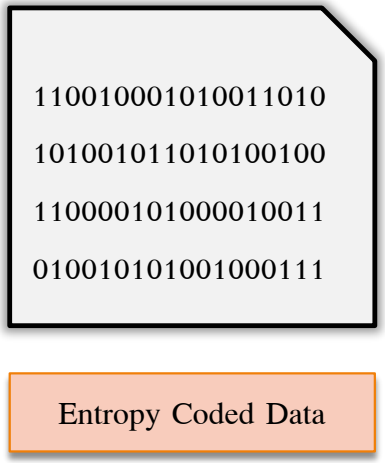
DYNAMIC PARTITIONING-BASED JPEG DECOMPRESSION ON HETEROGENEOUS MULTICORE ARCHITECTURES

Wasuwee Sodsong[1], Jingun Hong[1], Seongwook Chung[1], Yeongkyu Lim[2], Shin-Dug Kim[1] and Bernd Burgstaller[1]

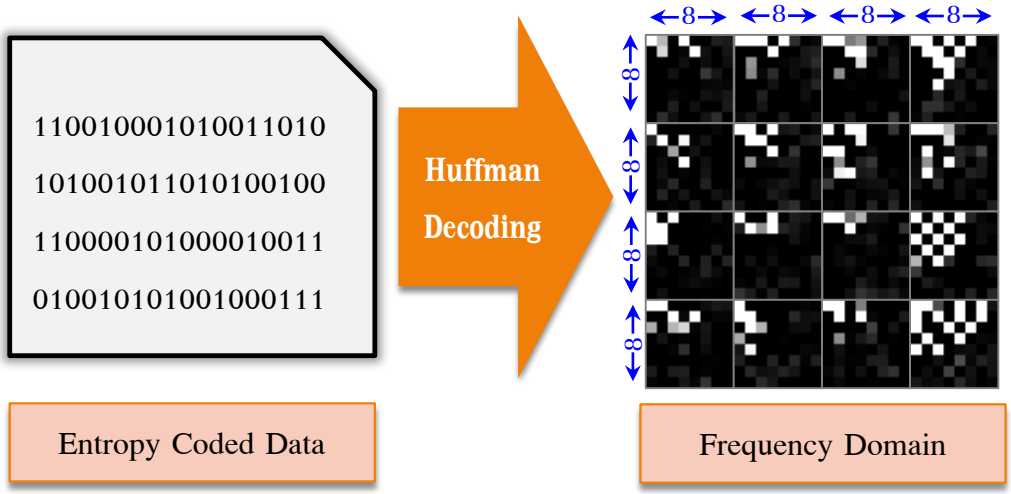
[1]Yonsei University

[2]LG Electronics

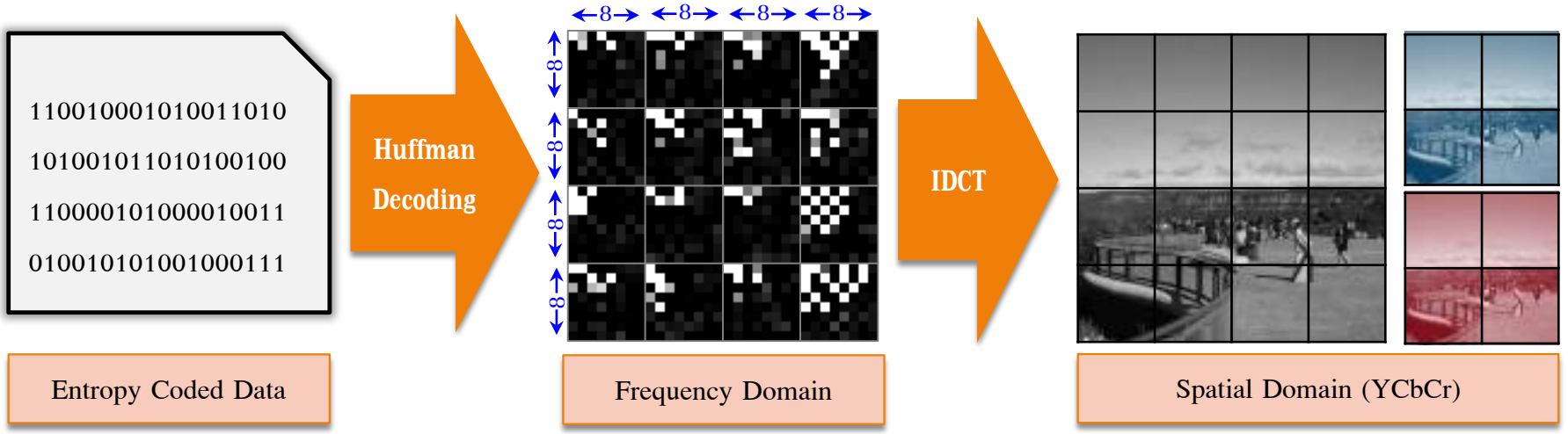
JPEG Decompression



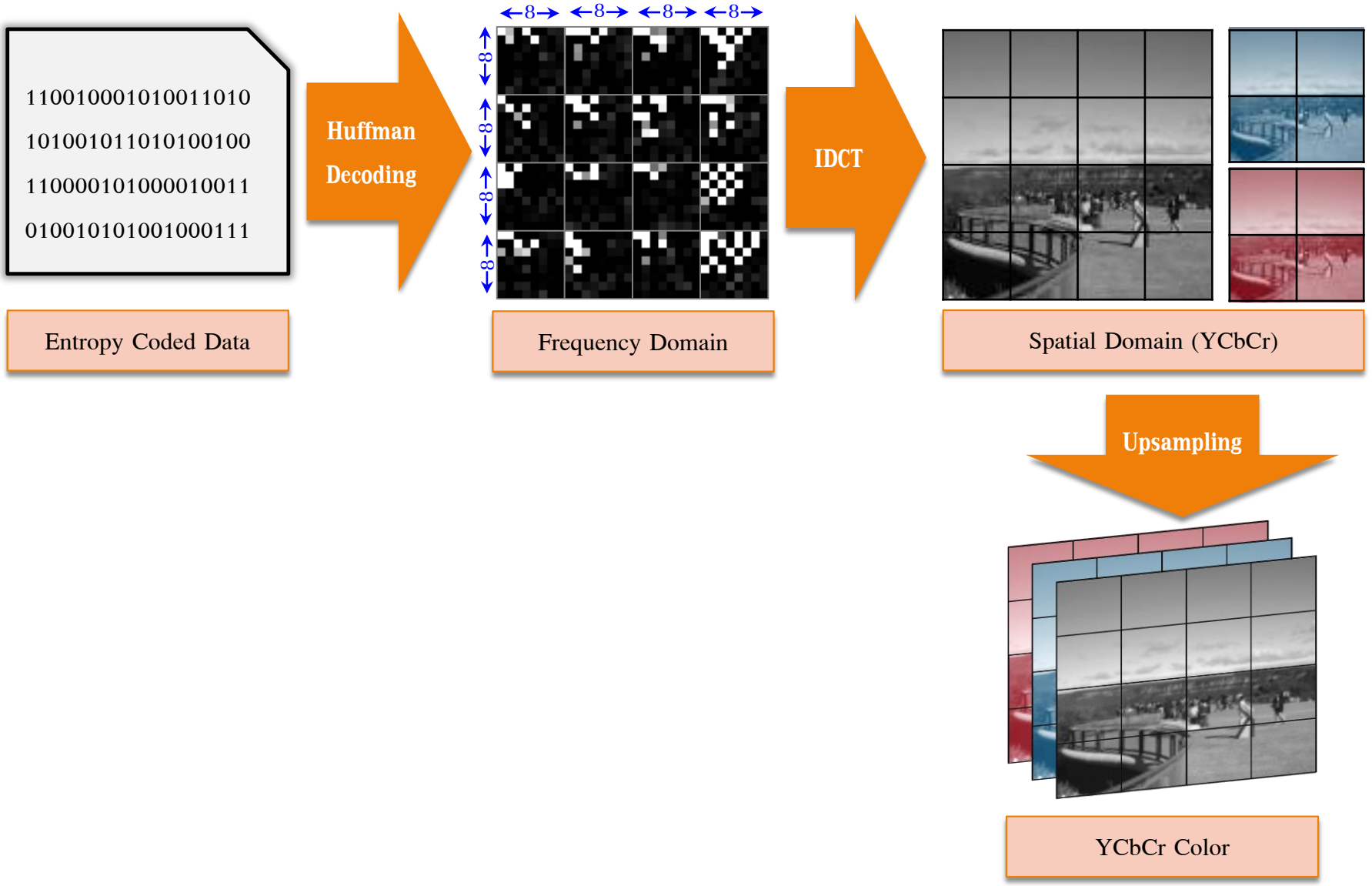
JPEG Decompression



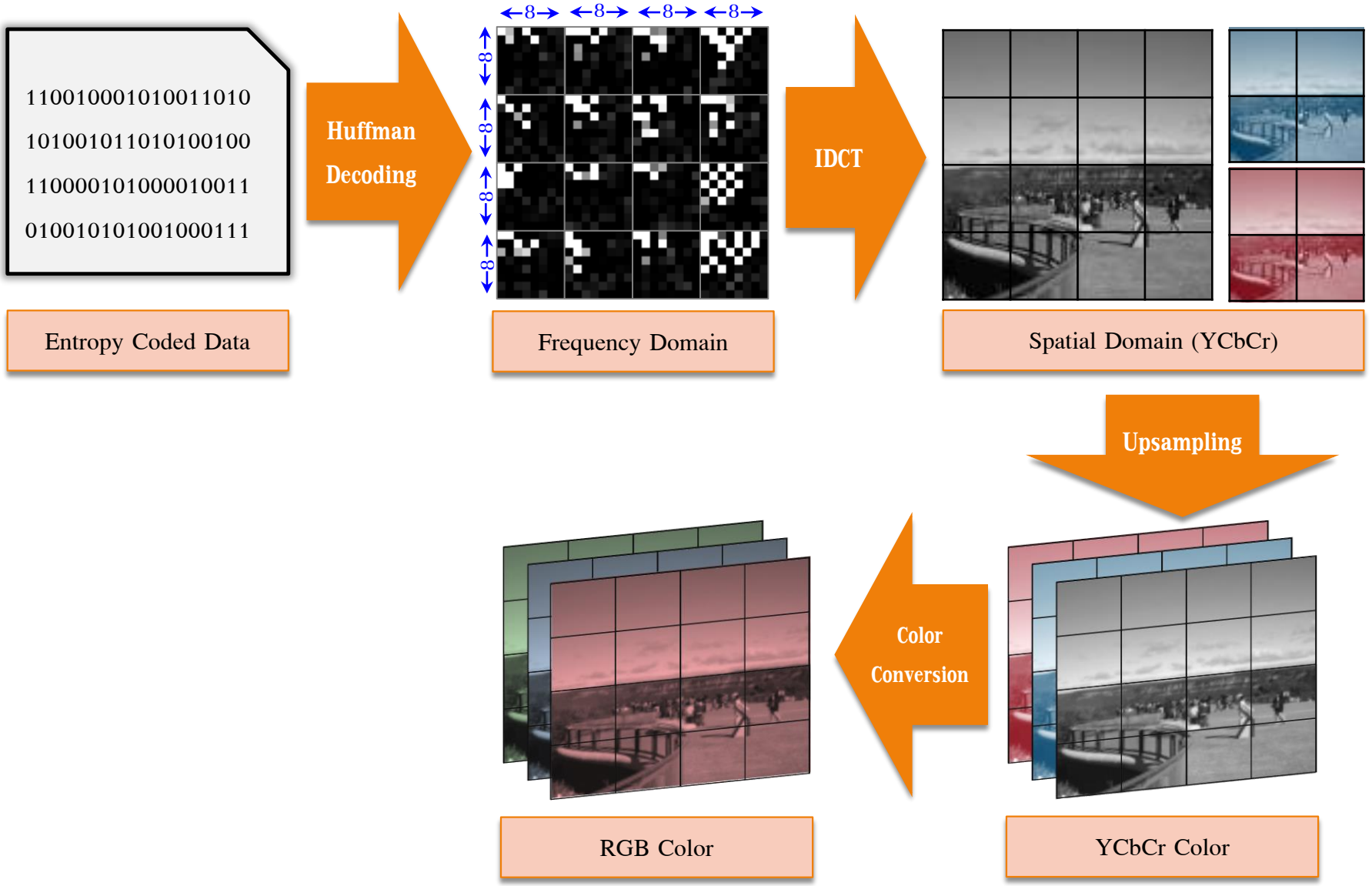
JPEG Decompression



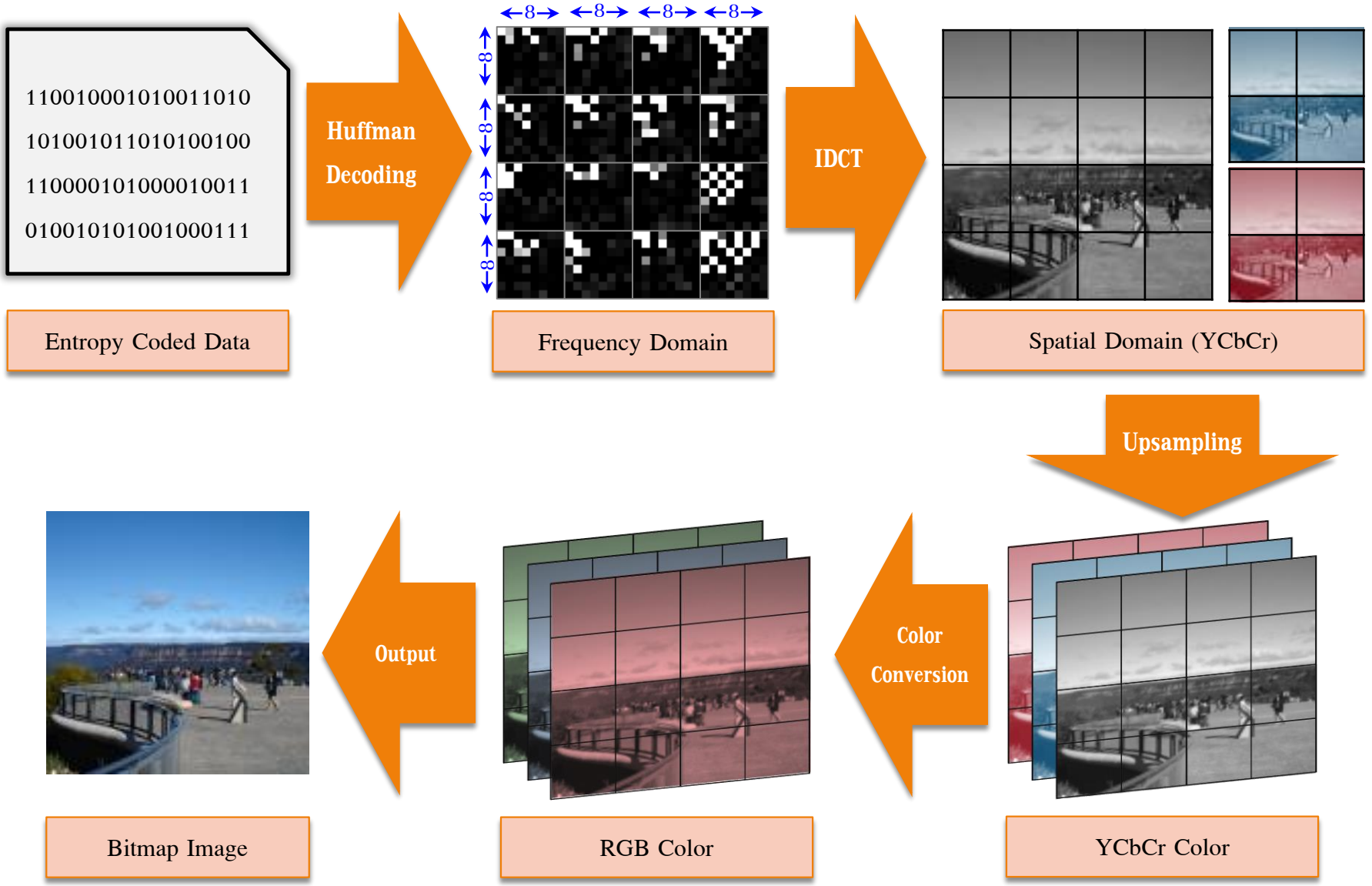
JPEG Decompression



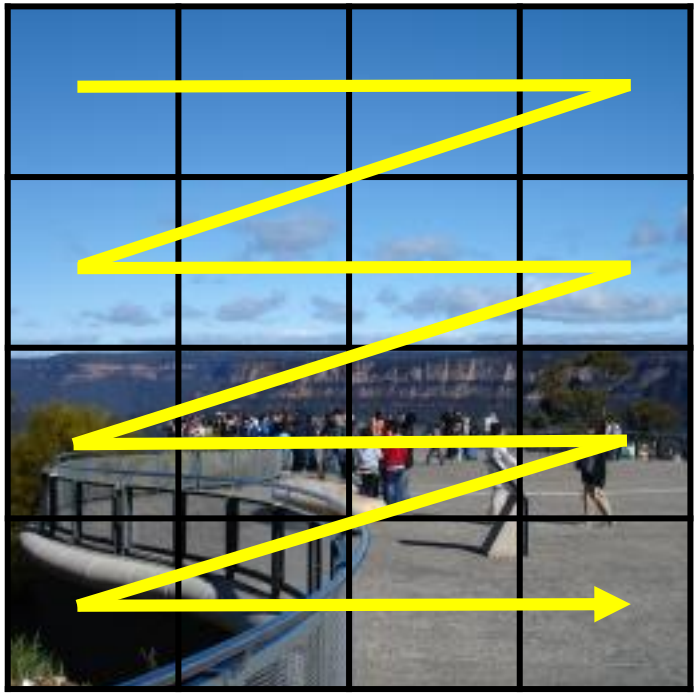
JPEG Decompression



JPEG Decompression

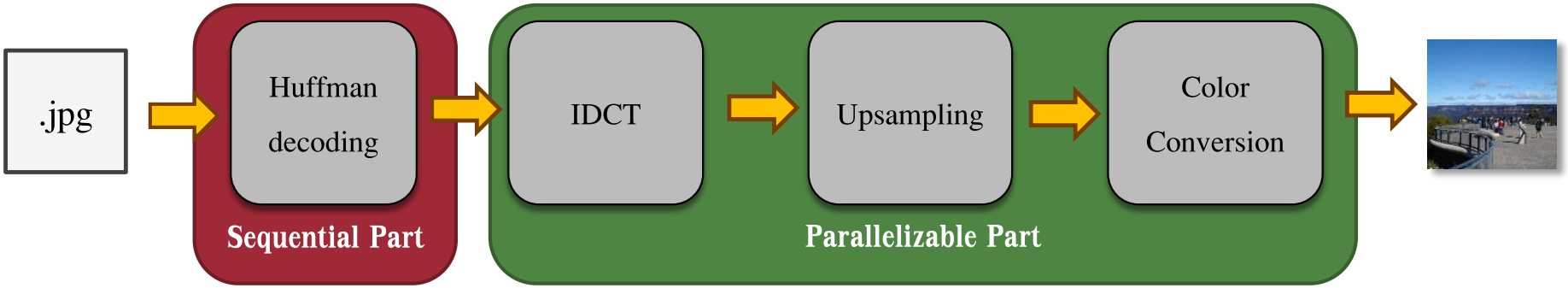


Sequential JPEG Decompression



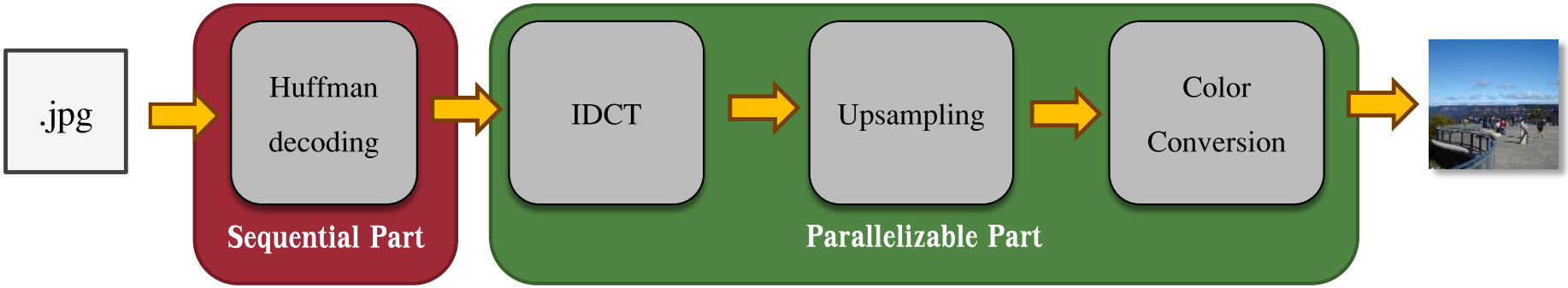
- JPEG is an **asymmetric compression**
  - Compression performs once per image
  - Decompression performs once per use
- 463 out of the 500 most popular websites use JPEG images
- Operates in blocks of 8x8 pixels
- Sequential JPEG decoders apply IDCT, upsampling and color conversion block-by-block

Parallelism in JPEG Decompression



- Sequential Part
  - Huffman decoding
  - NOT suitable for data-parallelism
    - Codewords have variable lengths.
    - The starting bit of a codeword in the encoded bitstream is only known once the previous codeword has been decoded.

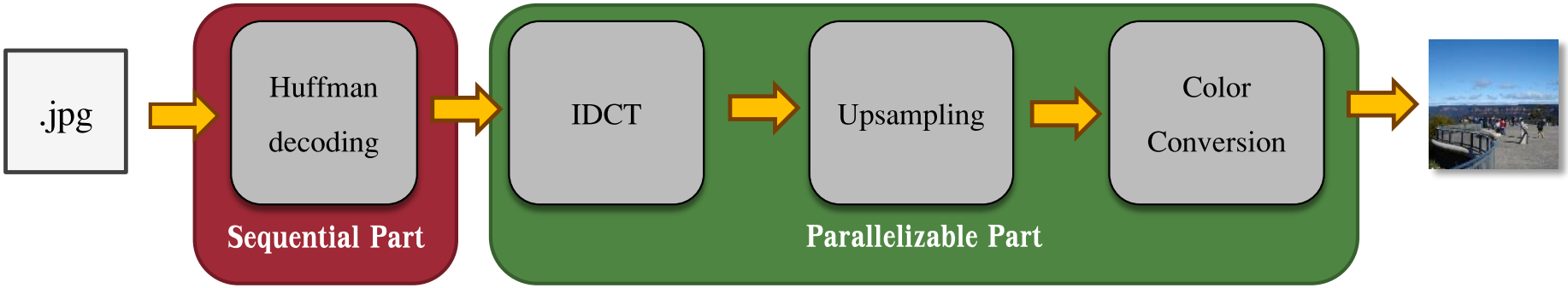
Parallelism in JPEG Decompression



- Sequential Part
  - Huffman decoding
  - NOT suitable for data-parallelism
    - Codewords have variable lengths
    - The starting bit of a codeword in the encoded bitstream is only known once the previous codeword has been decoded

- Parallelizable Part
  - IDCT, upsampling and color conversion
  - Suitable for GPU computing and SIMD operations on CPU
    - low data dependency
    - operates same instructions repeatedly
    - has fixed input and output sizes

Research Question



How to orchestrate JPEG decompression on CPU+GPU architectures?

- Input image characterized by
  - Width
  - Height
  - Entropy
- Need: work partitioning, schedule, execution infrastructure

Our Contributions

- Heterogeneous JPEG decoder on CPU+GPU architectures
  - profiling based performance model
  - dynamic partitioning scheme that automatically distributes the workload at run-time
- Pipelined execution model overlaps sequential Huffman decoding with GPU computations
- Parallelizable part is distributed across CPU and GPU
  - data-, task- and pipeline-parallelism
- GPU kernels designed to minimize memory access overhead
- libjpeg-turbo implementation and experimental evaluation for libjpeg-turbo library

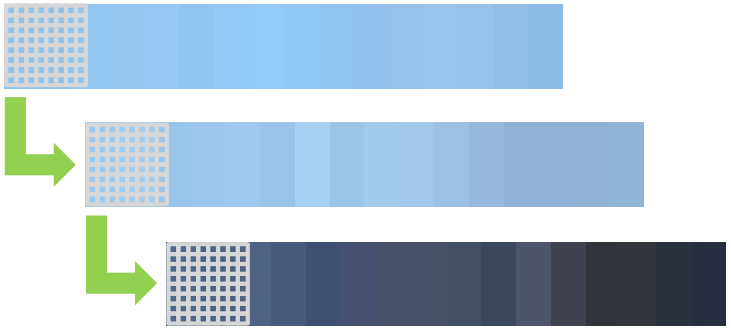
libjpeg & libjpeg-turbo

- libjpeg is a sequential JPEG compression reference implementation by Independent JPEG group
  - First version released in 1991
- libjpeg-turbo is a re-implementation of libjpeg
  - Utilizes SIMD instructions on x86 and ARM platforms.
  - Used by Google Chrome, Firefox, Webkit, Ubuntu, Fedora and openSUSE
- Both libraries strictly designed to conserve memory
  - Inhibits coarse-grained parallelism
  - A non-goal with today's target architectures

Re-engineering libjpeg-turbo

libjpeg-turbo

- To conserve memory, libjpeg-turbo decodes images in units of 8 pixel rows:



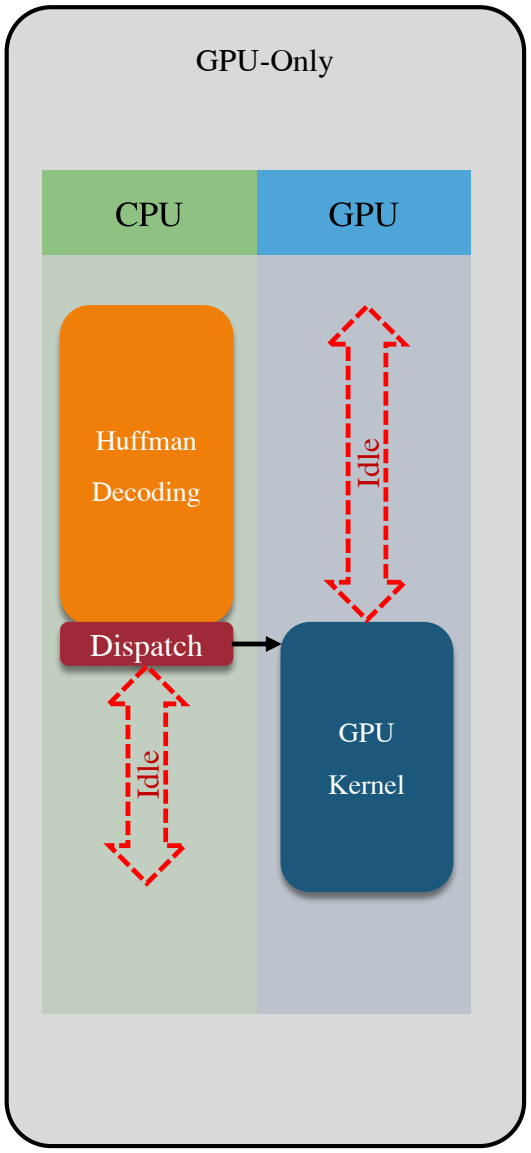
- 8 rows at a time do not contain enough computations to keep the data-parallel execution units of a GPU busy.
- Significant constant overhead per kernel invocation and data transfer (host→device→host).

Our Approach

- Store an entire image in memory:

- Fully utilizes all GPU cores by processing several larger image chunks.
- Reduce number of kernel invocations and data transferring overhead.

Heterogeneous JPEG Decompression Overview



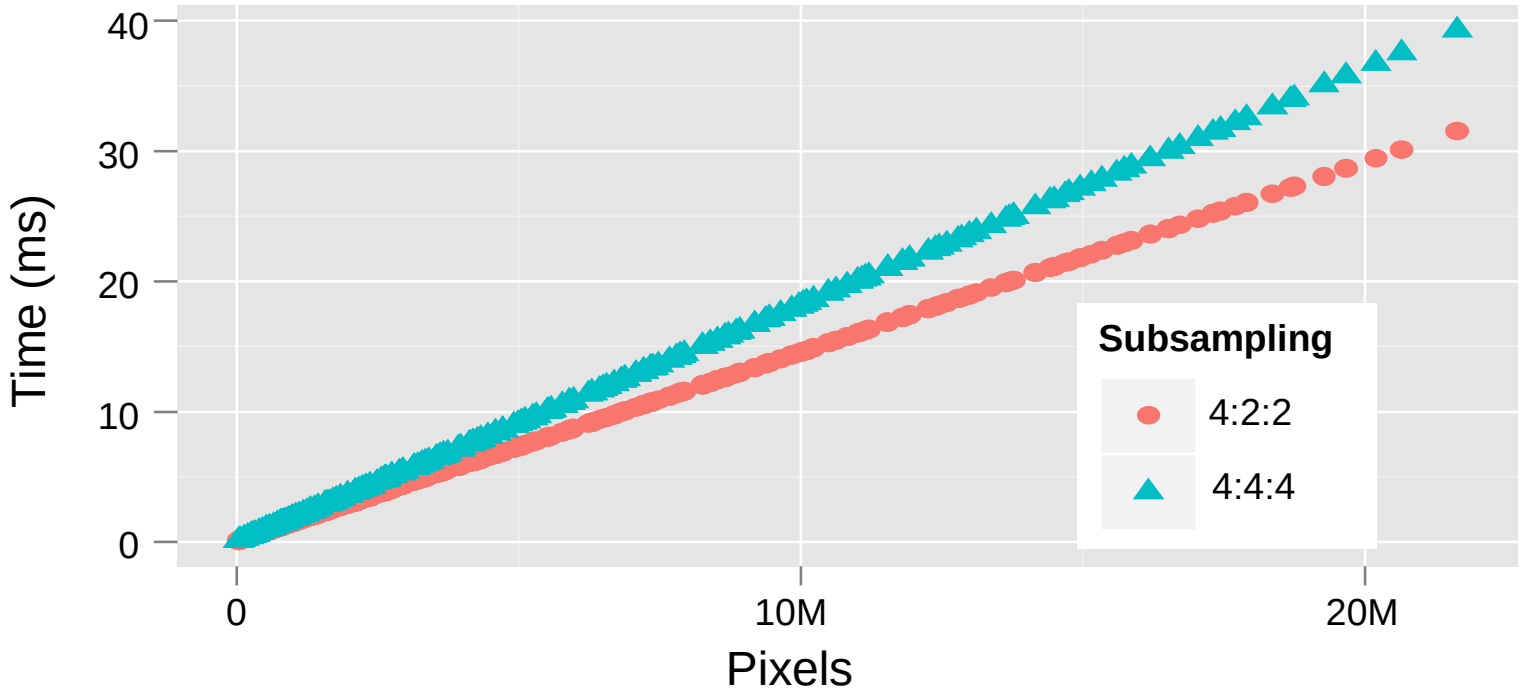
- Motivation: One architecture is unutilized when the other is processing
- Observation: No dependency among 8x8 pixel blocks. Thus, the CPU and the GPU can compute in parallel
- Goal: Find partitioning size at runtime such that the load on the CPU and the GPU are balanced
- Requirement: Performance model through offline profiling

Performance Model

- Offline profiling step on image training set
  - 19 master images cropped to various sizes
  - Maximum image size is 25 megapixels
- Profile execution time of the sequential part and the parallelizable part on CPU and GPU
- Model all decompression steps using multivariate polynomial regression up to degree 7
- Select the best-fit model by comparing Akaike information criterion (AIC) values

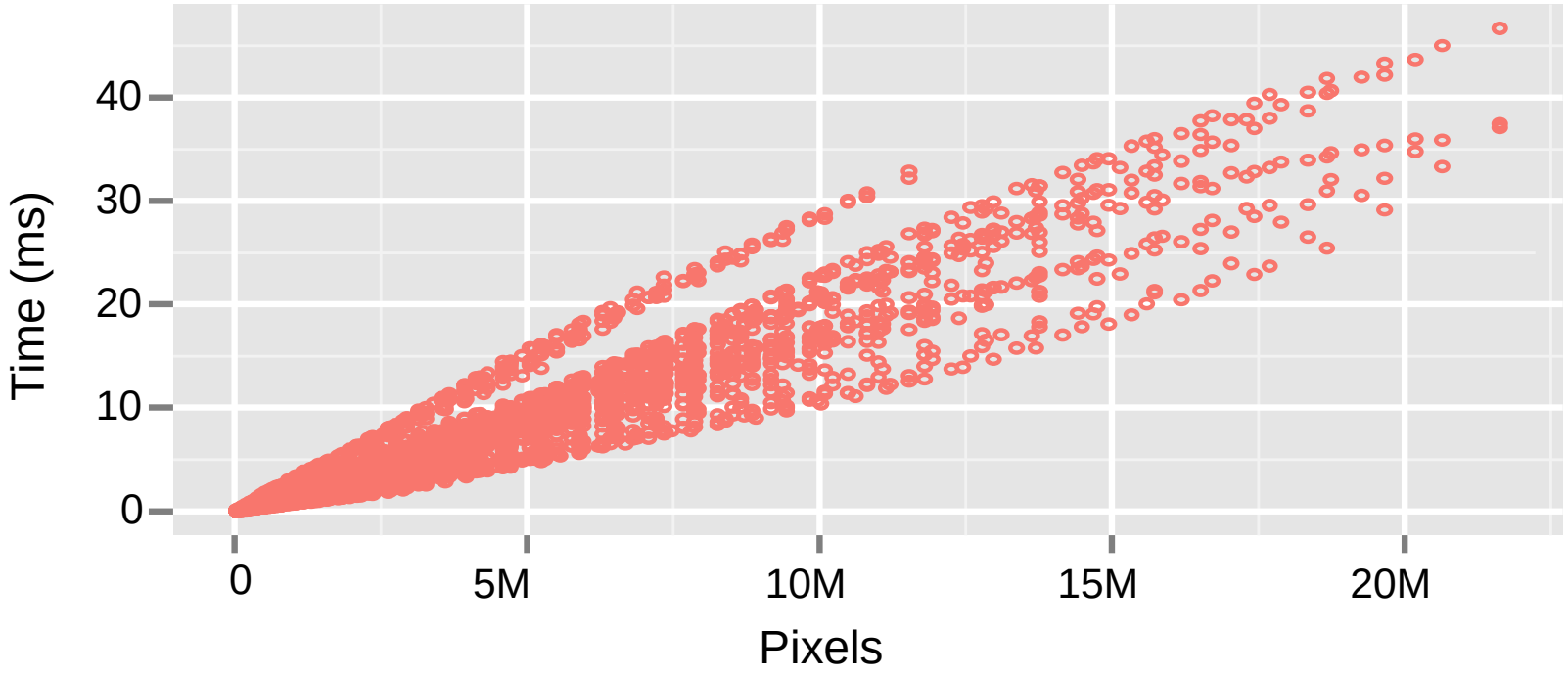
Performance Model for the Parallelizable Part

- Linearly scales as image size increased
- Image dimension is known at the beginning of the decompression step
- **Parameters: width and height**



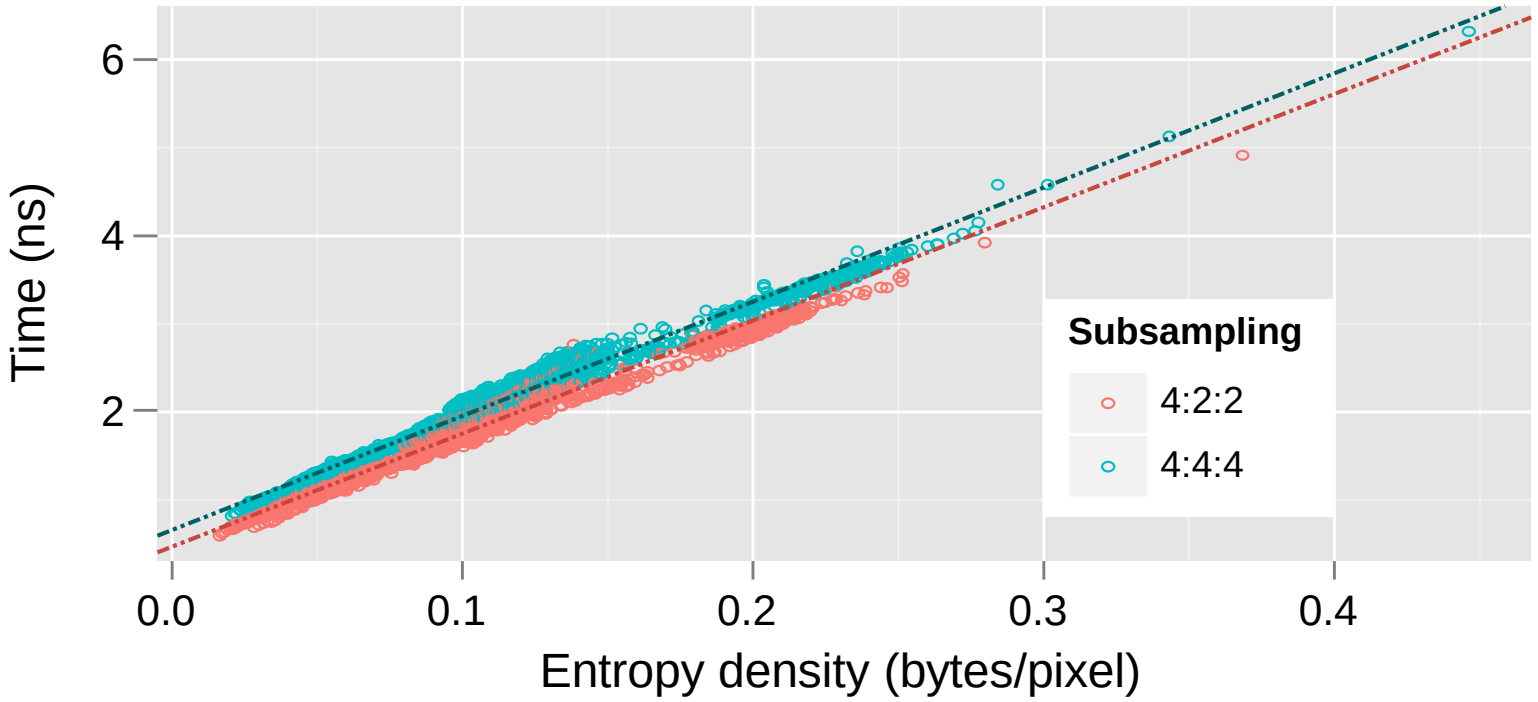
Performance Model for the Sequential Part

- Unlike the parallelizable part, Huffman decoding time does NOT have a high correlation with image width and height.

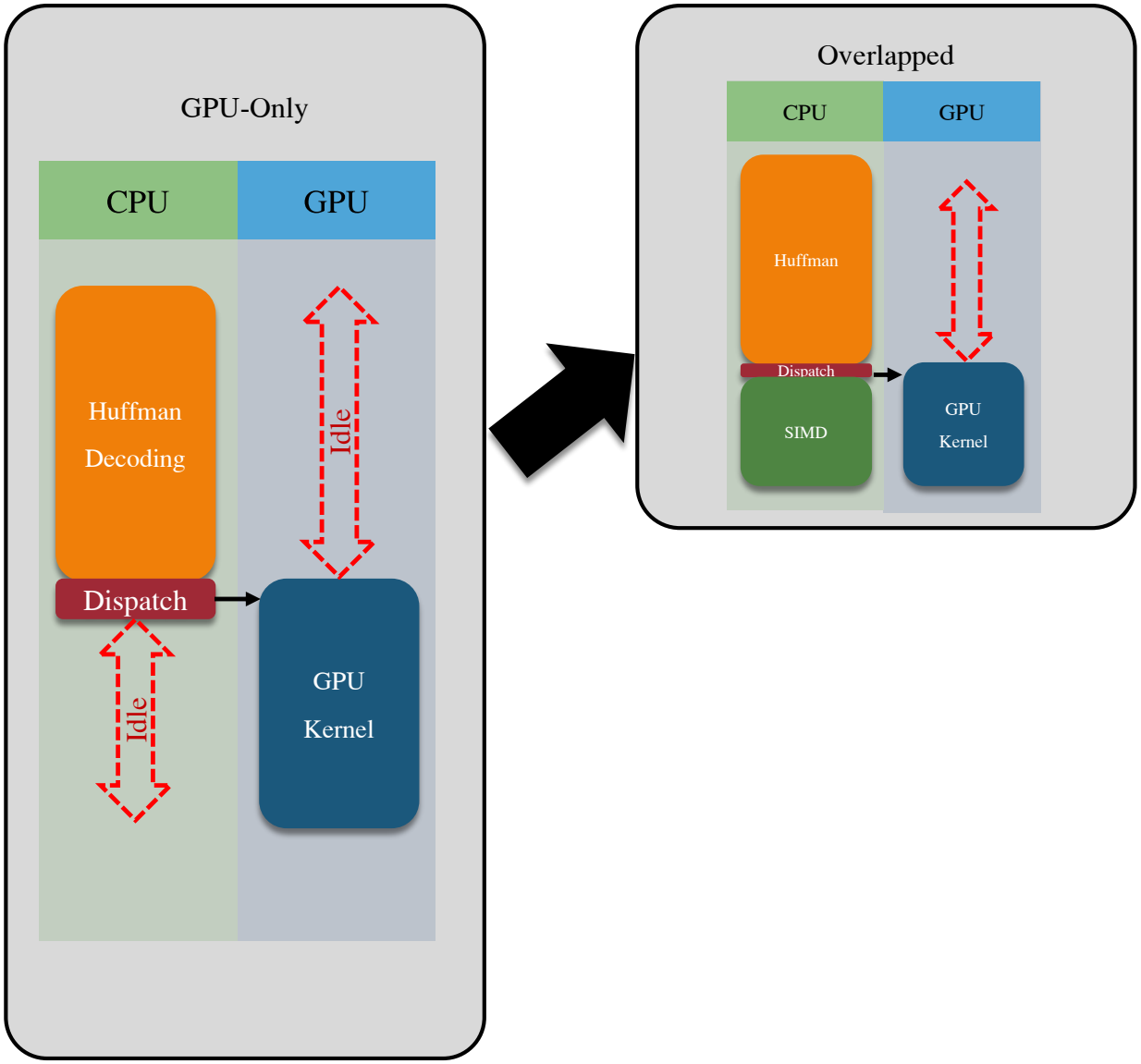


Performance Model for the Sequential Part

- Huffman decoding time has a high correlation with the size of entropy coded data.
- We have observed a linear trend as entropy density increased, entropy size in bytes per pixel.
- **Parameters: width, height and entropy size**
- Entropy size can be roughly approximated from JPEG file size.

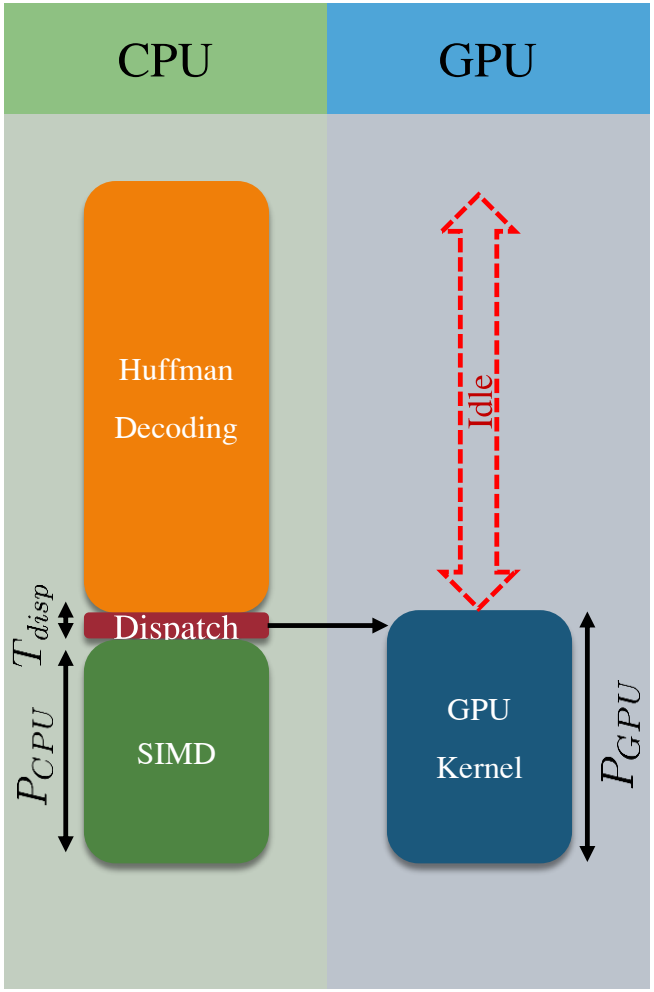


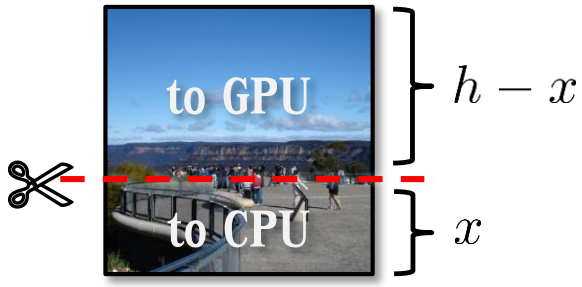
Overlapped Partitioning Scheme



- Sharing workload of the parallelizable part between CPU and GPU

Overlapped Partitioning Scheme





- Idea: Share workload of the parallelizable part on the CPU and the GPU.
- Partitioning equation can be formulated as

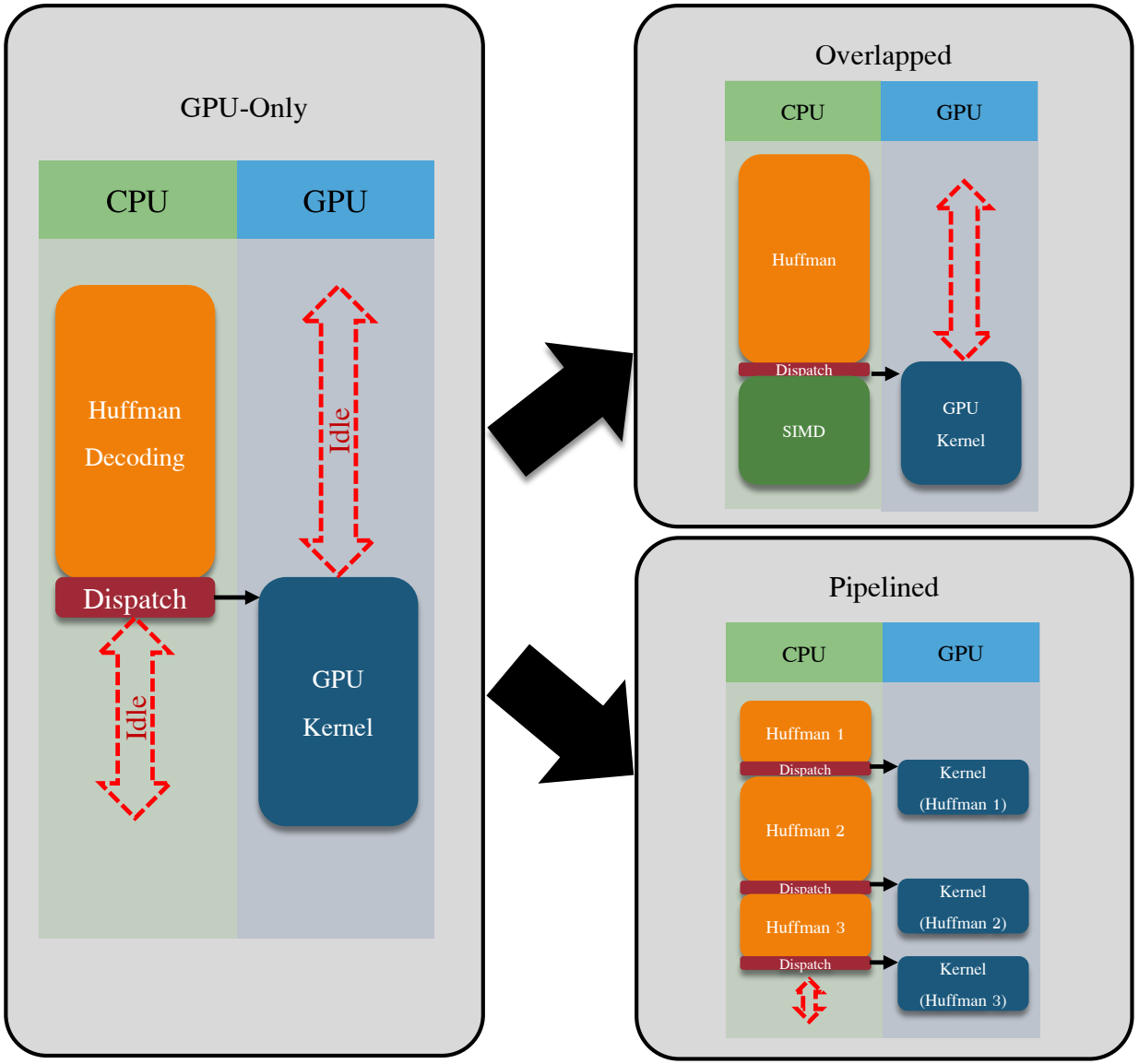
$$f(x) = T_{disp}(w, h-x) + P_{CPU}(w, x) - P_{GPU}(w, h-x),$$

where x is number of rows given to CPU, and

w, h are image width and height.

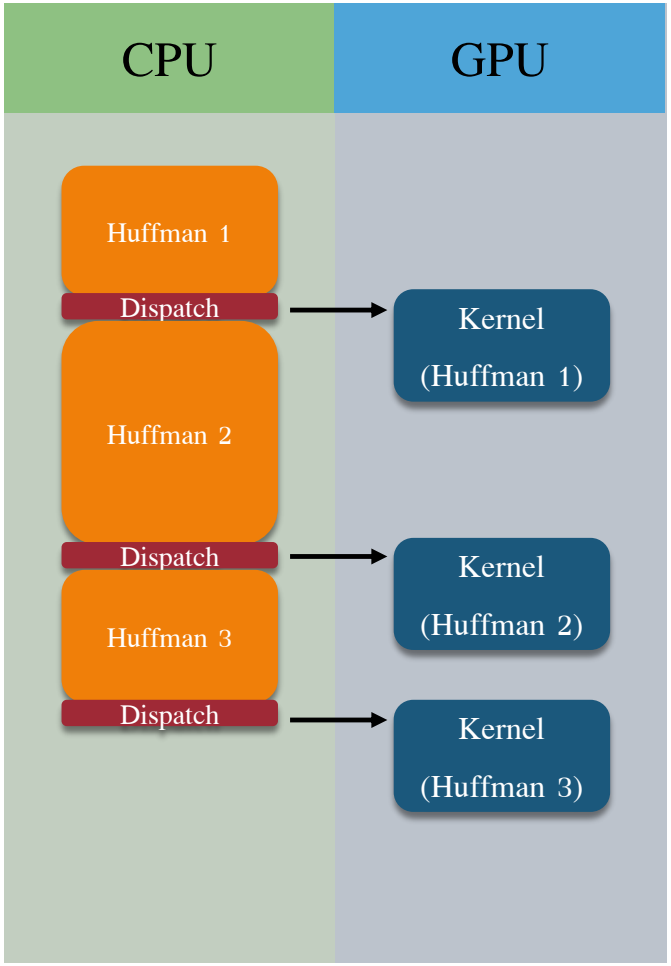
- When $f(x) = 0$, the time spent on the CPU and GPU are equaled.
- w and h are known at runtime. We can use Newton's method to solve for x.
- Problem: GPU is unutilized during Huffman decoding.

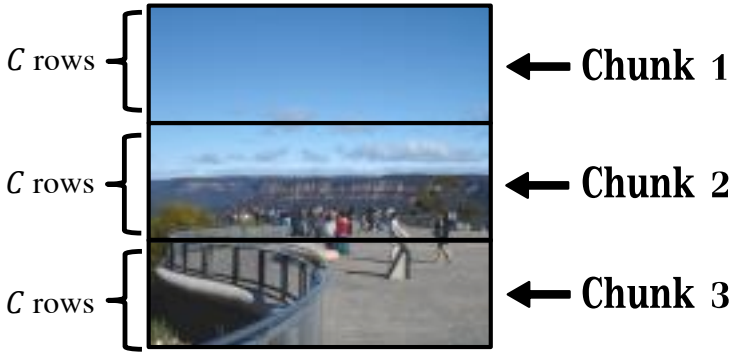
Pipelined Partitioning Scheme



- Sharing workload of the parallelizable part between CPU and GPU
- Increase parallelism by performing Huffman decoding and GPU kernel in pipelined fashion

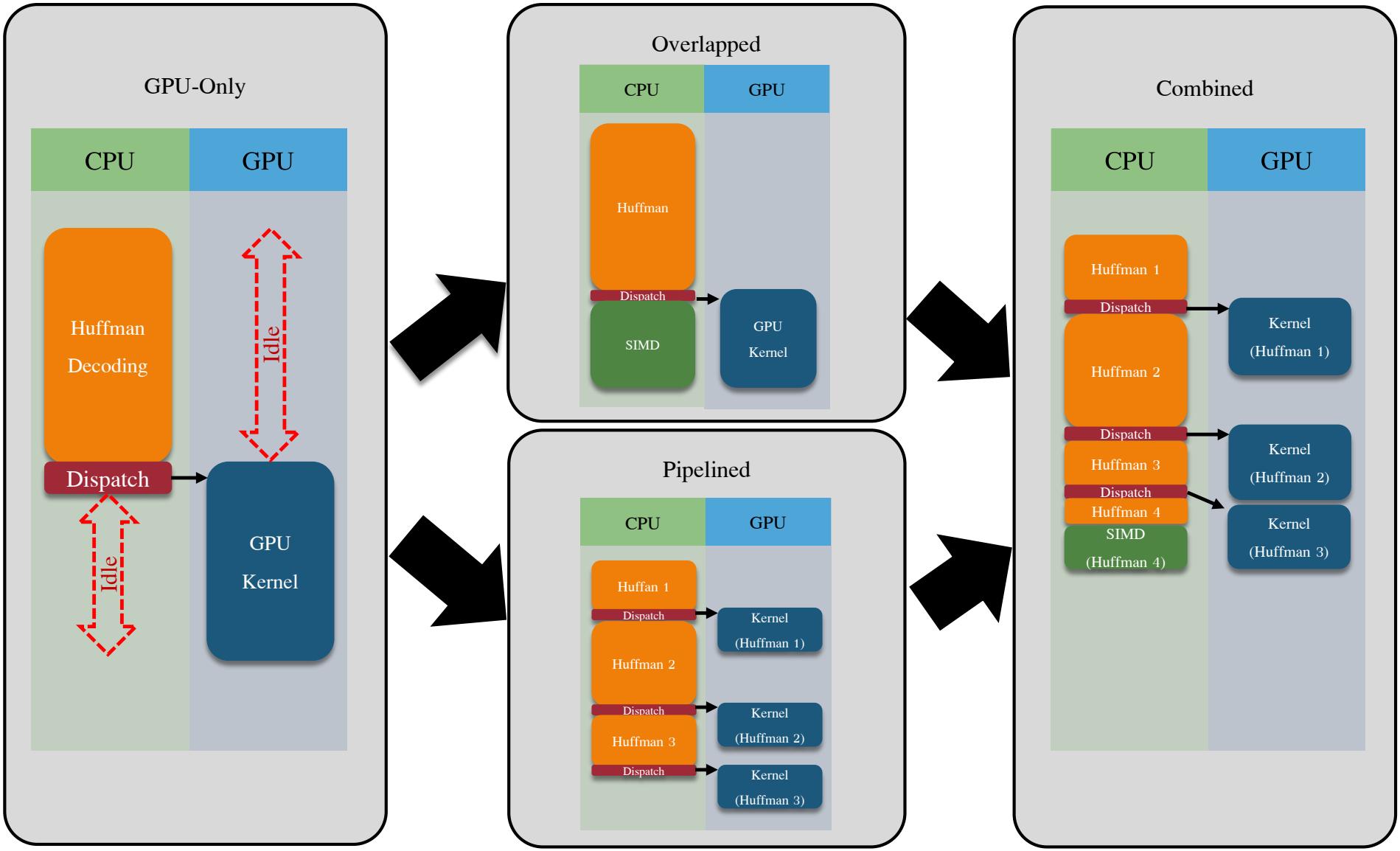
Pipelined Partitioning Scheme



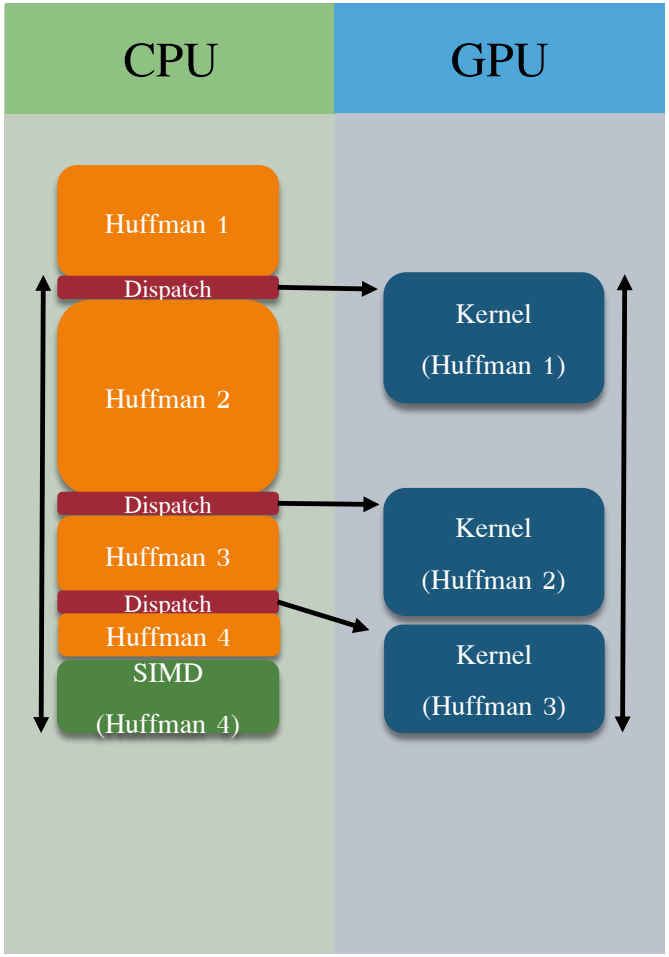


- Idea: Execute Huffman decoding in a pipelined fashion with GPU kernel.
- Split an image into several chucks of c rows.
- An optimal chunk size is found through a profiling.
- We can start kernel invocation as soon as an image chunk is decoded.
- On a fast GPU, only the execution time of last chunk is visible to users.
- Problem: Does NOT guarantee improvement over CPU computation.

Combined Partitioning Scheme



Combined Partitioning Scheme



- Combining overlapped and pipelined model to guarantee improvement.

$$f(x) = T_{Huff}(w, h-c, d) + T_{disp}(w, h-x) + P_{CPU}(w, x) - P_{GPU}(w, h-x),$$

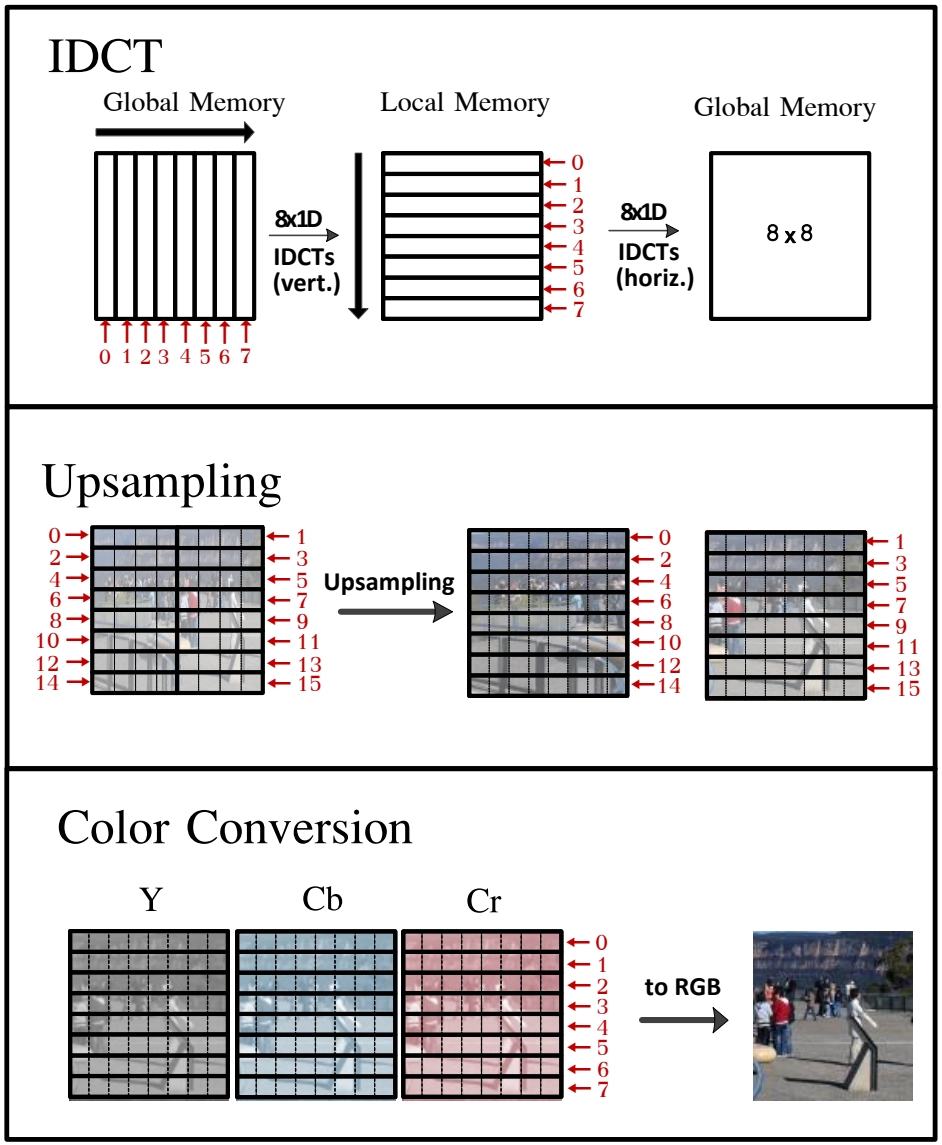
- where c is number of rows in a chunk, and

  d is entropy density in bytes per pixel

- Using Newton's method to solve for x at runtime.
- Estimation errors from Huffman decoding
  - Assume the same Huffman decoding time for every pixel across an image
  - Entropy is not distributed evenly in practice.
- Perform re-partitioning before Huffman decoding for the last GPU kernel

GPU Optimizations

- We implemented GPU kernels for IDCT, upsampling and color conversion.
- Optimizations
  - Vectorization to reduce number of reads/writes to global memory
  - Store intermediate results in local memory (NVIDIA's shared memory)
  - Map work-items with consideration of coalesced memory access
  - Combine color conversion kernel with the prior kernel to avoid global memory store between kernels.

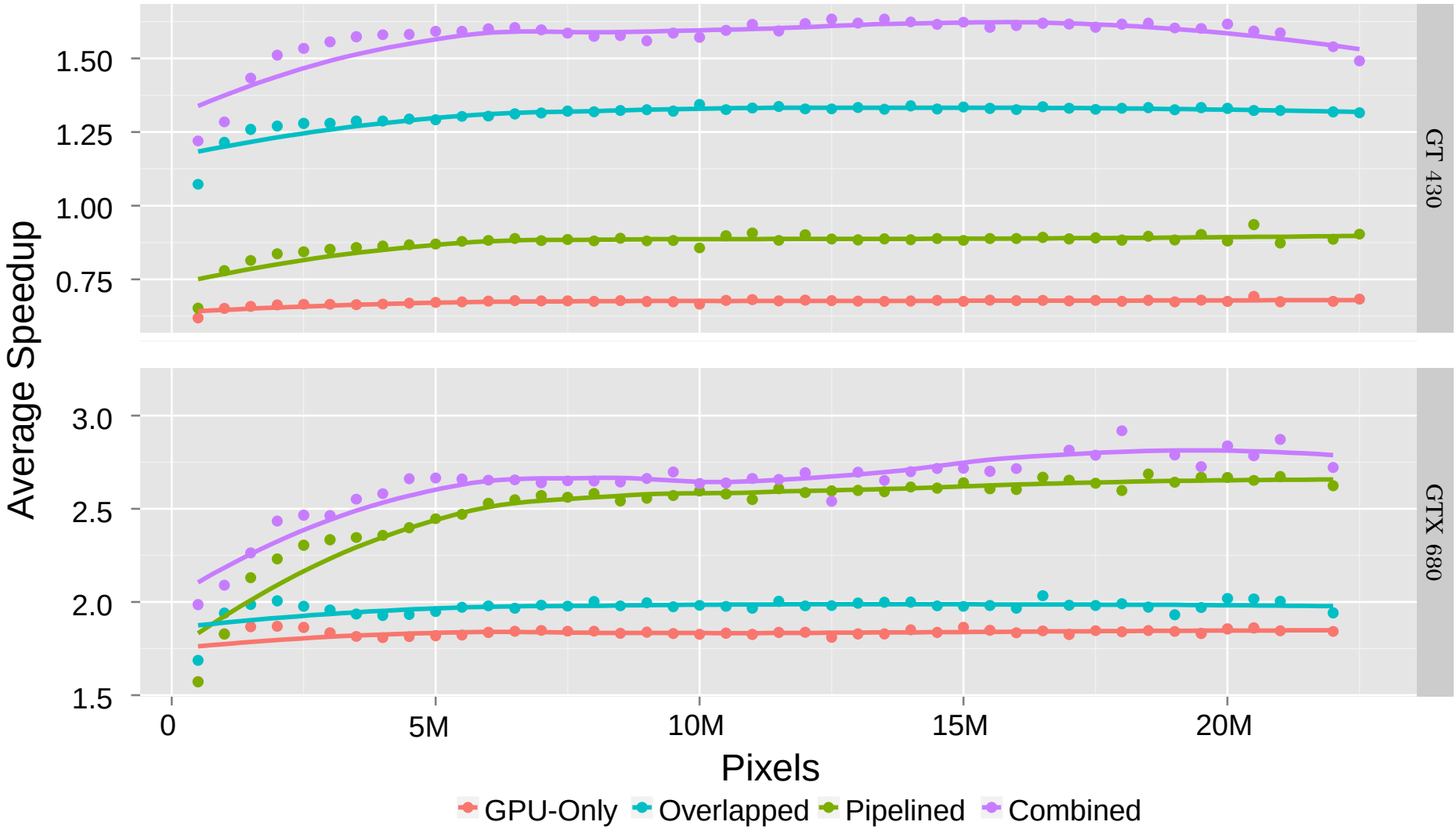


Experimental Setup

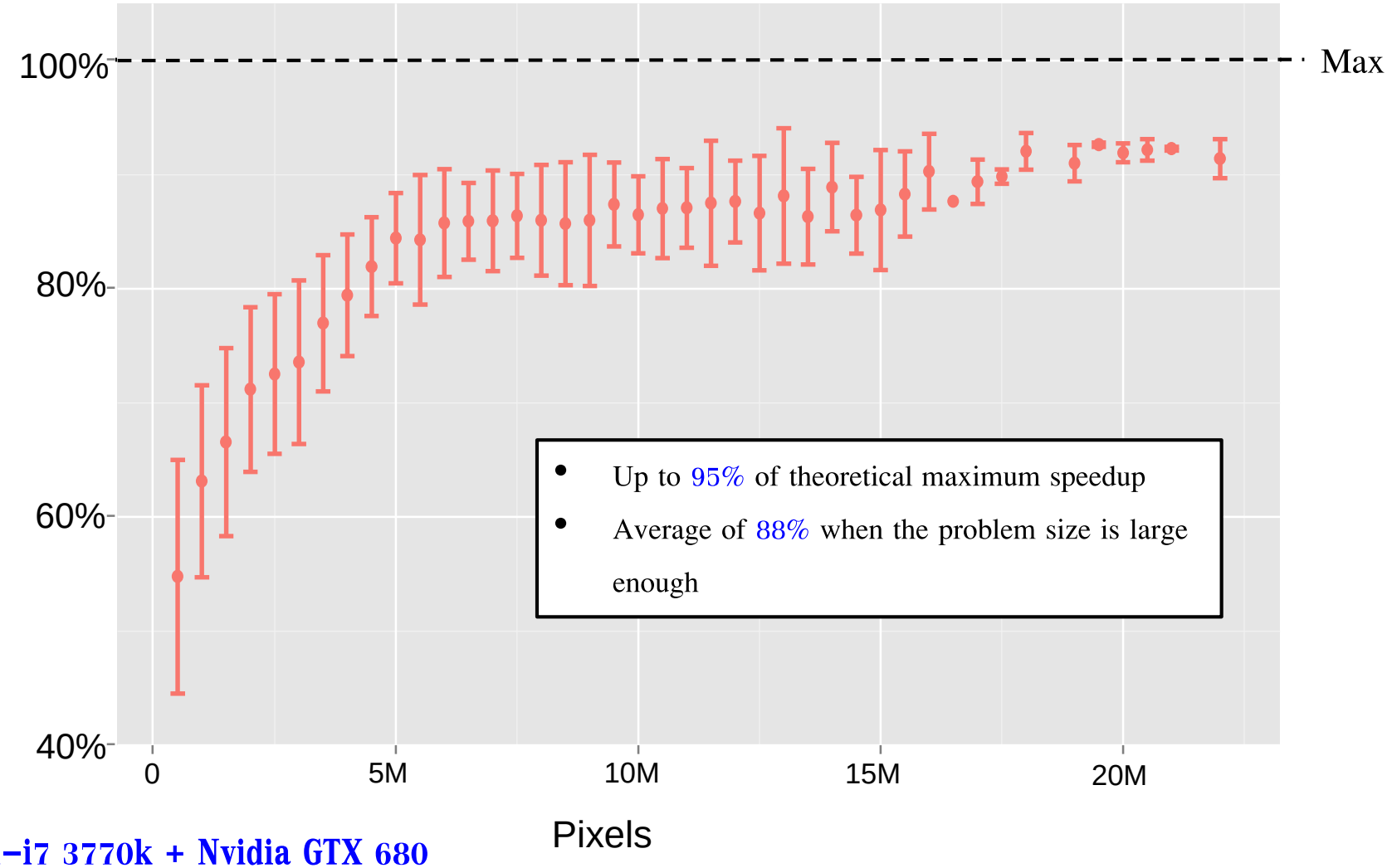
- Test set
  - A new set of images. No images are reused from the training set
  - 3591 images of various size
  - Maximum image size is 25 megapixels
- Hardware specification

Machine Name	GT 430	GTX 560	GTX 680
CPU model	Intel i7-2600k	Intel i7-2600k	Intel i7-3770k
CPU frequency	3.4 GHz	3.4 GHz	3.5 GHz
GPU model	NVIDIA GT 430	NVIDIA GTX 560Ti	NVIDIA GTX 680
GPU core frequency	700 MHz	822MHz	1006MHz
No. of GPU cores	**96**	**384**	**1536**
GPU memory size	1024 MB	1024 MB	2048 MB
Compute Capability	2.1	2.1	3.0

Speedup Over SIMD Version

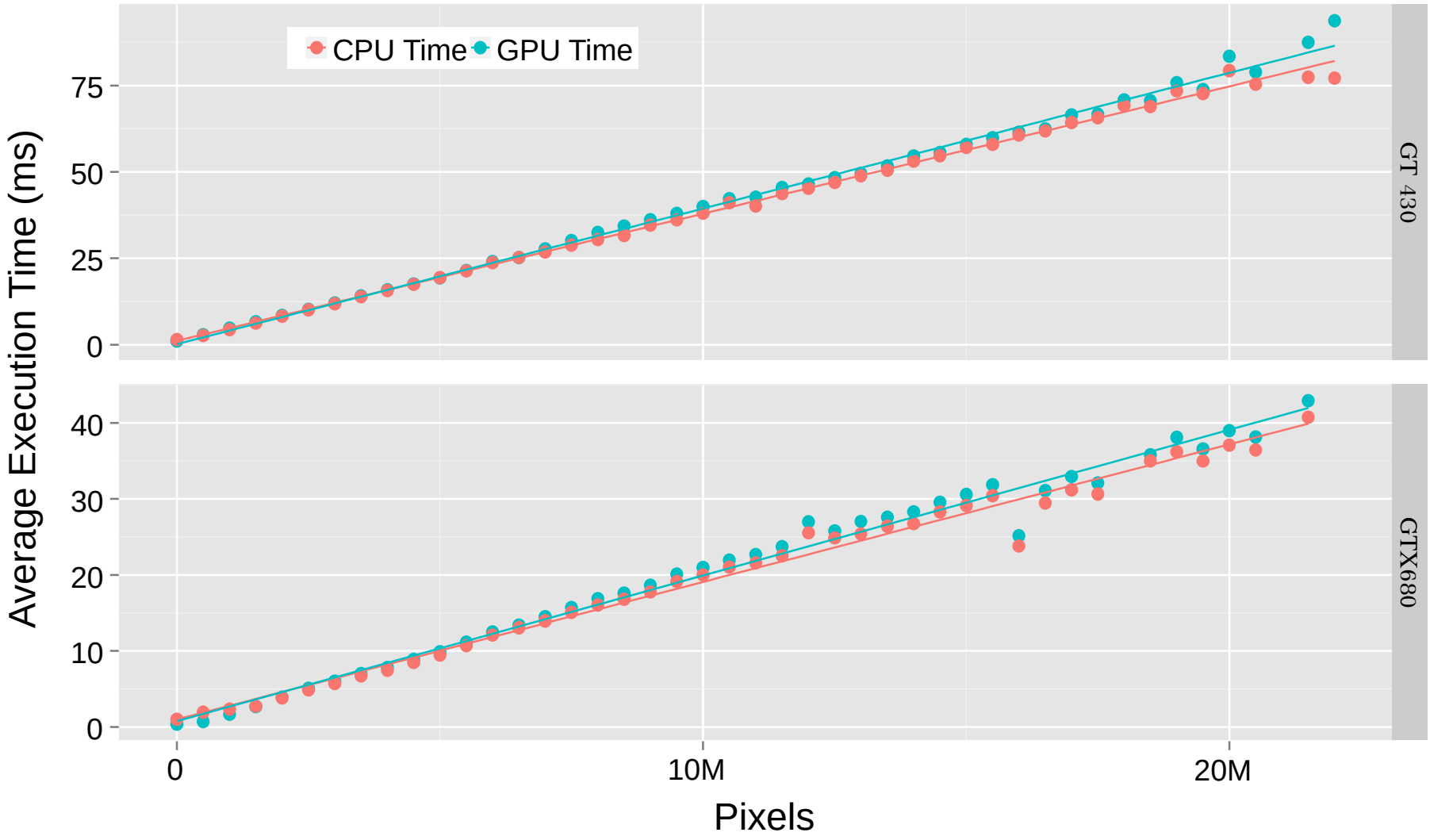


Amdahl's Law, Theoretical Maximum Speedup



Intel–i7 3770k + Nvidia GTX 680

Partitioning Errors



Conclusions

- JPEG decoding contains high amount of massive data computation
- We proposed JPEG decoding scheme for heterogeneous architectures
  - Performance model using polynomial regression
  - Dynamic partitioning scheme
  - Up to 4.2x (2.5x average) speedup over the SIMD version of libjpeg-turbo
  - Guaranteed improvement regardless of CPU+GPU combinations
  - Workload is well distributed across CPU and GPU
- Our combined partitioning scheme achieves up to 95% of the theoretically attainable speedup, with an average of 88%
- Future work
  - Extension to mobile systems

Q&A

Thank you

Backup Slides

Related Work

Parallel JPEG Decoding

[1] Parallel image processing based on CUDA [Yang '08]

[2] Design, implementation and evaluation of a task-parallel JPEG decoder for the libjpeg-turbo library [Hong '12]

[3] GPUJPEG: JPEG compression and decompression accelerated on GPU

Heterogeneous Computing

[1] Architectural exploration of heterogeneous multiprocessor systems for JPEG [Shee '08]

[2] MapReduce on a coupled CPU-GPU architecture [Chen '12]

[3] Qilin: exploiting parallelism on heterogeneous multiprocessors with adaptive mapping [Luk '09]

[4] Cooperative heterogeneous computing for parallel processing on CPU/GPU hybrids [Lee '12]

[5] CAP: co-scheduling based on asymptotic profiling in CPU+GPU hybrid systems [Wang '13]

GPU Results

- An 2048x2048 image