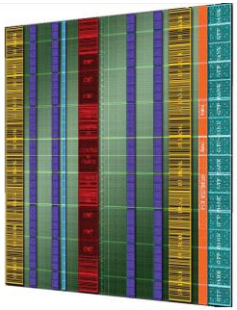
Graph-Morphing: Exploiting Hidden Parallelism of Non-Stencil Computation in HLS

Mingjie Lin @ ASAP2019

FPGA is a Great Computing Platform!

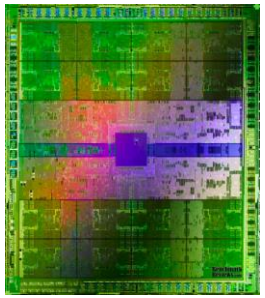
Expose “metal” to compiler

- Explicitly manage PEs
- Explicitly manage memory



Virtex 6

2.6B Transistor



GPU Fermi

3.0B Transistor

Raw memory bandwidth

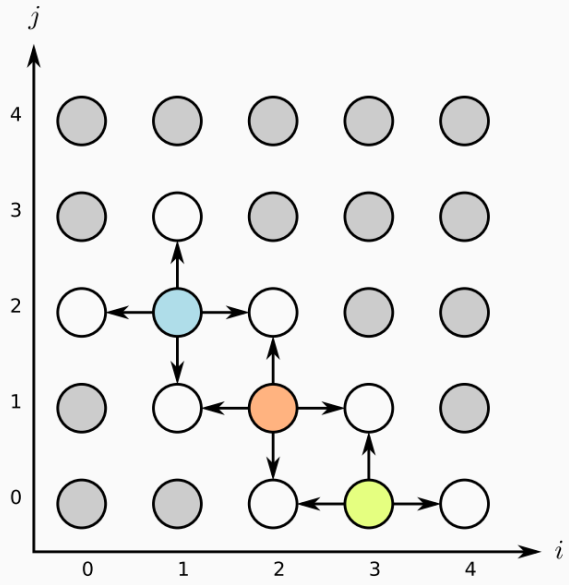
Device	Ports	Power	Capacity	Agg. Bandwidth
XC6V475T	1064	10s W	4.68 MB	**5.22 TB/s**
NVIDIA Fermi	~16	100s W	**many GBs**	~0.23 TB/s

Stencil Kernel

1) Already well studied in loop optimization.

2) Regular and determined data dependency.

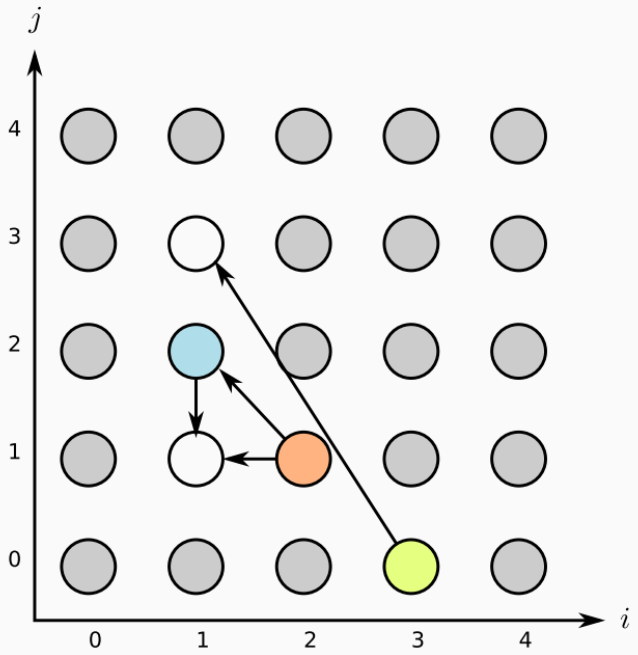
```
for(int i=1; i<3; i++)//outer loop
    for(int j=1; j<3; j++)//inner loop
        A[i][j]=A[i][j+1]+A[i][j-1]
                +A[i+1][j]+A[i-1][j])
```



Non-Stencil Kernel

- Irregular intra-loop and inter-loop memory dependencies.
- Commonly used in scientific codes.

```
for(int i=1; i<3; i++)//outer loop
    for(int j=1; j<3; j++)//inner loop
        A[i][j]=A[j][3*i-j]+A[2*i][j]
```



Challenges of Non-Stencil Kernel

- Intra-loop memory reference offsets are not static and iteration-dependent.
  - Memory banking becomes complicated.
- Inter-loop data dependencies are non-uniform across iterations.
  - Loop-carried dependencies are non-uniform.
  - Loop pipelining cannot analyze loop-carried dependencies at compile time, thus optimizing conservatively.
  - Loop unrolling is constrained since the compiler cannot determine which iterations can be parallelized without data conflicts.

Existing Approaches

- Loop splitting: pipeline the loop with non-uniform dependencies by splitting it at the iteration where RAW dependency cannot be guaranteed.

```
//Original loop with non-uniform
//dependences
for (i = 0; i < N; i++)
  A[2*i] = A[i] + 0.5f;
```

```
//Loop splitting
for (i = 0; i <= 1; i++)
  A[2*i] = A[i] + 0.5f;
for (k = 2; k <= 14; k += k)
  for (i = k; i <= min(14, 2*k-1); i++)
    A[2*i] = A[i] + 0.5f;
for (i = 15; i < N; i++)
  A[2*i] = A[i] + 0.5f
```

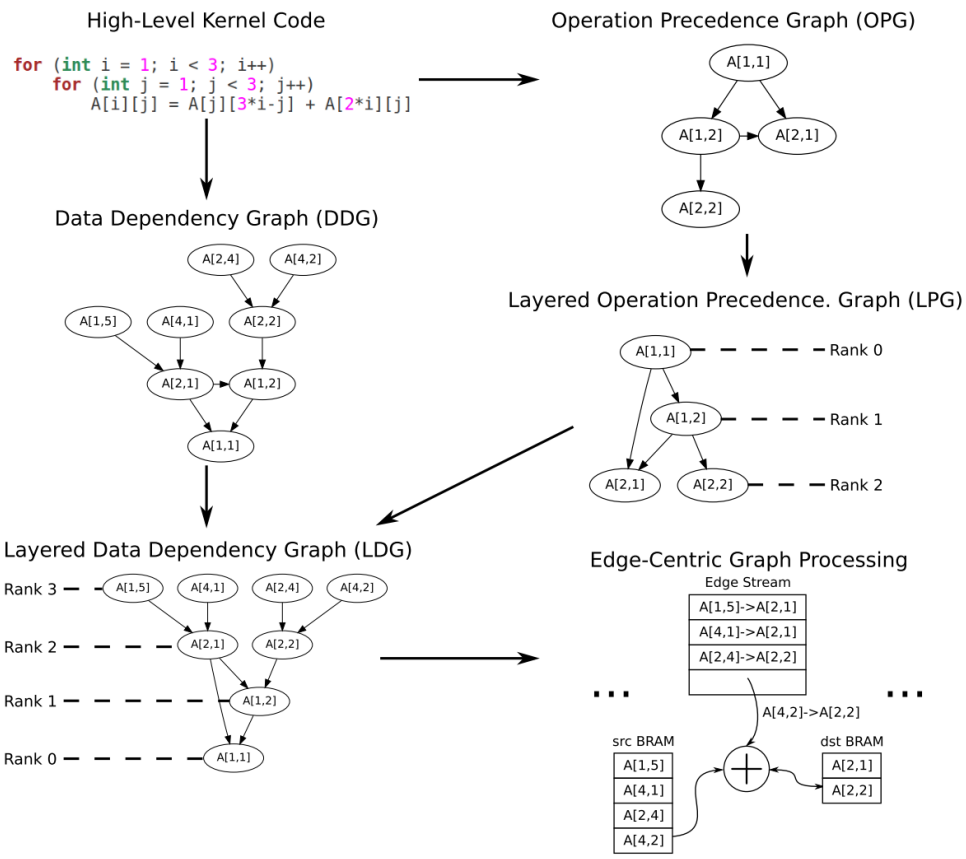
- Memory banking and data reuse: allocate data to multiple memory banks based on graph coloring.
  - Generate a conflict graph according to the kernel.
  - Map data to different banks by using a weighted graph coloring.

Research Objective

- How to exploit computation parallelism as well as data parallelism?
- Explored another optimization to accelerate non-stencil kernel computing, focusing on loop unrolling instead of loop pipelining.
- A systematic and automatic way to construct graphs from a high-level non-stencil kernel code.
- A workflow to regroup and reschedule data and computation by operating on the constructed graphs in graph theory domain.

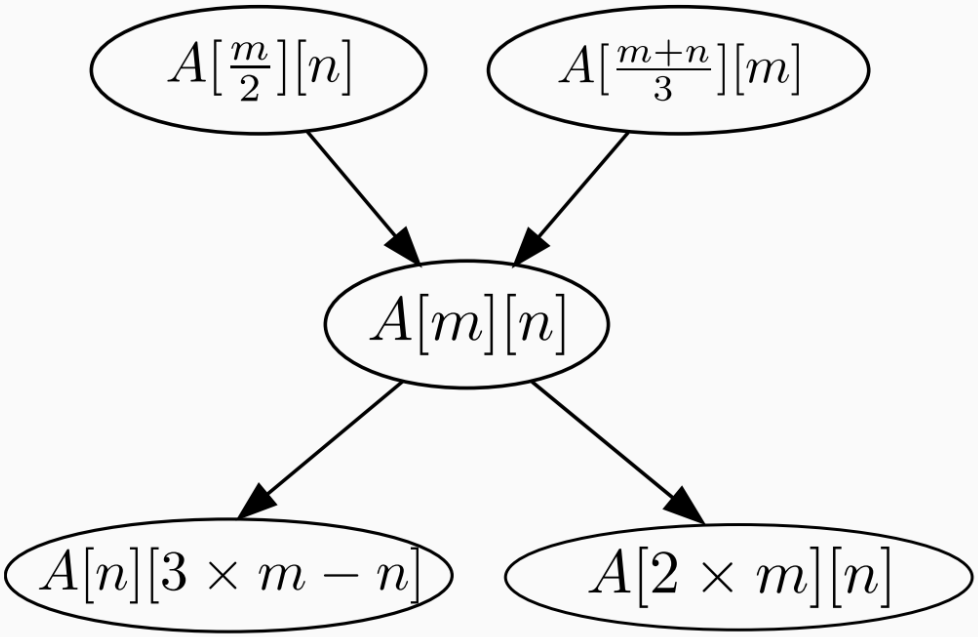
Key Idea of Graph Morphing

- A kernel is transformed into a graph.
- Nodes are memory locations, incoming edges are inputs to a iteration, outcoming edges are data outputs to other iterations.
- The graph is then partitioned and processed using an off-the-shelf graph processing engine.



Operation Precedence Graph Construction

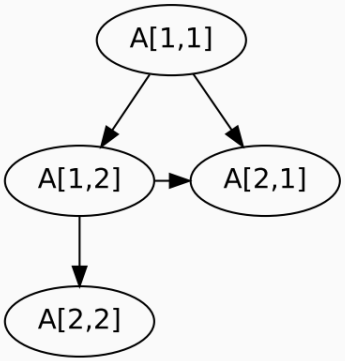
- Specifically, for the example kernel, a data-conflict graph is constructed over each iteration to describe RAW, WAR, and WAW constraints.
- A complete data-conflict graph is constructed by traversing over all iterations.



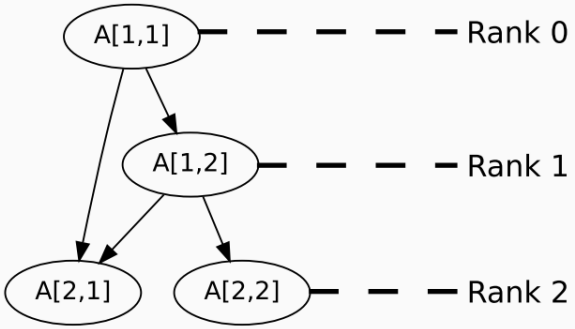
Layered Operation Precedence Graph Construction

- Reorganize iterations by putting the iterations without data conflicts in the same group, named **Ranks**.
- Obey the topologically sorted order. For an edge, source node is placed in a higher-priority rank than sink node.
- Bellman-ford single-source-shortest-path (SSSP) is used to assign ranks to nodes, by setting each edge's weight to -1.

Operation Precedence Graph (OPG)



Layered Operation Precedence. Graph (LPG)



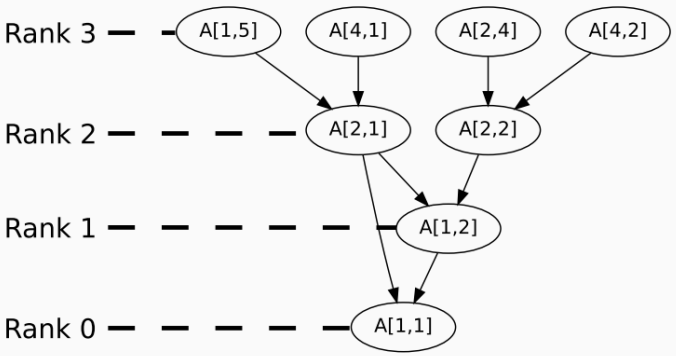
Layered Data Dependency Graph Construction

- In LPG, only LHS memory references are included as nodes.
- RHS memory references are instantiated as nodes and added to the graph, forming a layered data dependency graph (LDG).
- Now we have end-to-end transformed a high-level kernel code, into a layered graph.
  - Layer (Rank) means **computation order**.
  - Each node is an involved memory reference.
  - Each edge is a data dependency (dataflow).

High-Level Kernel Code

```
for (int i = 1; i < 3; i++)
    for (int j = 1; j < 3; j++)
        A[i][j] = A[j][3*i-j] + A[2*i][j]
```

Layered Data Dependency Graph (LDG)

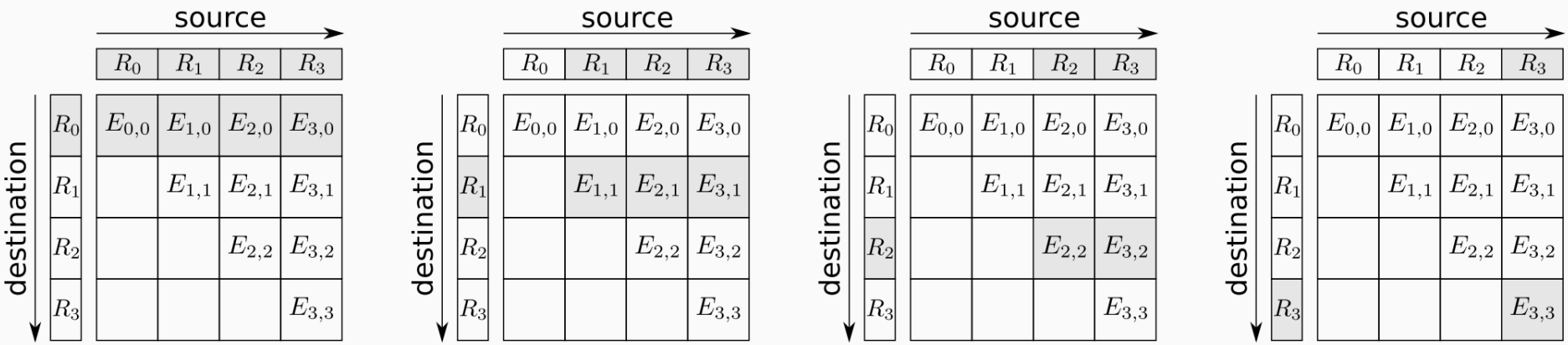


Edge-Centric Graph Processing

- We adopted the edge-centric and stream-and-apply method to process the constructed graph.
- Nodes are divided into N ranks, R_i, $i = 0 \ldots N$.
- Edges are divided into N^2 partitions, $E_{i,j}$, $i, j = 0 \ldots N$.
- $E_{m,n}$ stores edges pointing from R_m to R_n.
- Ranks R_i, $i = 0 \ldots N$ are updated sequentially starting from R_0.
- When updating R_k, all ranks with lower priority, i.e. R_j, $j \geq k$ serve as source, and edge partitions $E_{j,k}$, $j \geq k$ are streamed sequentially.

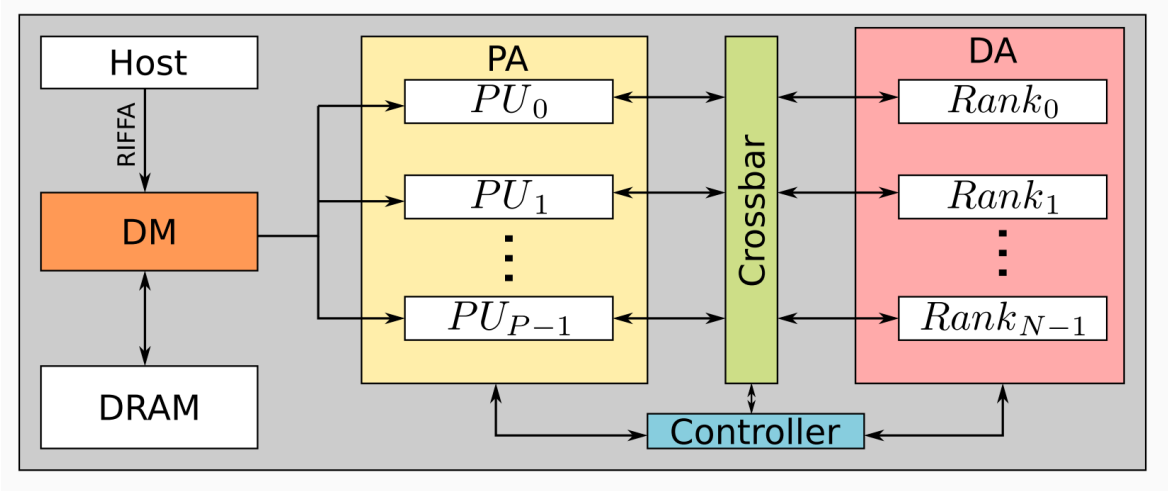
Edge-Centric Graph Processing

- Example: assume there are 4 ranks, N = 4
- Update R_0: R_0 serves as destination block, $E_{0,0}$, $E_{1,0}$, $E_{2,0}$, and $E_{3,0}$ are streamed in to update R_0.
- Update R_1: according to the constructed LDG, it's guaranteed there is no edge pointing from R_0 to R_1, therefore, $E_{1,0}$ doesn't exist. Other ranks are updated similarly.



Hardware Architecture

- Edges are pre-stored in DRAM sent from host through PCIe.
- Nodes are pre-stored in on-chip block memory, each rank is stored in a separate block.
- There are multiple processing units (PUs) working in parallel when updating each destination rank.



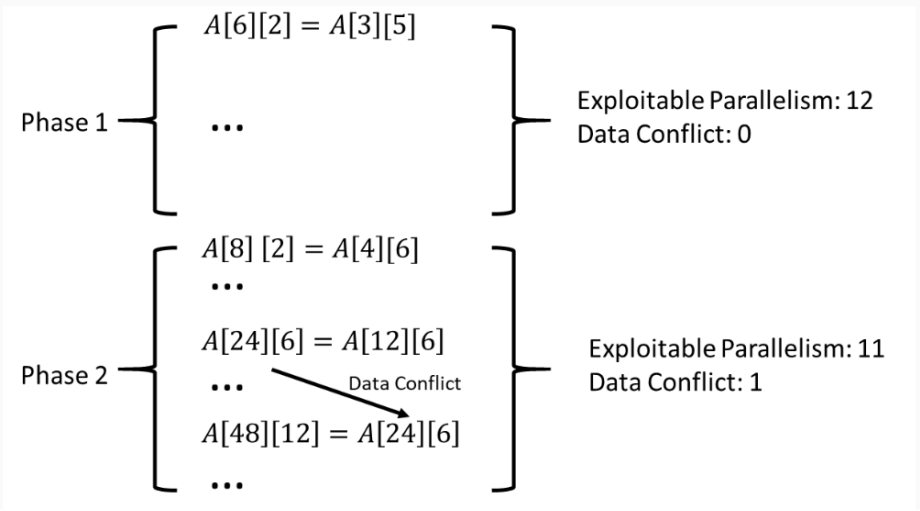
Experiments

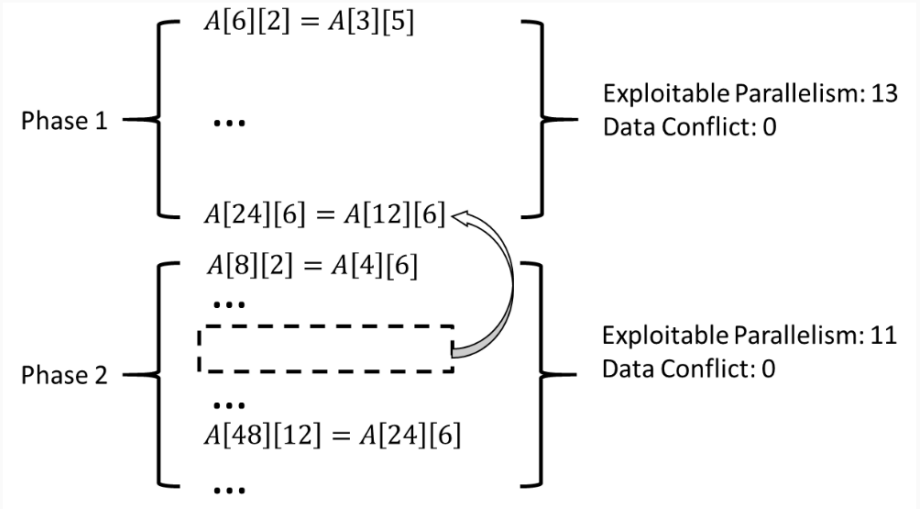
- Conducted experiments on four benchmarks which were studied in other literatures.

Problem & Problem Size (N)	Method	Clock Cycles	Achieved Clock Frequency (MHz)	Resource		
				LUT	FF	DSP
UNIQUE_KERNEL (N = 200) $A[2*i+4*j][j+1] = A[i+2*j][i+4]$	Baseline	79601	475	34	20	2
	Innerloop Pipelining	79204	362	34	20	2
	Innerloop Partial Unrolling (Factor = 12)	59701	159	341	145	24
	Outerloop Pipelining/Innerloop Unrolling	56915	140	5913	3723	391
	Graph-Morphing	6032	276	3304	1071	0
PLUTO_TEMPLATE (N = 60) $A[i][j] = A[j][i] + A[i][j-1]$	Baseline	7081	412	93	50	0
	Innerloop Pipelining	6964	369	101	62	0
	Innerloop Partial Unrolling (Factor = 12)	7022	331	769	496	0
	Outerloop Pipelining/Innerloop Unrolling	5253	278	3653	3849	0
	Graph-Morphing	1845269	247	3878	1310	0
DIST_ITR (N = 20000) $A[2*i] = A[i] + 0.5f$	Baseline	99996	153	208	158	2
	Innerloop Pipelining	20004	144	197	225	2
	Innerloop Unrolling (Factor = 12)	35001	142	602	493	4
	Outerloop Pipelining/Innerloop Unrolling	Out-of-Time[1]				
	Graph-Morphing	16407	125	5596	3243	24
DIST_ITR_PARAM (N = 8000) $A[2*i+3][j] = A[i][j] + 0.5f$	Baseline	12805	149	221	212	3
	Innerloop Pipelining	16009	146	239	227	4
	Innerloop Unrolling (Factor = 12)	32003	143	791	460	28
	Outerloop Pipelining/Innerloop Unrolling	Out-of-Time[1]				
	Graph-Morphing	10849	132	5925	3195	24

Little Analysis

- What is the difference between Graph-Morphing and loop unrolling?
  - Loop (partial) unrolling groups iterations into batches and processes each batch.
  - Essentially, Graph-Morphing reorders iterations first, and then groups into batches.

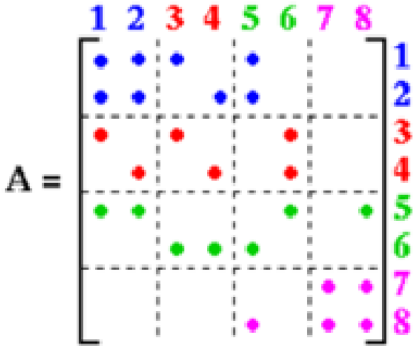


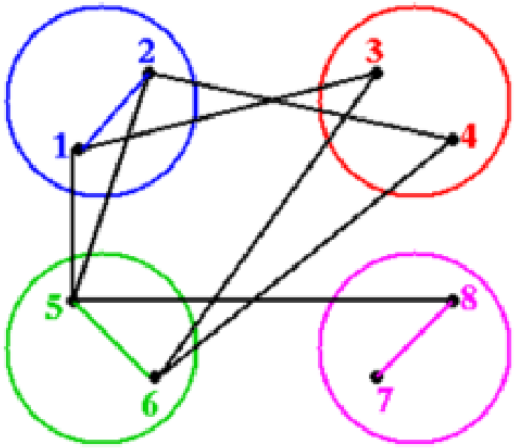


Graph-Based Combinatorial Optimization Problem Solver

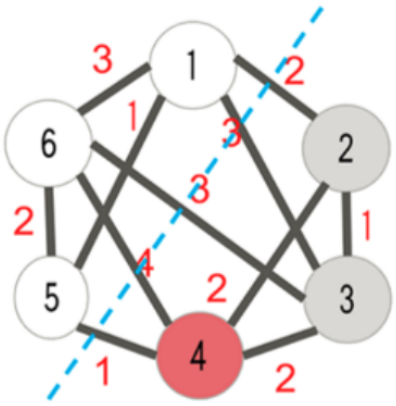
Graph-Based Combinatorial Optimization Problem Solver

- Combinatorial Optimization Problem
  - To find an optimal object from a finite set of candidates.
  - Widely used in scientific and engineering applications.
    - Graph partitioning, max-cut, max-flow, satisfiability, graph coloring,...
  - NP-hard problem and exhaustive search is not tractable.





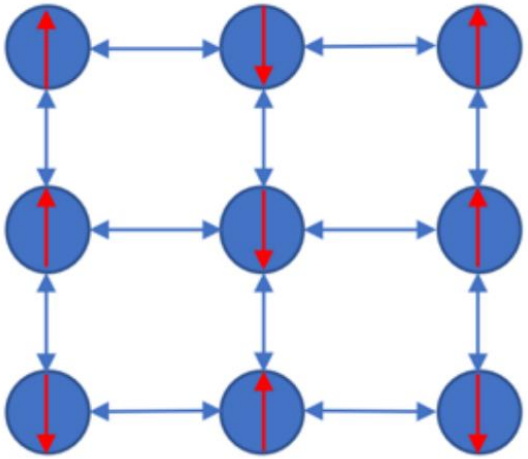
Graph partitioning



Max cut

Ising Model and Ising Computing

- Ising model is an abstract mathematical model to describe ferromagnetism in statistical mechanics.
- Ising model consists of a set of spins interconnected with each other by a weighted edge. Each spin σ_i has two discrete spin values $\sigma_i \in \{-1, +1\}$.



Ising Model and Ising Computing

- The energy of each spin is defined as:

$$H_i(\sigma_i) = -\sum_j J_{i,j}\sigma_i\sigma_j - h_i\sigma_i$$

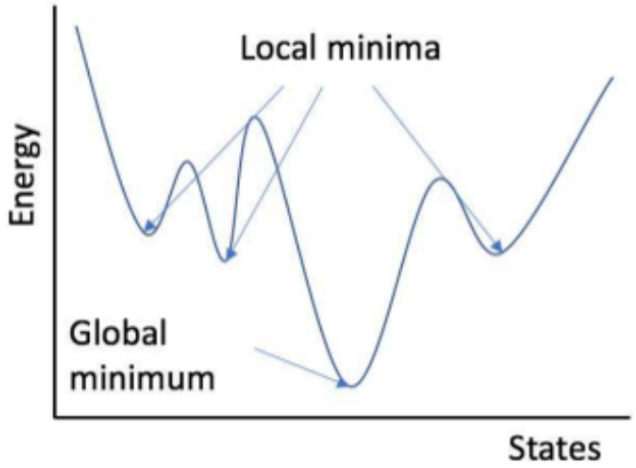
where $J_{i,j}$ is the interaction weight between σ_i and σ_j, and h_i is a bias or external force acting on σ_i.

- The total energy of the model is defined as:

$$H(\sigma_1, \sigma_2, \ldots, \sigma_n) = -\frac{1}{2}\sum_i\sum_j J_{i,j}\sigma_i\sigma_j - \sum_i h_i\sigma_i$$

Ising Computing

- **Ising computing** is to search for a setting of $\vec{\sigma} = \sigma_1, \sigma_2, \ldots, \sigma_n$ to minimize $H(\vec{\sigma})$.
- N number of spins contain 2^N number of settings, such that Ising model is a NP-hard problem.
- The most common approach to solve Ising model is Simulated Annealing.



Benefits of Ising Model

- Ising model provides a data structure general enough to express a lot of combinatorial optimization problems.
- Essentially, Ising model is a graph data structure, therefore, graph-specific optimizations such like vertex traversal and graph partitioning can be applied.
- By solving Ising model, we get a general combinatorial optimization problem solver.

Max-Cut Definition

Partitioning a graph into two subsets S and $\overline{S}$, such that the sum of weighted edges between vertices of the two sets are maximized. Given a graph $G = (V, E)$, assign variable x_i to each vertex:

$$max \frac{1}{2} \sum_i \sum_j \frac{w_{i,j}(1 - x_i x_j)}{2} \quad s.t. \quad x_i \in \{1, -1\}$$

Mapping Max-Cut To Ising Model

- Each vertex is assigned to an Ising spin $\sigma_i \in \{-1, +1\}$, where ± 1 indicates two groups.
- Set $J_{i,j} = -w_{i,j}$, and $h_i = 0$
- Then

$$E_{Ising} = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} J_{i,j}\sigma_i\sigma_j$$

$$C(\vec{\sigma}) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{i,j}\frac{1-\sigma_i\sigma_j}{2} = -\frac{E_{Ising}}{2} - \frac{1}{4}\sum_{i=1}^{N}\sum_{j=1}^{N} J_{i,j}$$

- To find a max cut value of a graph is mapped to searching for the minimum energy of an Ising model.

State-of-The-Art Approach: Simulated Annealing

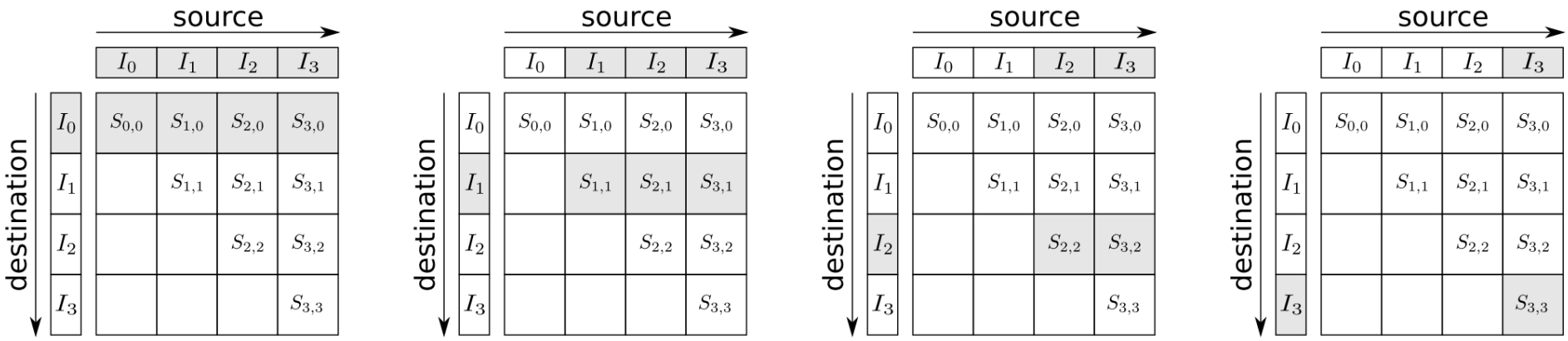
- SA is a heuristic mimicking the thermal annealing.
  - Utilizing randomness to avoid local minimum while getting closer to the global minimum.
  - Widely used to solve 2D Ising models. In 2D lattice model, each spin is only interacting with neighbor spins such that decreasing local energy also decreases the global energy.
  - But:
    - Multiple spins can be evaluated and updated concurrently. However, to guarantee the convergence, no neighbor spins can be evaluated.
    - Problematic for an arbitrary-topology graph

Three-Level Optimizations

- By partitioning the graph such that each partition can fit into on-chip block memory, FPGA computation resources are fully utilized.
- By splitting the vertex memory into multiple banks and scheduling edges to avoid bank conflicts, multiple edges can be processed concurrently to fully utilize FPGA parallelism and DRAM bandwidth.
- By designing a hazard detection unit to monitor the processing pipeline, data hazards are avoided such that the datapath is fully pipelined, and stalled and flushed on a conflict.

Graph Partitioning

- Vertices are partitioned into P intervals.
- Edges are partitioned into $\frac{P(P+1)}{2}$ shards instead of P^2 partitions since edges are undirected.
- When processing one edge shard, corresponding source intervals and destination intervals are loaded to two on-chip memories first before the edge shard is streamed in.



Memory Banking

- To increase parallelism, source memory and destination memory are both partitioned into M banks. Vertex v_i is put at $\frac{i}{M}$ offset within the $(i\%M)$-th bank.
- By memory banking, a batch of M edges can be streamed in and get processed if there is no bank conflict.
- To guarantee there is no bank conflict within each batch, edges within each shard are rescheduled such that edges without bank conflicts are grouped into batches.

• Data Hazard Detection

- The datapath is fully pipelined such that each clock cycle, a new edge batch can get processed.
- To guarantee there is no data hazard, we designed an unit to detect if the current batch has conflict with all the other batches being processed in the pipeline. If there is any data hazard, the new edge batch is held until the conflict gets resolved.

Conclusion

- Graph-Morphing has some limitations:
  - Off-chip DRAM is involved to pre-store edges.
  - Large hardware usage for some cases.
- Graph-Morphing is a good improvement for loop unrolling.
- All graph operations we designed in Graph-Morphing can also be used to guide the compiler to unroll a loop, and in other domains such as auto-parallelizing compiler design.
- Graph-Morphing provides a new perspective for synthesizing a kernel into a reconfigurable logic.

Thank you!