Pangaea: A Tightly-Coupled Heterogeneous IA32 Chip Multiprocessor

Henry Wong¹, Anne Bracy², Ethan Schuchman², Tor M. Aamodt¹, Jamison D. Collins², Perry H. Wang², Gautham Chinya², Ankur Khandelwal Groen³, Hong Jiang⁴, Hong Wang²

henry@stuffedcow.net, anne.c.bracy@intel.com

¹Dept. Of Electrical and Computer Engineering, University of British Columbia
²Microarchitecture Research Lab, Microprocessor Technology Labs, Intel Corporation
³Digital Enterprise Group, Intel Corporation
⁴Graphics Architecture, Mobility Groups, Intel Corporation
Pangaea

- Integrates IA32 CPU with GPU cores
- Improved area/power efficiency
- Tighter integration
- Modular design
Motivation

• GPUs have low Energy Per Instruction
  – ~100x less EPI than CPU
  – Parallel performance too
  – Pangaea targets non-graphics computation for further efficiency gains

• Tightly-coupled
  – easier to program
  – lower communication latency

• Minimize changes to existing software (OS)
Overview

• Background on GPU Computation
• Pangaea: IA32-GPU chip multiprocessor
  – User-Level Interrupt mechanism
  – Architecture trade-offs
  – Prototype performance
• Conclusion
Programmable GPU

• Rendering pipeline
  – Polygons go in
  – Pixels come out

• DX10 has 3 programmable stages

Fixed Graphics Pipeline Stage
Programmable Graphics Pipeline Stage
Nvidia CUDA, AMD Stream

• Use shader processors without graphics API
• C-like high-level language for convenience
GPU + CPU

• Loosely-coupled to the CPU
  – Off-chip latency
  – Explicit data copy between memory spaces
  – Cooperation?
GPU Integration

• Put them on the same chip
  – Off-chip latency
  – Explicit data copy between memory spaces
  – Cooperation??
Pangaea

• Single-chip, tightly-coupled
  – Off-chip latency
  – Shared memory address space: **Share**, not copy
  – Cooperation!
Pangaea Architecture

- Tightly-integrated
  - User-level interrupts (ULI) for communication
  - Shared memory and cache
- Use GPU cores for compute
  - "Execution Unit" (EU)
Overview

• Background on GPU Computation
• Pangaea: IA32-GPU chip multiprocessor
  – User-Level Interrupt mechanism
  – Architecture trade-offs
  – Prototype performance
• Conclusion
EU Thread Life Cycle

Done!

Work

User Handler

EU

EU

CPU

12 Pangaea. PACT 2008
User-Level Interrupts (ULI)

- **EMONITOR**
  - Watches for an address invalidation
  - Calls user interrupt handler in response
- **ERETURN**
  - Returns from user-level interrupt handler
- **SIGNAL**
  - Tells Thread Spawner to start new thread.
Using ULI – CPU Code

{ 
    task_complete = false;
    EMONITOR(&task_complete, &handler);
    SIGNAL(&eu_routine, &eu_data);

    { Do some work }
}

Using ULI – EU Code

{
    Do some work;

    task_complete = true;
}

Using ULI – User Handler

```
handler() {
    if (task_complete) {
        Use EU result or start EU task
    }
    ERETURN();
}
```
ULI Pipeline Modifications

- Instruction cache
- (a) machine status register
- page table logic
- bus controller
- segment description cache
- (b) exception / interrupt unit
- execution unit
- address generation
- (c) instruction decoder
- (d) microcode
- data cache
- Floating Point Unit


tlbt

tlbt

tlbt

Accept new EMONITOR, ERETURN, SIGNAL instructions
Overview

• Background on GPU Computation
• **Pangaea: IA32-GPU chip multiprocessor**
  – User-Level Interrupt mechanism
  – *Architecture trade-offs*
  – Prototype performance
• Conclusion
Shared Memory Hierarchy

• Shared address space
  – *Address Translation Remapping*: CPU handles memory translation when EU TLB misses

• Shared memory hierarchy
  – Share a cache with the CPU
  – Helps collaborative multithreading
  – Avoids copying data between CPU and GPU
Area/Power Efficiency

• Graphics pipeline area is 9.5 cores
  – 65 nm synthesis of Intel GMA X4500
• Power is 4.9 cores
• Replace graphics pipeline with Thread Spawner
  – Thread Spawner is tiny: 1% of core
Overview

• Background on GPU Computation
• Pangaea: IA32-GPU chip multiprocessor
  – User-Level Interrupt mechanism
  – Architecture trade-offs
  – Prototype performance
• Conclusion
Pangaea Prototype

• Synthesis of production-quality RTL code
  – 2-issue, in-order IA32 CPU (37% of design)
  – 2 EUs from Intel GMA X4500 (31% x 2)

• Virtex 5 LX330, 136772 LUTs, 17 MHz
  – 66% of LX330

• Boots Linux, Windows, DOS, ...
Thread Spawn Latency

<table>
<thead>
<tr>
<th></th>
<th>GPGPU</th>
<th>Pangaea</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D pipeline</td>
<td>~ 1500</td>
<td>Bus interface</td>
</tr>
<tr>
<td>Thread Dispatch</td>
<td>15</td>
<td>Thread Dispatch</td>
</tr>
<tr>
<td>Total</td>
<td>~ 1515</td>
<td>Total</td>
</tr>
</tbody>
</table>

Thread Spawn latency reduced by 60x when bypassing graphics pipeline
- GPGPU driver software overhead not included
Throughput Performance

2 EUs vs. 1 CPU

- k-means and svm collaborate with CPU
- k-means is CPU-bound
Latency Sensitivity

- Bicubic and FGT code larger than 4KB i-cache
- k-means is CPU-bound
- Insensitive to memory latency < ~60 cycles
  - Can trade off level of memory hierarchy to share
Conclusions

• Added ULI communication to IA32, built on cache coherency mechanisms
  – Modularity allows scalable design

• Shared memory and cache is good for ease of programming and collaboration
  – Highest-performance implementation not critical

• Legacy graphics takes up 9.5 EUs of area, 4.9 EUs of power. Remove if not necessary.
  – Prototype shows it is ok to remove
Conclusions

• IA32 ULI built on cache coherency mechanisms enables scalable, modular design
• Shared memory and cache is good for ease of programming and collaboration
• Legacy graphics fixed functions have high overhead
Questions?
EU vs. CPU Peak Throughput

- 2 EUs have 2x peak performance vs. CPU
- TLP increases utilization (92% vs. 65%, linear)
- Large register file (57% vs. 7.4% memory, bicubic)
- Multiply-accumulate (55% of bicubic)
- SIMD-8/16 instructions lowers instruction count
Shaders

• For each vertex, run a program.
  - ...or each pixel
• Program instances mutually independent
• Shaders designed to run many independent instances of the same short program
GPGPU

• For each _____, run a program.
• Write shader programs to do something non-graphics
• Sparse matrix solvers, linear algebra, sorting algorithms...
• Brook for GPUs
Other Stuff from Intel MRL

• Papers
  – Multiple Instruction Stream Processor
  – EXOCHI: Architecture and Programming Environment for a Heterogeneous Multi-core Multithreaded System

• Ideas
  – User-level “sequencer” so OS isn't modified
  – Let CPU handle exceptions on behalf of “sequencer”
  – Shared memory space for ease of programming

• Pangaea can be thought of as extension of Exo
Pangaea Resource Usage

- 2-issue, in-order IA32 CPU
- 2 EUs from Intel GMA X4500
- Virtex 5 LX330, 136772 LUTs, 17 MHz

<table>
<thead>
<tr>
<th></th>
<th>LUTs</th>
<th>Registers</th>
<th>Block RAMs</th>
<th>DSP48 blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA32 CPU</td>
<td>50621</td>
<td>24518</td>
<td>118</td>
<td>24</td>
</tr>
<tr>
<td>EU Subsystem</td>
<td>84547</td>
<td>36170</td>
<td>67</td>
<td>64</td>
</tr>
<tr>
<td>Other</td>
<td>1604</td>
<td>591</td>
<td>91</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 2.2: Virtex-5 FPGA Resource Usage for the Pangaea configuration in Table 2.1.
Earlier Pangaea Prototype

- 1 CPU, 1 EU, 256 kB memory
- Virtex 4 LX200, 130352 4-LUTs, 17.5 MHz

<table>
<thead>
<tr>
<th></th>
<th>LUTs</th>
<th>Registers</th>
<th>Block RAMs</th>
<th>DSP48 blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>x86 CPU</td>
<td>68949</td>
<td>27136</td>
<td>118</td>
<td>29</td>
</tr>
<tr>
<td>EU subsystem</td>
<td>59245</td>
<td>21189</td>
<td>32</td>
<td>40</td>
</tr>
<tr>
<td>Other</td>
<td>2158</td>
<td>634</td>
<td>129</td>
<td>1</td>
</tr>
</tbody>
</table>