Multithreading in Visual Effects
Jeff Lait
Side Effects Software Inc.
Multithreading in Houdini

Definitions
Houdini
What are Nodes?
Mantra
VEX

• Our SIMD shading language

• Superficially similar to Renderman Shading Language or GLSL
  – But is not just used for shading!
  – Deformation, Simulation, Procedurals, etc..
Multithreading in Houdini

Distribution
Distribution

• Why thread when you can run multiple processes?
  – Usual excuse for GIL in Python

• We already all distribute: Renderfarms
Only One Mouse
Distribution

- "All programming is an exercise in caching."
  - Terje Mathisen
- Difficulty is sharing cached structures
- CPU architectures hardware-accelerate sharing
Distribution

- Distribution forces data locality
Multithreading in Houdini

Challenges
Venerable

• Houdini 1.0 released in 1996
  – Some code survives from original PRISMS
• Core duo released in 2007
Interrelated

• Every part talks to every other part
• This behavior is not the exception!
Dependencies are Computed
Multithreading in Houdini

Methodology
Incremental Changes

• Continuous integration
• Bend without breaking
Statics: Not just for physics!

• Grepping for statics
  – `nm libGEO.so | c++filt | grep -i '^\[0-9a-F]* [bcdgs]'`
  – Need additional filters for guards, vtables, etc.

• Intel Compiler warnings
  – `icc -ww1711 -c foo.cc`
  – Warns about writes to statically allocated data
  – Avoids false positives about `const`-like structures
  – `#define TLA_THREADSAFE_STATIC_WRITE(code) \`
    __pragma(warning(disable:1711));
    CODE;
    __pragma(warning(default:1711))`
Focusing on the Stupidly Parallel

• Make it Easy
  – To Write
  – To Convert
  – To Undo
Stupidly Parallel: VEX
Stupidly Parallel: Task vs Thread

• Task Based
  – `tbb::parallel_for`
  – Functor runs once per chunk
  – Better load balancing

• Thread Based
  – Thread-pools like `UT_ThatchedAlgorithm`
  – Functor runs once per thread
  – Less marshalling
Stupidly Parallel: Thread

• For existing code, parameter list is often already minimized
  – The state before the critical for loop is often not!
• Task based functors require marshalling parms and locals
• The inner loop is textually separated from the setup code

```c
void foo(parmlist)
{
    // Setup
    float localvars = . . . ;

    // execute
    for all elements()
    {
        Code;
    }
}
```
Stupidly Parallel: Task

- UTparallelFor
  - Mandatory grain size
  - Early exit to inline version
- UTparallelForLightItems
- UTparallelForHeavyItems
- UTserialFor
Multithreading in Houdini

Patterns
Reentrancy

- **Always be reentrant.**

```cpp
class foo
{
public:
    void interface1()
    {
        UT::Lock::Scope scope(ourlock);
        // Do computation.
    }
    void interface2()
    {
        UT::Lock::Scope scope(ourlock);
        // Reuse interface1
        interface1();
    }
};
```
Never Lock

• Everything you were taught about clever synchronization is irrelevant
• You do not own the CPU
• You do not even know if there is a CPU
Atomics are Slow

- Treat them as guaranteed uncached memory

```cpp
class FOO_Array
{
public:
    shared_ptr<float> getData() const
    { return myData; }
private:
    shared_ptr<float> myData;
};

float sum_array(const FOO_Array &foo)
{
    float total = 0;
    for (int i = 0; i < foo.length(); i++)
        total += foo.getData()[i];
}
Never Blindly Fork

• Always consider grain size
• Always consider single vs multithreaded
Determinism

• Produce same results regardless of lock resolution.
  – Without this, debugging is a nightmare!

• Produce same results regardless of machine architecture (# threads, etc)
  – Task based is useful
  – We don’t do very well at this
Command Line Control

• `-j ncpu`
• Allows multiple processes to share a machine
• Allows you to easily debug and perf test
Memory Constant in Cores

• How to multithread a random write pattern?
  – Lock writes
  – Find a pattern that is thread safe
  – Copy the destination per thread and merge after

• What if you have a four-socket, ten-core, machine with hyperthreading?
Memory Allocation

• tbb::scalable_malloc fragments
• jemalloc
Copy on Write

- Reader/Writer Locks
- Const Correctness
- Importance of Ownership
- Sole Ownership is a Writer Lock
- How this Fails
Copy on Write: Reader/Writer

- Natural needs for a piece of geometry
  - Many readers
  - Many writers
- Need to ensure readers are not made inconsistent by writes
- Reader/Writer Lock is a solution
  - But we do not want any locks!
Barring other synchronization primitives; this is also a possible execution order.
Shared Geometry Cache

This has the same effect but with overlapping computation.
Copy on Write: Const

• Const Correctness
  – Does not provide speed improvements
  – Does create a compiler-enforced contract on the code!

• Easy to find lazy programming
  – const_cast
  – mutable
  – Most importantly, easier to just not make things const!
Copy on Write: Ownership

• Lack of garbage collection is a feature
Copy on Write: Ownership

- **shared_ptr** should be minimized!

```cpp
class FOO_Array
{
public:
    float *getData() const
    { return myData.get(); } 

    shared_ptr<float> copyData() const
    { return myData; }

private:
    shared_ptr<float> myData;
};
```
Copy on Write: Sole Ownership

- **Two types of readers**
  - Readers from foreign threads/algorithms
  - Our own readers

- Preventing external readers lets you reason about your own readers

- Sole ownership is a Writer Lock
  - If no external algorithm has a copy of your data structure, you can reason about a lock-free approach
class POO_Array
{
public:
  const float *readData() const
  { return mData.get(); }

  float *writeData()
  { makeUnique(); return mData.get(); }

  shared_ptr<float> copyData() const
  { return mData; }

private:
  void makeUnique()
  {
    if (mData.unique()) return;

    shared_ptr<float> copy(new float*[size]);
    memcpy(copy.get(), mData.get(),
           sizeof(float)*size);
    mData = copy;
  }

  shared_ptr<float> mData;
};
Copy on Write: Failure

- Excessive sharing

```c
void applyPartial(FOO_Array foo, RANGE partialrange)
{
    float *dst = foo.writeData();
    for (i in partialrange)
    {
        dst[i] *= 2;
    }
}

FOO_Array bar;

invoke_parallel(bar, applyPartial);
```
Copy on Write: Failure

- Pointer aliasing

```c
void apply(float *dst, const float *src)
{
    for (i = 0; i < size; i++)
    {
        dst[i] = src[i/2];
    }
}

void process(FOO_Array &foo)
{
    const float *src = foo.readData();
    float *dst = foo.writeData();
    apply(dst, src);
}
```
Multithreading in Houdini

Task Locks
Task Locks

- Not all threads are equal
- A cook lock block Python from reading intermediate state
- VEX tasks spawn to many threads
- VEX can then trigger cooks
- Which needs the cook lock!
- Task locks allow reentrancy to child tasks, even if they are currently on a different thread
Global Cook Lock

Main
TBB 1
TBB 2
TBB 3
Python
Cook
Cook
Cook
Global Cook Lock

Main

Cook

TBB 1

TBB 2

TBB 3

Posted Tasks
Vex1, Vex2, Vex3, Vex4
Global Cook Lock

Main

Cook

Vex1

TBB 1

Vex2

TBB 2

Vex3

TBB 3

Vex4
Global Cook Lock

- Main
- TBB 1
- TBB 2
- TBB 3
- Cook
- Vex1
- Vex2
- Vex3
- Vex4

The diagram shows the various levels and components involved in the Global Cook Lock mechanism.
Task Lock: Details

• Task Owner defines a tree of child tasks
• Thread Local stores current thread’s Task Owner
• Tasks update their thread’s owner on start
• Lock is a condition + mutex which allows a stack of locks provided owner is strictly a child
Task Cook Lock

Main

Cook

Vex1

Vex2

Cook

Vex2

Vex3

Cook

Vex4

TBB 1

TBB 2

TBB 3
Composability

Posted Tasks
Vex1, Vex2, Vex3, Vex4
Composability

Main

Cook

Vex1

Vex2

Vex3

Vex4

TBB 1

TBB 2

TBB 3
Composability

Main

Cook

Vex1

KD

TBB 1

Vex2

KD

TBB 2

Vex3

KD

TBB 3

Vex4

KD

Posted Tasks
KD1, KD2, KD3, KD4
Composability

Main

TBB 1

TBB 2

TBB 3

Cook

Vex1

KD

Vex2

KD

KD1

Vex3

KD

Vex4

KD

Posted Tasks
KD2, KD3, KD4
Composability

Main

Cook

Vex1

KD

Vex2

KD

KD1

KD2

KD3

KD4

TBB 1

Vex3

KD

TBB 2

Vex4

KD

TBB 3
How we hope it works…
Deadlock…

Posted Tasks
Vex1, Vex2, Vex3, Vex4
Deadlock…

- Main
- Cook
- Vex1
- Vex2
- Vex3
- TBB 1
- TBB 2
- TBB 3

Posted Tasks
Vex4
Deadlock…

Main
TBB 1
TBB 2
TBB 3

Cook Vex1 KD
Vex2 KD
Vex3 KD

Posted Tasks
Vex4, KD1, KD2, KD3, KD4
Deadlock...

Posted Tasks
Vex4, KD3, KD4
Deadlock...
Task Locks

• Task stealing leads to deadlocks
• The stack creates an implicit lock!
Multithreading in Houdini

Task Arenas
Task Arena

• Never call back to the TBB scheduler if a lock is held
• If you must, wrap a Task Arena
Task Arena

• Moved from Preview to mainline in TBB 4.3
• Documentation suggests for controlling thread pools
• Solves the locking problem
Task Arena

• Acquire your lock
• Build just-in-time a tbb::task_arena
• Execute code (task_arena::execute())
  – No external tasks will be assigned to this thread!
• Release your lock
Task Arena

Posted Tasks
Vex1, Vex2, Vex3, Vex4
Task Arena

Posted Tasks
Vex4, KD1, KD2, KD3, KD4
Task Arena

Posted Tasks
Vex4, KD3, KD4
Task Arena

- Posted Tasks
  - KD4

- Vex4 CANNOT be assigned to TBB1 until TBB1 leaves the arena!
Task Arena

Main

Cook  Vex1  KD

Vex2  KD  KD1  KD3  KD4

Vex3  KD

KD2  Vex4  KD

TBB 1

TBB 2

TBB 3
Task Arena

• Beware implicit locks from stack unwinding
• Do not re-use
• Catch exceptions
Multithreading in Houdini

Task Exclusive
Task Exclusive

• Many threads want the same cached item
• Task Lock results in single threaded computation of the item.
• Why can’t blocked threads contribute work?
Task Exclusive

- Lock Free Singleton
- Tasks compete to get right to compute resource
- Winning task computes as normal
- Losing tasks add a dependent task and return to scheduler
Task Exclusive

- Main
- Cook
- TBB 1
- TBB 2
- TBB 3

Posted Tasks
Vex1, Vex2, Vex3, Vex4
Task Exclusive

Main

Cook

Vex1

TBB 1

Vex2

TBB 2

Vex3

TBB 3

Vex4
Task Exclusive

Main

TBB 1

TBB 2

TBB 3

Cook  Vex1  KD  rKD1

Vex2  KD  rKD2

Vex3  KD  rKD3

Vex4  KD  rKD4

Posted Tasks

rKD3 wins race
Task Exclusive

- Main
- TBB 1
- TBB 2
- TBB 3
- Cook
  - Vex1
  - KD
  - rKD1
- Vex2
  - KD
  - rKD2
- Vex3
  - KD
  - rKD3
- Vex4
  - KD
  - rKD4

Posted Tasks: KD1, KD2, KD3, KD4
Waiting on rKD3: rKD1, rKD2, rKD4
Task Exclusive

Posted Tasks
Waiting on rKD3: rKD1, rKD2, rKD4
Task Exclusive

Main

Cook Vex1 KD rKD1 KD1 rKD1

TBB 1

Vex2 KD rKD2 KD2 rKD2

TBB 2

Vex3 KD rKD3 KD3

TBB 3

Vex4 KD rKD4 KD4 rKD4
Task Exclusive

• Lock Free Singleton
  – But the stack creates an implicit lock!
  – Need a Task Arena to ensure stack can empty back to the recycled task.
Stack Locks

(gdb) info threads
Id Target Id Frame
17 Thread 0x7f9c2c603760 (LWP 22414) "houdini" 0x00007f9c598a3de3 in select () at ../sysdeps/unix/syscall-template.S:81
16 Thread 0x7f9c211f8760 (LWP 22452) "houdini" pthread_cond_wait@GLIBC_2.3.2 () at ../sysdeps/unix/syscall-template.S:81
15 Thread 0x7f9c1a833760 (LWP 22453) "houdini" syscall () at ../sysdeps/unix/syscall-template.S:81
14 Thread 0x7f9c1a833470 (LWP 22456) "houdini" syscall () at ../sysdeps/unix/syscall-template.S:81
13 Thread 0x7f9c17f33700 (LWP 22457) "houdini" 0x00007f9c5987ecf7 in sched_yield () at ../sysdeps/unix/syscall-template.S:81
12 Thread 0x7f9c17f33270 (LWP 22458) "houdini" syscall () at ../sysdeps/unix/syscall-template.S:81
11 Thread 0x7f9c17f33170 (LWP 22459) "houdini" syscall () at ../sysdeps/unix/syscall-template.S:81
10 Thread 0x7f9c15f3d700 (LWP 22490) "houdini" syscall () at ../sysdeps/unix/syscall-template.S:81
9 Thread 0x7f9c15f38700 (LWP 22491) "houdini" syscall () at ../sysdeps/unix/syscall-template.S:81
8 Thread 0x7f9c15f39700 (LWP 22492) "houdini" syscall () at ../sysdeps/unix/syscall-template.S:81
7 Thread 0x7f9c14f3d700 (LWP 22493) "houdini" syscall () at ../sysdeps/unix/syscall-template.S:81
6 Thread 0x7f9c14f3d6700 (LWP 22494) "houdini" syscall () at ../sysdeps/unix/syscall-template.S:81
5 Thread 0x7f9c14f3a5700 (LWP 22495) "houdini" TBB_machine_pause (delay=80) at ../include/tbb/machine.cc:32_common.h:55
4 Thread 0x7f9c13f77f00 (LWP 22496) "houdini" 0x00007f9c598a3de3 in select () at ../sysdeps/unix/syscall-template.S:81
3 Thread 0x7f9c13f77e700 (LWP 22500) "houdini" 0x00007f9c598a3de3 in select () at ../sysdeps/unix/syscall-template.S:81
2 Thread 0x7f9c13f75d2700 (LWP 22561) "houdini" 0x00007f9c598a3de3 in select () at ../sysdeps/unix/syscall-template.S:81
1 Thread 0x7f9c13f75b0b58d0 (LWP 22511) "houdini" 0x00007f9c5987ecf7 in sched_yield () at ../sysdeps/unix/syscall-template.S:81
Stack Locks
Multithreading in Houdini

Conclusion
Final Thoughts

• Make it easy to parallelize
• Make it easy to switch to serial
• Use jemalloc
• You do not need to rewrite

Thank you