Using Games to Teach Parallelism to Computer Science Sophomore Students

Deepak Vembar, Rowena Turner, Amit Jindal
Intel Corporation
Agenda

- Teaching parallelism through game demos
  - Case study of a particle system

- Learn about performance profiling tools
  - Identify code bottlenecks to target optimization efforts

- Integrating code and lesson plans into sophomore curriculum
  - Preliminary evaluations from ASU, USC
What is a particle system?

Simulation phase - as fast as possible

- Spawn new particles
  - Initialize physical properties – position, orientation, velocity

- Check to see if particle lifetime exceeded
  - Remove particle from simulation

- Update physical properties for all particles
  - Position, velocity, orientation, angular velocity

- Collision detection (if needed)

Rendering phase - usually 30fps

- Draw each particle
  - Points, billboards, sprites
Physics update sequence

1. Compute forces due to wind, drag, gravity

2. Check for collision with the ground

3. Update tumble, flutter mode motion

4. Update position, normals

For all particles in the simulation
Particle systems: A good parallelism teaching tool

- Lots of parallel work
  - Each particle is separate, does not influence others

- Repetitive work being performed at each time step
  - Update physics for each particle
  - Compute collision for each particle
  - Update position of each particle
  - Render each particle

- Visual feedback of code changes
  - Improved fps through increased parallelism
Case study of particle system: Ticker Tape

- Particle system simulation in DX10
  - Falling leaves under gravity and wind effects
- Includes task-based threading with Intel TBB
  - Computation and updates broken into tasks
- Uses SSE for improved performance
  - UI features - thread count, # of particles, etc.
Using TickerTape in classroom

Target audience - undergraduate students (sophomore and beyond)
  Requirement: Understanding of C++

Problem: Existing code is complex to cover in 1-2 classroom sessions

Solution:
  - Use profiling tools for hotspot/ concurrency analysis
  - Identify hotspots - 50 lines or less
  - Make changes and verify improvement
Identify bottlenecks with Intel Amplifier XE

Hotspots with poor concurrency
Red: serial execution

No concurrent work, threads idling
Update VB to copy data from CPU to GPU
#pragma omp parallel for 
for(unsigned int i = 0; i < g_uNumActiveParticles; i++) { // do this computation for all the particles
    __declspec(align(16)) D3DXMATRIX mTransform, mTranslate, mRotate;
    float fScaleFactor = 0.15f;
    D3DXMatrixScaling( &mScaling, fScaleFactor, fScaleFactor, fScaleFactor );

    // Compute the combined translation/rotation matrix
    D3DXMatrixTranslation( &mTranslate, particles[i-start].pos_x, particles[i-start].pos_y, particles[i-start].pos_z );
    D3DXMatrixRotationYawPitchRoll( &mRotate, particles[i-start].rot_y, particles[i-start].rot_x, particles[i-start].rot_z );
    D3DXMatrixMultiply( &mTransform, &mScaling, &mRotate );
    D3DXMatrixMultiply( &mTransform, &mTransform, &mTranslate );

    // update all the vertices in the leaf
    for (unsigned int j = 4*i; j < 4*i+4; j++)
    {
        pContext->pVertexBuf[j].tex = g_vertices[j].tex;
        D3DXVec3TransformCoord( &(pContext->pVertexBuf[j].pos), &g_vertices[j].pos, &mTransform );
        D3DXVec3TransformNormal( &(pContext->pVertexBuf[j].normal), &g_vertices[j].normal, &mTransform );
    }
}
Analyzing improvement with OpenMP

TickerTape omp + TickerPhysicsSIMD omp

Hotspots with some concurrency
- **Red**: Poor
- **Orange**: OK
- **Green**: Best

Concurrent work
Analyzing improvement with TBB tasking

TickerTape tasking + TickerPhysicsSIMD tasking

Hotspots with ideal concurrency
Red: Minimal
Orange: OK
Green: Largest sections

Concurrent work
7.6x improvement in fps on high end desktop

<table>
<thead>
<tr>
<th>Copy VBs from CPU to GPU</th>
<th>Physics updates</th>
<th>fps (50000 particles, 6C)*</th>
<th>fps (10000 particles, 2C)+</th>
</tr>
</thead>
<tbody>
<tr>
<td>TickerTape10.cpp</td>
<td>TickerPhysicsSIMD.cpp</td>
<td>13</td>
<td>38</td>
</tr>
<tr>
<td>Serial</td>
<td>Serial</td>
<td>14</td>
<td>42</td>
</tr>
<tr>
<td>Parallel with OpenMP</td>
<td>Serial</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>Parallel with OpenMP</td>
<td>Parallel with OpenMP</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>Tasking with TBB</td>
<td>Tasking with TBB</td>
<td>99</td>
<td>77</td>
</tr>
</tbody>
</table>

* Testing carried out on a Gulftown Core i7 980X, with 4GB RAM and discrete graphics 6C-12T, and 50000 particles in the simulation, SSE ON

+ Testing carried out on a Core2Duo @2.8 GHz, 4GB RAM and 10000 particles
Suggested integration into curriculum

Classroom sessions
- Focus on API fundamentals, threading concepts
- Overview of the code, critical hotspots in the program

Lab sessions
- Compiling and running the game demos
- Understanding profiling tools and identifying bottlenecks

Homework and assignments
- Change serial code to introduce parallelism: OpenMP, Intel TBB
- Verify changes and performance improvement with code profilers
Preliminary results from select schools

2 schools integrated game demos into the curriculum for Spring 2011

- USC – OS, game design courses
- ASU – Integrated game demos into undergraduate courses
  CSE 430: Emphasis on threading, synchronization and deadlock

For more details, join us on Thursday 6:30-7:00pm, Kuskokwim Ballroom
Call to action

- Include graphics demos in academic curriculum
  - Visual feedback of additional parallelism through increased fps

- Comprehensive solution provided with TickerTape
  - Serial, OpenMP and task-based solution code, lesson plans, lab solutions

- Let us know what kind of demos you want to include in the classroom
  - gamedevinput@intel.com
Additional resources

Download academic-ready TickerTape source code and notes

Join the Intel Academic Community

Download latest graphics and game demos
  - Colony
  - Fireflies

Profiling tools
  - Intel Parallel Amplifier 2011
  - Intel Parallel Advisor 2011
  - Intel Parallel Inspector 2011
Backup
System requirements for TickerTape

To run:
• DX10 features graphics card
• Multi-core processor for visual differentiation (ideally)

To compile:
• DX SDK from Microsoft (tested with June, 2010)
• Visual Studio 2008 and beyond (sln files provided)
• Full source (provided)
Other resources

- Tools usage examples
  - Finding correctness issues using Intel® Parallel Inspector 2011
  - Identifying performance issues using Intel® Parallel Amplifier XE

- Intel® Cilk™ Plus solution with Ticker Tape
  - Simple Cilk keywords to achieve parallelism
  - Easier SIMD implementation using array notations, pragmas, and compiler notations