Argonne Training Program on Extreme-Scale Computing (ATPESC)

Presented to
ATPESC 2017 Participants

Peter Kogge
Data Intensive Computing, the 3rd Wall, and the Need for Innovation in Architecture

Q Center, St. Charles, IL (USA)
Date 08/04/2017

When Do We Need New Architectures

• Long-lasting architectural advances occur when a “wall” must be overcome
  • 1st Wall – Mid 90s: the **Memory Wall**
  • 2nd Wall – 2004: the **Power Wall**
  • 3rd Wall – Now: the **Locality Wall**

And this is largely due to emergence of apps with **Data Intensive Characteristics**
What Do I Mean by Data Intensive?

• Computation dominated by data access & movement – not flops
• Large sets of data often persistent
  – but little reuse during computation
• No predictable regularity
• Significantly different scaling
• Streaming becoming important

The “Locality” we have come to expect from our apps is disappearing
This Talk

• Moore’s Law and the Prior Walls
• Today’s Architectures
• Evidence of a New “Locality” Wall
  – Benchmarks
  – A Big Data Application
• Migrational Computing: a Possible Architectural Fix
Technology, Moore’s Law, and Beyond

• Moore’s Law: 2D transistors get smaller & faster
  – From 10um to 5nm feature size: \textbf{2,000X smaller & faster}

• Cores get smaller, faster, lower power
  – Power density approx. constant \textit{as long as} $V_{dd}$ declines

• Memory arrays get denser
  – To maximize density, access time \textit{drops at best slowly}
  – Can increase bandwidth, but power skyrockets

• After Moore’s Law: \textbf{we’re going 3D!}
  – With a mix of die types

http://www.micron.com/products/hybrid-memory-cube
The Memory Wall (mid 1990s)

- Core clock speeds outran memory latency

- **Breaking the Wall**: Use extra transistors for
  - Bigger on-chip SRAM caches
  - More ILP to find more memory accesses
  - Add additional floating point capability

- Enablers: Applications had *plenty* of locality

- Example: $Ax=b$, $A$ is large, dense, matrix
  - Tremendous temporal locality
  - Assume caches can save $nxn$ patch of $A$
  - $O(n^2)$ to read $nxn$ patch of $A$ to cache
  - $O(n^3)$ operations on this patch

- With **big enough cache**, don’t care *how slow* memory is
The Power Wall (2004)

• Flattening $V_{dd}$ increased power density
  – Bigger chips meant more logic to dissipate

• Result: at 120Watts, cooling uneconomical

• Breaking the wall:
  – *Lower* the clock rate
  – *Use multiple* simpler cores
  – *Increase* SIMD-style parallelism

• Side-effect: need more bandwidth

• Solution for dense apps: again *bigger caches*
2004: Emergence of Multi-core

Cores per Socket


100
10
1

https://cdn.arstechnica.net/wp-content/uploads/sites/3/2017/05/voltablockdiagram.png
Today’s Hybrid Multi/Many Core/Socket Architecture

- **Nothing** is uniform about memory references
- Multiple memory domains
- Multiple memory ports & types
- Multiple different link protocols
- Higher bandwidth parts needed (at energy costs)
- Growing “width” of data returned from an access (spatial locality)
## Energy Tightly Tied to Locality

<table>
<thead>
<tr>
<th>Operation</th>
<th>Energy (pJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64-bit integer operation</td>
<td>1</td>
</tr>
<tr>
<td>64-bit floating-point operation</td>
<td>20</td>
</tr>
<tr>
<td>256 bit on-die SRAM access</td>
<td>50</td>
</tr>
<tr>
<td>256 bit bus transfer (short)</td>
<td>26</td>
</tr>
<tr>
<td>256 bit bus transfer (1/2 die)</td>
<td>256</td>
</tr>
<tr>
<td>Off-die link (efficient)</td>
<td>500</td>
</tr>
<tr>
<td>256 bit bus transfer (across die)</td>
<td>1,000</td>
</tr>
<tr>
<td>DRAM read/write (512 bits)</td>
<td>16,000</td>
</tr>
<tr>
<td>HDD read/write</td>
<td>$\mathcal{O}(10^6)$</td>
</tr>
</tbody>
</table>

Perhaps 5 pJ in best of today

- Increasing with Non-Locality
- Largely unchanged by new technologies

Exascale goal of 20 pJ per flop unreachable if any memory references need to be made
Need for More Memory Bandwidth – Multi-level Memories

HBMs: 4-5X bandwidth, but wider transfer/access
### And Apps Are Changing – Lets look at some Benchmarks

<table>
<thead>
<tr>
<th>Benchmark Name</th>
<th>Function Performed</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LINPACK</strong></td>
<td>Solve $Ax=b$; $A$ is dense</td>
</tr>
<tr>
<td><strong>HPCG</strong>: Hi Perf Cong. Grad.</td>
<td>$Ax=b$; $A$ sparse but regular</td>
</tr>
<tr>
<td><strong>SpMV</strong>: Sparse Mat. Vec.</td>
<td>$Ab$; $A$ sparse &amp; irregular</td>
</tr>
<tr>
<td><strong>BFS</strong>: Breadth First Search</td>
<td>Find all reachable vertices from root</td>
</tr>
<tr>
<td><strong>FireHose</strong></td>
<td>Find “events” in streams of data</td>
</tr>
</tbody>
</table>
Performance vs Time

Peak Flops

Rapid increase driven by Moore's Law: Cache size & # FPUs

Flatness implies bound by something else

Top 500 Rmax

HPCG

Graph 500
Performance/Byte of B/W vs Time

- **Peak Flops**
  - Increase due to bigger caches in cores do more flops for same memory reads.

- **Top 500 Rmax**
  - Flatness of top systems implies memory bandwidth bound.

**Graph 500**

**HPCG**
Bigger systems have newer cores with bigger caches to do more flops for same memory reads

HPCG is Memory Bandwidth Bound

Decline vs size due to loss of injection bandwidth in bigger systems
Performance per Watt vs Time

**Peak Flops**
- Rapid Increase because ratio dependent on logic technology alone; more flops/s per socket

**Top 500 Rmax**
- Flat probably because near perfect weak scaling and no real memory improvement

**HPCG**

**Graph 500**
### Perf. per Watt vs Perf.

#### Peak Flops

Regular neighbor traffic places little demands on networking as system scales.

#### Top 500 Rmax

Traffic again regular; near perfect weak scaling.

#### HPCG

Random irregular traffic reduces node injection B/W as system scales.

#### Graph 500
Green-GRAPH500

The graph shows the GTEPS/Watt (Giga-Operations Per Second per Watt) on the y-axis and the number of Edges (G) on the x-axis. The data points are color-coded, with GPUs where the problem fits in GPU Memory highlighted.

The equation $(1/6)E^{-1/3}$ is also plotted, indicating a trend in the data.
Sparsity & Conventional Scalability

Observation: Extreme Sensitivity to
- Level of Sparsity
- # of physically separate memory domains

Across all benchmarks, it takes 10-1000 nodes of distributed memory systems to equal best of single domain systems for the sparsest problems

Firehose Streaming Benchmark

- **http://firehose.sandia.gov/**
- **Datum**: Comma separated ASCII string
  - **Key**: ASCII string representing 64b uint (IP adr)
  - **Value**: depends on benchmark variant
  - **Truth flag**: was the stream from this key biased
- **Event**: detection of 24 datums with same “key”
- **Anomaly**: value distribution biased towards 0s
  - 3 variants defined
- **Performance metric**: Datums/sec
Large Scale Anomaly 1 Processing

- MPI with PHISH runtime library
- Approx. 2.75M datums/s per node
- Or about 220 M/s per rack

- SNL SkyBridge, Cray-CS300 1848 2-socket nodes at 16 cores/node
- From “Stateful Streaming in Dist. Memory Supercomputers,” Berry & Porter, CLSAC 2016

Scaling line is fairly linear

**BUT** at 2.75M datums/s per **32 core node**, 0.09M datums/s per core is 1/60 that of a single core
### Summary: Basic Benchmarks – Non-traditional Have Locality Issues

<table>
<thead>
<tr>
<th>Benchmark Name</th>
<th>Function Performed</th>
<th>Performance Limiters</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINPACK</td>
<td>Solve $Ax=b$; $A$ is dense</td>
<td>Cache size &amp; # FPUs</td>
</tr>
<tr>
<td>HPCG: Hi Perf Cong. Grad.</td>
<td>$Ax=b$; $A$ sparse but regular</td>
<td>Memory B/W</td>
</tr>
<tr>
<td>SpMV: Sparse Mat. Vec.</td>
<td>$Ab$; $A$ sparse &amp; irregular</td>
<td>Memory B/W; some Network</td>
</tr>
<tr>
<td>BFS: Breadth First Search</td>
<td>Find all reachable vertices from root</td>
<td>Network B/W; Remote atomics</td>
</tr>
<tr>
<td>FireHose</td>
<td>Find “events” in streams of data</td>
<td>Managing the streaming</td>
</tr>
</tbody>
</table>
Real World Challenge Data Intensive Problem
(From Lexis Nexis)

Auto Insurance Co: “Tell me about giving auto policy to Jane Doe” in < 0.1sec

- 2012: 40+ TB of Raw Data
- Periodically clean up & combine to 4-7 TB
- Weekly “Boil the Ocean” to precompute answers to all standard queries
  - Does X have financial difficulties?
  - Does X have legal problems?
  - Has X had significant driving problems?
  - Who has shared addresses with X?
  - Who has shared property ownership with X?

Look up answers to precomputed queries for “Jane Doe”, and combine

“Jane Doe has no indicators

But

she has shared multiple addresses with Joe Scofflaw

Who has the following negative indicators ….”
Traditional Approach: Runaway Intermediate Data

- **14.2B recs**
- **325 B/rec**
- **4.6TB**
- **800M distinct IDs**

**Project**

- **14.2B recs**
- **100+ B/rec**
- **1.5TB**

**Join on Address**

- **1.6T recs**
- **200+ B/rec**
- **300+TB**

- **Compute adr hash**
- **Compare lnames**
- **Init score to 3**
- **Project**

**Hash ID1,2 & Distribute**

- **1.6T recs**
- **30B/rec**
- **48+TB**
- **“J”**

- **Send between nodes via TCP/IP datagrams**

**Sort & Remove Duplicates**

- **1.5T recs**
- **30B/rec**
- **45TB**
- **“D”**

**Group by ID pairs & Sum scores, Lname_match**

- **1.2T recs**
- **16B/rec**
- **20TB**

**Select on Score & Lname_match**

- **12B recs**
- **16B/rec**
- **200GB**

- **400M distinct IDs**
Projecting Performance for LexisNexis’ Implementation

2012: 400 2-socket nodes (10 racks)

2013 study looked at “future” alternatives:

- Upgrades to conventional

- “Lightweight” systems
  - Lower power, lower performance cores
  - Study assumed Calxeda 4-core ARMs
  - but systems like HP Moonshot similar

- Sandia’s X-Caliber project
  - Heavyweight with HMC-like memories
  - Resembles Intel’s Knights Landing

- All processing on bottom of 3D stack
  - System = “sea” of stacks
Heavyweight Alternatives Using LN’s App Flow

Performance Options:
- Socket: 6C to 24C
- Memory B/W by 3X
- Disk to SSD or RAMDisk
- Network to Infiniband

No one option grows performance more than 45%
Unconventional Alternatives

Baseline: 1026s, 10 racks

Lightweight: 1184s, 2 racks

2-level memory: 1068s, 3 racks

3D Stack Only: 5s, 1 rack
Migrational Computing: An Alternative Architecture

• **Thread Migration**: move site of a thread’s execution

• Rationale: make memory reference *LOCAL*

• Today: either *invisible* (e.g. during I/O call) or *explicit* (as in Chapel)

• New idea: make migration automatic on remote memory access
Nodelet: New unit of parallelism

All memory in single global address space

Memory
Memory
Memory

Front End
Core(s)
Front End
Core(s)
Front End
Core(s)

And they are free to spawn independent children

Until they make a non-local reference
And then moved to correct nodelet

Threads execute here

Smart Memory Controllers that also do atomics
A Real Migrational System

Atomics run in Memory Front End (MFE)

Memory

Nodelet

(a) Nodelet

Multi-Threaded Cores

Node

Node

Node

Migration Engine

N/W
I/F

I/O
I/F

Stationary Core

Runs OS, Launches Jobs

Migrating Threads are major traffic on Network

Intra Supernode Switch

Up Links

(c) SuperNode

(b) Node

Node

Node
Near Term Scaling

Emu Chick
- 8 Nodes, 64 Memory Channels
- Copy room environment

Emu1 Memory Server
- 256 nodes, 2048 Memory Channels
- Server room environment

The node boards are the same!
Sparse Matrix-Dense Vector with Migrating Threads

```
struct Aelt {
    int col;
    Aelt *next_rowelt;
};

nextj = Ahdr[i];
sum=0;
while (nextj != 0) {
    sum += x[(*nextj).col];
    nextj = (*nextj).next_rowelt;
}
q[i] = sum;
```

- One Aelt for each non-zero in some row of A
  - Non-zero value
  - Column index
  - Pointer to next non-zero

Thread carries i, sum, nextj with it

---

Migration before access

No migration before access
SpMV with Migrating Threads

Per row for migrating threads:
- Stinger-like multiple CSR blocks
- 32s+108 bytes
- At most s+1 migrations
Firehose with Migration

Packet Stream

Extract Datum → Parse Datum → Compute Hash → Probe Table → Hit 
Update Counts → Clear Entry → Prepare Report

Emu Nodelet Bounds:
• Memory access: 20.4 M/s
• Network bound: 18M/s
• Instr. Rate bound: 3.8M/s

Single Blade Multi-Core

Multi-Blade

Current Biggest Run: 400 Anomaly nodes 1.1 Billion keys/sec

- 1 Emu Chick (64 nodelets) = 88X a CS300 node
- 1 Emu Rack (2048 nodelets) = 35X a CS300 rack
Ultimate Scaling: Sea of Memory Stacks

• Add Cores below each vault
• Upgrade off-stack interfaces to full peer-peer protocol
• Add in second stack of non-volatile
• Result: standalone stack with 32 independent nodelets

http://www.micron.com/products/hybrid-memory-cube

Migrating Threads are the Glue!
Projection for Massive “Batch-Mode” LexisNexis Problem

Emu1 assumes 400MHz GCs
2400 MT/s DRAM Channels

Real-Time Streaming Version Even Better
Conclusions

- Non-locality increasing rapidly in real apps
- Current architectures becoming badly inefficient
- The problem is in the memory & scaling
- Growing need for “remote functions”
- Migrating threads greatly simplify all
- Natural projection to 3D systems
Acknowledgements

Benchmark Analysis in part by:

- NSF grant CCF-1642280

- US Dept. of Energy, Award# DE-NA0002377, as part of Center for Shock-Wave Processing of Advanced Reactive Materials (C-SWARM) at the Univ. of Notre Dame, Notre Dame, IN

Emu hardware design by Emu Solutions, Inc