Buffer allocation algorithm for Symbolic Compiler

Sixiao Zhu, Ningyi Xu, MSRA

4/24/2016

Situation

- In Symbolic Compiler, we want to provide high level automation.
- Specifically, given topo structure of some neuron network.

1. Map the computing nodes (like convolution layer in CNN) to FPGA kernel.
2. Map the storage nodes (like feature map in CNN) to DDR block.
3. Generate execution order of computing nodes.

Mapping Computing Nodes

- We generate one FPGA kernel instance for each kind of layer, like convolution layer, fully connect layer, pooling layer, etc.
- Multiple instances of the same kind of layer is mapped to the corresponding FPGA kernel instance, only that this kernel instance is runtime reconfigurable.

Mapping Storage Nodes: Problem

- Need to map numerous storage nodes to limited numbers of DDR buffers, one example is 152 layers deep residual net.
- Complex dependency may exist, the allocation must respect the semantics.
- A specific DDR buffer’s occupation / release depends on execution order. Better generate execution order together with the mapping.

Mapping Storage Nodes: Solution

- Use a BFS like approach to explore the DAG, use the BFS reach order as execution order. On this execution order, we analyze the dataflow dependency.
- Treat this mapping problem as Register Allocation problem in compiler backend. A well-known solution for this is graph coloring (minimizing color number on condition that adjacent node having different color).
- With the obtained dependency, establish graph coloring problem constrains, and use DSATUR heuristic solver to color the graph.

Algo: Convention

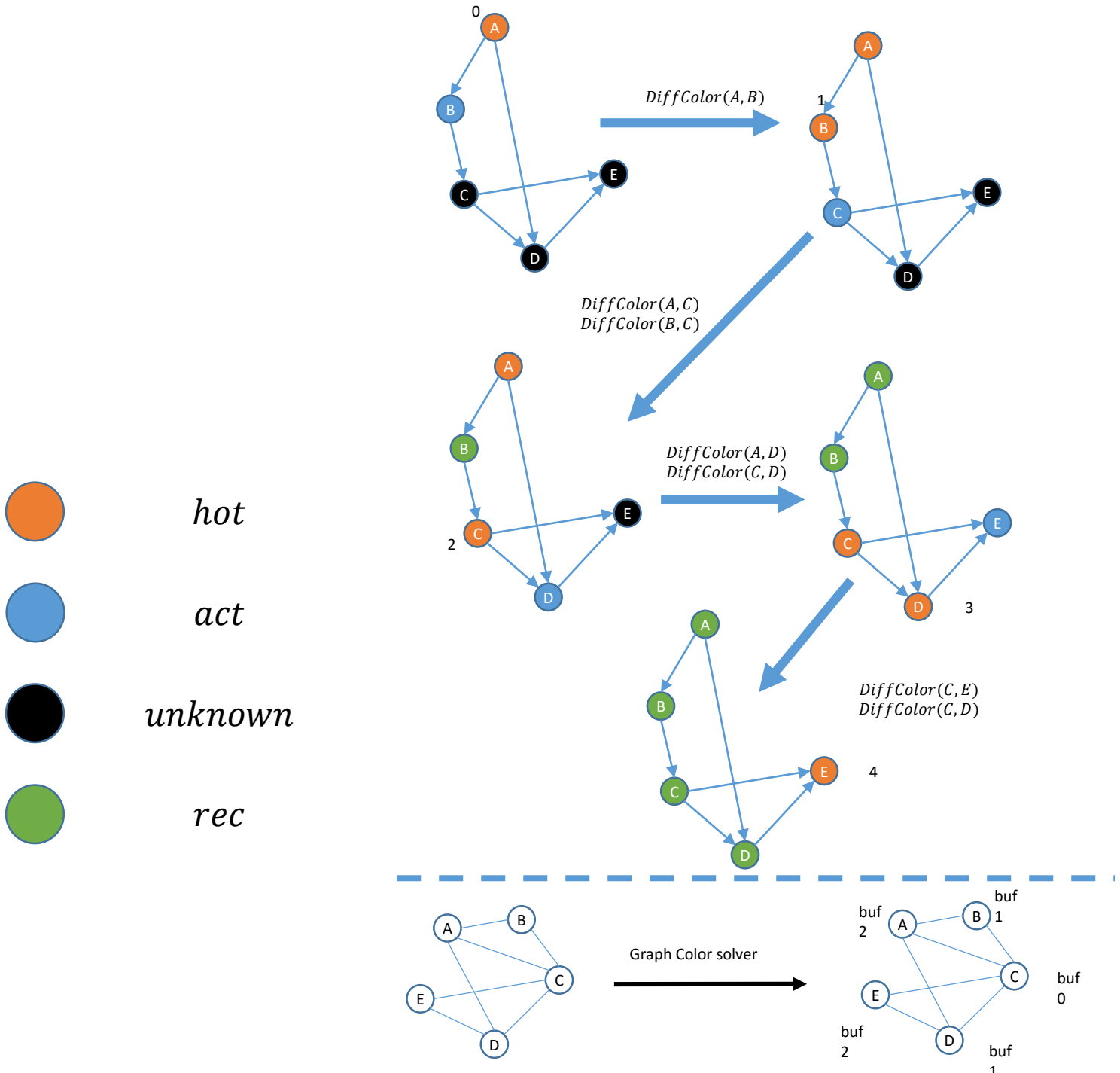
- Let set for all storage unit be U.
- Four sets: $unknown$, act (activated), hot, rec (recycled).
- Each storage unit moves along $unknown$-act-hot-rec. At certain phase, each storage unit belongs to exactly one set.
- Obviously, each storage unit corresponds to one computing that outputs it. So in the graph, we hide the computing units.
- We use a fifo queue to denote the execution order (Similar to `EnqueueTask()` in OpenCL). We use q for it.

Algo: Description Part1

- At the beginning all nodes are in the $unknown$, except the node for network's row input data.
- Update act set. For a node n in $unknown$ set, if all its dependencies are available in hot set, this node is available for computing (activated) and move to act
- Choose a node n from act set (Random choose currently, future research could be done to select the best one). Push the computing unit it corresponds to into q. Establish graph color constrain between n and all nodes in hot set, move it to hot set.
- Update hot set. A node is move from hot set to rec if all nodes that depends on it is already in hot or rec set (no use anymore).

Algo: Description Part2

- After all the nodes are moved to rec, solve the graph coloring problem with a solver ($DSATUR$ currently). After this, each storage unit is assigned to a actual DDR block.
- Change all logic storage units that each computing unit concerns to the assigned DDR block.
- Execute all the computing units in q, in a fifo order.


hot
act
unknown
rec
0
A
B
C
D
E
$DiffColor(A,B)$
1
$DiffColor(A,C)$
$DiffColor(B,C)$
2
$DiffColor(A,D)$
$DiffColor(C,D)$
3
$DiffColor(C,E)$
$DiffColor(C,D)$
4
Graph Color solver
buf 2
buf 1
buf 0
buf 2
buf 1

Results of the previous example

- Execution order: $B \rightarrow C \rightarrow D \rightarrow E$
- Storage unit mapping: (A, buf2), (B, buf1), (C, buf0), (D, buf1), (E, buf2)

Thanks