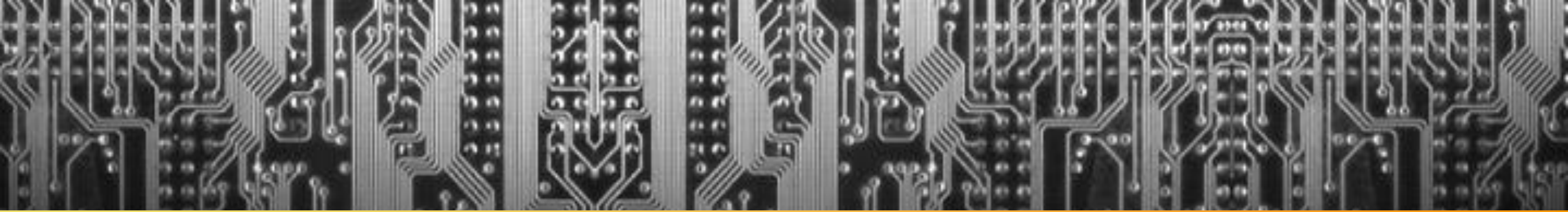


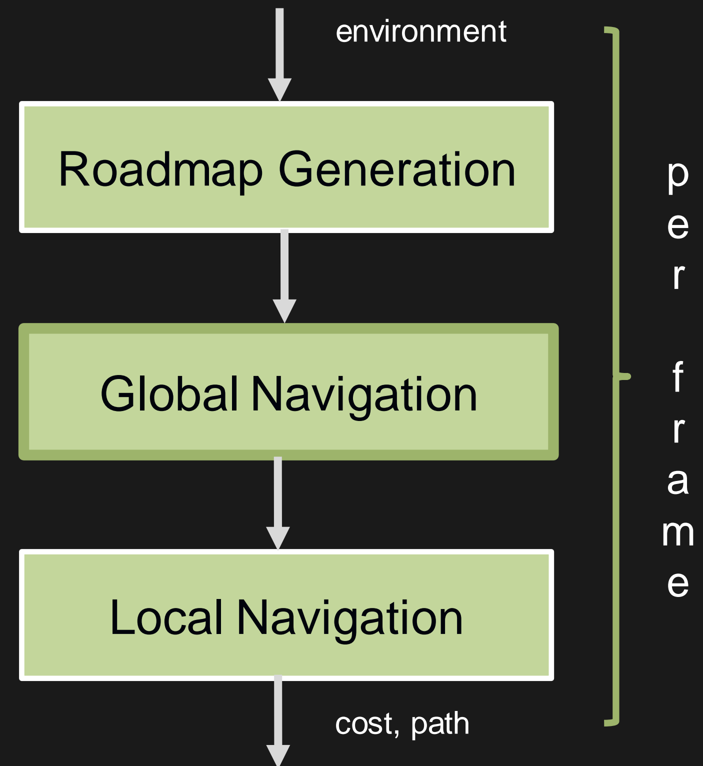
GPU Accelerated Pathfinding

Avi Bleiweiss

NVIDIA Corporation

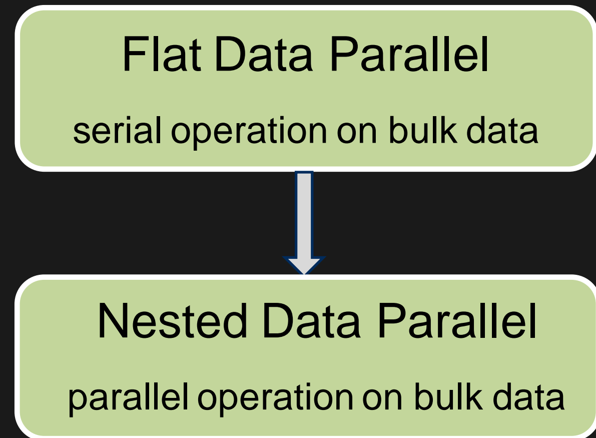
Introduction

- Navigation planning
  - Global and local
- Crowded game scenes
  - Many thousands agents
- Decomposable movement
  - Explicit parallelism
- Dynamic environment



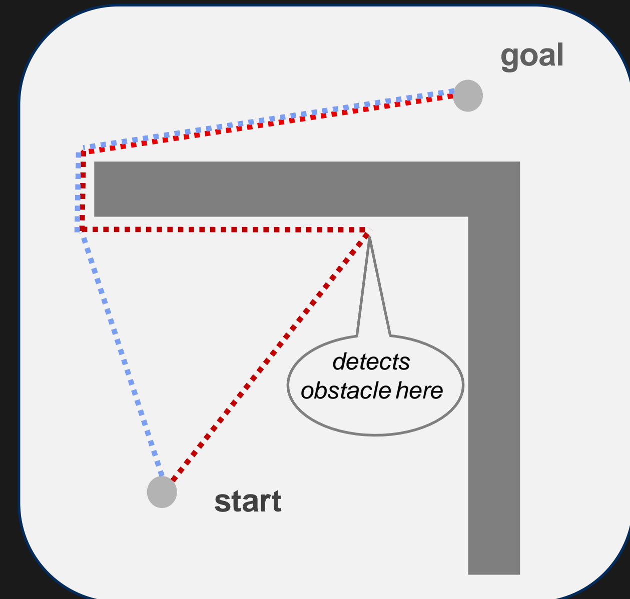
Motivation

- CUDA compute enabling
- Nested data parallelism
- Irregular, divergent algorithms
  - Large thread SIMD challenge
- Extend GPU game computing
  - Core game AI actions



Objective

- Optimally navigate agents
  - From start to goal state
- Roadmap representation
  - Graph data structure
- Parallel, arbitrary search
  - Varying topology complexity
- GPU performance scale



gh08

Outline

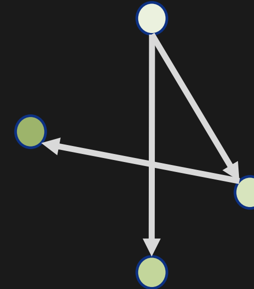
- Algorithm
- Implementation
- Performance
- Futures

gh08

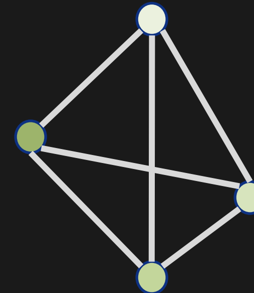
Algorithm

Graph

- Linked set of nodes, edges
  - *G* = {*N*, *E*}
- Dense or sparse
  - Edges (E) to nodes (N^2) ratio
- Directed or undirected
  - Ordered, unordered node pairs
- Consistent data structure



sparse, directed



dense, undirected

Data Structure

- Adjacency matrix
  - Intuitive edge presence
  - Wasteful for sparse graphs *O*(N^2)
- Adjacency lists
  - Immediate node indices
  - Compact storage *O*(N+E)
- Roadmap sparse graph
  - Adjacency lists default

	0	1	2	3
0	0	1	1	0
1	0	0	0	1
2	0	0	0	0
3	0	0	0	0

adjacency matrix

0	1	2
1	3	

adjacency lists

Search

- Feasibility and optimality
- Planning in state space
  - Unvisited, dead, or alive
- Priority queue alive states
- Cost based state
- Running time complexity
  - Worse than linear

```
1:   Q.Insert(nS) and mark nS as visited
2:   while Q not empty do
3:     n ← Q.Extract()
4:     if(n == nG) return SUCCESS
5:     for all u є U(n) do
6:       n' ←f(n, u)
7:       if n' not visited then
8:         Mark n' visited
9:         Q.Insert(n')
10:      else
11:        Resolve duplicate n'
12:  return FAILURE
```

Forward Search Algorithm Template

- alive nodes are placed on a priority queue Q
- n_s and n_g are start and goal positions, respectively
- u is an action in a list of actions U
- f(n, u), state transition function
- n is current node and n' the next adjacent node.

Algorithms

- Cost based search
- Priority queue sort function
- Search properties:

Search	Start	Goal	Heuristic	Optimal	Speed
Best First	no	yes	yes	no	fair
Dijkstra	yes	no	no	yes	slow
A*	yes	yes	yes	yes°	fast

° assumes admissible heuristic

- Dijkstra, A* without heuristic

gh08

Heuristic

- Admissible = optimistic
  - Never overestimate cost-to-goal
- A* with admissible heuristic
  - Guarantees optimal path
- Narrows search scope
- Suboptimal, weighted heuristic
  - Quality vs. efficiency tradeoff

Function	Definition
Manhattan	w * abs($\mathbf{n}_g$ - $\mathbf{n}$)
Diagonal	w * max(abs($\mathbf{n}_g$ - $\mathbf{n}$))
Euclidian	w * sqrtf(square($\mathbf{n}_g$ - $\mathbf{n}$))

n – position vector state

A*

- Irregular, highly nested
- Priority queue element
  - {node index, cost } pair
- Memory bound
  - Extensive scatter, gather
- Low arithmetic intensity
  - Embedded in heuristic
- Unrolling inner loop

```
 1: f = priority queue element {node index, cost}
 2: F = priority queue containing initial f (0,0)
 2: G = g cost set initialized to zero
 3: P, S = pending and shortest nullified edge sets
 4: n = closest node index
 5: E = node adjacency list
 6: while F not empty do
 7:    n ← F.Extract()
 8:    S[n] ← P[n]
 9:    if n is goal then return SUCCESS
10:    foreach edge e in E[n] do
11:       h ← heuristic(e.to, goal)
12:       g ← G[n] + e.cost
13:       f ← {e.to, g + h}
14:       if not in P or g < G[e.to] and not in S then
15:          F.Insert(f)
16:          G[e.to] ← g
17:          P[e.to] ← e
18: return FAILURE
```

Cost notation:

- g(n): cost from start to node n
- h(n): heuristic cost from n to goal
- f(n, cost): combined cost of g(n) and h(n)

Implementation

Software

- Game AI workloads
- GPU, CPU invocation paths
- Orthogonal multi core semantics
- CUDA graceful multi launch
- Scalar C++, SIMD intrinsics (SSE)

gh08

Tradeoffs

- Shared roadmap caching
- Working set coalesced access
- Efficient priority queue operations
- Divergent kernel parallel execution
- CUDA profiler for optimization

gh08

Roadmap Textures

- Linear device memory
  - Texture reference binding
- Flattened edge list
  - With adjacency directory
- Adjacency list cacheable
- Loop control direct map
- 2 or 4, 32 bit components

Node			
id	position.x	position.y	position.z

Edge			
from	to	cost	reserved

Adjacency	
offset	offset+count

gh08

Working Set

- Thread local storage
  - Global memory regions
- *O*(T*N) storage complexity
  - Node sized arrays
- 4, 8, 16 bytes data structures
- Exceeding available memory

Inputs		
List	Definition	Initialization
Paths	start, goal positions	user defined
G	cost from-start	zero
F	sum of costs from-start, to-goal	zero
P, S	visited, dead node (edge)	zero

O(T)

Outputs		
List	Definition	Initialization
Costs	accumulated path cost	zero
W	subtree, plotted waypoints	zero

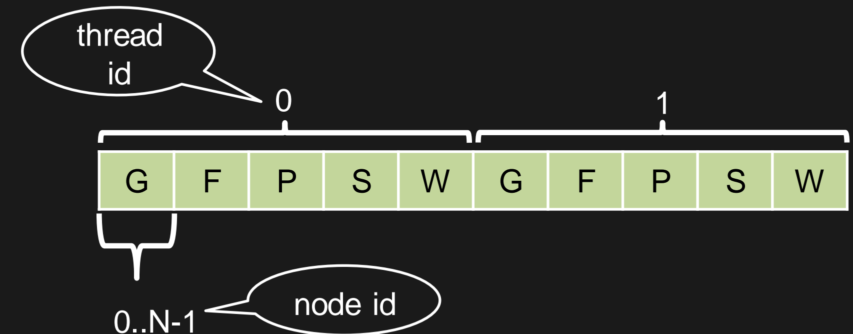
T - threads, N - roadmap nodes

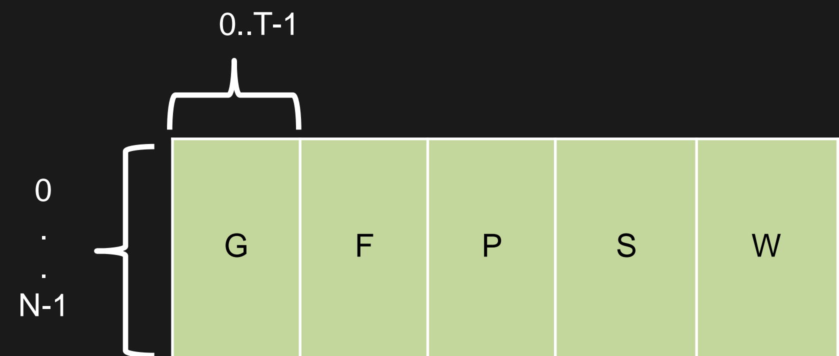
*O(T*N)*

gh08

Coalescing

- Strided memory layout
  - Node id fast axis
- Interleaved organization
  - Thread id running index
- Contiguous memory access
  - Across a thread warp
- Array indexing inexpensive





Priority Queue

- Element pairs
  - Cost, node id
- Fixed size array
- Heap based
  - Logarithmic running cost
- Operation efficiency
  - Insertion, extraction
- Insertions dominates
  - Early success exit

```
1:  __device__ void
2:  insert(CUPriorityQ* pq, CUCost c)
3: {
4:    int i = ++(pq→size);
5:    CUCost* costs = pq→costs;
6:    while(i > 1 && costs[i>>1].cost  >  c.cost) {
7:       costs[i] = costs[i>>1];
8:         i >>= 1;
9:     }
10:     pq→costs[i] = c;
11: }
```

```
1:  __device__ CUCost
2:  extract(CUPriorityQ* pq)
3: {
4:    CUCost cost;
5:    if(pq→size >= 1) {
6:        cost = pq→costs[1];
7:      pq→costs[1] = pq→costs[pq→size--];
8:        heapify(pq);
9:    }
10:   return cost;
11: }
```

gh08

Execution

- CUDA launch scope
  - Consult device properties
- An agent constitutes a thread
- One dimensional grid of
  - One dimension thread blocks
- Kernel resource usage
  - 20 registers
  - 40 shared memory bytes

CUDA Occupancy Tool Data	
Threads per block	128
Registers per block	2560
Warps per block	4
Threads per multiprocessor	384
Thread blocks per multiprocessor	3
Thread blocks per GPU (8800 GT)	42

gh08

Performance

Experiments

- Roadmap topology complexity (RTC)
- Fixed, varying agent count
- Dijkstra and non weighted, A* search
- SSE, multi core CPU scale
- CUDA interleaved kernel
- GPU timing includes copy

gh08

Benchmarks

Graph	Nodes	Edges	Agents	Blocks
G0	8	24	64	1
G1	32	178	1024	8
G2	64	302	4096	32
G3	129	672	16641	131
G4	245	1362	60025	469
G5	340	2150	115600	904
G6	5706	39156	64–9216	1–72

G0–G5: all pairs

G6: random pairs

- G0–G5: small to moderate RTC (<500 nodes)
- G6: large graph (>5000 nodes)

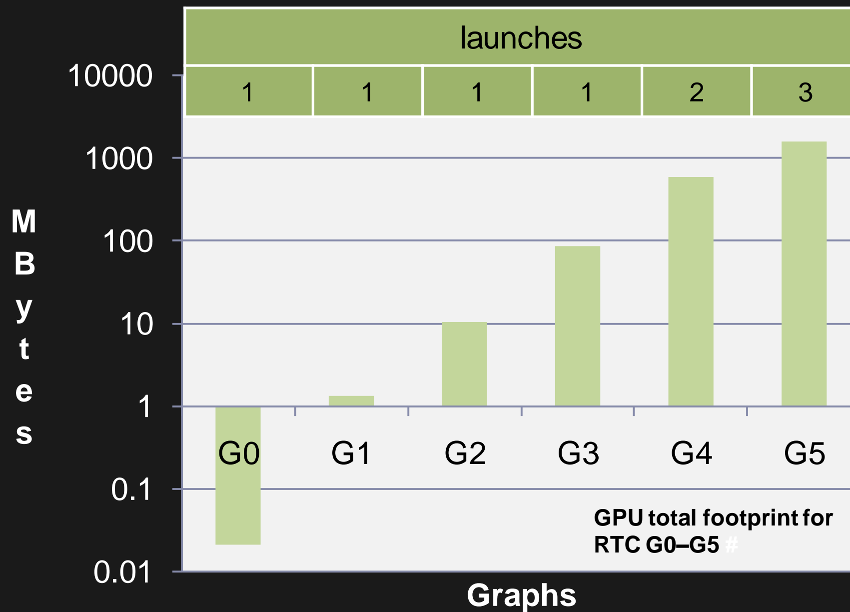
Processors

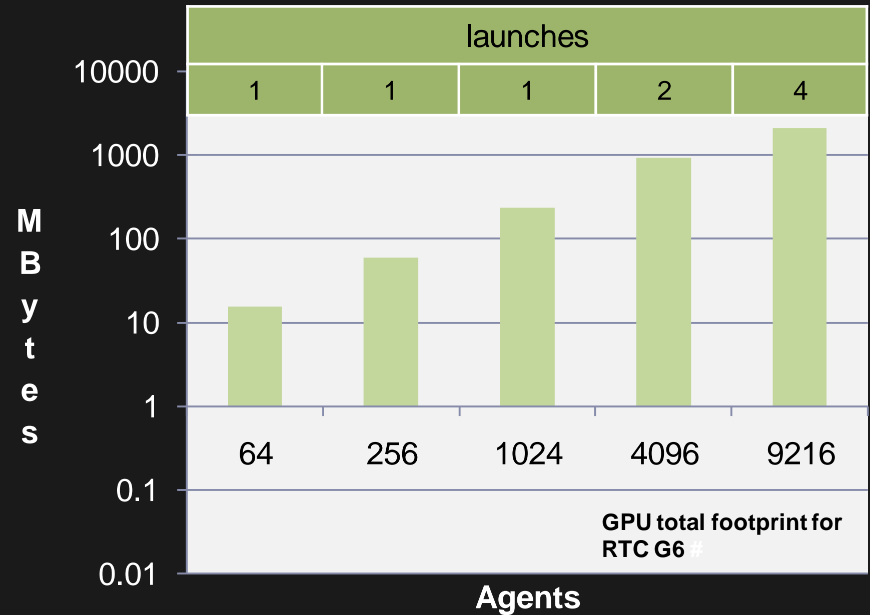
- Processor properties:

Property	Intel Core 2 Duo	AMD Athlon 64 X2	8400 M	8800 GT	GTX 280
Core Clock	2000	2110	400	600	600
Shader Clock	NA	NA	550	1500	1300
Memory Clock	1180	667	400	900	1000
Global Memory	2048	2048	256	512	1024
Memory Bus	64	64	64	256	512
Multiprocessor	1 per core	1 per core	8	14	30

clocks and memory size in millions

Footprint

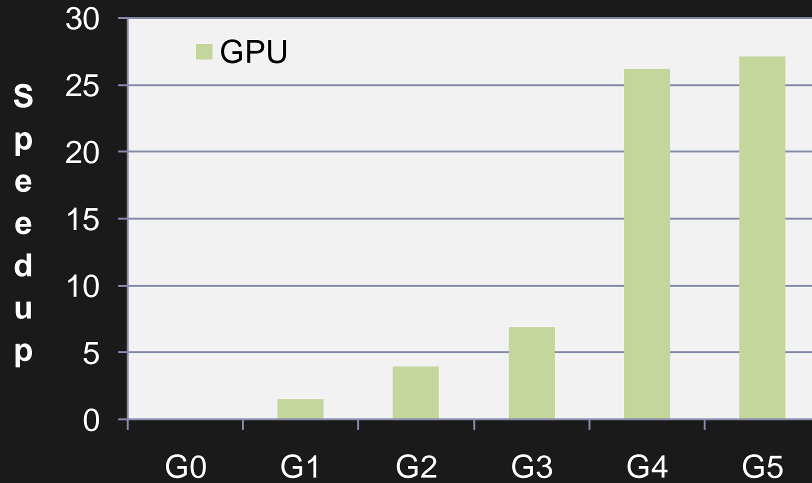




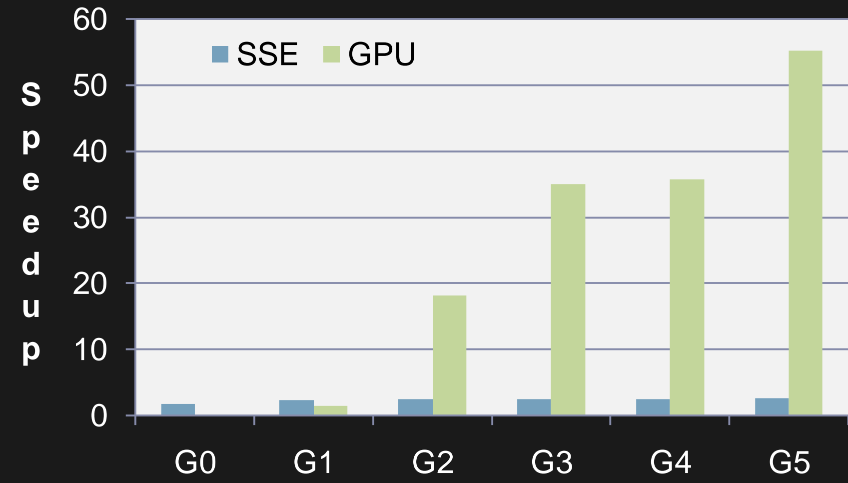
GPU	8800 GT

- Working set *O*(T*N) dominates roadmap *O*(N+E)
- Per thread local 0.33–13.6 KB (G0–G5), 230 KB (G6)

Search



GPU speedup vs. a single core CPU, optimized scalar code for RTC G0–G5, fixed agent #

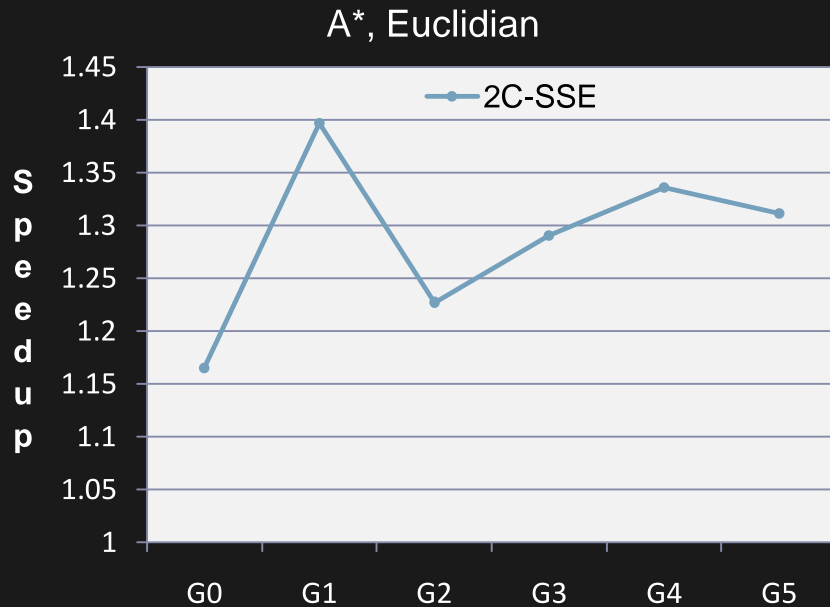


GPU speedup vs. a single core CPU, optimized scalar and SSE code for RTC G0–G5, fixed agent #

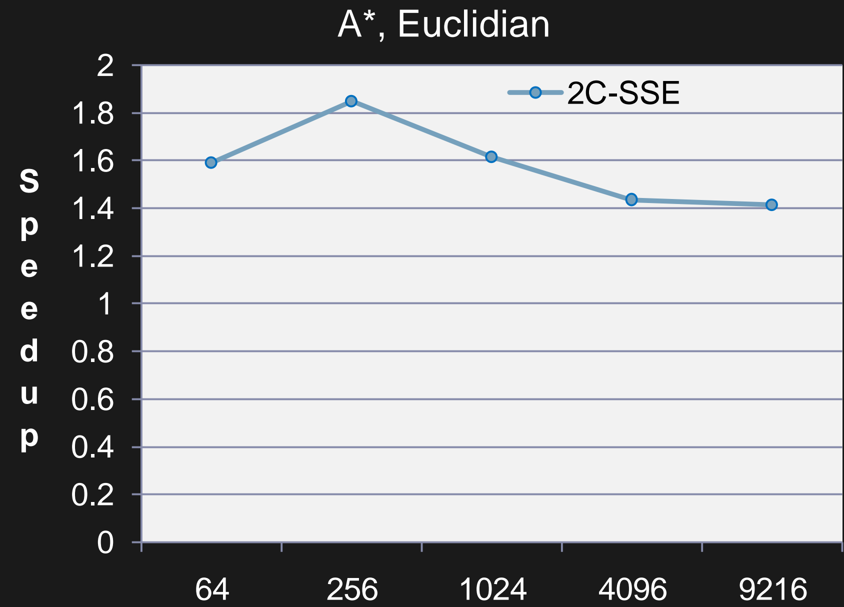
CPU	AMD Athlon 64 X2
GPU	8800 GT

- A* higher arithmetic intensity improves speedup

Multi Core



CPU speedup vs. a single core, SSE optimized code for RTC G0–G5, fixed agent #

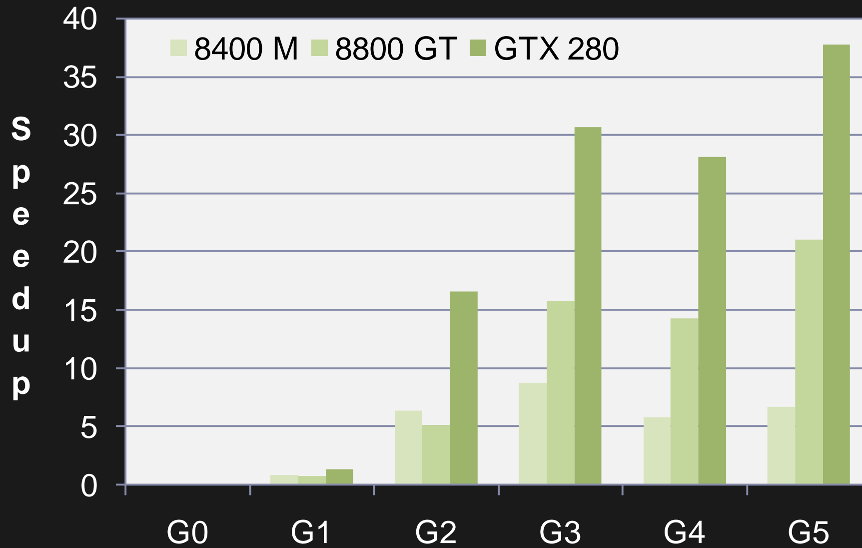


CPU speedup vs. a single core, SSE optimized code for RTC G6, ascending agent #

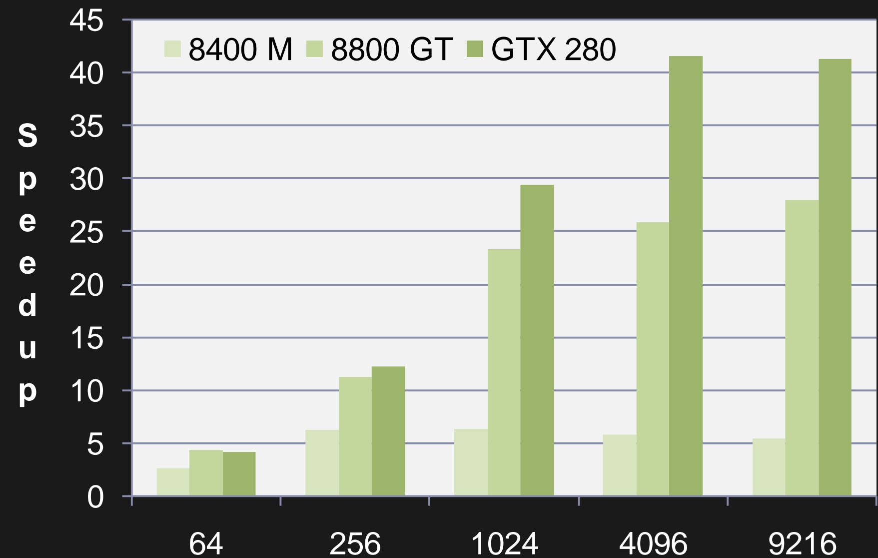
CPU	Intel Core 2 Duo

- Quad core vs. dual core speedup: 1.05X (G0–G5), 1.2X (G6)

Cross GPU



GPU speedup vs. a single core CPU, SSE optimized code for RTC G0–G5, fixed agent #



GPU speedup vs. a single core CPU, SSE optimized code for RTC G6, ascending agent #

CPU	AMD Athlon 64 X2

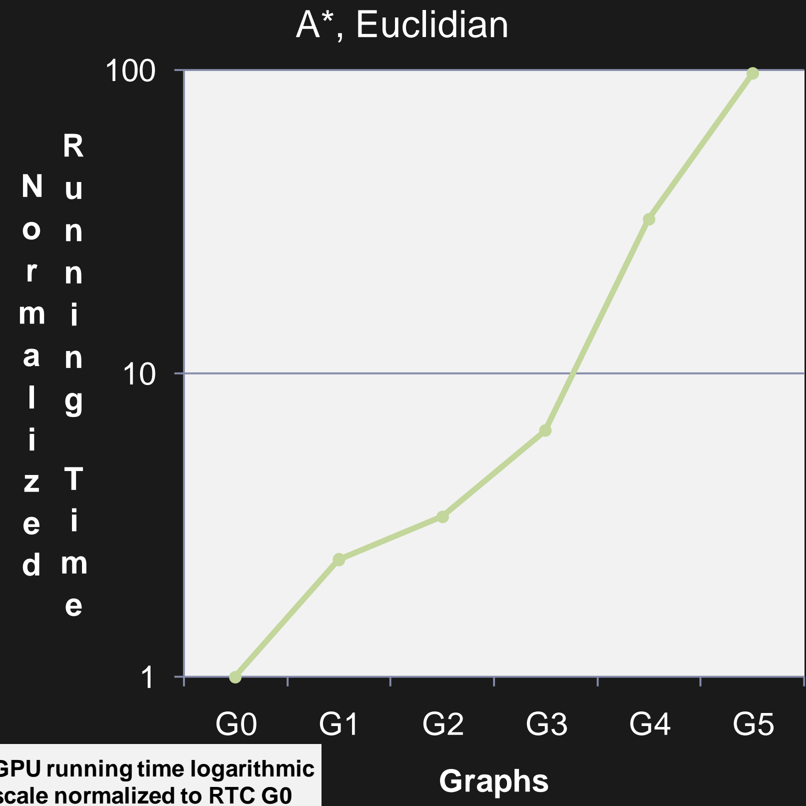
- GTX 280 vs. 8800 GT speedup up to 2X

Running Time

Parameter	G5	G6°
Total running time (seconds)	2.495	6.136
Average search time (seconds)	0.000021	0.000665
Average points per path	12.6576	15.8503

° agent count: 9216

- Unlocked copy overhead
  - Host-to-Device (up to 50%)
  - Device-to-Host (less than 5%)



GPU	8800 GT

Limitations

- Small agent count
- Unlocked copy expensive
  - Pinned memory 1.6X overall speedup
- Software memory coalescing
  - Limited, 1.15X performance scale
- Multi GPU linear scale
  - Replicated roadmap expense
- Weighted A* oscillating

gh08

Futures

Futures

- Working set greedy allocation
  - Dynamic, CUDA kernel malloc
- Global memory caching
- Kernel spawning threads
  - Unrolled A* inner loop
- Realigning agent blocks
- Local navigation

gh08

Conclusions

- Global navigation scalable
- GPU efficient search for
  - Many thousands agents
- Nested data parallelism
  - Evolving GPU opportunity
- GPU preferred platform
  - Integrating core game AI

gh08

Thank You!

Questions?