Augmenting GPU Hardware through Software Innovations
---A Reflection on a Dozen Years of Efforts

Xipeng Shen

Computer Science, North Carolina State University
Evolution of NVIDIA GPU

Adapted from NVIDIA's Presentation
Tremendous Successes

COMPUTATIONAL CHEMISTRY

COMPUTATIONAL FLUID DYNAMICS

COMPUTATIONAL STRUCTURAL MECHANICS

ELECTRIC DESIGN AUTOMATION

IMAGING & COMPUTER VISION

MEDICAL IMAGING

AI FOR PUBLIC GOOD

NUMERICAL ANALYTICS

WEATHER AND CLIMATE

Credit: NVIDIA
Three “Dark Clouds” Persist

Irregular Mem & Control

Non-Data Parallelism

Preemptive Scheduling
Massive Parallelism

A_kernel_function (...) {
  ...
  ... [thread_ID] ...
  ...
}

Each thread: a little work, a few data accesses.
But hundreds of thousands of them.

Great for **Regular Data-Parallel Computing**.
A_kernel_function (...) {
  ... [thread_ID] ...
  ... ...
}

Massive Parallelism

Streaming multiprocessor (SM)

GPU

Thread block

- “Cloud” 1: SPMT model is not designed for other types of parallelism
  - E.g., pipeline parallelism, task parallelism
- “Cloud” 2: GPU remains sensitive to divergence in controls & accesses
  - Memory coalescing, thread divergence
- “Cloud” 3: Massive states to store and recover for preemption
  - Despite added hardware support
Our Explorations on GPU Performance

Compiler-based software solutions
Three Persistent “Dark Clouds”

Irregular Mem & Control

Non-Data Parallelism

Preemptive Scheduling

GPU
Pipeline Applications

High throughput & low latency both desired

Network packet processing

Image rendering

Face detection

Deep Neural Networks

image credit: Whippletree, cs.adelaide.edu.au
Neither builds an actual pipeline on GPU.
Is actual pipeline possible on GPU?

Requires both spatial and temporal controls of threads executions.

SM-centric Kernel + SW-based Scheduling
SM-Centric Kernel

[ICS’15]

**Thread-centric task model** ⇒ **SM-centric task model**

**original_kernel:**
- taskID = f(workerID);
- processTask(taskID);

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

\[ i^{th} \text{ worker} \]
\[ i^{th} \text{ task} \]

\[ i^{th} \text{ SM} \]
\[ i^{th} \text{ task queue} \]
New Exec. Models Enabled

**Run-to-Completion**

```plaintext
gpuKernel{
    stage1(oneJob);
    stage2(oneJob);
    stage3(oneJob);
}
```

**Kernel-by-Kernel**

```plaintext
if (accumulateEnoughJob){
    stage1_kernel(ManyJobs);
    stage2_kernel(ManyJobs);
    stage3_kernel(ManyJobs);
}
```
# Model Comparisons

<table>
<thead>
<tr>
<th>Occupancy</th>
<th>KbK</th>
<th>Run-to-completion</th>
<th>Megakernel</th>
<th>Coarse-pipe</th>
<th>Fine-pipe</th>
</tr>
</thead>
<tbody>
<tr>
<td>registers</td>
<td>√</td>
<td>X (U)</td>
<td>X (U)</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>sharedMem</td>
<td>√</td>
<td>X (U/Max)</td>
<td>O (Max)</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>code footprint</td>
<td>√</td>
<td>X</td>
<td>X</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>data locality</td>
<td>X</td>
<td>√</td>
<td>O</td>
<td>X</td>
<td>√</td>
</tr>
<tr>
<td>load balancing</td>
<td>√</td>
<td>√</td>
<td>√ (schedule)</td>
<td>X (rigid)</td>
<td>O</td>
</tr>
<tr>
<td>simplicity control</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>responsiveness</td>
<td>X</td>
<td>O</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
</tbody>
</table>
VersaPipe
https://github.com/JamesTheZ/VersaPipe
A Versatile Programming Framework for Pipelined Computing on GPU
[Micro17]
Speedups

Megakernel: Whippletree [TOG2014]
Pipeline Beyond GPU

[Monday 11:30am Data Movement II Session]

“HiWayLib: A Software Framework for Enabling High Performance Communications for Heterogeneous Pipeline Computations”

- Communication speed.
- Termination checks.
- Task queue contention.
Dynamic Parallelism

• The amount of parallelism is dynamic.
• Example: Breadth-First Search (BFS).

Many other applications:
Graph Coloring,
Survey Propagation,
Shortest Path,
Minimal Spanning Tree,
Connected Components Labeling,
…
Current Support on GPU

• Sub-kernel Launch (SKL)

Main (...) {
    ...
    A_GPU_Kernel <<< ... >>> ( ... );
}
A_GPU_kernel ( ... ){
    ...
    subkernel <<< ... >>> ( ... );
}
Large overhead of SKL

- Practices: Careful programming to use task lists etc.
  - Programming productivity lost
- HW extensions (e.g., [Kim+:Micro2014 [Wang+:ISCA2015])
  - Add HW complexities
Solution: Free Launch  
[Micro’15] 

- A compiler-based source-to-source transformation 

Get the best of both worlds: 
- Intuitive coding 
- Low overhead 
- Superb load balance 

37X Speedup

Program: 
SKL style

Remove SKL

Program: 
Worklist style

Redistribute subtasks to parent threads
Three Persistent “Dark Clouds”

Irregular Mem & Control

Non-Data Parallelism

Preemptive Scheduling

GPU
Dynamic Irregularities

Degrade throughput by up to (warp size) times.
(warp size = 32 in modern GPUs)

A problem exists on all GPUs.
Solution 1: Thread-Data Remapping

4 trans/warp

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

Principle of solution: Turn intra-warp irreg. into reg. or inter-warp irreg.
G-Streamline
[ASPLOS’2011, PPOPP’13]

First framework enabling runtime thread-data remapping.

CPU-GPU pipeline to hide transformation overhead.

Kernel splitting to resolve dependences.

1.08—2.5X speedups
Solution 2: Data Placement

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

Data in a program

3X performance difference
PORPLE: Portable Data Placement Engine

[Micro’2014]

Offer portability

architect/user
microkernels

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

org. program

PORPLE-C (compiler)
Observations

- Decent speedups on K20c, M2075, C1060.
  - from better use of texture mem, constant mem & cache.
- Still matter on new GPUs?

Variations of Data Placements in Memory

[PMBS’2018 by Bari et al.]
Problem Shifts beyond one GPU

NVIDIA DGX2

16x Tesla V100 32GB
12x NVSwitch

NVLink Plane Card
8x EDR IB/100 GigE
2x Xeon Platinum
1.5TB System Memory
PCIe Switch Complex
30TB NVME SSDs
Three Persistent “Dark Clouds”

Irregular Mem & Control

Non-Data Parallelism

Preemptive Scheduling

GPU
Before Pascal: No preemption support.

Application 1
- Kernel A (long running, low priority)

Application 2
- Kernel B (high priority)

Wait, wait, wait …

A major issue for responsiveness, priority, and fairness.
EffiSha
[PPOPP’2017]

First software-controlled preemptive scheduling of GPU kernels.
Execution Time

EffiSha runtime daemon

Sched. policies

launching requests/notifications

transformed GPU program-1

transformed GPU program-2

... double-arrows

transformed GPU program-m

eviction flag

CPU

kernel (re)launch

read

GPU

evictable kernel-1

evictable kernel-2
Still a Problem?

Since Pascal: Instruction-level preemption becomes supported.

• Context size: 7MB Shared Mem & 20MB Registers on V100.

• Memory capacity limitation / performance interference.

• Unsafe isolation (e.g., leftover on shared memory).
Two More Clouds

- Reliability
- Security
- Non-Data Parallelism
- Preemptive Scheduling
- Irregular Mem & Control

GPU
Conclusion

• GPU changes fast; some challenges persist
• SW support is essential
  • Focusing on principled barriers gives lasting value
• Call for continuous innovations
  • Versatile support of parallelism
  • Portable support of memory opt. (intra- & inter-dev)
  • Efficient support of safe sharing
  • Reliable support of critical uses