1401 HETEROGENEOUS HPC
How Fusion Designs Can Advance Science

Ben Bergen
Los Alamos National Laboratory
Research Scientist

Marcus Daniels
Los Alamos National Laboratory
Research Scientist
VPIC Team

Brian J. Albright
Los Alamos National Laboratory
Research Scientist

Kevin J. Bowers (primary code author)
Los Alamos National Laboratory
Guest Scientist

Ben Bergen
Los Alamos National Laboratory
Research Scientist

William S. Daughton
Los Alamos National Laboratory
Research Scientist

Lin Yin
Los Alamos National Laboratory
Research Scientist
OVERVIEW | What I am going to talk about…

- Modeling Laser Plasma Interactions (LPI) with VPIC
  - Historical perspective on heterogeneous computing
  - What does VPIC actually do? *(content courtesy of Kevin Bowers)*
  - Simple observations on programming model and application mapping
  - Some lessons learned
- Fusion vs. Discrete GPUs
  - What can we expect?
- First Experiences with the G-T56N Prototype
  - What do we actually get?
- Conclusions
HETEROGENEOUS HPC | Historical Perspective

- **2002**: Dark Horse Cell, 3D RAM
- **2003**: Advanced Algorithms LDRD GPU, FPGA
- **2004**: Roadrunner Skunkworks Clearspeed, Cell
- **2007**: Intel Sandy Bridge Development Begins
- **2008**: AMD Fusion Announced
- **2008**: OpenCL 1.0 11/18/2008
- **2009**: CUDA
- **2010**: Intel Core i7 1/9/2011
- **2011**: Brazos 1/5/2011
- **2011**: NVIDIA Project Denver
Indirect-drive fusion ignition experiments are underway at the National Ignition Facility (NIF) at Lawrence Livermore National Laboratory.

192 lasers fire into a hohlraum to compress a Deuterium-Tritium capsule through soft x-ray ablation.
PARTICLE-IN-CELL (PIC) METHODS | Plasma Physics Simulations

- One application of VPIC is the simulation of Laser Plasma Interactions (LPI), which can affect energy delivery in ICF experiments
- Several difficulties arise during the compression of a hohlraum capsule
  - Laser scattering: Not enough energy to compress the capsule
  - Laser scattering: Laser does not target the desired areas (compression is not symmetric)
  - Pre-Heating: Electrons heat plasma making compression more difficult
- Understanding how much energy is delivered to the DT capsule is critical for a successful fusion ignition
PARTICLE-IN-CELL METHOD | Overview

Spatial Domain

Time Iteration

- Accumulate Currents
- Update Fields
- Advance Particles
- Interpolate Field Effects

Advance Particles

Interpolate Field Effects

Update Fields

Accumulate Currents

Time Iteration

Spatial Domain

cells

particles
WHAT DOES VPIC REALLY DO? | Slide courtesy of Kevin J. Bowers

Initial State

Read:
Write:
Compute:
32 bytes
0 bytes
0 flops

PIC: Particle In Cell
WHAT DOES VPIC REALLY DO? | Slide courtesy of Kevin J. Bowers

Initial State
Interpolate $E$ and $B$

Read: 72 bytes
Write: 0 bytes
Compute: 27 flops
WHAT DOES VPIC REALLY DO? | Slide courtesy of Kevin J. Bowers

Initial State
Interpolate $E$ and $B$
Update $u$

Read: 0 bytes
Write: 0 bytes
Compute: 107 flops
WHAT DOES VPIC REALLY DO? | Slide courtesy of Kevin J. Bowers

Initial State
Interpolate $E$ and $B$
Update $u$
Compute Motion

Read: $0 + 48$ bytes
Write: $0 + 48$ bytes
Compute: $42 + 70$ flops
WHAT DOES VPIC REALLY DO?  

Slide courtesy of Kevin J. Bowers

Initial State
Interpolate $E$ and $B$
Update $u$
Compute Motion
Update $r$ and $J$

Read: 56 bytes
Write: 48 bytes
Compute: 168 flops
WHAT DOES VPIC REALLY DO? | Slide courtesy of Kevin J. Bowers

Initial State
Interpolate $E$ and $B$
Update $u$
Compute Motion
Update $r$ and $J$

Update $r$ and $J$

Read: 56 bytes
Write: 48 bytes
Compute: 168 flops
WHAT DOES VPIC REALLY DO? | Slide courtesy of Kevin J. Bowers

Initial State
Interpolate $E$ and $B$
Update $u$
Compute Motion
Update $r$ and $J$
Update $r$ and $J$
Update $r$ and $J$

Read:
Write:
Compute:
56 bytes
48 bytes
168 flops
WHAT DOES VPIC REALLY DO? | Slide courtesy of Kevin J. Bowers

Initial State
Interpolate $E$ and $B$
Update $u$
Compute Motion
Update $r$ and $J$
Update $r$ and $J$
Update $r$ and $J$
Final State

Read:
0 bytes
Write:
32 bytes
Compute:
0 flops

Net Read:
152+ 56$n_c$ bytes
Net Write:
80+ 48$n_c$ bytes
Net Compute
246+168$n_c$ flops
WHAT DOES VPIC REALLY DO? | Slide courtesy of Kevin J. Bowers

Initial State
Interpolate $E$ and $B$
Update $u$
Compute Motion
Update $r$ and $J$
Update $r$ and $J$
Update $r$ and $J$
Final State

Requires $\sim$1 byte/flop

| Operation       | Read: 0 bytes | Compute: 0 flops | Net Read: 152+ 56$n_c$ bytes | Net Write: 80+ 48$n_c$ bytes | Net Compute: 246+168$n_c$ flops |
Naïve initial particle distribution by cell places particle data spatially “close” in memory
Contiguous Memory

Compute Grid

Advancing particles potentially moves them into new cells
Contiguous Memory

Compute Grid

New particle positions interleave memory access with respect to cells
PARTICLE SORTING / PIC Requires Unstructured Data Access

Contiguous Memory

Compute Grid

After several time iterations, particle data has lost spatial locality
PARTICLE SORTING | PIC Requires Unstructured Data Access

Contiguous Memory

Compute Grid

Loss of spatial locality in data access impacts temporal access of field data and hurts performance
PARTICLE SORTING | PIC Requires Unstructured Data Access

Contiguous Memory

Compute Grid

Numbering indicates original indices

Sorting particle data by cell restores spatial/temporal locality
VPIC ON ROADRUNNER | Adaptation to IBM Cell eDP

- **Particle Advance ➔ SPEs**
  - Optimized data structures (processed in blocks of $512 \times 32$ bytes (16 KB DMAs)
  - SPU ISA (128-bit SIMD)
  - Hand-optimized loop unrolling (x4)
  - Software cache for pre-computed field interpolants

- **Field Advance and Particle Sorting ➔ PPE**
  - 4% of overall computational time (as measured on CISC implementations)
  - Optimal sorting frequency

Original VPIC code was already highly optimized for single-precision throughput on modern multicore processors
Particle advance accelerated 15.7x

Overall Speed Up: 10x

Amdahl’s Law:
Rest of code relatively slower
Particle advance accelerated 15.7x

96%  4%
Overall Speed Up: 5.6x

Before  After

Amdahl’s Street Justice: Rest of code absolutely slower!!!
Particle advance accelerated 15.7x

96%

4%

Overall Speed Up: 5.6x

Before

After

4% 6% 12%

Amdahl’s Street Justice:
Rest of code absolutely slower!!!

Poor provisioning of the PPE reduced the utility of the Cell’s fusion design!!!
Particle advance accelerated 15.7x

96%
4%
Overall Speed Up: 5.6x
6% 12%

Before After

Amdahl’s Street Justice:
Rest of code absolutely slower!!!

Ultimately, solution was to adapt sorting and field advance to the SPEs ➔ Unpleasant!!!
COMPARING SYSTEMS | Discrete GPU vs. Fusion

- DDR3
- GDDR5
- PCIe
- Host
- GPU
- DDR3
- Host
- GPU
COMPARING SYSTEMS | Discrete GPU vs. Fusion

- Pros
  - High memory bandwidth on device
  - Many cores on device → high throughput
  - Low invocation latency on device

- Cons
  - Data motion across PCIe bus
  - Effectively doubles memory usage:
    - Host copy
    - Device copy
  - Handling multiple address spaces adds complication

\[
T_D = T_{ED} + T_{H \rightarrow D} + T_{CPD} + T_{H \leftarrow D}
\]
**COMPARING SYSTEMS | Discrete GPU vs. Fusion**

- **Pros**
  - Shared address space
  - Single copy of state data
  - More like SMP at kernel invocation level

- **Cons**
  - Reduced memory bandwidth to device
  - Fewer cores on device
  - Currently high invocation latency

\[ T_F = T_{EF} + T_{CPF} \]
COMPARING SYSTEMS | A Few Conjectures

- Discrete GPU
  - Should work best for arithmetically intensive kernels executed on large data buffers
  - Amount of computational work and superior memory subsystem outweigh data motion costs
    - \( T_{ED} + T_{H \rightarrow D} + T_{H \leftarrow D} << T_{CPD} << T_{EF} + T_{CPF} \)
  - Conjecture: We can afford to use a function offload execution model

- Fusion
  - Should work best for less arithmetically intensive kernels executed on small data buffers
  - Reduction in data motion costs outweigh poor efficiency and inferior memory subsystem
    - \( T_{EF} + T_{CPF} << T_{ED} + T_{H \rightarrow D} + T_{D \leftarrow H} + T_{CPD} \)
  - Conjecture: Need heterogeneous execution model

- NOTE: In both cases, it may be that \( T_{CPD} << T_{CPF} \)
# Fusion Hardware | G-T56N APU Details

<table>
<thead>
<tr>
<th>Specification</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microarchitecture (Platform)</td>
<td>Bobcat (Brazos)</td>
</tr>
<tr>
<td>CPU Cores (Frequency)</td>
<td>2 (1.6 GHz)</td>
</tr>
<tr>
<td>GPU Cores (Frequency)</td>
<td>80 (500 MHz)</td>
</tr>
<tr>
<td>Memory (Width)</td>
<td>Single-Channel DDR3 (64-bit)</td>
</tr>
<tr>
<td>L1 Cache (Associativity)</td>
<td>2x32 KB inst. (2-way) 2x32 KB data (8-way)</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>1024 KB (16-way)</td>
</tr>
<tr>
<td>Memory Bandwidth (Memory Freq)</td>
<td>8.5 GB/s (1066 Mhz)</td>
</tr>
<tr>
<td>SP Theoretical Peak</td>
<td>80 GF/s</td>
</tr>
<tr>
<td>Thermal Design Power</td>
<td>18 W</td>
</tr>
</tbody>
</table>
FUSION HARDWARE | Test System

Fusion System

Stand ;-)
## TEST HARDWARE | Summary

<table>
<thead>
<tr>
<th>Device</th>
<th>Fusion G-T56N</th>
<th>FirePro 3D 7800</th>
<th>Tesla C2050</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microarchitecture</td>
<td>Bobcat</td>
<td>Cypress</td>
<td>Fermi</td>
</tr>
<tr>
<td>GPU Cores (Frequency)</td>
<td>80 (500 MHz)</td>
<td>1440 (700 MHz)</td>
<td>448 (1150 MHz)</td>
</tr>
<tr>
<td>Memory (Width)</td>
<td>Single-Channel DDR3 (64-bit)</td>
<td>Quad-Channel GDDR5 (256-bit)</td>
<td>Dual-Channel GDDR5 (384-bit)</td>
</tr>
<tr>
<td>Memory Bandwidth (MemFreq)</td>
<td>8.5 GB/s (1 GHz)</td>
<td>128 GB/s (1 GHz)</td>
<td>144 GB/s (1.5 GHz)</td>
</tr>
<tr>
<td>SP Theoretical Peak</td>
<td>80 GF/s</td>
<td>2016 GF/s</td>
<td>1030 GF/s MAD (Mul+Add)</td>
</tr>
<tr>
<td>Thermal Design Power</td>
<td>18 W</td>
<td>138 W</td>
<td>238 W</td>
</tr>
</tbody>
</table>
Start with large host buffer

Data Array

Device
TEST PROBLEM | Data Motion Cost vs. Computational Efficiency

Data Array

Start with large *host* buffer

Compute random offset
Data Array

Start with large *host* buffer
Compute random offset
Write buffer to device

Fusion version uses zero-copy available with AMD APP SDK 2.4
Data Array

```c
__kernel void simple(__global float * data,
            __const float mult, __const unsigned
            count) {
    unsigned gid = get_global_id(0);
    float sum = 0.0;

    for(i=0; i<count; ++i) {
        sum += i*mult;
    } /* for */

    data[gid] = sum;
} /* simple */
```
**TEST PROBLEM | Data Motion Cost vs. Computational Efficiency**

Data Array

Start with large *host* buffer
Compute random offset
Write buffer to device
Execute OpenCL kernel
Read updated buffer

Fusion version uses zero-copy available with AMD APP SDK 2.4
Clear preference for specific buffer sizes

- Mapped buffers simply do not work for ATI
- Step function
  - Correlate to core full subscription
- Quadratic fall-off
  - Can’t explain this
  - We think that it should be linear
- Qualitatively similar behavior for NVIDIA (explicit)
- Mapping works well for NVIDIA driver
- Similar drop-off beyond full SM subscription
- Four-socket (48 cores)
- Mapping works well for CPU
- Drop-off
  - Hints at scheduling overhead and not hardware

NOTE: log-log plot
CONCLUSIONS

- Require more careful analysis of real codes
  - Hybrid Parallel Gas Dynamics Project (HyPGaD)
  - VPIC
    - Simple addition: use OpenCL event profiling to normalize results
    - Vectorization of kernels
- Llanos systems coming this week
- Hardware is clearly trending towards heterogeneous architectures
  - New programming models need to increase task-concurrency
Disclaimer & Attribution

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. There is no obligation to update or otherwise correct or revise this information. However, we reserve the right to revise this information and to make changes from time to time to the content hereof without obligation to notify any person of such revisions or changes.

NO REPRESENTATIONS OR WARRANTIES ARE MADE WITH RESPECT TO THE CONTENTS HEREOF AND NO RESPONSIBILITY IS ASSUMED FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

ALL IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED. IN NO EVENT WILL ANY LIABILITY TO ANY PERSON BE INCURRED FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AMD, the AMD arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. All other names used in this presentation are for informational purposes only and may be trademarks of their respective owners.

The contents of this presentation were provided by individual(s) and/or company listed on the title page. The information and opinions presented in this presentation may not represent AMD’s positions, strategies or opinions. Unless explicitly stated, AMD is not responsible for the content herein and no endorsements are implied.