Multi-Core/Player Game Loops

hannes@ru.is
Multi-Processor Game Loops

• 2004
  – Heat dissipation prevents faster CPUs!
  – Shift starts towards multi-core CPUs
  – Parallel processing techniques need to be adopted
• PS3 and XBox360 part of this development

• GOAL: Maximize hardware utilization!
  – Slow and painful shift, but mostly there now
XBox360 Hardware Architecture

- PowerPC Core 0
  - L1 Data
  - L1 Instr

- PowerPC Core 1
  - L1 Data
  - L1 Instr

- PowerPC Core 2
  - L1 Data
  - L1 Instr

- Shared L2 Cache

- Main RAM (512 MB)

- GPU
PS3 Hardware Architecture
5 Parallelization Techniques

- SIMD
- Fork and Join
- One Thread per Subsystem
- Job Scheduling
- Asynchronous Code
• Instructions that perform parallel operation on multiple data
  – e.g. Provided by XBox360 and PS3
• Typically 32 bit floating point data
• Good for 3D math code IF math library well encapsulated!
  – Do you have a function like d = dot(a,b) or are you always calculating d = a.x*b.x+a.y*b.y+a.z*b.z ?
2 of 5: Fork and Join

- **Divide work into smaller sub-units, distribute and finally merge results**
  - E.g. Blending animations with lerp: 5 characters with 100 joints each, resulting in 500 individual lerp operations divided N ways (on XBox360 either 3 ways or 6 ways, since each of the 3 cores can support 2 threads)

- Merge has to be programmed as well, may require waiting on a semaphore
3 of 5: One Thread per Subsystem

- Place specialized and fairly independent subsystems on separate threads
  - E.g. main loop, animation, dynamics, rendering
- May introduce restrictions on processor utilization!
One Thread per Subsystem
4 of 5: Job Scheduling

• Addresses the sub-system processor utilization restriction
• Work divided into multiple small, relatively independent jobs
• A job is essentially the pairing of data and code that operates on it
• Jobs are put on queue and picked up by the next available processing unit
• The PS3 SPURS job model does this!
4 of 5: Job Scheduling Approach
5 of 5: Asynchronous Code

- Design for possible wait until data is ready
- Do other stuff until you absolutely need the data
- Possibly not use data until next frame

See code fragment in book
Multiplayer Game Loops

• Typical architectures
  – Client-Server
  – Peer-to-peer
Client-Server

- Client does little more than player-prediction, rendering, audio and networking
- Server runs the game logic
- In client-on-top-of-server mode, the communication between client and server is made easier by putting both in same thread
- Client and server both serviced from same game loop, but at different rates!
Peer-to-Peer

• Each machine acts like the server for some objects and a client for others
• Authority over objects may migrate (e.g. when dropping and adding machines)
• Not always clear which code (server/client) is being run at any given time