Exploring the Design Space of a Parallel Object Recognition System

Bor-Yiing Su, subrian@eecs.berkeley.edu
Tasneem G. Brutch, t.brutch@samsung.com
Kurt Keutzer, keutzer@eecs.berkeley.edu

Parallel Computing Lab,
University of California, Berkeley
Category of This Work

Diagram showing various categories and subcategories:

- **Applications**
  - Personal Health
  - Speech
  - Parallel Browser

- **Design Patterns/Motifs**
  - Image Retrieval
  - Hearing, Music

- **Composition & Coordination Language (C&CL)**
  - C&CL Compiler/Interpreter
  - Parallel Libraries
  - Parallel Frameworks

- **Efficiency Language Compilers**
  - Efficiency Languages
  - Sketching
  - Autotuners
  - Legacy Code
  - Schedulers
  - Communication & Synch. Primitives

- **Legacy OS**
  - OS Libraries & Services
  - Hypervisor

- **Multicore/GPGPU**
  - RAMP Manycore

- **Correctness**
  - Static Verification
  - Type Systems
  - Directed Testing
  - Dynamic Checking
  - Debugging with Replay
What’s New?

- Exploring more design space to further optimize key kernels in the object recognition system
  - Resulting in performance boosts:
    - Training: from 77.8x to 115x
    - Classification: from 72.5x to 119x
- Propose plans of developing frameworks for automating the procedure of design space exploration on object recognition key kernels

Outline

- Design Space
- An Object Recognition System
- Exploring the Design Space of the Object Recognition system
- Future Work
The design space of parallel applications is composed of three layers:

<table>
<thead>
<tr>
<th>Design Space</th>
<th>Explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm Layer</td>
<td>Using <strong>different ways</strong> to transform same inputs into same or similar outputs</td>
<td>Lanczos Solver</td>
</tr>
<tr>
<td>Parallelization Strategy Layer</td>
<td>Using <strong>different strategies</strong> to parallelize the same algorithm</td>
<td>BFS Graph Traversal</td>
</tr>
<tr>
<td>Platform Layer</td>
<td>Using <strong>specific hardware features</strong> to optimize the same parallelization strategy</td>
<td>Blocking Dimensions</td>
</tr>
</tbody>
</table>
1. **Exploring the design space** is necessary to achieve high performance on a hardware platform of choice.

2. **Take advantage of domain knowledge** is necessary to understand trade-offs among different parallelization methods and achieve peak performance.

![Diagram]

- Algorithm Layer
- Parallelization Strategy Layer
- Platform Layer

Design Space
Outline

- Design Space
- An Object Recognition System
- Exploring the Design Space of the Object Recognition system
- Future Work
Object Recognition System

Trained Categories

- Bottles
- Giraffes
- Mugs
- Swans

Image Queries

Object Recognition System

Outputs
Outline

- Design Space
- An Object Recognition System
- Exploring the Design Space of the Object Recognition system
  - Breadth First Search (BFS) Graph Traversal Kernel
  - Histogram Kernel
  - Pair-wise Distance Kernel
  - Overall Performance
  - Demo
- Future Work
The image segmentation component heavily relies on the BFS graph traversal kernel.

Image Graph:
- Nodes represent image pixels
- Edges represent neighboring relationships

BFS graph traversal kernel: propagate information from some pixels to other pixels.
Exploring the Algorithm Layer

- Direct algorithm: Each source node propagates information to its neighbors
  - Traditional BFS graph traversal algorithm

- Reverse algorithm: Each node checks whether it can be updated by one or more neighboring nodes
  - Structured grid computation
Two strategies can be used to parallelize the traditional BFS graph traversal algorithm

- Graph partition
- Parallel task queue
Associating Design Space Exploration with Input Data Properties

- Explored Design Space
  - Parallel Task queue on Intel Core i7 using OpenMP with 8 threads
  - Graph partition on Intel Core i7 using OpenMP with 8 threads
  - Structured grid on Nvidia GTX 480

Conclusion:
- Use the **structured grid method on a GPU** in our system
- Each image region is represented by its contour features
- The contour feature of a region is represented by a 128-bin histogram
Exploring the Algorithm Layer

- **Data to bins algorithm:**
  - Each data point atomically accumulate itself into the corresponding histogram bin

  ```
  foreach pixel \( p \)
  accumulate \( p \) into bin \( b \)
  ```

- **Bins to data algorithm:**
  - Each bin process its responsible data points

  ```
  foreach histogram bin \( b \)
  process pixels \( p_1 \ldots p_n \)
  ```
Exploring the Parallelization Strategy Layer

- Process each region in parallel

- When dealing with one region, two strategies can be used to parallelize the bins to data algorithm
  - Geometric decomposition: Process each histogram bin in parallel
  - Parallel reduction: For a histogram bin, accumulate its corresponding data points by parallel reduction
Explored Design Space
- Process each region in parallel on Intel Core i7 using OpenMP with 8 threads
- Geometric decomposition on Nvidia GTX 480
- Atomic accumulation algorithm on Nvidia GTX 480
- Parallel reduction on Nvidia GTX 480

Conclusion:
- Use the parallel reduction method on a GPU in our system
In both the training stage and the classification stage, we need to compute the pair-wise distance between two region sets:

- Similar regions have shorter distance
- Different regions have longer distance

It is a matrix multiplication computation:

- Replacing dot product into $\chi^2$ distance

Definition of the $\chi^2$ distance:

$$\chi^2(x, y) = \frac{1}{2} \sum_i \frac{(x_i - y_i)^2}{x_i + y_i}$$
Exploring the Design Space

- **Algorithm Layer**
  - Inner $\chi^2$ distance

  Algorithm: Inner $\chi^2$
  1. for $i \leftarrow 1$ to $m$
  2. for $j \leftarrow 1$ to $n$
  3. for $s \leftarrow 1$ to $k$
  4. $distance_{ij} \leftarrow distance_{ij} + \frac{(X_{is} - Y_{js})^2}{X_{is} + Y_{js}}$

- Outer $\chi^2$ distance

  Algorithm: Outer $\chi^2$
  1. for $s \leftarrow 1$ to $k$
  2. for $i \leftarrow 1$ to $m$
  3. for $j \leftarrow 1$ to $n$
  4. $distance_{ij} \leftarrow distance_{ij} + \frac{(X_{is} - Y_{js})^2}{X_{is} + Y_{js}}$

- **Platform Layer**
  - Cache Mechanisms
    - No Cache
    - Hardware Controlled Cache (Texture memory on GPU)
    - Software Controlled Cache (Shared memory on GPU)
Associating Design Space Exploration with Input Data Properties

- Explored Design Space
  - The combination of two algorithms and three cache mechanisms on Nvidia GTX 480

Conclusion:
- Use the outer $\chi^2$ distance method with software controlled cache in the training stage
- Use the inner $\chi^2$ distance method with software controlled cache in the classification stage
# Overall Performance: Speedups

<table>
<thead>
<tr>
<th>Computation</th>
<th>Computation time (s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Serial</td>
<td>Parallel</td>
</tr>
<tr>
<td>Feature</td>
<td>543</td>
<td>15.97</td>
</tr>
<tr>
<td>Distance</td>
<td>1732</td>
<td>2.9</td>
</tr>
<tr>
<td>Weight</td>
<td>57</td>
<td>1.41</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>2332</strong></td>
<td><strong>20.28</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Computation</th>
<th>Computation time (s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Serial</td>
<td>Parallel</td>
</tr>
<tr>
<td>Contour</td>
<td>236.7</td>
<td>1.58</td>
</tr>
<tr>
<td>Segmentation</td>
<td>2.27</td>
<td>0.357</td>
</tr>
<tr>
<td>Feature</td>
<td>7.97</td>
<td>0.065</td>
</tr>
<tr>
<td>Hough Voting</td>
<td>84.13</td>
<td>0.779</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>331.07</strong></td>
<td><strong>2.781</strong></td>
</tr>
</tbody>
</table>
Overall Performance: Detection Accuracy

Detection Accuracy with Bounding Boxes

Detection Rate vs. FPPI for Serial and Parallel setups.
Conclusion

- Exploring the design space is necessary to achieve high performance on a hardware platform of choice.
- Take advantage of domain knowledge is necessary to understand trade-offs among different parallelization methods and achieve peak performance.
- We have developed a parallel object recognition system with comparable detection accuracy while achieving 110x-120x times speedup.

- Work presented at Workshop on Applications of Computer Vision 2011

Outline

- Design Space
- An Object Recognition System
- Exploring the Design Space of the Object Recognition system
- Future Work
  - Develop Frameworks for Object Recognition Key Computations
  - Integration with the Par Lab stack
Frameworks for Computer Vision

- Design space exploration is a very time consuming procedure
  - We need to develop frameworks to automate design space exploration

- What frameworks to develop?

**Object Recognition Patterns**

**Structures**
- Window Sliding
- Pyramid Image Scaling
- Region-Based Processing
- Image-Based Processing
- Pair-wise Vector Processing

**Computations**
- K-means
- Mean-Shift
- Agglomerative
- Vector Distance
- Histogram Accumulation
- Convolution
- Pixel-wise Graph Traversal
- Hough Transform
- Eigen Decomposition
- Quadratic Programming
Framework of Pair-wise Distance

User Customization
- Defining the distance between two vectors

Parallelize Computation
- Exploring the design space automatically

Optimize Computation
- Generating the customized library

\[ \chi^2(x, y) = \frac{1}{2} \sum_i \frac{(x_i - y_i)^2}{x_i + y_i} \]

- Inner \( \chi^2 \)
- Outer \( \chi^2 \)
- Transpose Matrix
- Blocking Dimension
- Cache Mechanisms
Integration with the Par Lab Stack

- Use the Par Lab stack to develop frameworks for object recognition
  - Use SEJITS from the productivity layer to efficiently express different parallelization strategies
  - Use autotuner from the efficiency layer to explore the design space of the platform layer

Diagram:

- **Algorithm Layer**
  - **Parallelization Strategy Layer**
  - **Platform Layer**
- **Par Lab Stack**
  - **SEJITS (Productivity Layer)**
  - **Autotuner (Efficiency Layer)**
Questions
Backup Slides
Relationship with Our Pattern Language (OPL)

Design Space

Our Pattern Language

Applications

Structural Patterns

Parallel Algorithm Strategy Patterns

Implementation Patterns

Execution Patterns

Hardware Features

Algorithm Layer

Parallelization Strategy Layer

Platform Layer