A GPU-Accelerated Augmented Lagrangian Based L^1-mean Curvature Image Denoising Algorithm Implementation

Mirko Myllykoski[1]
University of Jyväskylä
mirko.myllykoski@jyu.fi

Roland Glowinski[2]
University of Houston
roland@math.uh.edu

Tommi Kärkkäinen[1]
University of Jyväskylä
tommi.karkkainen@jyu.fi

Tuomo Rossi[1]
University of Jyväskylä
tuomo.j.rossi@jyu.fi

Abstract

This paper presents a graphics processing unit (GPU) implementation of a recently published augmented Lagrangian based L^1-mean curvature image denoising algorithm. The algorithm uses a particular alternating direction method of multipliers to reduce the related saddle-point problem to an iterative sequence of four simpler minimization problems. Two of these subproblems do not contain the derivatives of the unknown variables and can therefore be solved point-wise without inter-process communication. In particular, this facilitates the efficient solution of the subproblem that deals with the non-convex term in the original objective function by modern GPUs. The two remaining subproblems are solved using the conjugate gradient method and a partial solution variant of the cyclic reduction method, both of which can be implemented relatively efficiently on GPUs. The numerical results indicate up to 33-fold speedups when compared against a single-threaded CPU implementation. The pointwise treated subproblem that takes care of the non-convex term in the original objective function was solved up to 76 times faster.

Keywords

augmented Lagrangian method, GPU computing, image denoising, image processing, mean curvature, OpenCL

1 INTRODUCTION

Image denoising, or more generally noise reduction, is a process in which a given noisy signal, such as a digital image, is cleared from excess noise. This process has numerous applications since all recording devices have some traits that make them susceptible to interference. For example, thermal noise effecting digital image sensors is a typical source of interference in digital photography. The noise must be removed, or at least significantly reduced, before essential information can be successfully extracted from an image.

Image denoising methods are divided into multiple subcategories. For example, wavelet methods are based around the idea of decomposing the image into the wavelet basis and shrinking (or otherwise modifying) the wavelet coefficients in order to denoise the image. A somewhat similar approach is to process the image in the frequency domain using the fast Fourier transformation (FFT) method. Statistical methods, on the other hand, utilize local statistical information, such as median and mean, from the neighboring pixels that fall within an appropriately selected window.

The most relevant sub-category to the topic of this paper is referred to as variational-based methods. These methods treat the noisy image as a discretely differentiable function and denoise the image using derivate information. In more formal terms, let Ω be a rectangular domain of $\mathbb{R}^2$ and the function $f : \Omega \to \mathbb{R}$ represent the given noisy image. We want to find a function $u : \Omega \to \mathbb{R}$ that is a representative of the desired denoised image. A variety of variational-based techniques have been developed and a significant proportion of them (see, e.g., [Mum94, Rud92]) are based on solving an unconstrained minimization problem of the form

$$\begin{cases} u \in V, \\ \mathscr{J}(u) \leq \mathscr{J}(v), \forall v \in V, \end{cases} \tag{1}$$

where

$$\mathscr{J}(v) = \varepsilon \mathscr{J}_r(v) + \mathscr{J}_f(v), \tag{2}$$

V is a suitable function space, and $\varepsilon > 0$. Here $\mathscr{J}_r$ is so-called regularization term and $\mathscr{J}_f$ is so-called fidelity

[1] University of Jyväskylä, Department of Mathematical Information Technology, P.O. Box 35, FI-40014 University of Jyväskylä, Finland.
[2] University of Houston, Department of Mathematics, Houston, TX 77204, USA.

term whose role is to fit the obtained solution u to the noisy data f. The work in this field of research is aimed primarily at finding a suitable regularization term that is able to detect noise but preserves as much relevant information as possible.

During the last two decades, the variational-based image denoising scene has been dominated to a large extent by Rudin–Osher–Fatemi (ROF) method [Rud92] which uses the following objective function:

$$\mathscr{J}(v) = \varepsilon \int_\Omega |\nabla v|\, dx + \frac{1}{2}\int_\Omega |f-v|^2\, dx. \qquad (3)$$

Here, $\int_\Omega |\nabla v|\, dx$ is so-called total variation norm (TV norm) and the function space V made out of function whose total variation is bounded (a.k.a BV space). This very popular method has, however, some well-known drawbacks, such as the loss of image contrast, the smearing of corners, and the staircase effect.

Some attempts to remedy these drawbacks have led to higher-order variational-models which seek to take advantage of the higher-order derivatives. One approach suggested by Zhu and Chan [Zhu12] is to treat an image $v : \Omega \to \mathbb{R}$ as a surface in $\Omega \times \mathbb{R}$ and utilize the surface mean curvature information in the regularization term. More specifically, the surface in question is defined by the equation $F_v(x,y,z) = v(x,y) - z = 0$ and the mean curvature of the function F_v is given by

$$\kappa(F_v) = -\nabla\cdot \frac{\nabla F_v}{|\nabla F_v|} = -\nabla\cdot \frac{\nabla v}{\sqrt{1+|\nabla v|^2}}. \qquad (4)$$

All in all, the objective function used in the model suggested by Zhu and Chan is of the form:

$$\mathscr{J}(v) = \varepsilon \int_\Omega |\kappa(F_v)|\, dx + \frac{1}{2}\int_\Omega |f-v|^2\, dx. \qquad (5)$$

This model is commonly known these days as the L^1-mean curvature denoising model. As noted in [Zhu12], this model has the ability to remove noise without the undesirable drawbacks associated with the ROF model. However, the non-convex and non-smooth nature of the objective function (5) makes the problem very difficult to solve.

1.1 Related work

Although the model suggested by Zhu and Chan is very difficult to solve as noted above, effective solution algorithms for this particular formulation have been proposed, for example, in [Zhu13] and [Myl15]. The solution algorithm presented in [Zhu13] uses an augmented Lagrangian based approach and solves the arising saddle-point problem using a particular alternating direction method of multipliers. This leads to an iterative sequence of five simpler minimization problems. These subproblems can be solved using explicit formulas and the FFT method. The solution algorithm presented in [Myl15] uses the same alternating direction approach as [Zhu13] but relies on a different type of augmented Lagrangian functional. Two of the four arising subproblems can be solved pointwise using the Newton's method, a bisection search algorithm, and explicit formulas. The two remaining subproblems can be solved using the conjugate gradient method and a partial solution variant of the cyclic reduction (PSCR) method [Kuz85, Kuz96, Vas84, Val85].

It should be noted that the model suggested by Zhu and Chan is closely related to the model depicted in [Lys04]. The model uses the following regularization term:

$$\mathscr{J}_r(v) = \int_\Omega \left|\nabla\cdot \frac{\nabla v}{|\nabla v|}\right| dx. \qquad (6)$$

In [Lys04], the authors explained that their goal was to minimize the "TV norm" of the unit normal vectors of the level curves of the image. An alternative interpretation is that the regularization term (6) measures the total mean curvature at every level curve of the image. In contrast, the regularization term in the model suggested by Zhu and Chan measures the total mean curvature at the surface defined by the image (graph).

From a practical point of view the most relevant connection comes from the fact that the resulting model is often regularized in such a way that $|\nabla v|$ in (6) is replaced by $|\nabla v|_\beta = \sqrt{|\nabla v|^2 + \beta}$, $\beta > 0$. Thus, the solution algorithms developed for this denoising model and its variants (see, e.g., [Bri10, Sun14, Yan14]) could be in principle generalized for the model suggested by Zhu and Chan by taking $\beta = 1$.

The solution algorithm presented in [Bri10] solves the related Euler-Lagrange (EL) equation using a stabilized fixed point method and a geometric multigrid (MG) algorithm. The authors in [Sun14] aimed to improve upon that by introducing an additional operator splitting step. They then moved on to solving the EL equations associated with the related constrained minimization problem using a linearized fixed point method and a nonlinear MG method. In [Yan14], the problem is tacked with a relaxed fixed point method and a homotopy algorithm. The papers [Bri10, Sun14, Yan14] included comparisons where the value of the parameter β was varied (including the case $\beta = 1$). In other aspects these three recent papers were mostly interested in the case where the regularization term (6) is replaced by

$$\mathscr{J}_r(v) = \int_\Omega \left(\nabla\cdot \frac{\nabla v}{|\nabla v|_\beta}\right)^2 dx. \qquad (7)$$

and $\beta \ll 1$.

1.2 Motivation and structure

Now that the overall context and main related works have been dealt with, we can move on to the main topic of this paper. We present a graphics processing unit (GPU) implementation of the solution algorithm depicted in [Myl15] and compare the GPU implementation against a single-threaded CPU implementation.

Our main motivation is that the most demanding step in the solution algorithm is a pointwise treated subproblem that handles the non-convex term in the original objective function. This makes the solution algorithm very suitable for GPU computation since the solution of this subproblem does not require inter-process communication. Thus, it is very likely that GPU-acceleration would bring significant performance benefits.

The rest of this paper is organized as follows: Section 2 describes the augmented Lagrangian based image denoising algorithm closely following the presentation in [Myl15]. Section 3 gives a brief introduction to GPU computing and describes the GPU implementation. Section 4 presents the numerical results, comparisons, and discussion. The final conclusions are given in Section 5.

2 SOLUTION ALGORITHM

2.1 Augmented Lagrangian formulation

Augmented Lagrangian techniques are a well-established framework for analyzing (constrained) optimization problems and deriving solution algorithms for such problems. When applied to convex minimization problems, the basic idea is to decompose the problem with the help of auxiliary variables. This so-called operator splitting operation effectively splits the problem into subproblems which can be treated separately using methods that are best suited for each subproblem. This greatly improves the effectiveness of the resulting solution algorithm.

The addition of these new auxiliary variables leads to a new constrained minimization problem that has the same minimizer as the original minimization problem. This constrained minimization is then associated with a suitable augmented Lagrangian functional whose saddle-point correspond to the minimizer of the constrained minimization problem. The saddle-points can be solved by, for example, using an alternating direction type approach. See, for example, [For83] for further information.

Although the objective function in the model suggested by Zhu and Chan is not convex, we will now describe a formal augmented Lagrangian formulation for the minimization problem similarly to [Myl15]. To begin with, let us define

$$\begin{aligned}\Upsilon = [V \times \mathbf{E}_{12} \times H(\Omega;\mathrm{div}) \times L^2(\Omega)] \\ \times [(L^2(\Omega))^2 \times (L^2(\Omega))^2 \times L^2(\Omega)],\end{aligned} \tag{8}$$

where

$$H(\Omega;\mathrm{div}) = \left\{\mathbf{q} \in (L^2(\Omega))^2 : \nabla\cdot\mathbf{q} \in L^2(\Omega)\right\} \tag{9}$$

and

$$\begin{aligned}\mathbf{E}_{12} = \Bigg\{ &(\mathbf{q}_1,\mathbf{q}_2) \in \left(L^2(\Omega)\right)^{2\times 2} \\ &: \mathbf{q}_2 = \frac{\mathbf{q}_1}{\sqrt{1+|\mathbf{q}_1|^2}} \Bigg\}.\end{aligned} \tag{10}$$

Following the remarks made in [Myl15], we take $V = H^2(\Omega)$. The minimization problem (1) with $\mathscr{J}$ defined by (5) is associated with the following augmented Lagrangian functional $\mathscr{L} : \Upsilon \to \mathbb{R}$:

$$\begin{aligned}&\mathscr{L}(v,\mathbf{q}_1,\mathbf{q}_2,\mathbf{q}_3,\varphi;\boldsymbol{\mu}_1,\boldsymbol{\mu}_2,\mu_3) \\ &\quad= \varepsilon\int_\Omega |\varphi| dx + \frac{1}{2}\int_\Omega |f-v|^2\,dx \\ &\quad+ \frac{r_1}{2}\int_\Omega |\nabla v - \mathbf{q}_1|^2\,dx + \int_\Omega \boldsymbol{\mu}_1\cdot(\nabla v - \mathbf{q}_1)\,dx \\ &\quad+ \frac{r_2}{2}\int_\Omega |\mathbf{q}_2 - \mathbf{q}_3|^2\,dx + \int_\Omega \boldsymbol{\mu}_2\cdot(\mathbf{q}_2 - \mathbf{q}_3)\,dx \\ &\quad+ \frac{r_3}{2}\int_\Omega |\nabla\cdot\mathbf{q}_3 - \varphi|^2\,dx \\ &\quad+ \int_\Omega \mu_3(\nabla\cdot\mathbf{q}_3 - \varphi)\,dx,\end{aligned} \tag{11}$$

where $(\mathbf{q}_1,\mathbf{q}_2) \in \mathbf{E}_{12}$ and $r_i > 0$, $i = 1,2,3$. Above, $\mathbf{q}_1$, $\mathbf{q}_2$, $\mathbf{q}_3$, and φ are the previously mentioned auxiliary variables, and $\boldsymbol{\mu}_1$,$\boldsymbol{\mu}_2$, and μ_3 are called Lagrange multipliers. Note that the non-convex term $\mathbf{q}_2 = \frac{\mathbf{q}_1}{\sqrt{1+|\mathbf{q}_1|^2}}$ is treated by projection in (10) and thus does not appear in the augmented Lagrangian functional $\mathscr{L}$.

Now, if we can find a saddle-point

$$\omega = (u,\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3,\psi;\boldsymbol{\lambda}_1,\boldsymbol{\lambda}_2,\lambda_3) \in \Upsilon \tag{12}$$

for the augmented Lagrangian $\mathscr{L}$, that is

$$\begin{aligned}&\mathscr{L}(u,\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3,\psi;\boldsymbol{\mu}_1,\boldsymbol{\mu}_2,\mu_3) \\ &\le \mathscr{L}(u,\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3,\psi;\boldsymbol{\lambda}_1,\boldsymbol{\lambda}_2,\lambda_3) \\ &\le \mathscr{L}(v,\mathbf{q}_1,\mathbf{q}_2,\mathbf{q}_3,\varphi;\boldsymbol{\lambda}_1,\boldsymbol{\lambda}_2,\lambda_3),\end{aligned} \tag{13}$$

for all $(v,\mathbf{q}_1,\mathbf{q}_2,\mathbf{q}_3,\varphi;\boldsymbol{\mu}_1,\boldsymbol{\mu}_2,\mu_3) \in \Upsilon$, then

$$\begin{aligned}\mathbf{p}_1 = \nabla u,\ \mathbf{p}_2 = \frac{\mathbf{p}_1}{\sqrt{1+|\mathbf{p}_1|^2}}, \\ \mathbf{p}_3 = \mathbf{p}_2,\ \psi = \nabla\cdot\mathbf{p}_3,\end{aligned} \tag{14}$$

and, more importantly, u is a local minimizer of the minimization problem (1) with $\mathscr{J}$ defined by (5).

2.2 Subproblems

In [Myl15], the saddle-point problem (12) – (13) is solved using a particular alternating direction method

of multipliers called ALG-2 [For83, Glo89]. The idea is to minimize the augmented Lagrangian functional (11) one variable at a time until the method converges. The Lagrange multipliers are update accordingly after each iteration.

Since we have five variables and two of the auxiliary variables ($\mathbf{q}_1$ and $\mathbf{q}_2$) are coupled together, this leads to an iterative sequential solution of four subproblems. More precisely, the task of finding a saddle-point for the augmented Lagrangian functional (11) is transformed into one smooth but nonlinear and non-convex minimization problem in $\mathbb{R}^2$, one purely explicit pointwise treated minimization problem, and two linear minimization problems with positive definite and symmetric coefficient matrices.

Each outer iteration is defined as follows: Let $(u^n, \mathbf{p}_1^n, \mathbf{p}_2^n, \mathbf{p}_3^n, \psi^n; \boldsymbol{\lambda}_1^n, \boldsymbol{\lambda}_2^n, \lambda_3^n) \in \Upsilon$ be the output of the previous iteration. *The first subproblem* minimizes the augmented Lagrangian functional (11) with respect to the pair $(\mathbf{q}_1, \mathbf{q}_2) \in \mathbf{E}_{12}$. Using the nonlinear relation in (10), the function $\mathbf{p}_1^{n+1}$ can be solved pointwise from the following non-convex minimization problem:

$$\mathbf{x} = \arg\min_{\mathbf{y}\in\mathbb{R}^2} \left[\frac{|\mathbf{y}|^2}{2} \left(r_1 + \frac{r_2}{1+|\mathbf{y}|^2} \right) - \left(\mathbf{b}_1 + \frac{\mathbf{b}_2}{\sqrt{1+|\mathbf{y}|^2}} \right) \cdot \mathbf{y} \right], \tag{15}$$

where $\mathbf{b}_1, \mathbf{b}_2 \in \mathbb{R}^2$ depend on the other variables. Or, alternatively, by noticing that the following nonlinear relation must hold

$$\mathbf{x} = \alpha \left(\mathbf{b}_1 + \frac{\mathbf{b}_2}{\sqrt{1+|\mathbf{x}|^2}} \right), \tag{16}$$

where $\alpha \geq 0$, we can write the two-dimensional minimization problem (15) as

$$\mathbf{x} = \frac{\rho}{\left| \mathbf{b}_1 + \frac{\mathbf{b}_2}{\sqrt{1+\rho^2}} \right|} \left(\mathbf{b}_1 + \frac{\mathbf{b}_2}{\sqrt{1+\rho^2}} \right), \tag{17}$$

where

$$\rho = \arg\min_{\sigma\in[0,\infty)} \left[\frac{\sigma^2}{2} \left(r_1 + \frac{r_2}{1+\sigma^2} \right) - \sigma \left| \mathbf{b}_1 + \frac{\mathbf{b}_2}{\sqrt{1+\sigma^2}} \right| \right]. \tag{18}$$

In *the second subproblem* we minimize the augmented Lagrangian functional (11) with respect to the variable $\mathbf{q}_3$. More specifically, we solve the following linear vector-valued minimization problems with positive definite and symmetric coefficient matrix:

$$\begin{aligned} \mathbf{p}_3^{n+1} &\in H(\Omega; \mathrm{div}), \\ r_2 \int_\Omega \mathbf{p}_3^{n+1} \cdot \mathbf{q}\, dx + r_3 \int_\Omega \nabla\cdot \mathbf{p}_3^{n+1} \nabla\cdot \mathbf{q}\, dx &= \int_\Omega \left(r_2 \mathbf{p}_2^{n+1} + \boldsymbol{\lambda}_2^n \right) \cdot \mathbf{q}\, dx \\ &+ \int_\Omega (r_3 \psi^n - \lambda_3^n) \nabla\cdot \mathbf{q}\, dx, \\ \forall \mathbf{q} &\in H(\Omega; \mathrm{div}). \end{aligned} \tag{19}$$

The third subproblem minimizes the augmented Lagrangian functional (11) with respect to the variable φ and is of the form:

$$\psi^{n+1} = \arg\min_{\varphi\in L^2(\Omega)} \left[\varepsilon \int_\Omega |\varphi|\, dx + \frac{r_3}{2} \int_\Omega |\varphi|^2\, dx - \int_\Omega \left(r_3 \nabla\cdot \mathbf{p}_3^{n+1} + \lambda_3^n \right) \varphi\, dx \right]. \tag{20}$$

The minimization problem (20) has a closed-form solution

$$\psi^{n+1}(x) = \frac{1}{r_3} \operatorname{sgn}(\xi(x)) \max(0, |\xi(x)| - \varepsilon), \tag{21}$$

with $\xi(x) = (r_3 \nabla\cdot \mathbf{p}_3^{n+1} + \lambda_3^n)(x)$.

The fourth subproblem minimizes the augmented Lagrangian functional (11) with respect to the variable v. The subproblem can be written as the following linear scalar-valued minimization problems with positive definite, symmetric, and separable coefficient matrix:

$$\begin{aligned} u^{n+1} &\in V, \\ r_1 \int_\Omega \nabla u^{n+1} \cdot \nabla v\, dx + \int_\Omega (u^{n+1} - f)\, v\, dx &= \int_\Omega \left(r_1 \mathbf{p}_1^{n+1} - \boldsymbol{\lambda}_1^n \right) \cdot \nabla v\, dx, \forall v \in V. \end{aligned} \tag{22}$$

Finally, *the Lagrange multipliers* are updated as follows:

$$\begin{aligned} \boldsymbol{\lambda}_1^{n+1} &= \boldsymbol{\lambda}_1^n + r_1(\nabla u^{n+1} - \mathbf{p}_1^{n+1}), \\ \boldsymbol{\lambda}_2^{n+1} &= \boldsymbol{\lambda}_2^n + r_2(\mathbf{p}_2^{n+1} - \mathbf{p}_3^{n+1}), \\ \lambda_3^{n+1} &= \lambda_3^n + r_3(\nabla\cdot \mathbf{p}_3^{n+1} - \psi^{n+1}). \end{aligned} \tag{23}$$

2.3 Finite element realization

The domain Ω is triangulated using a uniform finite element triangulation $\mathscr{T}_h$. The function space V is approximated by a piecewise linear finite element space

$$V_h = \left\{ v \in C^0(\bar{\Omega}) : v|_T \in P_1,\ \forall\, T \in \mathscr{T}_h \right\}, \tag{24}$$

where P_1 is the space of the polynomials of two variables of degree ≤ 1. The spaces $(L^2(\Omega))^2$ and

$H(\Omega;\mathrm{div})$ are approximated by the following piecewise constant finite element space:

$$\mathbf{Q}_h = \left\{ \mathbf{q} \in (L^\infty)^2 : \mathbf{q}|_T \in (P_0)^2, \forall\, T \in \mathscr{T}_h \right\}, \qquad (25)$$

where P_0 is the space of the constant functions. Clearly, we have $\nabla V_h \subset \mathbf{Q}_h$.

Let $\{X_j\}_{j=1}^{N_h}$ be the set of vertices of $\mathscr{T}_h$ and $\mathbf{q} \in \mathbf{Q}_h$. The divergence operator is approximated by using an appropriate discrete Green's formula and the trapezoidal rule as follows:

$$(\mathrm{div}_h \mathbf{q})(X_j) = -\frac{3}{|\Omega_j|} \int_{\Omega_j} \mathbf{q} \cdot \nabla w_j \, dx, \qquad (26)$$

where X_j is a vertex that does not belong to $\partial\Omega$, Ω_j is the polygon that is the union of those triangles of $\mathscr{T}_h$ that have X_j as a common vertex, $|\Omega_j|$ is the measure of Ω_j, and the shape function $w_j \in V_h$ is uniquely defined as

$$\begin{cases} w_j(X_j) = 1, \\ w_j(X_k) = 0,\ k \neq j. \end{cases} \qquad (27)$$

3 GPU IMPLEMENTATION

3.1 GPU computing and OpenCL

The GPU implementation presented in this paper is written using the OpenCL framework. This section introduces the reader to general OpenCL concepts and terminology. Some additional information related to Nvidia's current hardware is provided since that information is essential for the understanding of the implementation and obtained numerical results.

A contemporary high-end GPU contains thousands of processing elements (cores) which are grouped into multiple computing units. The processing elements inside the same computing unit share a fast (on-chip) memory space called local memory which can be used for sharing data among the processing elements. The local memory is divided into 32-bit (or 64-bit) memory banks organized in such a way that successive 32-bit (or 64-bit) words map to successive memory banks. Multiple processing elements also share the same scheduler, which means that the processing elements are executing the program code in a synchronous manner. In addition to the local memory, all processing elements can access a much larger but slower (off-chip) memory space called global memory. The global memory can serve memory requests at the optimal rate when processing elements are synchronously accessing data that is located inside a same memory block.

GPU-side program code execution is based on the concept of a special kind of subroutine called (OpenCL) kernel. All work-items (threads) start from the beginning of the kernel but each work-item is given a unique index number which allows the execution paths of different work-items to branch off. The work-items are grouped into work groups which are also given unique index numbers and a work group can share a portion of the local memory. Nvidia uses the term warp when referring to a set of work-items that are executed together in a synchronized manner. Diverging execution paths, also known as warp divergences, lead to a suboptimal performance as all the necessary paths have to be evaluated by the whole warp. In contemporary Nvidia GPUs the warp size is 32 work-items.

3.2 General notes

The GPU implementation is principally identical with the CPU implementation described in [Myl15] but the low level details vary considerably. The less simplified two-dimensional form of the critical non-convex subproblem (15) is initially solved using the Newton's method, whose solution candidate is then tested against the explicit relation (16). If the solution candidate does not fulfill the explicit relation, the implementation proceeds to the one-dimensional form (18) which is solved using the bisection search algorithm as described in [Myl15].

The linear vector-valued subproblem (19) is solved using the conjugate gradient algorithm without preconditioning. While more generalized GPU implementations have been presented in the past (see, for example, [Ame10, Bol03, Hel12]), the conjugate gradient solver used in the GPU implementation described in this paper was tailored for this specific subproblem and the matrix-vector multiplication operation was hard coded into the kernels. The explicit subproblem (20) is solved using the closed form solution (21) and the linear scalar-valued subproblem (22) is solved using the PSCR method.

All computational operations are carried out in the GPU side. The floating point division operation was accelerated using a Newton-Raphson division algorithm [Fly70] and an initial approximation that leads to full double precision accuracy with only four iterations [Par92].

3.3 Element numbering

The elements of the finite element space V_h are numbered in a row-wide fashion. This means that the coefficient matrix in the linear scalar-valued subproblem (22) is block tridiagonal and presentable in a separable form using so-called Kronecker matrix tensor product. This is required by the PSCR method.

The numbering of the elements of the finite element space $\mathbf{Q}_h$ can be chosen more freely. Figure 1 shows two possible numbering schemes. If the numbering scheme shown on the left (referred to hereinafter as

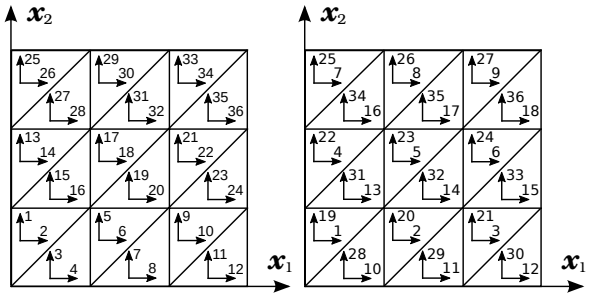

Figure 1: Two possible numbering schemes for the elements of the finite element space $\mathbf{Q}_h$ (3×3 grid): the dense numbering scheme (on the left) and the sparse numbering scheme (on the right).

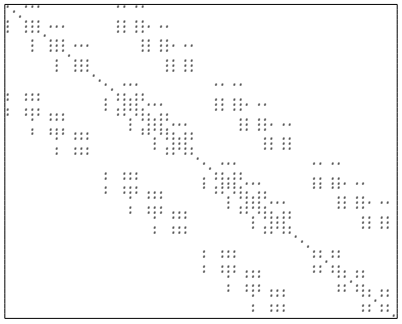

Figure 2: The non-zero elements of the coefficient matrix in the linear vector-valued subproblem (19) when the dense numbering scheme is used (4×4 grid).

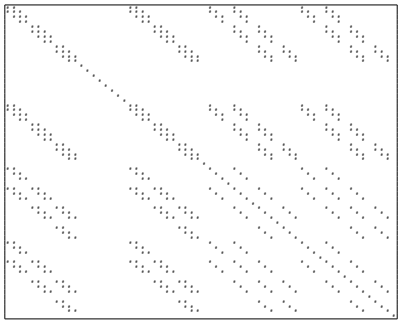

Figure 3: The non-zero elements of the coefficient matrix in the linear vector-valued subproblem (19) when the sparse numbering scheme is used (4×4 grid).

the dense numbering scheme) is chosen, then the coefficient matrix in the linear vector-valued subproblem (19) is of the form shown in Figure 2. On the one hand, if the numbering scheme shown on the right (referred to hereinafter as the sparse numbering scheme) is chosen, then the coefficient matrix is of the form shown in Figure 3. Each numbering scheme has its own advantages and disadvantages.

The dense numbering scheme leads to a more optimal global memory access pattern during the solution of the linear vector-valued subproblem (19) as the non-zero elements of the coefficient matrix are packed tightly to three bands and elements of each band can be shared among work-items using the local memory. This is particularly important because many contemporary high end GPUs have an extremely high peak floating-point performance but a relatively low peak global memory bandwidth. Thus, the use of the global memory should be kept at minimum. The most significant downside of this numbering scheme is that a straightforward implementation would lead to warp divergences throughout the implementation. Most of these warp divergences could be avoided by re-arranging the computational tasks appropriately with the help of the local memory. However, this re-arranging would complicate the implementation considerably and introduce memory bank conflicts in many places.

The sparse numbering scheme leads to a simpler implementation but the pattern of non-zero elements in the coefficient matrix is much more fragmented. This means that less data can be shared between the work-items using the local memory and, thus, the global memory usage increases significantly. Despite this, the sparse numbering scheme was chosen for the GPU implementation described in this paper because it was not clear whether this choice would lead to an actual global memory bottleneck that would limit the performance of the whole GPU implementation. In addition, if the dense numbering scheme is chosen, then the increased complexity in the other parts of the implementation might negate the potential benefits. The reference CPU implementation uses the dense numbering scheme because it allows more effective utilization of the CPU caches.

3.4 PSCR implementation

The PSCR method [Kuz85, Kuz96, Vas84, Val85] is a block cyclic reduction type direct solver which can be applied to certain separable block tridiagonal linear systems. To put it briefly, the PSCR method solves the linear scalar-valued subproblem (22) by recursively eliminating block-rows from the corresponding linear system and then solves the generated sub-systems in the reverse order during so-called back substitution stage. Each reduction and back substitution step produces a large set of tridiagonal linear system.

The GPU implementation of the PSCR method used in this paper is based on the radix-4 variant described in [Ros99] and it is in many respects similar to the simplified radix-4 block cyclic reduction GPU implementation presented in [Myl13]. However, the GPU implementation used in this paper is much more generalized as the problem size can be arbitrary.

The arising tridiagonal subproblems are solved using the cyclic reduction (CR) [Hoc65], the parallel cyclic reduction (PCR) [Hoc81] and the Thomas [Con80] methods. If a tridiagonal system does not fit into the allocated local memory buffer,then the system size is first reduced using the CR method and the global

memory data permutations depicted in [Myl13]. The tridiagonal systems that do fit into the allocated local memory buffer are solved using a CR-PCR-Thomas hybrid method. The CR stage of the hybrid solver uses the local memory data permutations depicted in [Myl13]. The PCR stage further splits the reduced tridiagonal systems into smaller subsystems which are eventually solved using the Thomas method in a manner similar to [Dav11, Kim11]. Somewhat similar tridiagonal solver techniques have been used, for example, in [Göd11, Lam12, Zha10].

4 NUMERICAL RESULTS

4.1 Test setting

GPU tests were carried out on a few years old consumer level Nvidia GeForce GTX580 GPU and a high-end computing orientated Nvidia Tesla K40c GPU. The CPU implementation is the same as was used in [Myl15]. It is written using C++ and Fortran. It utilizes a single-threaded variant of the radix-4 PSCR method presented in [Ros99]. CPU tests were carried out using an Intel Xeon E5-2630 v2 (2.60GHz) CPU. All the tests were performed using double precision floating point arithmetic and ECC memory (excluding the GTX580 GPU which does not support ECC).

Four test images (shown in Figure 4) were used in the numerical tests: Test9, Lena, Boat, and Mercado. In addition, four different sized versions of the test image Mercado were included: 256×256, 512×512, 1024×1024, and 2048×2048. The dimensions of the other test images are 512×512. The original test images were scaled to the range $[0,1]$. Two sets of noisy input images were generated: uniformly distributed zero-mean noise with the standard deviation $\sigma = 0.025$ and uniformly distributed zero-mean noise with the standard deviation $\sigma = 0.1$.

Based on the remarks made in [Myl15], the following initialization was used:

$$u^0 = 0,\ \mathbf{q}_1^0 = \mathbf{q}_2^0 = \mathbf{q}_3^0 = 0,\ \psi^0 = 0,\quad \boldsymbol{\lambda}_1^0 = \boldsymbol{\lambda}_2^0 = 0,\ \lambda_3^0 = 0. \tag{28}$$

In the same way, the stopping criterion read as

$$\frac{|\mathscr{L}_h(\upsilon^n) - \mathscr{L}_h(\upsilon^{n+1})|}{|\mathscr{L}_h(\upsilon^n)|} < 10^{-4}, \tag{29}$$

where $\mathscr{L}_h$ is the discrete counterpart of the augmented Lagrangian functional (11) and $\upsilon^n = (u^n, \mathbf{p}_1^n, \mathbf{p}_2^n, \mathbf{p}_3^n, \psi^n; \boldsymbol{\lambda}_1^n, \boldsymbol{\lambda}_2^n, \lambda_3^n)$. The parameter ε and the Lagrangian multipliers were coupled as follows: $\varepsilon = r_0 h$, $r_1 = 10 r_0 h$, $r_2 = 5 r_0$, and $r_3 = 5 r_0 h^2$, where $r_0 > 0$. We took $h = 0.005$ for the spatial discretization step.

4.2 Comparisons

Figure 4 shows the original test images, the generated input images ($\sigma = 0.1$), and the obtained output images. Table 1 shows the used parameter values, iteration counts and execution times for the Intel Xeon CPU. The input images and parameter values were the same for the three platforms. In addition, as the GPUs used in the numerical experiments are fully IEEE 754 compliant, the iteration counts, objective function values, and output images were also identical.

The "Whole" column shows the average total per iteration execution times; the "Sub. #1", "Sub. #2", "Sub. #3", and "Sub. #4" columns show the average per iteration execution times for each subproblem; and the "Misc." column shows the combined average per iteration execution times for augmentation term update kernels and an objective function value computation kernel. The CPU results show that a significant portion of the total execution time goes to solving the critical non-convex subproblem (15).

Tables 2 and 3 show the average per iteration execution times and the obtained speedups for the GTX580 and K40c GPUs, respectively. The GTX580 was on average 15.6 times faster than the Xeon CPU. The highest speedups were obtained in the case of the synthetic test image Test9 in which case the GTX580 GPU was up to 21.5 times faster. The K40c GPU was on average 26.0 times faster than the Intel Xeon CPU and the Test9 test image was processed up to 33.7 times faster. Both GPUs achieved the highest speedups in the case of the critical non-convex subproblem (15). The K40c GPU was at its best 76.0 times faster than the Intel Xeon CPU at solving the subproblem.

4.3 Discussion

A significant portion of the total execution time still goes to solving the critical non-convex subproblem (15) but the gap between it and the linear vector-valued subproblems (19) has narrowed considerably. However, even if we managed to overcome the potential global memory bottleneck associated with the linear vector-valued subproblems (19), the critical non-convex subproblem (15) would still dominate the total execution time in such a degree that it probably would not be of a significant improvement. Finally, the speedups obtained with the Mercado test images show that GPU's computational resources can be utilized best when the image size is relatively large.

Although the highest speedups were obtained in the case of the critical non-convex subproblem (15), the K40c GPU did not perform quite as well as expected. One culprit might be the Newton-bisection hybrid method which was used to solve the subproblem. For example, in the case of the Lena ($\sigma = 0.1$) input image, the Newton's method had an average success rate of

Figure 4: From top to bottom: the original images, the noisy input images ($\sigma = 0.1$), and the obtained output images. From left to right: Test9, Lena, Boat, and Mercado512[3].

Image	r_0	Iter.	Whole	Sub. #1	Sub. #2	Sub. #3	Sub. #4	Misc.
Test9, $\sigma = 0.025$	0.005	103	0.6170	0.4489	0.0985	0.0031	0.0412	0.0206
Lena, $\sigma = 0.025$	0.002	77	0.7463	0.5676	0.1097	0.0031	0.0405	0.0206
Boat, $\sigma = 0.025$	0.001	75	0.7967	0.6154	0.1115	0.0031	0.0412	0.0206
Mercado256, $\sigma = 0.025$	0.002	125	0.1719	0.1335	0.0233	0.0008	0.0081	0.0050
Mercado512, $\sigma = 0.025$	0.002	143	0.6813	0.5147	0.0977	0.0031	0.0406	0.0206
Mercado1024, $\sigma = 0.025$	0.002	172	2.6356	1.9665	0.3869	0.0125	0.1670	0.0847
Mercado2048, $\sigma = 0.025$	0.002	169	10.691	7.7660	1.6705	0.0499	0.7866	0.3460
Test9, $\sigma = 0.1$	0.015	163	0.5985	0.4348	0.0942	0.0031	0.0412	0.0206
Lena, $\sigma = 0.1$	0.005	158	0.6881	0.5189	0.0997	0.0031	0.0412	0.0206
Boat, $\sigma = 0.1$	0.005	163	0.6889	0.5181	0.1013	0.0031	0.0412	0.0206
Mercado256, $\sigma = 0.1$	0.005	213	0.1692	0.1309	0.0232	0.0008	0.0082	0.0050
Mercado512, $\sigma = 0.1$	0.005	205	0.6778	0.5104	0.0979	0.0031	0.0412	0.0206
Mercado1024, $\sigma = 0.1$	0.005	208	2.6719	1.9906	0.3958	0.0125	0.1694	0.0857
Mercado2048, $\sigma = 0.1$	0.005	205	10.661	7.7299	1.6658	0.0500	0.7982	0.3458

Table 1: Parameter values, iteration counts, and average per iteration execution times (in seconds) for the Intel Xeon CPU.

99.40%. This is perfectly fine for the CPU since the cost of processing the remaining triangles using the bisection search algorithm is neglectable. However, on the basis of the same data, there is on average 16.59% probability that an individual warp contains a work-item that has to process a triangle using the bisection search algorithm. This has a significant impact on the performance since the cost of processing a single triangle in this way is the same as processing similarly all the 32 triangles as the longest execution path of work-items within the warp determines the cost of completing the computational task assigned to this warp.

The above does not, however, explain why the considerably more powerful K40c GPU did not outperform the GTX580 GPU in such a large extent as would have been expected. The results could be partly explained by the fact that, based on our measurements, the K40c GPU is only 2-3 times faster than the GTX580 GPU at performing special operations such as computing reciprocals and square roots. The Newton-Raphson division algorithm improved performance less than 10%. In addition, we noticed that the K40c GPU was unusually sensitive to how the work group size was chosen. The critical non-convex subproblem (15) required us to set

[3] The Mercado test image is based on the works of Diego Delso and licensed under Wikimedia Commons license CC-BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0/legalcode).

Image	Whole		Sub. #1		Sub. #2		Sub. #3		Sub. #4		Misc.	
Test9, $\sigma = 0.025$	0.0287	**21.5**	0.0147	30.5	0.0085	11.6	0.0002	17.7	0.0041	10.0	0.0012	17.3
Lena, $\sigma = 0.025$	0.0507	**14.7**	0.0357	15.9	0.0095	11.5	0.0002	17.7	0.0041	9.9	0.0012	17.3
Boat, $\sigma = 0.025$	0.0654	**12.2**	0.0501	12.3	0.0097	11.5	0.0002	17.7	0.0041	10.0	0.0012	17.4
Mercado256, $\sigma = 0.025$	0.0136	**12.6**	0.0095	14.0	0.0027	8.6	0.0001	14.1	0.0010	8.5	0.0003	17.1
Mercado512, $\sigma = 0.025$	0.0472	**14.4**	0.0335	15.4	0.0082	11.8	0.0002	17.9	0.0041	10.0	0.0012	17.3
Mercado1024, $\sigma = 0.025$	0.1681	**15.7**	0.1124	17.5	0.0296	13.1	0.0006	19.6	0.0208	8.0	0.0046	18.5
Mercado2048, $\sigma = 0.025$	0.6457	**16.6**	0.4095	19.0	0.1178	14.2	0.0025	20.0	0.0975	8.1	0.0182	19.0
Test9, $\sigma = 0.1$	0.0278	**21.5**	0.0142	30.6	0.0080	11.7	0.0002	17.8	0.0042	9.9	0.0012	17.2
Lena, $\sigma = 0.1$	0.0454	**15.1**	0.0312	16.6	0.0087	11.4	0.0002	17.8	0.0041	10.0	0.0012	17.3
Boat, $\sigma = 0.1$	0.0452	**15.2**	0.0310	16.7	0.0087	11.6	0.0002	17.8	0.0041	10.0	0.0012	17.3
Mercado256, $\sigma = 0.1$	0.0133	**12.7**	0.0093	14.1	0.0027	8.5	0.0001	14.2	0.0009	8.7	0.0003	17.1
Mercado512, $\sigma = 0.1$	0.0464	**14.6**	0.0325	15.7	0.0084	11.7	0.0002	17.9	0.0041	10.1	0.0012	17.3
Mercado1024, $\sigma = 0.1$	0.1726	**15.5**	0.1167	17.1	0.0303	13.1	0.0006	19.6	0.0203	8.3	0.0046	18.7
Mercado2048, $\sigma = 0.1$	0.6440	**16.6**	0.4075	19.0	0.1178	14.1	0.0025	20.0	0.0977	8.2	0.0182	19.0
Average speedup		**15.6**		18.2		11.8		17.8		9.3		17.7

Table 2: Average per iteration execution times (in seconds) and obtained speedups for the Nvidia GeForce GTX580 GPU.

Image	Whole		Sub. #1		Sub. #2		Sub. #3		Sub. #4		Misc.	
Test9, $\sigma = 0.025$	0.0183	**33.7**	0.0059	76.0	0.0078	12.7	0.0001	29.7	0.0036	11.3	0.0009	23.3
Lena, $\sigma = 0.025$	0.0297	**25.1**	0.0163	34.8	0.0087	12.5	0.0001	29.5	0.0036	11.2	0.0009	23.4
Boat, $\sigma = 0.025$	0.0361	**22.1**	0.0225	27.3	0.0089	12.6	0.0001	29.5	0.0036	11.3	0.0009	23.4
Mercado256, $\sigma = 0.025$	0.0091	**18.8**	0.0045	29.4	0.0031	7.6	0.0000	16.6	0.0012	7.0	0.0003	17.1
Mercado512, $\sigma = 0.025$	0.0277	**24.6**	0.0153	33.6	0.0077	12.6	0.0001	29.9	0.0037	11.1	0.0009	23.2
Mercado1024, $\sigma = 0.025$	0.0970	**27.2**	0.0509	38.6	0.0262	14.8	0.0003	37.3	0.0163	10.2	0.0033	26.0
Mercado2048, $\sigma = 0.025$	0.3736	**28.6**	0.1841	42.2	0.1033	16.2	0.0012	40.6	0.0721	10.9	0.0127	27.2
Test9, $\sigma = 0.1$	0.0178	**33.6**	0.0057	75.7	0.0074	12.7	0.0001	29.5	0.0037	11.2	0.0009	23.3
Lena, $\sigma = 0.1$	0.0269	**25.5**	0.0143	36.3	0.0080	12.5	0.0001	29.6	0.0036	11.4	0.0009	23.4
Boat, $\sigma = 0.1$	0.0272	**25.4**	0.0144	35.9	0.0080	12.6	0.0001	29.7	0.0037	11.2	0.0009	23.2
Mercado256, $\sigma = 0.1$	0.0088	**19.1**	0.0045	29.3	0.0031	7.6	0.0000	16.0	0.0010	8.4	0.0003	18.2
Mercado512, $\sigma = 0.1$	0.0274	**24.8**	0.0149	34.2	0.0077	12.6	0.0001	29.8	0.0037	11.2	0.0009	23.2
Mercado1024, $\sigma = 0.1$	0.0996	**26.8**	0.0530	37.5	0.0267	14.8	0.0003	37.5	0.0162	10.4	0.0032	26.4
Mercado2048, $\sigma = 0.1$	0.3721	**28.7**	0.1826	42.3	0.1033	16.1	0.0012	40.7	0.0721	11.1	0.0127	27.2
Average speedup		**26.0**		40.9		12.7		30.4		10.6		23.5

Table 3: Average per iteration execution times (in seconds) and obtained speedups for the Nvidia Tesla K40c GPU.

the work group size as low as 64 work-items. In turn, the GTX580 GPU performed just fine when the work group size was set as high as 512 work-items. This suggest that Nvidia's OpenCL compiler might have problems with resource management. In general, the compiler seems to generate less optimal code for the K40c GPU in many situations. It also appears that the K40c GPU does not perform well in situations where the solution of a subproblem requires multiple kernel launches.

5 CONCLUSIONS

This paper presented a GPU implementation of an augmented Lagrangian based L^1-mean curvature image denoising algorithm and numerical results obtained while comparing the GPU implementation against a single-threaded CPU implementation. Up to 33-fold speedups were obtained, the average speedup being 26-fold. The pointwise handled non-convex subproblem predictably benefited most from the GPU-acceleration. The numerical results indicate that GPUs provide demonstrable benefits in the context of the higher-order variational-based image denoising algorithms and alternating direction type augmented Lagrangian methods.

6 ACKNOWLEDGMENTS

The authors thank anonymous reviewers for their valuable feedback. The presentation of our paper was significantly improved thanks to their comments and suggestions. The research of the first author was supported by the Academy of Finland (grant #252549), the Jyväskylä Doctoral Program in Computing and Mathematical Sciences (COMAS), and the Foundation of Nokia Corporation.

7 REFERENCES

[Ame10] Ament, M., Knittel, G., Weiskopf, D., and Strasser, W. A parallel preconditioned conjugate gradient solver for the Poisson problem on a multi-GPU platform. In: 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), pp. 583–592, IEEE, 2010.

[Bri10] Brito-Loeza, C., Chen, K. Multigrid algorithm for high order denoising. SIAM J. Imaging Sci., 3(3), pp. 363–389, 2010.

[Bol03] Bolz, J., Farmer, I., Grinspun, E., and Schröoder, P. Sparse matrix solvers on the GPU: conjugate gradients and multigrid. ACM Trans. Graph., 22(3), pp. 917–924, 2003.

[Con80] Conte, S. D., and de Boor, C. Elementary numerical analysis: an algorithmic approach. Mcgraw-Hill College, 1980.

[Dav11] Davidson, A., Zhang, Y., and Owens, J. D. An auto-tuned method for solving large tridiagonal systems on the GPU. In: Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium, pp. 956–965, IEEE, 2011.

[Fly70] Flynn, M. On division by functional iteration. IEEE T. Comput., C19(8), pp. 702–706, 1970.

[For83] Fortin, M., and Glowinski, R. Augmented Lagrangian methods: applications to the numerical solution of boundary value problems. North-Holland, Amsterdam, 1983.

[Glo89] Glowinski, R., and Le Tallec, P.. Augmented Lagrangian and operator-splitting methods in nonlinear mechanics. SIAM, Philadelphia, 1989.

[Göd11] Göddeke, D., and Strzodka, R. Cyclic reduction tridiagonal solvers on GPUs applied to mixed precision multigrid. IEEE T. Parall. Distr., Special Issue: High Performance Computing with Accelerators, 22(1), pp. 22–32, 2011.

[Hel12] Helfenstein, R., and Koko, J. Parallel preconditioned conjugate gradient algorithm on GPU. J. Comput. Appl. Math., 236(15), pp. 3584–3590, 2012.

[Hoc65] Hockney, R. W. A fast direct solution of Poisson's equation using Fourier analysis. J. Assoc. Comput. Mach., 12(1), pp. 95–113, 1965.

[Hoc81] Hockney, R. W., and Jesshope, C. R. Parallel computers: architecture, programming and algorithms. Bristol, UK, 1981.

[Kim11] Kim, H.-S., Wu, S., Chang. L., and Hwu W. W. A scalable tridiagonal solver for GPUs. In: 42nd International Conference on Parallel Processing, pp. 444–453, IEEE Computer Society, Los Alamitos, CA, USA, 2011.

[Kuz85] Kuznetsov, Y. A. Numerical methods in subspaces. Vychislitel'-nye Processy i Sistemy II. 37, pp. 265–350, 1985.

[Kuz96] Kuznetsov, Yu. A., and Rossi, T. Fast direct method for solving algebraic systems with separable symmetric band matrices. East-West J. Numer. Math., 4, pp. 53–68, 1996.

[Lam12] Lamas-Rodriguez, J., Arguello, F., Heras, D.B., and Boo, M. Memory hierarchy optimization for large tridiagonal system solvers on GPU. In: IEEE 10th International Symposium on Parallel and Distributed Processing with Applications (ISPA), pp. 87–94, IEEE Press, Piscataway, NJ, USA, 2012.

[Lys04] Lysaker, M., Osher, S., Tai, X.-C. Noise removal using smoothed normals and surface fitting. IEEE T. Image Process., 13(10), pp. 1345 – 1357, 2004.

[Mum94] Mumford, D. Elastica and computer vision. Algebraic geometry and its applications, pp. 491–506, Springer-Verlag, New York, 1994.

[Myl13] Myllykoski, M., Rossi, T., and Toivanen, J. Fast Poisson solvers for graphics processing units. In: Applied Parallel and Scientific Computing, Manninen, P., and Öster, P. (eds.), Lecture Notes in Computer Science, 7782, pp. 265–279, 2013.

[Myl15] Myllykoski, M., Glowinski, R., Kärkkäinen, T., and Rossi, T. A new augmented Lagrangian approach for L^1-mean curvature image denoising. SIAM J. Imaging Sci., 8(1), pp. 95–125, 2015.

[Par92] Parker, A., and Hamblen, J.O. Optimal value for the Newton-Raphson division algorithm. Inform. Process. Lett., 42(3), pp. 141–144, 1992.

[Ros99] Rossi, T., and Toivanen T. A parallel fast direct solver for block tridiagonal systems with separable matrices of arbitrary dimension. SIAM J. Sci. Comput. 20(5), pp. 1778–1796, 1999.

[Rud92] Rudin, L., Osher, S., and Fatemi, E. Nonlinear total variation based noise removal algorithms. Phys. D, 60(1–4), pp. 259–268, 1992.

[Sun14] Sun, L., and Chen, K. A new iterative algorithm for mean curvature-based variational image denoising. BIT, 54(2), pp. 523–553, 2014.

[Vas84] Vassilevski, P. Fast algorithm for solving a linear algebraic problem with separable variables. C.R. Acad. Bulgare Sci., 37, pp. 305–308, 1984.

[Val85] Vassilevski, P. Fast algorithm for solving discrete Poisson equation in a rectangle. C.R. Acad. Bulgare Sci., 38, pp. 1311–1314, 1985.

[Yan14] Yangab, F., Chenc, K., Yub, B., Fang, D. A relaxed fixed point method for a mean curvature-based denoising model. Optim. Method Softw, 29(2), pp. 274 – 285, 2014.

[Zha10] Zhang, Y. and Cohen, J., and Owens, J. D. Fast tridiagonal solvers on the GPU. In: Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming PPoPP 10, pp. 127–136, ACM Press, New York, NY, USA, 2010.

[Zhu12] Zhu, W., and Chan, T. Image denoising using mean curvature of image surface. SIAM J. Imaging Sci., 5(1), pp. 1–32, 2012.

[Zhu13] Zhu, W., Tai, X.-C., and Chan, T. Augmented Lagrangian method for a mean curvature based image denoising model. Inverse Probl., 7(4), pp. 1409–1432, 2013.