PORPLE: An Extensible Optimizer for Portable Data Placement on GPU

Guoyang Chen  
Xipeng Shen  
North Carolina State University

Bo Wu  
Dong Li  
Colorado School of Mines  
Oak Ridge National Lab
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
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 → MSL (mem. spec. lang.)
or. program → PORPLE-C (compiler)

PORPLE

PLACER (placing engine)

desired placement → EFFICIENT EXECUTION

input →
MSL
(Memory Specification Language)

OFFLINE

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][E|B] // 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:

- **constant mem:**
  - warp\{address1 \(!=\) address2\}

- **shared mem:**
  - block\{word1\(!=\)word2 \&\& word1\%banks == word2\%banks\}

- **global mem:**
  - warp\{⌊address1/blockSize⌋ \(!=\) ⌊address2/blockSize⌋\}

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

One-time effort per model by experts.

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{\[address1/blockSize\] != \[address2/blockSize\]};
L1 9 N rw na 16K 128B ? 80clk <> <L2 globalMem> sm ? warp{\[address1/blockSize\] != \[address2/blockSize\]};
L2 7 N rw na 768K 32B ? 390clk om om die ? warp{\[address1/blockSize\] != \[address2/blockSize\]};

sharedMem 4 Y rw na 48K ? 32 48clk <> <> sm ? block{word1!=word2 && word1%banks == word2%banks};
PORPLE: Portable Data Placement Engine

Por(PLACEMENT E)GINE

MSL (mem. spec. lang.)

PORPLE-C (compiler)

PLACER (placing engine)

architect

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
Enable Placement-Agnostic

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

// device code
x = A[tid];
```

(a) from global mem.

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

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

(d) from texture mem.

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

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

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

// device code
x = A[tid];
```

(b) through read-only cache

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

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

// device code
x = __ldg(&A[tid]);
```

(c) from constant mem.

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

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

(e) from shared mem.
Enable Placement-Agnostic

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

```c
if (memSpace[A1_id] == SHR)
    sA1[...] = _tempA0; // use local index
else
    A1[j] = _tempA0;
```
PORPLE: Portable Data Placement Engine

- architect
- org. program

MSL (mem. spec. lang.)
PORPLE-C (compiler)

PLACER (placing engine)

desired placement

EFFICIENT EXECUTION

input
PLACER (placing engine)

mem spec

access patterns

staged program

ONLINE

desired placement

EFFICIENT EXECUTION

input

online profile

Xipeng Shen xshen5@ncsu.edu
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 $M$

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

- Reuse Distance for cache miss rate estimation
- Simple consideration of interferences on shared cache
- Weighted total latency model

\[ \frac{1}{\sum_{i \in \text{all arrays}} \sum_{j \in \text{memHier}(i)} N_{ij} \times T_j \times \alpha_j} \]

- number of accesses
- latency of memory j
- concurrency factor
PORPLE in a Whole

OFFLINE

architect/user

MSL (mem. spec. lang.)

mem spec

access patterns

staged program

PLACER (placing engine)

desired placement

ONLINE

EFFICIENT EXECUTION

input

More details in our Micro’2014 paper.
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
Irregular benchmarks on M2075
Irregular benchmarks on C1060
Portability

• Cross architectures

<table>
<thead>
<tr>
<th></th>
<th>spmv</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>particlefilter</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A0</td>
<td>A1</td>
<td>A2</td>
<td>A3</td>
<td>A4</td>
<td>B0</td>
<td>B1</td>
<td>B2</td>
<td>B3</td>
<td>B4</td>
</tr>
<tr>
<td>Rule-Based</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>G</td>
<td>G</td>
<td>S&amp;G</td>
<td>G</td>
<td>G</td>
<td>G</td>
</tr>
<tr>
<td>PORPLE-C1060</td>
<td>C</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>G</td>
<td>C</td>
<td>S&amp;G</td>
<td>G</td>
<td>G</td>
<td>G</td>
</tr>
<tr>
<td>PORPLE-M2075</td>
<td>C</td>
<td>T</td>
<td>G</td>
<td>T</td>
<td>G</td>
<td>C</td>
<td>S&amp;G</td>
<td>G</td>
<td>G</td>
<td>G</td>
</tr>
<tr>
<td>PORPLE-K20c</td>
<td>C</td>
<td>R</td>
<td>T</td>
<td>R</td>
<td>G</td>
<td>C</td>
<td>S&amp;R</td>
<td>G</td>
<td>T</td>
<td>G</td>
</tr>
</tbody>
</table>

• Cross inputs

- **spmv**

- **particlefilter**
Runtime Overhead Breakdown

The chart shows the overhead breakdown for different tasks. The tasks include particlefilter, spmv, cfd, md, and bfs. The chart indicates the proportion of overhead attributed to Transform, Engine, and Profiling for each task.
Final Takeaways

• PORPLE
  • Portable, extensible, input-adaptive, general

• Key insights
  • Separation of hw complexities (MSL)
  • Online placement is necessary & feasible