**Locality-Aware Mapping of Nested Parallel Patterns on GPUs**

HyounJoong Lee*, Kevin Brown*, Arvind Sujeeth*, Tiark Rompf†‡, Kunle Olukotun*

*Pervasive Parallelism Lab (http://ppl.stanford.edu), Stanford University
†Purdue University, ‡Oracle Labs

---

**Motivation**

High-Level Languages for GPUs
- Provide higher productivity and portable performance
- Using parallel patterns (e.g., map, reduce, groupby) is becoming popular
- Parallel patterns encode high-level information on parallelism and synchronization

**Challenge:** Parallel patterns are often nested, which are difficult to map on GPUs

---

**Compiler Flow: Analysis and Optimizations**

- **Application**
- **Mapping Constraints** (e.g., Dim(x) for coalescing)
- **IR Traversal & Generate Constraints**
- **Memory Optimization** (layout, shared mem)
- **A Set of Templates for Each Pattern**

**Mapping Parameters**
- **Dimension** (x, y, z, ..)
  - A logical dimension assigned to the index domain of a nest level
  - Compiler controls how indices in each dimension are mapped to hardware threads
- **Block Size** (N)
  - Number of threads assigned for a given dimension
- **Degree of Parallelism (DOP)**
  - The amount of parallel computations enabled by a mapping
  - Span(k): assign k computations to each thread on a given index domain (decreases DOP by a factor of k)
  - Span(all): assign all indices of a given index domain to the threads within a single block
- **Example:** 2D index domain of size (N,M)
  - Split(3) on Dim x and Split(2) on Dim y, with an additional combiner kernel

**Mapping Constraints**
- Generated while traversing the IR to prune the mapping space
- Weights are associated with each constraint

**Selected Mapping**

**Search for an Efficient Mapping**

**Score Calculation**

**Performance vs Score**

- A: best performance region, B: warp-based mapping, C: false negatives

---

**Real World Applications**

- Evaluate the compiler on Rodinia Benchmark Suite
- Compare to 2D Strategies
  - Applications are written in different ways (row/col major)
  - Our compiler is not sensitive to how the application is written

---

**Parallel Patterns on GPUs**

- **Pattern (J)**
  - Dim(y), Size(16), Span(1)
- **Pattern (I)**
  - Dim(x), Size(32), Span(all)

**Equivalent mapping parameters for warp-based mapping**

---

**Dynamic Memory Allocation Optimization**

- Inner patterns may require dynamic allocations
- Allocate a temporary space for the entire threads at once
- Assign a proper offset / stride values for memory coalescing (depends on the mapping decision from the analysis)

---

**Evaluation (Nvidia K20c)**

- **Rodinia Benchmark Suite**
  - Applications written in different ways (row/col major)
  - Our compiler not sensitive to how the application is written

- **Comparison to 2D Strategies**
  - Applications written in different ways (row/col major)
  - Our compiler not sensitive to how the application is written

---

**Our Contributions**

- Define mapping parameters that are general enough to cover previous mapping strategies
- Present an analysis to automatically find an efficient mapping for nested parallel patterns, maximizing locality and resource utilization
- Present compiler optimizations that interact with the mapping analysis to further improve performance, avoiding dynamic allocations and using shared memory
- Implemented a compiler and show with a set of applications that our analysis and optimizations automatically generate efficient GPU code

---

**Motivation**

High-Level Languages for GPUs
- Provide higher productivity and portable performance
- Using parallel patterns (e.g., map, reduce, groupby) is becoming popular
- Parallel patterns encode high-level information on parallelism and synchronization

**Challenge:** Parallel patterns are often nested, which are difficult to map on GPUs

---

**Compiler Flow: Analysis and Optimizations**

- **Application**
- **Mapping Constraints** (e.g., Dim(x) for coalescing)
- **IR Traversal & Generate Constraints**
- **Memory Optimization** (layout, shared mem)
- **A Set of Templates for Each Pattern**

**Mapping Parameters**
- **Dimension** (x, y, z, ..)
  - A logical dimension assigned to the index domain of a nest level
  - Compiler controls how indices in each dimension are mapped to hardware threads
- **Block Size** (N)
  - Number of threads assigned for a given dimension
- **Degree of Parallelism (DOP)**
  - The amount of parallel computations enabled by a mapping
  - Span(k): assign k computations to each thread on a given index domain (decreases DOP by a factor of k)
  - Span(all): assign all indices of a given index domain to the threads within a single block
- **Example:** 2D index domain of size (N,M)
  - Split(3) on Dim x and Split(2) on Dim y, with an additional combiner kernel

**Mapping Constraints**
- Generated while traversing the IR to prune the mapping space
- Weights are associated with each constraint

**Selected Mapping**

**Search for an Efficient Mapping**

**Score Calculation**

**Performance vs Score**

- A: best performance region, B: warp-based mapping, C: false negatives

---

**Real World Applications**

- Evaluate the compiler on Rodinia Benchmark Suite
- Compare to 2D Strategies
  - Applications written in different ways (row/col major)
  - Our compiler not sensitive to how the application is written

---

**Our Contributions**

- Define mapping parameters that are general enough to cover previous mapping strategies
- Present an analysis to automatically find an efficient mapping for nested parallel patterns, maximizing locality and resource utilization
- Present compiler optimizations that interact with the mapping analysis to further improve performance, avoiding dynamic allocations and using shared memory
- Implemented a compiler and show with a set of applications that our analysis and optimizations automatically generate efficient GPU code