A Parallel Algorithm for Compressive Sensing

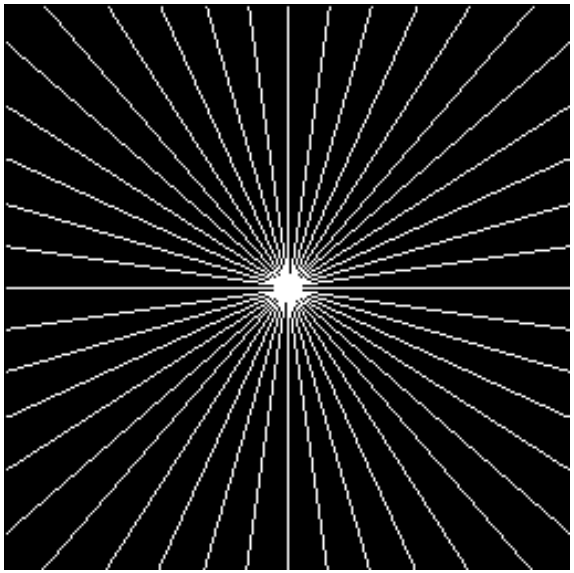
Alexandre Borghi

Laboratoire de Recherche en Informatique
Universite Paris-Sud 11

July 6 2011

Idea of Compressive Sensing

- Compressive Sensing idea [Candès et al. 2004]:
  under some sparsity assumptions one can exactly reconstruct a signal with few measurements
- Sparse signal = most of coefficients are 0 (in some basis).
- Measurements need to be taken in an appropriate space.
- Example: sparse gradient/measurements in Fourier domain



- Frequencies kept: 22 lines passing through the zero frequency

Compressive Sensing [Candes, Romberg, Tao, . . .]

- Wish to minimize

$$\begin{cases} \min_u J(u) \\ s.t.\ Au = f \end{cases}$$

  - with J being l_1 (or the Total Variation)
  - where A is a compressive sensing matrix, and f the observed data
- In this part of the talk, we consider a penalization approach (LASSO)

$$E_\mu(u) = J(u) + \frac{\mu}{2}\|Au - f\|_2^2$$

- Target: minimize $E_\mu(\cdot)$
  - many available algorithms!
  - mathematical design $\rightarrow$ data structures/algorithm $\rightarrow$ implementation
  - efficiency depends on the available computing technologies
    $\Rightarrow$ " parallel computing" capacities that are specific to each processor
    $\Rightarrow$ not all algorithms can take benefit from available features

l_1-Compressive Sensing

Sparsity $\equiv$ minimizing l_1-norm.

$$(\mathrm{CS}) \begin{cases} \min_u \|u\|_1 \\ s.t.\ Au = f \end{cases}$$

where

- $u \in \mathbb{R}^n$ is the signal to reconstruct
- $f \in \mathbb{R}^m$ is the observed data
- $A \in \mathbb{R}^{m \times n}$ is a Compressive Sensing matrix
- $m \ll n$

Outline

1. What kind of ” parallel computing” technologies are widely available?
2. Design of an approached l^1-compressive sensing algorithm (sparse signal)

Parallel Many-Core Architectures: CPU

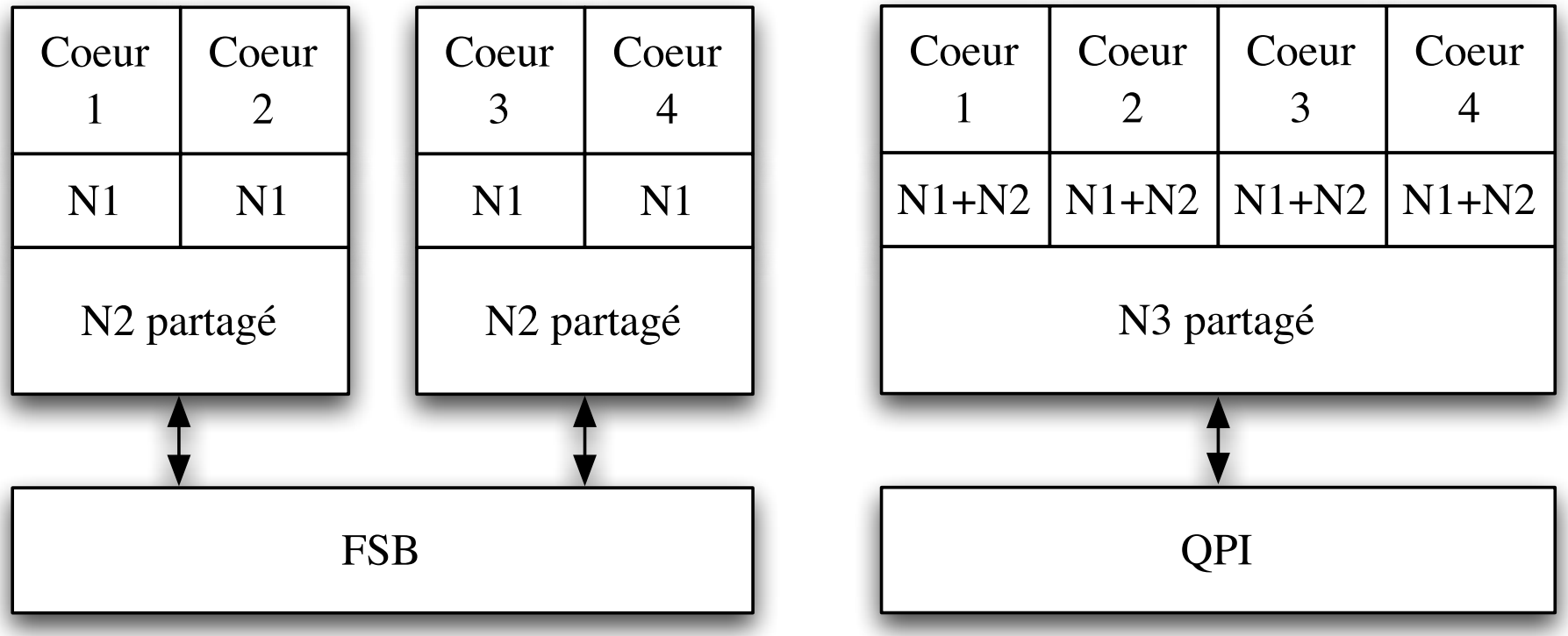


Figure: Intel CPU (left: Core 2 Quad; right: Core i7).

Parallel Many-Core Architectures: Cell

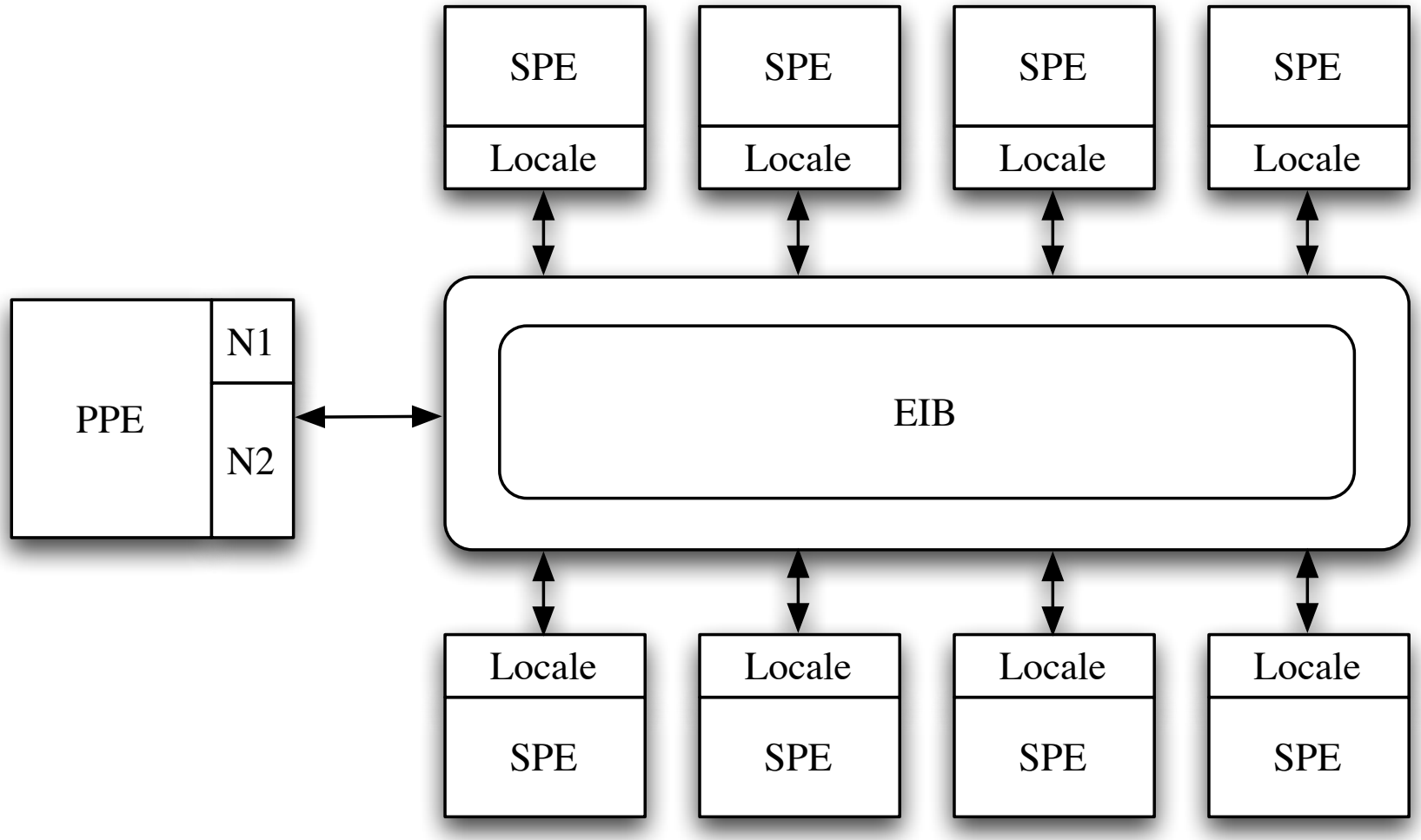


Figure: Cell.

Parallel Many-Core Architectures: GPU

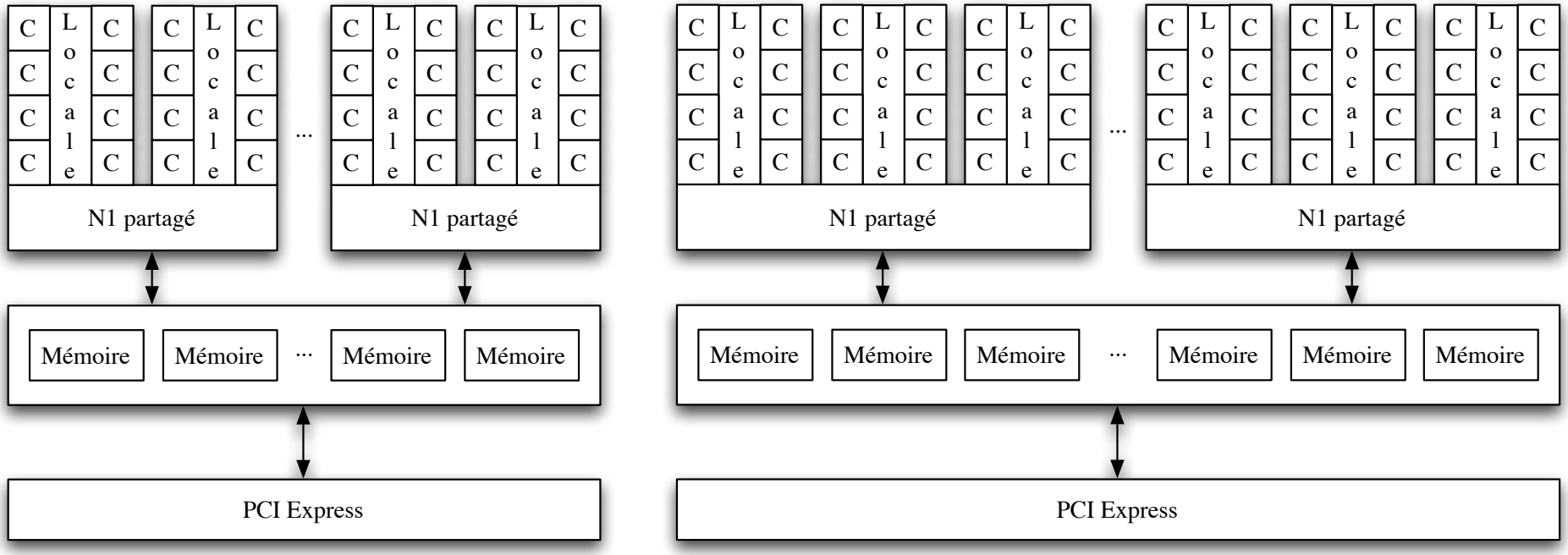


Figure: NVIDIA GPU (left : G80; right : G200).

Parallel Many-Core Architectures

- Restriction to Parallel Many-Core Architectures (widely available)
  - Video card: NVidia $\rightarrow$ GPU ($>$ 96 cores)
  - PC: Intel processors $\rightarrow$ multi-core ($>$ 4 Cores)
  - Playstation 3 video game console $\rightarrow$ Cell (6-8 cores) cheap!
- Specific characteristics: GFLOPS, energy consumption, price...
- Common features:
  - Parallel: composed of several computing units
  - Shared memory: central memory accessible from computing units
  - System of cache hierarchy to improve data transfer

Features and Requirements

- Coarse parallelism: Several computational units
  - Multi-core, GPU, Cell
  - synchronization issue
- Vectorization (SIMD): Process the data as a vector
  - alignment issue: data must be accessed at specific addresses
  - By far, the most important feature (because several SIMD units/core)
- Memory bandwidth: Amount of data that can be transfered
  - data are transferred back and forth from the memory to the processor
  - Starvation issue: a computing unit is waiting for the data
  - By far, the most important issue

$\rightarrow$ A compiler (or Matlab) tries to do that automatically

$\rightarrow$ Performance highly depends on a successful code optimization

$\rightarrow$ Design algorithms $\Rightarrow$ implementations meet these requirements

l^1-compressive sensing

1. What kind of ” parallel” computing technologies are widely available?
2. Design of an approached l^1-compressive sensing algorithm (sparse signal)
   - Moreau-Yosida Regularization/Proximal Points
   - Proximal operator choices
   - Algorithm
   - Experiments and implementation

Moreau-Yosida Regularization

- Recall, we wish to minimize

$$E_\mu(u) = J(u) + \frac{\mu}{2}\|Au - f\|_2^2$$

- Moreau-Yosida Regularization [Moreau 65, Yosida 66]:
  Given a metric M and any current point $u^{(k)}$

$$F_\mu(u^{(k)}) = \inf_{u \in \mathbb{R}^n} \left\{ E_\mu(u) + \frac{1}{2}\|u - u^{(k)}\|_M^2 \right\} \quad .$$

- This defines the proximal point $p_\mu(u^{(k)})$ and is characterized by:

$$0 \in \partial \left(E_\mu + \frac{1}{2}\| \cdot - u^{(k)}\|_M^2 \right) (p_\mu(u^{(k)}))$$

- That is

$$p_\mu\left(u^{(k)}\right) = (M + \partial E_\mu)^{-1}\left(Mu^{(k)}\right) \quad .$$

Proximal Operator Choices

- Standard convergence result [Lemarechal 97, Rockafellar 76, ...]: Iterating the proximal point generates a sequence that converges toward a minimizer of E_μ
- How to choose M for parallel computations?
  Rewriting $(M + \partial E_\mu)^{-1}$, we have

$$\partial J(u^{(k+1)}) + (\mu A^t A + M)u^{(k+1)} = \mu A^t f + M u^{(k)} \quad .$$

  $\Rightarrow$ We wish $\mu A^t A + M$ diagonal
  $\Rightarrow$ problem of the form $\ell_1 + \| \cdot - g\|_2^2$

- So we take $M = (1+\epsilon)\mu Id - \mu A^t A$

Proximal Operator Choices

- With such a metric

$$M = (1+\epsilon)\mu Id - \mu A^t A \ ,$$

- Solution is given by a shrinkage

$$u^{(k+1)} = \frac{1}{(1+\epsilon)\mu} \begin{cases} \mu A^t f + M u^{(k)} - sg\left(\mu A^t f + M u^{(k)}\right) & \text{if } \left|\mu A^t f + M u^{(k)}\right| > 1 \\ 0 & \text{otherwise} \ , \end{cases}$$

- Shrinkage has been very well studied [Chambolle, Daubechies, Elad, Figuereido, Yin, ...] and used in many efficient l^1-CS algorithms

$\rightarrow$ Shrinkage is easily and efficiently implemented on many-cores

$\rightarrow$ Need to perform Au and A^tAu in parallel

  - A is a fast transform $\rightarrow$ good (not great) parallel and vectorized version
  - A is an explicit matrix $\rightarrow$ great vectorized version but badly parallel (bandwith issue)

Algorithm

- Recall, we wish to minimize (with μ large)

$$E_\mu(u) = J(u) + \frac{\mu}{2}\|Au - f\|_2^2$$

- How to choose initial μ
  - too small $\rightarrow$ shrinkage yields 0
  - too big $\rightarrow$ slow convergence
  - choose of a good one by a bitonic search
- Algorithm:
  1. Start with $u = 0$
  2. Compute the initial value of μ using the above bitonic approach
  3. For k=0 to l_{max}
     - Iterate the proximal point until some convergence criteria are met

$$\frac{\|Au^k - f\|_2^2}{\|f\|_2^2} < tolerance$$

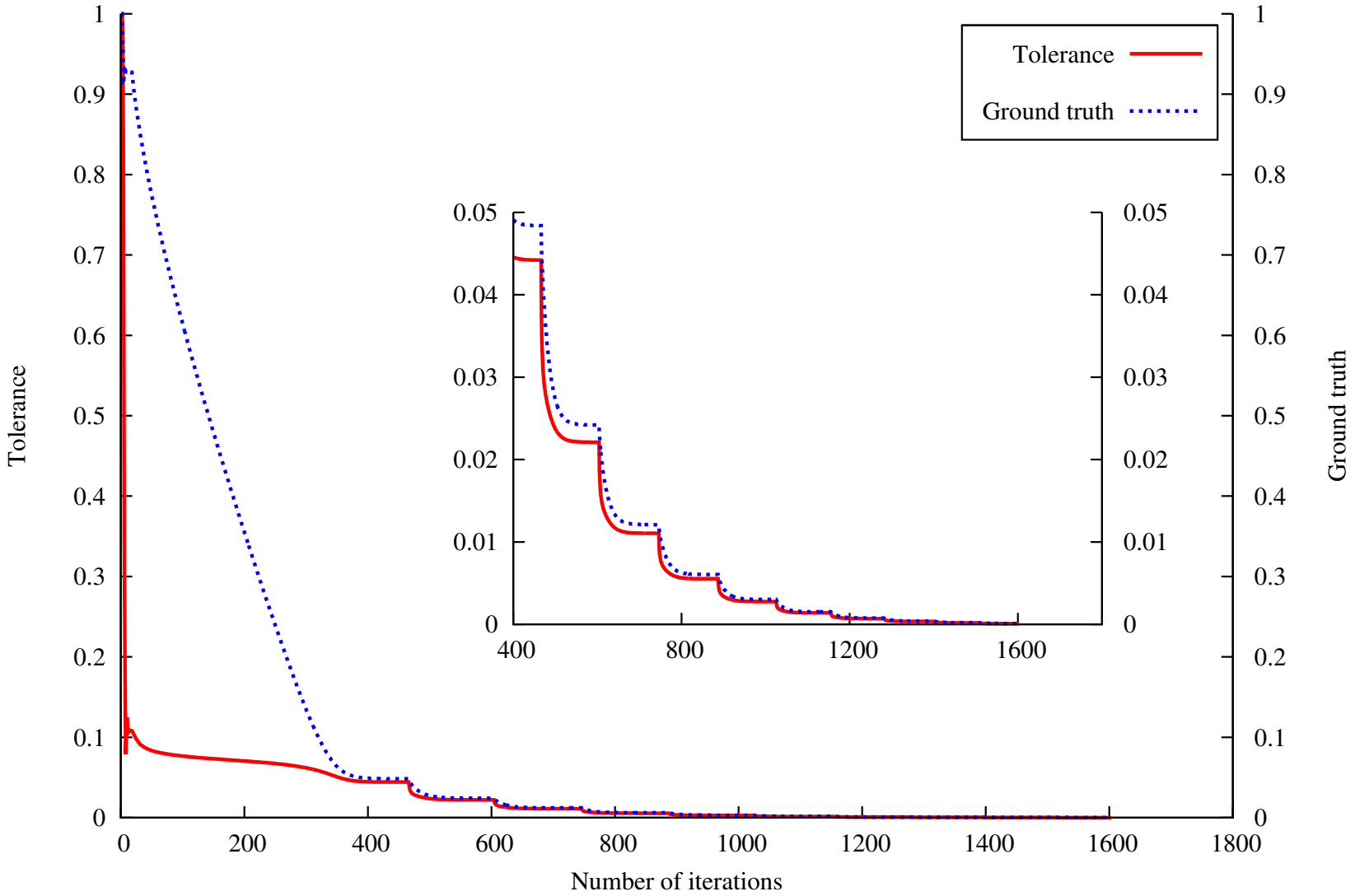
Exact solution from approximate solution

1. Compute the basis matrix B associated with $u^{(k+1)}$ by choosing the columns of A corresponding to the nonzero elements of $u^{(k+1)}$
2. Compute $v = B^{-1} \times f$
3. Return $u^{(k+2)}$ in base A from v in base B by adding zeros where necessary

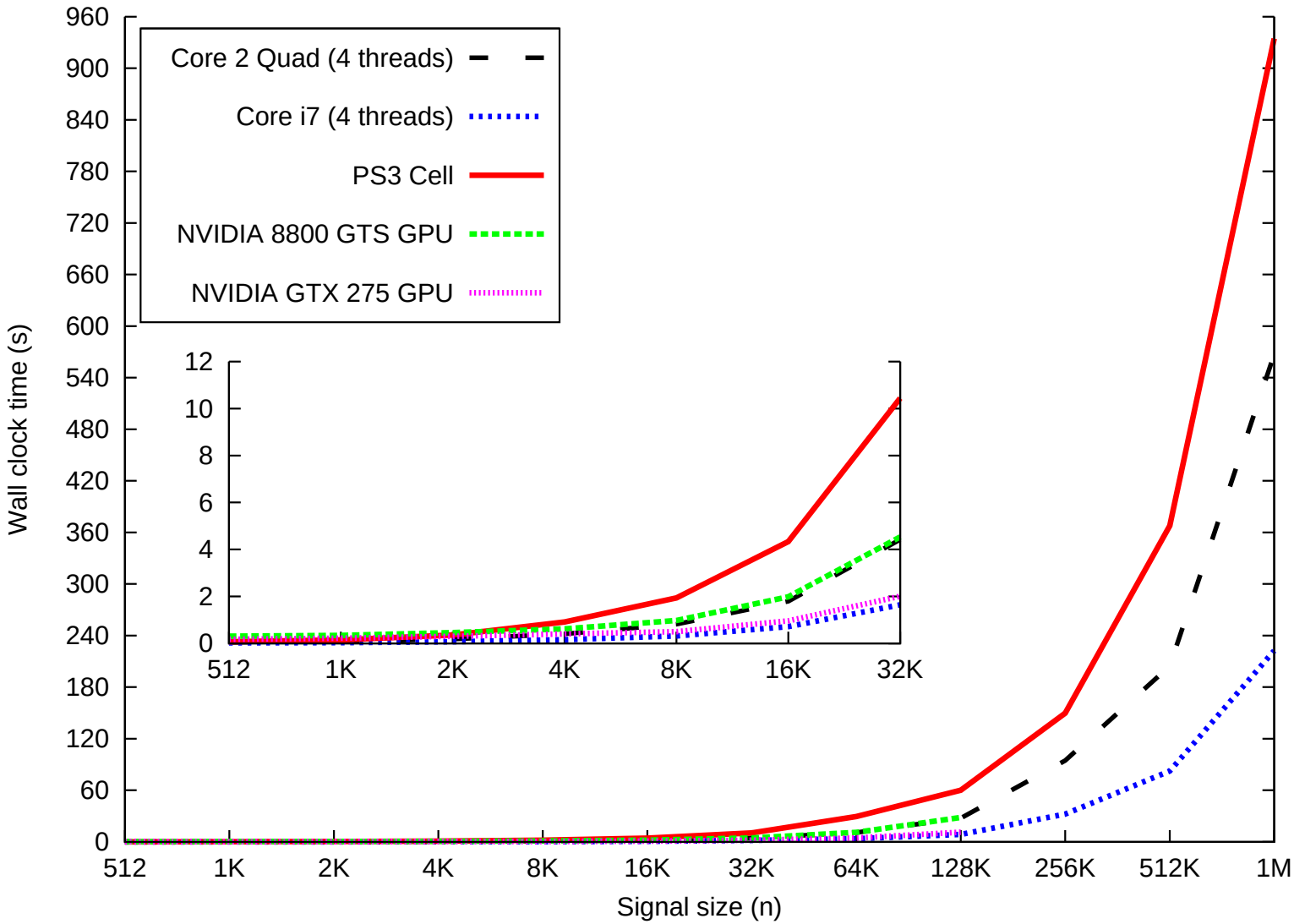
Experiments and Implementation

- Implementation:
  - shrinkage is parallel, vectorizable
  - Au is either implemented as an explicit matrix multiplication or a Fast Transform (e.g., DCT)
  - explicit multiplication $\rightarrow$ huge amount of data to transfer
- Experimental conditions
  - Multi-core: Intel Core 2 Quad Q6600, 76 GFLOPS (theoretical)
  - Multi-core: Intel Core i7 920, 86 GFLOPS (theoretical)
  - Cell: Sony Playstation 3, 159 GFLOPS (theoretical)
  - GPU: NVidia 8800GTS, 345 GFLOPS (theoretical)
  - GPU: NVidia GTX 275, 1010 GFLOPS (theoretical)

Experiments: Errors



Experiments: DCT



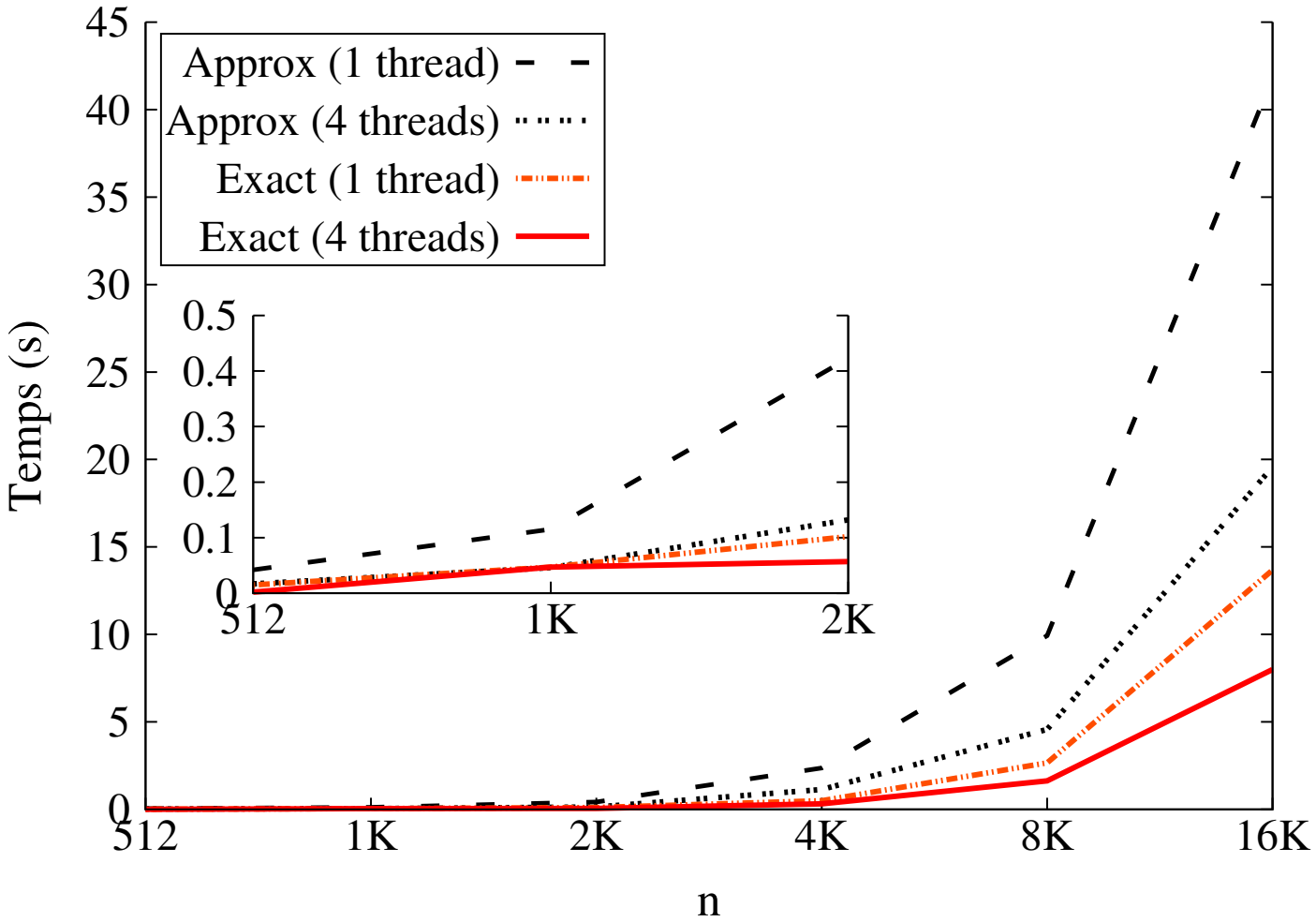
Experiments: Orthogonalized Gaussian matrices

- For Intel Core i7 920 (2.66 GHz)
- Signal of size n with m observations and $0.1m$ spikes
- Relative error $\frac{\|u_{found} - u_{true}\|_2}{\|u_{true}\|_2}$

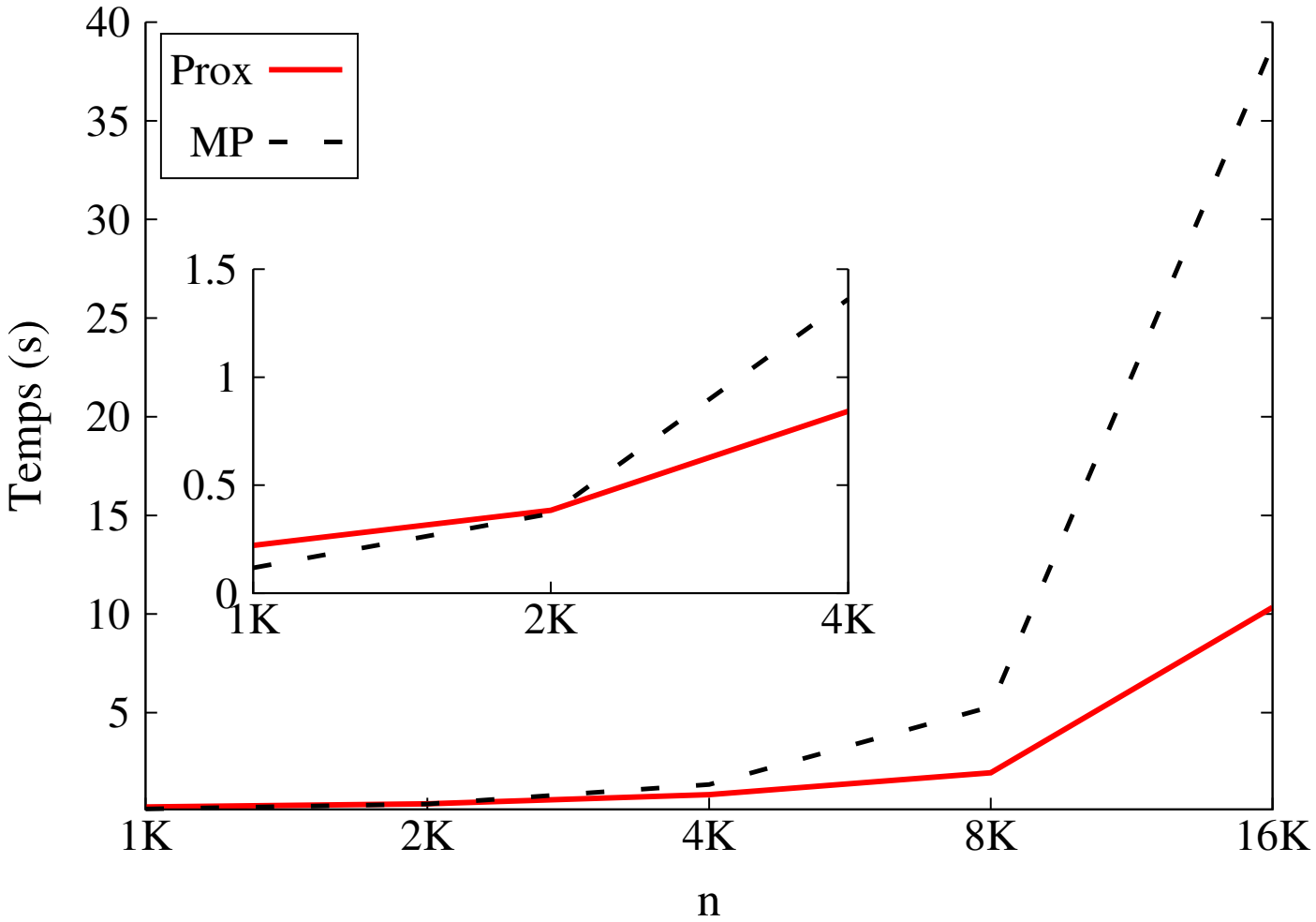
in		out		time (s)			
m	n	relative	#iter	1 th.	2 th.	4 th.	8 th.
64	512	1.39e-03	1162.0	0.042	0.025	0.017	0.252
128	1024	5.22e-04	1155.5	0.115	0.068	0.046	0.350
256	2048	3.26e-04	1321.5	0.423	0.227	0.132	0.375
512	4096	2.15e-04	1465.8	2.362	1.495	1.130	1.359
1024	8192	1.43e-04	1505.6	9.933	6.035	4.574	4.885
2048	16384	9.62e-05	1604.9	41.842	25.284	19.576	19.997

- Compare to Bregman based approach [Yin, Osher, Goldfarb,Darbon 08] between 5/10 times faster.

Experiments: Approximate versus Exact resolution



Experiments: Versus Matching Pursuit



Conclusion

- So far:
  - Use a proximal operator (to assure convergence)
  - Design the operator to allow parallel programming
  - The compiler will perform the parallel optimization for you
- On Moreau-Yosida/Proximal Point
  - Huge literature $\rightarrow$ gave birth to splitting/preconditioning
  - General theoretical properties: convergence, robustness...
  - [Rockafellar 98] or [Hiriart-Urriuty-Lemarechal 94]
- Many similar approaches
  - using shrinkage (linearization for instance)
  - using other Proximal points
- More in [Borghi, Darbon, Peyronnet, Chan, Osher 08] (UCLA Cam report)
- Use similar idea for $J(u) = |\nabla u|_{l^1}$