Register and Thread Structure Optimization for GPUs

Yun (Eric) Liang, Zheng Cui, Kyle Rupnow, Deming Chen

Peking University, China
Advanced Digital Science Center, Singapore
University of Illinois at Urbana Champaign, USA
Modern Computing Systems

- **GPUs**
  - Computation and bandwidth
  - Three of top-five high performance machines on the June 2011 Top 500 list
  - Top two machines on the Green 500 list of the most energy-efficient supercomputers
- **Heterogeneity (CPU, GPU, …) common**
- **Low-power embedded system**
  - Tegra 2/3/4
Modern Computing Systems

- CPUs ➞ i7 @ 263mm²
  - Control heavy
  - Complex core
  - Less space devoted to computation resources

- Good at ILP
Modern Computing Systems

- GPUs ➔ GF100 @ 529 mm²
  - Simple cores
  - Many cores

- Data parallelism
  - Good at TLP
  - Bad at control
GPU Performance Optimization

- Performance tuning is difficult
  - Many architecture and application parameters
  - Kernel development is a heavy lifting task

- Automatic analysis and performance optimization

- Register and thread structure optimization
  - Joint optimization problem
Register Allocation

- Large register file
  - GTX480: 49152 bytes; 32768 registers (32 bit) per SM
  - nvcc interface: -maxReg: maximum number of registers used per thread.
Thread Structure

- GridSize: number of thread blocks
- BlkSize: number of threads per thread block
- Total threads: gridSize x blkSize
- Thread structure
  - Workload of one thread
  - Number of active threads
  - Thread scheduling
Register and Thread Structure Optimization

- GPU kernel performance
  - Single thread performance
  - Number of active threads
- Thread Structure
- Register allocation per thread
Joint Design Space

- Large design space
- Counter-intuitive performance tradeoff
- Performance improvement potential
Joint Design Space – More Kernels
Challenge… and Opportunity

- Large design space
  - Consistent increase in the shared resource and register limit

- Need to estimate performance accurately
  - Measurement not feasible

- Big speedup opportunity
Performance Estimation

Single thread performance

- Latency of instructions
- Dependencies among instructions
- Control flow of the program
- Basic block execution frequency
Latency of Instruction

Assembly code: Cuobjdump

Micro-benchmarks approach


1. MOV R5, R4;
2. F2F R4, R4;
3. FFMA R52, R55, c[xxx], R5;
4. FMUL R53, R5, xxx;
5. MOV R55, xxx;
6. FSETP P0, xxx, R5, xxx;
7. F2F R56, R52;
8. FMUL R53, R5, R53;
9. FMUL R5, R52, xxx;
10. FMUL R56, R52, R5;
11. FMUL R58, R53, xxx;
Dependencies among Instructions

- Instruction dependency graph
  - RAW, WAR, WAW

- Basic block latency estimated as critical path
Control Flow Graph

- Analysis of cuobjdump code to gather CFG
- GPGPU-Sim to gather execution frequencies

Single Thread Performance Estimation

\[
\text{Cycle}(\text{thread}) = \sum_{b \in B} \text{cycle}[b] \times \text{freq}[b]
\]

- Instruction latency
- Dependencies among instructions
- Control flow graph
- Basic block execution frequencies
Kernel Performance Estimation

- Overall performance depends on
  - Single thread performance (Latency estimation)
  - Number of active threads (Occupancy)

- Register and occupancy
  - Reg ratio is a linear estimate of thread latency
  - Product of Reg ratio and occupancy

- Performance and occupancy
  - 2-tuple < T, C >
  - C = Cycle(thread)
  - T denotes the remaining space for active threads
Design Space Exploration

Different DSE algorithms with tradeoffs

- GPU kernel performance
- DSE runtime

Design space exploration approaches

- Exhaustive Search (ES) – Infeasible, but optimal
- RO Search (ROS)
- Performance and Occupancy Search (POS)
- POS with filtering (POSF)
**RO Search (ROS)**

- Use $\text{Reg ratio} \times \text{Occupancy}$ as performance metric
- $0 < \text{Reg ratio} \leq 1; 0 < \text{Occupancy} \leq 1$
- Find configurations with maximal RO value
- Break ties through empirical measurement
Performance and Occupancy Search (POS)

- **Design space parameters**
  - gridSize, blkSize, reg, PO metric (T,C)

- **Pareto-optimal problem**
  - Two candidates, A & B: if A is better in both T & C, it dominates B and B can be eliminated

- **Detailed Algorithm**
  - Step 1: build the pareto-optimal set of candidates using performance estimation
  - Step 2: compare candidates empirically to verify selection
PO Search – Example Pareto-Optimal Set

Find pareto-optimal points and compare
**POS with Filtering (POSF)**

- Prune candidates less likely to be the optimal

![Graph](image)

- **C**: Single thread performance (thousands cycles)
- **T**: Remaining space for active threads

**Pareto-Optimal curve**
**Solution Summary**

- **nvcc provides interface for register control**
  - Maxreg: maximal number of registers allocated per thread

- **Thread structure**
  - blkSize and gridSize are kernel call arguments

- **Algorithms**
  - ES, ROS, POS, POSF

- **Suitable for compiler integration and portable to any GPU architecture**
# Experiments

## GTX480

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>CUDA SDK</th>
<th>blkSize and gridSize</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blackscholes (BS)</td>
<td>CUDA SDK</td>
<td>blkSize/gridSize</td>
</tr>
<tr>
<td>MarchingCubes (MC)</td>
<td>CUDA SDK</td>
<td>blkSize/gridSize</td>
</tr>
<tr>
<td>Nbody (NB)</td>
<td>CUDA SDK</td>
<td>blkSize/gridSize</td>
</tr>
<tr>
<td>Particles (Par)</td>
<td>CUDA SDK</td>
<td>blkSize/gridSize</td>
</tr>
<tr>
<td>3D Audio (Aud)</td>
<td>Real-applications</td>
<td>blkSize</td>
</tr>
<tr>
<td>CFD Solver (CFD)</td>
<td>Rodinia</td>
<td>blkSize/gridSize</td>
</tr>
</tbody>
</table>
Design Space

- Register per thread
  - 16 – 63

- Threads per block
  - multiple of 32 as warp size is 32
  - 32 to 512

- POSF filter range
  - 0.3 – 0.5
Speedup on GTX480

Graph showing GPU kernel performance speedup over default setting for various applications.
Speedup on GTX480

GPU kernel performance speedup over default setting

- BS
- MC
- NB
- Par
- Aud
- CFD
- Average

- ES
- POS
- POSF
- ROS
## Design Space Exploration Runtime

<table>
<thead>
<tr>
<th>Benches</th>
<th>Runtime (sec)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ES</td>
<td>RO</td>
</tr>
<tr>
<td>BS</td>
<td>14472</td>
<td>55</td>
</tr>
<tr>
<td>MC</td>
<td>25746</td>
<td>95</td>
</tr>
<tr>
<td>NB</td>
<td>76490</td>
<td>225</td>
</tr>
<tr>
<td>Par</td>
<td>40560</td>
<td>183</td>
</tr>
<tr>
<td>Aud</td>
<td>17454</td>
<td>70</td>
</tr>
<tr>
<td>CFD</td>
<td>4364</td>
<td>21</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

GPU optimization of register & thread structure
- Acceleration opportunity, but design space very large
- Accurate performance estimation
- Efficient design space exploration

POS, POSF algorithm
- High improvement with small runtime overhead
- Kernel latency speedup 1.33X
- Design space exploration speedup 355X
Thank you !!!