Fast ℓ_1-Minimization and Parallelization for Face Recognition

Victor Shia, Allen Y. Yang, and S. Shankar Sastry
Department of EECS, UC Berkeley
Berkeley, CA 94720, USA
{vshia, yang, sastry} @ eecs.berkeley.edu

Andrew Wagner and Yi Ma
Coordinated Science Lab, University of Illinois
Urbana, IL 61801, USA
{awagner, yima} @ illinois.edu

***Abstract*—While ℓ_1-minimization (ℓ_1-min) has recently been studied extensively in optimization, the high computational cost associated with the traditional algorithms has largely hindered their application to high-dimensional, large-scale problems. This paper discusses accelerated ℓ_1-min techniques using augmented Lagrangian methods and its parallelization leveraging the parallelism available in modern GPU and CPU hardware. The performance of the new algorithms is demonstrated in a robust face recognition application. Through extensive simulation and real-world experiments, we provide useful guidelines about applying fast ℓ_1-min on large-scale data for practitioners.**

I. Introduction

In the past few years, there has been a lot of interest in finding numerical solutions to a set of sparsity minimization problems. In particular, in compressive sensing (CS) theory, it has been shown that a linear program known as ℓ_1-minimization (ℓ_1-min) can recover sparse solutions to certain underdetermined systems of linear equations [4], [7]. The ℓ_1-min problem refers to finding the minimum ℓ_1-norm solution to an underdetermined linear system $\boldsymbol{b} = A\boldsymbol{x}$:

$$\min \|\boldsymbol{x}\|_1 \quad \text{subj. to} \quad \boldsymbol{b} = A\boldsymbol{x}. \tag{1}$$

Under certain conditions [3], the minimum ℓ_1-norm solution is also the *sparsest* solution to the system (1).

Among the various applications of sparse representation and ℓ_1-min, face recognition has been formulated using a sparsity-based classification framework (SBC) in [15]. If we stack the pixels of the training images of K subject classes into the columns of matrices ($A_1 \in \mathbb{R}^{d\times n_1}, \cdots, A_K \in \mathbb{R}^{d\times n_K}$), combine the matrices into a larger matrix $A = [A_1, \cdots, A_K] \in \mathbb{R}^{d\times n}$, and arrange the pixels of a new query image into a vector $\boldsymbol{b} \in \mathbb{R}^d$ of the same dimension, SBC solves the following minimization problem:

$$\min_{\boldsymbol{x},\boldsymbol{e}} \|\boldsymbol{x}\|_1 + \|\boldsymbol{e}\|_1 \quad \text{subj. t.} \quad \boldsymbol{b} = A\boldsymbol{x} + \boldsymbol{e}, \tag{2}$$

This work was partially supported by ARO MURI W911NF-06-1-0076, NSF IIS 08-49292, NSF ECCS 07-01676, and ONR N00014-09-1-0230.

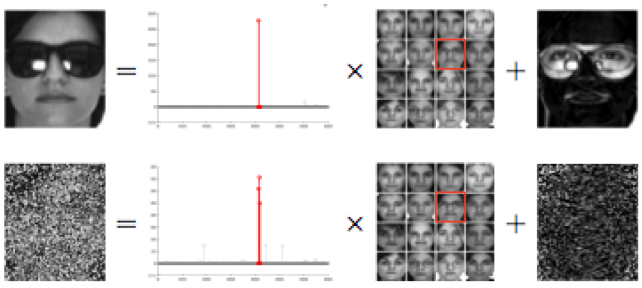

Fig. 1. Robust face recognition via sparse representation. The method represents a test image (left), which may be partially occluded or corrupted, as a sparse linear combination of all the normal training images (middle) plus sparse errors (right). The largest nonzero coefficients in red correspond to training images of the correct individual. The nonzero coefficients in the error recover the locations and ground-truth values of the corrupted pixels in the input image.

where $\boldsymbol{e}$ provides a means to compensate for pixels that are corrupted due to occlusion on some part of the query image, and $\boldsymbol{x}$ is the sparse representation of $\boldsymbol{b}$ under the training dictionary A.

Once the sparse signals $\boldsymbol{x}$ and $\boldsymbol{e}$ are recovered, tasks such as recognition and validation can be performed in a very natural way. For a corrupted face image $\boldsymbol{b}$, subtracting $\boldsymbol{e}$ from the measurements recovers an estimate of the appearance of the face image: $\boldsymbol{b}_0 \doteq \boldsymbol{b} - \boldsymbol{e}$. One can further show that in fact the objective function (2) can handle noise correction even when $\boldsymbol{e}$ is quite dense provided its signs and support are random [14], as two examples shown in Figure 1.

For sparse representation $\boldsymbol{x}$, one can simply define the concentration of a vector $\boldsymbol{x} = [\boldsymbol{x}_1^T, \boldsymbol{x}_2^T, \cdots, \boldsymbol{x}_K^T]^T$ on Class i as

$$\alpha_i \doteq \|\boldsymbol{x}_i\|_1 / \|\boldsymbol{x}\|_1. \tag{3}$$

Then the label of the query image $\boldsymbol{b}$ is assigned to the class that maximizes α_i. The algorithm may also reject $\boldsymbol{b}$ as not relevant to any subject class in the training database (i.e., an outlier) if the maximum value of α_i is smaller than a predetermined threshold, which corresponds to a *dense* representation in

$\boldsymbol{x}$. Alternatively, one can also classify $\boldsymbol{b}$ by minimizing the residual of the reconstruction error:

$$r_i(\boldsymbol{b}) \doteq \|\boldsymbol{b} - A_i\boldsymbol{x}_i\|_2. \quad (4)$$

In face recognition, the accuracy of the classification also depends on the condition that the query image is aligned with the training images (in vector form). In the case misalignment may occur, SBC has been extended in [13] to be able to align a query image to each subject class individually. The alignment algorithm solves the following problem:

$$\hat{\tau}_i = \arg\min_{\boldsymbol{x},\boldsymbol{e},\tau_i} \|\boldsymbol{e}\|_1 \quad \text{subj. to} \quad \boldsymbol{b} \circ \tau_i = A_i\boldsymbol{x} + \boldsymbol{e}, \quad (5)$$

where $\tau_i \in T$ is in a finite-dimensional group T of transformations (e.g., affine or homography) acting on the image domain. The sequence τ_i converges to a transformation that aligns the test image $\boldsymbol{b}$ with the training images from the i-th class A_i. The alignment algorithm works reasonably well with poses up to $\pm 45^\circ$, which easily exceeds the pose requirement for real-world access-control applications [13].

In pursuit of a practical face recognition system, one concern is the availability of efficient algorithms to minimize ℓ_1-min and its variations for face alignment (5) and face recognition (2) in real time. Although ℓ_1-min can be formulated as a linear program, conventional algorithms such as interior-point methods [5], [12] scale poorly in high-dimensional space. Due to the increasing attention on sparse representation and compressive sensing, a number of accelerated ℓ_1-min algorithms have been recently proposed, which explicitly take advantage of the special structure of the ℓ_1-min problems:

1) Gradient projection methods [11] follow closely the idea of interior-point methods, but reformulate ℓ_1-min as a quadratic programming problem. As a result, the computation of the Newton update can be accelerated by conjugate gradient methods.
2) Homotopy methods [9], [8] utilizes a warm-start strategy to minimize a series of noisy versions of ℓ_1-min, and the nondifferentiable ℓ_1-min objective can be approximated by its subgradients on the sparse support.
3) Iterative soft-thresholding methods [6], [1] belong to a category of first-order methods, which refer to those algorithms that have at most linear local error, typically based on local linear approximation. In the iterative update rule, an efficient soft-thresholding function is applied element-wise to update the value of the unknown variables.

In [17], a new solution based on augmented Lagrangian methods (ALM) was proposed, which we will discuss in Section II. ALM is also a first-order method, and it has been shown to perform much faster than the other existing methods in a systematic comparison [17]. However, due to the high dimensionality of the face recognition problem, the system still fails to achieve real-time performance using a naive implementation of the algorithm on a typical workstation. In Sections III and IV, we discuss how to properly parallelize ALM ℓ_1-min on multi-core CPU/GPU architectures.

II. Augmented Lagrangian Methods

In this section, we present the ALM approach for ℓ_1-min (1) [17] and analyze its complexity. Lagrange multipliers have been frequently used in convex programming to eliminate equality constraints via adding a penalty term to the cost function for infeasible points. ALM methods differ from other penalty-based approaches by simultaneously estimating the optimal solution and Lagrange multipliers in an iterative fashion. By introducing a quadratic penalty term, the ALM algorithm converges at a faster rate than iterative thresholding algorithms. For standard ℓ_1-min problem (1), the augmented Lagrange function is defined as:

$$L_\mu(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x}\|_1 + \boldsymbol{y}^T(\boldsymbol{b} - A\boldsymbol{x}) + \frac{\mu}{2}\|\boldsymbol{b} - A\boldsymbol{x}\|_2^2, \quad (6)$$

where $\frac{1}{2}\|\boldsymbol{b} - A\boldsymbol{x}\|_2^2$ is the augmented Lagrangian term, $\mu > 0$ is a constant that penalizes infeasibility, and $\boldsymbol{y}$ is a vector of Lagrange multipliers

In Lagrange Multiplier Theory [2], if there exists a Lagrangian y^* that satisfies the second-order sufficiency conditions for optimality, then for a sufficiently large μ, the optimal ℓ_1-min solution also minimizes

$$\boldsymbol{x}^* = \arg\min L_\mu(\boldsymbol{x}, \boldsymbol{y}^*). \quad (7)$$

In practice, the optimal values for the triplet $(\boldsymbol{x}^*, \boldsymbol{y}^*, \mu)$ are all unknown. Furthermore, it has been observed that solving (7) with a large initial value of μ tends to lead to slower convergence speed [16], [17]. In [2], [18], an alternating procedure has been shown to iteratively update $\boldsymbol{x}$ and $\boldsymbol{y}$:

$$\begin{cases} \boldsymbol{x}_{k+1} &= \arg\min_{\boldsymbol{x}} L_{\mu_k}(\boldsymbol{x}, \boldsymbol{y}_k) \\ \boldsymbol{y}_{k+1} &= \boldsymbol{y}_k + \mu_k(\boldsymbol{b} - A\boldsymbol{x}_{k+1}) \end{cases}, \quad (8)$$

where $\mu_k \to \infty$ increases monotonically. The iteration terminates when the estimates $(\boldsymbol{x}_k, \boldsymbol{y}_k)$ converge.

Note that in the iterative procedure (8), the second step only involves vector algebra and matrix-vector multiplication. Therefore, the procedure is computationally efficient if it is easier to minimize the augmented Lagrangian $L_{\mu_k}(\boldsymbol{x}, \boldsymbol{y}_k)$ compared to solving the original problem (1) directly. In fact, this problem can be solved element-wise iteratively by a soft-thresholding algorithm [16], [1], whose time complexity is bounded by $O(n^2)$ and can be easily parallelized. Algorithm 1 summarizes the generic ALM ℓ_1-min algorithm. [1]

III. Parallelism in Face Alignment

In order to speed up ℓ_1-min for specific applications such as face recognition, one should take advantage of the problem parallelism to carefully map and implement the problems on state-of-the-art parallel computing architectures, such as multi-core CPUs and general-purpose GPUs. As a first-order method, ALM mainly uses vector-vector and matrix-vector multiplications, which make it ideal for use in parallel computing environments using standard LAPACK and BLAS numerical libraries. In this section, we describe the parallelization of the the ALM algorithm for the face alignment problem.

[1] For conciseness, we only present the ALM algorithm in the primal domain. There also exist implementations in the dual domain [18], [17].

Algorithm 1 Augmented Lagrangian Method (ALM)

INPUT: $\boldsymbol{b} \in \mathbb{R}^m$, $A = [A_1, \cdots, A_K] \in \mathbb{R}^{m \times n}$, $\tau \leftarrow \max \text{eig}(A^T A)$, and constant $\rho > 1$.

while not converged ($k = 1, 2, \ldots$) **do**
  $t_1 \leftarrow 1$, $\boldsymbol{z}_1 \leftarrow \boldsymbol{x}_k$, $\boldsymbol{u}_1 \leftarrow \boldsymbol{x}_k$
  while not converged ($l = 1, 2, \ldots$) **do**
    $\boldsymbol{u}_{l+1} \leftarrow \text{shrink}(\boldsymbol{z}_l - \frac{1}{\tau} A^T (A\boldsymbol{z}_l - \boldsymbol{b} - \frac{1}{\mu_k}\boldsymbol{y}_k), \frac{1}{\mu_k \tau})$
    $t_{l+1} \leftarrow \frac{1}{2}(1 + \sqrt{1 + 4t_l^2})$
    $\boldsymbol{z}_{l+1} \leftarrow \boldsymbol{u}_{l+1} + \frac{t_l - 1}{t_{l+1}}(\boldsymbol{u}_{l+1} - \boldsymbol{u}_l)$
  end while
  $\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{u}_{l+1}$
  $\boldsymbol{y}_{k+1} \leftarrow \boldsymbol{y}_k + \mu_k(\boldsymbol{b} - A\boldsymbol{x}_{k+1})$
  $\mu_{k+1} \leftarrow \rho \cdot \mu_k$
end while

OUTPUT: $\boldsymbol{x}^* \leftarrow \boldsymbol{x}_k$.

Face alignment (5) estimates an image transformation τ that rectifies the query image $\boldsymbol{b}$ with possible pose variation w.r.t. each training class A_i, which leads to the minimal sparsity in error $\boldsymbol{e}$ after the alignment. Note that directly solving (5) is inefficient since it is a non-convex problem and may exhibit local minima. However, given a good initial guess of the transformation τ (*e.g.*, provided by a good face detector), the optimal solution for τ can be sought iteratively by linearization at each jth step:

$$\min_{\boldsymbol{x}, \boldsymbol{e}, \Delta\tau_j} \|\boldsymbol{e}\|_1 \quad \text{subj. t.} \quad \boldsymbol{b} \circ \tau_j + J_j \Delta\tau = A_i \boldsymbol{x} + \boldsymbol{e}, \tag{9}$$

where $J_j \doteq \nabla_{\tau_j}(\boldsymbol{b} \circ \tau_j)$ is the Jacobian, and $\Delta\tau$ is a step update to τ. Denote $\boldsymbol{b}_j = \boldsymbol{b} \circ \tau_j$, $B_j = [A_i, -J_j]$, and $\boldsymbol{w}^T = [\boldsymbol{x}^T, \Delta\tau^T]$, then the update $\Delta\tau$ can be computed by solving the following linear program:

$$\min_{\boldsymbol{w}, \boldsymbol{e}} \|\boldsymbol{e}\|_1 \quad \text{subj. t.} \quad \boldsymbol{b}_j = B_j \boldsymbol{w} + \boldsymbol{e}. \tag{10}$$

A. CPU Implementation

The face alignment algorithm was implemented on an Linux workstation with two quad-core Intel Nehalem E5530 processors clocked at 2.4 GHz. Each processor has its own memory interface and is interfaced to half of the RAM installed, totaling 16GBs. Each core has 32 KiB L1 cache and 256 KiB L2 cache. Each processor has 8 MiB of L3 cache shared by the four cores.

For alignment, each core is able to store 16 64x64 grayscale face images. We implemented two versions of the parallel algorithm, one using the standard Basic Linear Algebra Subprograms (BLAS) libraries without any optimization and a manually optimized version. In the manually optimized version, to make efficient use of 8 cores, we manually map eight instances of (10) to run in parallel, one problem on each core. The advantage of this implementation over the standard BLAS implementation is that no synchronization is necessary between the cores. Both implementations are compiled using the Intel Make Kernel Library (MKL) and Intel Integrated Performance Primitives (IPP), which are optimized for Intel multi-core CPUs. Operations that are not provided by Intel libraries are written and parallelized with the Intel C++ compiler.

B. GPU Implementation

The GPU implementation is similar in spirit to our second CPU implementation where each streaming multi-processor solves a set of problems independently from the other processors. We use a Nvidia Fermi GPU with 14 streaming multiprocessors (MP) with 64 cores per MP. Each MP has its 64 KiB of L1 cache and all MPs share a common 768 KiB L2 cache. The small amount of cache on the GPU is balanced by a significantly higher bandwidth between the processor and off-chip memory (DRAM) on the GPU. We have empirically determined that performance is highest with 5-7 subject classes aligned simultaneously on each MP.

IV. Parallelism in Face Recognition

In this section, we describe the parallelization of the ALM algorithm for the face recognition problem.

In our implementation of the face recognition system, when the face alignment stage is complete, only the top 20 subject classes with lowest alignment error $\boldsymbol{e}$ are selected for recognition, as large alignment error indicates the query image does not align well with the appearance of the subject class in question. This heuristic reduces the computational complexity of the recognition stage when the database contains a large number of subjects and was empirically found to contain the correct subject 95% of the time. Clearly, this heuristic can be relaxed if higher recognition rate is required at the cost of slower speed.

After $\boldsymbol{b}$ and A_i are re-sampled to a common alignment using τ_i, and A_i are concatenated into a new matrix A, a sparse representation of $\boldsymbol{b}$ w.r.t A is then recovered by solving a single ℓ_1-min problem. The coefficients $\boldsymbol{x}$ are then used to compute error residuals that are used for classification in (4). In this stage of the algorithm, there is no problem-level parallelism to exploit, instead we will discuss how to exploit pixel-level parallelism on both CPU and GPU hardware. Then in Section V, we will benchmark the performance of the two architectures and further demonstrate the speed gain achieved by our proposed implementations over previously published implementations.

On the CPU, most of the operations map well onto MKL BLAS calls, and operations that do not can be easily parallelized using OpenMP and auto-vectorization.

On the GPU, most of the operations map well onto similar calls in NVIDIA's GPU BLAS library (CUBLAS) which, like MKL, can take advantage of both levels of concurrency available in the hardware architecture. Operations that do not map well onto the CUBLAS API are implemented directly in CUDA code. Additionally, some operations that could have been implemented via multiple BLAS calls, performance improvements are achieved by combining multiple vector-vector operations into a single kernel, due to reduced bandwidth and kernel call overhead.

In order to avoid expensive data transfer across the PCI express bus connecting the GPU card and the CPU motherboard,

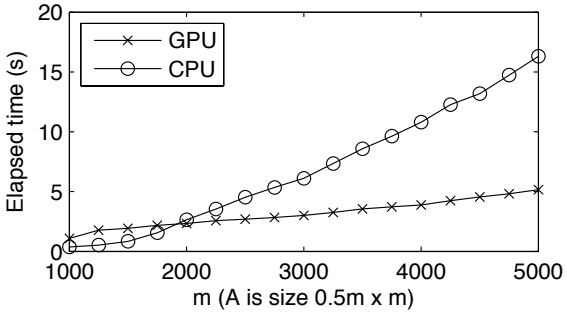

Fig. 2. Comparison of ℓ_1-min runtime vs. dimensions of A on random data.

all of the data is transferred to GPU DRAM once, and all non-trivial tasks are performed on the GPU on data stored in GPU DRAM.

V. Experiments

In this section, we first benchmark the performance of our parallel implementations of ALM-based ℓ_1-min on CPU and GPU platforms using simulated data. We then demonstrate the speed and accuracy of our implementation applied to the alignment and recognition stages of the face recognition system.

A. Simulation on Random Data

The first experiment compares the performance of our proposed CPU and GPU implementation of the general ℓ_1-min solver. The $d \times n$ measurement matrix A is a random Gaussian matrix, with each entry generated from the standard normal distribution and normalized to unit column norm. The ratio of d/n is fixed at $1/2$ (without loss of generality for the simulation purpose) with n varying from 1000 to 8000. The ground truth signal $\boldsymbol{x}_0$ has a sparsity rate of 10% of d with the values of the nonzero elements sampled from a normal distribution and normalized to unit column norm. Because the ground truth signal $\boldsymbol{x}_0$ is known, the algorithm terminates when $\|\boldsymbol{x} - \boldsymbol{x}_0\| < \epsilon$ with $\epsilon = 10^{-3}$. The measurement vector is generated by $\boldsymbol{b} = A\boldsymbol{x}_0$.

The results of this benchmark can be found in Figure 2. The x-axis represents the size of the A matrix, and the y-axis represents the average runtime to complete a single problem instance. The GPU implementation tends to be faster than the CPU implementation at solving a single large problem, whereas the CPU implementation is faster at solving a single small problem. The change in the slope of the CPU curve represents a transition for the CPU speed to be bounded from cache bandwidth to memory bandwidth. Since there are other data structures besides A, the transition occurs slightly before the size of A reaches $2000 \times 1000 \times 4 = 8MB$, i.e., the size of the CPU L3 cache.

B. Face Recognition Pipeline Benchmark

First, in order to measure the impact of solving many ℓ_1 problems-per concurrently on the GPU, we benchmark three implementations of the alignment ℓ_1-min solver with A of size 5120×32, and a fixed 50 inner loop iterations for each of 50 outer loop iterations. The runtime on the GTX480 GPU is averaged over a large number of trials, which are run sequentially or concurrently depending on the implementation. The results are shown in Table I. Our proposed parallelization of the ℓ_1-min used in the alignment stage is *eight times* faster than the implementation solving a single problem at a time using the stock BLAS libraries.[2]

TABLE I
Benchmark of three different parallelization approaches for alignment ℓ_1-min on the GTX480 GPU.

Sequential solver using CUBLAS	302 ms
One problem per SMP using CUDA streams	70 ms
Four problems per SMP using single kernel	**36 ms**

Using an implementation motivated by the previous result, we now benchmark our iterative alignment implementation on real face data. For experiments on face data we use subsets of the CMU Multi-PIE Face Database [10]. For gallery images we use frontal images from session 1, which contains 20 images per subject taken under different illuminations. For test images we use images from session 2. The training images are prepared as follows: The iterative alignment stage seeks a similarity transformation between the coordinate frame of the full-resolution test image and a "canonical" coordinate frame in which images are compared. The training images are aligned by applying a similarity transormation that maps two manually clicked outer eye corners to fixed locations in the canonical frame. A 64×64 pixel window in the canonical frame is used for resampling.

Figure 3 shows the average runtime of the CPU and GPU implementations to align a query image against all the subject classes separately. We vary the total number of subject classes to show how the algorithms scale. The plateaus seen in the GPU curve result from the GPU hardware scheduling the computation of alignment problems in concurrent batches, but the overall trend is linear as expected. The manually threaded CPU implementation slightly outperforms the GPU implementation, and it beats the *naive* library threaded implementation by a wide margin. The new implementation can align the query image in $\approx$ 40 ms per subject, while the fastest previously published result [13] required $\approx$ 600 ms per subject in a similar setting.

The next experiment compares the speed of the GPU and CPU implementation of the recognition stage. It was determined that keeping 20 gallery subjects is sufficient to ensure that the correct subject is kept for the recognition stage with 95% probability. Since recognition failures are typically caused by a poor alignment, we find that keeping more subjects for the recognition stage does not necessarily improve recognition rate.

Figure 4 shows the recognition stage runtime for different image sizes: 32×32, 48×48, 64×64, 96×96, and $128 \times$

[2]The streams implementation is limited significantly by a cap on the number of concurrent streams in the current version of CUDA.

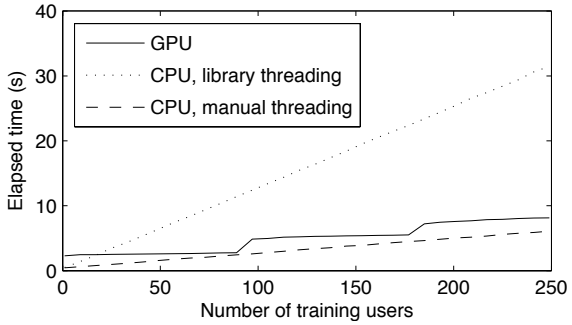

Fig. 3. Alignment stage runtime vs. size of training database.

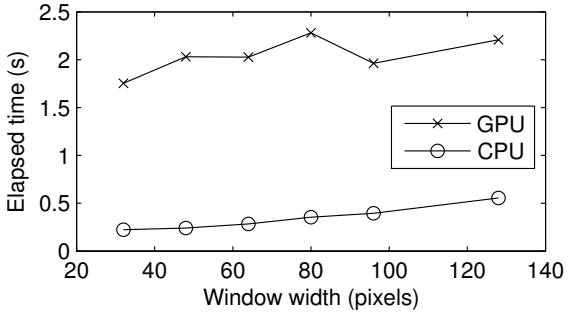

Fig. 4. Recognition stage runtime vs. face window resolution

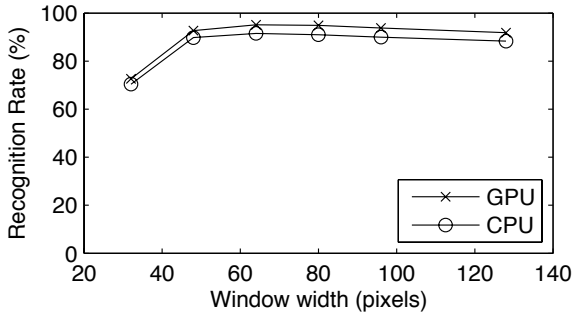

Fig. 5. Recognition rate of the full pipeline vs. face window resolution.

128. For all image resolutions, the problem size is sufficiently small that the CPU is significantly faster than the GPU. Note also that for both implementations, the recognition stage takes much less time than the alignment stage.

Finally, we perform an experiment verifying the recognition accuracy of the overall pipeline. As shown in Figure 5, at the optimal resolution, which happens to match the alignment stage resolution, the GPU implementation reaches 95% recognition rate, the max achievable given the alignment selection process. For significantly lower resolutions, the accuracy drops off significantly. The slight difference in CPU vs. GPU accuracy may be a result of numerical precision differences in our matrix inversion and vector reduction routines.

VI. Conclusion

In this paper, we have presented an efficient ℓ_1-min solver based on augmented Lagrangian methods. The ALM solution involves alternating between optimizing the ℓ_1-min objective function in the primal and dual domains with an augmented quadratic penalty term. The complexity of the algorithm is in the same order of other first-order methods for ℓ_1-min, and is significantly lower than the traditional interior-point methods.

Since the ALM algorithm mainly consists of vector-vector and matrix-vector operations, we have investigated its parallelization on modern multi-core CPU and GPU architectures. Our experiment shows that parallelizations of ALM that solve multiple face alignment problems concurrently are significantly faster than implementations that purely rely on vendor-supplied BLAS libraries. Furthermore, thanks to a combination of faster hardware and more efficient implementations, dramatic improvements in recognition speed and limits on the image resolution are achieved over previously reported implementations.

References

[1] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. on Im. Sci.*, 2(1):183–202, 2009.
[2] D. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2003.
[3] A. Bruckstein, D. Donoho, and M. Elad. From sparse solutions of systems of equations to sparse modeling of signals and images. *(in press) SIAM Review*, 2007.
[4] E. Candès and T. Tao. Decoding by linear programming. *IEEE Transactions on Information Theory*, 51(12), 2005.
[5] S. Chen, D. Donoho, and M. Saunders. Atomic decomposition by basis pursuit. *SIAM Review*, 43(1):129–159, 2001.
[6] I. Daubechies, M. Defrise, and C. Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Comm. Pure Appl. Math.*, 57:1413–1457, 2004.
[7] D. Donoho. For most large underdetermined systems of linear equations the minimal l_1-norm solution is also the sparsest solution. *Comm. on Pure and Applied Math*, 59(6):797–829, 2006.
[8] D. Donoho and Y. Tsaig. Fast solution of ℓ^1-norm minimization problems when the solution may be sparse. *preprint*, 2006.
[9] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *The Annals of Statistics*, 32(2):407–499, 2004.
[10] R. Gross, I. Matthews, J. Cohn, T. Kanade, and S. Baker. Multi-pie. *Image and Vision Computing*, 28:807–813, 2010.
[11] S. Kim, K. Koh, M. Lustig, S. Boyd, and D. Gorinevsky. An interior-point method for large-scale ℓ_1-regularized least squares. *IEEE J. Sel. Top. Sig. Proc.*, 1(4):606–617, 2007.
[12] R. Tibshirani. Regression shrinkage and selection via the LASSO. *J. Royal Stat. Soc. B*, 58(1):267–288, 1996.
[13] A. Wagner, J. Wright, A. Ganesh, Z. Zhou, and Y. Ma. Toward a practical face recognition system: Robust alignment and illumination by sparse representation. *(accepted) PAMI*, 2011.
[14] J. Wright and Y. Ma. Dense error correction via ℓ^1-minimization. *(accepted) IEEE Trans. Info. Theory*, 2010.
[15] J. Wright, A. Yang, A. Ganesh, S. Sastry, and Y. Ma. Robust face recognition via sparse representation. *IEEE Trans. PAMI*, 31(2):210 – 227, 2009.
[16] S. Wright, R. Nowak, and M. Figueiredo. Sparse reconstruction by separable approximation. In *ICASSP*, 2008.
[17] A. Yang, A. Ganesh, S. Sastry, and Y. Ma. Fast l1-minimization algorithms and an application in robust face recognition: a review. In *Proceedings of the International Conference on Image Processing*, 2010.
[18] J. Yang and Y. Zhang. Alternating direction algorithms for ℓ_1-problems in compressive sensing. *preprint*, 2009.