Parallel Computation Methods in *Radiance*

Greg Ward
Anyhere Software

Development History

- *Radiance* development began around 1986
- Originally written in K&R standard C
  - Most code updated by now to ANSI-C
  - Follows Unix "toolbox" model, consisting of over 120 C programs
  - 36 utilities written with C-shell interpreter, 5 recent additions use Perl
- First parallel processing tool **rpiece** added in 1993
  - Tile-based network image renderer using NFS locking mechanism
- Eleven *Radiance* tools now offer parallel mode
  - Rendering tools: **rad**, **ranimate**, **ranimove**, **rholo**, **rpiece**, **rvu**
  - Calculation tools: **mkillum**, **pabopto2bsdf**, **rcontrib**, **rsensor**, **rtrace**
  - Linear speed-up realized for common tools and use cases
- Most tools rely on shared memory arch (MISD)
  - Exploits Unix `fork()` call and copy-on-write memory sharing
  - Most communication depends on Unix `pipe()` call

Relevant *Radiance* Tools

mkillum	compute illum sources for a RADIANCE scene
pabopto2bsdf	compute BSDF interpolant from PAB-Opto measurements
rad	render a RADIANCE scene
ranimate	compute a RADIANCE animation
ranimove	render a RADIANCE animation with motion
rcontrib	compute contribution coefficients in a RADIANCE scene
rholo	generate/view a RADIANCE holodeck
rpiece	render pieces of a RADIANCE picture
rsensor	compute sensor signal from a RADIANCE scene
rtrace	trace rays in RADIANCE scene
rvu	generate RADIANCE images interactively

Radiance Parallelization

- Spawning multiple **rtrace** or **rpict** processes
  - Original method, can be made to work on Windows with NFS installed
  - Used by **rad**, **ranimate***, **rholo**, **rpiece***
- Library module `raypcalls.c`
  - Modern method, may be possible under Windows (currently calls stubs)
  - Used by **mkillum**, **ranimove**, **rsensor**, **rvu**
- Library module `rayfifo.c`
  - Layer on `raypcalls.c` that manages first-in/first-out queue
  - Used by **rtrace**
- Pure Unix (no abstraction layer)
  - Calls `fork()` and `pipe()` or `mmap()`
  - Used by **rcontrib**, **pabopto2bsdf**

*supports network rendering

Problematic Aspects

- Windows largely unsupported by parallel extensions
  - Georg Mischler ported original parallelization method (working?)
  - 7 out of 11 programs use new interface or call `fork()` directly
  - No replacement for `fork()` under native Windows
- NFS lock manager not historically reliable
  - Some Linux implementations are badly broken (newer ones seem OK)
  - Causes synchronization issues and corruption of ambient cache
  - Broken locks mess up **rpiece** even without shared ambient
- No leveraging of GPU or vector processing
  - Most vectors have length 3, so set-up time kills any SIMD gains
- Missing –n option to **rpict**
  - **rpiece** is difficult to use, but reasonably well-supported by **rad**
  - Setting up network rendering of single image a manual process

Parallelization Challenges

Property of *Radiance*	Implication	Remedy
Use of global and static variables	No multi-threading	~4 man-months, not including actual threads
Shared ambient cache	Need interprocess communication	Pure Monte Carlo may be faster in some cases
Large scene data	Shared memory	Limit scene size?
Few SIMD segments	Poor fit for GPU	Nothing feasible

Untried Acceleration Tech

- Optimizing compiler
  - Should be easy enough to compare a few optimizers / options
  - Need validation / test suite to confirm proper operation
- OpenCL
  - Most effective for SIMD problems, of which we have few
- Code tuning / assembler
  - Usually processor-specific, requiring maintenance year to year
- Cloud computing
  - Requires subscription service or DOE generosity

Low-hanging Fruit

- Add Hessian matrix to irradiance cache
  - Schwarzhaupt, Jensen & Jarosz paper from SIGGRAPH Asia 2012
  - Should improve calculation efficiency by a factor of 2-3
- Update Georg Mischler's port of original method
  - Would enable **rad**, **ranimate**, **rholo**, and **rpiece** under Windows
- Implement `raypwin.c` for Windows
  - Would enable **mkillum**, **ranimove**, **rsensor** and **rvu**
- Make full use of 3- and 5-phase annual simulation
  - Tools enhanced for time-step calculations over the whole year
  - Needs a good user interface to manage controls and complexity
- I need help on everything after the first item
  - Reflexively, I also need help on what I need help on

MIMD Implementation of **rcontrib**

- Critical tool for annual simulations and complex fenestration system analysis
  - Currently calls `fork()` and `pipe()` directly, thus broken under Windows
- Uses pure Monte Carlo
  - No interprocess communication required, unlike most *Radiance* renderers
- Tends to be used with simple geometry
  - Memory sharing is probably optional
- Estimate a 6-8 man-month project with testing
  - This is not something I can do by myself
- Need careful analysis of long-term benefit