Can FPGAs accelerate Data Analytics?  
A Skeptic's View

Maya Gokhale
maya@llnl.gov
Supercomputing application landscape

- Simulations
  - physical phenomena
  - physics-based codes
  - behavior of physical entities in space and time
  - run single large problem with high interaction between parts

- Analysis
  - Bioinformatics - BLAST, genomic expression
  - Financial predictions - Monte Carlo methods to (eg.) price derivatives
  - Text mining
  - Database search
  - Run large number of independent problems on compute/data intensive backend processors
  - Gather results at front end

To benefit from acceleration, applications must follow 90-10
90-10 is hard to find in scientific simulations

- Acceleration using only library routines will be negligible for scientific codes.
  - Even Linpack needs at least 5X DGEMM acceleration of 64-bit floating point operations
- Acceleration of compute kernels is problematic.
  - Long, complex double precision code sequences: not a good fit for FPGA
  - Collection of little hot spots whose data structures are enmeshed in surrounding serial code

- The Amdahl’s Law limitation also applies to other co-processors.
Assessing FPGAs as simulation co-processors

- Study execution profile
  - oprofile, PAPI, TAU
  - quantify time spent at loop or even line granularity
  - find representative data sets
    - execution profile may vary greatly depending on data set
  - want 90% time in a small kernel, preferably library function

- Study code of likely acceleration candidates
  - data type - integer, single precision FP, double FP
  - types of operations - divides, transcendental functions
  - numbers of operations - how many FP units are needed

- Study data profile
  - data consumed and produced in a region must be communicated between global microprocessor memory and FPGA board memory
  - need to know amount of data transferred (per loop iteration)
  - need to know if communication and computation can be overlapped

- ERSA 2005, RSSI 2006
## Scientific simulation profiles

<table>
<thead>
<tr>
<th>Code</th>
<th>Compute Kernel %</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINPACK</td>
<td>74%</td>
<td>DGEMM</td>
</tr>
<tr>
<td>MILC</td>
<td>55%</td>
<td>Collection of matrix algebra routines</td>
</tr>
</tbody>
</table>
| GAMESS   | 40%              | EDIMER
Large, complex routine |
| POP      | 10%              | Conjugate gradient solver                    |
| GROMACS  | 55%              | In 5 routines, already optimized in sse       |

**Speedup of 2X-3X at the very most**
FPGAs for analysis problems

- Signal and image processing
  - Integer and single precision FP
  - Amenable to streaming, pipelining
  - Compute within the data acquisition pipeline
  - Lots of working, real world implementations, eg. Cibola Flight Experiment with 9 Virtex 1000’s for signal processing on-board a satellite, launched March 2007

Los Alamos 09 March 2007 A small-but-smart satellite experiment, the Cibola Flight Experiment (CFE) developed at Los Alamos National Laboratory for the Department of Energy's National Nuclear Security Administration (NNSA), has launched aboard a United Launch Alliance Atlas-5 rocket and was successfully placed in orbit 350 miles above Earth. The satellite will test leading-edge technologies that will be incorporated into future generations of satellites that will monitor the globe for nuclear detonations.
Discussion

- Science mission to study lightning, ionospheric disturbances, other sources of RF atmospheric noise
- Understand effects of ionosphere on communications

- Algorithm development was difficult
  - needs team of scientists, embedded systems programmers, hardware designers
  - tried tools such as simulink
    - custom board made integration within existing platform difficult
Graph algorithms on FPGA

- Graph algorithms
  - Point to point shortest path on road network (a la mapquest): **Zach Baker** at LANL
  - All-to-all shortest path of very large sparse semantic graphs: **Scott Kohn** and **Andy Yoo** at LLNL
Example: Point to point shortest path

- Used for route planning in TranSIMs, simulating road traffic on road networks in large metropolitan areas
- Opportunity for parallel execution of route planning
  - 100K - 1,000K routes to compute
  - Can complete in nearly arbitrary order
  - Lat/long of all road nodes provided
- Latency dominated computation
  - Mitigate through application-specific multi-threaded approach
- Implemented on Cray XD1 node
  - Dual Opteron + Xilinx Virtex2Pro50 FPGA for every
  - RapidArray HT connection to FPGA

IEEE FCCM 2007
Point-to-point shortest path

- **A* algorithm**
  - Uses distance to destination to decide which possible paths to pursue
  - Hardware-friendly priority queue implementation needed
  - Bandwidth to road network graph critical to performance

```python
add_queue(start, 0)
while(queue != empty)
    u = extract_min(queue)
    if(u.explored == true)
        continue
    elseif(u == dest)
        return path
    else
        foreach(edge (u,v) out from u)
            d(v) = distance(v, dest)
            add_queue(v, path_dist + d(v))
            previous[v] = u
            u.explored = true
```
Adaptations for hardware: priority queue

- Software A* uses Fibonacci Heap
  - $O(\log(n))$ average performance
  - Complicated data structure
  - Heap can grow without bound during execution
- Hardware approach uses bubble sort!
  - Needs only a single memory port
  - Sort speed not as important: parallel A* units
  - Buffer limited to 32 entries, determined by analysis of data set
Parallel A* units

- Each unit contains
  - Distance calculation block
  - Bubble sort hardware block to sort the queue
- Customized cache in memory controller
  - Cache from DRAM into QDRII SRAM
  - LRU page replacement
Performance

<table>
<thead>
<tr>
<th>Num Units</th>
<th>Area (slices)</th>
<th>Area (%)</th>
<th>Mult (of 232)</th>
<th>BRAM (of 232)</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>18165</td>
<td>76</td>
<td>48</td>
<td>112</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Time (sec)</th>
<th>Loads</th>
<th>Rate/sec</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>35</td>
<td>2024294 edges</td>
<td>86376</td>
<td>1</td>
</tr>
<tr>
<td>XD-1 (DRAM)</td>
<td>0.68</td>
<td>778 blocks</td>
<td>2925280</td>
<td>50</td>
</tr>
</tbody>
</table>

- Compare Opteron only to Opteron + FPGA
- FPGA algorithm uses Opteron’s DRAM to load SRAM cache
- Access to 2024294 edges, with 778 page loads
Discussion

- A **50X** speedup for a latency-driven random-access algorithm truly demonstrates a disruptive technology

**BUT …**

- Hardware implementation in VHDL by experienced hardware/software designer
- Six months to build/debug hardware
- Needs considerable expertise with FPGA, CAD tools, board level architecture, system level architecture, software algorithms and their implication for hardware
- Needs coordination between software and hardware
- Hard to get out into general use due to expense of custom hardware
Example: Path finding in semantic graphs

Semantic graphs are used to analyze relationships in large data sets coming from heterogeneous data sources.

How is A related to B?
Is a certain activity pattern in the data?

We need to analyze graphs that are orders of magnitude larger than those processed using current technology, $10^{12}$ nodes, faster - in minutes instead of days.


T. Coffman, S. Greenblatt, S. Marcus, Graph-based technologies for intelligence analysis, CACM, 2004
Database storage appliance

**Bulk data movement:** 250 GB/hour - uncompressed (1 TB/hour Target)

**ODBC 3.X**
**JDBC Type 4**
**SQL/92**

**Streaming data, joins, & aggs @ 50MB/sec**

**Scalability**

**SMP Host**

**Netezza Performance Server**

**Intelligent Query Streaming™**
Methodology

- Represent graph as SQL table where each row represents an edge
- Pose the shortest path problem as SQL query
- Query is optimized on host
- Sub-queries dispatched to all the FPGA/PPC processors who read their part of the database and filter the table rows
- Rows matching the query are returned back to host
- User sees SQL interface only
- SC 2007
### FPGA-accelerated storage server vs. BG/L

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Graph Edges</strong></td>
<td>30 billion</td>
<td>300 billion</td>
</tr>
<tr>
<td><strong>Graph Description</strong></td>
<td>Random (academic problem)</td>
<td>Scale-Free (“real world” problem)</td>
</tr>
<tr>
<td><strong>Processors</strong></td>
<td>65, 538</td>
<td>648</td>
</tr>
<tr>
<td><strong>Avg. Search Time</strong></td>
<td>1.4 sec</td>
<td>218 sec</td>
</tr>
<tr>
<td><strong>Level of Effort</strong></td>
<td>6 months, 2000 lines of C code</td>
<td>2 week, 100 lines of SQL code</td>
</tr>
</tbody>
</table>

- **Bi-directional Breadth-First Graph Search Algorithm**
  - 10X More Edges
  - 12X Productivity Improvement
  - 300 Billion Edge Problem Not Achievable on BG/L
Discussion

- **Pros**
  - Appliance approach hides existence of FPGA from user
  - Familiar software-oriented interface
  - No need to do synthesis, place, and route

- **Cons**
  - **Cost**
  - Limited application classes
  - Closed source, not extensible

- **But …**
  - Software package tailored to graph algorithms runs on commodity cluster, beats Netezza parallel architecture on representative graph queries
Comparison of various co-processors

- Matched filter over hyperspectral imagery
  - Locate geographic, atmospheric features
  - Wide spectral content, divided into 100’s of bands
  - Large data cubes (e.g., 240 x 240 x 1000) collected in real time

- Compare Cell, FPGA, and GPU

- Justin Tripp, Zach Baker of LANL (FCCM, 2007)
Hyperspectral imagery applications

Geology

Mineral Detection

Wetland Studies

Atmospherics
Matched filter algorithm

<table>
<thead>
<tr>
<th>Time break down for each task (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load File</td>
</tr>
<tr>
<td>Copy Data</td>
</tr>
<tr>
<td>Whiten Signature</td>
</tr>
<tr>
<td>Mean Subtraction</td>
</tr>
</tbody>
</table>

Processing of each signature takes 3.05 seconds on 3.2GHz Xeon.

<table>
<thead>
<tr>
<th>Impl.</th>
<th>Read File</th>
<th>Transpose</th>
<th>Mean Calc.</th>
<th>Mean Sub.</th>
<th>Dot Product</th>
<th>Scalar Ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>GPU</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CELL</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>FPGA</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Speedup over CPU</th>
<th>Time per Signature</th>
<th>Speedup per $k$</th>
<th>Speedup per kWatt</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPGA (V2Pro50)</td>
<td>3.91</td>
<td>0.78 sec</td>
<td>.39</td>
<td>11.2</td>
</tr>
<tr>
<td>GPU (nv 7900)</td>
<td>3.1</td>
<td>1.0 sec</td>
<td>1.24</td>
<td>8.86</td>
</tr>
<tr>
<td>Cell (3GHz)</td>
<td>8.0</td>
<td>0.38 sec</td>
<td>1.0</td>
<td>25.4</td>
</tr>
<tr>
<td>CPU (3GHz Xeon)</td>
<td>1</td>
<td>3.05 sec</td>
<td>.5</td>
<td>3.33</td>
</tr>
</tbody>
</table>
Discussion

- FPGA was fastest on the portions it could do, but our implementation couldn’t do all the parts
  - soft processor could be used, but
  - takes area, memory, bandwidth from replicated data paths
- Cell was overall fastest, but data had to be reorganized to work on the Cell version of the algorithm
- GPU was judged over all “winner” in terms of speedup/$
  - newer GPU architectures are even more amenable to general purpose data intensive processing
    - GPU/CUDA used to calculate Reed-Solomon codes for RAID (Sandia 2008)
Summary

- FPGA co-processors have shown orders of magnitude speed up on certain problems, BUT
- Application domains must be carefully chosen to circumvent Amdahl’s Law
- Cost of FPGAs and associated algorithm development tools is too high for most applications
- Design-code-test cycle is too long
- Multi-core processors are competitive to FPGAs, especially for floating point dominated kernels, and may overtake FPGAs for algorithm acceleration
- Good niche for FPGA algorithm acceleration might be
  - part of data acquisition pipeline
  - high performance appliance
  - low power/energy scenarios