Heterogeneous Parallel Programming COMP4901D

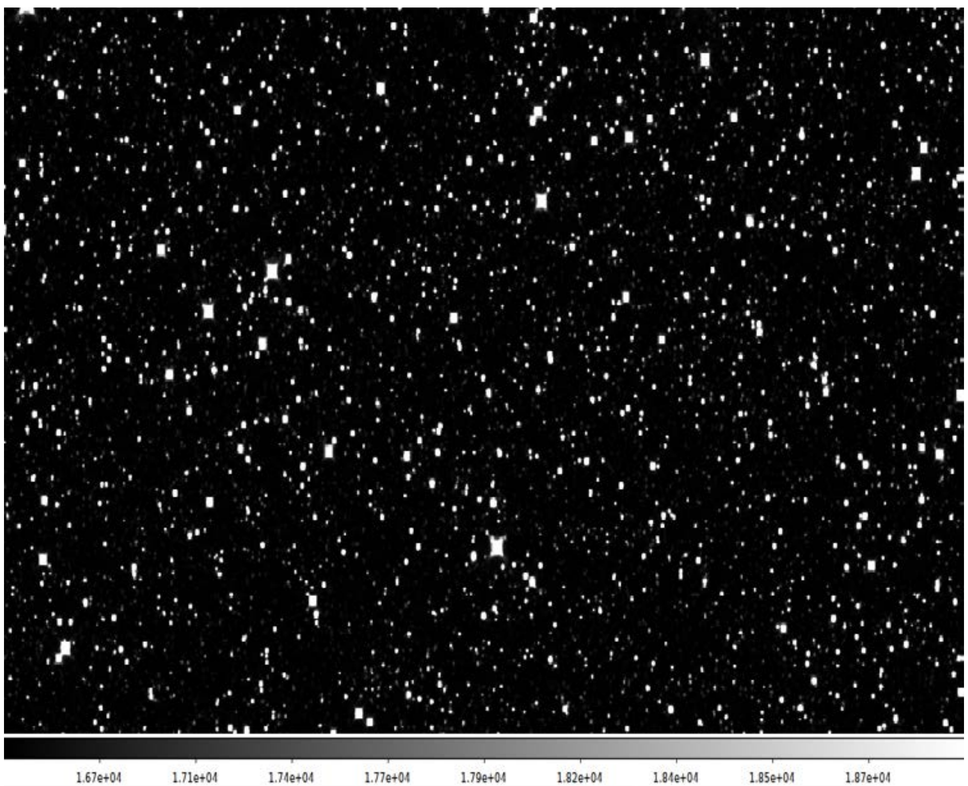
GPU-Based Source Extraction
from Astronomical Images

Overview

- Astronomical Image Format
- Source Extraction from Astronomical Images
- GPU-based Source Extraction from Astronomical Images

Astronomical Images

- Produced by telescopes every few seconds



FITS format

Header part contains meta-data

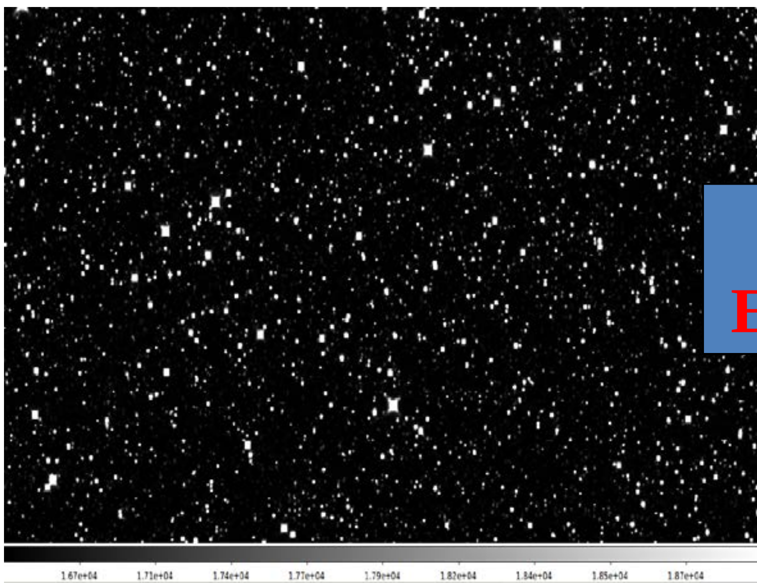
Each pixel only contains light intensity, no color.

Astronomical Catalogs

Each record in catalogs represents the information of astronomical objects, such as position, brightness, shape, and so on.

ID	X_POS	Y_POS	FLUX	THETA	ELONGATION
1	224.236	1.390	8024.492	2.6	0.557
2	866.526	1.486	4279.636	-0.8	0.499
3	928.903	1.469	7838.13	2.6	0.643
4	673.492	1.474	6777.753	0.2	0.512
5	429.787	1.881	4735.281	-48.9	0.144
…	….	….	….	….	….

Source Extraction

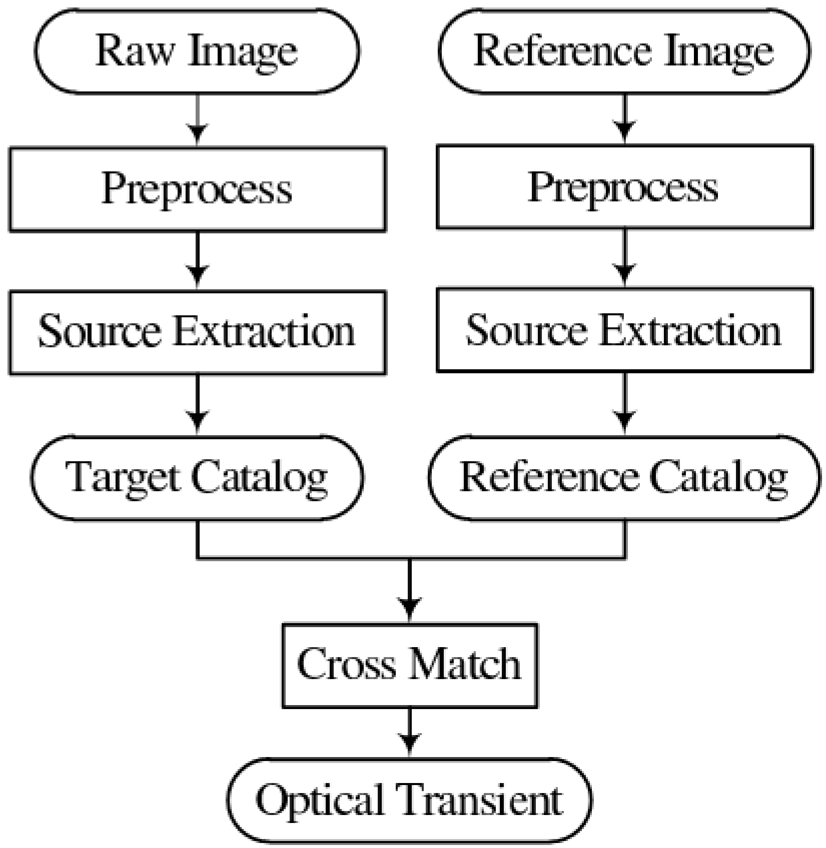


Source Extraction

ID	X_POS	Y_POS	…
1	224.236	1.390	…
2	866.526	1.486	…
3	928.903	1.469	…
4	673.492	1.474	…
5	429.787	1.881	…
…	….	….	….

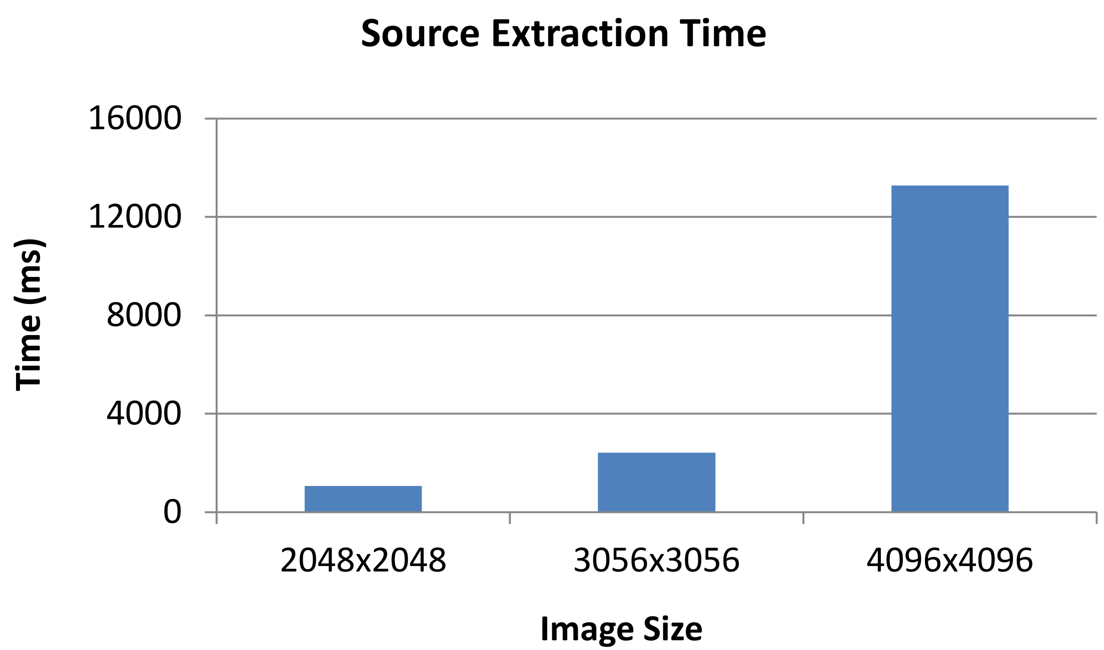
Source Extraction In Context

A typical scenario of using Source Extraction for searching Optical Transients(OT)



Performance Problem

- Raw images are generated every few seconds from telescopes, but source extraction takes much longer than that.



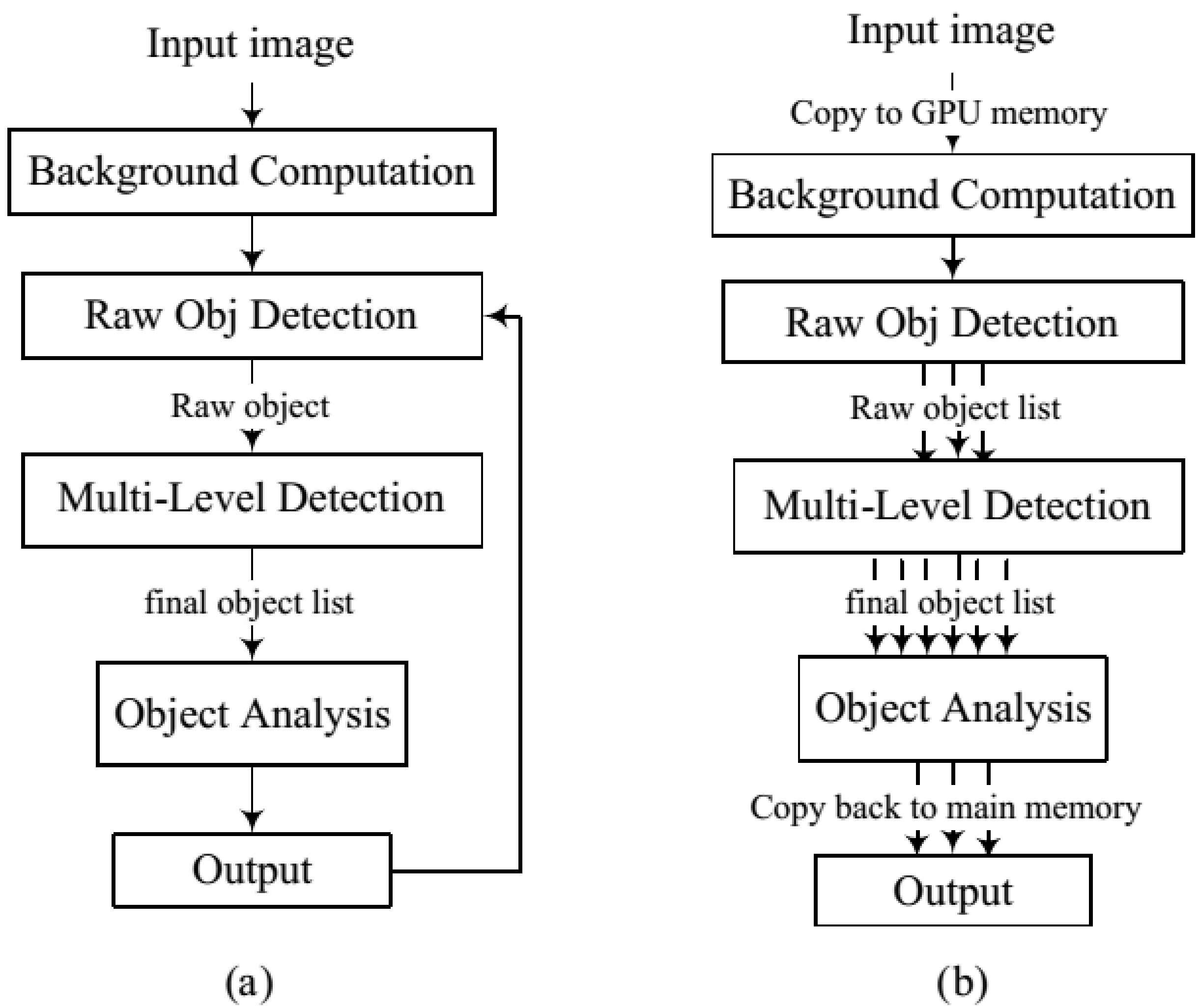
GPU-based Source Extraction

The entire extraction pipeline parallelized on the GPU

Efficient parallel multi-level detection algorithms

Simple yet efficient parallel analysis algorithms

Serial vs. Parallel Source Extraction

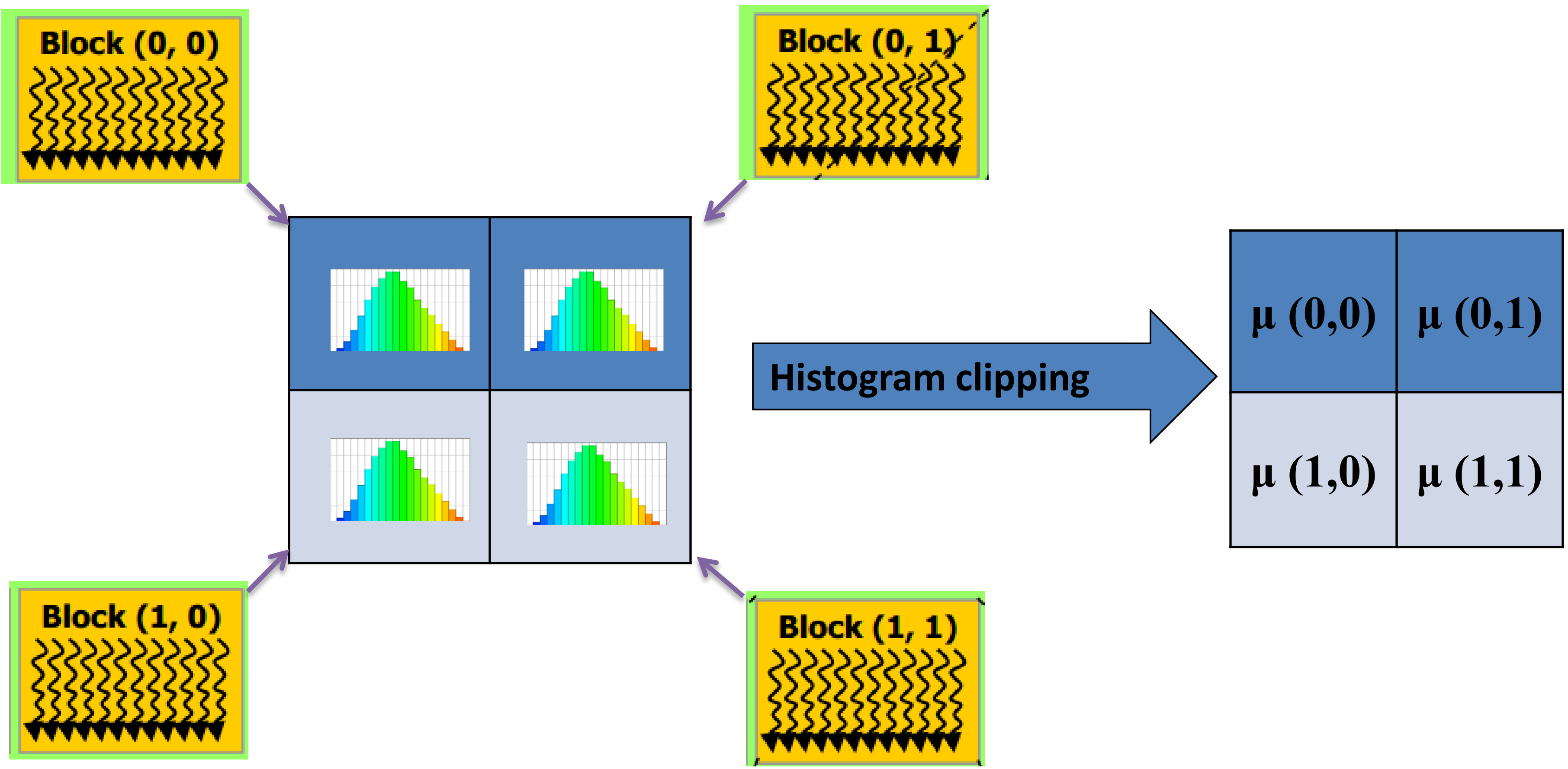


Source Extraction Time Breakdown

Time (millisecond)	**2048x2048**	**3056x3056**	**4096x4096**
Background Computation	92 (9%)	198 (8%)	406 (3%)
Multi-Level Detection	527 (49%)	1146 (47%)	7190 (54%)
Object Analysis	448 (42%)	1077 (45%)	5684 (43%)
Total	1067	2421	13280

Parallel Background Computation

Partition the image into a 2-D grid; Each thread block for one cell



Parallel Background Computation (Cont.)

- Smooth the local mean by a median filter
- Subtract the background value from each pixel

Parallel Object Detection

- Raw Object Detection
  - Detect connected component with basic threshold
  - Compute object properties (summarization)

- Multi-Level Detection
  - Compute multi-level thresholds for each raw object
  - Perform detection on each level
  - Resolve and prune object trees

Raw Object Detection: Label Initialization

6.9	8.3	7.4	4.9	4.3
4.1	5.3	4.8	3.2	7.8
7.5	9.8	8.1	3.0	9.2
4.3	4.6	4.7	3.3	8.5
3.9	5.7	6.4	2.2	4.9

Image (Pixel Array)

Basic Threshold 5.0

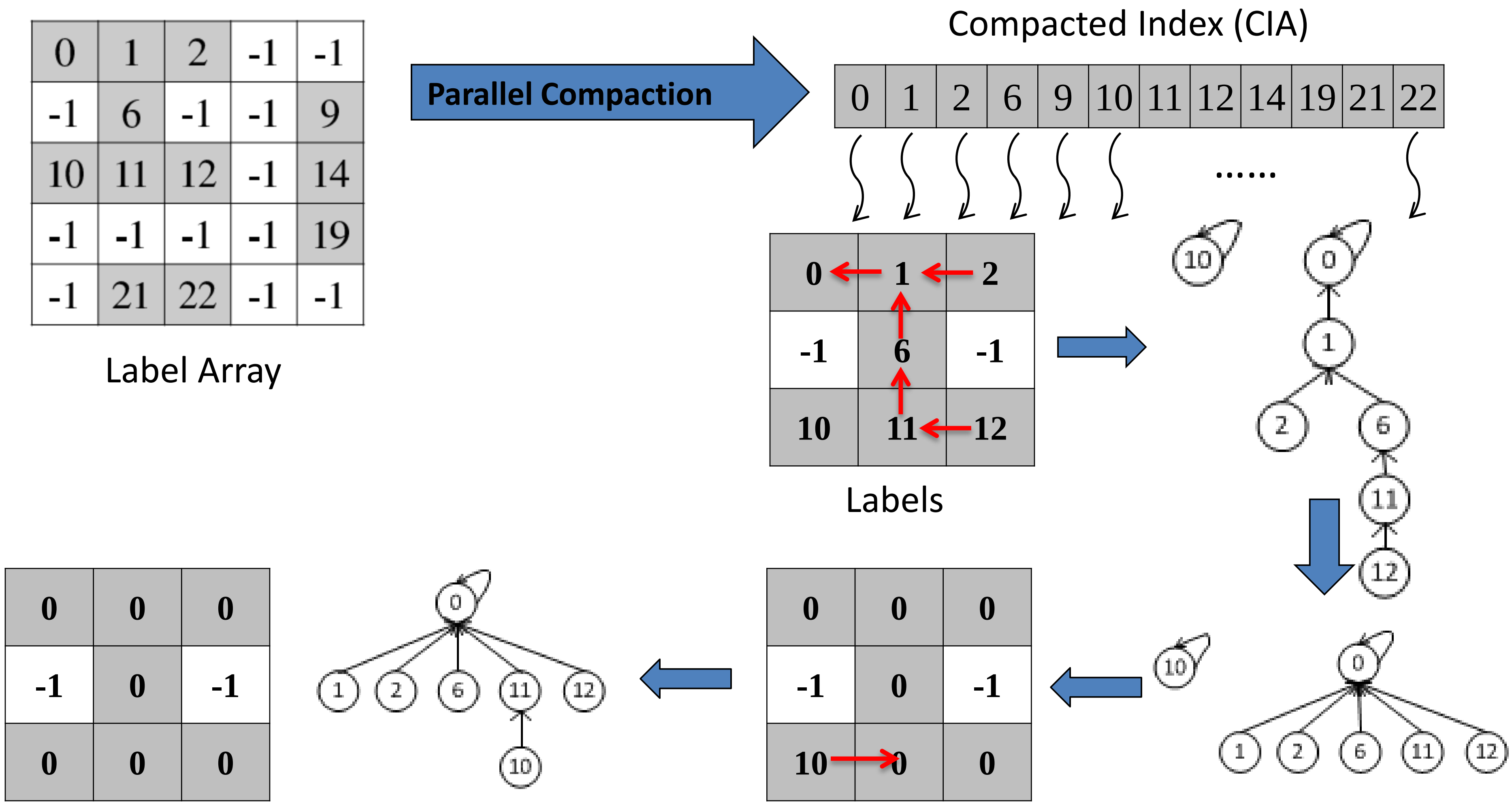
1	1	1	0	0
0	1	0	0	1
1	1	1	0	1
0	0	0	0	1
0	1	1	0	0

Binary Mask Array

0	1	2	-1	-1
-1	6	-1	-1	9
10	11	12	-1	14
-1	-1	-1	-1	19
-1	21	22	-1	-1

(Index) Label Array

Connected Component Labelling in Parallel



Updated Label Array

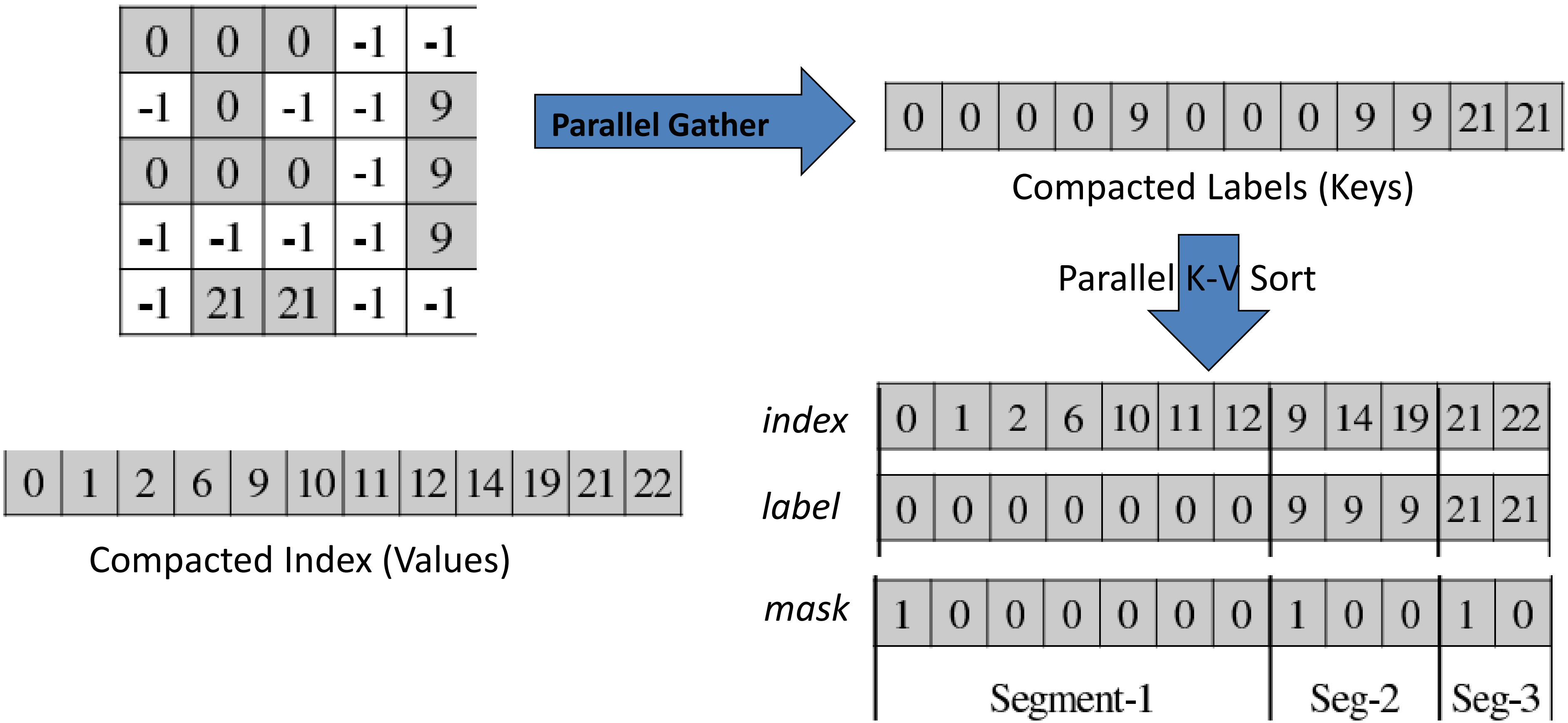
0	1	2	-1	-1
-1	6	-1	-1	9
10	11	12	-1	14
-1	-1	-1	-1	19
-1	21	22	-1	-1

Label Array

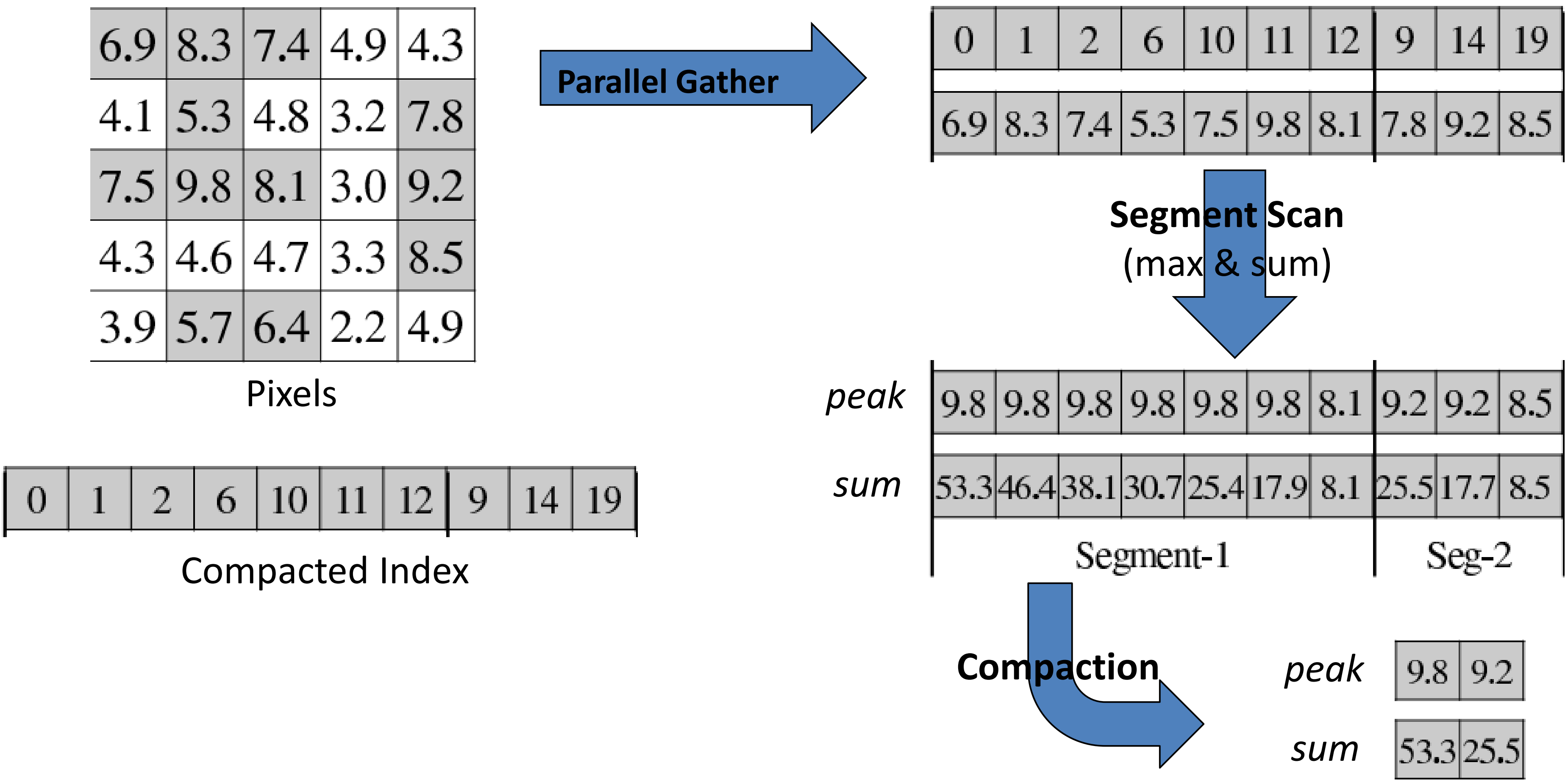
0	0	0	-1	-1
-1	0	-1	-1	9
0	0	0	-1	9
-1	-1	-1	-1	9
-1	21	21	-1	-1

Updated Label Array

Parallel Sorting of Labels



Parallel Object Summarization



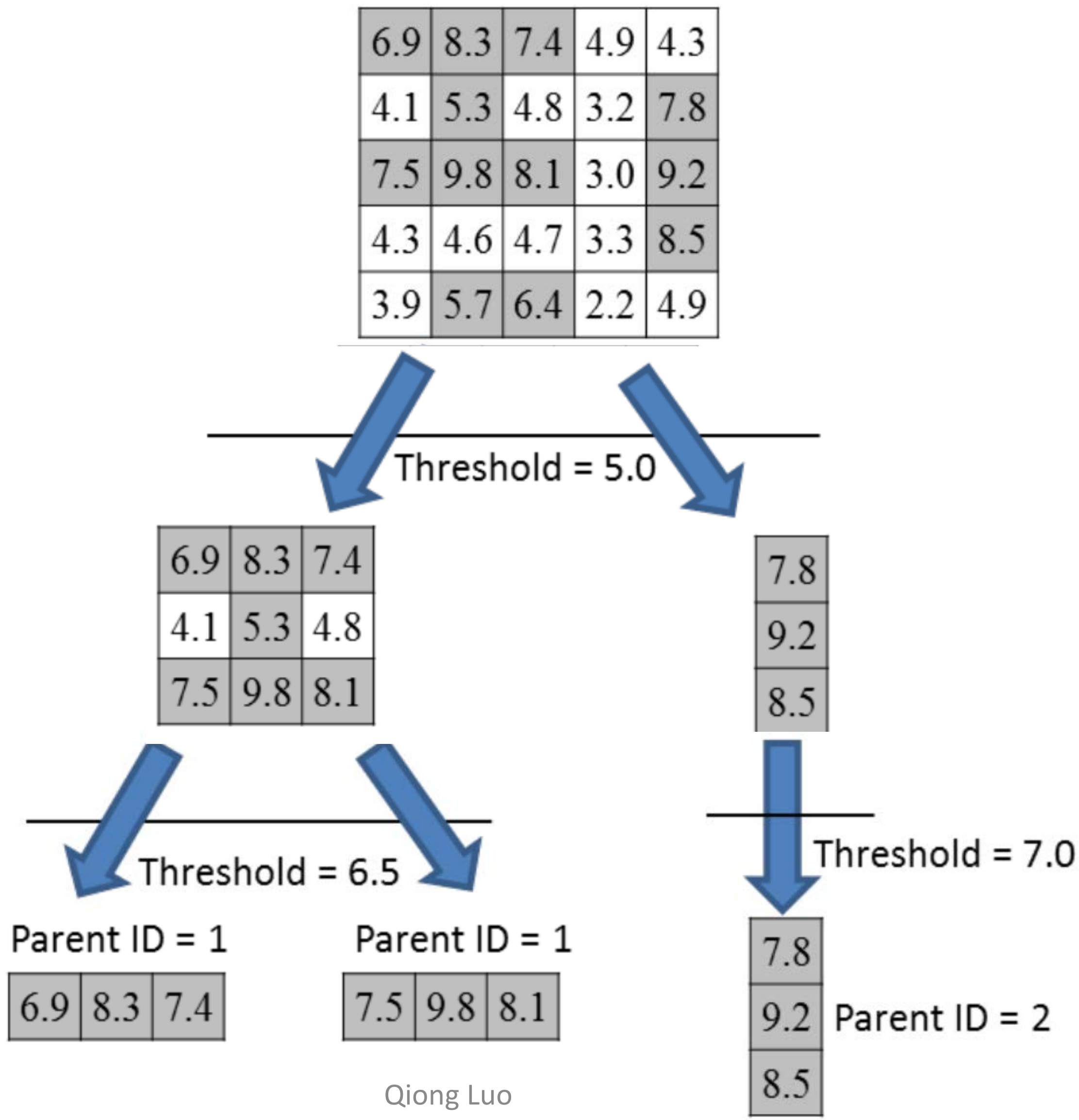
Why Multi-Level Detection?

- Raw detection threshold is set very low.
- Neighboring objects may be detected as a single raw object.
- Increasing the thresholds iteratively can split raw objects to a finer level.

Parallel Multi-Level Detection

- Initialization
  - Compute the multiple levels of threshold for each raw object in parallel
- Multi-Level Detection
  - Perform Detection from level 1 to level N
  - Compact the pixels after each level of detection
- Object Pruning
  - Flatten the trees and get a list of final objects

An Example: Two-Level Detection



Object Cleaning: Naïve Approach

- Remove falsely detected objects by pair-wise comparison
- Naive approach: use NxN threads for a list of N objects, with each thread responsible for one comparison operation
- Weakness: There are hundreds of thousands of objects, thus more than 10^10 comparison operations. Too time-consuming!

Object Cleaning: Window Approach

- Compare objects within a sliding window
  - “two objects close in the list are also close on the image”
  - “objects relatively far away on the image do not interfere with each other”

Advantage: Only N x *win_size* comparison operations required.

win_size is the size of the sliding window.

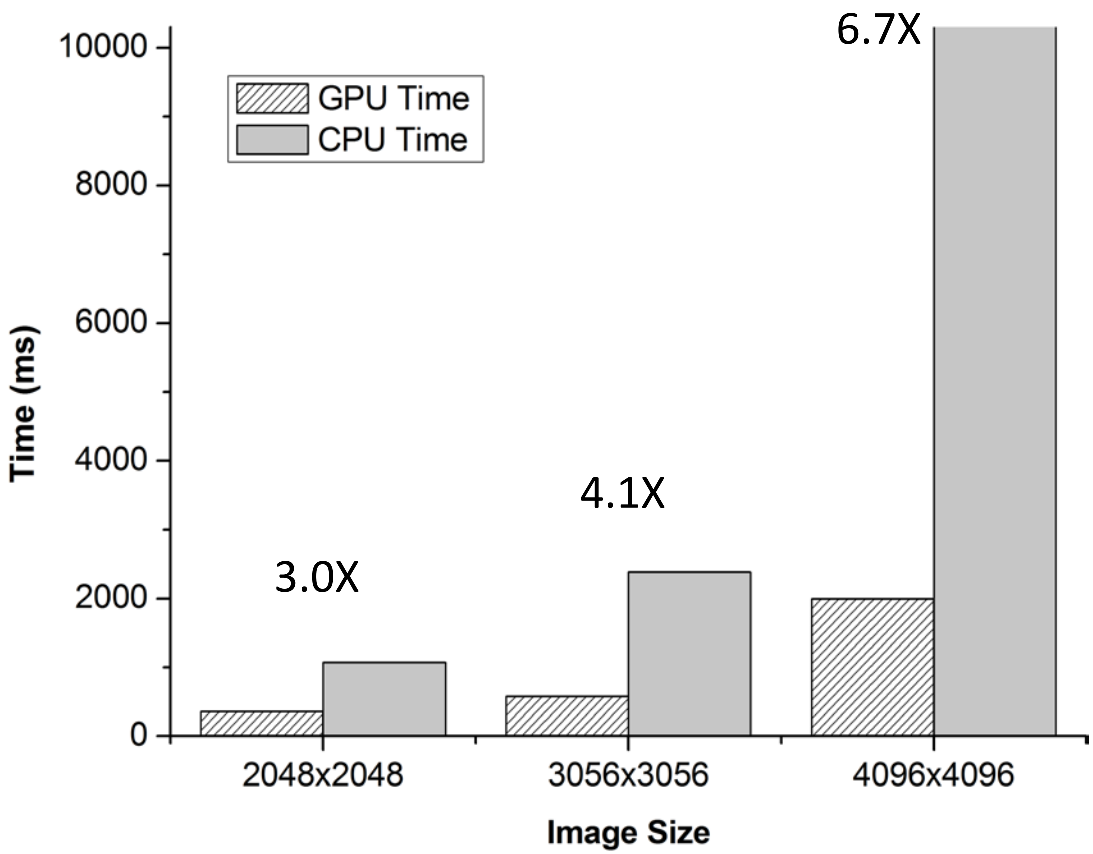
Parallel Object Measurement

- Compute the final attributes of each remaining object after cleaning
- Completely data-parallel since the measurement of each object is independent of others

Experimental Setup

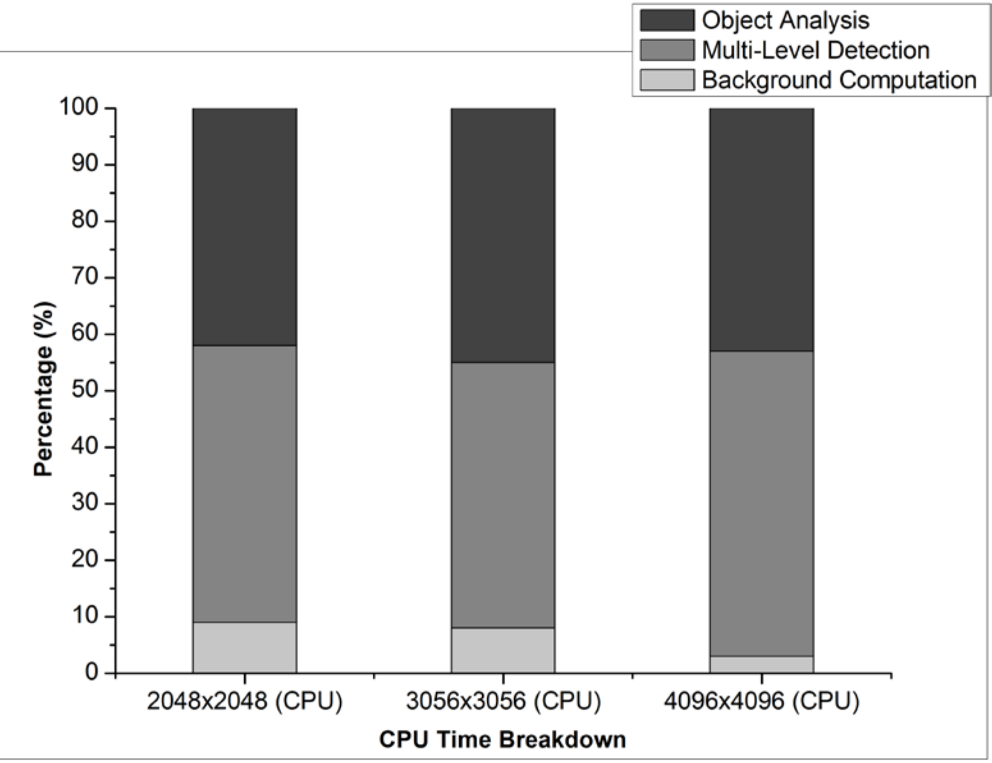
- Hardware & Software Setup:
  - CPU: Intel Core i7-3770 with 32GB main memory
  - GPU: NVIDIA GeForce GTX 670 with 4GB device memory
  - Ubuntu Linux 12.04 and CUDA SDK 5.0
- Data Set:
  - Both synthetic and real astronomical images
  - Synthetic images are generated using SkyMaker
  - Real-world image produced by telescope

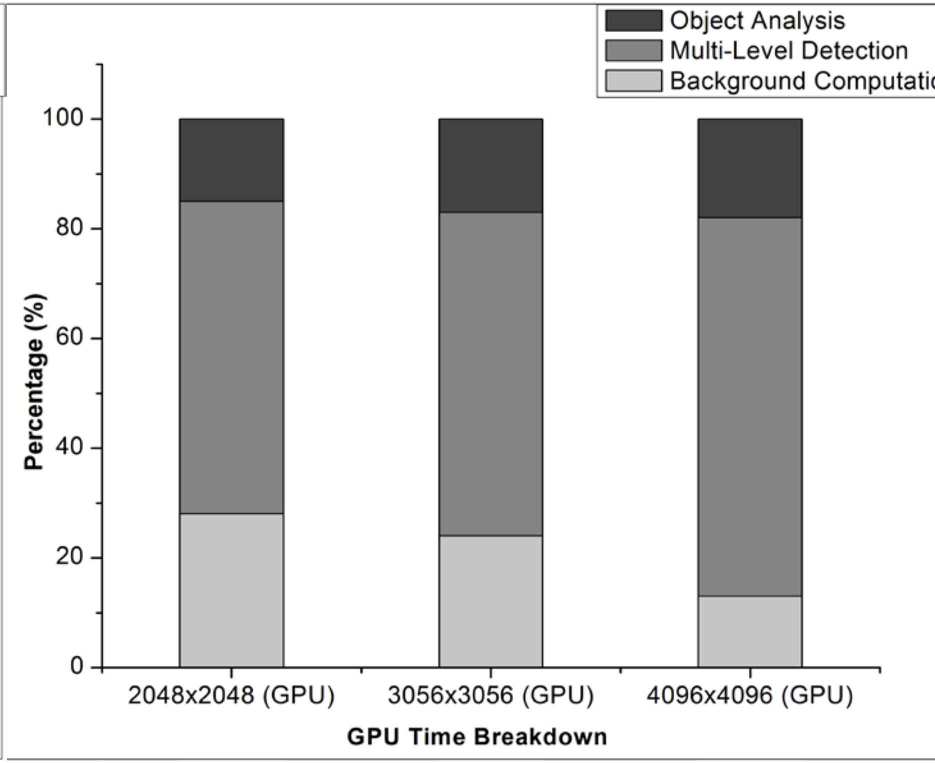
Overall Performance



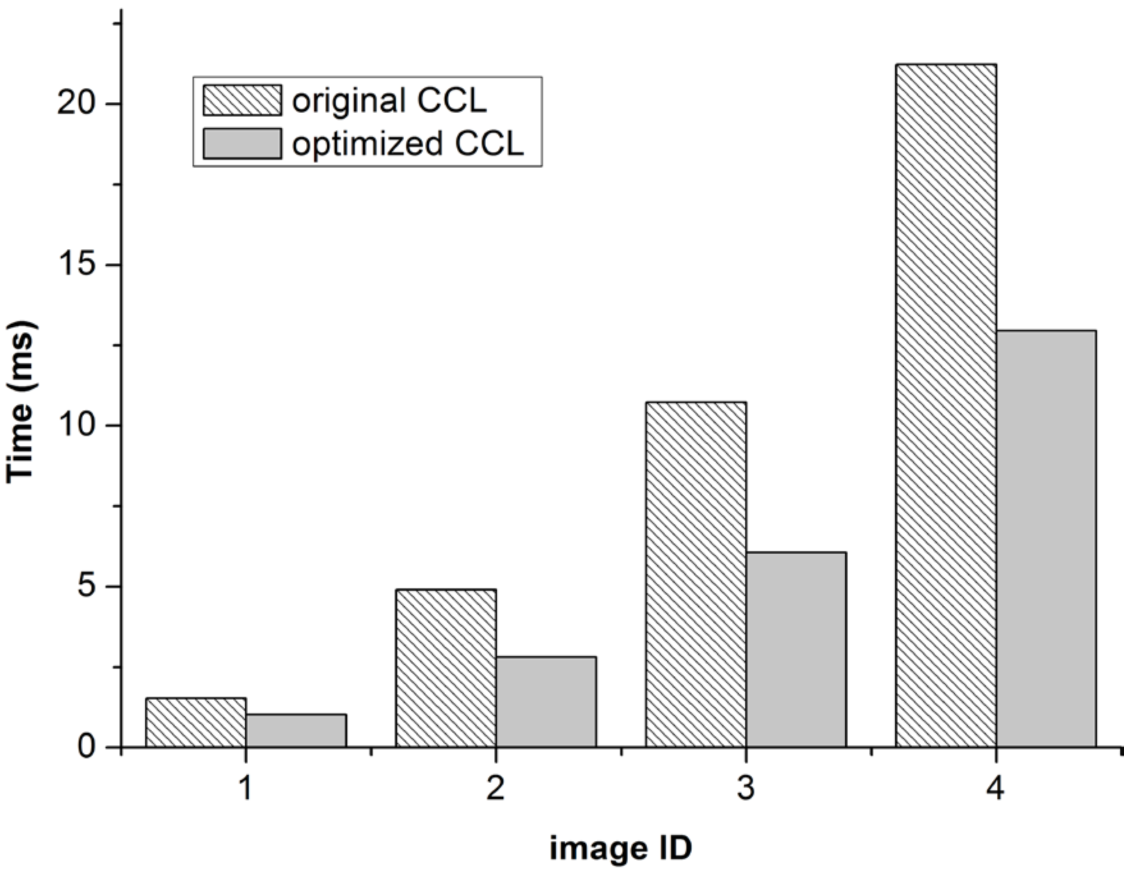
ID	Size	Objects
2	2048 x 2048	15,809
3	3056 x 3056	35,845
4	4096 x 4096	167,344

Time Breakdown



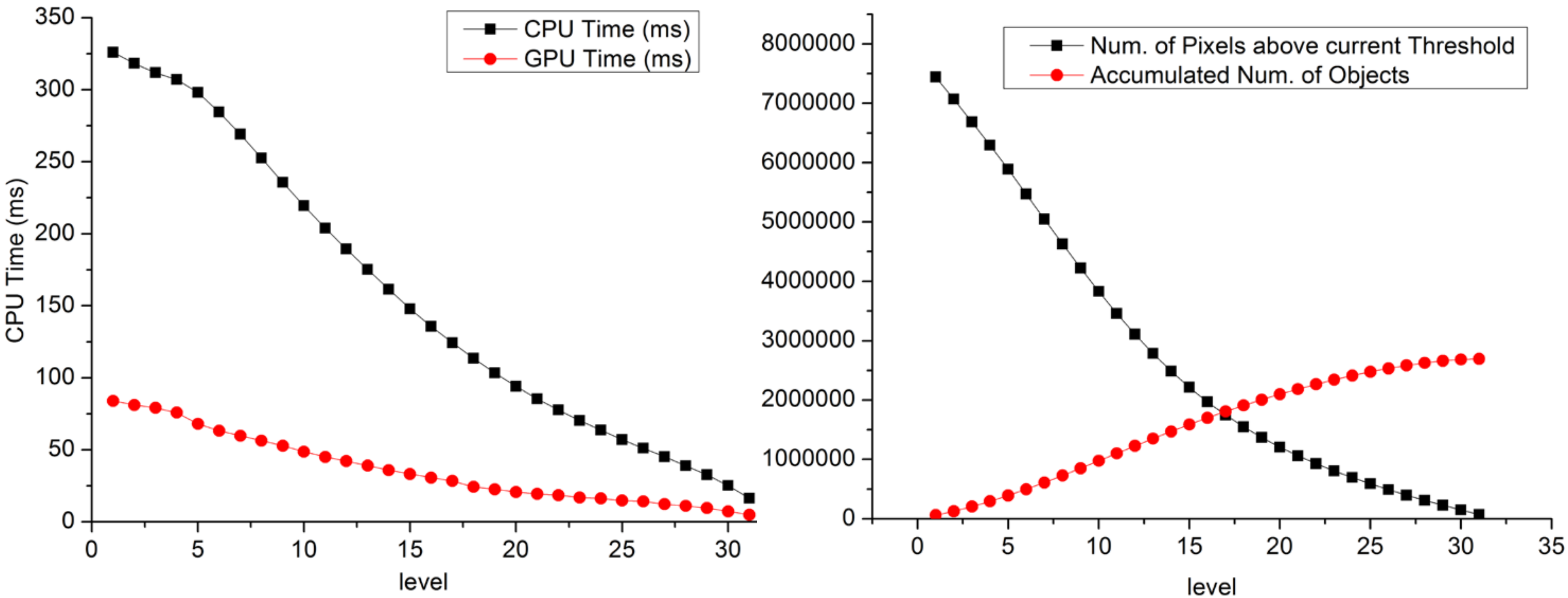


Connected Component Detection Performance



ID	Size	Objects
1	1601 x 1601	5,244
2	2048 x 2048	15,809
3	3056 x 3056	35,845
4	4096 x 4096	167,344

Multi-level Object Detection Performance



Summary

- The entire source extraction workflow is parallelized on the GPU.
- The maximum speedup is 6.7, reducing the time from >10 seconds to <2 seconds.
- The online processing requirements are satisfied.

The original SEXTRACTOR:
http://www.astromatic.net/software/sextractor

Baoxue's GPU-SEXTRACTOR:
https://github.com/starmsg/GPU-SExtractor