Efficient scaling in a Task-Based Game Engine

Leigh Davies, Intel
leigh.davies@intel.com

www.intel.com/software/gdceurope2011/
Agenda

- How do we want to program for multi-core?
- Introduction to tasking
- Building a Dependency Graph
  - Typical data flow
  - Overlapping Frames
- CPU post-processing
  - MLAA
- Conclusion
Why scaling to \( n \) cores is important

42% of PCs using Steam have 4+ cores*

Up from 26% last year

What can we get from more cores?

• More compute, which means…
• Improved visual fidelity, gameplay

* Physical Cores, data taken from https://store.steampowered.com/survey
## The Utopian Goals

<table>
<thead>
<tr>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic scaling (no algorithm/code changes) with number of cores</td>
</tr>
<tr>
<td>Algorithmic parallelism decoupled from machine parallelism</td>
</tr>
<tr>
<td>Engine systems can remain autonomous</td>
</tr>
<tr>
<td>Performance increases linearly with number of cores</td>
</tr>
</tbody>
</table>
Tasking - What is it?

Traditional “linear” model: All work items are executed sequentially on a single thread

- Thread 0
- Thread 1
- Thread 2
- Thread 3
Tasking - What is it?

With tasking: Work items are executed on multiple threads, as threads become available to perform work.
## Data flow for a frame

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
<th>Thread 2</th>
<th>Thread 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Tasking System Implementation

Tasking System

Tasking API

Scheduler

Application

Application logic

Task set 1

Task set N
Tasking API

- Used in Intel samples to implement a dependency graph, abstracts the base scheduler from the game code allowing alternative schedulers. (www.software.intel.com/en-us/articles/game-engine-tasking-animation)

- CreateTaskSet
  - Inputs:
    - Task callback function and dependencies
    - Task count, name
  - Returns TASKSETHANDLE and begins execution when dependencies are satisfied
  - TASKSETHANDLE used in future CreateTaskset calls to express dependency

- WaitForTaskSet
  - Main thread processes tasks until specified taskset completes

- Init/Shutdown/ReleaseHandle
Creating a Dependency Graph

\[ T0 = \text{CreateTaskSet}(\ldots); \]
\[ T1 = \text{CreateTaskSet}(\ldots); \]
\[ T2 = \text{CreateTaskSet}(\{ T0, T1 \}, \ldots); \]

- T0 and T1 will execute immediately
- T2 will not start to execute until T0 and T1 have completed
Task Function

• User-defined callback to execute one unit of work

• Parameters:
  – Data pointer global to the taskset
  – Task id (which task in the set is this)
  – Task count (how many total tasks are in the set)
  – Context ( [0, NumThreads] used for lock-free access to thread-specific data)
    • Ex: D3D11DeviceContext for multithreaded rendering
Making a good Task function

• Task length less than 5% of taskset time
  – Allows scheduler to load-balance
  – Scheduling overhead constant per task

• Experiment with task working set size
  – Be aware of issues with cache contention, etc.

• Prefer per-context data results to using InterlockedXXX
  – Aggregate per-context data with a dependent taskset if needed
  – InterlockedXXX expensive for memory intensive algorithms
Example: Computing average luminance

- Each task processes n-scanlines
- InterlockedXAdd causes sync point

```c
task* Task::execute(pData, TaskID, TaskSize, ContextID)
{
    LocalValue = Sum(....)
    InterlockedExchangeAdd(&gLuminance, LocalValue);
}
```
Example: Computing average luminance

- Use two tasksets
  - Compute sum per-context id
  - Sum per-context id sums and compute average

```c
// T0 Main
// T1 Worker0
// T2 Worker1
// T3 Worker2

task* Task::execute(pData, TaskID, TaskSize, ContextID)
{
    gLuminance[ContextID] += Sum(...);
}
```
A scheduler needs to
- Create and manage task worker threads
- Manage where tasks get executed

It’s a complex problem

Various options available:
- Intel Thread Building Blocks™ (TBB)
- Microsoft® Concurrency Runtime (ConCRT)
- Roll-your-own using standard threading
Data flow for a frame

- Simple system taskset (parallel)
- Complex system taskset (parallel)
- Complex system taskset (1 task)
- Simple system taskset (DX11 CmdLists)
- Execute CmdLists

<table>
<thead>
<tr>
<th>Main</th>
<th>N-1</th>
<th>N</th>
<th>N</th>
<th>N</th>
<th>N</th>
<th>N</th>
<th>N</th>
<th>N</th>
<th>N</th>
<th>N+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worker0</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N+1</td>
</tr>
<tr>
<td>Worker1</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N+1</td>
</tr>
<tr>
<td>Worker2</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N+1</td>
</tr>
</tbody>
</table>
Data flow for a frame

Main

Worker0

Worker2
**Execution flow for overlapped frames**

- **Simple system taskset** (parallel)
- **Complex system taskset** (parallel)
- **Complex system taskset** (1 task)
- **Simple system taskset** (DX11 CmdLists)

![Diagram](image-url)

**Execution flow details**:

- **Main**:
  - N-2
  - N
  - N
  - N
  - N
  - N
  - N
  - N
  - N
  - N
  - N
  - N

- **Worker0**:
  - N-1
  - N
  - N
  - N
  - N
  - N
  - N
  - N
  - N
  - N
  - N

- **Worker1**:
  - N-1
  - N
  - N
  - N
  - N
  - N
  - N
  - N

- **Worker2**:
  - N-1
  - N
  - N
  - N
  - N

**Execute CmdLists**
Implications of overlapped frames

- Buffers need to be duplicated or copied for the frame
  - Size can be limited with partial frame overlap
- Latency will increase by up to 1 frame
  - CPU submits previous frame to GPU while computing current frame
  - Use dependencies to control where overlap occurs
- Maximal benefit when combined with CPU load-balancing
  - If frame is GPU bound, move work to CPU
Beware! Serialization ahead…

• How to avoid it
  – Do not wait in a task
  – No Sleep, WaitForSingleObject, etc.
  – Don’t take locks

• How to mitigate:
  – Use taskset dependencies and context id
  – Post events to main thread and allow it to schedule tasksets
  – Use lock-free constructs
Serialization Not Always Obvious

• Implicit serialization:
  – Memory allocation (even CRT’s alloc/new)
  – Library calls that use locking

• Mitigation:
  – Pre-allocate memory, custom allocator, etc.
  – Instrument engine code (e.g. GPA Platform Analyzer)
  – Validate task running time is as expected using Platform Analyzer
Debugging your tasks

• Various tools available to help debug tasking
  – Use Platform Analyzer in GPA to visualize task execution
  – Instrument tasks to view where/when they execute
  – Instrument locking code for Platform Analyzer to see locks/waits in tasks

• Xperf can help see the bigger picture
  – See last year’s Gamefest talk and Bruce Dawson’s talk “How Valve Makes Games Better with Xperf”

• Make the entire frame a DG to prevent dependency confusion
Great I’m so fast I’m GPU bound!

- When GPUView shows the GPU is behind the CPU
  - Option 1: Increase fidelity of CPU based talks, it’s free!
  - Option 2: Move some GPU work back to the CPU
- Lots of options but post processing plays to CPU strengths
CPU post-processing sample

- Morphological antialiasing (MLAA) plus HDR processing
- Uses both CPU tasking and GPU to CPU pipelining
- Helper Pipeline class in MLAA sample to simplify scheduling of data transfer
• MSAA better than FSAA, but still brute-force
  – HW MSAA4x on PS3 not used, because of perf. cost
  – MSAA4x can be expensive on PC as well

• MLAA: new CPU-based antialiasing algorithm
  – Getting tons of traction, more games integrating, efforts to run on GPUs (drivers, SIGGRAPH talk,...)
The MLAA algorithm

- Two tasksets implement the algorithm
  - Find discontinuities between pixels in image buffer
  - Identify predefined edge patterns and blend weights. Blend colors in the neighborhood of these patterns

- Extra steps needed on PC/DX
  - Copy FB back and forth to CPU-accessible memory
MLAA Taskset 1

• Find pixels discontinuities
  – Do an horizontal pass, and a vertical pass
  – Horizontal pass check for discontinuities between rows
    • If found, pixel is marked as an edge pixel
  – Vertical pass is the same with two substitutions: row -> column and horizontal -> vertical
  – Step 2 and 3 also work with a horizontal, then vertical pass

• Instruction-level parallelism:
  – SIMD code is used to process multiple pixels at once

• Task-based parallelism:
  – Each task processes a block of 8 rows/columns
MLAA Taskset 2

• Identify predefined edge patterns
  – “walk” discontinuity flags
  – Compute discontinuity lines

• Most edges result in L-shaped patterns
  – Other types decompose to L patterns
• L-shapes have a primary segment (0.5+ pixels) and a secondary segment.
  – Connect the middle point of the secondary segment to the extremity of the primary segment
  – Forms a trapezoid with the pixel
• Area of the trapezoid is the blend weight for that pixel
MLAA Taskset 2

• Blend colors
  – Blending weights calculated for L-shape
  – Calculations are a bit more complicated for color images
    • Need minimize color differences at the stitching positions of different L-shapes.

• Once we are done with the blending passes, the color buffer is copied back to GPU memory
Pipelining GPU data to CPU

- D3D provides pipelining from CPU to GPU
- Application must pipeline GPU to CPU

1. Read-back RT from Frame n
2. Render Frame n+1 to RT
3. MLAA Frame n, update and Present
4. Read-back RT from Frame n+1
5. Render Frame n+2 to RT
A Frame moving through the pipeline

- Scene Render
- MLAA Post-Proc
- DMA+Present

CPU:
- Worker Threads
- Main Thread

GPU:
- Frame n
- DMA n

Present Frame n
MLAA Sample

Scene Stats

37 Draw Calls
50K Vertex Invocations
34K Primitive Count
650K PS Invocations
MSAAx4 and MLAA on SNB @ 1280x800

Scene complexity 1 to 100

- MSAAx4
- MLAA

(ms / frame)

- MSAAx4:
  - Scene complexity 1: 4.55 ms
  - Scene complexity 10: 98.2 ms

- MLAA:
  - Scene complexity 1: 4.45 ms
  - Scene complexity 10: 3.3 ms
MSAAx4 and MLAA on Corei7/5870 @ 1280x800

Scene complexity 1 to 100

- MSAAx4
- MLAA
Conclusion/call to action

• Task your systems to scale across the PC ecosystem

• Use dependencies
  – Data synchronization
  – Overlap frames

• Remove OS synchronization

• Use Platform Analyzer to visualize performance

• Check out the tasking samples for yourself!
Enter the proving ground for independent game demos.

LEVEL UP 2011

Turn your passion for games into game-industry glory. Enter your game demo to compete for industry recognition and up to 10,000 USD.

To learn more, visit:
www.intel.com/software/LevelUp2011

With the chance to sell your game on STEAM*

*Other names and brands may be claimed as the property of others.
Legal Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY RELATING TO SALE AND/OR USE OF INTEL PRODUCTS, INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT.

Intel products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Intel Corporation may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights that relate to the presented subject matter. The furnishing of documents and other materials and information does not provide any license, express or implied, by estoppel or otherwise, to any such patents, trademarks, copyrights, or other intellectual property rights.

Intel may make changes to specifications, product descriptions, and plans at any time, without notice. The Intel processor and/or chipset products referenced in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

All dates provided are subject to change without notice. All dates specified are target dates, are provided for planning purposes only and are subject to change.

Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

* Other names and brands may be claimed as the property of others.

Copyright © 2011, Intel Corporation. All rights reserved.
Intel® compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel® and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the “Intel® Compiler User and Reference Guides” under “Compiler Options." Many library routines that are part of Intel® compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel® compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel® and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.
Appendix I

Thread Building Blocks:
www.threadingbuildingblocks.org/

Graphics Performance Analyzers:
www.intel.com/software/GPA/

Visual Computing Home Page

Graphics Samples Home Page
Keep up to date with samples releasing throughout the year

Graphics Samples Page:

Sandy Bridge Samples Page:
http://software.intel.com/en-us/articles/sandy-bridge/
Appendix II

• MLAA Algorithm paper details
  • Developed and published in 2009 by Alexander Reshetov from Intel Labs
  • http://www.realtimerendering.com/blog/morphological-antialiasing/