GPU TECHNOLOGY CONFERENCE

NVIDIA

HARNESSING IRREGULAR PARALLELISM: A CASE STUDY ON UNSTRUCTURED MESHES

Cliff Woolley, NVIDIA

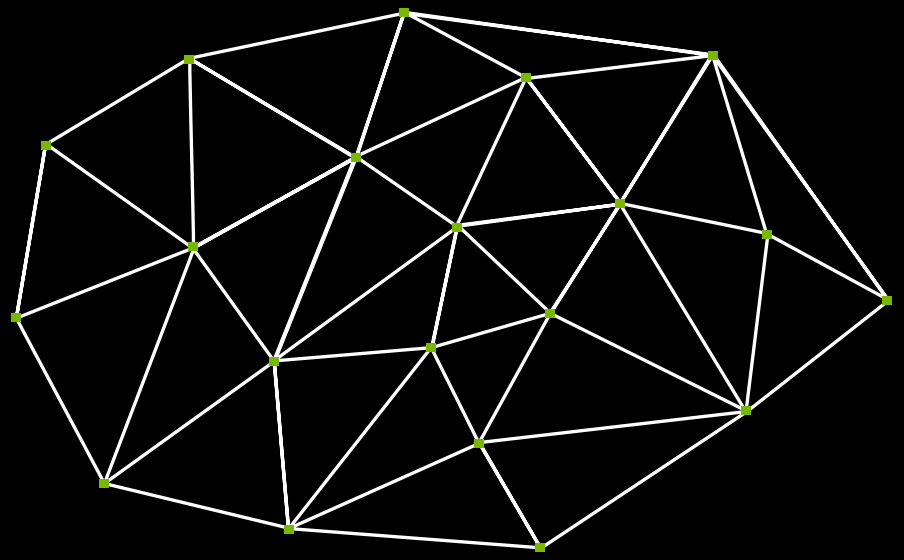
PREFACE

- This talk presents a case study of extracting parallelism in the UMT2013 benchmark for 3D unstructured-mesh photon transport

- Note: You do not need to have a background in photon transport for this talk
  - This talk is about the parallelism, not about the physics
  - Various other kinds of solvers use unstructured meshes, as do many graphical applications; similar ideas might apply to some of those

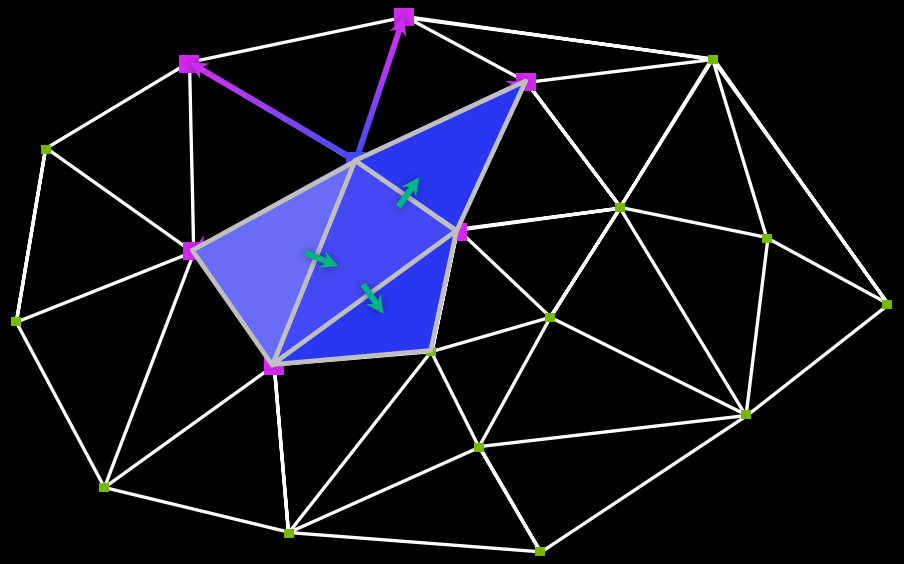
INTRODUCTION: WHAT IS AN UNSTRUCTURED MESH?

- An unstructured mesh (grid) is a spatial decomposition using simple shapes (e.g., triangles, tetrahedra) in an irregular pattern



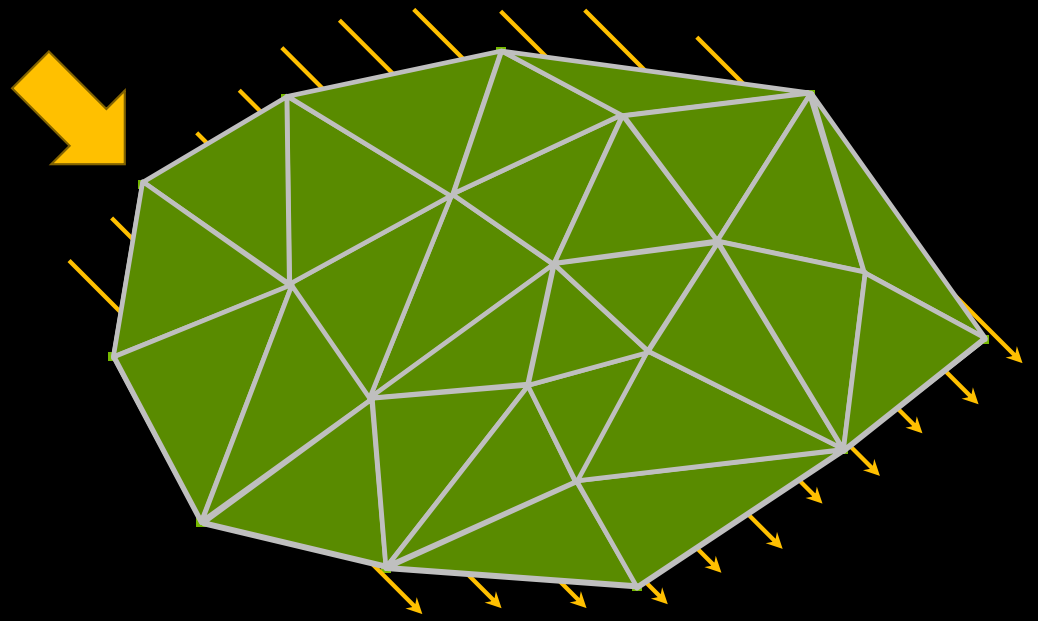
INTRODUCTION: WHAT IS AN UNSTRUCTURED MESH?

- Unstructured methods require an explicit description of the geometry and its connectivity
  - Sometimes the vertices (or the edges) are the important parts
  - Other times the facets/volumes are the important parts



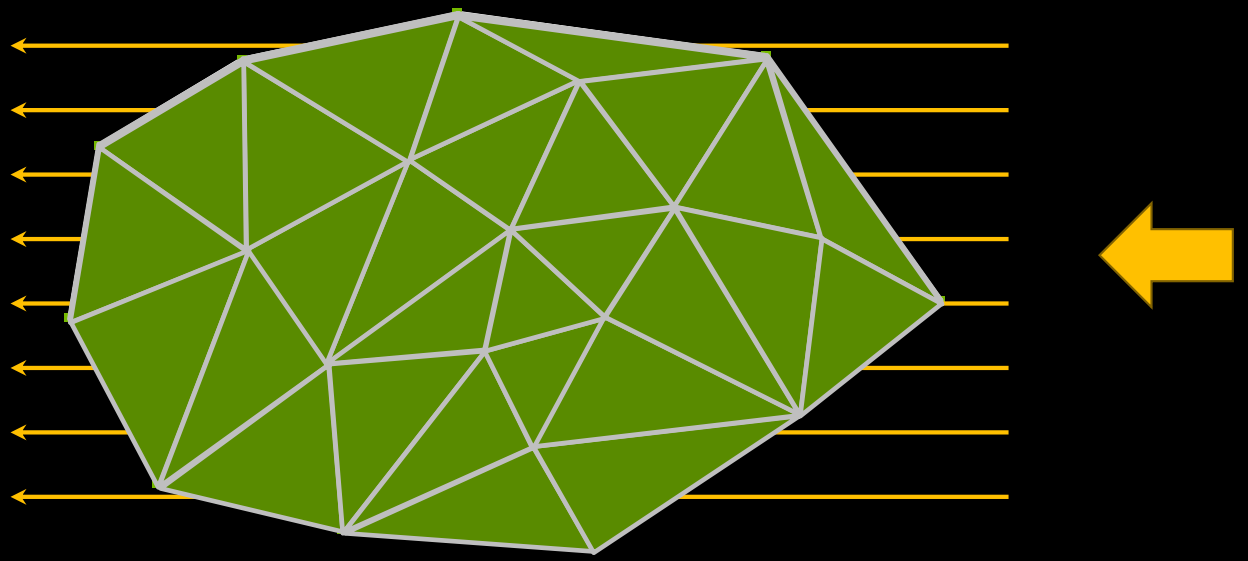
INTRODUCTION: WHY IS THIS CHALLENGING?

- The amount of concurrency we can extract as we sweep across the mesh is irregular
- Traversing an unstructured mesh requires indirect (and likely incoherent) memory addressing



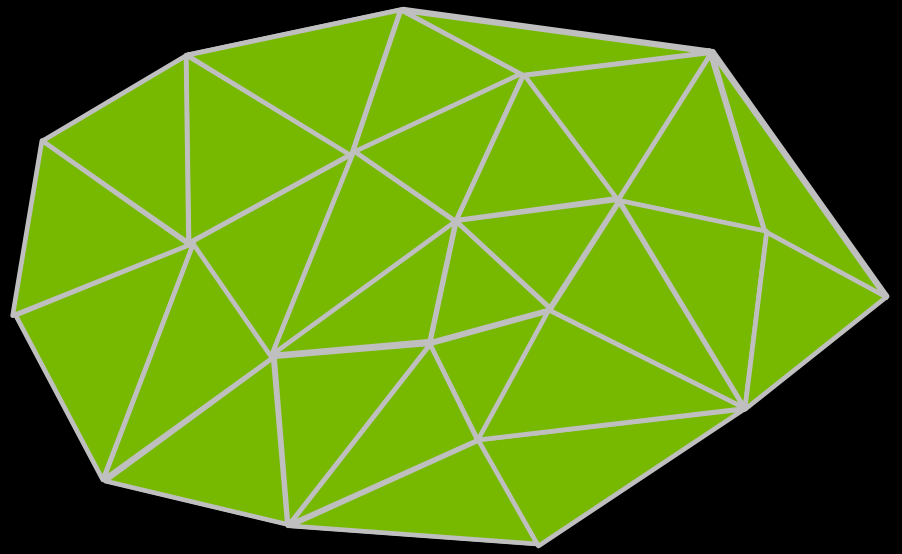
INTRODUCTION: WHY IS THIS CHALLENGING?

- The amount of concurrency we can extract as we sweep across the mesh is irregular
- Traversing an unstructured mesh requires indirect (and likely incoherent) memory addressing



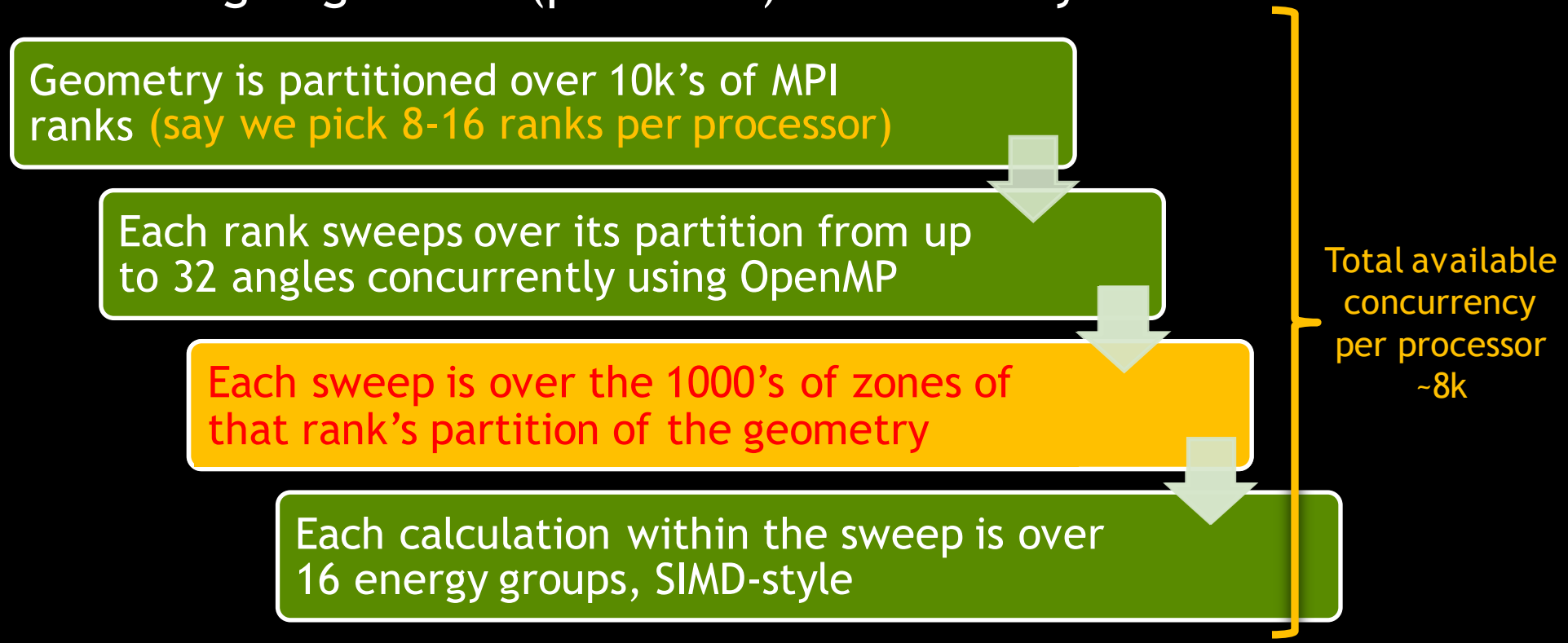
TRADITIONAL PARALLELIZATION OF UNSTRUCTURED MESH SWEEPS

- Partition the mesh into patches (like tiling a structured mesh)
- Each processor sweeps its local patch



PARALLELISM IN UMT2013

- Out of the box, the UMT2013 benchmark presents the following degrees of (potential) concurrency:



SWEEP SCHEDULING IN UMT2013

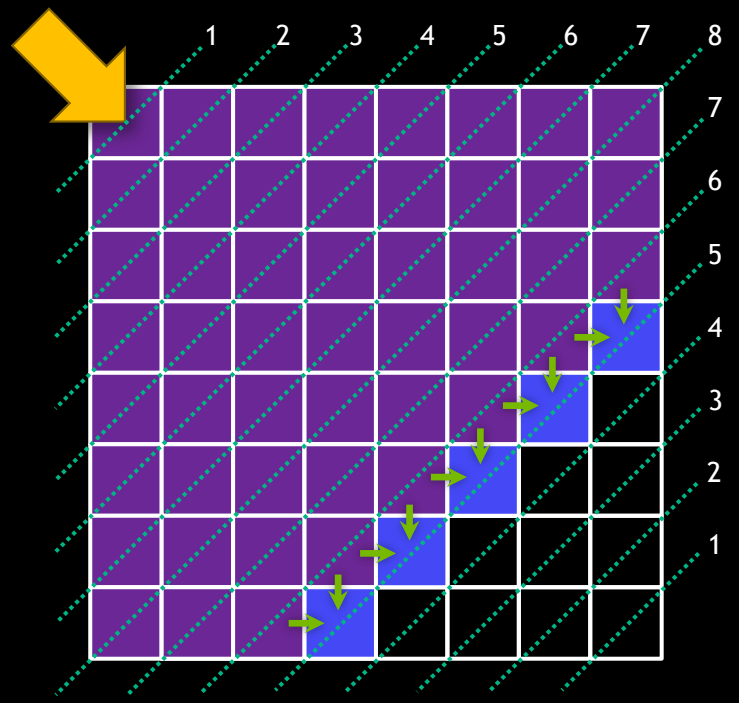
- The sweep order is angle-dependent but pre-scheduled
- Zone indices are ordered (as best as possible) such that zones are processed after zones on which they depend when processing the ordered list sequentially
- UMT2013 makes no attempt to schedule *for parallelism*

STRUCTURED GRID = (MOSTLY) REGULAR PARALLELISM

- To understand the parallelization opportunities for traversal of unstructured meshes, let's look first at structured meshes
- In a structured grid, the number of neighbors each grid cell is known (and generally constant) and addressing the neighbors is straightforward.
- Parallelism is (usually) regular:
  - We can often consider all grid cells concurrently at any given step (or at least all cells along a given row or column)
  - Worst case is when we sweep across the diagonal, but parallelizing these *wavefront* computations through pipelining is well understood

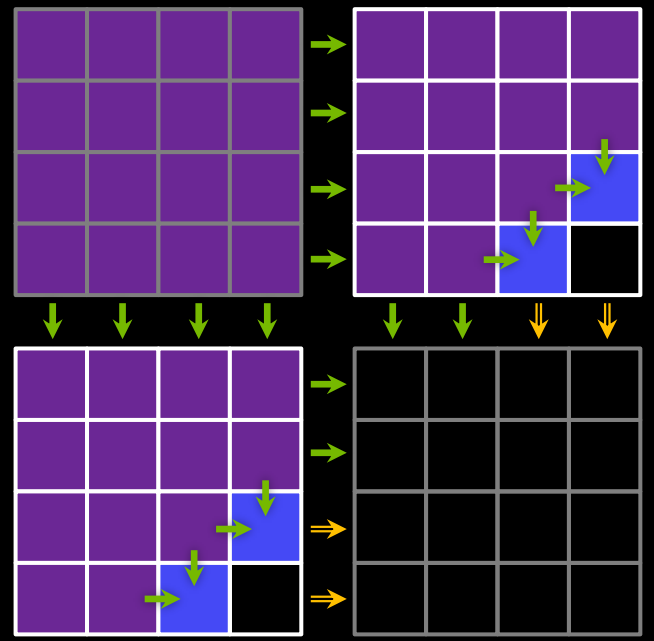
PARALLEL WAVEFRONTS

- When sweeping across the diagonal, available parallelism at each step follows a triangular pattern



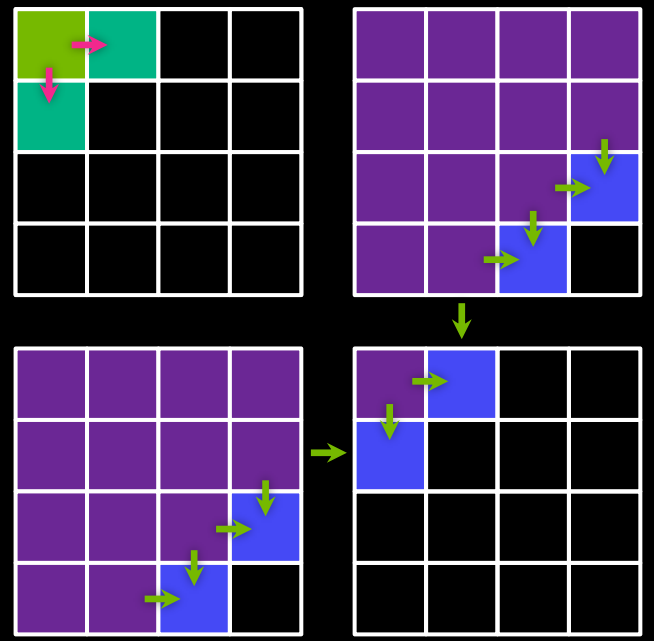
PARALLEL WAVEFRONTS

- Sweeping along the diagonal sequentially creates load imbalances (both within a node and across nodes)



PARALLEL WAVEFRONTS

- The standard solution to this is pipelining parallel wavefronts
  - Communicate each boundary value as soon as it's ready
  - Start on the next iteration's wavefront on the heels of the previous



SWEEP SCHEDULING IN UMT2013

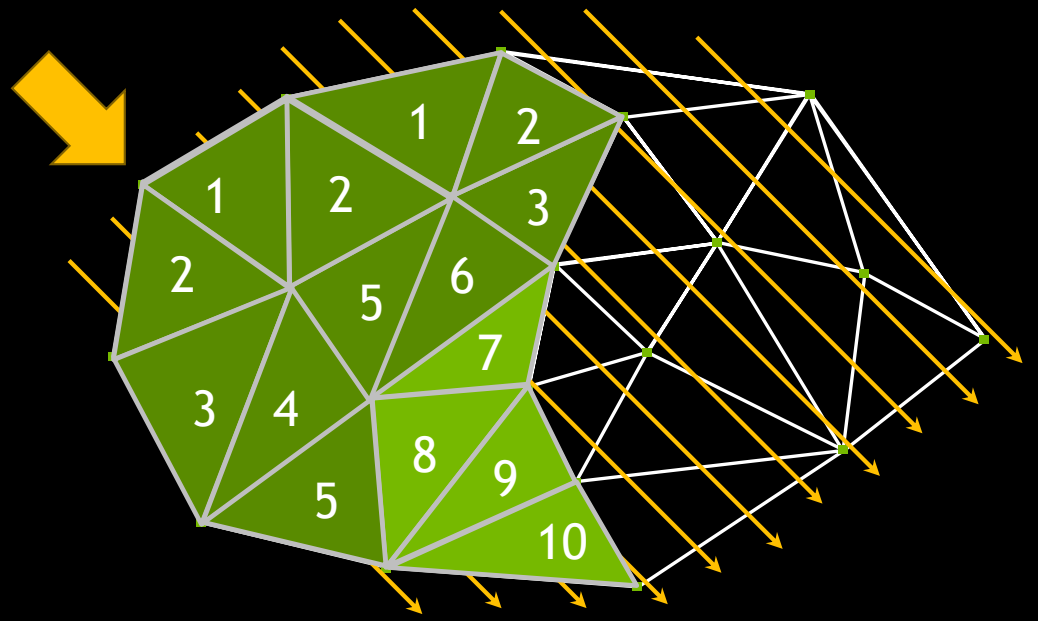
- Fortunately don't have to contend with the pipelining problem – UMT2013 already deals with this in another way
- We *do* have variable parallel widths – like sweeping diagonals in structured meshes, but worse due to the irregularity
- This is why UMT2013 makes no attempt to schedule sweeps for parallelism

SWEEP SCHEDULING IN UMT2013

- Still there *is* some degree of parallelism of zones within a given step of a single sweep
- Need a way to detect and extract that parallelism

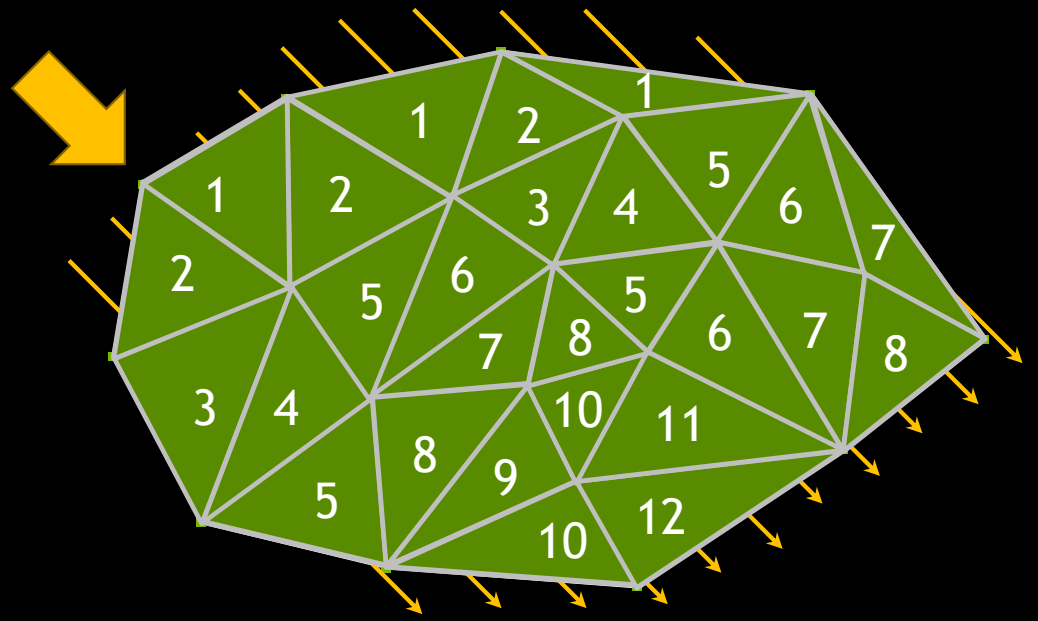
SWEEP SCHEDULING FOR PARALLELISM

- After scheduling the sweep, iterate over the schedule on the CPU and calculate the first step in which a zone can appear such that its dependencies will have been resolved



SWEEP SCHEDULING FOR PARALLELISM

- Use this value as a “batch number” and sort the sweep schedule using the batch number as the keys
- Resulting parallelism is irregular but still useful

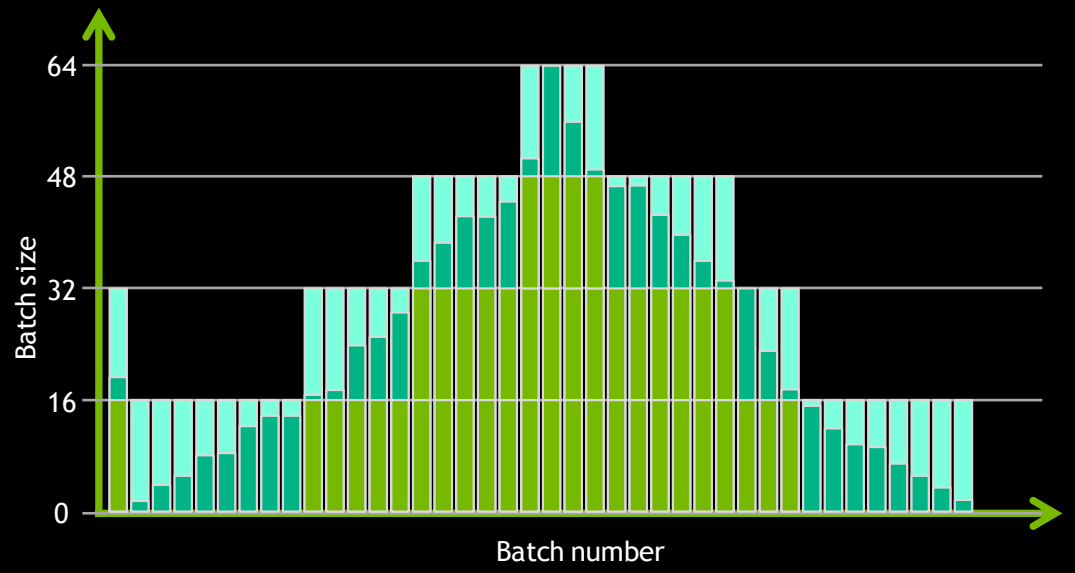


SWEEP SCHEDULING FOR PARALLELISM

- Batch size follows a (loosely) triangular trend similar to what was seen in structured mesh sweeps, though less predictable
- When zones/rank is ~2800, number of batches is ~100 and max batch size is ~64

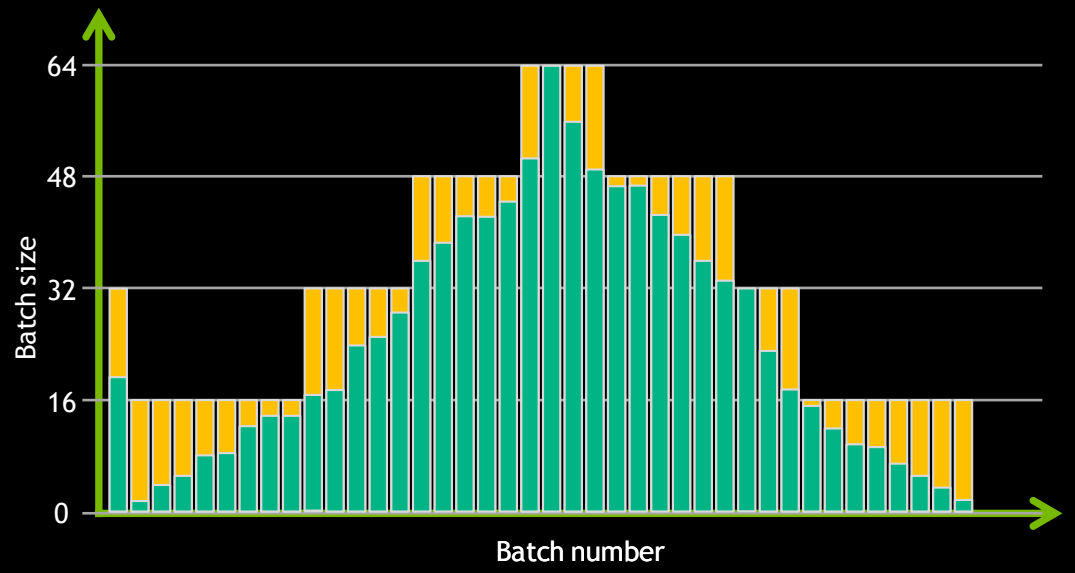
REGULARIZING UNSTRUCTURED MESH SWEEPS IN UMT2013

- To regularize this, pick a parallel width of, say, 16
  - Process sets of 16 zones from a single batch concurrently
  - Loop until all zones in batch are done
  - Synchronize threads, since some will have had one extra zone (loop iteration)
  - When whole batch is done, move on to next batch



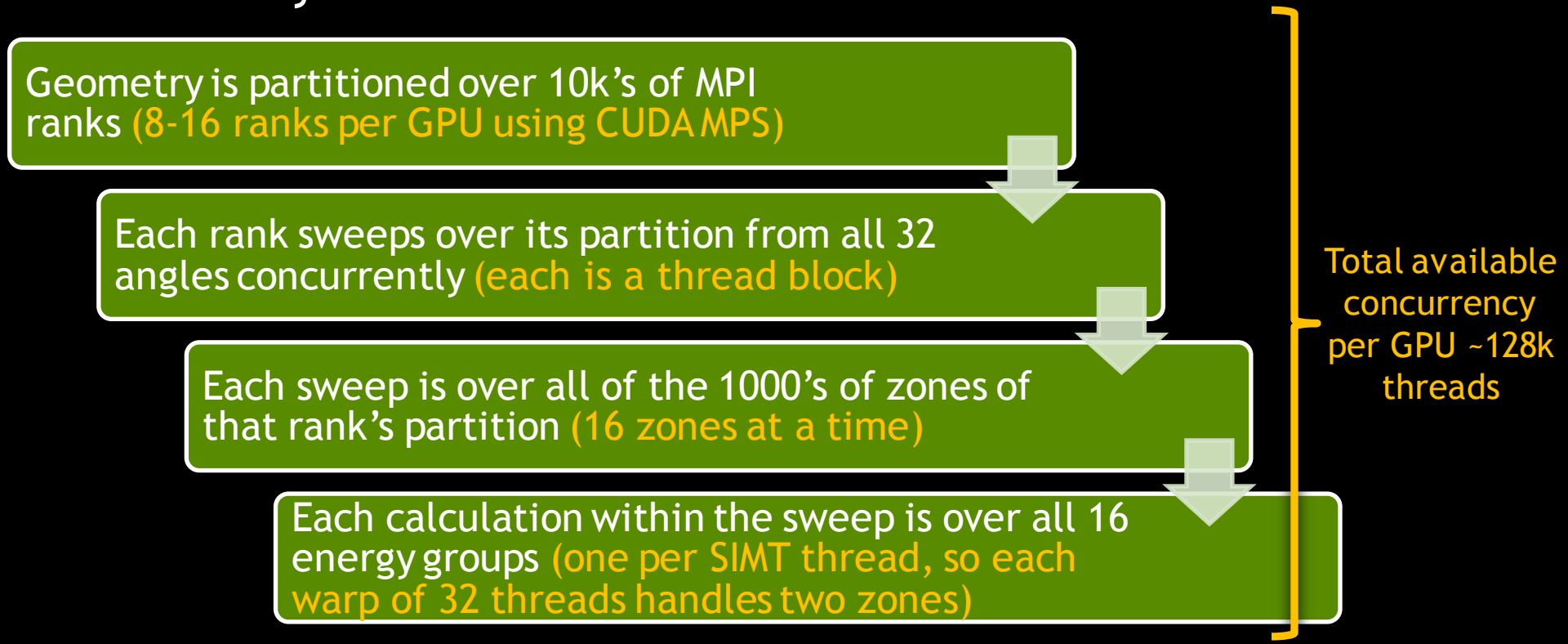
REGULARIZING UNSTRUCTURED MESH SWEEPS IN UMT2013

- Some processors idle some of the time; about 25% overall
- Possibilities for future improvement here
  - Different batch size
  - Work stealing



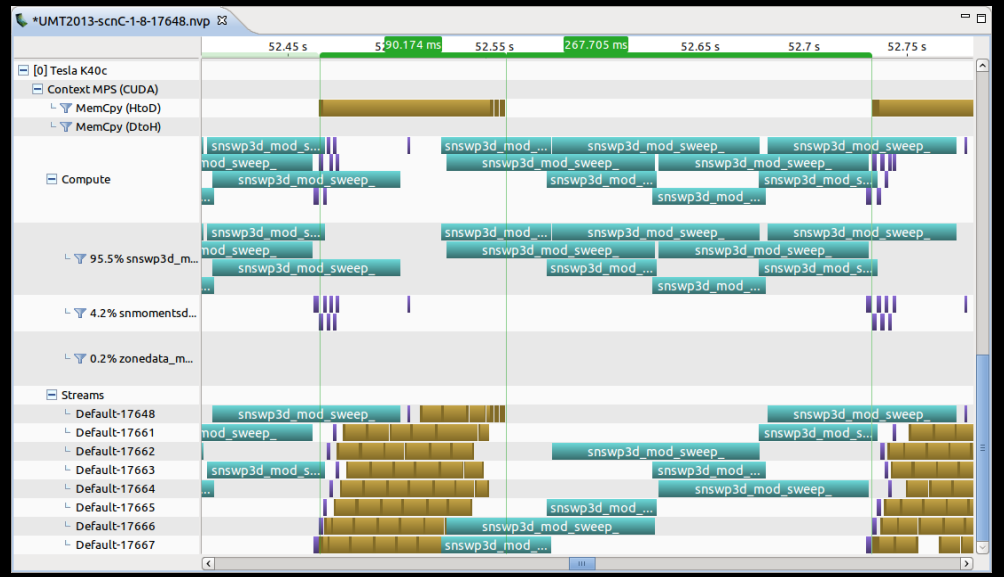
PARALLELISM IN UMT2013

- After exposing more parallelism, UMT2013 now has ~16x the concurrency:



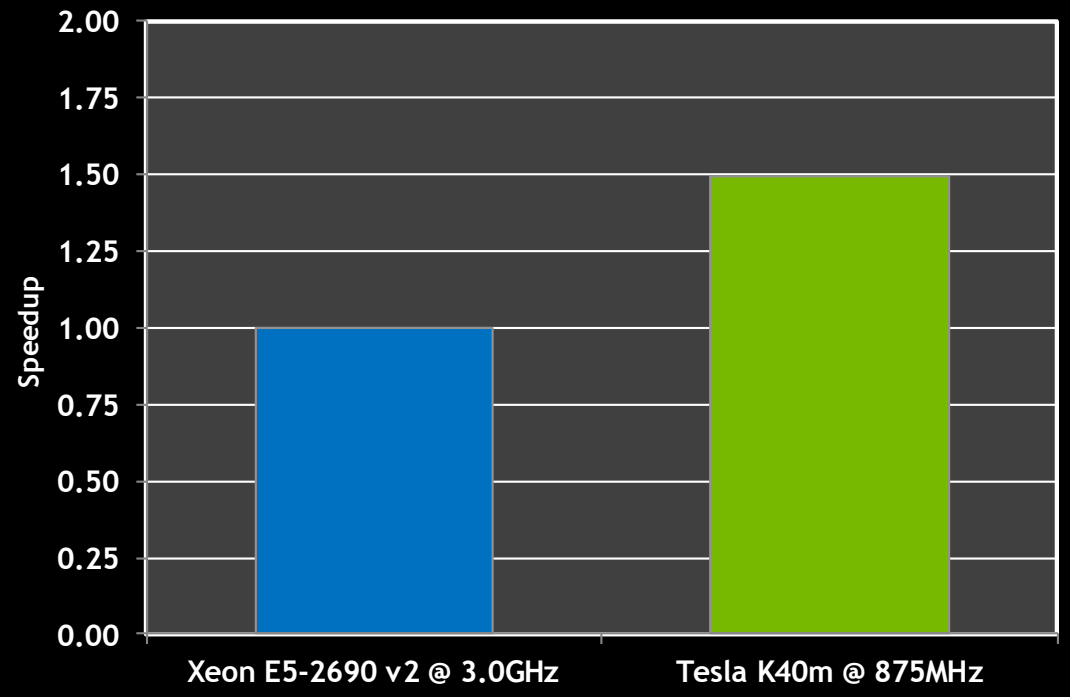
VISUAL PROFILER TIMELINE

- K40 can fit 60 of UMT's 16x16-thread blocks concurrently
- 32 blocks per rank, so ~2 ranks to fill the GPU once over
  - Note though that we prefer to fill the GPU many times over
- Memory transfers and computation overlapped automatically



RESULTS

- GPUs *can* do well on traversal of unstructured meshes!
- With this initial work, K40 already 1.5x faster than Ivy Bridge for whole-app performance



CUDA 6.0 RC, driver 331.44
PGI 13.9 compiler
10 ranks, 12^3 zones/rank

FUTURE WORK

- We’re not done yet!
- While the CPU run pegs all cores at 100% utilization throughout, the CUDA version has lots of headroom for further optimization, e.g.:
  - There’s still about 20% of total time outside the sweeps that is currently CPU-only; could GPU accelerate more of that
  - Find ways to reduce register pressure and improve occupancy
- Exposing sufficient parallelism was just the first step