Bridging the Gap between Memory Performance and Massive Parallelism:  
—The Critical Role of Programming Systems Innovations

Xipeng Shen  
North Carolina State University, USA
Unprecedented Scale

Key challenge: moving, processing, storing data

44x as much Data and Content Over Coming Decade

2020 35 zettabytes

Velocity
Variety
Volume

80% of world’s data is unstructured

2009 800,000 petabytes

Sources: IBM
Memory System: A Shared View

Xipeng Shen

Courtesy from Onur Mutlu.
Scaling Issue

Memory capacity per core expected to drop by 30% every two years

Core count doubling ~ every 2 years
DRAM DIMM capacity doubling ~ every 3 years

Source: Lim+ ISCA’2009
# Supercomputers [USA DOE SciDAC]

<table>
<thead>
<tr>
<th>Systems</th>
<th>2009</th>
<th>2011</th>
<th>202?</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Peak Flops/s</td>
<td>2 Peta</td>
<td>20 Peta</td>
<td>1 Exa</td>
</tr>
<tr>
<td>System Memory</td>
<td>0.3 PB</td>
<td>1 PB</td>
<td>10 PB</td>
</tr>
<tr>
<td>Node Performance</td>
<td>125 GF</td>
<td>200 GF</td>
<td>1-10 TF</td>
</tr>
<tr>
<td>Node Memory BW</td>
<td>25 GB/s</td>
<td>40 GB/s</td>
<td>200-400 GB/s</td>
</tr>
<tr>
<td>Node Concurrency</td>
<td>12</td>
<td>32</td>
<td>0(100)</td>
</tr>
<tr>
<td>Interconnect BW</td>
<td>1.5 GB/s</td>
<td>10 GB/s</td>
<td>50 GB/s</td>
</tr>
<tr>
<td>System Size (Nodes)</td>
<td>18,700</td>
<td>100,000</td>
<td>100 Million</td>
</tr>
<tr>
<td>Total Concurrency</td>
<td>225,000</td>
<td>3 Million</td>
<td>50 Million</td>
</tr>
<tr>
<td>Storage</td>
<td>15 PB</td>
<td>500,000</td>
<td>0(Million)</td>
</tr>
<tr>
<td>I/O</td>
<td>0.2 TB/s</td>
<td>2 TB/s</td>
<td>20 TB/s</td>
</tr>
<tr>
<td>MTTI</td>
<td>Days</td>
<td>Days</td>
<td>0(1Day)</td>
</tr>
<tr>
<td>Power</td>
<td>6 MW</td>
<td>~10 MW</td>
<td>~20 MW</td>
</tr>
</tbody>
</table>

- **10X**
- **50X**
- **5-10X**
- **50-100X**
- **70% power on data movement.**
- **2X**
Observation & Expectation

Memory systems are critical for modern computing.

But traditional designs are hard to catch up with the demands.
Architecture Trends

• Deep memory **non-uniformity**
  • Inside a machine
  • Across machines

• **Blurred** boundaries
  • Memory & Storage
  • Memory & Computing

Heterogeneous massively parallel processors
Trend 1: Deep Non-uniformity

NVIDIA DGX-1V
(960TeraFlops halfPrc)

8X V100 GPUs
40960 GPU cores
8X 16GB HBM2 Memory

2X Intel Xeon E5-2698 v4s
40 CPU cores
16X 32GB DDR4
4X 1.92TB SSD
Trend 1: Deep Non-uniformity

Inside a GPU

- Global memory
- Texture memory
- Shared memory
- Constant memory
- L1/L2 cache
- Read-only cache
- Texture cache

Better mem. usage gives over 4X speedups [Che+:SC’11, Chen+:Micro’14].

Keeps changing across models.
Trend II: Blurred Boundaries

Memory & Storage

Non-Volatile Memory

Memory & Computing

Processing In Memory (PIM)
Implications to Programmers

Something they prefer to ignore.

Programming Systems
(compiler, runtime)
Implications to Prog. Systems

- Leveraging non-uniformity
  - Inter-thread centric
  - Controllability
  - Portability
- Embracing blurred boundaries

Heterogeneous massive parallel processors

"The only source of knowledge is experience"
- Albert Einstein
Each thread: a little work, a few data accesses. But hundreds of thousands of them.

Patterns within a thread are less crucial than across threads.
Memory Coalescing

... = A[P[tid]];  

Degrade throughput by up to \((\text{warp size} - 1)\) times.  
\((\text{warp size} = 32 \text{ in modern GPUs})\)
Insights for Solution

4 trans/warp

Irregularity **in** a warp: problematic; **across** warps: okey!

1 trans/warp

**Solution:**
Turn intra-warp irreg. into reg. or inter-warp irreg.
Trans-1: Data Reordering

Original

\[ P[\ ] = \{0, 5, 2, 3, 2, 3, 7, 6\} \]

\[ ... = A[P[tid]]; \]

Transformed

\[ Q[\ ] = \{0, 1, 2, 3, 2, 3, 6, 7\} \]

\[ ... = A'[Q[tid]]; \]

Maintain mapping between threads & data values

\[ \text{tid: thread ID; } \bigtriangleup: \text{a thread; } \downarrow: \text{data access; } \uparrow: \text{data relocation} \]
Trans-2: Job Swapping

- Job = operations + data elements accessed

Both reduce 1 mem trans.
1 more than the optimal....
Insight 1: Take an inter-thread—centric view.

[ICS’10, ASPLOS’11, PPOPP’13]
Implications to Prog. Systems

- Leveraging non-uniformity
  - Inter-thread centric
  - Controllability
  - Portability
- Embracing blurred boundaries

Heterogeneous massively parallel processors
Example: Affinity-Based Scheduling

Streaming multiprocessor (SM)

Thread block

GPU
Massive Parallelism

Each thread block has a task to process.

A task = kernel (thread block ID).
Enable Controllability

Thread-centric task model \(\Rightarrow\) SM-centric task model

**Original Kernel:**
- `taskID = f(workerID);`
- `processTask(taskID);`

**New Kernel:**
- `smID = getSMID();`
- `taskID = JobQ[smID].next();`
- `if (taskID!=NULL)`
- `processTask(taskID);`

**Simplified:**

**Other complexities [ICS’15]:**
- Possibly unbalanced threads on SMs;
- Randomness in hardware scheduler;
- Atomic operations;
- Etc.

**i\(^{th}\) worker**

**i\(^{th}\) task**

**i\(^{th}\) task queue**
Affinity-based Scheduling
[ICS’2015]

- Speedup
- L1 misses

Speedup and L1 misses for different categories:
- Irreg: Speedup 1.28, L1 misses 0.61
- MD: Speedup 1.05, L1 misses 0.52
- CFD: Speedup 1.31, L1 misses 0.72
- NBF: Speedup 1.2, L1 misses 0.82
- Average: Speedup 1.21, L1 misses 0.68
Enabled Versatile Pipeline

Coarse-pipe

pipeline among SMs

Fine-pipe

pipeline among thread blocks
Enabled Versatile Pipeline

Speedups (X)

- FaceDet
- Reyes
- CFD
- Pyramid
- Raster
- LDPC
- Average
Insight II: Important to go around controllability limitations, which is feasible and beneficial.  
[ICS’15, PPOPP’17]
Implications to Prog. Systems

• Leveraging non-uniformity
  • Inter-thread—centric
• Controllability
• Portability

• Embracing blurred boundaries

Heterogeneous massively parallel processors
GPU Data Placement

- Global memory
- Texture memory
- Shared memory
- Constant memory
- L1/L2 cache
- Read-only cache
- Texture cache
GPU Data Placement

Global memory
Texture memory
Shared memory
Constant memory
(L1/L2 cache)
(Read-only cache)
(Texture cache)

Data in a program

3-4X performance difference
Data Placement Problem

Properties:

- Machine dependent
  Changes across models/generations
- Input dependent
  Changes across runs

Options:

- Manual efforts by programmers?
- Offline autotuning?
PORPLE: Portable Data Placement Engine
[Micro’2014]

Offer portability

OFFLINE

MSL
(mem. spec. lang.)

mem
spec

access
patterns

staged
program

ONLINE

PLACER
(placing engine)

desired
placement

online profile

EFFICIENT
EXECUTION

Create placement-agnostic code

architect/user

microkernels

org. program

Offer input adaptivity

Xipeng Shen xshen5@ncsu.edu
Insight III: Portable solutions are essential & feasible. [Micro’2014]

- K20c
- C1060
Implications to Prog. Systems

- Leveraging non-uniformity
  - Inter-thread—centric
  - Controllability
  - Portability
- Embracing blurred boundaries
  - Data reuse
  - Programmability
  - Security

Heterogeneous massively parallel processors
Data Reuse

- How to support efficient data reuse?
- E.g., position independence

(a) Addr. Space in Run-1

(b) Addr. Space in Run-2

Memory & Storage
Non-Volatile Memory
Slowdown by Pos-Indep Pointers

![Bar chart showing slowdown by data structures with "Swizzling" and "Fat pointer" categories.](chart)

- Data structures: list, btree, hashset, trie
- Category: Swizzling, Fat pointer

Xipeng Shen
Other Questions

- Programmability support?
- How to capitalize on the new features (persistency/PIM)?
- Data organization and management?
- Legacy code?
- Sharing and security?

Memory & Storage
Non-Volatile Memory
Final Takeaways

• Innovations in programming systems are essential for tapping into the potential of modern memory systems

• Key principles
  • Leveraging non-uniformity
  • Embracing blurred boundaries

• Many research opportunities

*Imagination is more important than knowledge. Knowledge is limited. Imagination encircles the world.*

-Albert Einstein