

Making Civilization V Rock For Multi-Core

- Steve Smith
- Senior Graphics Software Engineer, Intel Corporation
- <http://software.intel.com/en-us/visual-computing>
- With Thanks: Dan Baker, Firaxis Games



Civilization IV CPU usage hurts game play

- Civilization is a CPU intensive game
- Performance issues increased as the game progressed
- Civilization Revolutions on console had similar issues
- Civilization IV architecture could not scale with the current and future hardware platforms.



VISUAL
ADRENALINE



What the Civilization V Engine can do

- Civilization needs unique, engine designed for massive throughput
- Typical Max Load for an engine (Fully animated, unique animated character)

FPS:	20 characters
MMORPG:	100 characters
RTS:	500 characters
Civ5:	10,000 characters

How did we get there?



Bismarck demands tasking, or else...



VISUAL
ADRENALINE

intel
Software



Civilization V Engine required a rethink

- Major redesign of all graphics game systems
 - Think about how the data is processed
- CPU scaling problems addressed:
 - Removed repetitive work, e.g. memory copies, useless virtual functions
 - Decoupled internal systems (AI, Physics, Rendering, etc)
 - Systems use Intel TBB to scale with tasks
 - Rendering system split up internally
 - Thread safety a property of decoupling, not enforced by locks
 - Data layout based on access pattern rather than logical objects



Message based systems

- Major systems communicate via message channels
 - Forces decoupling
 - Each system maintains any state that it requires, ignores messages doesn't care about
 - Deferring deletion/creation to specific points in code
 - Decentralizing state means easy to guarantee a tree of functions doesn't tunnel through the entire system
 - Much easier to write thread safe code - have had very few threading bugs in Civ5



Intel Graphics Performance Analyzer: See the whole system at once

- Platform View
 - Whole system view
 - CPU and GPU on the same scale
 - Use deep-zoom to find performance issues
 - Visualizes tasks running on the CPU
 - Virtual tasks allow grouping of tasks by system
- Task instrumentation is easy
 - TAL (Task Analyzer) instrumentation with 3 lines of code



www.intel.com/software/gpa

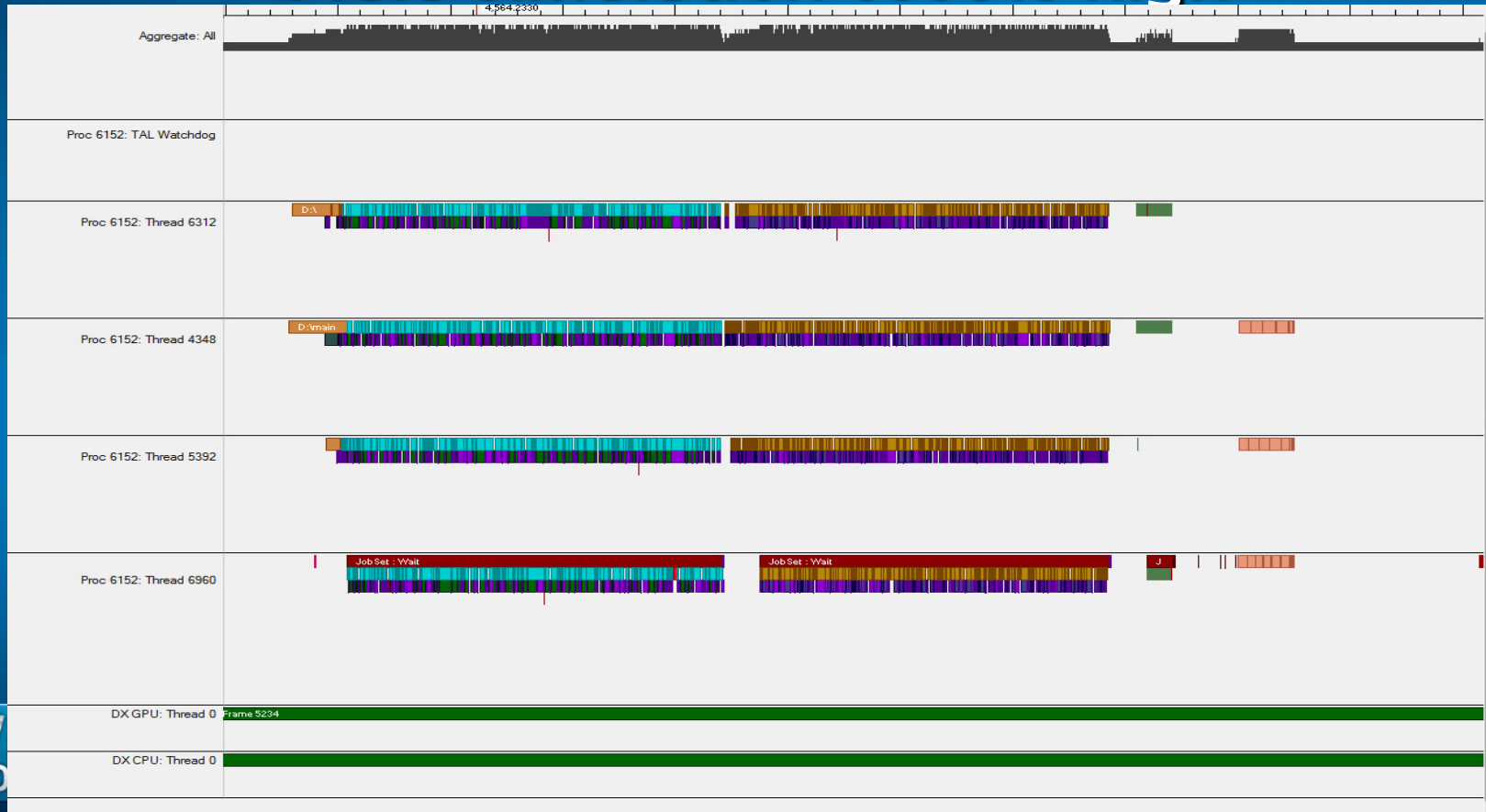


Fluid simulation cost is high

- Lagrangian simulation
- Must be implemented on CPU for non-DX11 hardware
- Several stages of parallel tasks
- Visualized what was happening on the CPU with Platform View



Fluid simulation cost is high



Fluid simulation cost is high

- First observation - lots of unaccounted for CPU time
 - Originally did not instrument allocators
 - Only 1 allocation per job, but that was enough...
 - Allocator was slow and contained a lock
 - Caused the Sim to run almost serially
- Instrumented Allocators showed problem with system
 - Re-organized code to remove dynamic allocations
 - Used placement-new instead

The Problem

```
for(CellIndex.j = PaddedMin.j; CellIndex.j < PaddedMax.j; CellIndex.j++)
{
    for(CellIndex.i = PaddedMin.i; CellIndex.i < PaddedMax.i; CellIndex.i++)
    {
        FluidPackPartInfoVector &Cell = m_ParticleGrid.pPackGrid[CellIndex.j*m_ParticleGrid.nBinWidth + CellIndex.i];
        FluidPackPartInfoVector::iterator end = Cell.end();
        for(FluidPackPartInfoVector::iterator iter = Cell.begin(); iter != end; iter++)
        {
            ParticleSystemDensityJob *pJob = new ParticleSystemDensityJob(this, iter);
            MyJobs.AddJob(pJob, "Fluid Density", &rtID);
        }
    }
}

class ParticleSystemDensityJob : public FJob
{
    ...
public:
    void Execute(uint uJob)
    {
        m_pParticleSystem->ComputeDensitySSE2(m_pPackedPart);
        delete this;
    }
};
```



Using placement New

```
ParticleSystemDensityJob *pJobs = (ParticleSystemDensityJob*)
    FNEW(char[sizeof(ParticleSystemDensityJob)*MaxJobs], c_eCiv5FOW, 0);
uint uCurrentJob = 0;

for(CellIndex.j = PaddedMin.j; CellIndex.j < PaddedMax.j; CellIndex.j++)
{
    for(CellIndex.i = PaddedMin.i; CellIndex.i < PaddedMax.i; CellIndex.i++)
    {
        FluidPackPartInfoVector &Cell = m_ParticleGrid.pPackGrid[CellIndex.j*m_ParticleGrid.nBinWidth + CellIndex.i];
        FluidPackPartInfoVector::iterator end = Cell.end();
        for(FluidPackPartInfoVector::iterator iter = Cell.begin(); iter != end; iter++)
        {
            ParticleSystemDensityJob *pJob = new (&pJobs[uCurrentJob]) ParticleSystemDensityJob(this, iter);
            MyJobs.AddJob(pJob, "Fluid Density", &rtID );
            uCurrentJob++;
        }
    }
}
```



Better, but still some problems

- Seeing a large amount of white space between jobs
- Means our jobs are too small for the scheduler
- Intel TBB has a `parallel_for` construct, similar to OpenMP construct
 - Can make a meta loop which spawns smaller loops
 - Scheduler can now aggregate many jobs together

Using parallel_for

```
for(CellIndex.j = PaddedMin.j; CellIndex.j < PaddedMax.j; CellIndex.j++)  
{  
  for(CellIndex.i = PaddedMin.i; CellIndex.i < PaddedMax.i; CellIndex.i++)  
  {  
    FluidPackPartInfoVector &Cell = m_ParticleGrid.pPackGrid[CellIndex.j*m_ParticleGrid.nBinWidth + CellIndex.i];  
    FluidPackPartInfoVector::iterator end = Cell.end();  
    for(FluidPackPartInfoVector::iterator iter = Cell.begin(); iter != end; iter++)  
    {  
      ParticleSystemDensityJob *pJob = new (&pJobs[uCurrentJob]) ParticleSystemDensityJob(this, iter);  
      MyJobs.AddJob(pJob, "Fluid Density", &rtID );  
      uCurrentJob++;  
    }  
  }  
}
```

Replaced With:

```
DoDensity DensityRoutine(this);
```

```
tbb::parallel_for(tbb::blocked_range2d<int>( PaddedMin.j, PaddedMax.j, PaddedMin.i, PaddedMax.i), DensityRoutine);
```



Using parallel_for, cont...

```
class DoDensity
{
    FluidParticles *m_pFluidSim;
public:
    DoDensity(FluidParticles *pFluidSim) : m_pFluidSim(pFluidSim) {}

    void operator() ( const tbb::blocked_range2d<int>& r) const
    {
        FGXInt2 CellIndex;
        TAL_TRACE* pTrace = TAL_GetThreadTrace();
        TAL_BeginNamedTask(pTrace, "Fluid Density");

        for( CellIndex.i = r.rows().begin(); CellIndex.i < r.rows().end(); CellIndex.i++)
        {
            for( CellIndex.j = r.cols().begin(); CellIndex.j < r.cols().end(); CellIndex.j++)
            {
                FluidPackPartInfoVector &Cell =
                    m_pFluidSim->m_ParticleGrid.pPackGrid[CellIndex.i*m_pFluidSim->m_ParticleGrid.nBinWidth + CellIndex.j];
                FluidPackPartInfoVector::iterator end = Cell.end();
                for(FluidPackPartInfoVector::iterator iter = Cell.begin(); iter != end; iter++)
                    m_pFluidSim->ComputeDensitySSE2(iter);
            }
        }
        TAL_EndTask(pTrace);
    }
}
```



One more problem

- Fluid Sim has a few passes:
 - Density Calculation
 - Force computation
 - Update
- Each pass must be completed before other can begin
- Implemented via `waits()`, but this causes the entire CPU to halt before submitting the next step
- Instead, just need to fence between the stages
 - Can be accomplished via `task_groups`, or sub-spawning in `tbb`



FluidSim perf dramatically increased

- FluidSim went from 8.5 ms on quad core to 1.3 ms. 600% increase in performance.
- Before optimization with TAL, frame was CPU bound on Quad core.
- After optimization FluidSim was not blocking framerate



Multicore conclusion

- Can't just thread systems, have to design them well first
- Still some work to do on Civilization V, but lessons learned:
 - Put America back to work, make more jobs! Work up front = time saved later
 - Don't stick serial points just to make old serial code safe, write new code!
 - Any serial point is bad (mutexes, crit sections, spin locks)
 - Few threading heisenbugs
 - Decoupled systems easier to maintain
 - Easy scalability
- Achieved our goal: Civ 5 will not typically be CPU bound for rendering



Where to get GPA 3.0

Available now!

Free

For members of Intel's Visual Adrenaline Program (our free partner program for game developers).

www.intel.com/software/gpa



\$299

Through Intel Business Exchange. No membership required.

ibx.intel.com

