Gradient projection methods for image deblurring and denoising on graphics processors

Thomas SERAFINI, Riccardo ZANELLA and Luca ZANNI
Department of Mathematics, University of Modena and Reggio Emilia, Italy

Abstract. Optimization-based approaches for image deblurring and denoising on Graphics Processing Units (GPU) are considered. In particular, a new GPU implementation of a recent gradient projection method for edge-preserving removal of Poisson noise is presented. The speedups over standard CPU implementations are evaluated on both synthetic data and astronomical and medical imaging problems.

Keywords. Image deblurring, image denoising, gradient projection methods, graphics processing units

Introduction

Image deblurring and denoising have given rise to interesting optimization problems and stimulated fruitful advances in numerical optimization techniques. Nowadays several domains of applied science, such as medical imaging, microscopy and astronomy, involve large scale deblurring problems whose variational formulations lead to optimization problems with millions of variables that should be solved in a very short time. To face these challenging problems a lot of effort has been put into designing effective algorithms that have largely improved the classical optimization strategies usually applied in image processing. Nevertheless, in many large scale applications also these improved algorithms do not provide the expected reconstruction in a suited time. In these cases, the modern multiprocessor architectures represent an important resource for reducing the reconstruction time. Among these architectures we are considering the Graphics Processing Units (GPUs), that are non-expensive parallel processing devices available on many up-to-date personal computers. Originally developed for 3D graphics applications, GPUs have been employed in many other scientific computing areas, among which signal and image reconstruction [7,9]. Recent applications show that in many cases GPUs provide performance up to those of a medium-sized cluster, at a fraction of its cost. Thus, also the small laboratories, which cannot afford a cluster, can benefit from a substantial reduction of computing time compared to a standard CPU system. Here, we deal with the GPU implementation of an optimization algorithm, called Scaled Gradient Projection (SGP) method, that applies to several imaging problems [3,11]. A GPU version of

1Corresponding Author: Thomas Serafini, Department of Mathematics, University of Modena and Reggio Emilia, Via Campi 213/B, I-41100 Modena, Italy; E-mail: thomas.serafini@unimore.it.
Choose the parameter Descent direction: Set Projection: Backtracking loop: \( \alpha \) \( M \) = 1. We observe that this line-search reduces to the standard monotone Armijo rule when \( \alpha \) method. It is worth to stress that any choice of the steplength procedure described in the Step 5, that implies convergence of the algorithm is obtained by means of the nonmonotone line-search procedure. Furthermore, let \( D \) be the set of the \( n \times n \) symmetric positive definite matrices \( D \) such that \( \|D\| \leq L \) and \( \|D^{-1}\| \leq L \), for a given threshold \( L > 1 \). The main SGP steps can be described as in Algorithm 1. At each SGP iteration the vector is defined by combining a scaled steepest descent direction with a projection onto the feasible region; it is possible to prove that the resulting search direction is defined by combining a scaled steepest descent direction with a projection onto the feasible region. We briefly recall the SGP algorithm for solving the general problem

\[
\min_{x \geq \eta} f(x)
\]

(1)

where \( \eta \in \mathbb{R} \) is a constant and \( f : \mathbb{R}^n \to \mathbb{R} \) is a continuously differentiable function, we need some basic notation. We denote by \( \| \cdot \| \) the 2-norm of vectors and we define the projection operator onto the feasible region of the problem (1) as

\[
P(x) = \arg \min_{y \geq \eta} \|y - x\|.
\]

Furthermore, let \( D \) be the set of the \( n \times n \) symmetric positive definite matrices \( D \) such that \( \|D\| \leq L \) and \( \|D^{-1}\| \leq L \), for a given threshold \( L > 1 \). The main SGP steps can be described as in Algorithm 1. At each SGP iteration the vector is defined by combining a scaled steepest descent direction with a projection onto the feasible region; it is possible to prove that the resulting search direction is defined by combining a scaled steepest descent direction with a projection onto the feasible region. We briefly recall the SGP algorithm for solving the general problem

\[
\min_{x \geq \eta} f(x)
\]

(1)

where \( \eta \in \mathbb{R} \) is a constant and \( f : \mathbb{R}^n \to \mathbb{R} \) is a continuously differentiable function, we need some basic notation. We denote by \( \| \cdot \| \) the 2-norm of vectors and we define the projection operator onto the feasible region of the problem (1) as

\[
P(x) = \arg \min_{y \geq \eta} \|y - x\|.
\]

Furthermore, let \( D \) be the set of the \( n \times n \) symmetric positive definite matrices \( D \) such that \( \|D\| \leq L \) and \( \|D^{-1}\| \leq L \), for a given threshold \( L > 1 \). The main SGP steps can be described as in Algorithm 1. At each SGP iteration the vector

\[
y^{(k)} = P(x^{(k)} - \alpha_k D_k \nabla f(x^{(k)}))
\]

is defined by combining a scaled steepest descent direction with a projection onto the feasible region; it is possible to prove that the resulting search direction is a descent direction for \( f(x) \) in \( x^{(k)} \), that is \( \Delta x^{(k)} \nabla f(x^{(k)}) < 0 \). The global convergence of the algorithm is obtained by means of the nonmonotone line-search procedure described in the Step 5, that implies \( f(x^{(k+1)}) \) lower than the reference value \( f_{\text{max}} \).

We observe that this line-search reduces to the standard monotone Armijo rule when \( M = 1 \) (\( f_{\text{max}} = f(x^{(k)}) \)). In the following, we describe the choice of the steplength \( \alpha_k \) and the scaling matrix \( D_k \), while we refer to [3] for a convergence analysis of the method. It is worth to stress that any choice of the steplength \( \alpha_k \in [\alpha_{\text{min}}, \alpha_{\text{max}}] \) and of the scaling matrix \( D_k \in D \) are allowed; then, this freedom of choice can be fruitfully
exploited for introducing performance improvements. An effective selection strategy for the steplength parameter is obtained by adapting to the context of the scaling gradient methods the Barzilai-Borwein (BB) rules [1], widely used in standard non-scaled gradient methods. When the scaled direction $D_k \nabla f(x^{(k)})$ is exploited within a step of the form $(x^{(k)} - \alpha_k D_k \nabla f(x^{(k)}))$, the BB steplength rules become

$$
\alpha_k^{(1)} = \frac{r^{(k-1)T}D_k^{-1}D_k^{-1}r^{(k-1)}}{r^{(k-1)T}D_k^{-1}z^{(k-1)}}, \quad \alpha_k^{(2)} = \frac{r^{(k-1)T}D_kz^{(k-1)}}{z^{(k-1)T}D_kD_kz^{(k-1)}},
$$

where $r^{(k-1)} = x^{(k)} - x^{(k-1)}$ and $z^{(k-1)} = \nabla f(x^{(k)}) - \nabla f(x^{(k-1)})$. The recent literature on the steplength selection in gradient methods suggests to design steplength updating strategies by alternating the two BB rules. We recall the adaptive alternation strategy used in [3], that has given remarkable convergence rate improvements in many different applications. Given an initial value $\alpha_0$, the steplengths $\alpha_k$, $k = 1, 2, \ldots$, are defined by the following criterion:

```plaintext
IF $\alpha_k^{(2)}/\alpha_k^{(1)} \leq \tau_k$ THEN
    $\alpha_k = \min \left\{ \alpha_j^{(2)}, \ j = \max \{ 1, k - M_\alpha \}, \ldots, k \right\}; \quad \tau_{k+1} = \tau_k \times 0.9;
ELSE
    $\alpha_k = \alpha_k^{(1)}; \quad \tau_{k+1} = \tau_k \times 1.1;$
ENDIF
```

where $M_\alpha$ is a prefixed non-negative integer and $\tau_1 \in (0, 1)$.

Concerning the choice of the scaling matrix $D_k$, a suited updating rule generally depends on the special form of the objective function and then we discuss this topic separately for image deblurring and denoising.

2. Optimization-based approaches for image deblurring and denoising
2.1. Image deblurring

It is well-known that the maximum likelihood approach to image deblurring in the case of Poisson noise gives an approximation of the image to be reconstructed by solving the following nonnegatively constrained minimization problem [2]:

$$
\min_{\mathbb{R}^n} \sum_{i=1}^n \left( \sum_{j=1}^n A_{ij} x_j + b_i - b_i \ln \sum_{j=1}^n A_{ij} x_j + bg \right),
\text{sub. to } x \geq 0
$$

(2)

where $b \in \mathbb{R}^n$ is the observed noisy image, $bg > 0$ denotes a constant background term and $A \in \mathbb{R}^{n \times n}$ is the blurring operator. Due to the ill-posedness of this convex problem, a regularized solution can be obtained by early stopping appropriate iterative minimization methods. Among these iterative schemes, the most popular is the Expectation Maximization (EM) method [10] that, under standard assumption on $A$, can be written as:

$$
x^{(k+1)} = X_k A^T Y_k^{-1} b = x^{(k)} - X_k \nabla J(x^{(k)}), \quad x^{(0)} > 0,
$$

(3)

where $X_k = \text{diag}(x^{(k)})$ and $Y_k = \text{diag}(A x^{(k)} + bg)$. The EM method is attractive because of its simplicity, the low computational cost per iteration and the ability to preserve the non-negativity of the iterates; however, it usually exhibits very slow convergence rate that highly limits its practical performance. When SGP is equipped with the scaling
\[ D_k = \text{diag} \left( \min \left[ L, \max \left\{ \frac{1}{L}, x^{(k)} \right\} \right] \right), \]

it can be viewed as a generalization of EM able to exploit variable steplengths for improving the convergence rate. The computational study reported in [3] shows that the fast convergence allows SGP to largely improve the EM computational time on the standard serial architectures. In Section 4.1 we report some numerical results showing that the same holds when these algorithms are implemented on GPU.

2.2. Image denoising

In order to develop a GPU implementation of SGP also for denoising problems, we follow the optimization-based approach recently proposed in [11] for the edge-preserving removal of Poisson noise. This approach consists in the minimization of a functional obtained by penalizing the Kullback-Leibler divergence by means of a special edge-preserving functional (we remark that for removing different kinds of noise, other approaches, simple and suited to parallelization, are also available [5]). Let \( y \in \mathbb{R}^n \) be the detected noisy image and \( x \) be the value associated to the \( i \)-th pixel of the image; we denote by \( x_i^+ \) and \( x_i^- \) the values associated to the pixels that are just below and above the \( i \)-th pixel, respectively; similarly, \( x_i^+ \) and \( x_i^- \) denote the values associated to the pixels that are just after and before the \( i \)-th pixel, respectively. Then, an estimate of the noise-free image can be obtained by solving the problem

\[
\min F(x) = \sum_{i=1}^{n} \left( x_i - y_i - \ln \frac{x_i}{y_i} \right) + \tau \left( \frac{1}{2} \sum_{i=1}^{n} \psi_\delta \left( \Delta_i^2 \right) \right),
\]

subject to \( x \geq \eta \)

where \( \Delta_i^2 = (x_i^+ - x_i)^2 + (x_i^- - x_i)^2 \) and \( \psi_\delta(t) = 2\sqrt{t + \delta^2} \), \( \delta \) being a small quantity tuning the discontinuities into the image, \( \eta > 0 \) is a small positive constant smaller than the background emission and \( \tau > 0 \) is a regularization parameter. In the feasible region the objective function of the problem (4) is continuously differentiable, thus the SGP method can be applied. In particular, following the gradient splitting idea presented in [6], a diagonal scaling matrix for SGP can be defined as

\[
(D_k)_{i,i} = \min \left( L, \max \left\{ \frac{1}{L}, \frac{x_i^{(k)}}{1 + \tau V_i(x^{(k)})} \right\} \right),
\]

where \( V_i(x^{(k)}) = \left\{ 2\psi_\delta' \left( \Delta_i^2 \right) + \psi_\delta' \left( \Delta_i^+ \right) + \psi_\delta' \left( \Delta_i^- \right) \right\} x_i^{(k)} \).

In the numerical experiments described in [11], the SGP equipped with the scaling matrix (5) largely outperformed other popular gradient approaches in terms of reconstruction time. Here we propose a GPU implementation of SGP and we evaluate its performance on a simulated test problem and on a denoising problem arising in medical imaging.

3. A GPU implementation of the SGP algorithm

Our GPU implementation is developed on NVIDIA graphics adapters using the CUDA (Compute Unified Device Architecture) framework for programming their GPUs [8]. By means of CUDA it is possible to program a GPU using a C-like programming language. A CUDA program contains instructions both for the CPU and the GPU. The CPU controls the instruction flow, the communications with the peripherals and it starts the single computing tasks on the GPU. The GPU performs the raw computation tasks, using
the problem data stored into the graphics memory. The GPU core is highly parallel: it is composed by a number of streaming multiprocessors, which depend on the graphics adapter model. Each streaming multiprocessor is composed by 8 cores, a high speed ram memory block, shared among the 8 cores and a cache. All the streaming multiprocessors can access to a global main memory where, typically, the problem data are stored. For our numerical experiments we used CUDA 2.0 [8] and a NVIDIA GTX 280 graphics card, which has 30 streaming multiprocessors (240 total cores) running at 1296 MHz. The total amount of global memory is 1GB and the connection bus with the cores has a bandwidth of 141.7 GB/sec. The peak computing performance is 933 GFLOPS/sec. The GPU is connected to the CPU with a PCI-Express bus, which grants a 8GB/sec transfer rate. It should be noted that this speed is much slower than the GPU-to-GPU transfer so, for exploiting the GPU performances, it is very important to reduce the CPU-GPU memory communications and keep all the problem data on the GPU memory. Besides, the full GPU-to-GPU bandwidth can be obtained only if a coalesced memory access scheme (see NVIDIA documentation [8]) is used; so, all our GPU computation kernels are implemented using that memory pattern. For the implementation of SGP, two kernel libraries of CUDA are very important: the CUFFT for the FFT computations required by the deblurring application and the CUBLAS for exploiting the main BLAS subroutines. The use of these libraries is highly recommended for maximizing the GPU performances. For the image deblurring problem, the main computational block in the SGP iteration consists in a pair of forward and backward FFTs for computing the image convolutions. We face these task as follows: after computing a 2-D real-to-complex transform, the spectral multiplication between the transformed iterate and PSF is carried out and the 2-D inverse complex-to-real transform is computed. For the image denoising problem, one of the most expensive task is the computation of the regularization term in (4). In the C implementation, the ghost cell technique is used for computing more efficiently the regularizer: according to this technique, the computational domain is extended by adding null cells on the borders. In this way, all the corner and border points can be treated as if they were interior points: this ensures a more regular instruction flow on a standard CPU. On a GPU this approach can add a computational penalty: in fact, working with a regularizer which have a domain size different than the image size, leads to a memory unalignment, and the memory accesses may not be coalesced. Bad memory accesses yield a performance loss much greater than the introduction of conditional instructions into the GPU kernels, for checking if each point is a border or a corner point; then, as usually done on a GPU, we prefer to add more cpu cycles for optimizing memory accesses. Both the deblurring and denoising cases need a division and a logarithm computation for each pixel in the image. These tasks are particularly suited for a GPU implementation: in fact there is no dependency among the pixels and the computation can be distributed on all the scalar processors available on the GPU. Besides, divisions and logarithms require a higher number of clock cycles than a simple floating point operation, thus the GPU memory bandwidth is not a limitation for these operations. Finally, we must discuss a critical part involved by the SGP: the scalar products for updating the steplength. The “reduction” operation necessary to compute a scalar product implies a high number of communications among the processors and there are dependencies that prevent a straight parallelization. In [4], NVIDIA reports an analysis of 7 different strategies for computing a reduction and suggests which is the best one for a high volume of data. In our experiments, the CUBLAS function for scalar product (cublasSdot) generally achieves the best
performance and then we exploit the kernel libraries provided by NVIDIA also for this task. Apart from the above analysis on the reduction operation, the GPU development of SGP has not required particular coding techniques since the main computational tasks of the algorithm are well suited to the architectural features of a GPU.

4. Computational results

We show the practical behaviour of the GPU implementation of SGP on the considered deblurring and denoising problems. We evaluate the speedups provided by the GPU implementations in comparison to a standard CPU implementation of the algorithm written in C. The test platform consists in a personal computer equipped with an AMD Athlon X2 Dual-Core at 3.11GHz, 3GB of RAM and the graphics processing unit NVIDIA GTX 280 described in the previous section; the codes are developed within the Microsoft Visual Studio 2005 environment. Double precision is used in the CPU version of SGP, while the mixed C and CUDA implementation exploits the single precision. The SGP parameter settings are as described in the computational study reported in [3] and [11]; in particular, the monotone version of SGP \((M = 1)\) is used.

4.1. Numerical results for deblurring problems

A deblurring test problem is obtained by considering an astronomical image corrupted by Poisson noise: the original image of the nebula NGC5979 sized 256 x 256 is convolved with an ideal PSF, then a constant background term is added and the resulting image is perturbed with Poisson noise. The original image, the blurred noisy image and a reconstruction obtained by solving the problem (2) with SGP are shown in Figure 1. Test problems of larger size are generated by expanding the original images and the PSF by means of a zero-padding technique on their FFTs. For both the SGP and EM methods, we evaluate the relative reconstruction error, defined as \(\|x^{(k)} - x\|/\|x\|\), where \(x\) is the image to be reconstructed and \(x^{(k)}\) is the reconstruction after \(k\) iterations; then we report the minimum relative reconstruction error (err.), the number of iterations (it.) and the computational time in seconds (time) required to provide the minimum error. The results in Table 1 show that the GPU implementations allow us to save time over the CPU implementation for more than one order of magnitude. Furthermore, SGP largely outperforms EM also on graphics processors, even if the additional operations required in each SGP iteration (in particular the scalar products involved in the steplength selection) make this algorithm less suited than EM for a parallel implementation on GPU. We
Table 1. Behaviour of EM and SGP on deblurring problems

<table>
<thead>
<tr>
<th>n</th>
<th>CPU (C-based)</th>
<th>GPU (C_CUDA-based)</th>
<th>CPU (C-based)</th>
<th>GPU (C_CUDA-based)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>err. time</td>
<td>err. time</td>
<td>Speedup</td>
<td>err. time</td>
</tr>
<tr>
<td>256</td>
<td>0.070 0.72</td>
<td>0.071 0.05</td>
<td>14.7 0.070 4.41</td>
<td>0.071 0.19</td>
</tr>
<tr>
<td>512</td>
<td>0.065 2.69</td>
<td>0.065 0.16</td>
<td>16.8 0.064 19.91</td>
<td>0.064 0.89</td>
</tr>
<tr>
<td>1024</td>
<td>0.064 10.66</td>
<td>0.064 0.58</td>
<td>18.4 0.063 97.92</td>
<td>0.063 3.63</td>
</tr>
<tr>
<td>2048</td>
<td>0.064 49.81</td>
<td>0.063 2.69</td>
<td>18.5 0.063 523.03</td>
<td>0.063 23.05</td>
</tr>
</tbody>
</table>

Table 2. Behaviour of SGP on denoising problems.

<table>
<thead>
<tr>
<th>Test problem</th>
<th>n</th>
<th>CPU (C-based)</th>
<th>GPU (C_CUDA-based)</th>
<th>CPU (C-based)</th>
<th>GPU (C_CUDA-based)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>it.</td>
<td>err. time</td>
<td>it.</td>
<td>err. time</td>
<td>Speedup</td>
</tr>
<tr>
<td>Phantom</td>
<td>256²</td>
<td>83</td>
<td>1.27 79</td>
<td>0.025</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td>512²</td>
<td>97</td>
<td>5.89 73</td>
<td>0.018</td>
<td>0.16</td>
</tr>
<tr>
<td></td>
<td>1024²</td>
<td>60</td>
<td>15.13 85</td>
<td>0.014</td>
<td>0.53</td>
</tr>
<tr>
<td></td>
<td>2048²</td>
<td>62</td>
<td>64.45 87</td>
<td>0.011</td>
<td>1.88</td>
</tr>
<tr>
<td>Dental radiography</td>
<td>512²</td>
<td>28</td>
<td>1.86 25</td>
<td>0.029</td>
<td>0.06</td>
</tr>
</tbody>
</table>

Refer to [3] for more details on the comparison between EM, SGP and other well known deblurring methods; furthermore, other examples of deblurring problems solved by the GPU implementation of SGP can be found in [9].

4.2. Numerical results for denoising problems

Two denoising test problems are considered: the former is generated from the phantom described in [11], consisting in circles of intensities 70, 135 and 200, enclosed by a square frame of intensity 10, all on a background of intensity 5; the latter is derived from a dental radiography (gray levels between 3 and 64). The corresponding images corrupted by Poisson noise are shown in Figure 2, together with the original images and the reconstructions obtained by applying SGP to the problem (4) with \( \eta = 10^{-3} \). The reconstructions are obtained by a careful tuning of the parameters \( \tau \) and \( \delta \): the values \( \tau = 0.25, \delta = 0.1 \) for the phantom image and \( \tau = 0.3, \delta = 1 \) for the dental radiography are used. The SGP behaviour on both CPU and GPU is described in Table 2. Here the number of iterations refers to the number of SGP steps required to satisfy the criterion \( |F(x^k) - F(x^{k-1})| \leq 10^{-6}|F(x^k)| \); the differences between the CPU and GPU iterations are due to the different precision and organization of the computations used in the two environments. However, it is important to observe that the relative reconstruction errors are very similar on the two architectures. The speedups are significantly better than in the case of the deblurring problems since now the computation of the denoising objective function and of its gradient are dominated by simple pixel-by-pixel operations (no image convolutions are required) that are well suited for an effective implementation on the graphics hardware.

5. Conclusions

GPU implementations of a gradient projection method for image deblurring and denoising have been discussed. For both the imaging problems a time saving of more than one order of magnitude over standard CPU implementations has been observed. In particular, the GPU implementation presented in this work for denoising problems achieves very
promising speedups and confirms that the considered optimization-based approaches are well suited for facing large imaging problems on graphics processors. Work is in progress about the extension of the proposed algorithms and their GPU implementations to related optimization problems in signal and image restoration (compressed sensing, blind deconvolution, multiple image restoration).

References