Tackling Memory and Concurrency Barriers for Modern Parallel Computing

Xipeng Shen

Computer Science, NCSU
Unprecedented Scale

Y2020
1000 petaflops

Y2021
20 petaflops

Exascale Roadmap

Delivering the Next 1000x Capability in a Decade

20 PF

100-300 PF System

Exascale system: 1000 PF

FY2009
FY2011
FY2015
FY2018

2020
35 zettabytes

44x

as much Data and Content Over Coming Decade

Velocity
Variety
Volume

80%
Of world’s data is unstructured

2009
800,000 petabytes

sources: SciDAC, IBM
Key Factors

- Concurrency
- Memory Performance

sources: SciDAC, IBM
<table>
<thead>
<tr>
<th>Systems</th>
<th>2009</th>
<th>2011</th>
<th>2015</th>
<th>2018</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Peak Flops/s</td>
<td>2 Peta</td>
<td>20 Peta</td>
<td>100-200 Peta</td>
<td>1 Exa</td>
</tr>
<tr>
<td>System Memory</td>
<td>0.3 PB</td>
<td>1 PB</td>
<td></td>
<td>10 PB</td>
</tr>
<tr>
<td>Node Performance</td>
<td>125 GF</td>
<td>200 GF</td>
<td>400 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>
<td></td>
</tr>
<tr>
<td>Node Concurrency</td>
<td>12</td>
<td>32</td>
<td>0(100)</td>
<td>0(1000)</td>
</tr>
<tr>
<td>Interconnect BW</td>
<td>1.5 GB/s</td>
<td>10 GB/s</td>
<td>25 GB/s</td>
<td>50 GB/s</td>
</tr>
<tr>
<td>System Size (Nodes)</td>
<td>18,700</td>
<td>100,000</td>
<td>500,000</td>
<td>0(Million)</td>
</tr>
<tr>
<td>Total Concurrency</td>
<td>225,000</td>
<td>3 Million</td>
<td>50 Million</td>
<td>0(Billion)</td>
</tr>
<tr>
<td>Storage</td>
<td>15 PB</td>
<td>30 PB</td>
<td>150 PB</td>
<td>300 PB</td>
</tr>
<tr>
<td>I/O</td>
<td>0.2 TB/s</td>
<td>2 TB/s</td>
<td>20 TB/s</td>
<td></td>
</tr>
<tr>
<td>MTTI</td>
<td>Days</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>~10 MW</td>
<td>~20 MW</td>
</tr>
</tbody>
</table>

**Expected Performance** [DOE SciDAC]

- **5-10X**
- **50-100X**
- **>300X**
Memory Gets More Complex

Tesla K20 (NVIDIA)
More than 8 types of memory

Non-volatile memory

3D stacked memory
Hardware Trends

- Rapidly growing node concurrency
- Increasingly complex memory systems

Pivotal Role of Program Optimizers

- Expose & exploit parallelism
- Capitalize on memory systems potential

Implications to COSMIC
Two Recent Advancements

• 1. Data Placement Optimizations (MICRO’14)
  - Memory Performance of GPU

• 2. Principled Speculative Parallelization (ASPLOS’14, ASPLOS’15)
  - Execution Concurrency of FSM
**Graphic Processing Unit (GPU)**

- Massive parallelism
- Favorable
  - computing power
  - cost effectiveness
  - energy efficiency
Complex Memory

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

- Global memory: coalescing; cache hierarchy
- Texture memory: 2D/3D locality; texture cache; read-only (except surface write)
- Shared memory: on-chip; bank conflicts
- Constant memory: broadcasting; cached; read-only
- L1/L2 cache: private/shared
- Read-only cache: read-only data
- Texture cache: 2D/3D locality; read-only
Data Placement Problem

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

Data in a program

3X 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

architect

org. program

MSL (mem. spec. lang.)

PORPLE-C (compiler)

PLACER (placing engine)

desired placement

EFFICIENT EXECUTION

input

Xipeng Shen  xshen5@ncsu.edu
Expose memory properties formally for automatic analysis. Separate hw. complexities.
MSL
(Memory Specification Language)

latency, hierarchy, size, sharing, constraints, serialization condition, ... ...

Keywords:
address1, address2, index1, index 2, banks, blockSize, warp, block, grid, sm, core, tpc, die, clk, ns, ms, sec, na, om, ?;
// na: not applicable; om: omitted; ?: unknown;
// om and ? can be used in all fields

Operators:
C-like arithmetic and relational operators, and a scope operator {};

Syntax:
• specList ::= processorSpec memSpec*
• processorSpec ::= die=Integer tpc; tpc=Integer sm; sm=Integer core; end-of-line
• memSpec ::= name id swmng rw dim size blockSize banks latency upperLevels
  lowerLevels shareScope concurrencyFactor serialCondition ; end-of-line
• name ::= String
• id ::= Integer
• swmng ::= Y | N // software manageable or not
• rw ::= R|W|RW // allow read or write accesses
• dim ::= na | Integer // special for arrays of a particular dimensionality
• sz ::= Integer[K|M|G][T][E] // for data elements
• size ::= sz | <sz sz> | <sz sz sz>
• blockSize ::= sz | <sz sz> | <sz sz sz>
• lat ::= Integer[clk|ns|ms|sec] // clk for clocks
• latency ::= lat | <lat lat>
• upperLevels ::= <id | name>*
• lowerLevels ::= <id>*
• shareScope ::= core | sm | tpc | die
• concurrencyFactor ::= < Number Number>
• serialCondition ::= scope{RelationalExpr}
• scope ::= warp | block | grid
### Serialization Condition

- Key to handle peculiar properties of some memory
- Insight: they are all about when accesses get serialized.

**Examples of serialization conditions:**

<table>
<thead>
<tr>
<th>Type</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>constant mem:</strong></td>
<td>warp{address1 != address2}</td>
</tr>
<tr>
<td><strong>shared mem:</strong></td>
<td>block{word1!=word2 &amp;&amp; word1%banks == word2%banks}</td>
</tr>
<tr>
<td><strong>global mem:</strong></td>
<td>warp{⌊address1/blockSize⌋ != ⌊address2/blockSize⌋}</td>
</tr>
</tbody>
</table>

- **broadcasting**
- **bank conflict**
- **coalescing**
Example of Mem Spec (Tesla M2075)

One-time effort by experts.
Encapsulate hw complexity (extensibility; portability).

Mem spec of Tesla M2075:

die = 1 tpc; tpc = 16 sm; sm = 32 core;
globalMem 8 Y rw na 5375M 128B ? 600clk <L2 L1> <> die <0.1 0.5> warp{\lfloor address1/blockSize \rfloor \neq \lfloor address2/blockSize \rfloor};
L1 9 N rw na 16K 128B ? 80clk <> <L2 globalMem> sm ? warp{\lfloor address1/blockSize \rfloor \neq \lfloor address2/blockSize \rfloor};
L2 7 N rw na 768K 32B ? 390clk om om die ? warp{\lfloor address1/blockSize \rfloor \neq \lfloor address2/blockSize \rfloor};
constantMem 1 Y r na 64K ? ? 360clk <cL2 cL1> <> die ? warp{address1 != address2};
cL1 3 N r na 4K 64B ? 48clk <> <cL2 constantMem> sm ? warp{\lfloor address1/blockSize \rfloor \neq \lfloor address2/blockSize \rfloor};
cL2 2 N r na 32K 256B ? 140clk <cL1> <cL2 constantMem> die ? warp{\lfloor address1/blockSize \rfloor \neq \lfloor address2/blockSize \rfloor};
sharedMem 4 Y rw na 48K ? 32 48clk <> <> sm ? block{word1!=word2 && word1%banks==word2%banks};

...
PORPLE: Portable Data Placement Engine

PORPLE

MSL (mem. spec. lang.)

PORPLE-C (compiler)

PLACER (placing engine)

architect

microkernels

org. program

desired placement

EFFICIENT EXECUTION

input

Xipeng Shen xshen5@ncsu.edu
PORPLE-C Compiler
(source-to-source compiler)

Finding out data access patterns
Complemented by online profiler
Making code placement-agnostic
Code can work regardless of data placement

OFFLINE
Enable Placement-Agnostic

// host code
float * A;
cudaMalloc(A,...);
cudaMemcpy(A, hA, ...);

// device code
x = A[tid];

// host code
float * A;
cudaMalloc(A,...);
cudaMemcpy(A, hA, ...);
texture <float, ...> Atex;
cudaBindTexture(null, Atex, A);

// device code
x = tex1Dfetch(Atex, tid);

// global declaration
__constant__ float * A[sz];

// host code
cudaMemcpyToSymbol (A, hA, ...);

// device code
x = A[tid];

(a) from global mem.  (b) through read-only cache  (c) from constant mem.

// host code
float * A;
cudaMalloc(A,...);
cudaMemcpy(A, hA, ...); texture <float, ...> Atex;
cudaBindTexture(null, Atex, A);

// device code
x = tex1Dfetch(Atex, tid);

// host code
float * A;
cudaMalloc(A,...);
cudaMemcpy(A, hA, ...);

// device code
__shared__ float s[sz];
s[localTid] = A[tid];
__synchthreads();
x = s[localTid];

(d) from texture mem.  (e) from shared mem.
Enable Placement-Agnostic

Coarse-grained & fine-grained versioning: code size V.S. time overhead

// host code
if (memSpace[A0_id]==TXR && memSpace[A1_id]==SHR)
    kernel_1(...); // version 1
else
    kernel_others (...); // version 2

// code in kernel_others
if (memSpace[A1_id]==SHR)
    sA1[...] = _tempA0;
else
    A1[j] = _tempA0;
PORPLE: **Portable Data Placement Engine**

**PORPLE**
- MSL (mem. spec. lang.)
- PORPLE-C (compiler)
- PLACER (placing engine)

architect
microkernels
org. program

desired placement
EFFICIENT EXECUTION

input

---

Xipeng Shen xshen5@ncsu.edu
Placer

ONLINE

mem
spec
access
patterns
staged
program

PLACER
(placing engine)

online profile

desired
placement

EFFICIENT
EXECUTION

input

online
input

Placer
Lightweight On-Line Profiling

- Array Sizes
- Lightweight profiling on CPU for irreg. programs
  - Work of the first thread block
  - At most 10 iterations for a loop
Placer

• Search for the best placement
  • A placement: how all data are placed on men
• Key component: Performance model \( M \)

\[
\text{Mem perf.} = M (\text{a placement, access patterns, mem spec})
\]
Lightweight Performance Model

- Reuse Distance for cache miss rate estimation
- Consideration of interferences on shared cache
- Maximal transfer time among data paths

\[
\max_{p \in P} \left\{ \sum_{i \in A(p)} \sum_{j \in \text{memHier}(i)} N_{ij} \times T_j \times \alpha_j \right\}
\]

- all data paths
- arrays through path \( p \)
- number of accesses
- latency of memory \( j \)
- concurrency factor
PORPLE in a Whole

OFFLINE

architect/user

MSL
(mem. spec. lang.)

PORPLE-C
(compiler)

mem spec

access patterns

staged program

ONLINE

desired placement

EFFICIENT EXECUTION

online profile

input

More details in Micro’2014.

Xipeng Shen  xshen5@ncsu.edu
Properties of PORPLE

• Easy expansion to new memory
  • New MSL spec
• Good portability to new memory
  • Program runs with new suitable placement automatically
• Adaptivity to new program inputs
  • On-the-fly placement with placement-agnostic code.
• Generality to regular & irregular programs
  • Static analysis + lightweight online profiling
Evaluation

- 3 generations of NVIDIA GPU: K20c, M2075, C1060
  - Different
    - mem size, cache, latency, etc.
- 14 benchmarks
  - from SDK, SHOC, Rodinia
  - 9 regular, 5 irregular

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mm</td>
<td>SDK</td>
<td>dense matrix multiplication</td>
</tr>
<tr>
<td>convolution</td>
<td>SDK</td>
<td>signal filter</td>
</tr>
<tr>
<td>trans</td>
<td>SDK</td>
<td>matrix transpose</td>
</tr>
<tr>
<td>reduction</td>
<td>SHOC</td>
<td>reduction</td>
</tr>
<tr>
<td>fft</td>
<td>SHOC</td>
<td>fast Fourier transform</td>
</tr>
<tr>
<td>scan</td>
<td>SHOC</td>
<td>scan</td>
</tr>
<tr>
<td>sort</td>
<td>SHOC</td>
<td>radix sort</td>
</tr>
<tr>
<td>traid</td>
<td>SHOC</td>
<td>stream triad</td>
</tr>
<tr>
<td>kmeans</td>
<td>Rodinia</td>
<td>kmeans clustering</td>
</tr>
<tr>
<td>particlefilter</td>
<td>Rodinia</td>
<td>particle filter</td>
</tr>
<tr>
<td>cfd</td>
<td>Rodinia</td>
<td>computational fluid</td>
</tr>
<tr>
<td>md</td>
<td>SHOC</td>
<td>molecular dynamics</td>
</tr>
<tr>
<td>spmv</td>
<td>SHOC</td>
<td>sparse matrix vector multi.</td>
</tr>
<tr>
<td>bfs</td>
<td>SHOC</td>
<td>breadth-first search</td>
</tr>
</tbody>
</table>
Regular Benchmarks on K20c

Rule-based method from Jang+:TPDS.
Irregular Benchmarks on K20c
• K20c

• M2075

• C1060
Portability

• Cross architectures

<table>
<thead>
<tr>
<th></th>
<th>spmv</th>
<th></th>
<th>particlefilter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A0</td>
<td>A1</td>
<td>A2</td>
</tr>
<tr>
<td>Rule-Based</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>PORPLE-C1060</td>
<td>C</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>PORPLE-M2075</td>
<td>C</td>
<td>T</td>
<td>G</td>
</tr>
<tr>
<td>PORPLE-K20c</td>
<td>C</td>
<td>R</td>
<td>T</td>
</tr>
</tbody>
</table>

• Cross inputs

![Graph](spmv.png)  
![Graph](particlefilter.png)
Runtime Overhead Breakdown

- **particlefilter**
  - Transform: 0.001
  - Engine: 0.003
  - Profiling: 0.005

- **spmv**
  - Transform: 0.002
  - Engine: 0.003
  - Profiling: 0.005

- **cfd**
  - Transform: 0.001
  - Engine: 0.005
  - Profiling: 0.005

- **md**
  - Transform: 0.0001
  - Engine: 0.0005
  - Profiling: 0.0005

- **bfs**
  - Transform: 0.0001
  - Engine: 0.0005
  - Profiling: 0.0005
Two Recent Advancements

• 1. Data Placement Optimizations (MICRO’14)
  - Memory Performance of GPU

• 2. Principled Speculative Parallelization (ASPLOS’14, ASPLOS’15)
  - Execution Concurrency of FSM
Finite State Machine (FSM)

Example: Check whether a string contains “aabaaabb”.

Widely used:
- Web Browsers (HTML lexing/parsing)
- Decompression (Huffman decoding)
- Pattern Matching (grep)
- Security (intrusion detection)
- Software Engineer (model checking)
- Hardware design (VHDL state machine)
“Embarrassingly Sequential”

—The Landscape of Parallel Computing Research: A View from Berkeley

input string: a a b a b b a a b b a b a b a a b b a b a b a a a

thread 1

thread 2

starting from which state?
Speculative Parallelization

• Basic idea
  • Pick a guess to start processing
  • Reprocessing upon a wrong guess

input string: a a b a b a b b a a a b a b a b a a a b b a b a a a

thread 1

thread 2
starting from which state?
Key Open Question

How to make the guess to maximize performance?

the state of the art

(Prabhu+: PLDI’10)
Key Open Question

How to make the guess to maximize performance?

• How far should lookback go?

• Which state should be used to start lookback?

• Would starting from multiple states help? If so, which state should be chosen after lookback?

• How to combine lookback and partial commit effectively?

• How to characterize FSMs and their inputs so that the speculation can be FSM and input-aware?
Principled Speculation
—A rigorous way to design speculative parallelization

Benefits & Best Design

Probabilistic Performance Model

FSM Properties

Spec. Para. Scheme

Obtained through offline profiling
Performance

Fig. Speedup on 8-core Intel machine
Input Sensitivity

![Bar chart showing input sensitivity across different programs. The x-axis represents programs: lexing, huff, pval, commadot, likeapple, xval. The y-axis represents speedup with two categories: similar inputs and different inputs. The chart indicates varying levels of speedup for each program under different input conditions.]
Enabling Online Deployment

**Static FSM Analysis**
- FSM Convergence
- Minimal Convergent Length

**Dynamic Profiling Optimization**
- Intermediate Results Reuse
- Early Stop

*Offline (one-time)* + *Online*
Overhead & Performance

Table 3: Profiling Time

<table>
<thead>
<tr>
<th></th>
<th>default (s)</th>
<th>optimized (s)</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>lexic</td>
<td>0.50</td>
<td>0.016</td>
<td>31X</td>
</tr>
<tr>
<td>huff</td>
<td>1.03</td>
<td>0.033</td>
<td>31X</td>
</tr>
<tr>
<td>pval</td>
<td>4.54</td>
<td>0.023</td>
<td>195X</td>
</tr>
<tr>
<td>strl</td>
<td>1353.5</td>
<td>0.212</td>
<td>6381X</td>
</tr>
<tr>
<td>str2</td>
<td>16.99</td>
<td>0.069</td>
<td>247X</td>
</tr>
<tr>
<td>xval</td>
<td>148</td>
<td>2.272</td>
<td>65X</td>
</tr>
<tr>
<td>div</td>
<td>1.84</td>
<td>0.019</td>
<td>98X</td>
</tr>
</tbody>
</table>

Table 5: Speedups over sequential executions

<table>
<thead>
<tr>
<th></th>
<th>offline</th>
<th>online (naïve)</th>
<th>online (optimized)</th>
</tr>
</thead>
<tbody>
<tr>
<td>lexic</td>
<td>6.47X</td>
<td>4.46X</td>
<td>6.60X</td>
</tr>
<tr>
<td>huff</td>
<td>6.76X</td>
<td>4.05X</td>
<td>6.68X</td>
</tr>
<tr>
<td>pval</td>
<td>0.94X</td>
<td>1.53X</td>
<td>6.49X</td>
</tr>
<tr>
<td>strl</td>
<td>3.29X</td>
<td>0.01X</td>
<td>3.76X</td>
</tr>
<tr>
<td>str2</td>
<td>3.79X</td>
<td>0.39X</td>
<td>3.98X</td>
</tr>
<tr>
<td>xval</td>
<td>1.52X</td>
<td>0.12X</td>
<td>3.50X</td>
</tr>
<tr>
<td>div</td>
<td>1.26X</td>
<td>1.00X</td>
<td>1.26X</td>
</tr>
<tr>
<td>geomean</td>
<td>2.68X</td>
<td>0.49X</td>
<td>4.08X</td>
</tr>
</tbody>
</table>

Performance on 8-core Intel machine
Summary

Memory: Portable Data Placement Opt.

Concurrency: Principled Speculation

Systematic

Rigor

Ad Hoc

Design of Speculative Parallelization of FSM
To be explored...

Memory: Portable Data Placement Opt.
- Interplay w/ data layout
- Beyond GPU
- Emerging memory technology
  - 3D stacked mem
  - non-volatile mem
  - ...

Concurrency: Principled Speculation
- Energy efficiency
- Beyond FSM
  - Dyn. programming
  - Other sequential prog.
- Unified framework
  - ...

Final Takeaways

• Two key factors: Memory & Concurrency

• Software solution is critical & promising
  • PORPLE: portability, extensibility, input adaptivity
  • Principled speculation: rigor is powerful