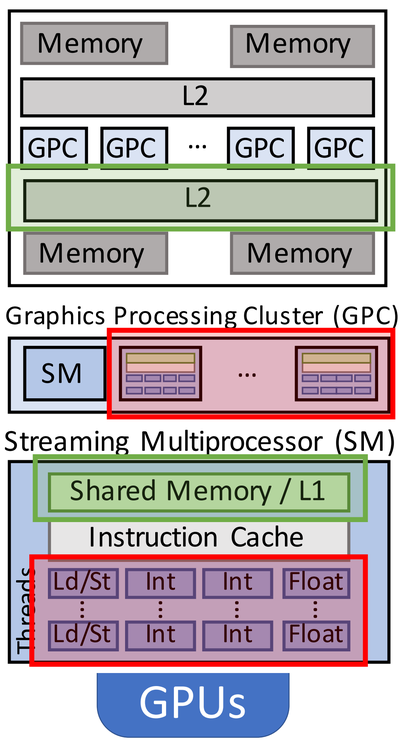
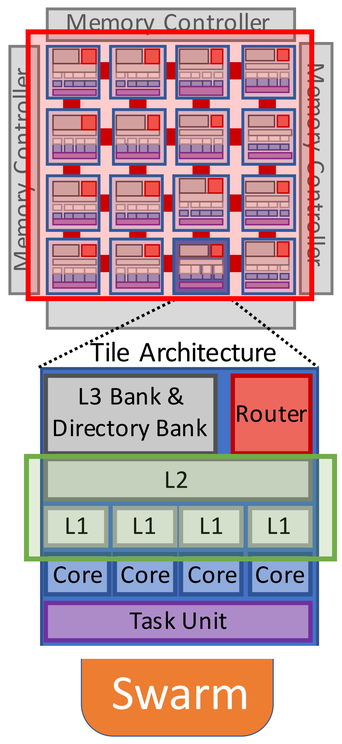
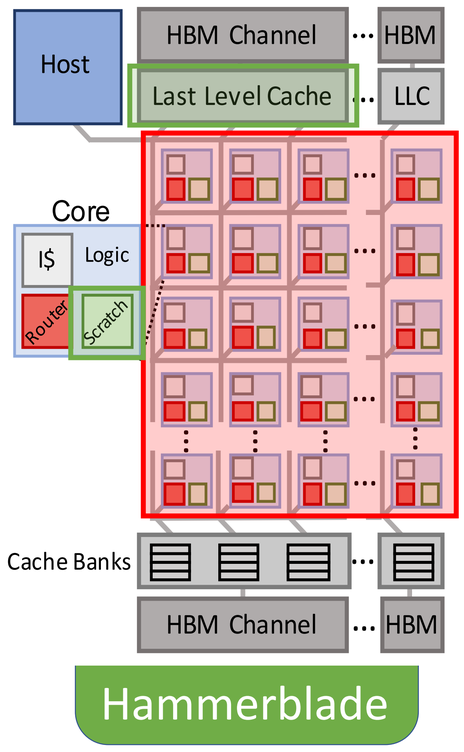
Taming the Zoo: A Unified Graph Compiler Framework for Novel Architectures

Ajay Brahmakshatriya[1], Emily Furst[2], Victor Yang[1], Claire Hsu[1], Changwan Hong[1], Max Ruttenberg[2], Yunming Zhang[1], Tommy Jung[2], Dustin Richmond[2], Michael Taylor[2], Julian Shun[1], Mark Oskin[2], Daniel Sanchez[1], Saman Amarasinghe[1]

1 – Massachusetts Institute of Technology, 2 – University of Washington







Similar features in architectures inspire similar optimizations

Massive parallelism available
- Load Balance optimizations

Programmable and mapped caches
- Blocking for improving memory bandwidth

Application domains require hardware independent optimizations

- PUSH/PULL traversal in graph applications
- Ordered processing for better work efficiency

Single compiler framework for maximizing reusability

```
…
func toFilter(v: Vertex) -> output: bool
  output = (parent[v] == -1);
end
func updateEdge(src: Vertex, dst: Vertex)
  parent[dst] = src;
end
func BFS()
  …
  #s0# while (frontier.getVertexSetSize() != 0)
    #s1# var output: vertexset{Vertex} =
      edges.from(frontier)
      .to(toFilter)
      .applyModified(updateEdge, parent, true);
    …
    end
  delete frontier
end
```

GraphIt domain specific program input
Algorithm + Schedule(s)

GPU

```
SimpleGPUSchedule s1;
s1.configDirection(PUSH);
s1.configFrontierCreation(FUSED);
SimpleGPUSchedule s2;
s2.configDirection(PULL, BITMAP);
s2.configFrontierCreation(BITMAP);
CompositeGPUSchedule h1 (INPUT_SET_SIZE,
0.15, s1, s2);
```

Swarm

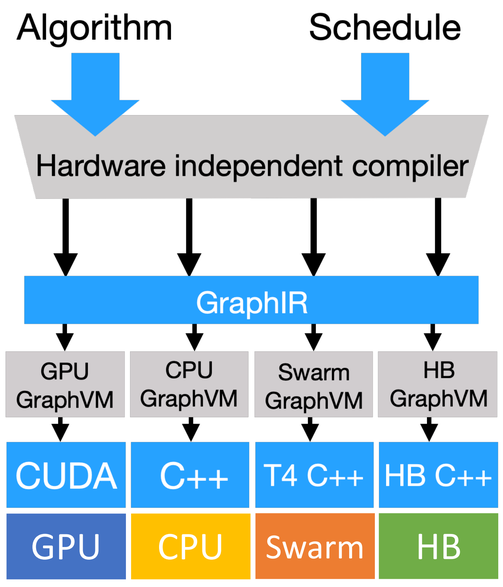
```
SimpleSwarmSchedule s1;
s1.taskGranularity(FINE_GRAINED);
s1.configFrontiers(VERTEXSET_TO_TASKS);
```

HB

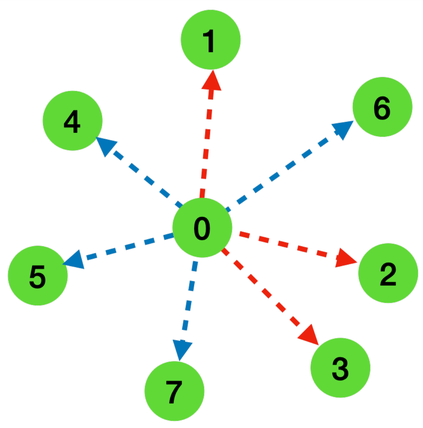
```
SimpleHBSchedule s1;
s1.configLoadBalance(ALIGNED);
s1.configDirection(HYBRID);
```

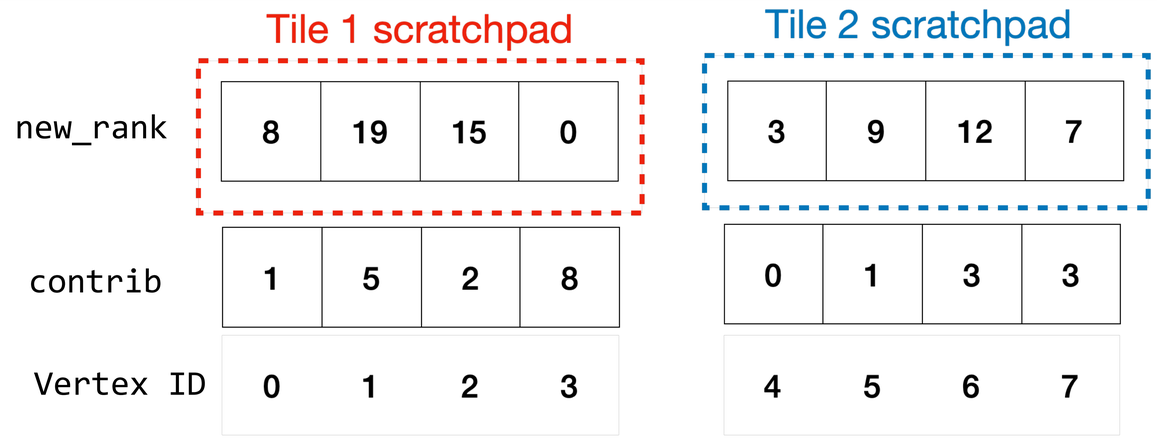
```
Function updateEdge (int32_t src, int32_t dst,
 VertexSet output_frontier, {
 bool enqueue = CompareAndSwap<is_atomic=true>(
  parent[dst], -1, src),
 If (enqueue, {
  EnqueueVertex<format=SPARSE>(output_frontier, dst)
 }, {}) }
Function main (int32_t argc, char* argv[], {
 WhileLoopStmt<needs_fusion=true>(VertexSetSize(frontier), {
  EdgeSetIterator<requires_output=true,
   , can_reuse_frontier=true,
   , direction=PUSH,
   , is_edge_parallel=true> (
   edges, frontier, output, updateEdge, toFilter),
 AssignStmt(frontier, output)
}), )}
```

Generated Graph Intermediate Representation (GraphIR) with arguments and metadata



Target specific GraphVMs (backend) for hardware-dependent optimization and code generation

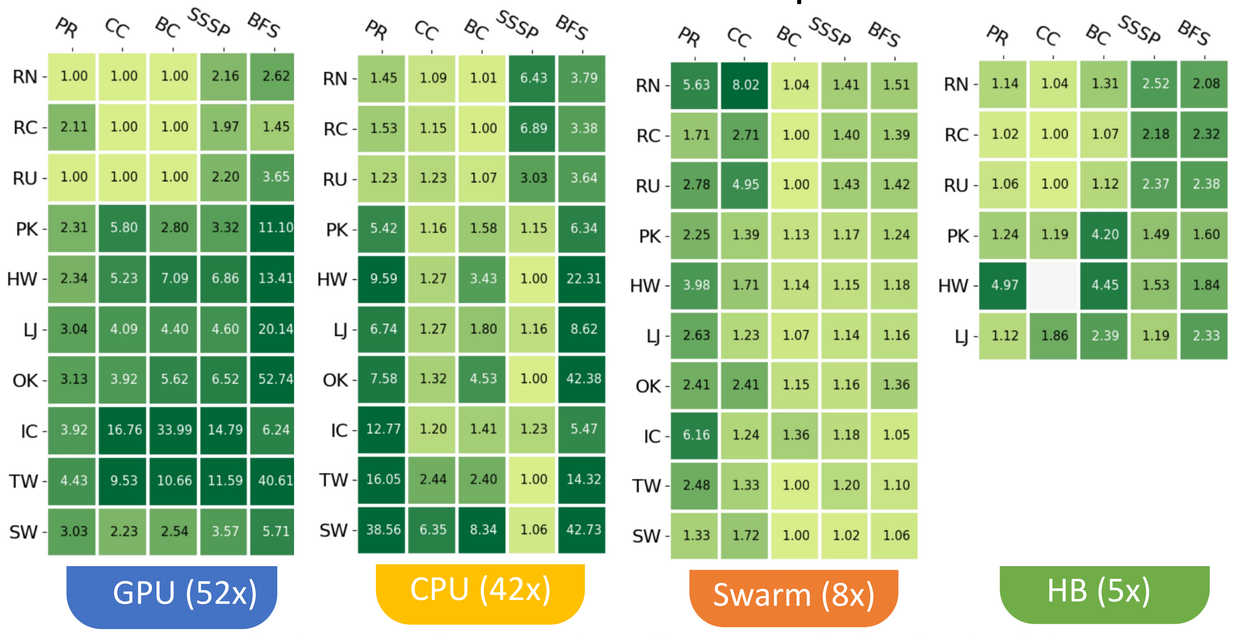




PageRank: Graph split into V/b subgraphs
(V = Size of vertex data, b = scratchpad size)

Graph blocking optimization implemented for the GPU GraphVM, reused by the Hammerblade GraphVM

Our framework allows for performance critical optimizations with minimum developer effort



Relative speedup over baseline-unoptimized schedule

		Base	GPU	CPU	Swarm	HB
Frontend	Parser + AST	10,900	0	0	0	0
	Scheduling Lang.	136	+306	+385	+524	+89
Hardware Independent	Frontier Reuse	125	Not Used	0	0	0
	Property Analysis	536	0	0	Not Used	0
	Ordered Processing	286	0	+120	0	0
	Other Lowering	4,171	0	0	0	0
Hardware dependent	Ordered Specilization	104	0	Not Used	0	Not Used
	Kernel Fusion	0	Not Used	+276	Not Used	Not Used
	Code Generation	0	+3,843	+1,874	+959	+2,282
	Runtime Library	0	+10,385	+2,470	+156	+1,127

LoC for various hardware independent and dependent parts of framework

https://graphit-lang.org

CSAIL