Floorplanning and Topology Generation for Application-Specific Network-on-Chip

Bei Yu[1] Sheqin Dong[1] Song Chen[2] Satoshi GOTO[2]

[1]Department of Computer Science & Technology
Tsinghua University, Beijing, China

[2]Graduate School of IPS
Waseda University, Kitakyushu, Japan

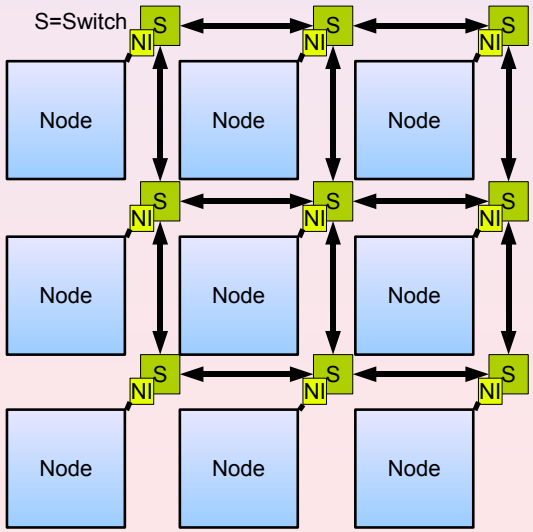
2010.01.20

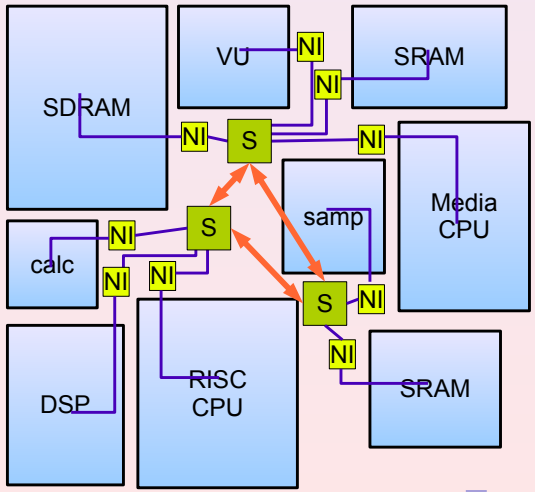
Outline

1. Introduction
   - Previous Works
   - Problem Formulation
2. Topology Synthesis Algorithm
   - Partition Driven Floorplanning
   - Switches and Network Interfaces Insertion
   - Energy Aware Path Allocation
3. Experimental Results

Network-on-Chip

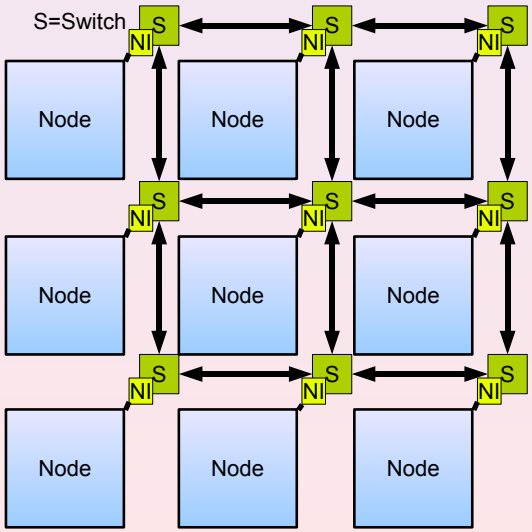
- Solution to global communication challenges
- Alternative to Bus communication architectures
  - Better modularity
  - Lower power consumption
  - Scalability
- Regular NoCs and Application-Specific NoCs
- Network components:
  - Switch
  - Network Interface (NI)

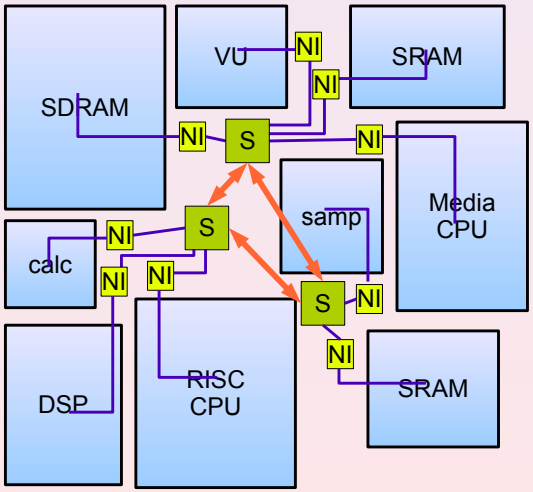




Regular or Application-Specific Topology

- Regular Topology
  - Task Scheduling and Mapping problem
- Application-Specific Topology?
  1. Irregular core sizes
  2. Different communication flow requirements
  3. Reducing energy by reducing hop count and switch count
  4. Possibly higher performance

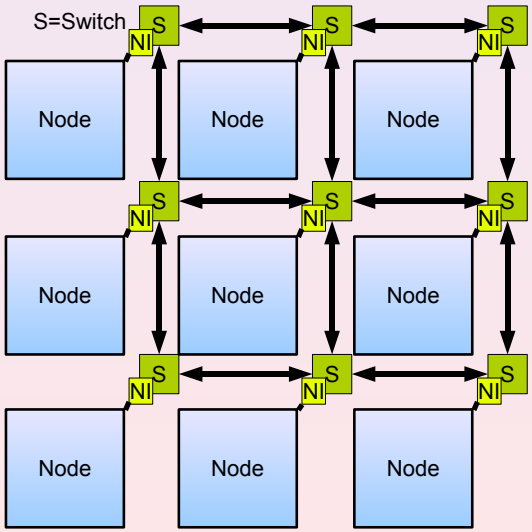


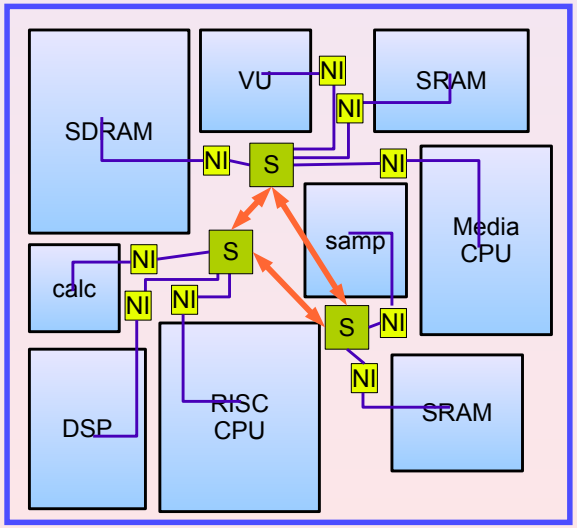


Regular or Application-Specific Topology

- Regular Topology
  - Task Scheduling and Mapping problem
- Application-Specific Topology?
  1. Irregular core sizes
  2. Different communication flow requirements
  3. Reducing energy by reducing hop count and switch count
  4. Possibly higher performance

Focus on Application-Specific Topology Generation!





Previous Works

- **K.Srinivasan et al. TVLSI 06**:
  - Used fixed floorplan as optimization starting point
  - Switch at corners of cores
- **Murali et al. ICCAD06**:
  - Two steps topology generation procedure using min-cut partitioner
  - Greedy based path allocation assignment
- **Chan & Parameswaran, ASPDAC08**:
  - Iterative refinement strategy
  - supports both packet-switched networks and point to point connections
- **Murali et al. ASPDAC09**:
  - Synthesis approach for 3D NoC
  - LP based switch position computation

Motivations

In previous works:

- Partition w/o physical information
- Fail to consider area consumption of NI and Switch

In our works:

- Integrate partition into floorplanning phase
- Consider Switches and NI area consumption
- Min-Cost-Flow algorithm to insert NI
- Effective paths allocation to minimize power consumption

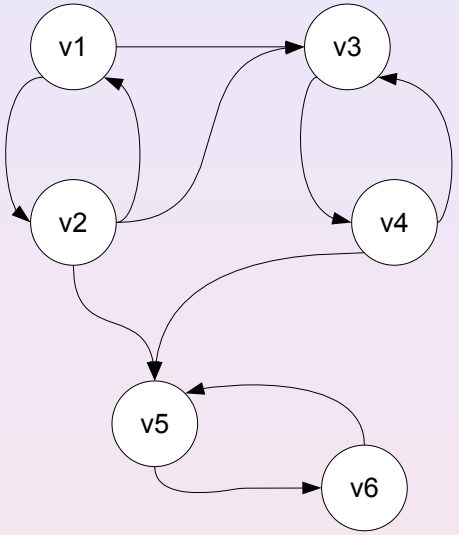
Problem Formulation

Input:

- a set of n cores $C = \{c_1, c_2, \ldots, c_n\}$.
- switches number m.
- core communication graph(CCG).
- network components power model.

Output: an NoC topology satisfying

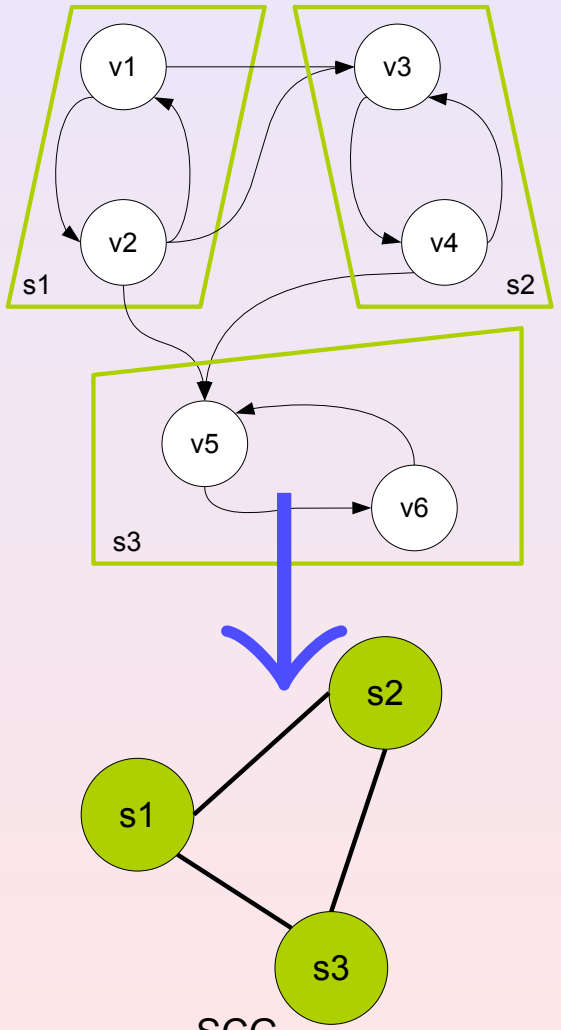
- minimize area consumption.
- minimize the communication energy.



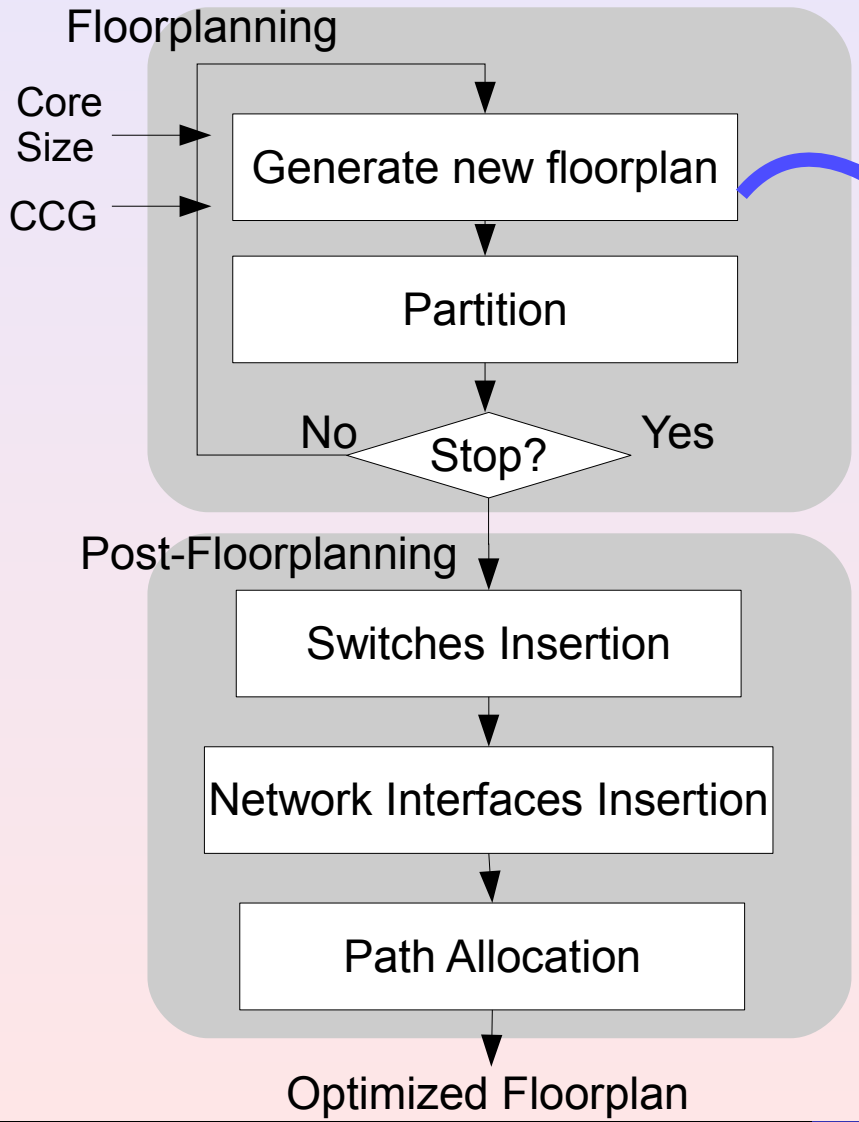
CCG: Core Communication Graph.

Synthesis Algorithm

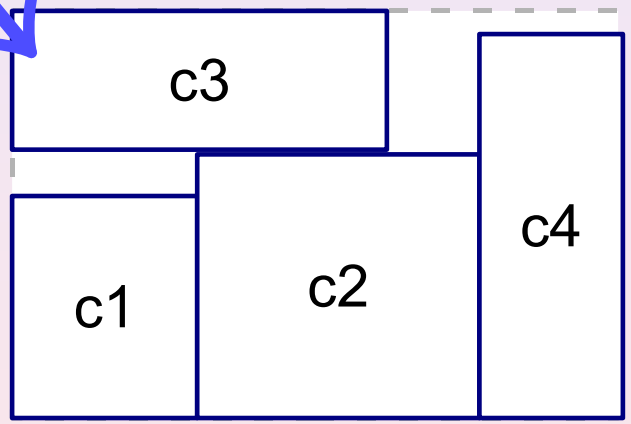
- Obtain min-cut partitions of CCG
  - Communication Requirement
  - Distances between cores
- Cores in a cluster share a switch
- Switch Communication Graph(SCG)
- Path Allocation on SCG
  - Minimize power consumption
  - Minimize hop-count
  - Satisfy width constraints



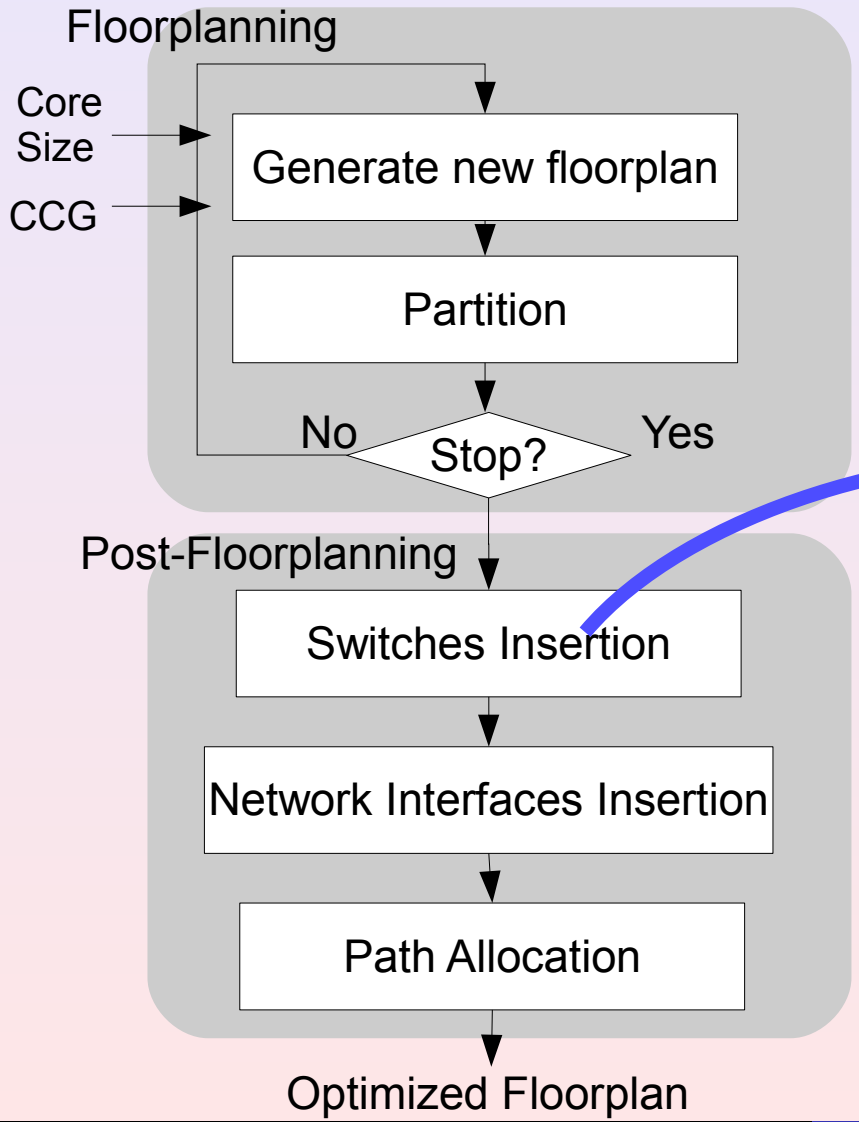
Overview of Algorithm

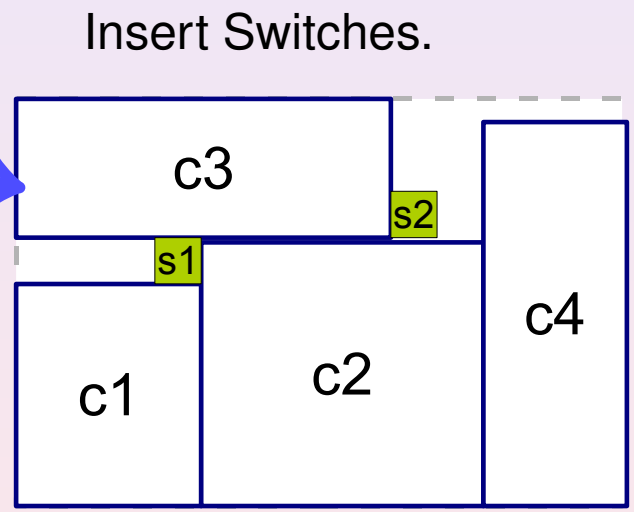


Generate floorplan with partitions.

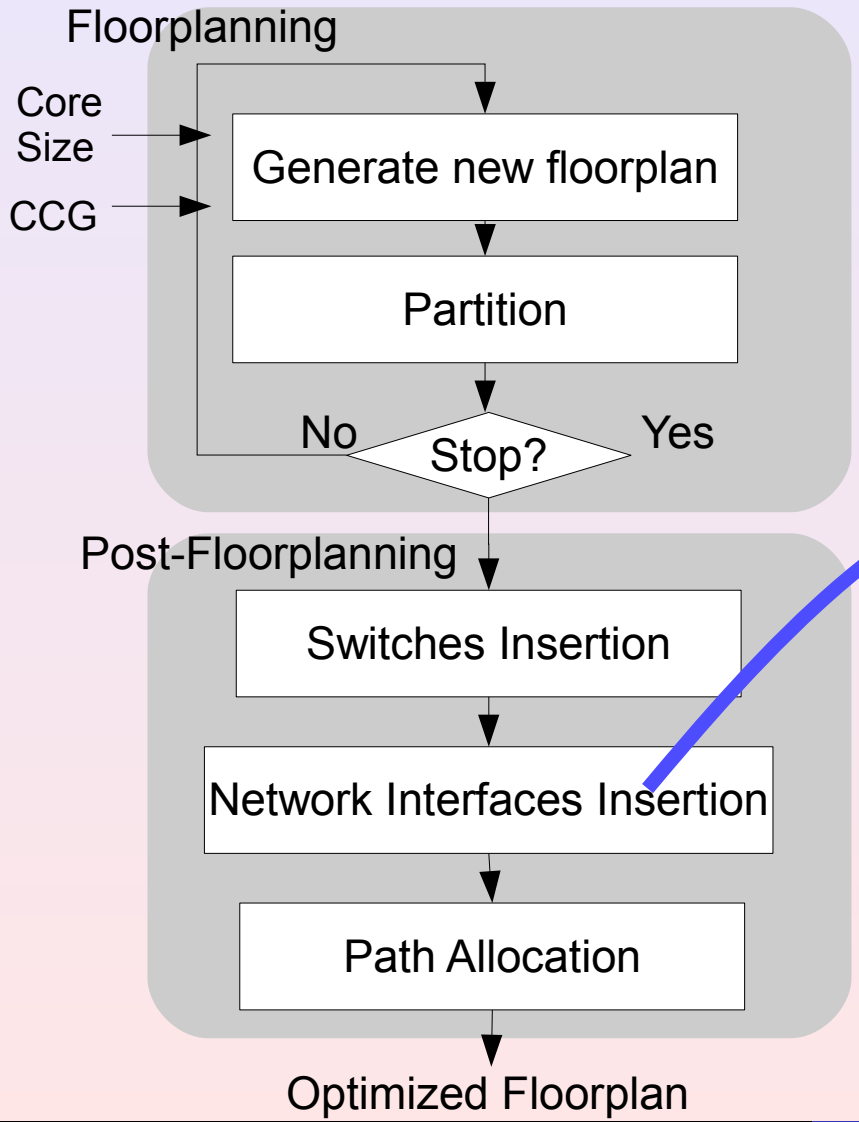


Overview of Algorithm

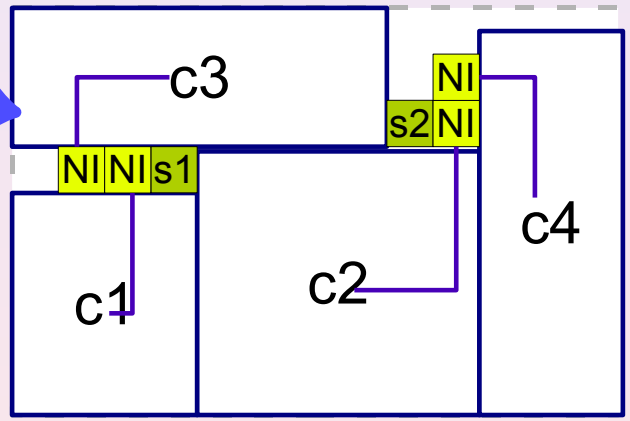




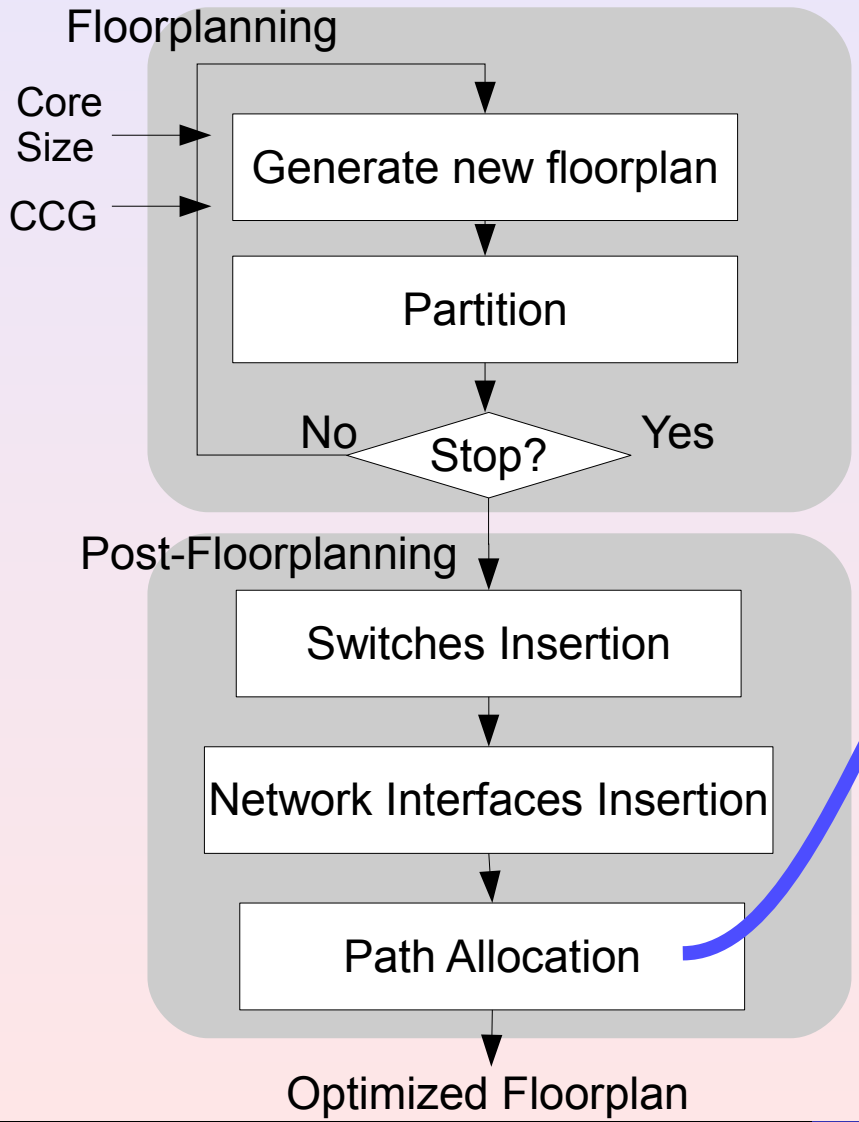
Overview of Algorithm



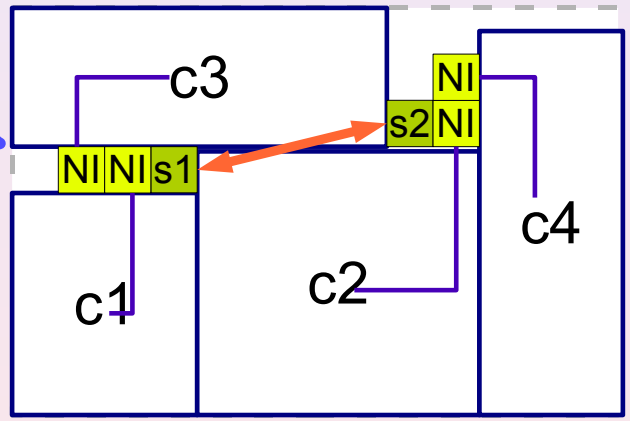
Insert NI with Min-Cost Flow Algorithm



Overview of Algorithm



Dynamic Programming based Path Allocation.



Partition Driven Floorplanning

- Traditionally, partition before floorplanning

  (-)Lose physical information
- In our work
  - Integrate partition into floorplanning
  - Cores with larger communication incline to one cluster
  - Minimize interconnect power consumption
- Define new edge weight w'_{ij} in CCG:

$$w'_{ij} = \alpha_w \times \frac{w_{ij}}{max\_w} + \alpha_d \times \frac{mean\_dis}{dis_{ij}}$$

- Using CBL[1] as topological representation
  - Record white space information

[1] X. Hong et al, *IEEE Transaction on CAS* 2004.

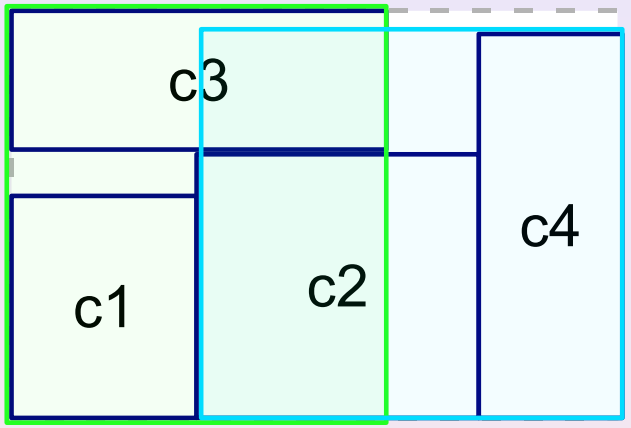
Switches Insertion

After floorplanning stage

- Each cluster has a minimal bounding box.
- Heuristical method to insert switches:
  1. Switch initially in the center of bounding box.
  2. Partition the white space into grids.
  3. Sort switches.
  4. Insert switches in grids one by one.
- In cluster p_k, cost of insert switch k to grid g:

$$Cost_{gk} = \sum_{i,j} w_{ij} \times (dis_{gi} + dis_{gj}), \forall e_{ij} \in \bar{E}$$

Choose free grid with smallest *Cost*.



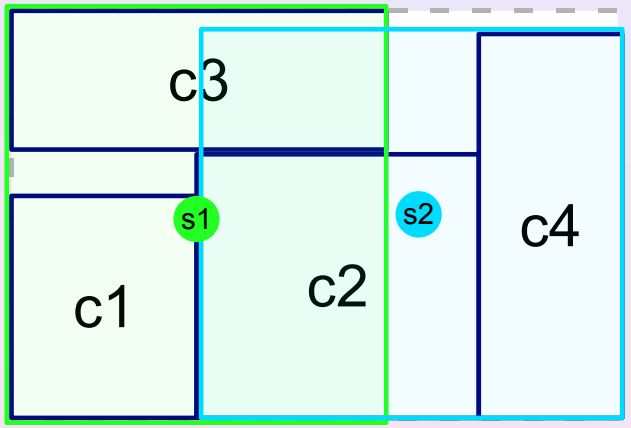
Switches Insertion

After floorplanning stage

- Each cluster has a minimal bounding box.
- Heuristical method to insert switches:
  1. Switch initially in the center of bounding box.
  2. Partition the white space into grids.
  3. Sort switches.
  4. Insert switches in grids one by one.
- In cluster p_k, cost of insert switch k to grid g:

$$Cost_{gk} = \sum_{i,j} w_{ij} \times (dis_{gi} + dis_{gj}), \forall e_{ij} \in \bar{E}$$

Choose free grid with smallest *Cost*.



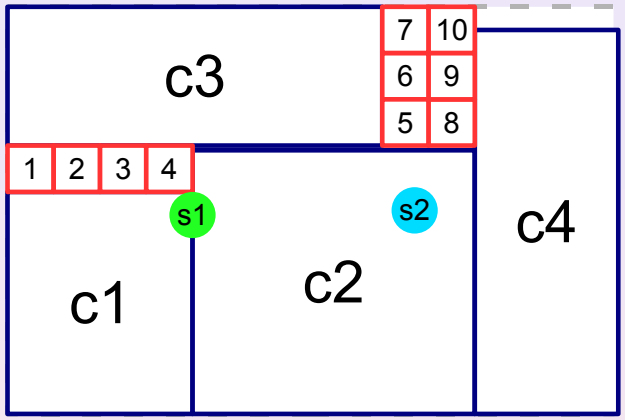
Switches Insertion

After floorplanning stage

- Each cluster has a minimal bounding box.
- Heuristical method to insert switches:
  1. Switch initially in the center of bounding box.
  2. Partition the white space into grids.
  3. Sort switches.
  4. Insert switches in grids one by one.
- In cluster p_k, cost of insert switch k to grid g:

$$Cost_{gk} = \sum_{i,j} w_{ij} \times (dis_{gi} + dis_{gj}), \forall e_{ij} \in \bar{E}$$

Choose free grid with smallest *Cost*.



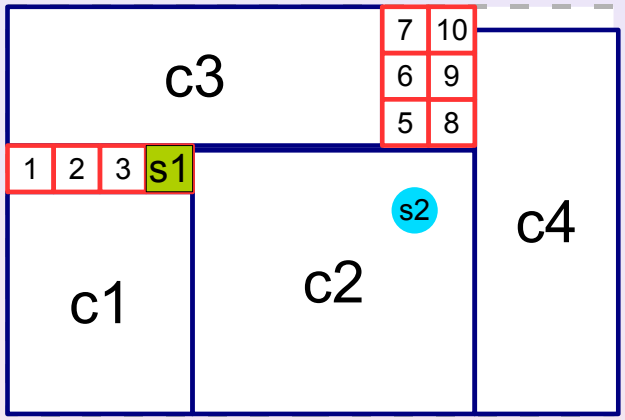
Switches Insertion

After floorplanning stage

- Each cluster has a minimal bounding box.
- Heuristical method to insert switches:
  1. Switch initially in the center of bounding box.
  2. Partition the white space into grids.
  3. Sort switches.
  4. Insert switches in grids one by one.
- In cluster p_k, cost of insert switch k to grid g:

$$Cost_{gk} = \sum_{i,j} w_{ij} \times (dis_{gi} + dis_{gj}), \forall e_{ij} \in \bar{E}$$

Choose free grid with smallest *Cost*.



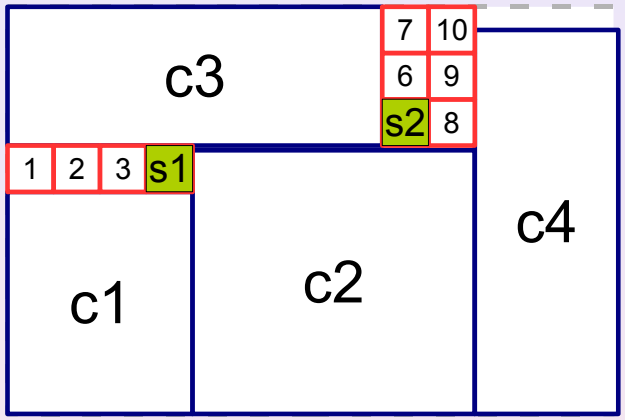
Switches Insertion

After floorplanning stage

- Each cluster has a minimal bounding box.
- Heuristical method to insert switches:
  1. Switch initially in the center of bounding box.
  2. Partition the white space into grids.
  3. Sort switches.
  4. Insert switches in grids one by one.
- In cluster p_k, cost of insert switch k to grid g:

$$Cost_{gk} = \sum_{i,j} w_{ij} \times (dis_{gi} + dis_{gj}), \forall e_{ij} \in \bar{E}$$

Choose free grid with smallest *Cost*.

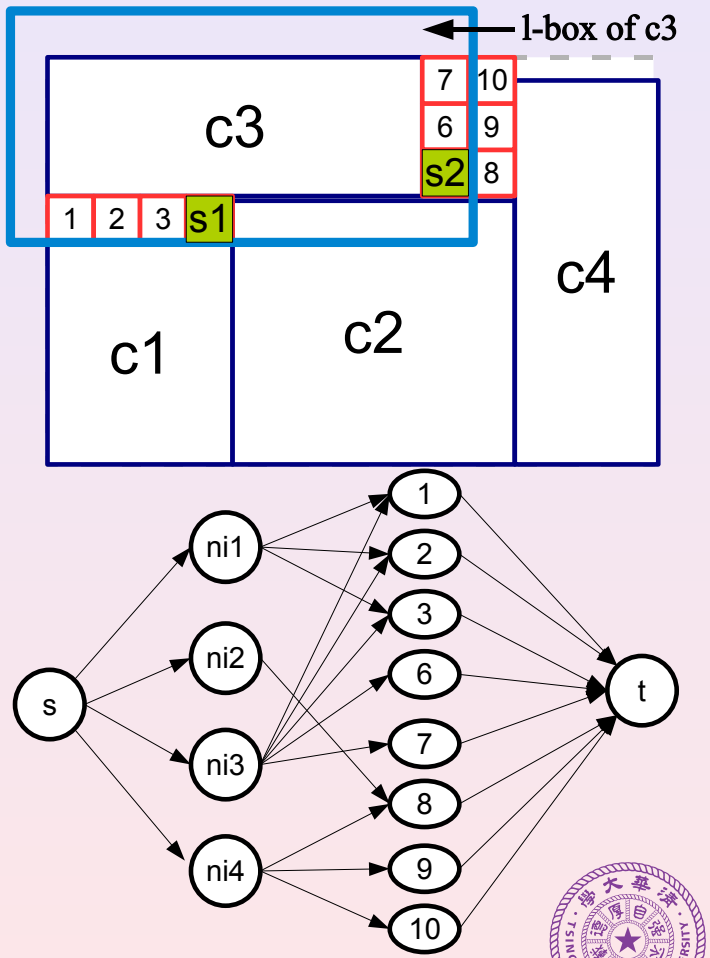


Network Interfaces Insertion

- For each core, construct l-bounding box
- Insert NI in l-bounding box
- Construct network graph $G^* = (V^*, E^*)$:

Network Graph

- $V^* = \{s, t\} \cup NI \cup Grids$.
- $E^* = \{(s, ni_k) | ni_k \in NI\} \cup \{(ni_k, g_j) | \forall g_j \in CG_k\} \cup \{(g_j, t) | g_j \in Grids\}$.
- Capacities:
  $C(s, ni_k) = 1, C(ni_k, g_j) = 1, C(r_j, t) = 1$.
- Cost:
  $F(s, ni_k) = 0, F(g_j, t) = 0; F(ni_k, g_j) = F_{kj}$.



- Min-cost flow algorithm, polynomial time.

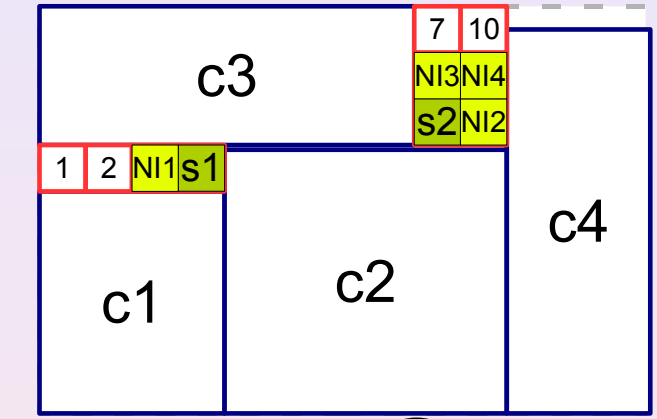
Network Interfaces Insertion

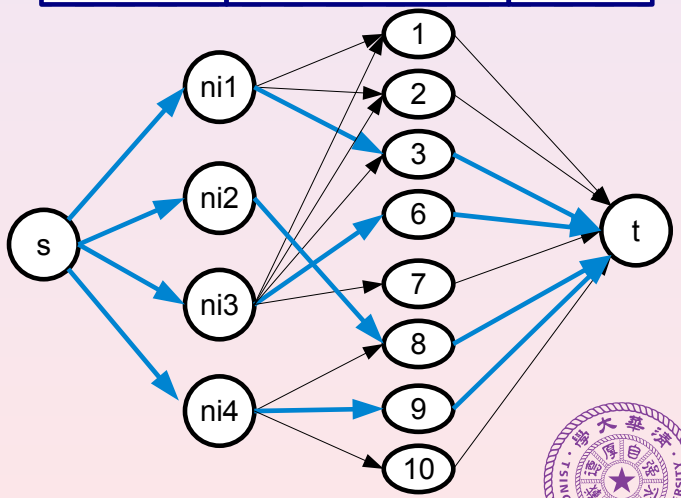
- For each core, construct l-bounding box
- Insert NI in l-bounding box
- Construct network graph $G^* = (V^*, E^*)$:

Network Graph

- $V^* = \{s, t\} \cup NI \cup Grids$.
- $E^* = \{(s, ni_k) | ni_k \in NI\} \cup \{(ni_k, g_j) | \forall g_j \in CG_k\} \cup \{(g_j, t) | g_j \in Grids\}$.
- Capacities:
  $C(s, ni_k) = 1, C(ni_k, g_j) = 1, C(r_j, t) = 1$.
- Cost:
  $F(s, ni_k) = 0, F(g_j, t) = 0; F(ni_k, g_j) = F_{kj}$.

- Min-cost flow algorithm, polynomial time.





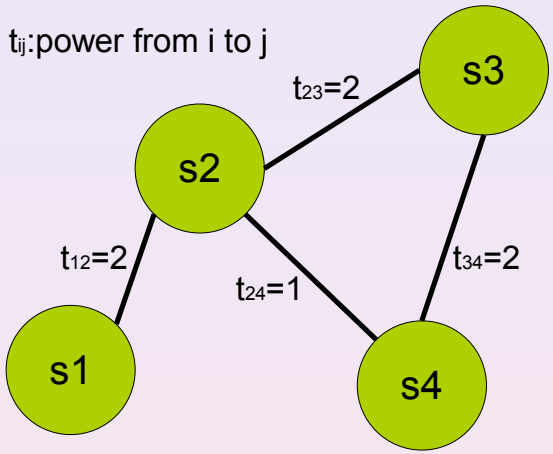
Energy Aware Path Allocation

Solve once is enough? **No**!

- Consider Power Consumption.
- Path with minimal power consumption may change.

Simple example:

- Two flows: $(s1 \rightarrow s3), (s2 \rightarrow s3)$.
- Solve $(s1 \rightarrow s3)$ first.
- First,
  - shortest path from $s2$ to $s3$ is $s1 \rightarrow s3$.
- After flow $(s1 \rightarrow s3)$:
  - shortest path from $s2$ to $s3$ is $s1 \rightarrow s4 \rightarrow s3$.



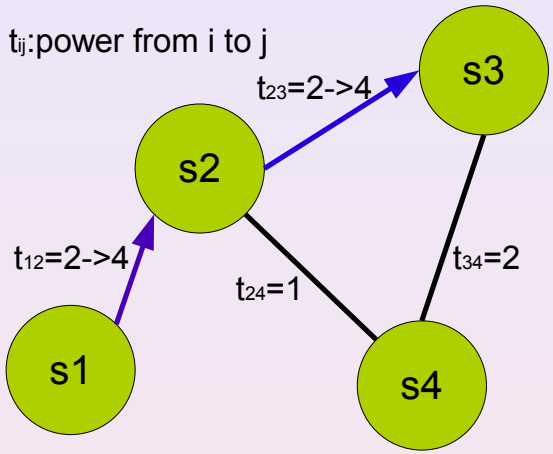
Energy Aware Path Allocation

Solve once is enough? **No**!

- Consider Power Consumption.
- Path with minimal power consumption may change.

Simple example:

- Two flows: $(s1 \rightarrow s3), (s2 \rightarrow s3)$.
- Solve $(s1 \rightarrow s3)$ first.
- First,
  - shortest path from $s2$ to $s3$ is $s1 \rightarrow s3$.
- After flow $(s1 \rightarrow s3)$:
  - shortest path from $s2$ to $s3$ is $s1 \rightarrow s4 \rightarrow s3$.



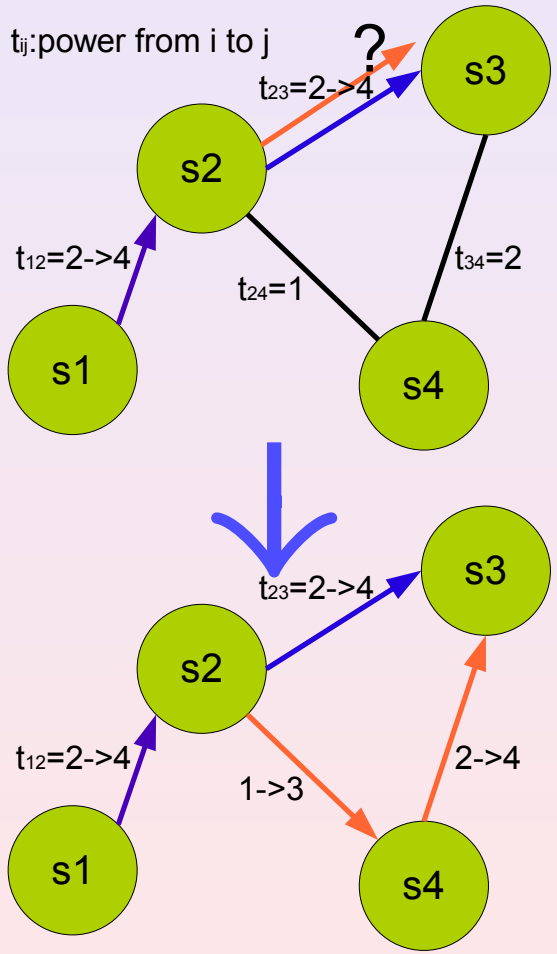
Energy Aware Path Allocation

Solve once is enough? **No**!

- Consider Power Consumption.
- Path with minimal power consumption may change.

Simple example:

- Two flows: $(s1 \rightarrow s3), (s2 \rightarrow s3)$.
- Solve $(s1 \rightarrow s3)$ first.
- First,
  - shortest path from $s2$ to $s3$ is $s1 \rightarrow s3$.
- After flow $(s1 \rightarrow s3)$:
  - shortest path from $s2$ to $s3$ is $s1 \rightarrow s4 \rightarrow s3$.



Energy Aware Path Allocation

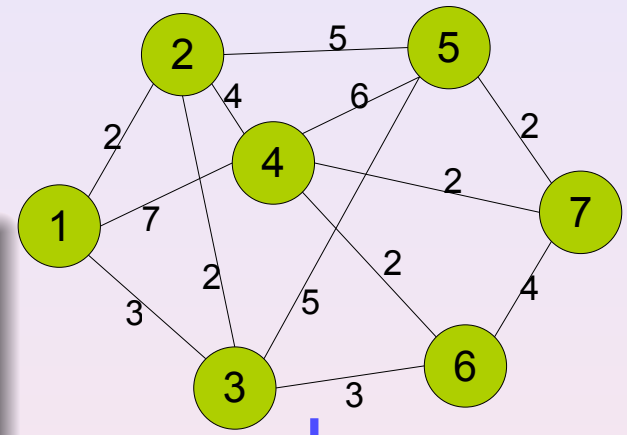
- $dis_n(i, d)$: distance from node i to d
- $dis_e(i, j, d)$: distance i to d using e_{ij}

DP based method to find paths:

$$dis_e(i, j, d) = \begin{cases} t_{id}, & j = d \\ t_{ij} + dis_n(j, d), & otherwise \end{cases}$$

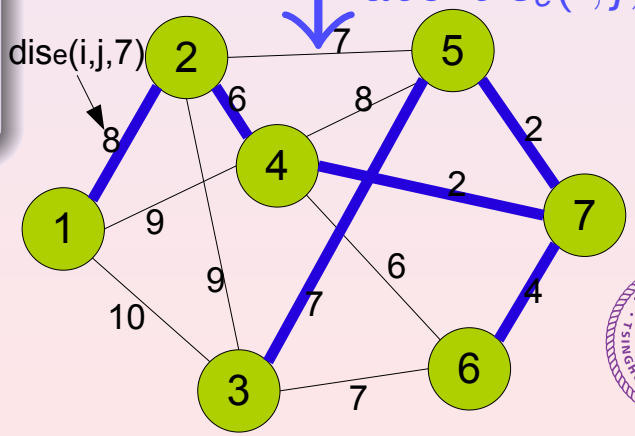
$$dis_n(i, d) = \begin{cases} 0, & i = d \\ min_k\ dis_e(i, k, d), & otherwise \end{cases}$$

- run time is bounded by $O(|V| \cdot |E|)$
- if $dis_e(i, j, d) = dis_n(i, d)$, then $path(i, d) = j$.



Find initial paths.

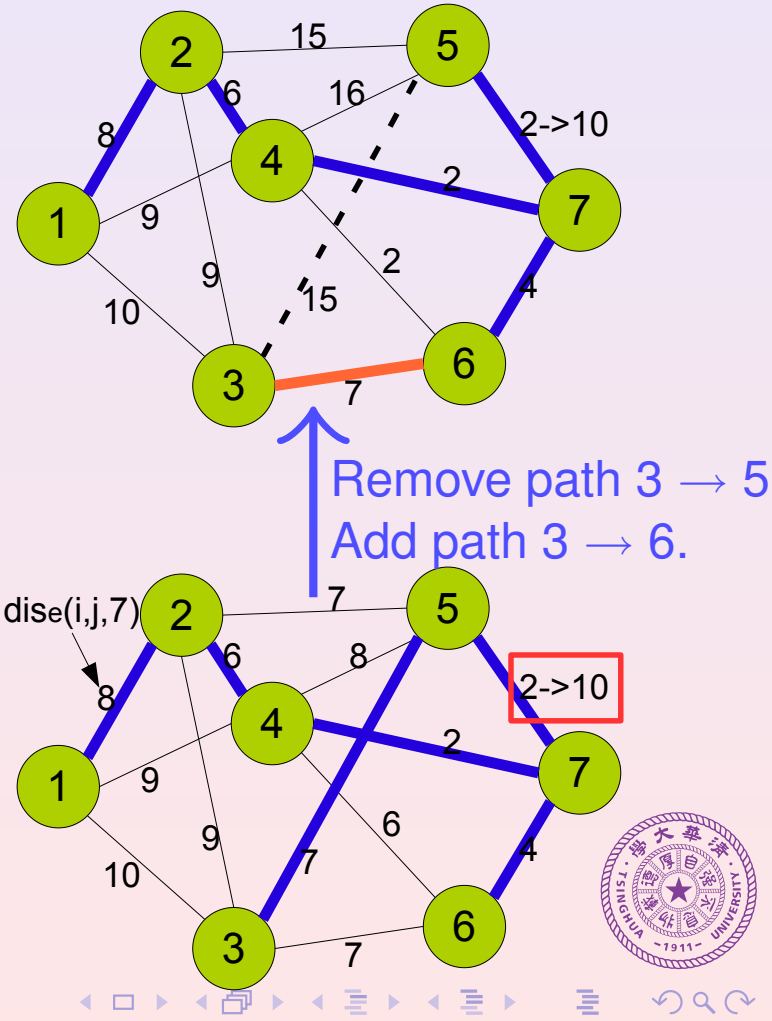
Label $dis_e(i, j, 7)$.



Update Paths

1: //Update when t_{ij} change to $(t_{ij} + \Delta t)$;
2: $t_{ij} \leftarrow (t_{ij} + \Delta t)$;
3: queue q.push(e_{ij});
4: **while** q is not empty **do**
5: $e_{ab} \leftarrow$ q.pop();
6: $dis_e(a, b, d) \leftarrow t_{ab} + dis_n(b, d)$;
7: **if** $PATH[a][d] = b$ **then**
8: Find $k \in Post(a)^a$ to minimize $dis_n(k, d) + t_{ak}$;
9: $dis_n(a, d) \leftarrow dis_n(k, d) + t_{ak}$;
10: $path(a, d) \leftarrow k$;
11: q.push(e_{pa}), $\forall p \in Pre(a)^b$;
12: **end if**
13: **end while**

$^a Post(a) = \{v_k | \forall v_k \in V \ \& \ e_{ak} \in E\}$
$^b Pre(a) = \{v_k | \forall v_k \in V \ \& \ e_{ka} \in E\}$



Experimental Setup

Power Model:

- Switch power model

ports	2	3	4	5	6	7	8
(pJ/bit)	0.22	0.33	0.44	0.55	0.66	0.78	0.90

- Interconnect power model

length(mm)	1	4	8	12	16
(pJ/bit)	0.6	2.4	4.8	7.2	9.6

Benchmark:

- Bertozzi et al. (G1, G2, G3)
- Srinivasan et al. TVLSI06 (G4, G5, G6)
- Murali et al. ASPDAC09 (G7)

	Benchmark	V#	E#
G1	MPEG4	12	13
G2	MWD	12	12
G3	VOPD	12	14
G4	263decmp3dec	14	15
G5	263encmp3dec	12	12
G6	mp3encmp3dec	13	13
G7	D_38_tvopd	38	47

Experimental Results

The Consumption Between the PBF and the PDF:

	Part#	Power(mW)		Hops		W.S(%)		Time(s)
		PBF	ours	PBF	ours	PBF	ours	ours
G1	3	25.9	16.0	1.17	1.0	12.25	16.43	13.86
	4	24.3	14.1	1.25	1.041	7.63	16.43	15.07
G2	3	3.05	3.08	1.33	1.33	12.22	11.82	13.37
	4	3.19	3.02	1.25	1.25	12.22	12.22	15.46
G3	3	7.43	6.12	1.0	1.0	12.16	13.54	14.54
	4	7.62	6.59	1.0	1.15	12.17	13.85	17.32
G4	3	4.96	3.92	1.0	1.0	14.24	13.44	23.78
	4	7.86	4.35	1.25	1.0	13.59	14.50	24.96
G5	3	24.7	19.2	1.0	1.0	6.06	8.82	13.19
	4	58.6	19.2	1.0	1.0	9.58	9.58	15.42
G6	3	8.4	4.4	1.0	1.0	15.23	17.60	20.29
	4	11.2	8.6	1.0	1.0	15.23	15.24	21.0
G7	3	12.7	8.2	1.33	1.33	15.1	24.5	92.7
	4	12.3	6.8	1.44	1.4	14.7	22.60	104.0
Avg	-	15.16	8.83	1.14	1.11	12.31	13.92	28.93
Diff	-	-	-41.8%	-	-2.6%	-	-	-

- PBF: similar to Murali ICCAD06, **P**artition **B**efore **F**loorplanning.
  PDF: our methods, **P**artition **D**riven **F**loorplanning.
- Can save 41.8% of power and 2.6% of hops number.

Experimental Results

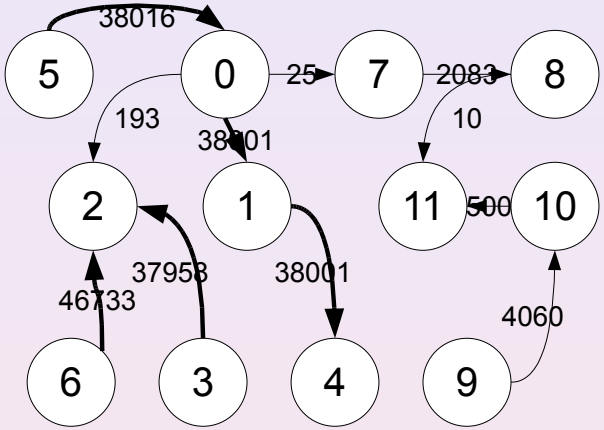
The Consumption Between the PBF and the PDF:

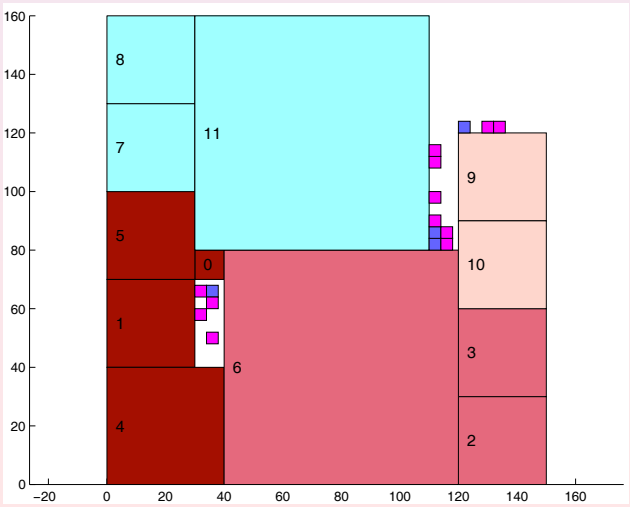
	Part#	Power(mW)		Hops		W.S(%)		Time(s)
		PBF	ours	PBF	ours	PBF	ours	ours
G1	3	25.9	16.0	1.17	1.0	12.25	16.43	13.86
	4	24.3	14.1	1.25	1.041	7.63	16.43	15.07
G2	3	3.05	3.08	1.33	1.33	12.22	11.82	13.37
	4	3.19	3.02	1.25	1.25	12.22	12.22	15.46
G3	3	7.43	6.12	1.0	1.0	12.16	13.54	14.54
	4	7.62	6.59	1.0	1.15	12.17	13.85	17.32
G4	3	4.96	3.92	1.0	1.0	14.24	13.44	23.78
	4	7.86	4.35	1.25	1.0	13.59	14.50	24.96
G5	3	24.7	19.2	1.0	1.0	6.06	8.82	13.19
	4	58.6	19.2	1.0	1.0	9.58	9.58	15.42
G6	3	8.4	4.4	1.0	1.0	15.23	17.60	20.29
	4	11.2	8.6	1.0	1.0	15.23	15.24	21.0
G7	3	12.7	8.2	1.33	1.33	15.1	24.5	92.7
	4	12.3	6.8	1.44	1.4	14.7	22.60	104.0
Avg	-	15.16	8.83	1.14	1.11	12.31	13.92	28.93
Diff	-	-	-41.8%	-	-2.6%	-	-	-

- PBF: similar to Murali ICCAD06, **P**artition **B**efore **F**loorplanning.
  PDF: our methods, **P**artition **D**riven **F**loorplanning.
- Can save 41.8% of power and 2.6% of hops number.

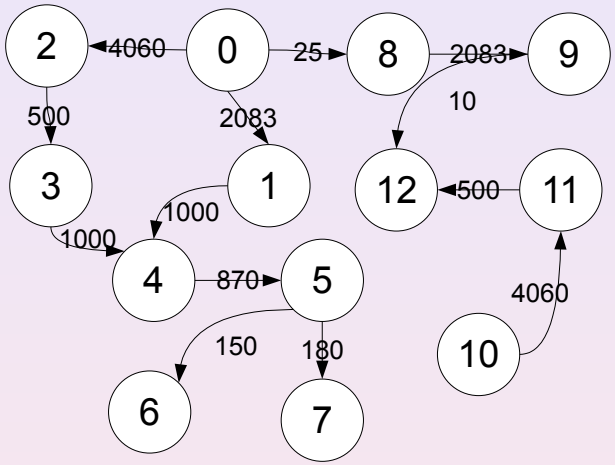
Experimental Results(cont.)

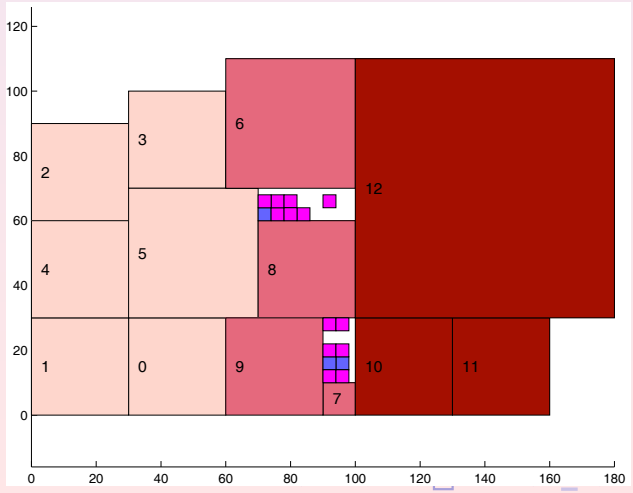
263encmp3dec (4 clusters):





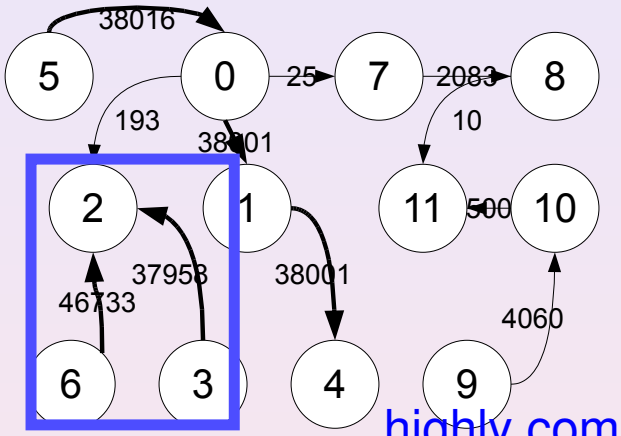
mp3encmp3dec (3 clusters):



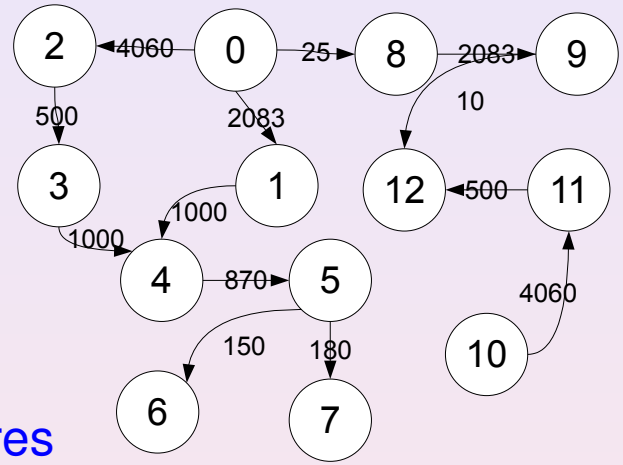


Experimental Results(cont.)

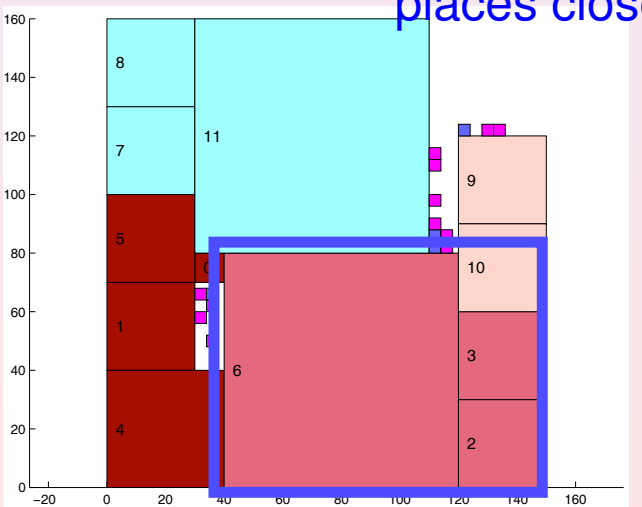
263encmp3dec (4 clusters):

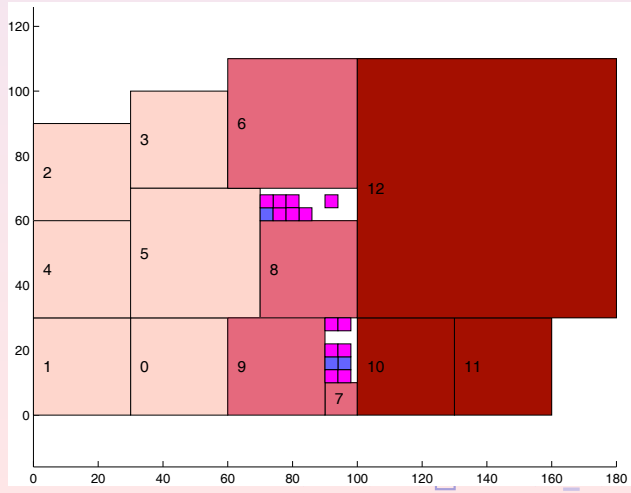


mp3encmp3dec (3 clusters):



highly communicating cores places close to each other.





Experimental Results(cont.)

- Effectiveness of Path Update Algorithm:

	V#	Flow#	Update#	Run Time(s)		Diff
				DSP	ours	
t_01	20	34	20	0.024	0.008	-66.7%
t_02	100	130	30	0.604	0.016	-97.4%
t_03	300	457	50	20.35	0.08	-99.6%

- DSP: re-solves all distances by **D**ijkstra's **S**hortest **P**ath Algorithm.
  Ours: effective path update algorithm.
- Larger graph, more effective.

Conclusion

In our works:

- Intgrate partition into floorplanning phase
- Consider Switches and NI area consumption
- Min-Cost-Flow algorithm to insert NI
- Effective paths allocation to minimize power consumption

Thank You !