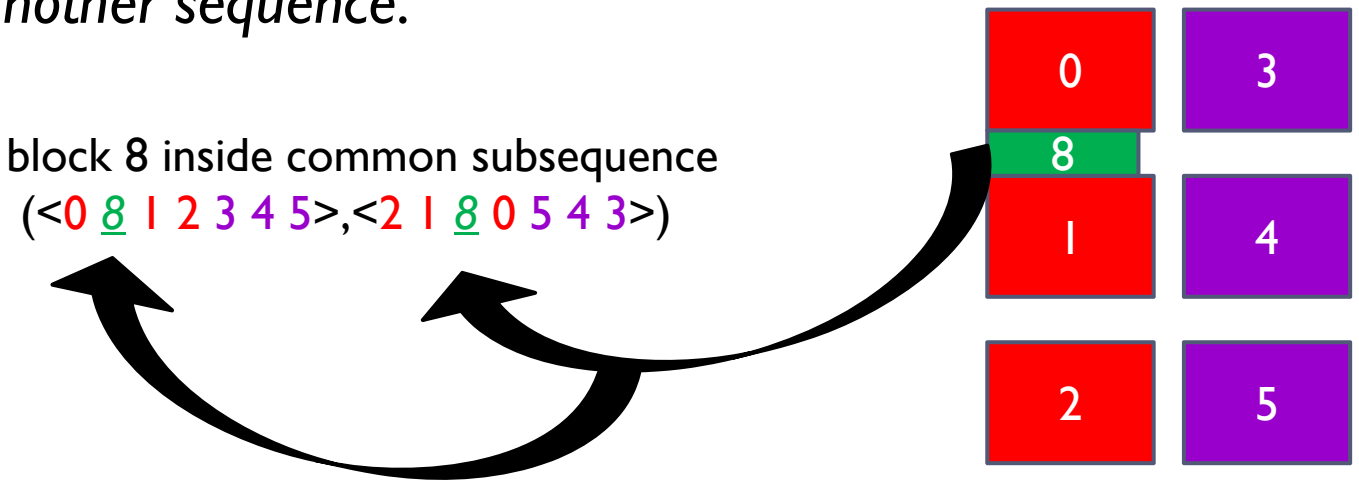
- Rule 1: *A non-array block*
- *Allowed: between both or neither of sequences of a regularity subsequence pair*
- *Disallowed: between any one sequence but outside of the other*

For example: in the right figure, we do not allow
(<0 1 2 ***8*** 3 4 5>,<***8*** 2 1 0 5 4 3>)



Non-array Block in Common Subsequence

- Rule 2: *A non-array block*
- *Allowed: inside both or neither of a contiguous (reversely) common subsequence in a row (column) base regularity subsequence-pair*
- *Disallowed: within one common subsequence, but outside that one in another sequence.*



Packing Methods

- Longest Path Algorithm, [Murata, Fujiyoshi, Nakatake and Kajitani, *TCAD 1996*]
- Longest Common Sequence (LCS), [Tang, Tian and Wong, *DATE 2000*]
- In this work, we adopt the LCS approach

Packing with Regularity

- Regularity implies the alignment and spacing constraints:
  Array blocks must be horizontally (vertically) aligned

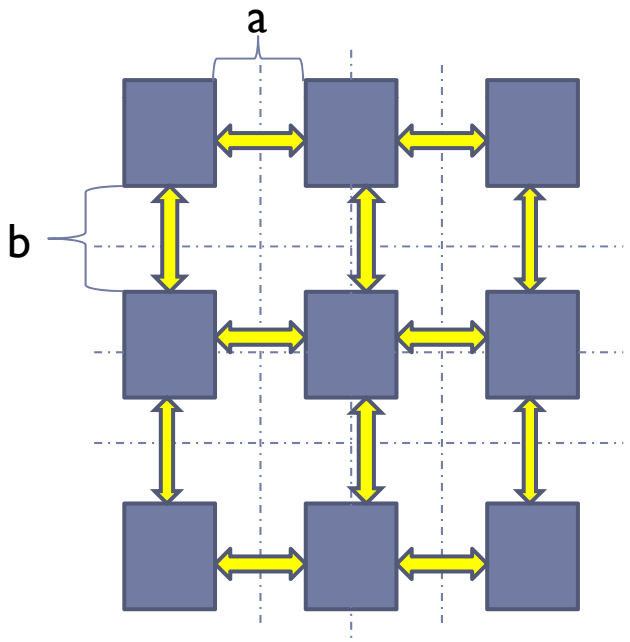
- Math expression:

$$X_{i,j} - X_{i,j-1} = X_{i,j+1} - X_{i,j}$$
$$Y_{i,j} - Y_{i-1,j} = Y_{i+1,j} - Y_{i,j}$$

1. where X, Y are x and y coordinates of the lower-left corner of an array block
2. i(j) represents row (column) index

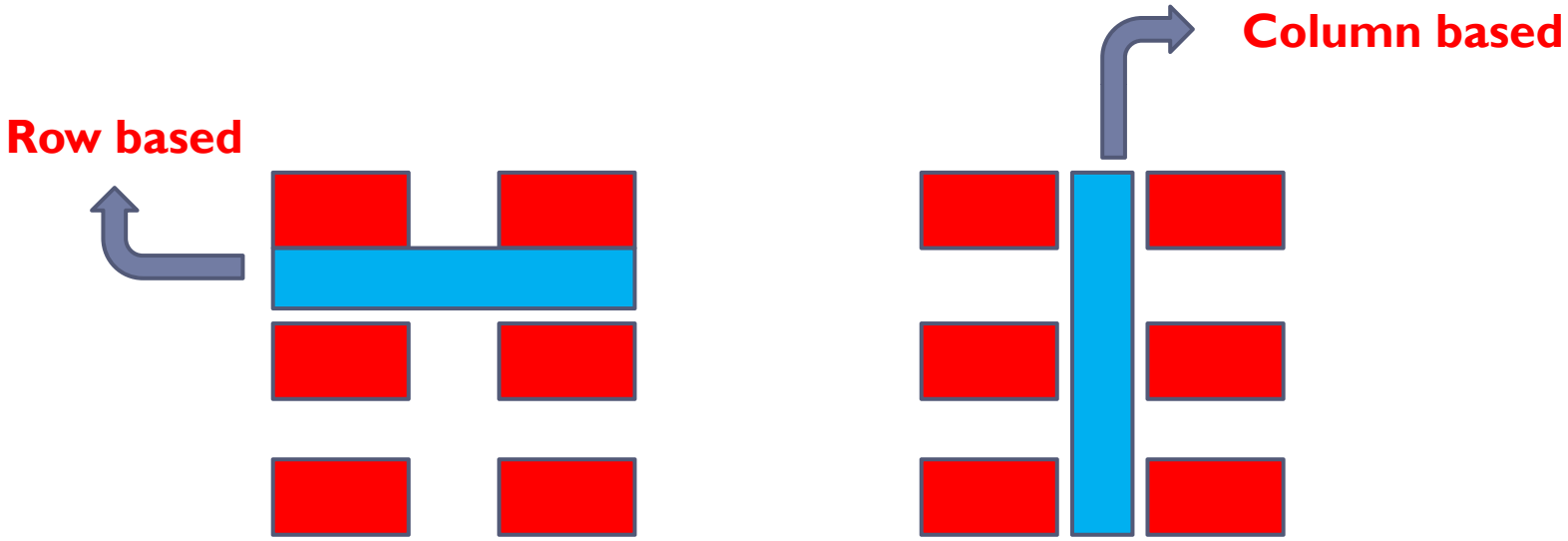
Regularity Illustration

$$X_{i,j} - X_{i,j-1} = X_{i,j+1} - X_{i,j}$$
$$Y_{i,j} - Y_{i-1,j} = Y_{i+1,j} - Y_{i,j}$$



Column-based and Row-based Encoding

- **Column-based** and **Row-based** encoding are both needed.

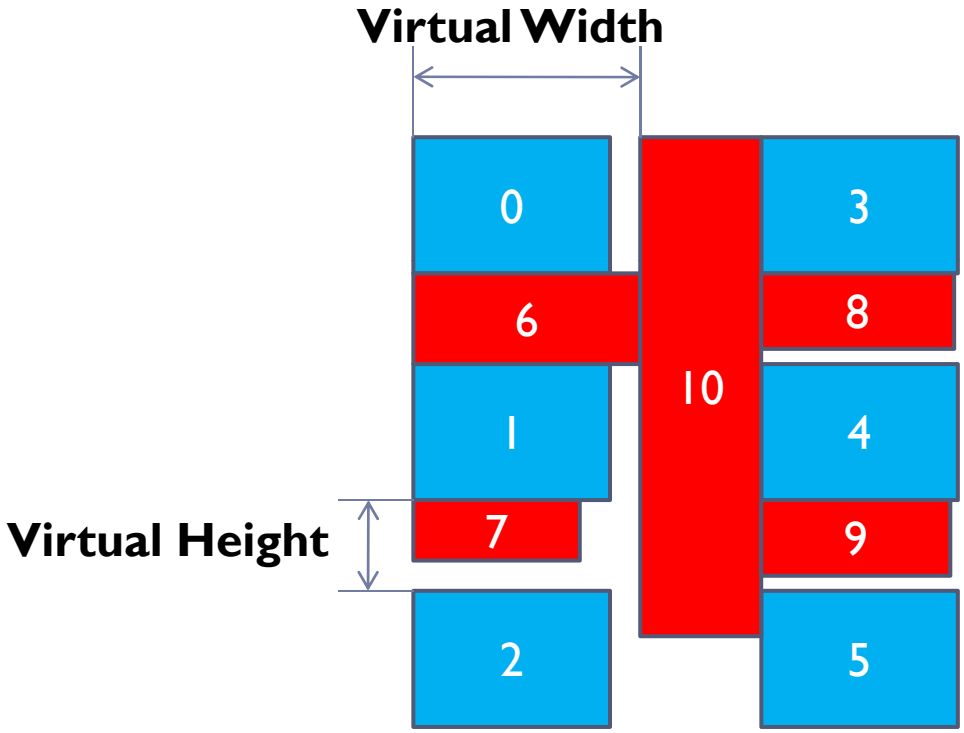


Packing Process

- If there is no non-array block inside an array, the array can be packed with longest common sequence directly
- If there is any non-array block inside an array, decided the minimum uniform spacing, then call longest common sequence and restore to original dimensions

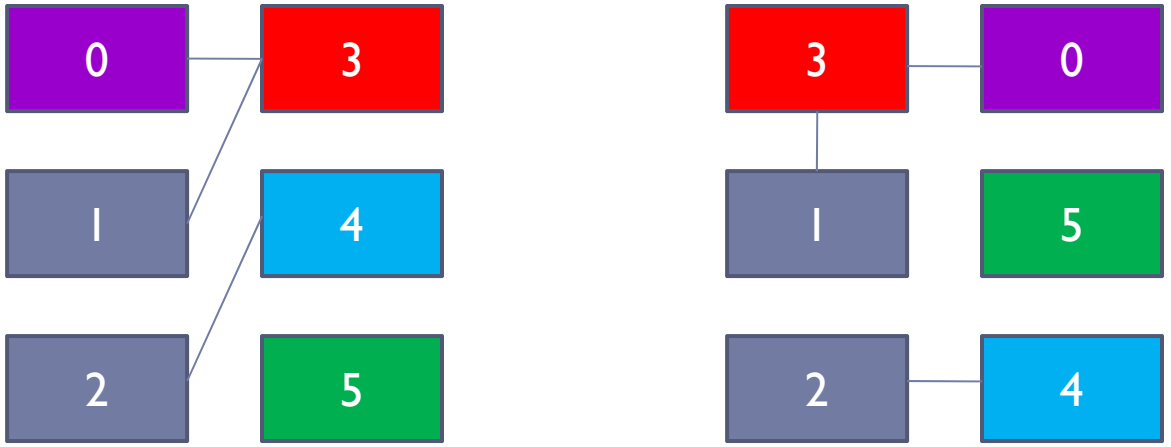
Packing Example

- Example:

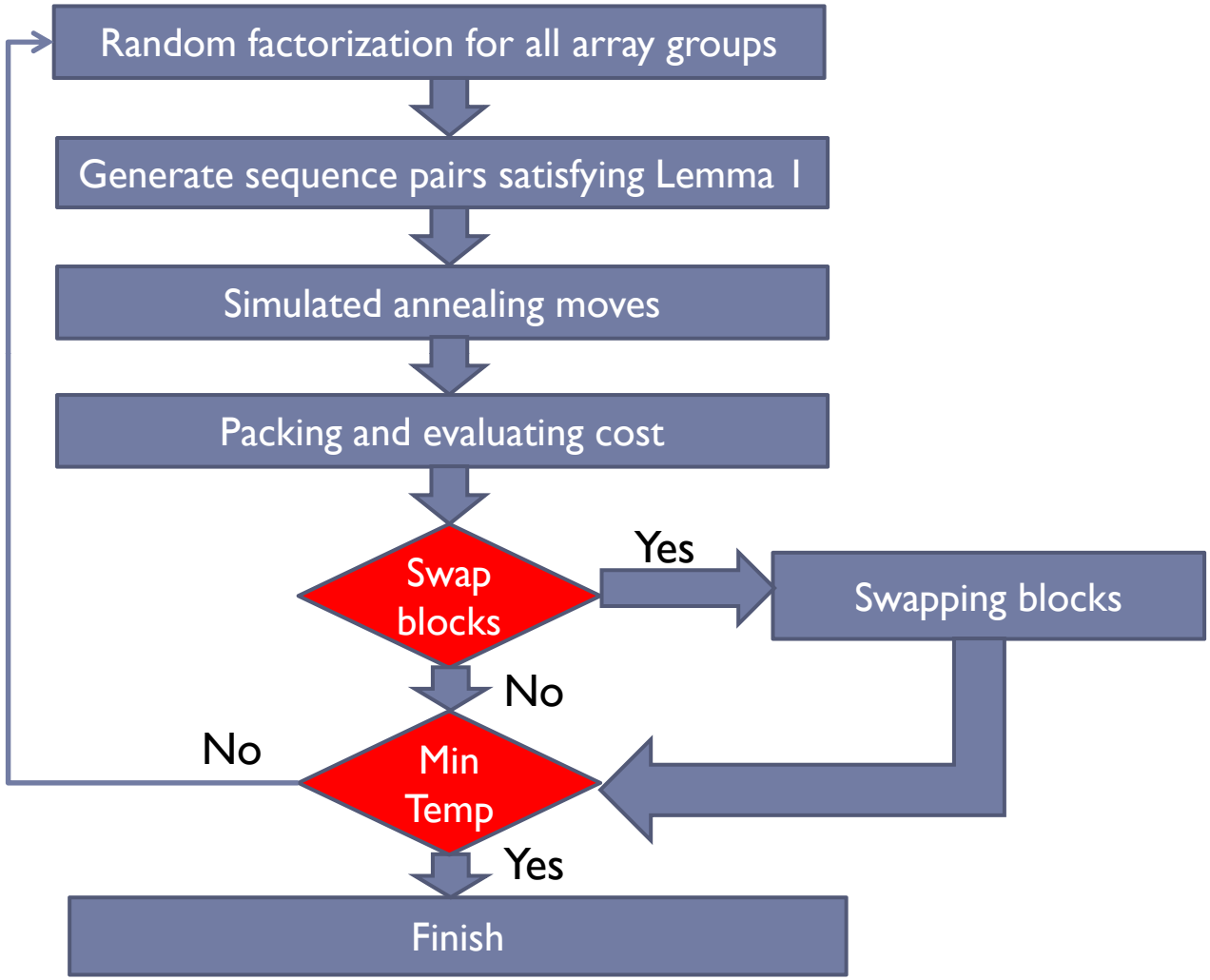


Swapping Array Blocks

- Array blocks have same dimensions
- Swapping array blocks:
  - No effect on area
  - Reduce wirelength



The Floorplanning Algorithm



Simulated Annealing Moves

- Changing the **factorization** of an array group
- Changing the **regularity sequence-pair** for an array group between row-based and column-based
- Moving a **non-array block** into (or outside) a regularity subsequence-pair
- Swapping two **non-array blocks**

Outline

- Introduction
- Floorplanning with Regularity Constraint
- Experimental Results
- Conclusions and Future Research

Experiment Setup

- Compared with a manual prefix method
- Prefix method: preplaced array blocks then run simulated annealing for non-array blocks
- Go through all prefix factorizations, pick the best to compare
- Slightly modifications to the **MCNC** and **GSRC** benchmarks
- Experiment environment:

  (1) Implemented in C++

  (2) Performed on a Windows OS

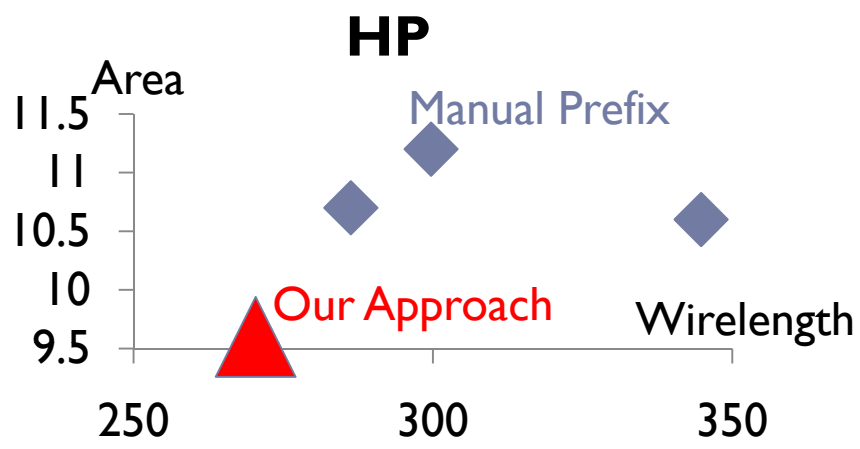
  (3) 2.5GHz Intel core 2 Duo and 2 GB memory

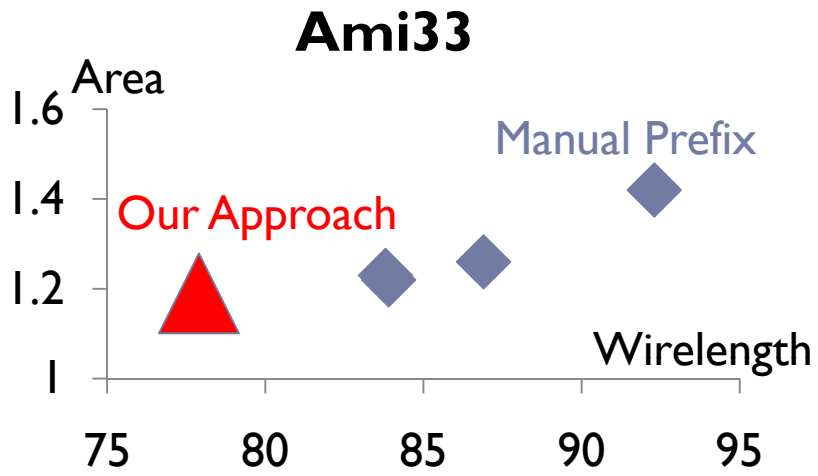
Wirelength and Area-driven Results

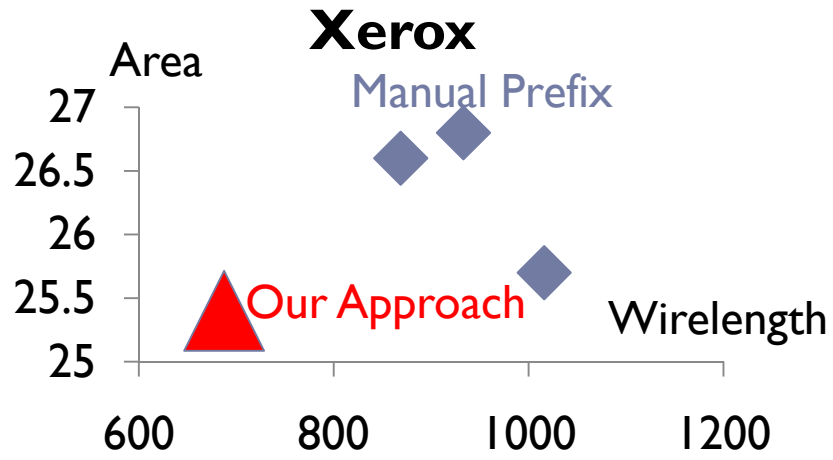
MCNC benchmark, λ=0.5. Our approach can reduce wirelength by 22% on average
Meanwhile, achieving the same or less area and mostly faster runtime

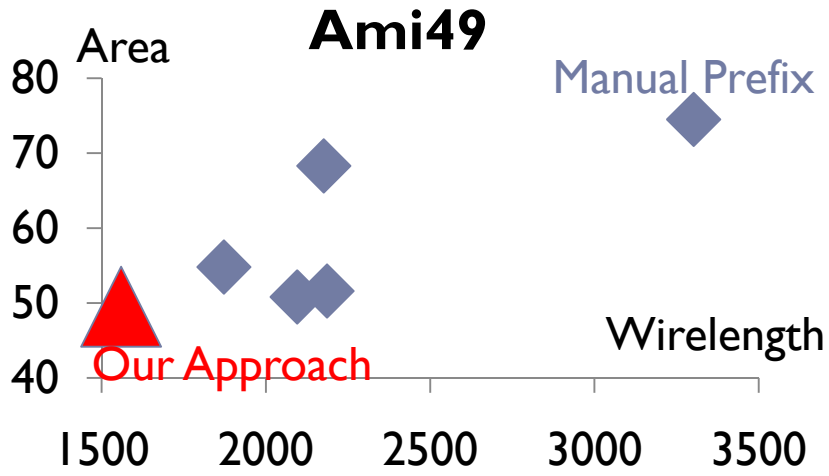
MCNC Circuit	Manual Prefix(MP)				Our Approach				
	Min cost array	Area(mm^2)	Wirelength (mm)	CPU(s)	Area(mm^2)	Area reduction vs. MP	Wirelength (mm)	Wirelength reduction vs. MP	CPU(s)
Apte	4*1	48.21	628.5	19.6	48.21	0%	472.3	24.8%	22.0
Hp	1*4	10.65	344.8	30.5	9.67	9.2%	279.4	18.9%	27.2
Xerox	1*4	25.74	1061.1	144.6	25.45	1.1%	687.5	32.3%	102.0
Ami33	4*2	1.22	83.9	525.8	1.19	2.5%	77.9	7%	474.3
Ami49	4*4	50.85	2095.3	1931.5	49.53	2.6%	1559.5	25.5%	1354.6

Area vs. Wirelength









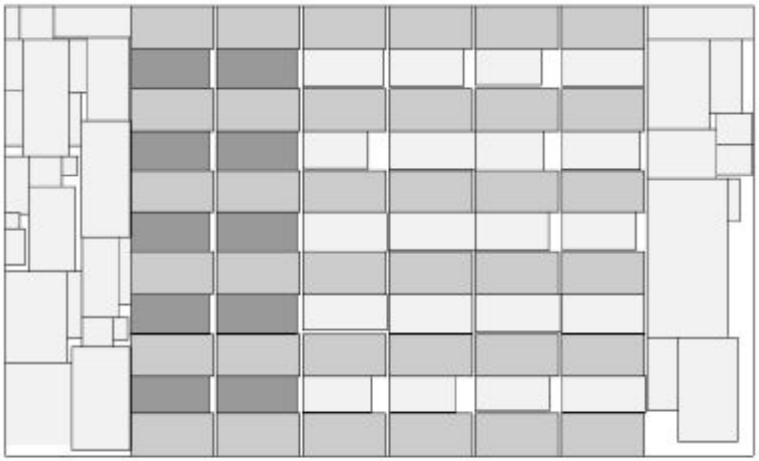
Area-driven Results

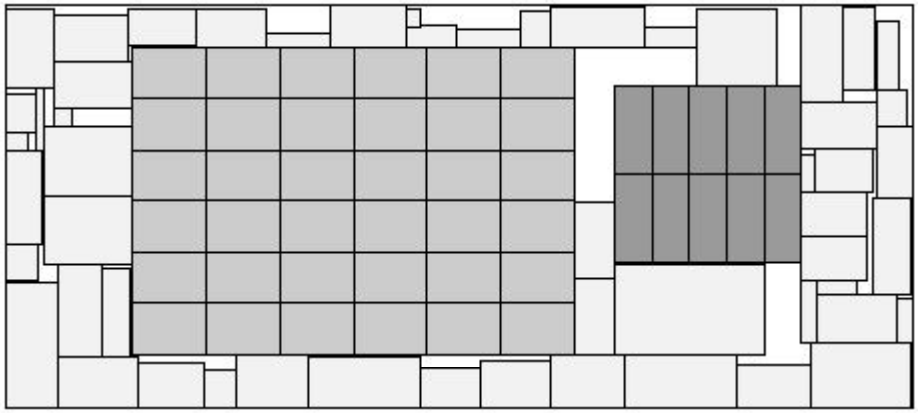
We also compared the two approaches for area-driven only formulation with GSRC benchmark

Circuit	Total No. of blocks	No. of array blocks	Manual Prefix			Our Approach	
			Min area arrays	Area Usage(%)	CPU(s)	Area Usage(%)	CPU(s)
Apte	9	4	4*1	95.56	32.52	96.56	3.20
Hp	10	4	2*2	90.63	22.59	90.64	16.41
Xerox	11	4	1*4	96.71	14.07	97.13	29.87
Ami33	33	8	2*4	94.63	379.74	95.42	331.30
Ami49	49	16	8*2	93.69	713.98	93.80	231.3
n50	50	16,12	4*4,4*3	88.06	71.367	93.05	42.89
n70	70	24,9	4*6,3*3	87.02	149.45	90.53	465.1
n100	100	36,10	6*6,2*5	90.16	461.33	92.20	259.3
n200	200	56,21	7*8,7*3	84.11	3016.45	92.89	5007.4
n300	300	81,40	9*9,10*4	86.25	5429.79	89.82	6370.9

An Example

Floorplan of n100 generated by our approach and manual prefix method





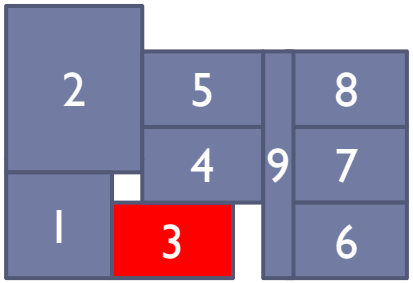
Conclusion and Future Research

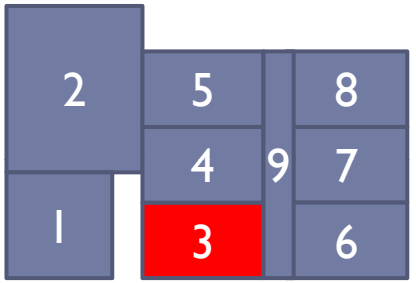
- A floorplanning approach under regularity constraint
- In future, study other representations like TCG
- Performance under fixed-outline constraint

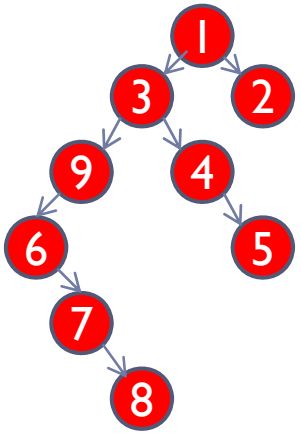
Thanks

Other Floorplan Representations

- Tree-based Representation
- Sequence Pair Representation
- TCG Representation







(215439876,123459678)