Scalable architecture model for real time Ray Tracing.

Asim Kadav  (kadav@cs.wisc.edu)

CS752
Summary

• What?
  – To device an architecture for ray tracing.
  – Processor organization, caches, number and type of cores.
• Why?
  – Increasingly important application for rendering.
• Application Characterization
  – Application cache sensitivity
  – Cache sensitivity on a per data structure basis
• Done using simulating address trace using sim-cache.
• Key Findings/Results
  – Cache sizes
  – Cache locality and type of caches
Outline

• Background
  – Ray tracing application
  – Software and hardware requirements
• Architectural Implications
• Recent Work
• Experiments
• Key Findings/Results
  – Cache Size
  – Number of threads/cores, number of cores.
• Conclusion
• Future Work
Background

• Classic rendering: Z buffering algorithm
  – Limited details
  – Local illumination model

• Ray Tracing
  – More realistic.
  – Real time and dynamic scenes.
  – But we are still struggling with hardware specifications and efficient software implementations.
Background

Local Illumination model

Global Illumination model

Image: intel corporation
Background

• Software
  – We need to build efficient algorithms for each of the stages of ray tracing.
  – Use acceleration structures for scene representation to help rendering. (like BSP trees)
  – Render components of a scene in parallel.
Architectural Implications

• Hardware
  – Build efficient multithreaded cores to support parallel rendering.
  – Decide the number of cores, type of cores and cache configuration.
Recent Efforts

• Existing GPUs
  – GPU performance model has limitations on general performance computations.

• Nvidia 8800 (G80 series)
  – Still optimized for Z buffer techniques.

• Ray Tracing on Cell
  – Cache accesses is a bottleneck.
  – Small local store of each SPEs is a bottleneck. (bring all scene data in memory)

• Many other studies on core requirements
  – Do they have parallelizable application.
Goals

• Determine cache behavior on the application wide basis.
• Determine cache behavior on data structure specific basis.
• Draw inferences on cache architecture and cache sizes.
Experimental Setup

• Using ray tracing software “Razor” that
  – Builds acceleration structure lazily.
  – Is multi threaded.
  – Excellent basis for architecture analysis.
• Instrument the Razor code with pin tool and generated address traces.
• Modified sim-cache to run these address traces under different cache configurations.
• Generated the top loads and stores and annotated with appropriate data structures.
Cache Miss rates

L1/L2 Cache miss rates versus L2 cache sizes (L1 cache size = 8K)
Cache Miss rates

L1/L2 Cache miss rates versus L1 cache sizes (L2 cache size = 2M)
L1/L2 Cache miss rates versus L2 cache sizes (L1 cache size = 8K)
Cache Miss rates

L1 Cache miss rate versus L1 cache size (L2 cache size = 4M)
Data Structure Cache Analysis

![Bar Chart]

- **Kd tree**
  - Stack: 2200
  - Non-Stack: 1500
- **Kd tree Builds**
  - Stack: 100
  - Non-Stack: 100
- **Kd tree leaf**
  - Stack: 200
  - Non-Stack: 100
- **Isec/box/grid**
  - Stack: 1000
  - Non-Stack: 500

Legend:
- **Stack**
- **Non Stack**
Data Structure Analysis and Conclusions

• Most loads and stores are for:
  – Small low level data structures.
  – Leaf node of acceleration structures.
  – NOT large acceleration structures.

• Implication on cache design?
  – Prefetech buffers
  – Hardware cache of fine granularity possible
Conclusions

• Application Characteristics
  – Irregular data structures
  – Dynamic data structures
  – Data dependent control flow

• L1 Size of 8K and L2 size of 4M/8M.
  – Implication of multiple cores.

• A hardware cache/prefetch of fine granularity for low level structures and a general purpose software managed cache will be helpful.
Future Work

• More application characterization
  – Multiple threads
• Type of multi threaded architecture
• Type of Cores
  – All general purpose or hybrid.
• Control Flow
  – MIMD/SIMD
• Power requirements
Questions