Applying Parallel Design Techniques to Template Matching with GPUs

Robert Finis Anderson, J. Steven Kirtzic, Ovidiu Daescu

Department of Computer Science
Jonsson School of Engineering
University of Texas at Dallas

VECPAR 2010 Conference on High Performance Computing

UTD FEARLESS engineering

Outline

Current State of Image Processing on GPUs

Ideally Suited

- Massively Parallel
- GPU Structures (texture, etc) directly applicable
- $\leq$ 32 bit precision ok!
- OpenCV announced work on IP on the GPU at CVPR 2010

Widespread

- Basically all discrete NVIDIA cards now support, some embedded
- 100's of cores can be had for $\leq$ 100 dollars US.

Current State of Image Processing on GPUs

Underutilized

- Few algorithms adapted
  - FFT in NVIDIA Computing SDK, and others
  - Stereo Registration at MIT IAP 09 and CVPR 2010
    - Gary Bradski of OpenCV showing live HD stereo registration
  - A little optical flow work at IAP 09
- Often just sub-steps (ie Di Stefano 2005)
- Typically don't account for architectural differences

Current State of Image Processing on GPUs

Current Approach

- Existing Algorithm
  - Designed for SISD or MIMD execution
- Note Some Portions are Parallel
  - Independent
  - Repetitive
- Cram those portions into GPU architecture
- Does it improve??
- Optimize
- Repeat

Current State of Algorithm Design for GPUs

Current Image Processing Algs- Critique

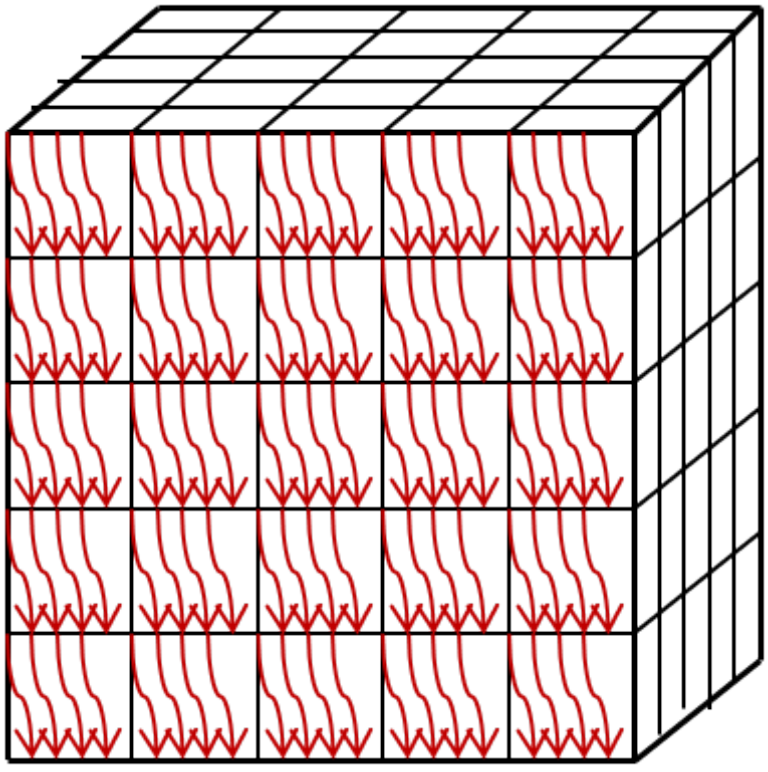
- Ad-hoc
- Doesn't consider differences in architecture
- Ignores translation costs during design
- Ignore important components of distributed type computation, such as inter-unit overhead
- No clear analysis of what makes a good GPU algorithm

Computing Architecture Overview

SIMD

- Single Instruction Multiple Data
- Architecture of super-computers of years past
- Known as Data-parallelism
- Has evolved into the current GPGPU processors
  - More graceful handling of divergence
  - Known as SIMT (Single Instruction Mutltiple Thread)

Computing Architecture Overview



SIMT Thread Structure

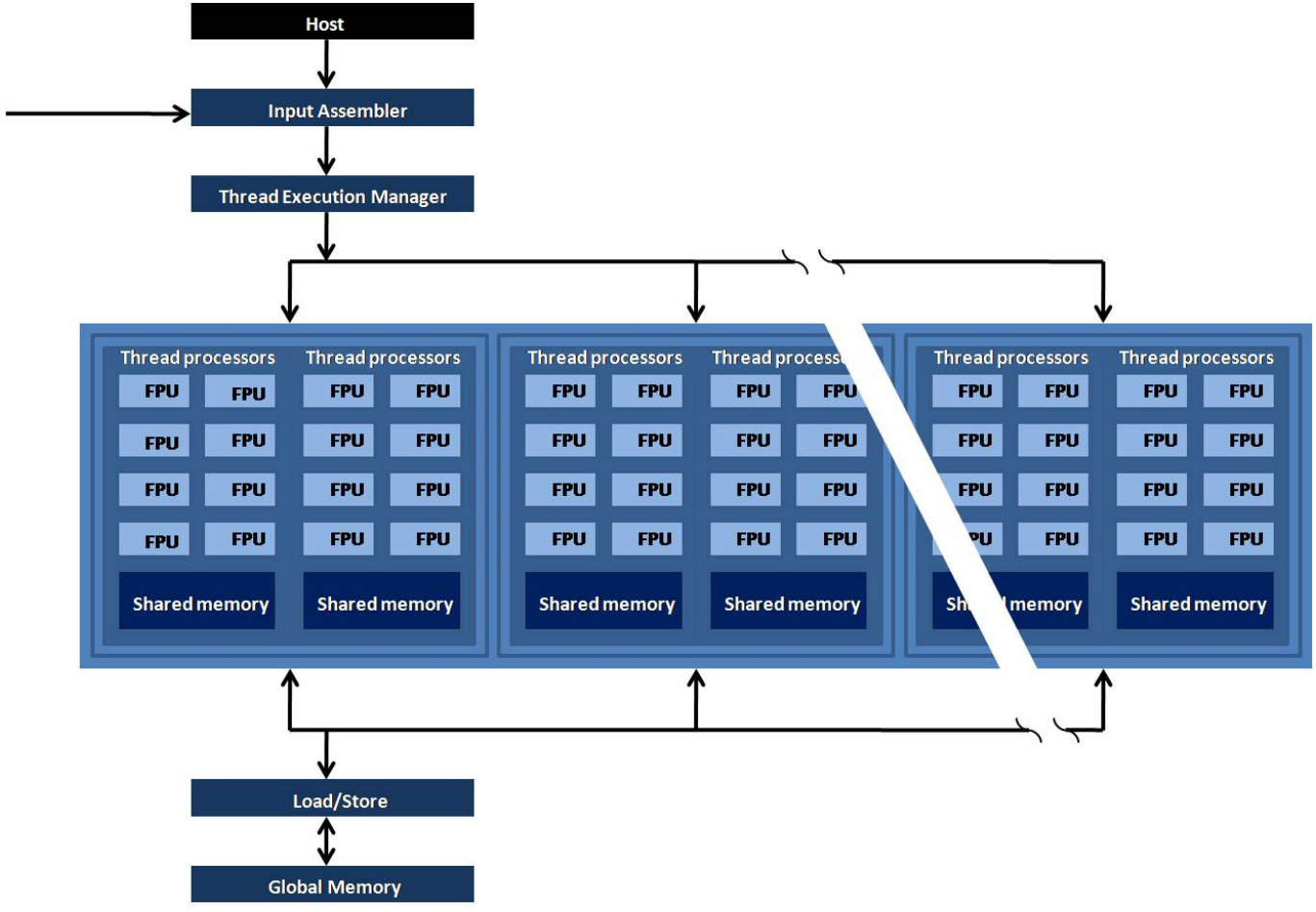
Data Parallelism

- Different way of looking at the problem
- Problems are divided by output data, rather than input
  - What input is required to get this part of the answer
  - Instead of how do I get from here to the answer
- Common to program in ‘kernels’

Memory Architecture Overview

- Multi-Threading
  - Shared memory addressing space
  - Synchronization, consistency, and liveness complex
- Multi-Processing
  - Separate memory addressing space
  - Memory is physically shared
  - Consistency is somewhat simplified, but synchronization and liveness can still be complex

Memory Architecture Overview



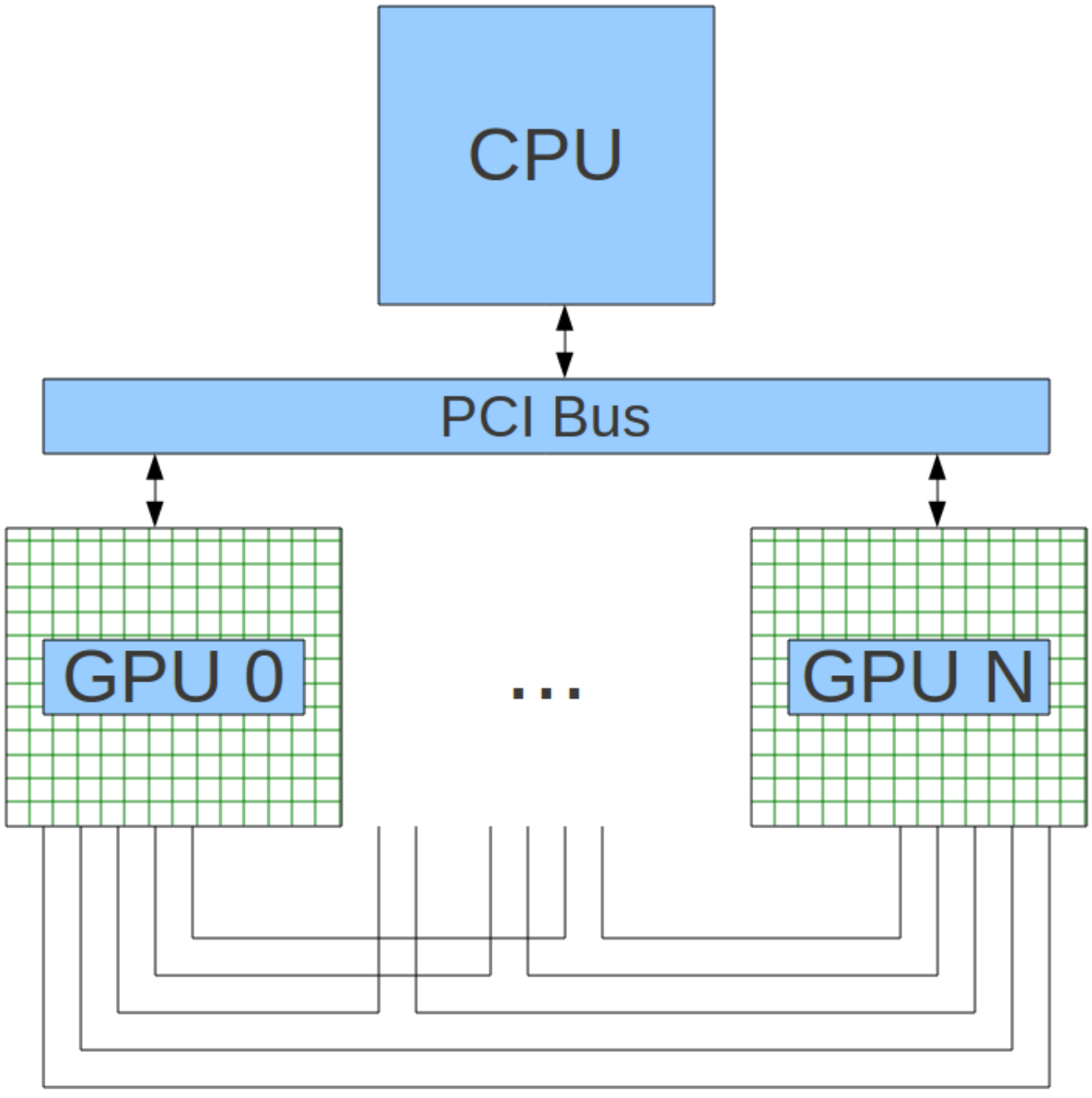
Memory Architecture Overview Cont'd

- GPU
  - Semi-shared memory space
  - Some memory is physically shared, some is available only to sub-sets of processing units
  - Programming model eliminates most synchronization, consistency, and liveness issues.

Memory and Communications

- CPU memory not shared with GPU
  - Nor are other host resources
- Basically a separate computer living inside the host
- Synchronization and consistency between GPU and host considered difficult
  - typically avoided
- But...
- Can be viewed as a distributed computation on a network

Communications Architecture

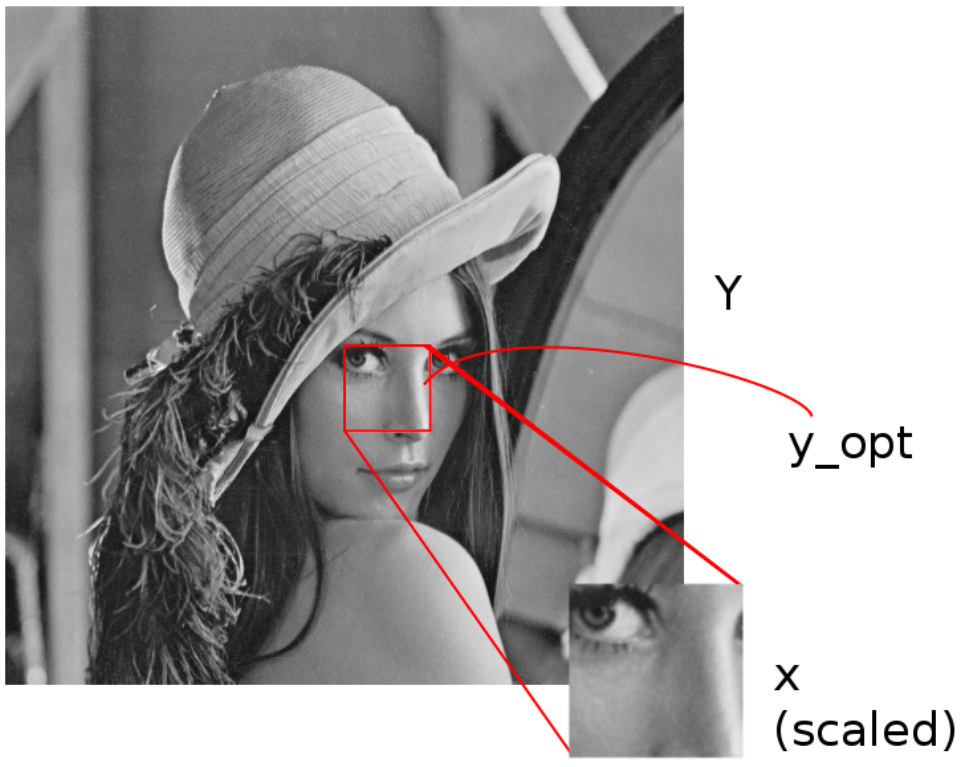


Our Work

- Application of insights and principles above to Template Matching
  - Classic Machine Vision problem
  - Simplistic method of finding similar images
  - Used in video compression, face detection, handwriting recognition, and motion estimation
- Achieved considerable speedups over existing GPU algorithms
- 40x!!!

Template Matching: Overview

An Illustration



Techniques Applied

- Start with result, go backwards from there
  - Single result, from entire image
    - Some kind of Reduction Tree needed
    - Min val
  - Need the values to calculate the min from
    - Divide task
  - Not necessary to compute all
    - Move in rounds

Techniques Applied

- Design solution around memory
  - Read only, fast, shared, not very limited ($\approx$ 100MB)
    - Input data
  - Read/write, slow, shared, not very limited (same)
    - Avoid, used for inter-core communication during reduction
  - Read/write, fast, semi-shared, limited in size ($\approx$ 10KB per processor block)
    - Operate on data here
    - Synced back to slow shared read/write
  - Read/write, very fast, local, very limited ($\approx$ 200B)
    - Processor working space
    - Partly registers- the fewer that are used, the more threads that can run concurrently

Techniques Applied
Input Segmentation

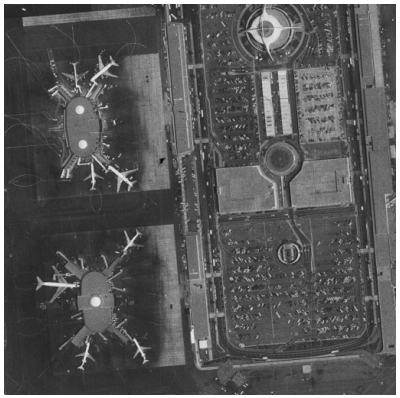
- Each overlapping subwindow of y is a potential match
- Overlap size is basically $(x_{width} - 1) * (x_{height} - 1)$
- Makes the semi-shared memory only somewhat useful
- We made use of on-card Read Only Memory to speed access
- Meta-information (such as likelihood of match) is divided by output location

- Match information calculated at all possible locations
  - Results stored in very fast, semi-shared memory
- How to find minimum?
  - Global minimum found through *Reduce*
  - Each thread compares itself to its neighbor
  - Next iteration, compares itself to neighbor twice as far away
  - 'In between' threads deactivated to conserve processing power
- 15X speedup!!!

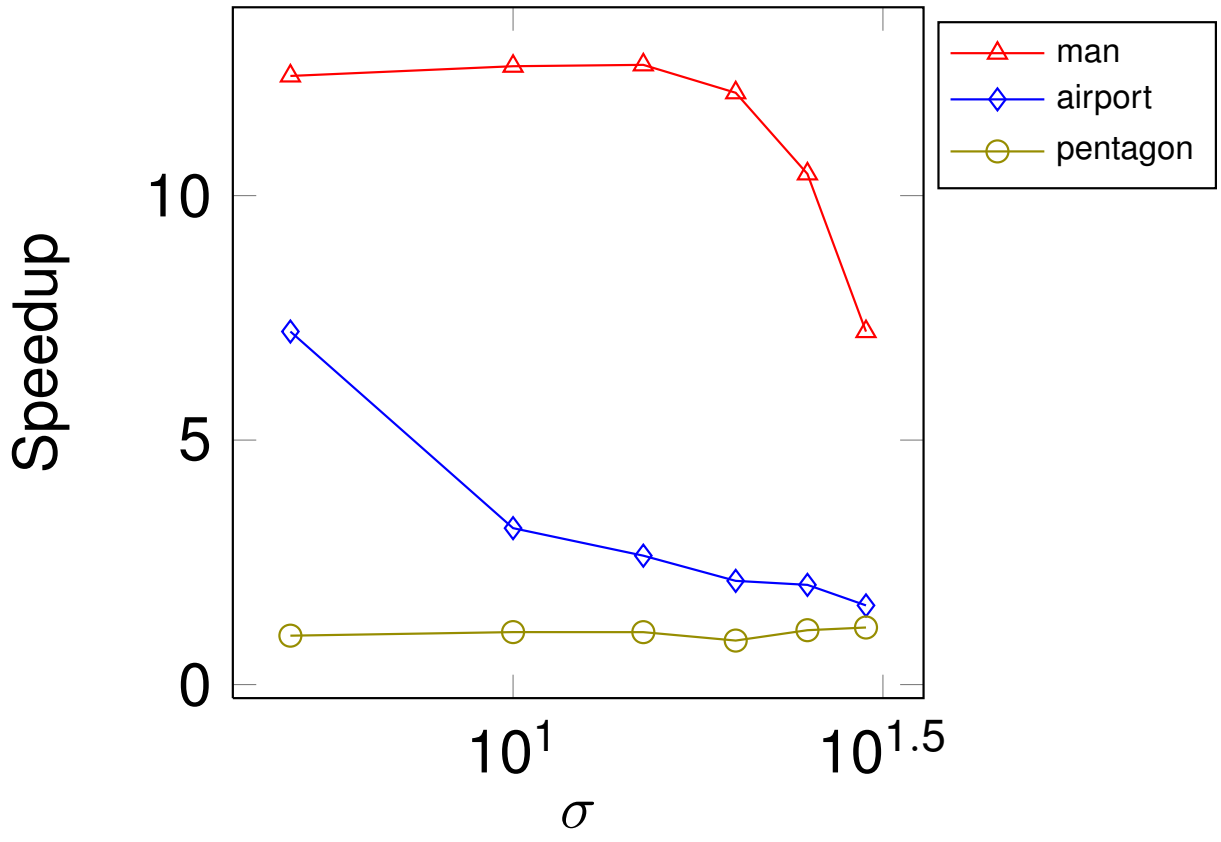
Techniques Applied
Memory Bottleneck

- Host-GPU link is slow, synchronization expensive
  - Remember network model
  - Inter-GPU links are much faster
- Host pre-processing used to reduce memory transfer
- Additional 5x speedup on average, up to 39x!!!
- Total of over 8000x over pure host full-search naive implementation

Test Images



The Numbers



Runtime ratio of our algorithm compared to a normally optimized algorithm not making use of memory preprocessing

The Numbers

Image	Noise	Parallel	Full Search	Improvement
second	0	3979.879	27930.175	7.018
rob ref	0	6123.839	237215.515	38.736

Runtime ratio and absolute runtime (in ms) of our algorithm compared to the normally optimized algorithm on large ($7 \cdot 10^6$ pixel) images

Future Work

- Currently, only metadata is segmented
  - Compute minimal rectangular covering of areas of interest
  - NP Complete
  - Approximate, transfer only portions of image
- ‘Interesting’ portions of image tend to be clustered
- Direct mapping results in uneven balancing
- Full load balancing more complex than we need
  - Very fast custom hashing algorithm to distribute across GPU

Questions

- Questions??

For Further Reading I

[1] Robert Finis Anderson, J. Steven Kirtzic, and Ovidiu Daescu

Applying Parallel Design Techniques to Template Matching with GPUs

VECPAR 2010 2010

[2] J. Nickolls, I. Buck, M. Garland, and K. Skadron

Scalable Parallel Programming with CUDA

ACM's Queue vol. 6 2008

[3] L. Di Stefano, S. Mattoccia and F. Tombari

Speeding-up NCC-based Template Matching Using Parallel Multimedia Instructions

CAMP 2005 2005