Signal Reconstruction Algorithms on Graphical Processing Units

Sangkyun Lee[†] and Stephen J. Wright

[†]sklee@cs.wisc.edu

Computer Sciences Department, University of Wisconsin-Madison

Oct. 11, 2009

1 Signal Reconstruction Problems

2 Graphical Processing Units (GPUs)
- General Computation on GPUs

3 GPU Implementations of Algorithms
- Compressive Sensing
- Image Reconstruction

4 Numerical Results
- Speedup of SpaRSA
- Speedup of PDHG

5 Conclusion

Problems of Interest

Want to find regularized solutions of systems of linear equations

$$\min_{x \in X} \ \frac{\lambda}{2} ||y - Ax||_2^2 + r(x),$$

where X is a closed convex set, y is an observation, A is a linear operator, and $r(x)$ is a regularizer ($\lambda > 0$).

We focus on two specific instances, compressive sensing and image reconstruction.

$$\min_{x \in X} \frac{\lambda}{2} ||y - Ax||_2^2 + r(x).$$

Compressive Sensing (CS)

- $x \in X = \mathbb{R}^n$ is sparse; at most S nonzero components.

Image Reconstruction (IR)

- $X \subset \mathbb{R}^{n \times n}$ is the set of pixelated images with BV.

$$\min_{x \in X} \frac{\lambda}{2} ||y - Ax||_2^2 + r(x).$$

Compressive Sensing (CS)

- $x \in X = \mathbb{R}^n$ is sparse; at most S nonzero components.
- $A \in \mathbb{R}^{m \times n}$ is dense, $m < n$.

Image Reconstruction (IR)

- $X \subset \mathbb{R}^{n \times n}$ is the set of pixelated images with BV.
- $A \in \mathbb{R}^{n \times n}$ is dense in general.

$$\min_{x \in X} \frac{\lambda}{2} ||y - Ax||_2^2 + r(x).$$

Compressive Sensing (CS)

- $x \in X = \mathbb{R}^n$ is sparse; at most S nonzero components.
- $A \in \mathbb{R}^{m \times n}$ is dense, $m < n$.
- $y \in \mathbb{R}^m$ contains noisy observations, $y = Ax + z$.

Image Reconstruction (IR)

- $X \subset \mathbb{R}^{n \times n}$ is the set of pixelated images with BV.
- $A \in \mathbb{R}^{n \times n}$ is dense in general.
- $y \in \mathbb{R}^{n \times n}$ is a distorted image, $y = Ax + z$.

$$\min_{x \in X} \frac{\lambda}{2} ||y - Ax||_2^2 + r(x).$$

Compressive Sensing (CS)

- $x \in X = \mathbb{R}^n$ is sparse; at most S nonzero components.
- $A \in \mathbb{R}^{m \times n}$ is dense, $m < n$.
- $y \in \mathbb{R}^m$ contains noisy observations, $y = Ax + z$.
- $r(x) = ||x||_1$.

Image Reconstruction (IR)

- $X \subset \mathbb{R}^{n \times n}$ is the set of pixelated images with BV.
- $A \in \mathbb{R}^{n \times n}$ is dense in general.
- $y \in \mathbb{R}^{n \times n}$ is a distorted image, $y = Ax + z$.
- $r(x) = TV(x)$.

$$\min_{x \in X} \frac{\lambda}{2} ||y - Ax||_2^2 + r(x).$$

Compressive Sensing (CS)

- $x \in X = \mathbb{R}^n$ is sparse; at most S nonzero components.
- $A \in \mathbb{R}^{m \times n}$ is dense, $m < n$.
- $y \in \mathbb{R}^m$ contains noisy observations, $y = Ax + z$.
- $r(x) = ||x||_1$.
- A satisfies a property (RIP) which guarantees the exact recovery of the original signal with a very high probability.

Image Reconstruction (IR)

- $X \subset \mathbb{R}^{n \times n}$ is the set of pixelated images with BV.
- $A \in \mathbb{R}^{n \times n}$ is dense in general.
- $y \in \mathbb{R}^{n \times n}$ is a distorted image, $y = Ax + z$.
- $r(x) = TV(x)$.
- $A = I$ (denoising) or A is a linear blur operator (deblurring).

$$\min_{x \in X} \frac{\lambda}{2} ||y - Ax||_2^2 + r(x).$$

Compressive Sensing (CS)

- $x \in X = \mathbb{R}^n$ is sparse; at most S nonzero components.
- $A \in \mathbb{R}^{m \times n}$ is dense, $m < n$.
- $y \in \mathbb{R}^m$ contains noisy observations, $y = Ax + z$.
- $r(x) = ||x||_1$.
- A satisfies a property (RIP) which guarantees the exact recovery of the original signal with a very high probability.
- For certain A (e.g. DCT), we can perform Au or $A^T v$ efficiently without storing A.

Image Reconstruction (IR)

- $X \subset \mathbb{R}^{n \times n}$ is the set of pixelated images with BV.
- $A \in \mathbb{R}^{n \times n}$ is dense in general.
- $y \in \mathbb{R}^{n \times n}$ is a distorted image, $y = Ax + z$.
- $r(x) = TV(x)$.
- $A = I$ (denoising) or A is a linear blur operator (deblurring).
- Can perform Au or $A^T v$ via (de-)convolution.

Calls for Efficient Implementations

- The number of variables can be huge.
  - In CS, we are often interested in the signals with large bandwidth.
  - In IR, nowadays cameras create huge images.
- Time constraints for solving problems.
  - CS for MRI: doctors and patients are waiting for the solutions.
  - IR for computer vision: fast (realtime) processing of streamed images is required.

Graphics Processors as Computation Devices

Graphics adapters have been evolved into massively parallel and programmable computation units, in order to meet needs for realtime graphics and realtime rendering.

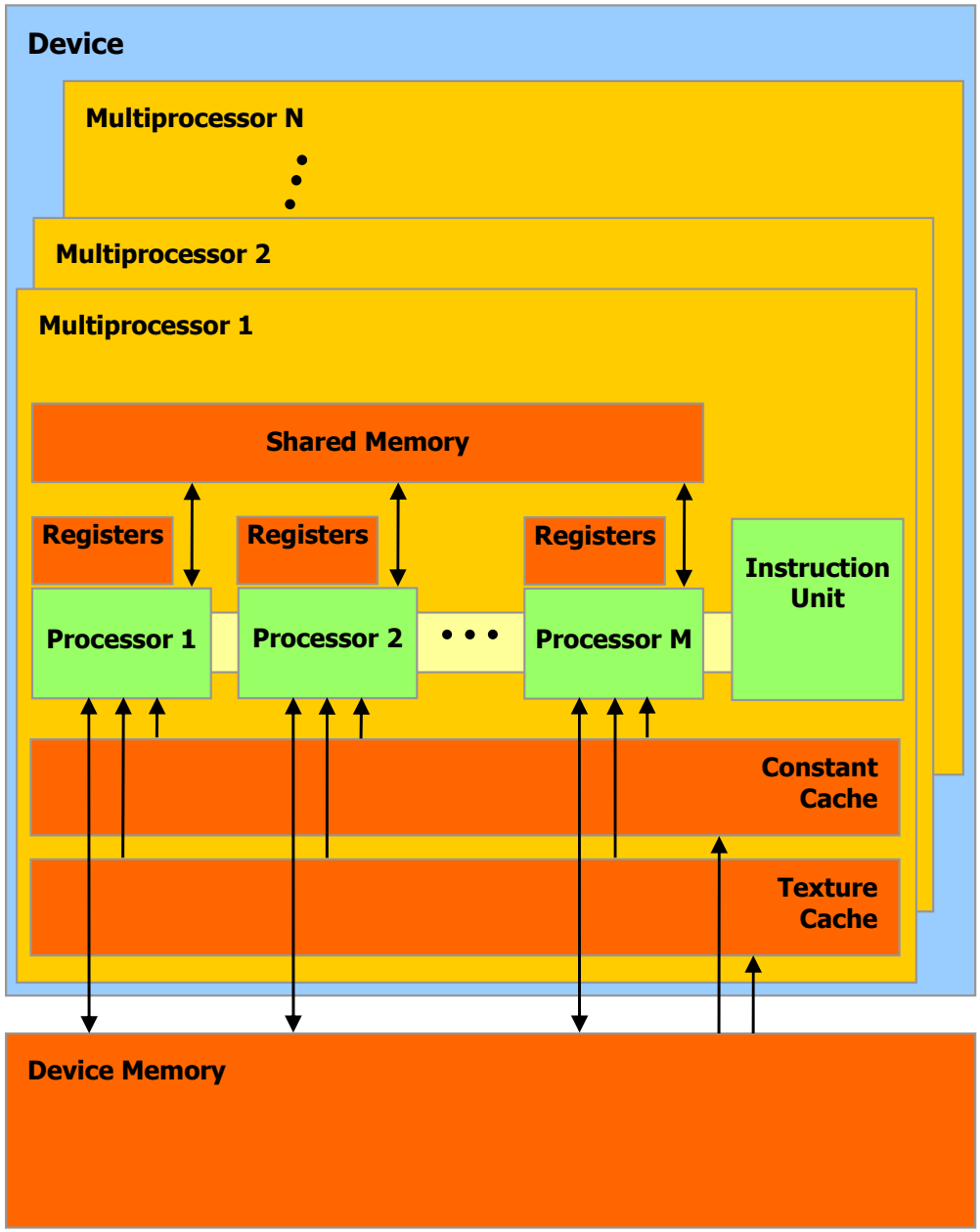
The idea of using GPUs for generic computation goes back to late 70's. But it gets spotlights only recently, as regular PCs (and laptops!) begin to equip powerful GPUs, getting a name GPGPU.

History of GPGPU - General Purpose Computation using GPUs.

- GPGPU using OpenGL API (2000$\sim$).
  - An industrial standard graphics library; not designed for computation.
- GPGPU using vendor-specific softwares (2007$\sim$present).
  - Software depends on a vendor, but shows better performance.
- GPGPU using OpenCL (2009$\sim$present).
  - An open-standard API for GPGPU, driven by Apple.

We consider CUDA (Compute Unified Device Architecture) from NVIDIA, which defines a small extension of the standard C language.

GPU Internals in CUDA



GPU Computing

Pros.

- Easy to parallelize existing algorithms.
  - Rather than splitting the entire logic of algorithms in complicating ways, focusing on parallelizing smaller logical units, e.g. each line of the algorithm.
- Cost effective.
  - GeForce GTX 260 provides 216 cores at $200 ($.93 per core).
  - Intel Core i7-920 CPU provides 4 cores at $280 ($70 per core).
- Pervasive.
  - My laptop has a GPU with 32 cores!.

Cons.

- Limited data transfer bandwidth between host and GPU memory.
  - GPU will be embedded in CPU chips soon.
- Limited availability of GPU memory.
  - Top-edge GPUs have up to 4GB, but smaller in general.

Conditions for Efficient GPU Implementations

- No frequent transfer of data between host and GPU memory.
  - Data transfers only in the beginning and in the end of the algorithm.
- Small memory footprint due to memory limitation.
  - No $O(n^2)$ storage requirements.
  - Choose A matrices in CS and IR which don't have to be explicitly stored.
- Elementary logical units of the algorithm is simple.
  - First-order methods are particularly suitable for creating many small jobs to make all cores in a GPU busy.

SpaRSA Algorithm [Wright and Nowak, 07] for CS

$$\min_{x \in \mathbb{R}^n} \ \frac{1}{2}||y - Ax||_2^2 + \tau||x||_1 = h(x) + \tau||x||_1. \qquad (1)$$

SpaRSA Algorithm [Wright and Nowak, 07] for CS

$$\min_{x \in \mathbb{R}^n} \ \frac{1}{2}||y - Ax||_2^2 + \tau||x||_1 = h(x) + \tau||x||_1. \tag{1}$$

Consider a separable quadratic approximation $\bar{h}(x)$ of the smooth part $h(x)$ at some point x^k (dropping constant term):

$$\bar{h}(x) = \frac{\alpha_k}{2}||x - x^k||_2^2 + \nabla h(x^k)^T(x - x^k). \tag{2}$$

SpaRSA Algorithm [Wright and Nowak, 07] for CS

$$\min_{x \in \mathbb{R}^n} \ \frac{1}{2}||y - Ax||_2^2 + \tau||x||_1 = h(x) + \tau||x||_1. \tag{1}$$

Consider a separable quadratic approximation $\bar{h}(x)$ of the smooth part $h(x)$ at some point x^k (dropping constant term):

$$\bar{h}(x) = \frac{\alpha_k}{2}||x - x^k||_2^2 + \nabla h(x^k)^T(x - x^k). \tag{2}$$

$$x^{k+1} \in \arg\min_x \ \bar{h}(x) + \tau||x||_1. \tag{3}$$

SpaRSA Algorithm [Wright and Nowak, 07] for CS

$$\min_{x \in \mathbb{R}^n} \ \frac{1}{2}||y - Ax||_2^2 + \tau||x||_1 = h(x) + \tau||x||_1. \tag{1}$$

Consider a separable quadratic approximation $\bar{h}(x)$ of the smooth part $h(x)$ at some point x^k (dropping constant term):

$$\bar{h}(x) = \frac{\alpha_k}{2}||x - x^k||_2^2 + \nabla h(x^k)^T(x - x^k). \tag{2}$$

$$x^{k+1} \in \arg\min_x \ \bar{h}(x) + \tau||x||_1. \tag{3}$$

Replacing x^k with $u^k := x^k - \nabla h(x^k)/\alpha_k$,

$$\begin{aligned} \bar{h}(x) &= \frac{\alpha_k}{2}||x - \left(u^k + \nabla h(x^k)/\alpha_k\right)||_2^2 + \nabla h(x^k)^T\{x - \left(u^k + \nabla h(x^k)/\alpha_k\right)\} \\ &= \frac{\alpha_k}{2}||x - u_k||^2 \cancel{-\nabla h(x^k)^T(x - u^k)} + \cancel{\nabla h(x^k)^T(x - u^k)} + (const.) \end{aligned} \tag{4}$$

SpaRSA Algorithm [Wright and Nowak, 07] for CS

$$\min_{x \in \mathbb{R}^n} \ \frac{1}{2}||y - Ax||_2^2 + \tau||x||_1 = h(x) + \tau||x||_1. \tag{1}$$

Consider a separable quadratic approximation $\bar{h}(x)$ of the smooth part $h(x)$ at some point x^k (dropping constant term):

$$\bar{h}(x) = \frac{\alpha_k}{2}||x - x^k||_2^2 + \nabla h(x^k)^T(x - x^k). \tag{2}$$

$$x^{k+1} \in \arg\min_x \ \bar{h}(x) + \tau||x||_1. \tag{3}$$

Replacing x^k with $u^k := x^k - \nabla h(x^k)/\alpha_k$,

$$\begin{aligned} \bar{h}(x) &= \frac{\alpha_k}{2}||x - \left(u^k + \nabla h(x^k)/\alpha_k\right)||_2^2 + \nabla h(x^k)^T\{x - \left(u^k + \nabla h(x^k)/\alpha_k\right)\} \\ &= \frac{\alpha_k}{2}||x - u_k||^2 \cancel{-\nabla h(x^k)^T(x - u^k)} + \cancel{\nabla h(x^k)^T(x - u^k)} + (const.) \end{aligned} \tag{4}$$

Then

$$x_i^{k+1} = \arg\min_x \ \left\{\frac{1}{2}(x_i - u_i^i)^2 + \frac{\tau}{\alpha_k}|x_i|\right\} = sign(u_i^k) \cdot \max\left\{|u_i^k| - \frac{\tau}{\alpha_k}, 0\right\}. \tag{5}$$

SpaRSA Algorithm (cont')

1: $k \leftarrow 0$
2: Choose intial x^0.
3: **repeat**
4: choose α_k.
5: **repeat**
6: $x^{k+1} \leftarrow$ `solution of sub-problem`.
7: Adjust α_k.
8: **until** x^{k+1} satisfies an acceptance criterion.
9: $k \leftarrow k + 1$.
10: **until** stopping criterion is satisfied.

Choice of α_k

We choose α_k so that $\alpha_k \mathrm{I}$ mimics the true Hessian $\nabla^2 h(x)$ over the most recent two steps:

$$\begin{aligned} \alpha_k &= \arg\min_{\alpha} \; ||\alpha_k \mathrm{I}(x^k - x^{k-1}) - (\nabla h(x^k) - \nabla(h(x^{k-1})))||_2^2 \\ &= \frac{(s^k)^T r^k}{(s^k)^T (s^k)} \end{aligned} \tag{6}$$

where $s^k = x^k - x^{k-1}$ and $r^k = \nabla h(x^k) - \nabla h(x^{k-1})$. This choice of α is inspired by [Barzilai and Borwein 88].

Numerical Results

Algorithm	CPU time (secs.)	MSE
SpaRSA	0.44	2.42e-3
SpaRSA-monotone	0.45	2.49e-3
GPSR-BB	0.55	2.81e-3
GPSR-Basic	0.69	2.59e-3
l1_ls	6.56	2.51e-3
IST	2.76	2.51e-3

Figure: Compressive sensing with a random sensing matrix A of dimension $2^{10} \times 2^{12}$ and 160 spikes, and with Gaussian noise with variance 10^{-4}.

Image Reconstruction

We observe a distorted image y of $u \in \Omega \in \mathbb{R}^2$, where Ω is a image domain with bounded variation, via a transform:

$$y = Au + z \tag{7}$$

We obtain an error-free image by solving the following problem introduced by Rudin, Osher and Fatemi '92:

ROF Model

$$\min_{u \in \Omega} \int_{\Omega} \frac{\lambda}{2} ||y - Au||_2^2 + TV(u), \; TV(u) := |\nabla u|_2. \tag{8}$$

$TV(u)$ is referred as the total-variation semi-norm, which is suitable for penalizing fine distortions but preserving edges.

Image Denoising ($A = \mathrm{I}$)

As Ω has bounded variation,

$$\int_\Omega |\nabla u| = \max_{||w||_2 \leq 1} \int_\Omega \nabla u \cdot w = \max_{||w||_2 \leq 1} \int_\Omega -u \nabla \cdot w \qquad (9)$$

So the problem can be rewritten as:

$$\min_{u \in \Omega} \max_{||w||_2 \leq 1} \ell(x, w) := \int_\Omega -u \nabla \cdot w + \frac{\lambda}{2} ||y - u||^2. \qquad (10)$$

The saddle-point is attained.

Primal-Dual Hybrid Gradient Projection Algorithm (PDHG) [Zhu and Chan '08]

1: Initial x^0, w^0.
2: $k = 0$.
3: **repeat**
4: Update the primal and dual variables:

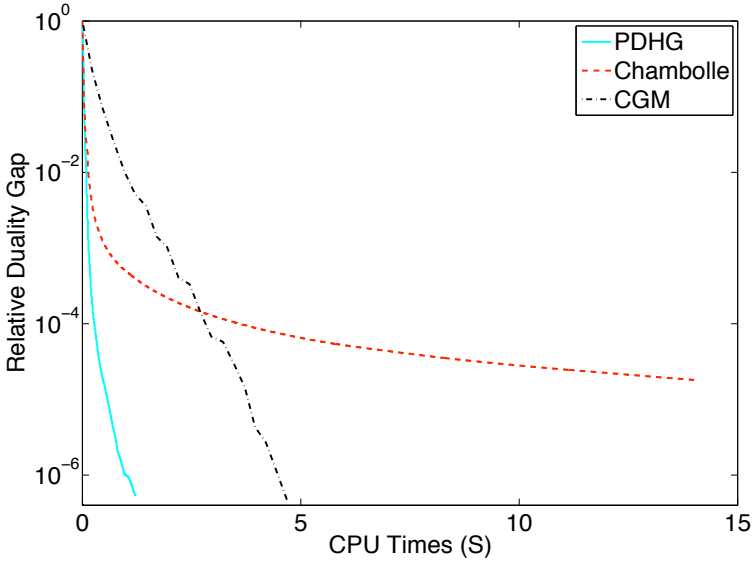
$$\begin{aligned} w^{k+1} &= P_{\{w:||w||_2 \leq 1\}}(w^k + \tau_k \nabla_w \ell(x^k, w^k)) \\ x^{k+1} &= x^k - \sigma_k \nabla_x \ell(x^k, w^{k+1}). \end{aligned} \tag{11}$$

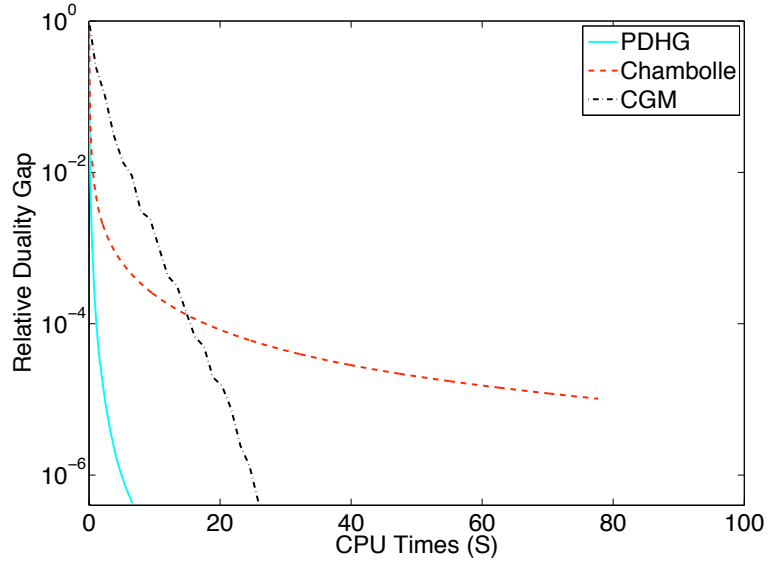
5: $k \leftarrow k + 1$.
6: **until** Duality gap falls below a threshold.

Steplength:

$$\tau_k := (.2 + .8k)\lambda \qquad \sigma_k := (.5 - 1/(3 + .2k))/\tau_k. \tag{12}$$

Numerical Results





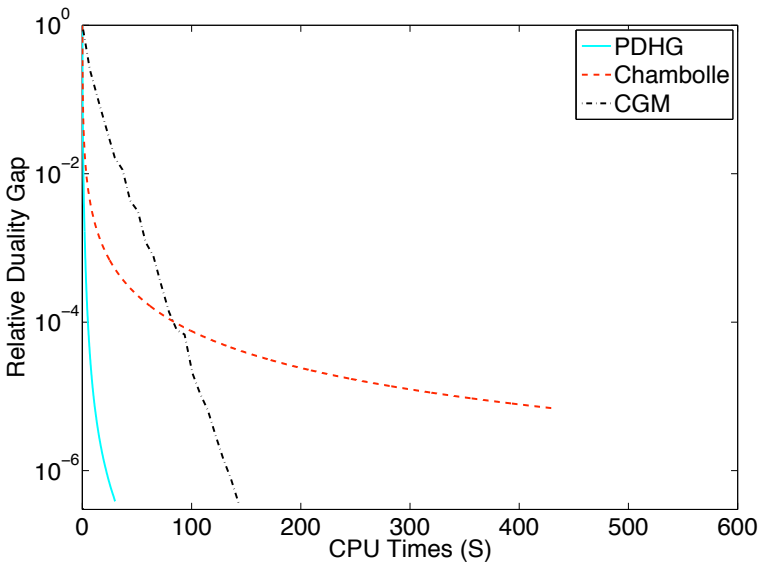


Figure: Duality gap of denoising problems of size 128^2, 256^2, and 512^2.

Implementing elementary operations on GPUs

CS: In SpaRSA algorithm, the major operations are:

- Ax, $A^T x$.
  - If A consists of m rows of a $n \times n$ DCT matrix, Ax and $A^T x$ takes $O(nlogn)$ using FFT, no need to store A.
  - Only $O(m)$ storage is required.
- Level-1 BLAS operations.
  - Parallel design/coding is required for each operation.
  - Can use CUBLAS, but custom codes are often better.

IR: PDHG algorithm.

- ∇u, $\nabla \cdot w$ by finite difference methods.
  - The (i, j)-th output element is computed by looking neighboring positions of (i, j) in u or w.
  - GPUs provide great features to speedup these 2-D data access patterns with spatial locality.
- Ax, $A^T x$.
  - 2-D DFT and inverse DFT (CUFFT) for deblurring.
- Level-1 BLAS operations.

1-D Compressive Sensing (SpaRSA)

We use one of the two GPUs in GeForce 9800 GX2 device, i.e. 128 cores, with 512MB global memory at 64GB/s.

$\tau/\tau_{\max}$	CPU			GPU			Speedup	
	iters	time (s)	MSE	iters	time (s)	MSE	total	iter
.000100	103	4.32	8.1e-10	129	0.16	7.2e-10	26	33
.000033	135	5.52	1.3e-10	126	0.15	2.0e-10	37	34
.000010	143	5.81	9.8e-11	139	0.17	1.3e-10	35	34

Table: DCT sensing matrix of dim. 8192×65536, with 1638 spikes

$\tau/\tau_{\max}$	CPU			GPU			Speedup	
	iters	time (s)	MSE	iters	time (s)	MSE	total	iter
0.000100	107	107.08	9.1e-10	129	2.08	8.5e-10	51	62
0.000033	131	129.10	1.7e-10	131	2.10	1.6e-10	61	61
0.000010	149	145.31	1.0e-10	160	2.57	9.0e-11	57	61

Table: DCT sensing matrix of dim. 131072×1048576, with 26214 spikes

2-D Compressive Sensing (SpaRSA)

$\tau/\tau_{\max}$	CPU			GPU			Speedup	
	iters	time (s)	MSE	iters	time (s)	MSE	total	iter
0.10	62	2.56	1.9e-05	67	0.09	1.9e-05	27	30
0.05	67	2.68	4.8e-06	68	0.08	4.8e-06	32	32
0.02	77	3.06	9.7e-07	83	0.10	9.6e-07	30	32

Table: DCT sensing matrix of dim. 1311×65536, with 60 spikes

$\tau/\tau_{\max}$	CPU			GPU			Speedup	
	iters	time (s)	MSE	iters	time (s)	MSE	total	iter
0.10	64	98.25	3.2e-05	64	1.00	3.1e-05	98	98
0.05	65	103.10	7.9e-06	70	1.08	7.9e-06	95	102
0.02	80	117.97	1.5e-06	84	1.30	1.5e-06	91	95

Table: DCT sensing matrix of dim. 20972×1048576, with 1031 spikes

Image Denoising

Image size	Tol	CPU		GPU		Speedup	
		iters	time (s)	iters	time (s)	total	iter
128^2	1.e-2	11	0.03	11	0.02	2	2
	1.e-4	79	0.21	79	0.02	11	11
	1.e-6	338	0.90	329	0.07	14	13
256^2	1.e-2	13	0.17	13	0.02	9	9
	1.e-4	68	0.81	68	0.03	32	32
	1.e-6	304	3.57	347	0.11	33	38
512^2	1.e-2	12	0.95	12	0.03	31	31
	1.e-4	54	3.96	54	0.05	76	76
	1.e-6	222	16.08	238	0.19	84	90
1024^2	1.e-2	14	5.42	14	0.08	64	64
	1.e-4	69	25.80	69	0.24	106	106
	1.e-6	296	103.54	324	1.02	102	111
2048^2	1.e-2	13	31.41	13	0.28	114	114
	1.e-4	67	149.24	67	0.90	165	165
	1.e-6	319	694.16	338	4.12	169	179

Table: Computational results of image denoising (λ=0.041.)

Conclusion

GPUs provide good platforms to speed up algorithms which

- incur no frequent data transfer,
- have small memory footprints,
- and consist of simple units easy to parallelize.

Analogy between GPUs and coprocessors in 80s.

- Add 80287 to speed-up flops, along with 80286.
- Add GPUs to speed-up flops by parallelism, along with CPUs.
- Intel is planning to embed GPUs in CPU chips.

GPUs provide a promising platform for:

- Speeding-up existing algorithms.
- Designing new parallel optimization algorithms.

Thank you.

Paper: `http://www.cs.wisc.edu/~sklee/`

Code: `http://www.cs.wisc.edu/~swright/GPUreconstruction/`