

High Performance Computing for Audio-Visual Signal Processing

Ramani Duraiswami
 Perceptual Interfaces and Reality Lab.
 Computer Science & UMIACS
 University of Maryland, College Park, MD

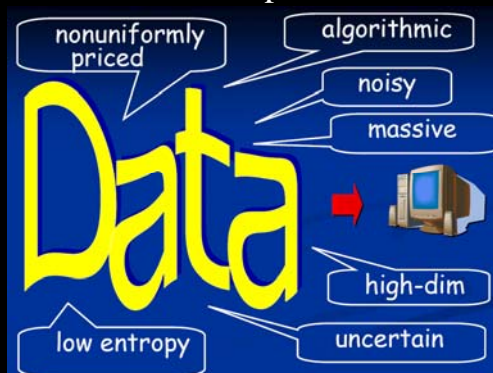
<http://www.umiacs.umd.edu/~ramani>

ramani@umiacs.umd.edu

Ever larger collections of audio visual data

- Sensors are getting varied and cheaper
 - Cameras, microphones
 - Camera megapixels double every 18 months
- Storage is getting cheaper
- Other Large data
 - Text (all the newspapers, books, technical papers)
 - Genome data
 - Medical/biological data (X-Ray, PET, MRI, Ultrasound, Electron microscopy ...)
 - Climate (Temperature, Salinity, Pressure, Wind, Oxygen content, ...)

The data needs to be processed!



Ways to attack problem size growth

- Faster processors
 - “Moore’s law will take care of it”
- Faster algorithms with better asymptotic complexity
- Go parallel!
- Clusters of computers
 - Way Google and Yahoo do it
 - Hadoop and MapReduce
 - Coarse grained, task based parallelism
- New shared memory multiprocessor chips (multicore processors, GPUs)
 - Data parallelism

Actually Moore’s law doesn’t help

- Argument:
 - Moore’s law: Processor speed (used to) double every 18 months
 - If we wait long enough the computer will get fast enough and let my inefficient algorithm tackle the problem
- Is this true?
 - Yes for algorithms with linear asymptotic complexity
 - No!! For algorithms with different asymptotic complexity
 - Most scientific algorithms are $O(N^2)$ or $O(N^3)$
 - For a million variables, we would need about 16 generations of Moore’s law before a $O(N^2)$ algorithm was comparable with a $O(N)$ algorithm
- Did no one tell you that Moore’s law is dead?

Moore’s Law is dead!

- Feature sizes and clock speeds on commodity chips have been stagnant over the past 4 years
 - ~3 GHz and 45 nm
 - Smaller or faster not possible because of Physics!
- All manufacturers are going with multicore to maintain performance
 - Core-2, core-2-duo, quad-core, ...
- Shared memory multiprocessing
 - Intel has demo’ed several many core systems
- Graphics processors and gaming consoles have already been on the multicore path for a decade!


Gamer Power





Sony Playstation 3
2.18 teraflops <\$300
Difficult to program



Microsoft X-Box 360
1.04 teraflops <\$200
Difficult to program







Multicore Intel box with 3 GPUs in Slots
~ 1 Teraflop for < 3000 (shown with 1 GPU)

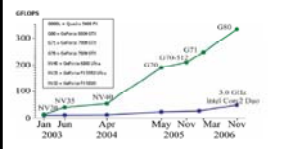


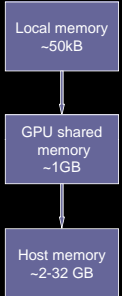
Figure 1-1. Floating-Point Operations per Second for the CPU and GPU

Current NVIDIA GPU

- 240 scalar processor cores per GPU
- 933 GFlops Single Prec and 78 GFlops Double Prec
- Hardware Thread Execution Manager enables thousands of concurrent threads per GPU
- 1.5 GB memory per GPU
- Upto 4 GPUs per CPU node
 - More would saturate the bus

Programming on the GPU

- GPU organized as groups of multiprocessors
- Each multiproc. has 8 relatively slow processors
- Each MP has small amount of own memory and access to GPU global memory
- Factor of 100s difference in speed as one goes up the memory hierarchy
- Single Program Multiple Data Framework
- Many practically important tasks do map well and we are working on converting others
 - Image and Audio Processing
 - Some types of linear algebra cores
 - Many machine learning algorithms
 - Physics algorithms



GPU work in my lab.

- Signal Processing
 - Audio processing (A. O Donovan, CVPR'07, ICASSP 08)
 - Image processing (Yuancheng Luo, WACV'08/CVGPU'08, JPDC)
 - Machine Learning (Balaji Vasan)
- Fundamental scientific computing algorithms
 - Fast multipole methods on GPUs ($10^6 \times 10^6$ dense matrix vector product in 1s on the 8800 GTX), J. Comput. Phys. Sep 2008 (with N. Gumerov)
 - Simulation of plasma turbulence (with Bill Dorland and NG, JPDC Sep 2008)
- Flagon: Programming environment/library
 - Provides objects, operators, and algorithms for using the GPU from C, Fortran-90 and Matlab
 - without knowing CUDA (Supercomputing 2007) (NG, RD, BD, YL, Kate Despain)

GPU programming issues

- Cost of global memory access is ~ 500 clock cycles
- access to local data is free (~ 1-4 clock cycles)
- **Lesson 1: Compute the heck out of the data each time you load it in to local memory**
 - Computing features on image, voice data
 - Block based linear algebra algorithms
- Coalesced reads and writes: When contiguous data is accessed from global memory, cost of access to subsequent pieces is only 4 clock cycles
- **Lesson 2: Use data structures to coalesce access**

GPU programming 2

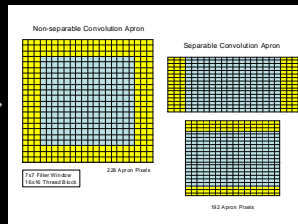
- Many models of parallelism considered in 80s and 90s
- GPU: closest to data parallelism
 - Identify a basic set of primitive algorithms
- Build a library of efficient procedures
 - Blelloch's prefix scan algorithm
 - Sorts, reductions
- Block based higher level linear algebra algorithms
 - LU, QR,
- Lesson 3: Build a library of useful functions
- Lesson 4: Build tools and model for programming the GPU from high level languages

Algorithms for video/audio processing

- Filters and convolution
 - Feature extraction based on filter output
- Morphological operations
 - Involve branching and logic, and can be slow
- Tracking
 - Computing similarities, geometric transforms
- Segmentation
 - Classification based, Graph cuts,
- Machine Learning
 - Classification, Regression, Ranking
 - Linear algebra and optimization on features/cost functions

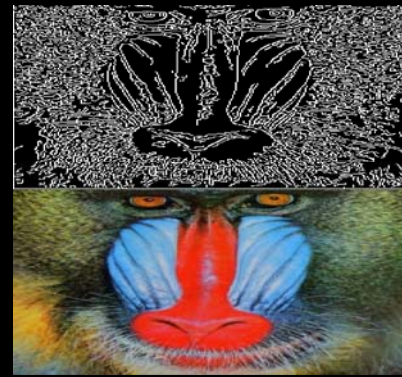
Feature computation, data smoothing etc

- Ideal for the GPU
- Several implementations widely available
 - Sobel, Gaussian convolution, Laplacian of Gaussian etc.
 - Gradients, Harris operator
- Divide images into possibly overlapping patches
 - Compute
 - Fix stuff at borders
 - Some tuning and data structures can help
 - E.g. for memory access at borders



- Other algorithms that port well
- Sorts
 - Fourier transforms
 - Matrix vector/Matrix Matrix products

Image Processing :Canny Edge detection on NVIDIA CUDA



Comparison with OpenCV

- CPU: Intel Core2 CPU 6600 @ 2.40 GHz, 2 GB RAM
- GPU: NVIDIA GeForce 8800 GTX, 768 MB global memory (not overclocked)

Image Size	CUDA (ms)	OpenCV	Speedup
256x256	1.82	3.06	1.681
512x512	3.40	8.28	2.435294
1024x1024	10.92	28.92	2.648352
2048x2048	31.46	105.55	3.353465
3936x3936	96.62	374.36	3.874560

LENA

MANDRILL

Matlab

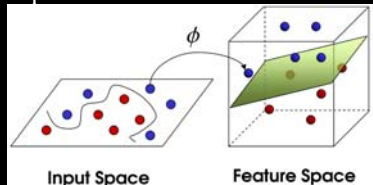
- CUDA interoperability with Matlab via compiled mex files (SDK example)
- Adapted existing code to use
- `glCudaEdgeDetector(A,nChannels,thresholdLow,thresholdHigh,hysteresisIts,sigma);`

$N \times M$	MC (ms)	M (ms)	$N \times M$	MC (ms)	M (ms)
256x256	4.7	102	240x256	4.4	103
512x512	8.6	408	496x512	8.6	429
1024x1024	22.7	1706	992x1024	24.9	1827
2048x2048	75.4	6726	1968x2048	92.1	7065
3936x3936	227.7	26020	3936x4096	309	28388

- More substantial speedups

Learning: Classification and Regression

- Regression/Classification: Given lots of data (x,y)
- $x \in \mathbb{R}^d$;
- given a new “target” point x_* predict y there
- Regression if y is real valued
- Classification if y is categorical
- Kernel methods are powerful here as well
- “Lift problem to infinite dimensional space where separation is linear”



Kernel Methods

- Key programming primitives
 - Computation of kernel sums (Matrix vector products)
 - Solution of linear systems
 - Evaluation of cost functions
- All can be mapped well to GPU
- Cost of matrix vector product is $O(N^2)$
- Algorithms to speed this up to $O(N)$
 - First for CPUs
 - Now for GPUs

Fast Multipole Methods for GPUs

- General method for accelerating large classes of dense matrix vector products
- Reduce computational/memory complexity from $O(N^2)$ to $O(N)$
- Allow reduction of $O(N^2)$ and $O(N^3)$ operations to linear order
- We are applying it to many areas
 - Fast statistics, similarity measures, image processing, segmentation, tracking, learning
 - Non uniform fast Fourier transforms and reconstruction
 - Elastic registration, fitting thin-plate splines
 - Acoustics, Synthetic beamforming
 - Fluid mechanics (vortex methods, potential flow, Stokes flow)
 - Electromagnetic scattering and Maxwell's equations

Results (Gumerov & Duraiswami, J Comp. Phys, 2008)

- 30 fold improvement over FMM on CPU
- 30*N improvement over serial algorithm on CPU

$N=1,048,576$ (potential only)

	serial CPU	GPU	Ratio
p=4	22.25 s	0.683 s	33
p=8	51.17 s	0.908 s	56
p=12	66.56 s	1.395 s	48

$N=1,048,576$ (potential+forces (gradient))

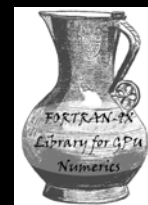
	serial CPU	GPU	Ratio
p=4	28.37 s	0.979 s	29
p=8	88.09 s	1.227 s	72
p=12	116.1 s	1.761 s	66

GPU Programming is now a lot easier

- **but is still quite hard!**
- Legacy code is on the CPU whereas success using CUDA is mostly with shipping entire programming to GPU
 - this limits size of problems to that which can be held in one “box”
 - Will not yet fit extremely large problems
 - Will do problems that fit in ~ 1 GB and can be parallelized
- Unless programming model and memory issues are understood, performance not achieved

Approach to use GPU: Flagon Middleware

- Programming from higher language on CPU (Matlab/C++/Fortran)
- Pointers to Device Variables on the GPU
- Execute small, well written, CU functions to perform primitive operations on device
 - avoid data transfer overhead
- Provide wrappers to BLAS, FFT, and other software (random number, sort, screen dump, etc.)
- Allow incorporation of existing mechanisms for doing distributed programming (OpenMP, MPI, Hadoop, etc.) to handle clusters



Sourceforge Flagon

Processing large video streams

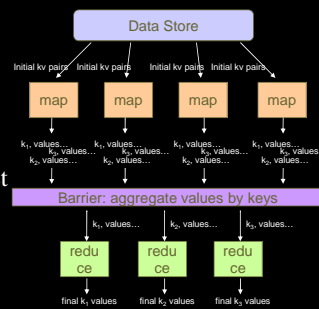
- Will need to combine both cluster and GPU paradigms
- At the point video is enrolled in to database we will need to reduce it to features
 - Features, objects, actions, people, words
- Need data parallel algorithms for these
- Use it with large search algorithms from Google, Amazon, Yahoo etc.
 - Map Reduce/Hadoop
 - Parallel file systems

Hadoop/Map Reduce

- Google, Yahoo and Amazon have built huge search, and transaction engines on extremely large clusters of computers
- Also built a distributed file system for handling extremely large data sets
- Map/reduce paradigm works on files which are extremely large sets of (key, value) pairs
- Map step takes data from the store according to some rule on the keys
- Reduce performs same computation on data according to key.

Map Reduce

- Divide and Conquer!
- Extremely fertile area of research
- However not much yet on analysis of video and audio
- Reduce can be computationally intensive and use GPUs as an accelerator



Picture from Jimmy Lin's course notes

Audio processing on the GPU

- Recently we developed a novel spherical microphone array
 - Audio Camera: Invention of the year, 2008
- Allows panning of the environment and measure the intensity of sound coming from various directions
- Very expensive to process
- Prevents real-time
- Used G80 GPU to convert to real-time
- Demo

Gaussian Process Regression

- Bayesian approach to regression/classification
- Provides both a prediction and a variance estimate
- Mackay (2005), Rasmussen and Williams (2006)

