Compiling Graph Applications for GPUs with GraphIt

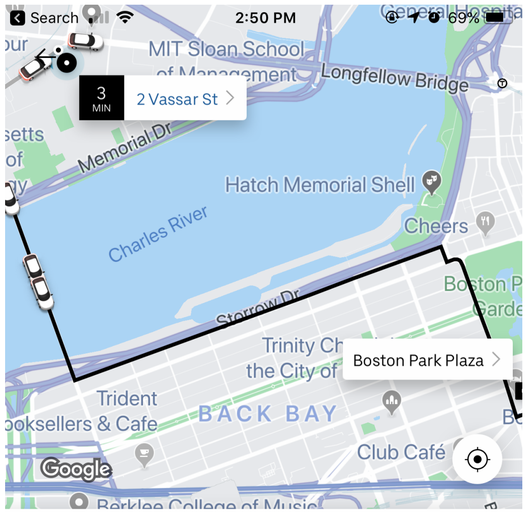
Ajay Brahmakshatriya[1]

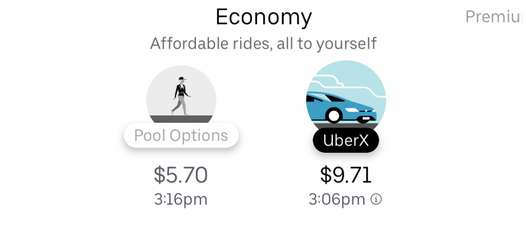
Yunming Zhang[1], Changwan Hong[1], Shoaib Kamil[2], Julian Shun[1], Saman Amarasinghe[1]

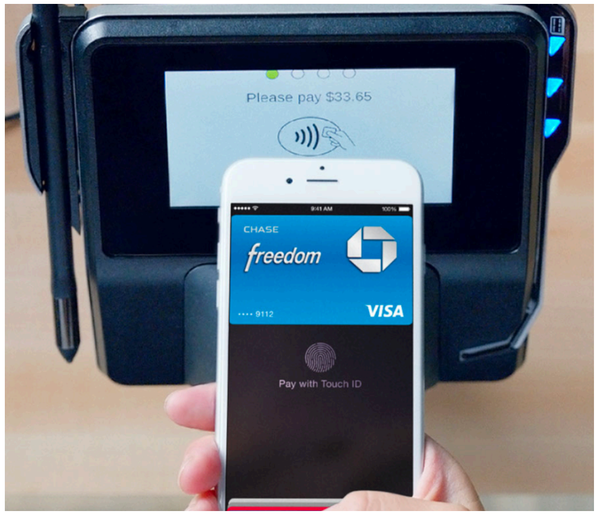
1 Massachusetts Institute of Technology
2 Adobe Research

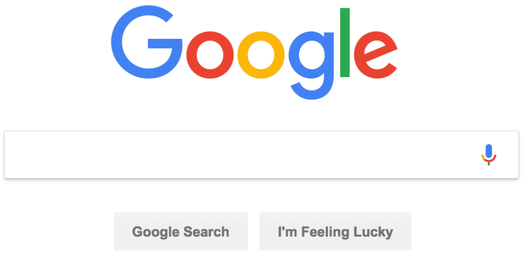
3rd March 2021

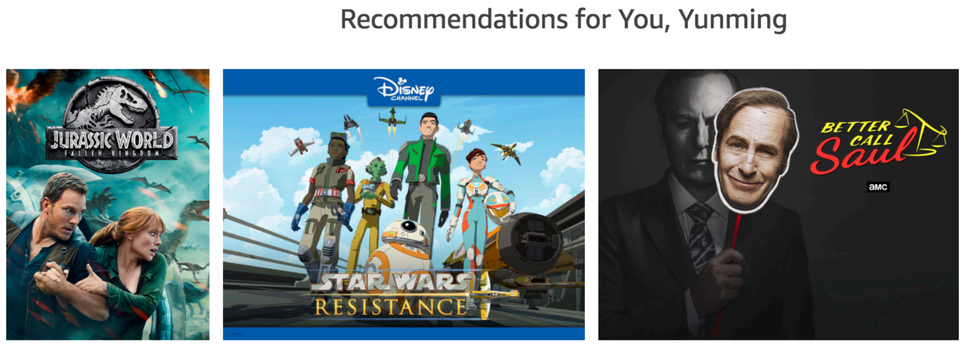
Performance of Graph applications is critical





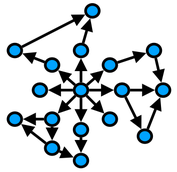






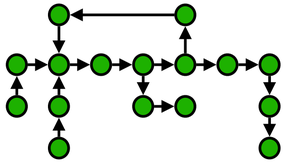
http://google.com . https://en.wikipedia.org/wiki/Polygon_mesh#/media/File:Dolphin_triangle_mesh.png, https://www.bankinfosecurity.com/webinars/customer-awareness-what-works-in-fraud-detection-prevention-w-423 http://amazon.com https://makeawebsitehub.com/social-media-sites/

High performance implementations are hard to write

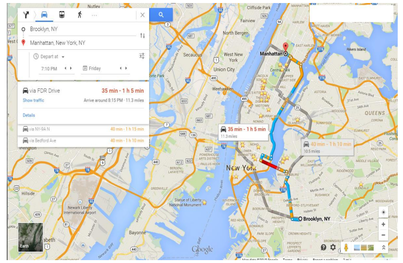


Power-Law Degree Distribution

Social Networks

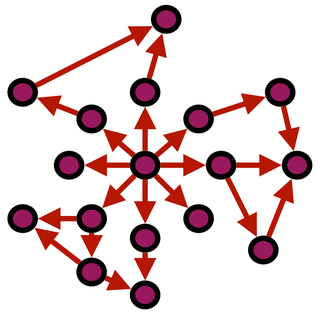


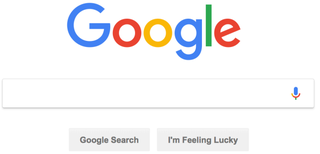
Bounded Degree Distribution



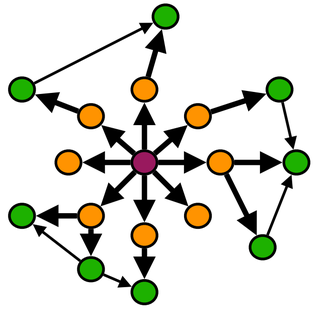
Maps

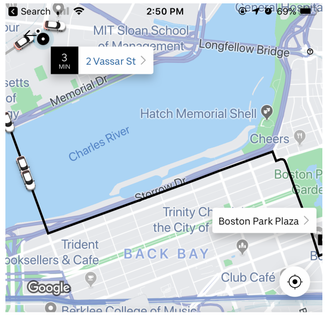
Variations in Graph types





Topology driven algorithms like PageRank

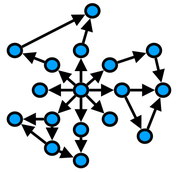




Data driven algorithms like Single Source Shortest Path

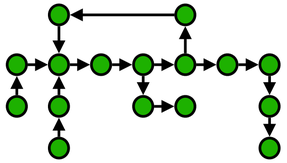
Variations in Algorithms

High performance implementations are hard to write

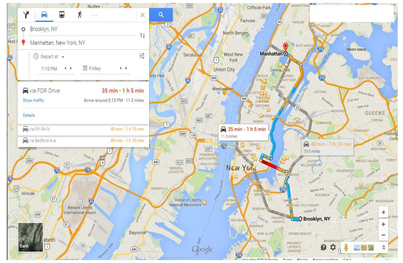


Power-Law Degree Distribution

Social Networks



Bounded Degree Distribution

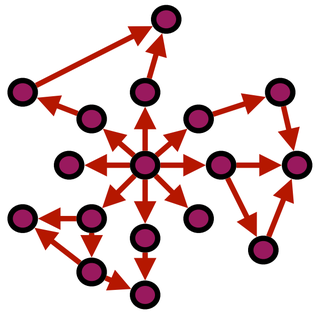


Maps

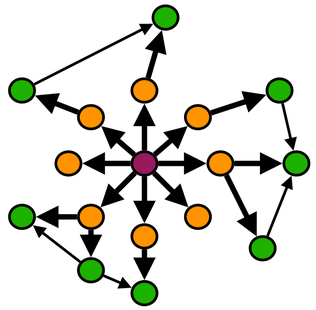
Variations in Graph types

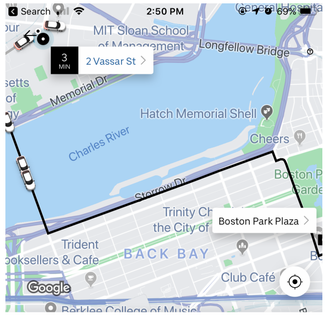
GraphIt

Domain Specific Language and compiler [1][2]



Topology driven algorithms like PageRank





Data driven algorithms like Single Source Shortest Path

Variations in Algorithms

1. Zhang et.al. , GraphIt: a high-performance graph DSL. Proc. ACM Program. Lang. 2, OOPSLA, Article 121 (November 2018)
2. Zhang et.al. , Optimizing ordered graph algorithms with GraphIt. International Symposium on Code Generation and Optimization (CGO 2020)

GraphIt domain specific language

Separate algorithm from schedule

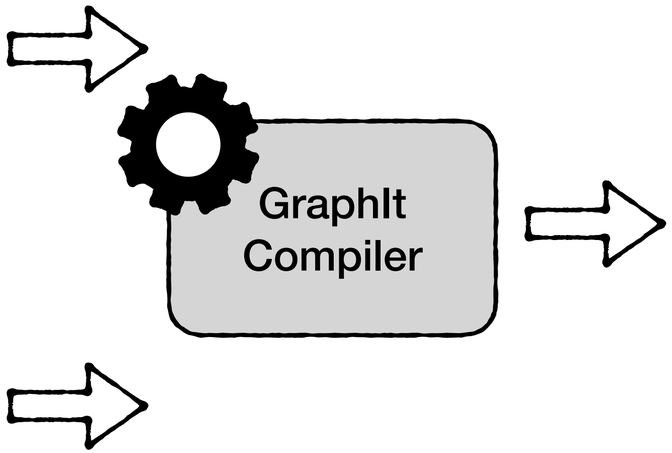
Breadth First Search

Algorithm

```
const edges : edgeset{Edge}(Vertex,Vertex) = load(argv[1]);
const vertices : vertexset{Vertex} = edges.getVertices();
const parent : vector{Vertex}(int) = -1;

func toFilter(v : Vertex) -> output : bool
  output =  (parent[v] == -1);
end

func updateEdge(src : Vertex, dst : Vertex)
  parent[dst] = src;
end

func main()
  var frontier : vertexset{Vertex} = new vertexset{Vertex}(0);
  var start_vertex : int = atoi(argv[2]);
  frontier.addVertex(start_vertex);
  parent[start_vertex] = start_vertex;
  #s0# while (frontier.getVertexSetSize() != 0)
    #s1# var output : vertexset{Vertex} = edges.from(frontier).to(toFilter).
      applyModified(updateEdge, parent, true);
    delete frontier;
    frontier = output;
  end
  delete frontier;
end
```

Schedule

```
program->configApplyDirection("s0:s1", "SparsePush");
program->configApplyParallelization("s0:s1", "dynamic-vertex-parallel");
```




```
int * parent = new int[num_verts];
int * out_degree = new int[num_verts];

…

parallel_for (NodeID src : vertices) {
  for(NodeID dst : G.getOutNgh(src)){
     compareAndSwap(parent[dst], -1, src);
   }
}

….
```

GraphIt domain specific language

Separate algorithm from schedule

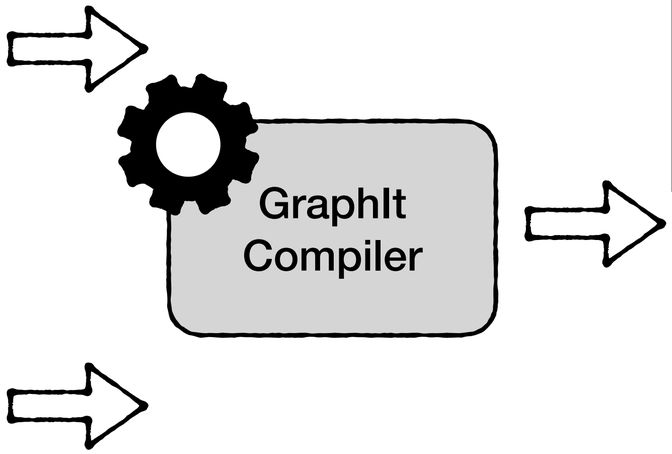
Breadth First Search

Algorithm

```
const edges : edgeset{Edge}(Vertex,Vertex) = load(argv[1]);
const vertices : vertexset{Vertex} = edges.getVertices();
const parent : vector{Vertex}(int) = -1;

func toFilter(v : Vertex) -> output : bool
  output =  (parent[v] == -1);
end

func updateEdge(src : Vertex, dst : Vertex)
  parent[dst] = src;
end

func main()
  var frontier : vertexset{Vertex} = new vertexset{Vertex}(0);
  var start_vertex : int = atoi(argv[2]);
  frontier.addVertex(start_vertex);
  parent[start_vertex] = start_vertex;
  #s0# while (frontier.getVertexSetSize() != 0)
    #s1# var output : vertexset{Vertex} = edges.from(frontier).to(toFilter).
      applyModified(updateEdge, parent, true);
    delete frontier;
    frontier = output;
  end
  delete frontier;
end
```

Schedule

```
program->configApplyDirection("s0:s1", "DensePull");
program->configApplyParallelization("s0:s1", "dynamic-vertex-parallel");
```




```
double * new_rank = new double[num_verts];
double * old_rank = new double[num_verts];
int * out_degree = new int[num_verts];

…

parallel_for (NodeID dst : vertices) {
  for(NodeID src : G.getInNgh(dst)){
     new_rank[dst]  += old_rank[src] / out_degree[src];
   }
}
```

GraphIt domain specific language

Separate algorithm from schedule

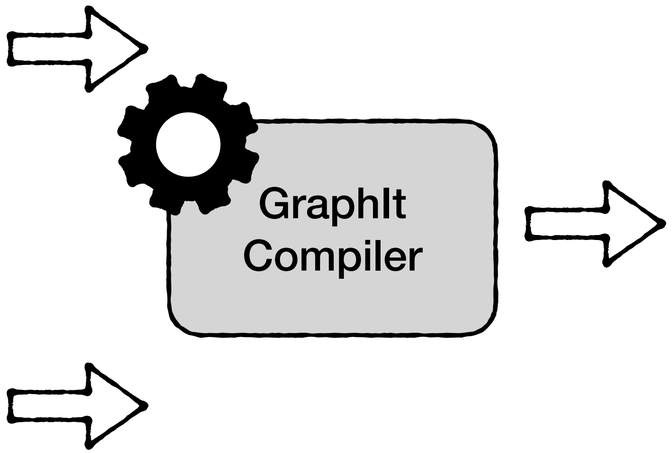
Breadth First Search

Algorithm

```
const edges : edgeset{Edge}(Vertex,Vertex) = load(argv[1]);
const vertices : vertexset{Vertex} = edges.getVertices();
const parent : vector{Vertex}(int) = -1;

func toFilter(v : Vertex) -> output : bool
  output =  (parent[v] == -1);
end

func updateEdge(src : Vertex, dst : Vertex)
  parent[dst] = src;
end

func main()
  var frontier : vertexset{Vertex} = new vertexset{Vertex}(0);
  var start_vertex : int = atoi(argv[2]);
  frontier.addVertex(start_vertex);
  parent[start_vertex] = start_vertex;
  #s0# while (frontier.getVertexSetSize() != 0)
    #s1# var output : vertexset{Vertex} = edges.from(frontier).to(toFilter).
      applyModified(updateEdge, parent, true);
    delete frontier;
    frontier = output;
  end
  delete frontier;
end
```

Schedule

```
program->configApplyDirection("s0:s1", "SparsePush");
program->configApplyParallelization("s0:s1", "dynamic-vertex-parallel");
program->configApplyNumSSG("s0:s1", "fixed-vertex-count", 10);
```




```
int * parent = new int[num_verts];
int * out_degree = new int[num_verts];

…

parallel_for (NodeID src : vertices) {
  for(NodeID dst : G.getOutNgh(src)){
     compareAndSwap(parent[dst], -1, src);
   }
}

….
```

GraphIt domain specific language

Algorithm

```
const edges : edgeset{Edge}(Vertex,Vertex) = load(argv[1]); % CSR, COO, ..
...
func updateEdge(src : Vertex, dst : Vertex)
  parent[dst] = src;
end

func main()
  ...
  var frontier: vertexset{Vertex} = new vertexset{Vertex}(); % Bitmap, Sparse Queues

  while (frontier.getVertexSetSize() != 0)
    vertices.apply(applyVertex);

    output = edges.from(frontier).to(toFilter).applyModified(updateEdge, parent);

    ...
  end
end
```

GraphIt domain specific language

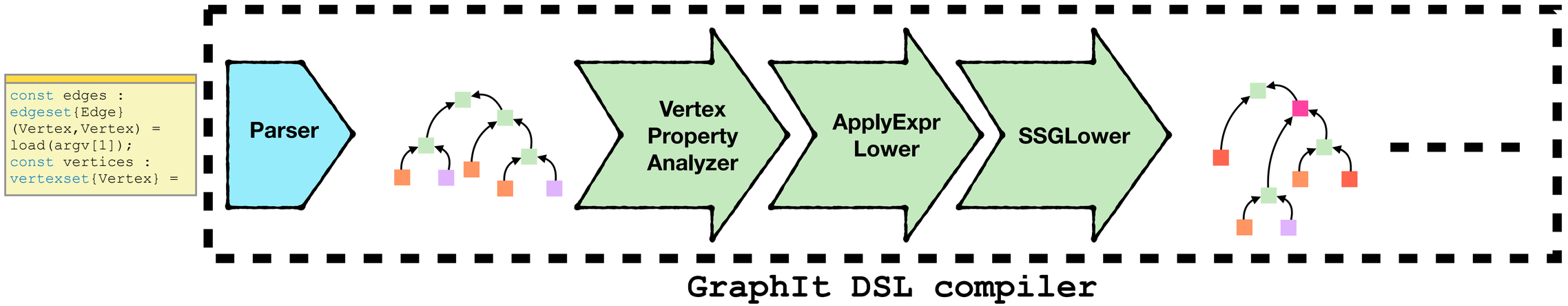
Algorithm

```
const edges : edgeset{Edge}(Vertex,Vertex) = load(argv[1]); % CSR, COO, ..
...
func updateEdge(src : Vertex, dst : Vertex)
  parent[dst] = src;
end

func main()
  ...
  var frontier: vertexset{Vertex} = new vertexset{Vertex}(); % Bitmap, Sparse Queues

  while (frontier.getVertexSetSize() != 0)
    vertices.apply(applyVertex);

    #s1# output = edges.from(frontier).to(toFilter).applyModified(updateEdge, parent);

    ...
  end
end
```

GraphIt domain specific language

Schedule

```
program->configApplyDirection("s1", "SparsePush");
program->configApplyParallelization("s1", "dynamic-vertex-parallel");
program->configApplyNumSSG("s1", "fixed-vertex-count", 10);
```



- Whole program analysis / transformations

Hardware variations?

```
...
gpu_runtime::vertex_set_apply_kernel<gpu_runtime::AccessorAll, reset><<<NUM_CTA,
CTA_SIZE>>>(__host_edges.getFullFrontier());
...
while ((__local_pq.device_finished()) == (0)) {
    local_frontier = __local_pq.device_dequeueReadySet();
    ...
    if (gpu_runtime::builtin_getVertexSetSize(frontier) < frontier.max_num_elems *
    hybrid_threshold_var2) {
        gpu_runtime::vertex_set_prepare_sparse(frontier);
        ...
        gpu_runtime::TWCE_load_balance_host<char, gpu_operator_body_3,
    gpu_runtime::AccessorSparse, gpu_runtime::true_function>(__host_edges, frontier,
    output);
        ...
    } else {
        gpu_runtime::vertex_set_prepare_bitmap(frontier);
        gpu_runtime::vertex_set_create_reverse_sparse_queue_host<toFilter>(frontier);
        ...
        gpu_runtime::vertex_based_load_balance_host<char, gpu_operator_body_4,
    gpu_runtime::AccessorSparse, toFilter>(__host_edges__transposed, frontier, output);
        ...
    }
    ...
```

GPU

???

CPU

Vertices Processed

1 1 1 3

Hardware variations?

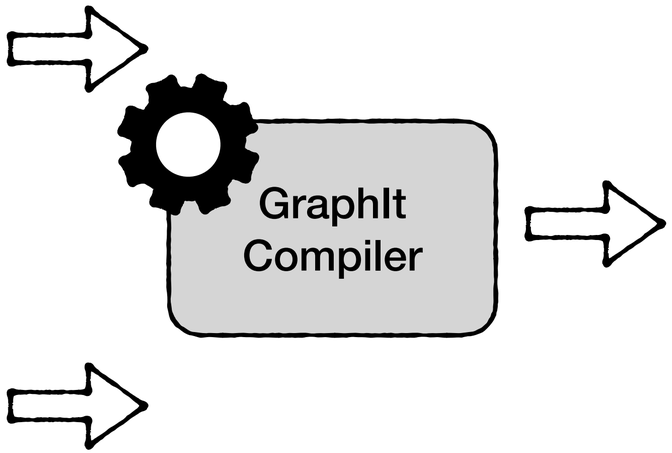
Separate algorithm from schedule

Algorithm

```
const edges : edgeset{Edge}(Vertex,Vertex) = load(argv[1]);
const vertices : vertexset{Vertex} = edges.getVertices();
const parent : vector{Vertex}(int) = -1;

func toFilter(v : Vertex) -> output : bool
  output =  (parent[v] == -1);
end

func updateEdge(src : Vertex, dst : Vertex)
  parent[dst] = src;
end

func main()
  var frontier : vertexset{Vertex} = new vertexset{Vertex}(0);
  var start_vertex : int = atoi(argv[2]);
  frontier.addVertex(start_vertex);
  parent[start_vertex] = start_vertex;
  #s0# while (frontier.getVertexSetSize() != 0)
    #s1# var output : vertexset{Vertex} = edges.from(frontier).to(toFilter).
      applyModified(updateEdge, parent, true);
    delete frontier;
    frontier = output;
  end
  delete frontier;
end
```

Schedule

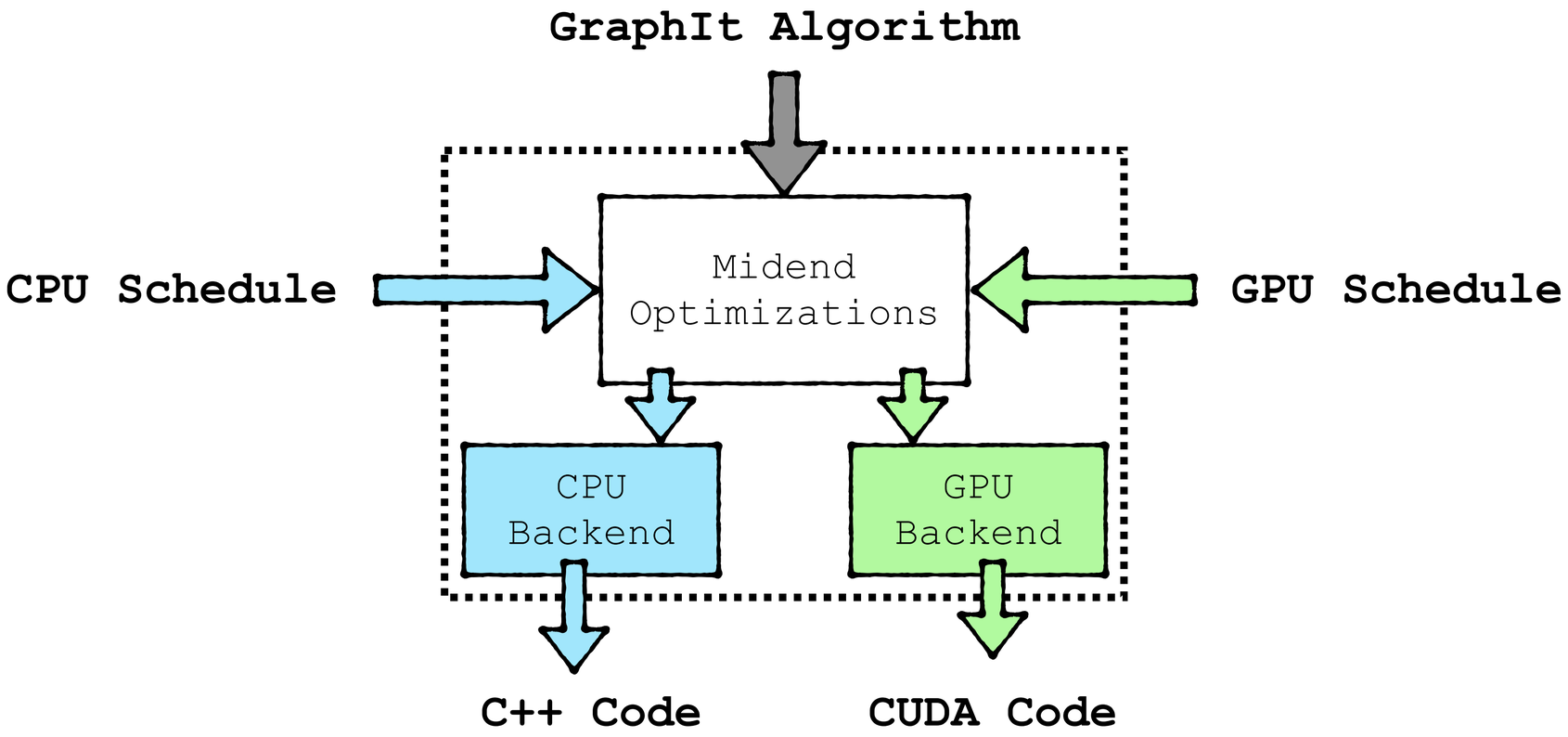
```
GPU!!
```



CUDA C++

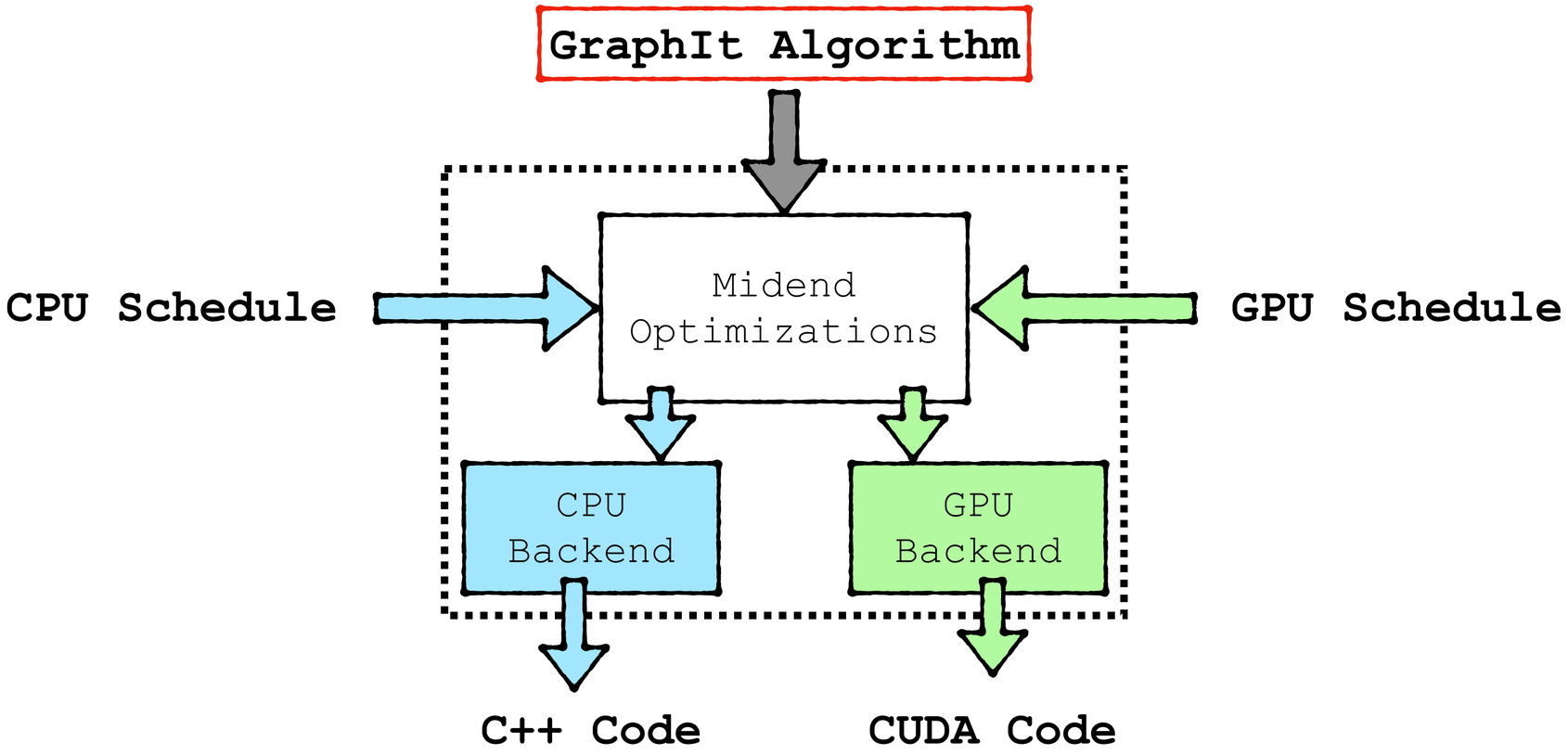
```
void __global__ fused_kernel_body_3(void) {
    grid_group _grid = this_grid();
    int32_t _thread_id =
         threadIdx.x + blockIdx.x * blockDim.x;
    auto __local_frontier =
         fused_kernel_body_3_frontier;


….


    cudaLaunchCooperativeKernel(fused_kernel_body_3,
        NUM_CTA,
        CTA_SIZE, gpu_runtime::no_args);
```

G2: GPU Extensions for GraphIt



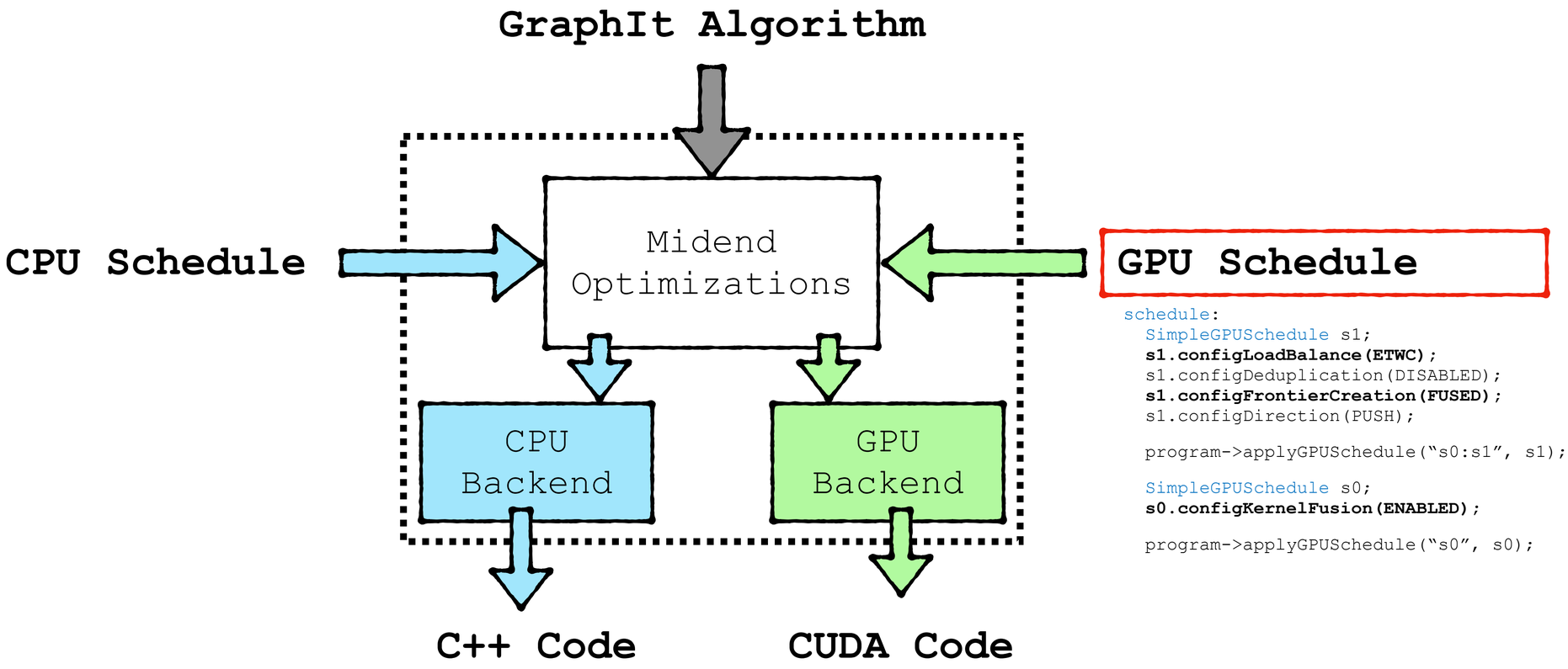
The G2 Domain Specific Compiler

G2: GPU Extensions for GraphIt



The G2 Domain Specific Compiler

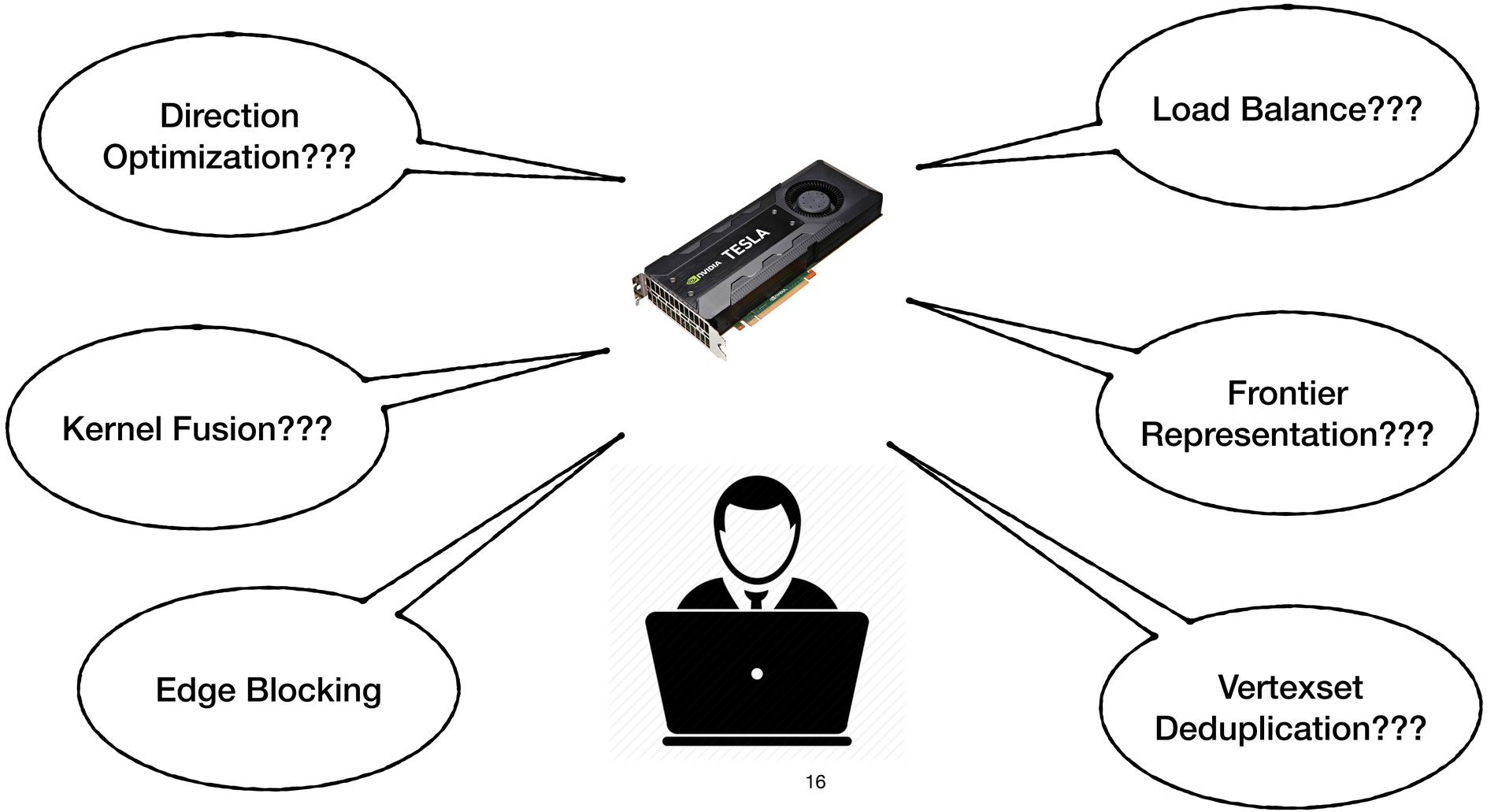
G2: GPU Extensions for GraphIt




```
schedule:
  SimpleGPUSchedule s1;
  s1.configLoadBalance(ETWC);
  s1.configDeduplication(DISABLED);
  s1.configFrontierCreation(FUSED);
  s1.configDirection(PUSH);

  program->applyGPUSchedule("s0:s1", s1);

  SimpleGPUSchedule s0;
  s0.configKernelFusion(ENABLED);

  program->applyGPUSchedule("s0", s0);
```

The G2 Domain Specific Compiler

Key GPU Optimizations



Key GPU Optimizations

- Load Balance Strategy
- Iteration direction
- Representation of output frontier
- Deduplication of output frontier

- Fusing multiple CUDA kernels
- Graph partitioning for cache utilization
- Runtime combinations of above

```
configLoadBalance(CM|WM|TWC|TWCE|EB|VB|STRICT)

configDirection(PUSH|PULL)

configFrontierCreation(FUSED|UNFUSED, BITMAP|
                       BOOLMAP)

configDeduplication(ENABLED|DISABLE, BITMAP|
                    BOOLMAP|MONOTONIC_COUNTERS)

configKernelFusion(ENABLED|DISABLED)

configEdgeBlocking(ENABLED|DISABLED)

HybridGPUSchedule
```

Scheduling in G2

Optimization	Gunrock	GSWITCH	SEP-Graph	G2
Load Balancing	VERTEX BASED, EDGE BASED, TWC	CM, WM, TWC, STRICT	VERTEX BASED	ETWC, TWC, STRICT, CM, WM, VERTEX BASED, EDGE BASED
Edge Blocking	Not supported	Not supported	Not supported	Supported
Vertex Set Creation	Fused/Unfused	Fused/Unfused	Fused	Sparse Queue/Bitmap/ Boomlet
Direction Optimization	Push/Pull/Hybrid	Push/Pull/Hybrid	Push/Pull/Hybrid	Push/Pull/Hybrid
Deduplication	Supported	Not supported	Supported	Supported
Vertex Ordering	Supported	Supported	Supported	Supported
Kernel Fusion	Supported	Not supported	Supported	Supported

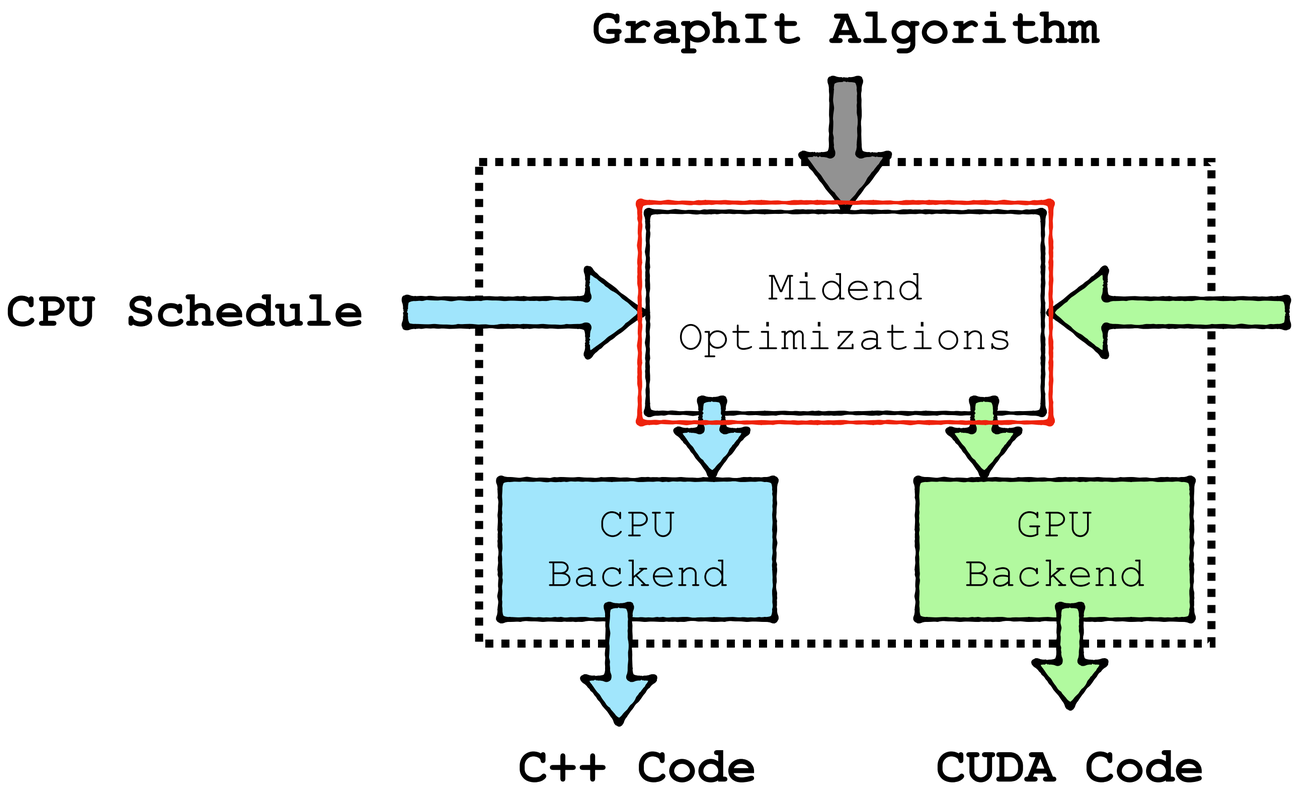
Comparison of number of variations of different optimizations available in G2 and other state of the art graph-frameworks - Gunrock, GSWITCH, SEP-Graph

Scheduling in G2

Optimization	Gunrock	GSWITCH	SEP-Graph	G2
Load Balancing	VERTEX BASED, EDGE BASED, TWC	CM, WM, TWC, STRICT	VERTEX BASED	ETWC, TWC, STRICT, CM, WM, VERTEX BASED, EDGE BASED
Edge Blocking	Not supported	Not supported	Not supported	Supported
Vertex Set Creation	Fused/Unfused	Fused/Unfused	Fused	Sparse Queue/Bitmap/ Boomlet
Direction Optimization	Push/Pull/Hybrid	Push/Pull/Hybrid	Push/Pull/Hybrid	Push/Pull/Hybrid
Deduplication	Supported	Not supported	Supported	Supported
Vertex Ordering	Supported	Supported	Supported	Supported
Kernel Fusion	Supported	Not supported	Supported	Supported
Total combinations	48	32	16	**576**

Comparison of number of variations of different optimizations available in G2 and other state of the art graph-frameworks - Gunrock, GSWITCH, SEP-Graph

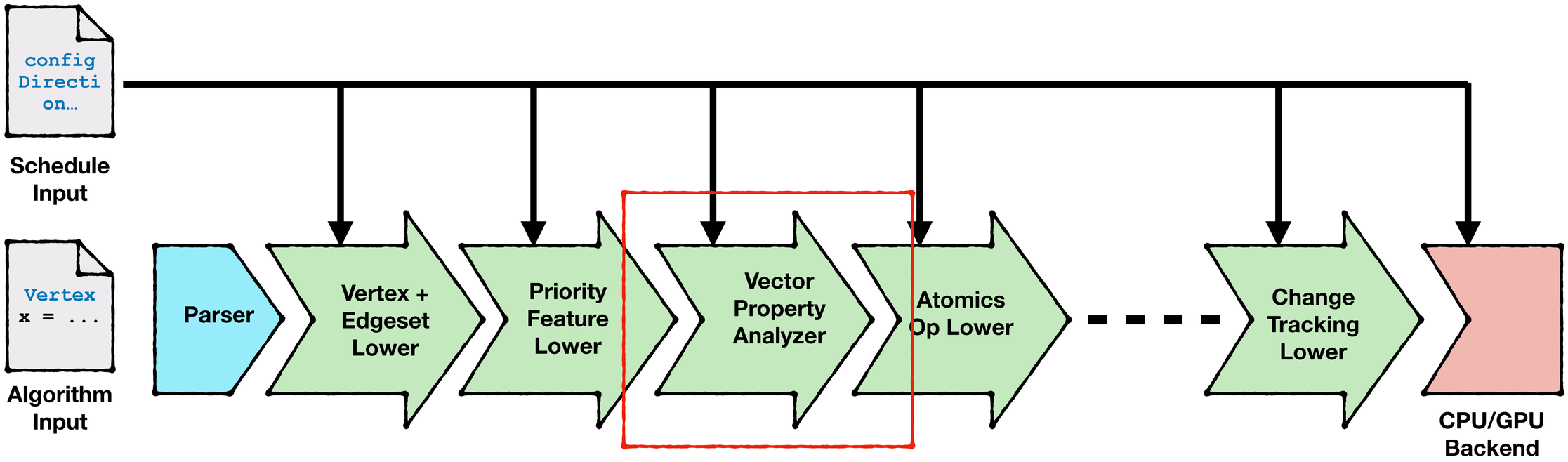
G2: GPU Extensions for GraphIt




```
schedule:
  SimpleGPUSchedule s1;
  s1.configLoadBalance(ETWC);
  s1.configDeduplication(DISABLED);
  s1.configFrontierCreation(FUSED);
  s1.configDirection(PUSH);

  program->applyGPUSchedule("s0:s1", s1);

  SimpleGPUSchedule s0;
  s0.configKernelFusion(ENABLED);

  program->applyGPUSchedule("s0", s0);
```

The G2 Domain Specific Compiler

G2: Midend optimizations



G2 Midend optimization pass infrastructure

G2: Vector Property Analyzer

Algorithm

```
..

func updateEdge(src : Vertex, dst : Vertex)
   IDs[dst] += IDs[src];
end


func main()
    ..
    edges.from(frontier).to(toFilter).applyModified(updateEdge, IDs, true);
    ..
end
```

Algorithm Input

- Updates to `IDs[dst]` can happen in parallel based on the schedule applied at the call site
- Data flow analysis to identify SHARED and INDEPENDENT variables

G2: Vector Property Analyzer

SHARED | INDEPENDENT

```
func updateEdge(src : Vertex, dst : Vertex)
    IDs[dst] += IDs[src];
end
```

- CASE 1:
  - PUSH iteration - Iterate over sources and PUSH values to destinations
  - Vertex Based - Each vertex is iterated on by a single thread

G2: Vector Property Analyzer

	SHARED	INDEPENDENT
`func updateEdge(src : Vertex, dst : Vertex)`	`dst`	`src`
`  IDs[dst] += IDs[src];`	`dst`	`src`
`end`		

- CASE 1:
  - PUSH iteration - Iterate over sources and PUSH values to destinations
  - Vertex Based - Each vertex is iterated on by a single thread

G2: Vector Property Analyzer

	SHARED	INDEPENDENT
`func updateEdge(src : Vertex, dst : Vertex)`	`dst`	`src`
`  IDs[dst] += IDs[src];`	**`dst`**	`src`
`end`		

- CASE 1:
  - PUSH iteration - Iterate over sources and PUSH values to destinations
  - Vertex Based - Each vertex is iterated on by a single thread

G2: Vector Property Analyzer

	SHARED	INDEPENDENT
`func updateEdge(src : Vertex, dst : Vertex)`	`dst`	`src`
`  atomicSum(&IDs[dst], IDs[src]);`	**`dst`**	`src`
`end`		

- CASE 1:
  - PUSH iteration - Iterate over sources and PUSH values to destinations
  - Vertex Based - Each vertex is iterated on by a single thread

G2: Vector Property Analyzer

	SHARED	INDEPENDENT
`func updateEdge(src : Vertex, dst : Vertex)`	`src`	`dst`
`  IDs[dst] += IDs[src];`	`src`	`dst`
`end`		

- CASE 2:
  - PULL iteration - Iterate over destinations and PULL values from sources
  - Vertex Based - Each vertex is iterated on by a single thread

G2: Vector Property Analyzer

	SHARED	INDEPENDENT
`func updateEdge(src : Vertex, dst : Vertex)`	`src`	`dst`
`  IDs[dst] += IDs[src];`	`src`	**`dst`**
`end`		

- CASE 2:
  - PULL iteration - Iterate over destinations and PULL values from sources
  - Vertex Based - Each vertex is iterated on by a single thread

G2: Vector Property Analyzer

	SHARED	INDEPENDENT
`func updateEdge(src : Vertex, dst : Vertex)`	`src, dst`	
`  IDs[dst] += IDs[src];`	`src, dst`	
`end`		

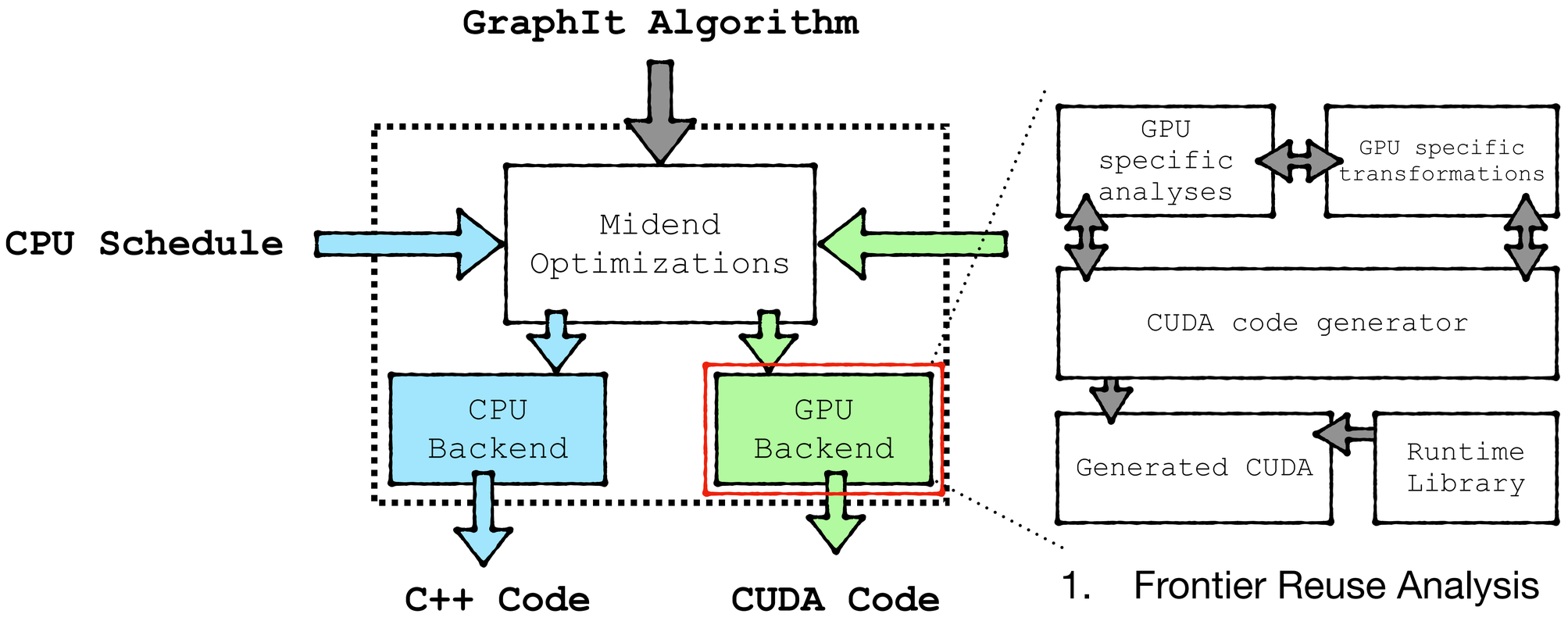
- CASE 3:
  - PUSH iteration - Iterate over sources and PUSH values to destinations
  - TWC Load Balance - Each vertex is iterated on by multiple threads

G2: Vector Property Analyzer

	SHARED	INDEPENDENT
`func updateEdge(src : Vertex, dst : Vertex)`	`src, dst`	
`  atomicSum(&IDs[dst], IDs[src]);`	`src, dst`	
`end`		

- CASE 3:
  - PUSH iteration - Iterate over sources and PUSH values to destinations
  - TWC Load Balance - Each vertex is iterated on by multiple threads

G2: GPU Extensions for GraphIt



1. Frontier Reuse Analysis
2. Kernel Fusion Transformations

The G2 Domain Specific Compiler

G2: Frontier Reuse Analysis

```
...
while (frontier.getVertexSetSize() != 0)
    output = edges.from(frontier).to(toFilter).applyModified(updateEdge, parent);
    delete frontier;
    frontier = output;
  end
...
```

```
while (frontier.getVertexSetSize() != 0) {
    cudaMalloc(output, ...);
    ApplyModified<<<, >>>(frontier, output, …);
    ...
    cudaFree(frontier);
    frontier = output;
}
```

Allocations and deletions on GPUs are costly unlike CPUs

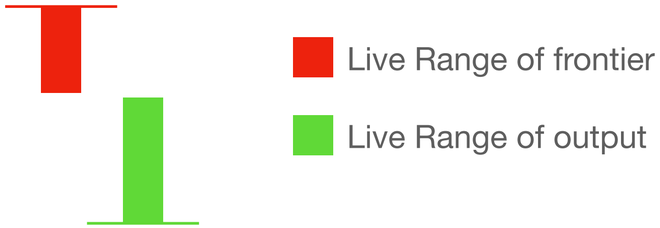
- Do we really need to allocate and free every round?
- We should reuse the memory that was allocated for `frontier`
- Is it always safe to do so?
  - What if the frontier is used again?

Liveness Analysis!

G2: Frontier Reuse Analysis

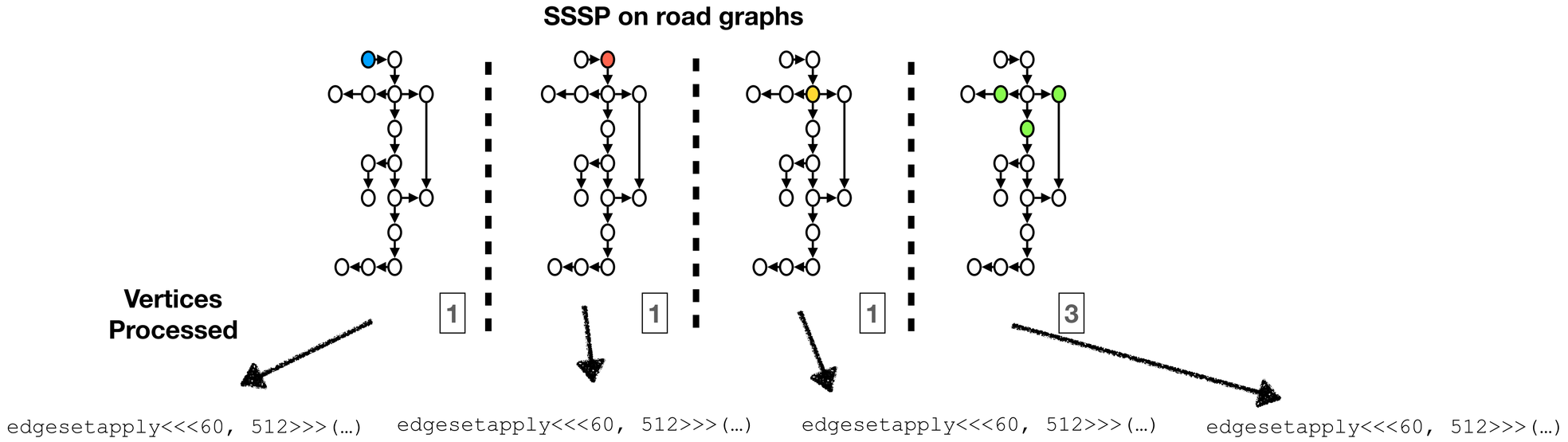
- Constructs live ranges for each `Vertexsubset` variable

```
...
while (frontier.getVertexSetSize() != 0)
    output = edges.from(frontier)…applyModified(…);
    delete frontier;
    frontier = output;
end
...
```



- `frontier`'s last use is at `applyModified` before it is deleted
- `output` is created at the same `applyModified`
- Disjoint live ranges → Reuse memory

G2: Kernel Fusion



Limited amount of work in each iteration!
Kernel Launch overhead dominates

- Launch a single CUDA kernel for the entire `while` loop!!
- Is the Transformation Valid? How does the code need to be changed? …

G2: Kernel Fusion

- Ensure that each operation inside the loop can be run on the GPU
  - Allocations?? Frontier Reuse Analysis!

```
...
while (frontier.getVertexSetSize() != 0)
    output = edges.from(frontier).to(toFilter).applyModified(updateEdge, parent); // FrontierReusable = True;
    frontier = output;
  end
...
```

G2: Kernel Fusion

- Ensure that each operation inside the loop can be run on the GPU
  - Allocations?? Frontier Reuse Analysis!

```
...
while (frontier.getVertexSetSize() != 0)
    output = edges.from(frontier).to(toFilter).applyModified(updateEdge, parent); // FrontierReusable = True;
    frontier = output;
  end
...
```

  - Graph loading
  - Transpose/Blocking
  - Input/Output

G2: Kernel Fusion

Copying state to-and-fro

- Analysis pass to identify local variables used inside the while loop
- Transform the CUDA kernel to copy the variables (host and device)

Prologue Insertion

```
VertexFrontier __device__ fused_6_frontier;
int32_t __device__ fused_6_round;
VertexFrontier __device__ fused_6_output;
VertexFrontierList __device__ fused_6_frontier_list;

void __global__ fused_kernel_body_6(void) {
    grid_group _grid = this_grid();
    int32_t _thread_id = threadIdx.x + blockIdx.x * blockDim.x;
    auto __local_frontier = fused_6_frontier;
    auto __local_round = fused_6_round;
    auto __local_output = fused_6_output;
    auto __local_frontier_list = fused_6_frontier_list;
    …
```

Epilogue Insertion

```
…
if (_thread_id == 0) {
    fused_6_frontier = __local_frontier;
    fused_6_round = __local_round;
    fused_6_output = __local_output;
    fused_6_frontier_list = __local_frontier_list;
}
```

Host Side

```
cudaMemcpyToSymbol(fused_6_round, &round, ...
cudaMemcpyToSymbol(fused_6_frontier_list, &frontier_list, ...
cudaLaunchCooperativeKernel(fused_kernel_body_6, ...
cudaMemcpyFromSymbol(&frontier, fused_6_frontier, ...
cudaMemcpyFromSymbol(&round, fused_6_round,  ...
```

G2: Kernel Fusion

Loop Transformations and Synchronization

```
cudaLaunchCooperativeKernel(fused_kernel_body_6, 80, 512, gpu_runtime::no_args)
```

Multiplexing threads? Synchronization?

```
void __global__ fused_kernel(…) {

  …

  vertices.apply(thread_id);




  …

}
```

Total required threads = graph.vertices

G2: Kernel Fusion

Loop Transformations and Synchronization

```
cudaLaunchCooperativeKernel(fused_kernel_body_6, 80, 512, gpu_runtime::no_args)
```

Multiplexing threads? Synchronization?

```
void __global__ fused_kernel(…) {

  …

  for (v_thread_id = thread_id; v_thread_id < graph.vertices; v_thread_id + 80 * 512) {
    vertices.apply(v_thread_id);
    __sync_threads();
  }

  …

}
```

Total required threads = 80 * 512

G2: Kernel Fusion

Loop Transformations and Synchronization

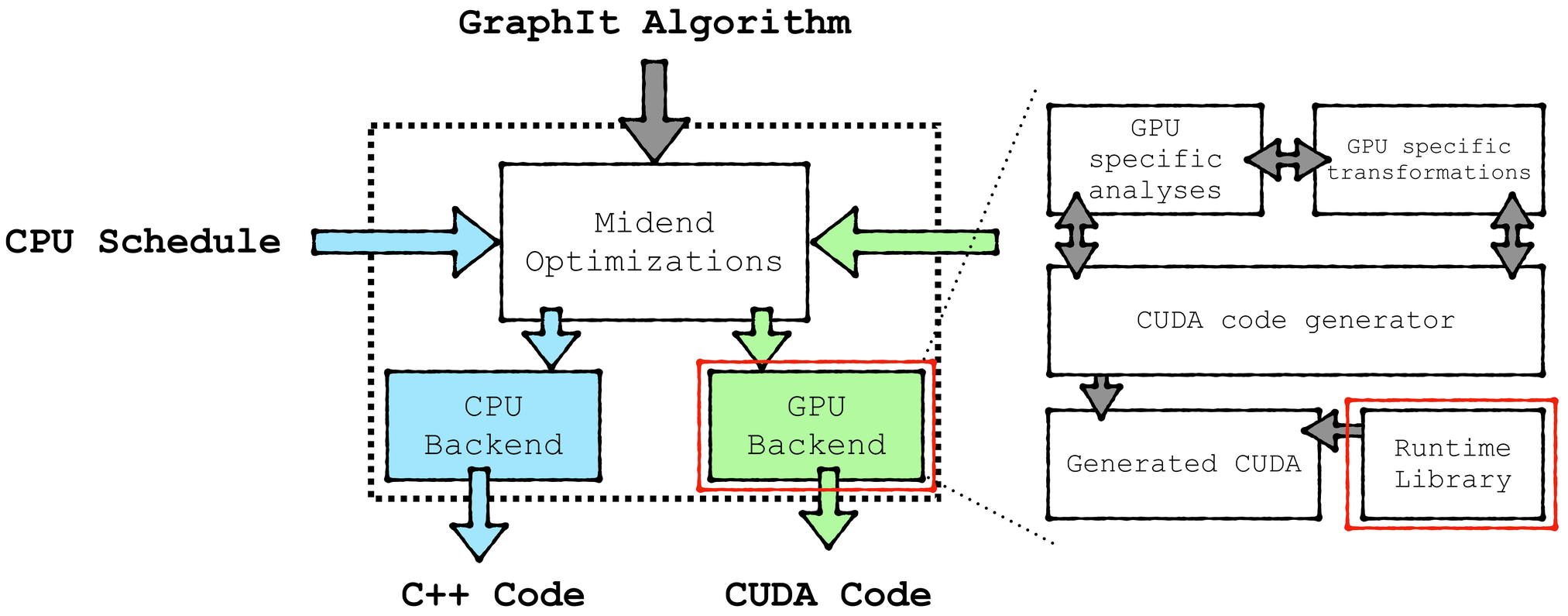
```
cudaLaunchCooperativeKernel(fused_kernel_body_6, 80, 512, gpu_runtime::no_args)
```

Multiplexing threads? Synchronization?

```
void __global__ fused_kernel(…) {

  …

  for (v_thread_id = thread_id; v_thread_id < graph.vertices; v_thread_id + 80 * 512) {
    vertices.apply(v_thread_id);
    __sync_threads();
  }

  this_grid().sync();

  for (v_thread_id = thread_id; v_thread_id < graph.vertices; v_thread_id + 80 * 512)
  …

}
```

Total required threads = 80 * 512

G2: GPU Extensions for GraphIt



The G2 Domain Specific Compiler

G2: GPU Extensions for GraphIt

```
// TWCE LOAD BALANCE FUNCTIONS
#define STAGE_1_SIZE (8)
#define WARP_SIZE (32)
template <typename EdgeWeightType, void load_balance_payload (..), typename AccessorType, bool src_filter(int32_t)>

static void __device__ TWCE_load_balance(GraphT<EdgeWeightType> graph, VertexFrontier input_frontier, VertexFrontier
output_frontier, unsigned int cta_id, unsigned int total_cta) {
..
```

Templated Load Balance Library

```
void __global__ initialize_frontier_all(VertexFrontier frontier) {
    for (int32_t idx = threadIdx.x + blockIdx.x * blockDim.x; idx < frontier.max_num_elems; idx += blockDim.x * gridDim.x)
        frontier.d_sparse_queue_input[idx] = idx;
    if (threadIdx.x + blockIdx.x * blockDim.x == 0) {
        frontier.d_num_elems_input[0] = frontier.max_num_elems;
   }
}
```

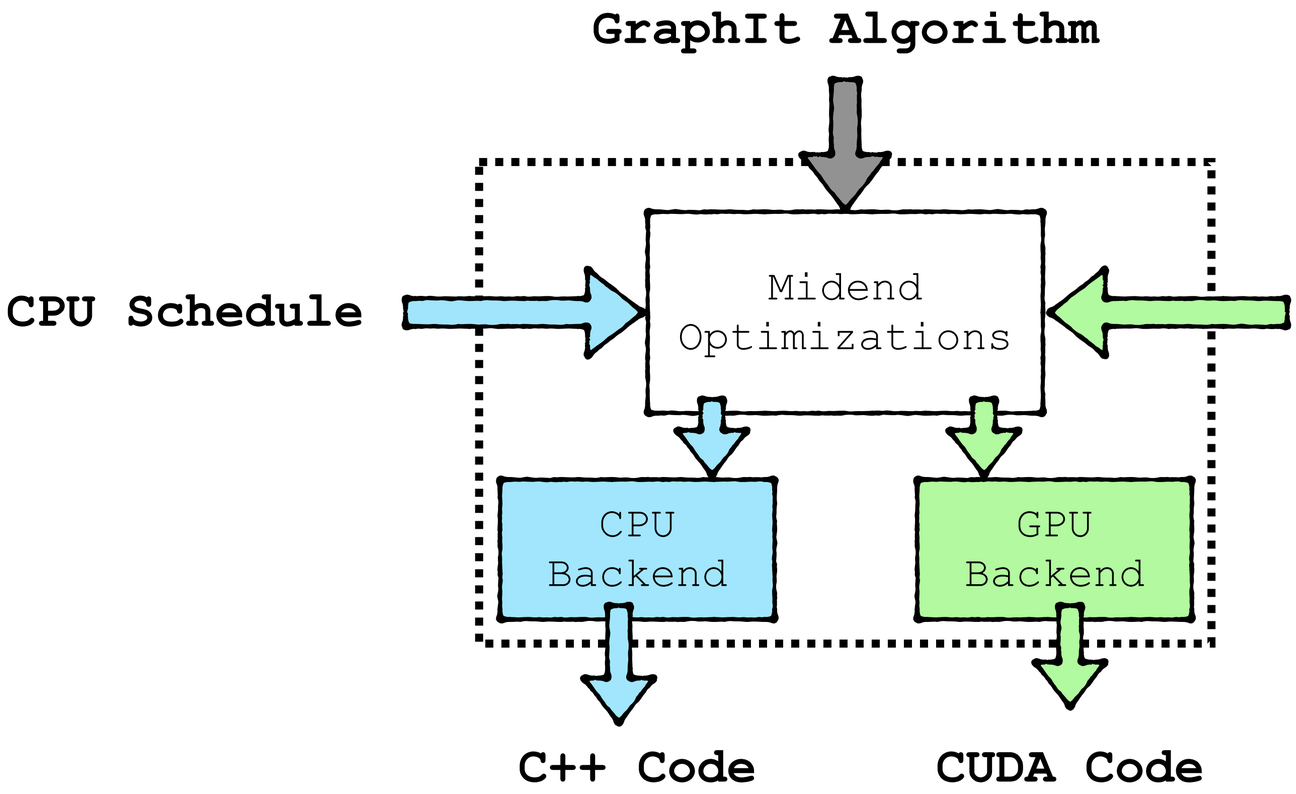
Vertexset management

```
void __device__ updatePriorityMin(GPUPriorityQueue<PriorityT_> * device_gpq,  PriorityT_ new_priority, VertexFrontier
output_frontier, int32_t node){
    bool output = gpu_runtime::writeMin(&(device_gpq->device_priorities_[node]), new_priority);
    if (device_gpq->device_priorities_[node] >= (device_gpq->current_priority_ + device_gpq->delta_)) return;
    if (output){
        enqueueVertexBytemap(output_frontier.d_byte_map_output, output_frontier.d_num_elems_output, node);
    }
}
```

Priority Queue management

The G2 Domain Specific Compiler

G2: GPU Extensions for GraphIt




```
schedule:
  SimpleGPUSchedule s1;
  s1.configLoadBalance(ETWC);
  s1.configDeduplication(DISABLED);
  s1.configFrontierCreation(FUSED);
  s1.configDirection(PUSH);

  program->applyGPUSchedule("s0:s1", s1);

  SimpleGPUSchedule s0;
  s0.configKernelFusion(ENABLED);

  program->applyGPUSchedule("s0", s0);
```

The G2 Domain Specific Compiler

Performance evaluations

3 graph frameworks (Gunrock, GSWITCH, Sep-Graph)

5 algorithms - Breadth First Search, Single Source Shortest Path(Delta Stepping), Connected Components, Betweenness Centrality and PageRank

9 datasets

2 Generations of NVIDIA GPUs (Pascal and Volta)

- **Gunrock**: Y. Wang, Y. Pan, A. Davidson, Y. Wu, C. Yang, L. Wang, M. Osama, C. Yuan, W. Liu, A. T. Riffel et al., “Gunrock: GPU graph analytics,” ACM Transactions on Parallel Computing (TOPC), vol. 4, no. 1, p. 3, 2017
- **GSWITCH**: K. Meng, J. Li, G. Tan, and N. Sun, “A pattern based algorithmic autotuner for graph processing on GPUs,” in Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming (PPoPP), 2019, pp. 201–213.
- **SEP-Graph**: H. Wang, L. Geng, R. Lee, K. Hou, Y. Zhang, and X. Zhang, “SEPGraph: Finding shortest execution paths for graph processing under a hybrid framework on GPU,” in Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming (PPoPP), 2019, pp. 38–52.

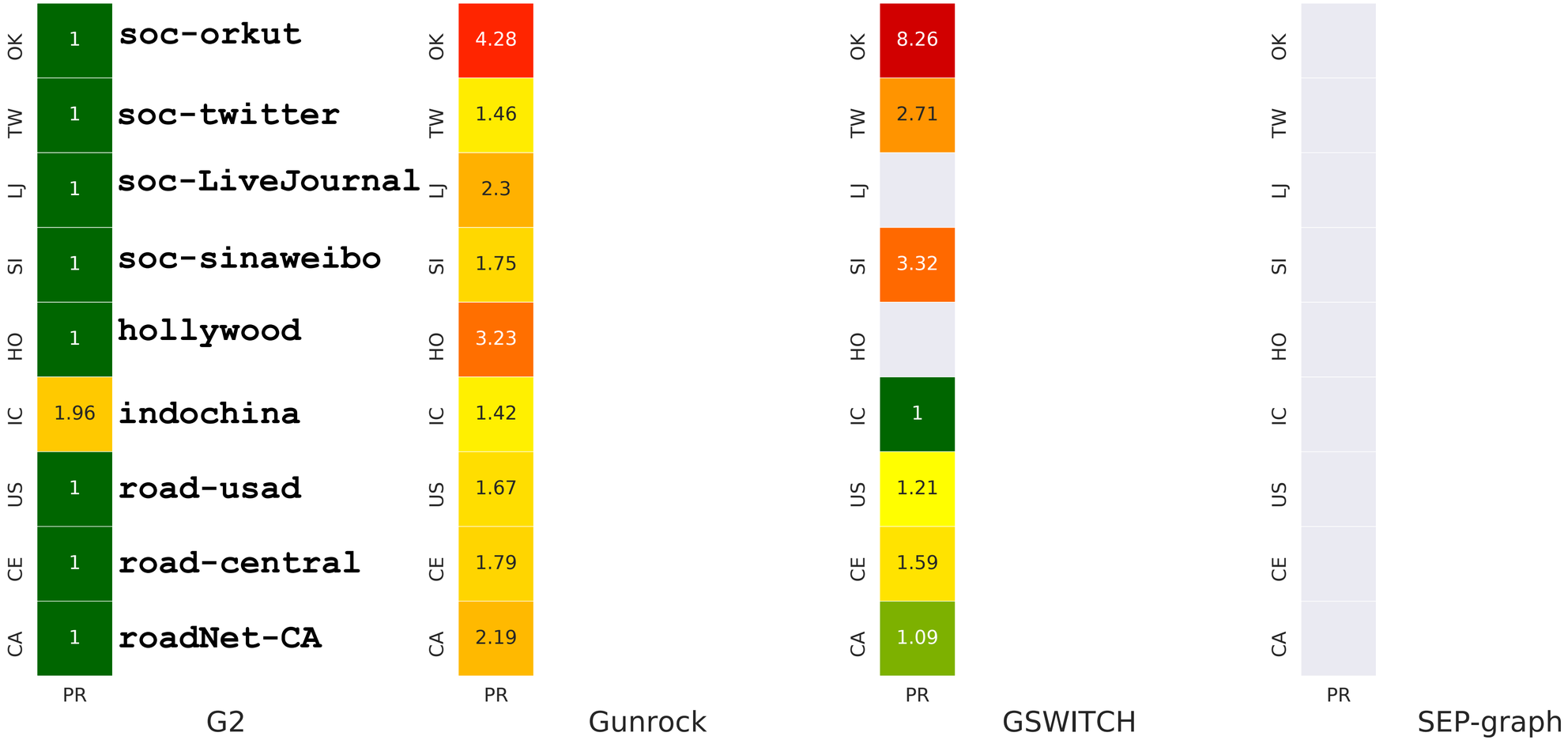
Performance evaluations

Performance comparison of G2 generated code for 5 applications and 9 graphs against SoA graph frameworks Gunrock, GSWITCH, SEP-Graph when run on Titan Xp (Pascal) GPU

G2 Gunrock GSWITCH SEP-graph

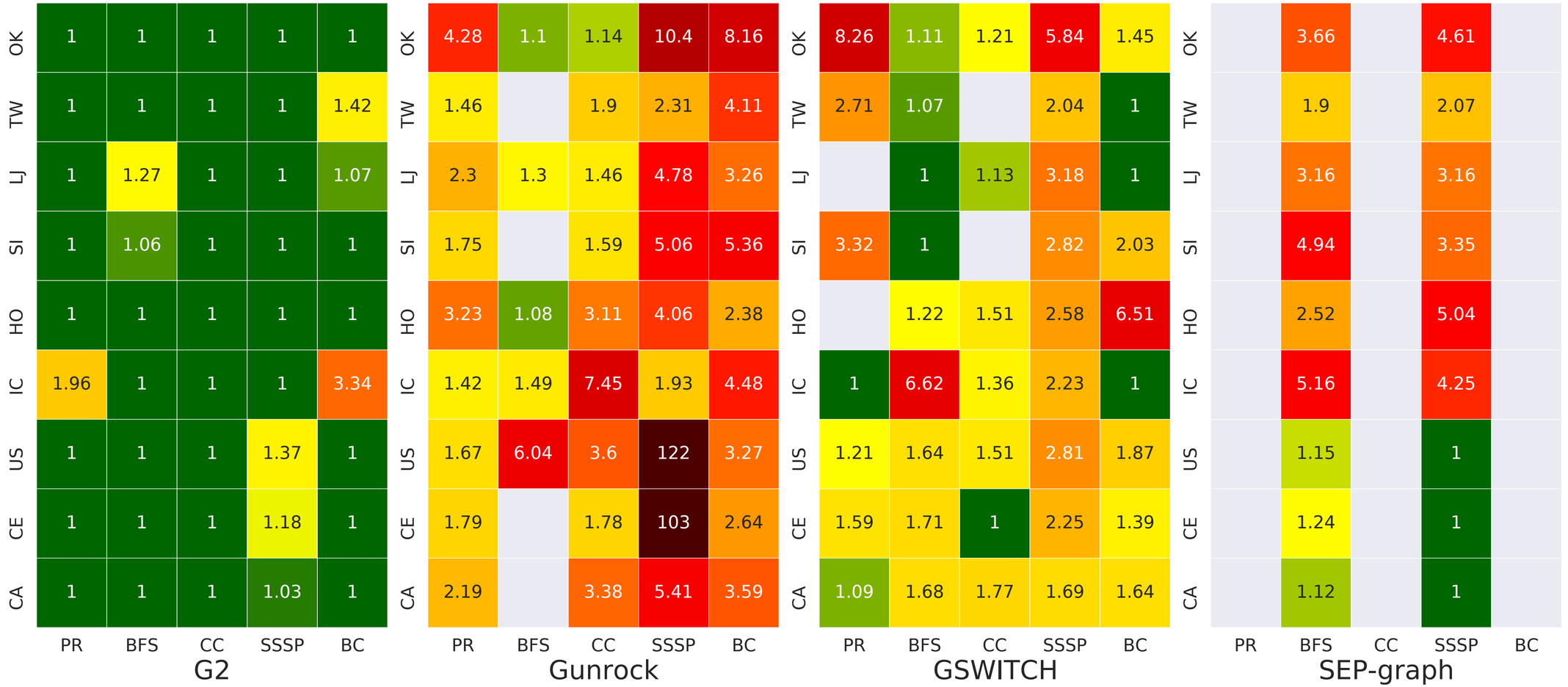
Performance evaluations

Performance comparison of G2 generated code for 5 applications and 9 graphs against SoA graph frameworks Gunrock, GSWITCH, SEP-Graph when run on Titan Xp (Pascal) GPU



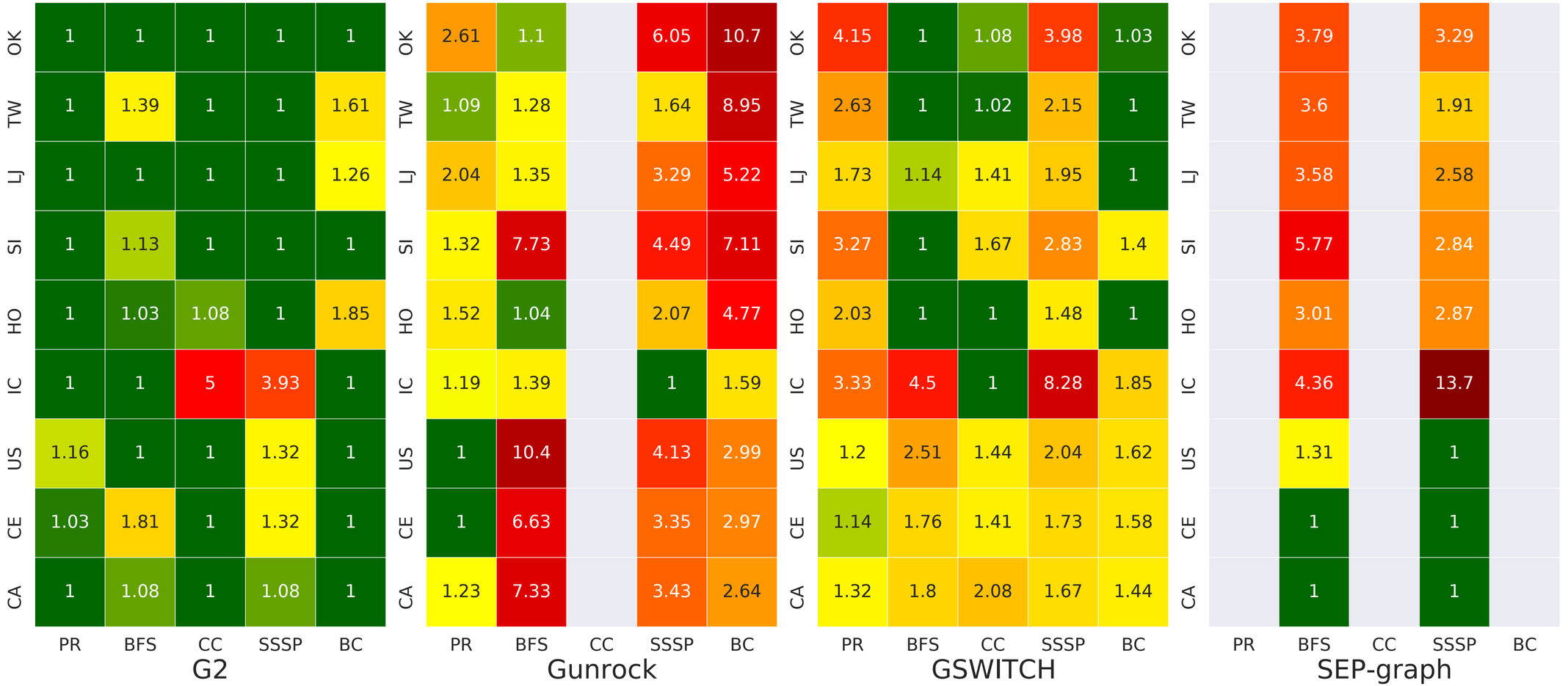
Performance evaluations

Performance comparison of G2 generated code for 5 applications and 9 graphs against SoA graph frameworks Gunrock, GSWITCH, SEP-Graph when run on Titan Xp (Pascal) GPU



Performance evaluations

Performance comparison of G2 generated code for 5 applications and 9 graphs against SoA graph frameworks Gunrock, GSWITCH, SEP-Graph when run on V-100 (Volta) GPU



G2 fastest On 65/90 experiments

Performance evaluations

Comparison of lines of code when programming in each of the SoA frameworks and G2

Application	Gunrock	GSWITCH	SEP-Graph	G2 (Schedule + Algorithm)
Breath First Search	2189	164	481	**66**
PageRank	2207	159	-	**61**
Connected Components	3014	160	-	**62**
Betweenness Centrality	1792	280	-	**128**
SSSP with Delta Stepping	1438	203	473	**50**

The lines of code required to implement each of the 5 algorithms in the SoA graph frameworks: Gunrock, GSWITCH and SEP-Graph and G2. The lines of code for G2 includes both the algorithm and the schedule. "-" indicates that the algorithm is not supported by the framework

Summary

- G2 is an extension to the GraphIt DSL and compiler to generate high-performance CUDA code for graph applications from the same high level representation
- G2 introduces a new scheduling language tailored specifically for GPU related optimizations
- G2 outperforms other state of the art graph frameworks on most applications and graphs while consistently requiring fewer lines of code to implement

G2 is available open-source at
https://graphit-lang.org

This Work Supported By: