Larrabee:
A Many-Core Intel® Architecture
for Visual Computing

Ronny Ronen
Sr. Principal Engineer
Intel - Mobility Group
February 3, 2009

Disclaimer – IMPORTANT

- All the material is taken from “Public Sources”
- No specific product information is provided (number of cores, frequencies, etc.).
- Performance numbers aims to demo trends and estimations and are not actual product numbers.
Agenda

• The Graphics Evolution
• Larrabee Architecture/Microarchitecture
• The Larrabee Parallelism story
• The Larrabee Graphics Story
Graphics Pipeline Evolution

Pre 1996 Customized Software Rendering

Pre 2001
Input Data
Transformation and Lighting
Primitive Setup
Rasterization
Pixel Processing
Frame Buffer Blend
Frame Buffer

DX8-DX10
Input Data
Vertex Shading
DX10 Geometry Shading
Primitive Setup
Rasterization
Pixel Shading
Frame Buffer Blend
Frame Buffer

Larrabee
Input Data
Vertex Shading
Geometry Shading
Primitive Setup
Rasterization
Pixel Shading
Frame Buffer Blend
Frame Buffer

Alternative Larrabee: Customized Pipeline

Software Rendering
Frame Buffer
Architecture Convergence

**Need:** CPU programmability & GPU parallelism
“GPU vs CPU”

• What to do with billions of transistors?

• CPU
  - **Latency** – Speedup the *single* thread workload
  - **Tactics:**
    - Cache (area limiting), prefetch, Speculative execution
    - limited by “perimeter” – dependencies, communication bandwidth
    - … multi-core comes later…

• GPU
  - **Throughput** – Speedup the *many* threads workload
  - **Tactics:**
    - Parallelism (1000s of threads)
    - Latency tolerance
    - limited by “area” – **compute density**

**Larrabee – Best of both**
Agenda

• The Graphics Evolution
• Larrabee Architecture/Microarchitecture
• The Larrabee Parallelism story
• The Larrabee Graphics Story
Cores for Throughput Tasks

• Simpler, Smaller cores
  – In order, no complex microarchitecture

• Longer vectors
  – 512b vs. 128b (16 vs. 4 elements)

• Vectorizable Instruction Set
  – Scatter & Gather
  – Multiply-Add
  – Predication

• More threads
  – 4 threads per core (vs. 2 in Intel® Core™ i7 Processor).
Hardware intro

- Lots of real IA (=X86) cores
- A wide/fast bidirectional ring bus
- Fully coherent cache hierarchy
- Fixed-function texture units
Processor Core Block Diagram

- Derived from Original Pentium
  - In order, dual issue

- Added vector units with separate registers
  - New instructions - 512 bit width
  - 16-bit predication registers
  - Gather/scatter instructions

- Direct connection to each core’s subset of the L2 cache

- Prefetch instructions load L1 and L2 caches

- 4 hardware threads per core
Vector Unit Block Diagram

- Vector complete instruction set
  - Scatter/gather for vector load/store
  - Mask registers select lanes to write, which allows data-parallel flow control
  - This enables mapping a separate execution kernel to each VPU lane

- Vector instructions support
  - Fast read from L1 cache
  - Numeric type conversion and data replication while reading from memory
  - Rearrange the lanes on register read
  - Fused multiply add (three arguments)
  - Int32, Float32 and Float64 data
Caches

- Each core has its own pair of caches
  - 32kbyte L1(data), 256kbyte L2
- All fully coherent with each other
  - L2s can share data with each other
  - Cores can co-operate on data quickly
- Lots of explicit cache controls
  - Controls for “non-temporal” streaming data
  - Prefetch, evict, make-LRU, initialize instructions
  - Combines the fine control of a scratchpad with the “soft edges” of a cache
Texture Sampler

• Fixed function texture sampler
  – Performs usual texture operations, including decompression, anisotropic filtering, etc.
  – Communicates with the cores via the L2 cache
  – Supports virtual address translation

• Why use fixed function texture logic?
  – Texture filtering needs specialized data access to unaligned 2x2 blocks of pixels
  – Filtering is optimized for 8-bit color values
  – Code would take 12x longer for filtering or 40x longer if texture decompression is required *

Key Differences from Typical GPUs

• Each Larrabee core is a complete Intel processor
  – Context switching & pre-emptive multi-tasking
  – Virtual memory and page swapping
  – Fully coherent caches at all levels of the hierarchy

• Efficient inter-block communication
  – Ring bus for full inter-processor communication
  – Low latency high bandwidth L1 and L2 caches
  – Fast synchronization between cores and caches

• Fixed function logic doesn’t get in the way
  – No backend blender between cores and memory
  – No rasterization logic between vertex and pixel stages
  – Result: flexible load balancing & general functionality
Agenda

• The Graphics Evolution
• Larrabee Architecture/Microarchitecture
• The Larrabee Parallelism story
• The Larrabee Graphics Story
Programming Current GPUs

- Current GPUs
  - Efficiently execute only wide data-parallel computation
  - Cannot submit work to self (reliant on CPU)
Programming Larrabee

- Larrabee
  - Efficiently executes *braided parallelism* that intermixes data- and task-parallelism with sequential code
  - Can submit work to itself
Data-Parallel Programming on Larrabbe

• Key ideas of data parallel programming
  – Define an array (grid) of parallel program invocations
  – Define groups within the grid that can share local memory

• Mapping program invocations on Larrabee
  – **Strand**: a program invocation that runs in one SIMD lane
  – **Fiber**: a SW-managed context that runs 16-64 strands
  – **Thread**: a HW-managed context. Can SW-switch among 2-10 fibers in order to cover long latencies (e.g. texture filtering)
  – **Core**: independent processor that runs all the strands in a group (their shared memory is in the core’s L2 cache)

• Scheduling choices are all under SW control

**Efficient Scheduling, Efficient Memory Access**
Data-Parallel Example

Core: Runs multiple threads (executes a strand group)

- Thread: HW-managed context (hides short unpredicted latencies)
- Fiber: SW-managed context (hides long predictable latencies)

16-Wide Vector Unit

...  

16-Wide Vector Unit

...  

More Fibers (typically 2 to 10, depending on latency being covered)

...  

More Threads (up to 4 per core, share memory via L1 & L2 caches)
Agenda

• The Graphics Evolution
• Larrabee Architecture/Microarchitecture
• The Larrabee Parallelism story
• The Larrabee Graphics Story
Graphics on Larrabee

• Industry-standards:
  – Direct3D, DX ComputeShader
  – OpenGL, OpenCL

• Larrabee Native mode:
  – Access full power and flexibility of LRB architecture
Larrabee Software Stack

- Development Environment
  - Compiler
  - Debugger
- Tools/Libraries
  - Performance Analysis Tools
- User Programs
  - Graphics Apps
  - Larrabee Native Apps etc.
- Utilities
  - Driver Control Panel

Application

API

Driver

Larrabee Hardware

Drivers

- DirectX
- OpenGL
- Larrabee Native C/C++
- PCIe/Display Driver
- Rendering Pipeline
- Larrabee Native App
- Driver Executive (μOS)
There Is No “Typical” Workload

Large variance between games & within frames

Tests use 25 widely spaced frames in each game


**F.E.A.R.* Per-Frame Breakdowns**


---

**Larrabee: practical solution to GPU limitations**
Graphics Rendering Pipelines

Pre 1996
Customized Software Rendering

Pre 2001
Input Data
Transformation and Lighting
Primitive Setup
Rasterization
Pixel Processing
Frame Buffer Blend
Frame Buffer

DX8-DX10
Input Data
Vertex Shading
DX10 Geometry Shading
Primitive Setup
Rasterization
Pixel Shading
Frame Buffer Blend
Frame Buffer

Larrabee
Input Data
Vertex Shading
Geometry Shading
Primitive Setup
Rasterization
Pixel Shading
Frame Buffer Blend
Frame Buffer

Alternative Larrabee: Customized Pipeline
Input Data
Software Rendering
Frame Buffer

Software Rendering
Frame Buffer
Larrabee’s Binning Renderer

- **Front end: vertices**
  - Processes multiple separate primitive streams in parallel
  - Puts results into a separate set of bins per primitive set

- **Back end: pixels**
  - Multiple tiles in parallel
  - Pixel data stored in L2 cache
  - Scoreboarding orders the pixel accesses within a tile
Bandwidth of binned rendering vs. immediate mode per frame

**Binning mode**
- Includes bin reads & writes
- Reads/writes each pixel once due to tiles in the L2 cache

**Immediate mode**
- Assumes perfect HeirZ cull
- Assumes 1MB each for the depth and color caches

Larrabee Scalability Over Cores

Efficient and scalable software renderer

Non-graphics Application Scaling

Flexible & programmable for many applications

Summary

• Need: CPU programmability & GPU parallelism
• Larrabee: practical solution to GPU limitations
• Larrabee: efficient and scalable SW renderer
• Larrabee: flexible & programmable for throughput applications

Larrabee: throughput computing architecture
Relevant Resources

• “Intel’s Larrabee Redefines GPUs”, Microprocessor report 29-09-08


• “Beyond Programmable Shading” workshop, SIGGRAPH 2008 (http://s08.idav.ucdavis.edu/):
  – “Programming Larrabee: Beyond Data Parallelism”, Aaron Lefohn, Intel
  – “Larrabee: Software is the New Hardware”, Tom Forsyth, Intel