Parallel SAH k-D Tree Construction

Byn Choi, Rakesh Komuravelli, Victor Lu,
Hyojin Sung, Robert L. Bocchino,
Sarita V. Adve, John C. Hart

UPCRC Illinois
Universal Parallel Computing Research Center

Motivation

- Real-time Dynamic Ray Tracing
  - Efficient rendering with the right spatial data structure
- SAH-based k-D tree proven very effective
  - Dynamic content requires rebuilding tree every frame

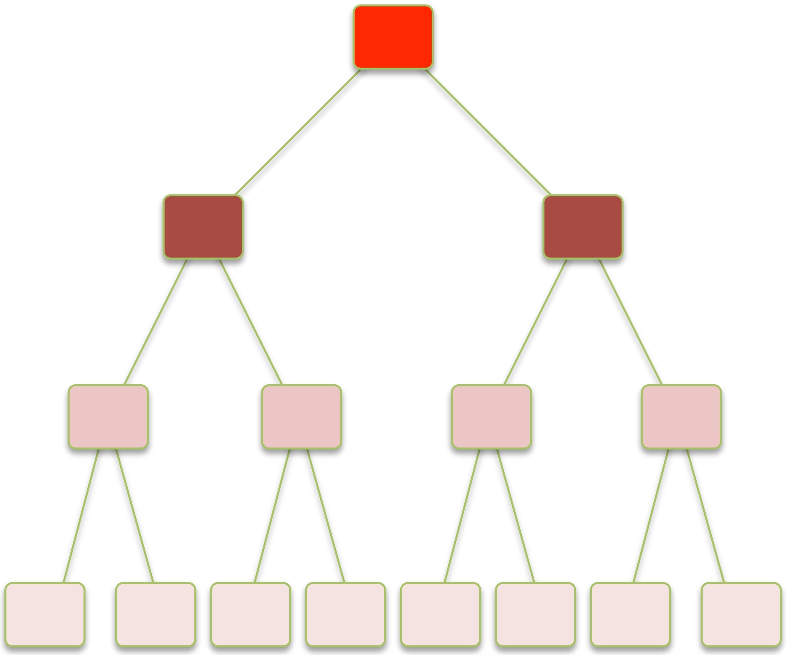
  $\Rightarrow$ Tree build becomes bottleneck in rendering pipeline
- Prior parallelization efforts abandon SAH
  - Sacrifice tree quality, increase rendering time

Contributions

- First to parallelize k-D tree construction with precise SAH
  - High quality AND high performance
- Two parallel algorithms: **Nested** and **In-place**
  - Different performance/scalability characteristics
- Up to 8x speedup on 32 cores

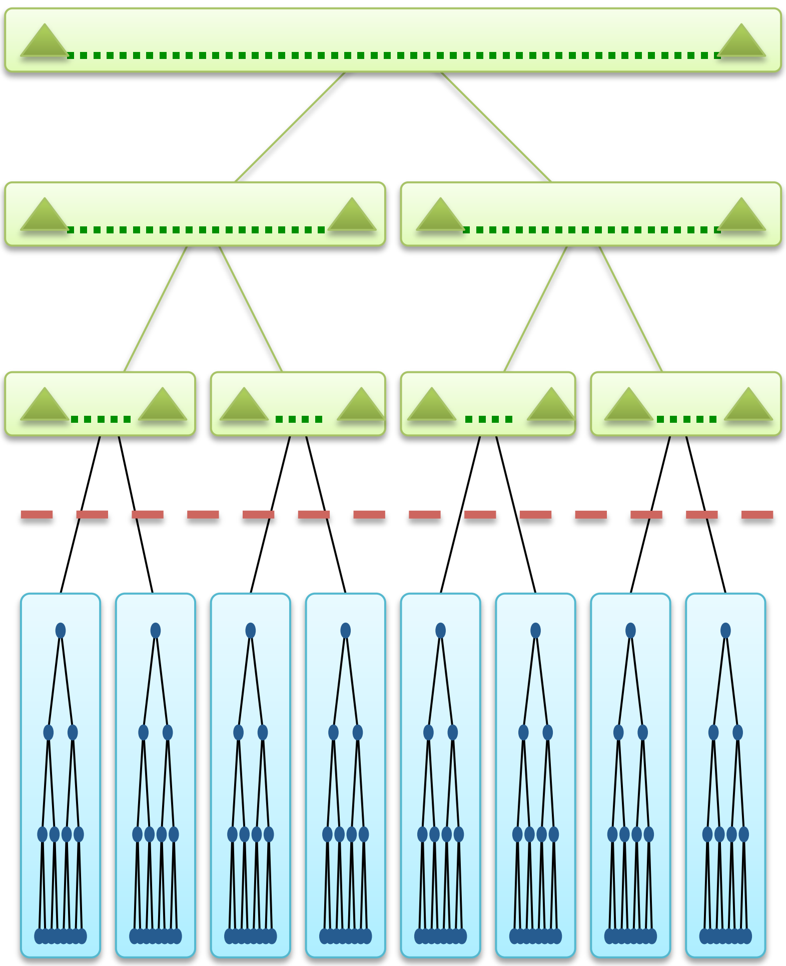
Straightforward || Construction

- Top-down – Recursive subdivision
- Divide-n-Conquer style
  - Recursive parallelism
  - Each node == a task
- Problem
  - Not enough parallelism at top
  - More work per node at top
  - Serial for top nodes:
    - Benthin PhD, 2006
    - Popov et al. IRT 2006
    - Hunt et al. IRT 2006



UPCRC Illinois
Universal Parallel Computing Research Center

k-D Tree Parallel Pattern



Geometry Parallel

Process triangles in parallel

Challenge: Compute precise SAH here

Moore's Law

Every 18 months the transition line descends a level

Node Parallel

Subtrees built in parallel

UPCRC Illinois
Universal Parallel Computing Research Center

Previous Approaches

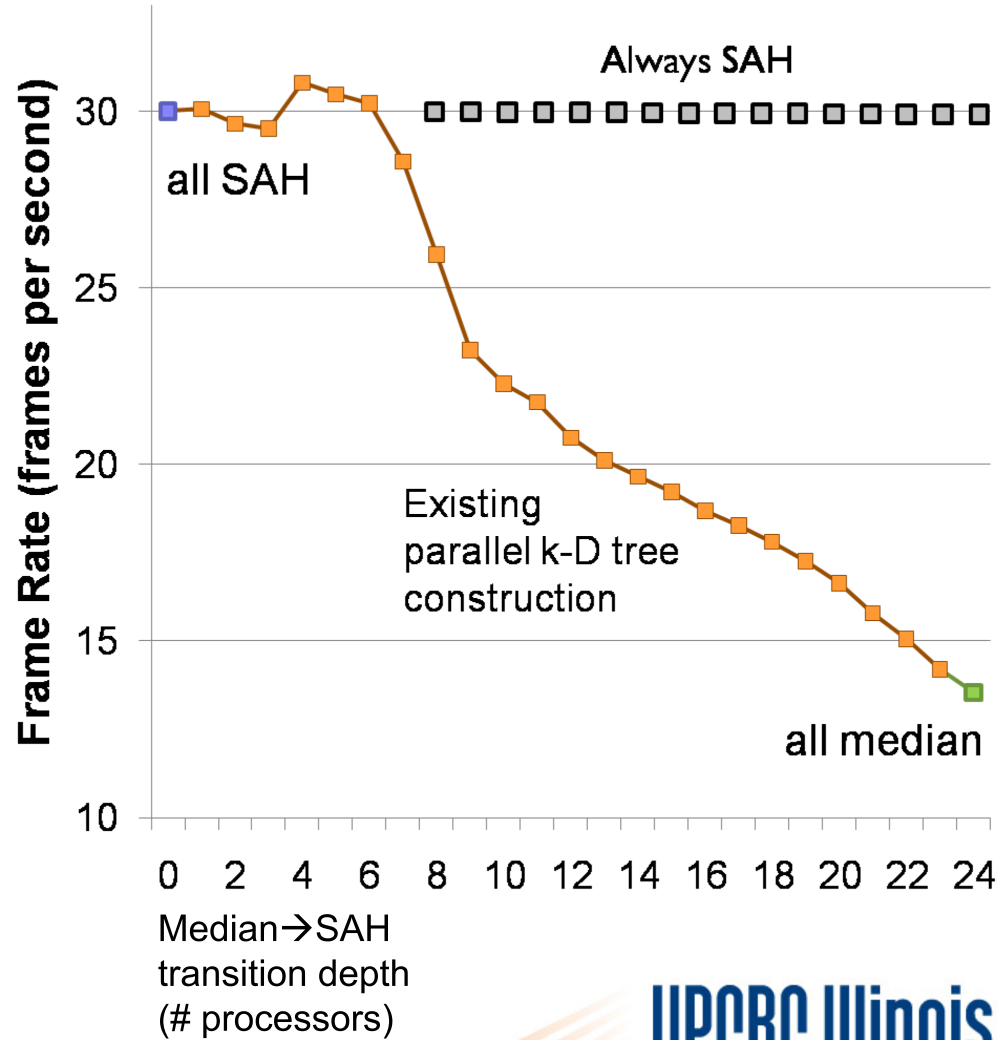
Shevtsov et al. CGF 07

- 4-core CPU, LRB
- Δ count median for upper-tree nodes

Zhou et al. SA 08

- Streaming GPU
- Spatial median for upper-tree nodes

As core counts increase, median (non-SAH) constructions degrade rendering performance

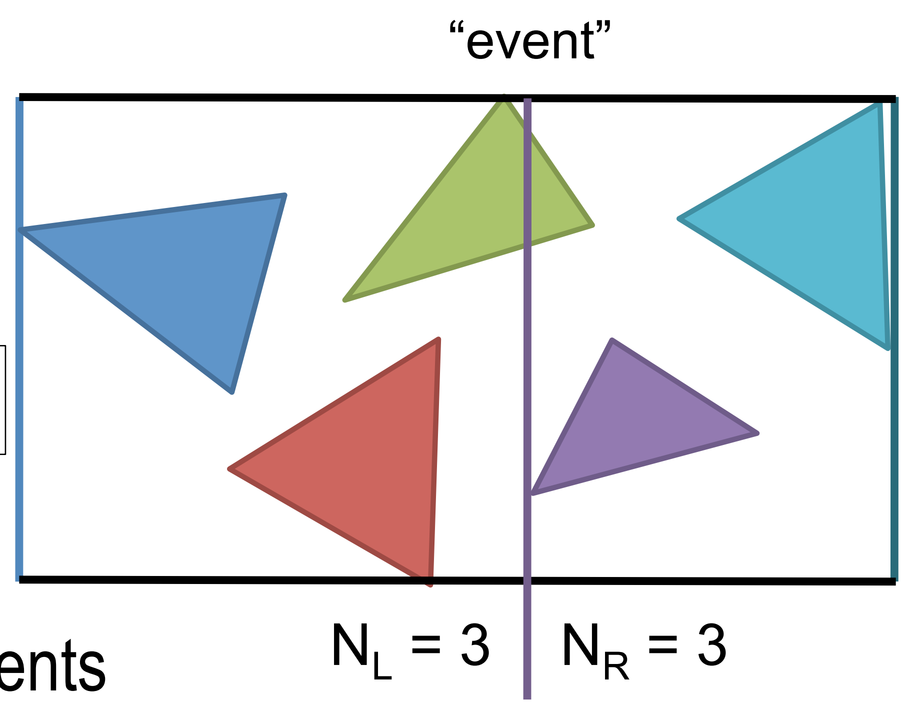


UPCRC Illinois
Universal Parallel Computing Research Center

Calculating SAH

- Prob. of hitting triangles ∝ surface area of bounding box
- Largest number of triangles in least surface area
- Need to find out…
  - $Area_L$, $Area_R$ – Surface area
  - $\#_L$, $\#_R$ – # of triangles

$$SAH \propto \#_L * Area_L + \#_R * Area_R$$

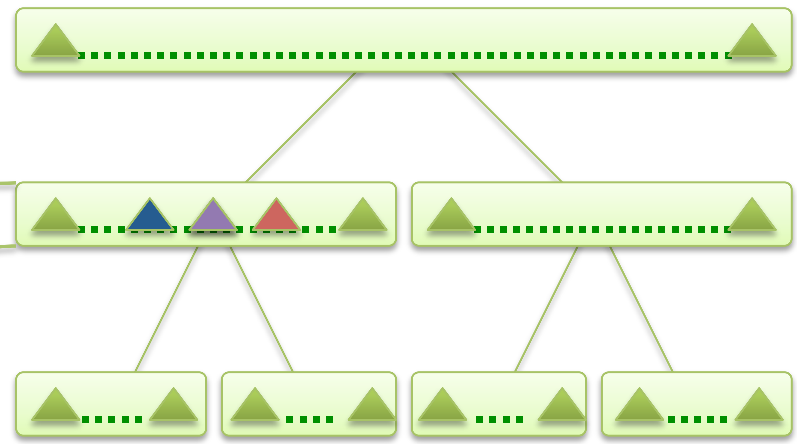


- Linear dependency between events
- Events need to be sorted!

Seq. k-D Tree Construction & Nested

(Wald and Havran, 2006)

- Recursive tree-building algorithm O(n log n)
- Sorted list of events as input
- 3 Major phases within a node
  - FindBestPlane (41%)
  - ClassifyTriangles (4%)
  - FilterGeom (55%)



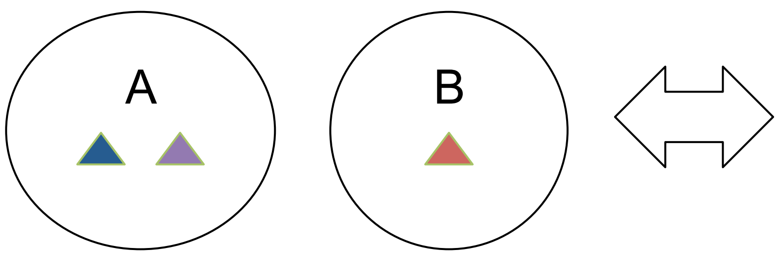
- Parallelization
  - FindBestPlane
    Linear dependence → Parallel Prefix
  - FilterGeom
    Sorted output → Parallel Prefix

Issues

- Two extra full-scans introduced (Parallel Prefix)
  - FindBestPlane
  - FilterGeom
- Data movement
  - Events moved from one container to another
- ClassifyTriangles hard to parallelize
  - Arbitrary bit writes by multiple threads into a shared bit vector
    - Synchronization overhead
    - False-sharing
  - 4% execution time == 25x maximum theoretical speedup

In-place Algorithm

- Events are kept “in-place” – no need to preserve ordering
  - Eliminates FilterGeom phase
  - Does less work
- Events responsible for tracking own membership(s)

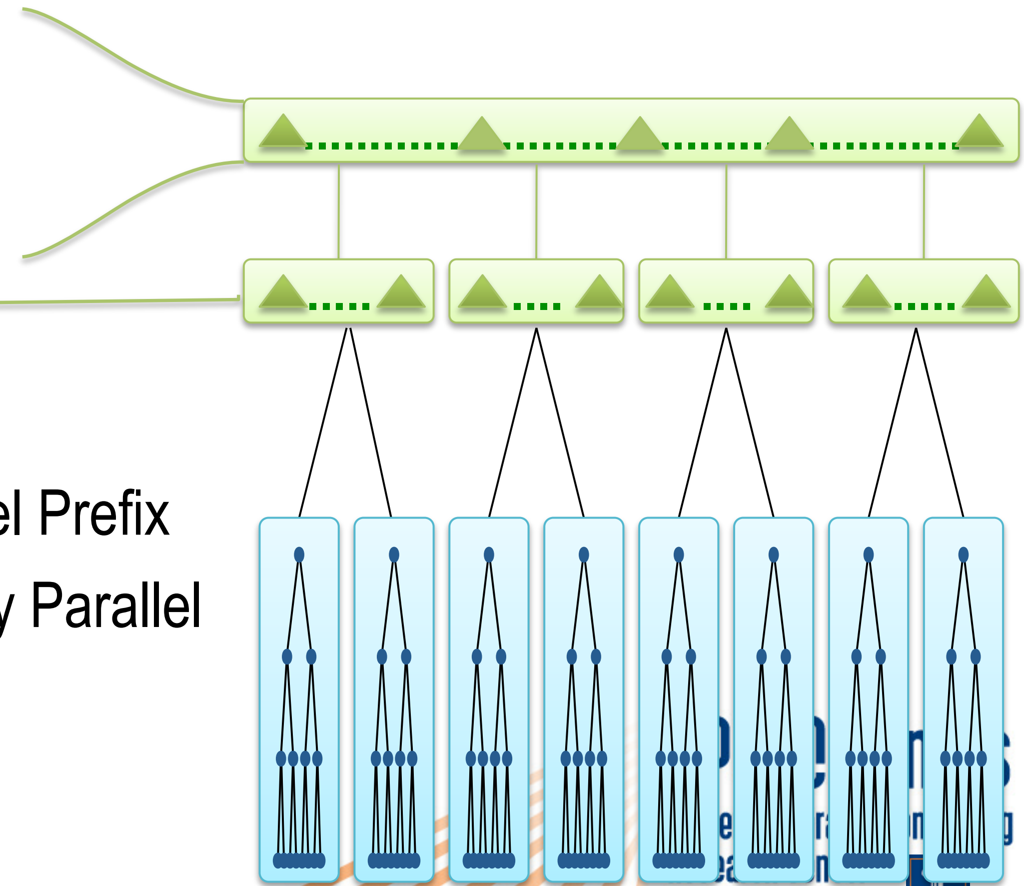


	Node
	A
	A
	B

- Change of membership is an update, not a move/copy

In-place Algorithm

- Iterative tree-building algorithm
- 3 Major phases within an iteration
  - FindBestPlane (85%)
  - Newgen (0.04%)
  - ClassifyTriangle (14%)
- Fill phase (0.52%)
- Parallelization
  - FindBestPlane → Parallel Prefix
  - ClassifyTriangles → Fully Parallel

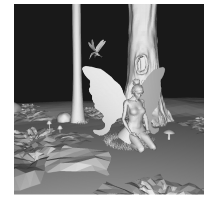


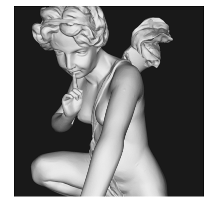
Methodology

- Both algorithms implemented using Intel TBB
- Five 3D models from
  - Stanford 3D Scanning Repository
  - Georgia Tech's Large Geometric Models Archive
  - The Utah 3D Animation Repository
- Machine configurations

Processor	Xeon E7450 ("Dunnington")	Xeon X7550 ("Beckton")
µarch	Core	Nehalem
Core Count	24	32
Cache	12 MB (L2)	18 MB (L3)
Memory b/w	1x	9x
Memory	48 GB	64 GB


bunny (69K)

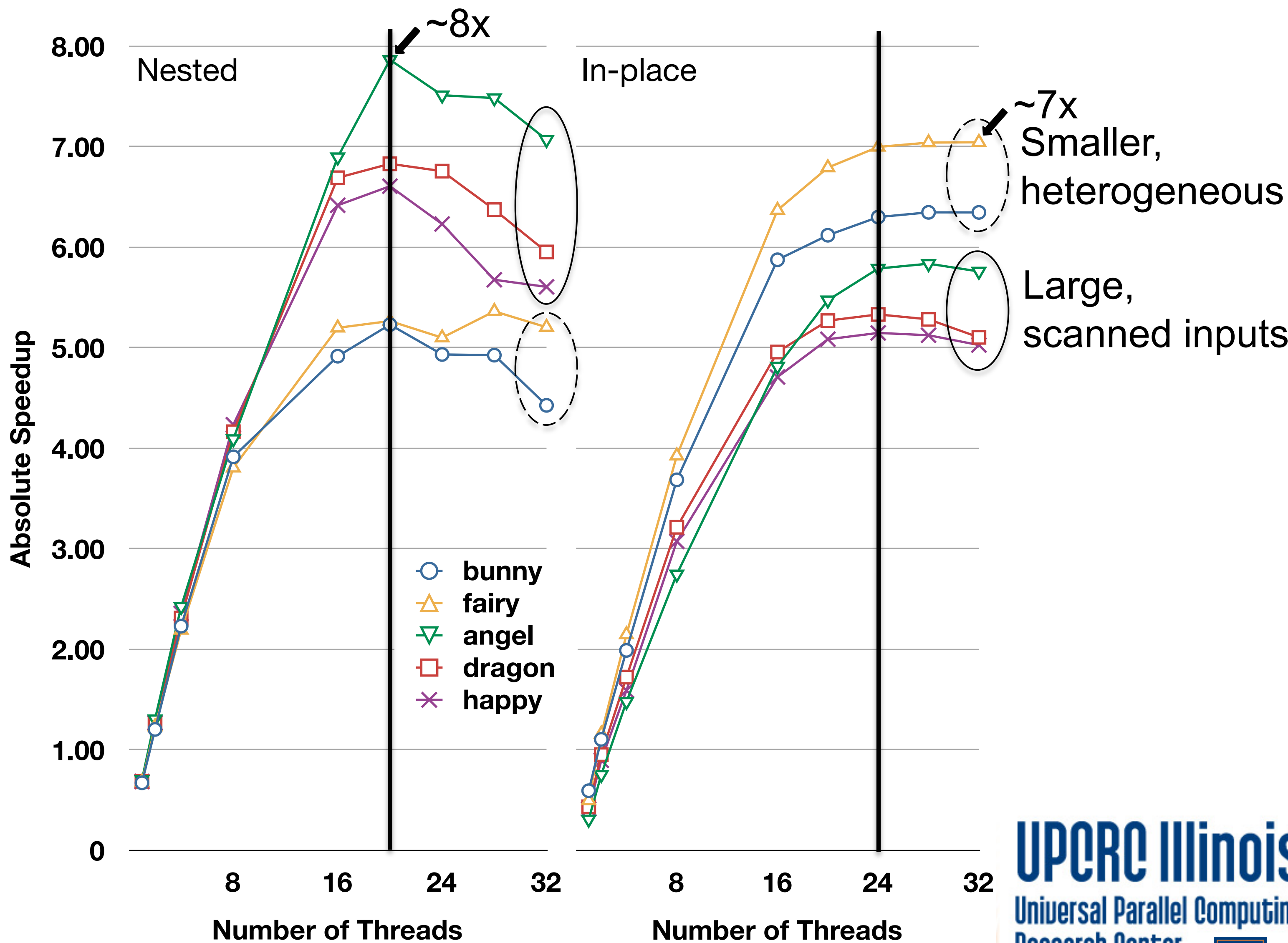

fairy (173K)


angel (474K)


dragon (871K)


happy (1M)

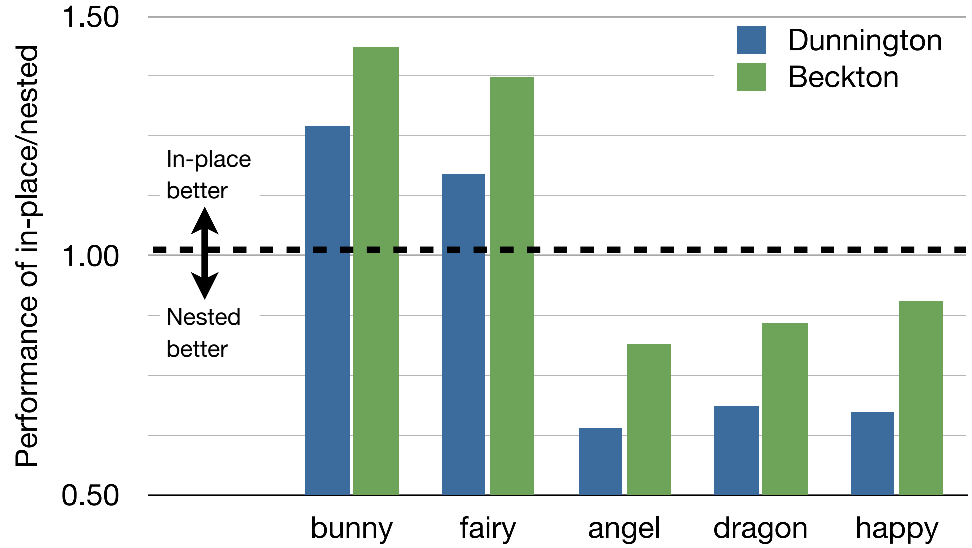
Results (Beckton)



UPCRC Illinois
Universal Parallel Computing Research Center

Scalability Analysis

- Future hardware
  - More cores
  - Larger caches
  - More memory b/w



- Nested has sequential portion => Amdahl's Law

Input	32 threads	∞ threads
bunny	12.9	21.0
fairy	13.1	21.6
angel	11.2	16.7
dragon	10.3	14.7
happy	10.3	14.8

Conclusion / Future Work

- Parallel build of high-quality k-D tree critical for ray tracing
  – Prior work trades quality for performance
- We show parallel build with high quality AND performance
  – Two algorithms with up to 8x speedup
  – Different performance/scalability characteristics
- Future work
  – GPU implementation of in-place
    Streaming nature more amenable to SIMD-fication

Thank You!

- Questions?