Performance analysis of a parallel algorithm for restoring large-scale CT images

Stanislav Harizanov [a], Ivan Lirkov [a,*], Krassimir Georgiev [a], Marcin Paprzycki [b,d], Maria Ganzha [b,c]

[a] Institute of Information and Communication Technologies, Bulgarian Academy of Sciences, Acad. G. Bonchev, bl. 25A, 1113 Sofia, Bulgaria
[b] Systems Research Institute, Polish Academy of Sciences, ul. Newelska 6, 01-447 Warsaw, Poland
[c] Department of Mathematics and Information Sciences, Warsaw University of Technology, ul. Koszykowa 75, 00-661 Warsaw, Poland
[d] Department of Management and Technical Sciences, Warsaw Management Academy, ul. Kaweczynska 36, 03-772 Warsaw, Poland

ARTICLE INFO

ABSTRACT

In multiple areas of image processing, such as Computed Tomography, in which data acquisition is based on counting particles that hit a detector surface, Poisson noise occurs. Using variance-stabilizing transformations, the Poisson noise can be approximated by a Gaussian one, for which classical denoising filters can be used. This paper presents an experimental performance study of a parallel implementation of the Poissonian image restoration algorithm, introduced in Harizanov et al. (2013). Hybrid parallelization based on MPI and OpenMP standards is investigated. The convergence rate of the algorithm heavily depends on both the image size and the choice of input parameters (ρ, σ), thus maximizing its parallel efficiency is vital for real-life applications. The implementation is tested for high-resolution radiographic images, on Linux clusters with Intel processors and on an IBM supercomputer.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

Industrial Computed Tomography (CT) uses large series of two-dimensional radiographic images for the volume reconstruction of the scanned object. The gray-scale pixel intensity, for each of the planar images, is based on the corresponding X-ray amount, captured by a flat panel detector placed behind the object. In such processes, where images are obtained by counting particles, Poisson noise becomes one of the main noise sources. It needs to be removed from every 2D radiographic image, separately, since it generates a non-structured noise in the 3D volume reconstruction that cannot be theoretically treated.

---

* Corresponding author.
E-mail addresses: sharizanov@parallel.bas.bg (S. Harizanov), ivan@parallel.bas.bg (I. Lirkov), georgiev@parallel.bas.bg (K. Georgiev), marcin.paprzycki@ibspan.waw.pl (M. Paprzycki), maria.ganzha@ibspan.waw.pl (M. Ganzha).
URLs: http://parallel.bas.bg/~sharizanov/ (S. Harizanov), http://parallel.bas.bg/~ivan/ (I. Lirkov), http://parallel.bas.bg/~georgiev (K. Georgiev), http://www.ibspan.waw.pl/~paprzyck/ (M. Paprzycki), http://pages.mini.pw.edu.pl/~ganzham (M. Ganzha).

Poissonian image denoising is a nonlinear process, unlike in the case of Gaussian noise. Restoration is usually performed via the minimization of a cost function, formulation of which includes representation of data fidelity and regularization terms. The use of data fidelity ensures that the restored image approximates the ground-truth one, while the use of the regularizer guarantees noise removal. There are two mainstream approaches for choosing the data fidelity term. The first one is based on the direct Maximum A Posteriori (MAP) estimation of the Poisson distribution, in which the neg-log-likelihood function is the I-divergence (generalized Kullback–Leibler divergence [1]). The alternative is to apply a variance-stabilizing transformation (VST), in order to remove the mean/variance relationship of the Poisson distribution. A typical example of a VST is the Anscombe transform [2] that transforms the Poisson noise to an approximately Gaussian one, with zero-mean and unit variance. For normally distributed data, the least squares estimators have the best denoising properties and are used in computational practice. They lead to a linear optimization, where different pixels can be independently and simultaneously treated. Therefore, an optimal parallelization of the implementation of this approach can significantly improve the performance of the algorithm. The choice of the regularization term depends on the concrete application and the initial properties of the image. Here, we use the total variation (TV) functional [3], due to its robustness.

There are two different optimization models that are commonly used. In the penalized optimization, both the regularization and the data fidelity terms are a part of the cost function, and their sum is weighted by a parameter $\lambda > 0$. In the constrained optimization, only one of the terms is minimized, while the magnitude of the other one is globally restricted by a parameter τ. The two problems are closely related (see, for instance, [4]) and (under mild assumptions, satisfied for a large class of regularization and data fidelity terms) there is a one-to-one correspondence $\lambda \leftrightarrow \tau$, such that their minimizers coincide. In general, the penalized methods are both theoretically easier and computationally faster to apply, but the parameter λ needs to be carefully tuned, which might be a difficult task. On the other hand, the constrained methods are both mathematically and computationally more complex, but a meaningful choice of τ is a priori known, due to the statistical arguments.

In [5], a TV-regularized Anscombe-constrained mathematical model for deblurring Poissonian images is considered. There, the parameter τ is estimated to be $\tau = n$, where n is the image size. The main advantage of the Anscombe transform-based approach is that it can be straightforwardly generalized to mixed Poisson–Gaussian noise [6] that commonly materializes in practice. However, the Anscombe inverse transform is unstable for low-count noise levels [7] and is ill-posed in the presence of blur (a linear degradation operator). Therefore, to the best of our knowledge, this is the first Anscombe-constrained numerical algorithm for Poissonian blurry data restoration. The algorithm is a realization of the **p**rimal-**d**ual **h**ybrid **g**radient algorithm [8], with an extrapolation (**m**odification) of the dual variable. In order to avoid solving the inverse problem, the authors use a novel technique of component-wise projections onto the epigraph of a 1D convex function related to the Anscombe transform (for more details, see [9]). Those projections can be efficiently computed numerically. Finally, it should be noted that, in CT imaging, Gaussian blur is often used for smoothing the input data, thus, it should also be considered in the recovery process. Combining all the above arguments, we conclude that the optimization model [5] is well-adapted to the real life. Henceforth, this algorithm has huge potential for practical applications.

This being the case it is worthy considering actual potential for parallelization of this approach to image denoising. This is particularly important since, as noted above, CT image processing involves sequences of 2D images that have to be processed (i.e. denoised) to "compose" a resulting 3D image. Therefore the aim (and the main contribution of this paper) is to investigate parallel performance characteristic of the proposed method, when executed on modern (in 2016) parallel computers. To this effect we proceed as follows. In the next section we introduce the notation and outline the algorithm. We follow with a description of applied parallelization. Afterwards, we introduce the experimental setup (including description of computers used in our experiments) and analyze the obtained results.

2. Notation and algorithm description

To represent the investigated real-life situation in mathematical terms, let us consider solution of an ill-posed inverse problem of recovering an original 2D image $\bar{\mathbf{u}} \in [0, \nu]^{M \times N}$ from observations

$$\mathbf{f} = \mathscr{P}(H\bar{\mathbf{u}}), \tag{1}$$

where ν is the maximal gray-scale intensity, $n = MN$, $\mathscr{P}$ denote an independent Poisson noise corruption process, and $H \in [0, +\infty)^{n \times n}$ is a blur operator, corresponding to a convolution with a Gaussian kernel. The Anscombe transform is defined as

$$T : [0, +\infty)^n \to (0, +\infty)^n : \mathbf{v} = (v_i)_{1 \leq i \leq n} \mapsto 2 \left(\sqrt{v_i + \frac{3}{8}} \right)_{1 \leq i \leq n}.$$

The constrained optimization problem can be thus stated as

$$\underset{\mathbf{u} \in [0, \nu]^n}{\operatorname{argmin}} \|\nabla \mathbf{u}\|_{2,1} \quad \text{subject to } \|T(H\mathbf{u}) - T(\mathbf{f})\|_2^2 \leq n, \tag{2}$$

where $\nabla \in \mathbb{R}^{2n \times n}$ is the discrete gradient operator (forward differences and Neumann boundary conditions are used), and $\| \cdot \|_{2,1}$ denotes the TV semi-norm, namely the sum of the Euclidean lengths of the 2D gradient vectors.

Projections are performed onto the epigraph of the function

$$\varphi(x) := \begin{cases} (2\sqrt{x} - z)^2 & \text{if } x \geq 0, \\ +\infty & \text{otherwise,} \end{cases} \tag{3}$$

where $z = T(f)_i > 0$ varies for different pixel data.

In what follows, we will use the following notation

$$\mathbf{w} = \text{prox}_{\sigma^{-1}\|\cdot\|_{2,1}}(\mathbf{v})$$

for the proximal operator $\mathbf{w}$ considered in the space $\mathbb{R}^{n \times 2}$. The operator is component-wise defined as:

$$\mathbf{w}_i = \max\left\{1 - \frac{\sigma^{-1}}{|\mathbf{v}_i|}, 0\right\} \mathbf{v}_i, \quad \text{where } \mathbf{w}_i, \mathbf{v}_i \in \mathbb{R}^2.$$

The proposed algorithm is taken from [5] and can be written as follows:

Algorithm 1. Initialization:

$$\mathbf{u}^{(0)}, \boldsymbol{\zeta}^{(0)}, \left(\mathbf{p}_j^{(0)}\right)_{1 \leq j \leq 3} = \left(\bar{\mathbf{p}}_j^{(0)}\right)_{1 \leq j \leq 3}, \quad (\rho, \sigma) \in (0, +\infty)^2, \ \rho\sigma < 1/9.$$

For $k = 0, 1, \ldots$ repeat until a stopping criterion is reached

1. $\mathbf{u}^{(k+1)} = \max\left\{\min\left\{\left(\mathbf{u}^{(k)} - \sigma\rho\left(H^*\bar{\mathbf{p}}_1^{(k)} + \nabla^*\bar{\mathbf{p}}_2^{(k)}\right)\right), \nu\mathbf{1}_n\right\}, \mathbf{0}\right\}$

2. $\boldsymbol{\zeta}^{(k+1)} = P_{V_n}\left(\boldsymbol{\zeta}^{(k)} - \sigma\rho\bar{\mathbf{p}}_3^{(k)}\right)$

3. $\left(v_{1,i}, \eta_i\right) = P_{\text{epi}\varphi_i}\left(p_{1,i}^{(k)} + \left(H\mathbf{u}^{(k+1)}\right)_i + 3/8, \ p_{3,i}^{(k)} + \zeta_i^{(k+1)}\right), \quad i = 1, \ldots, n$

4. $\mathbf{v}_2 = \mathbf{p}_2^{(k)} + \nabla\mathbf{u}^{(k+1)}$

5. $\mathbf{p}_1^{(k+1)} = \mathbf{p}_1^{(k)} + H\mathbf{u}^{(k+1)} + 3/8 - \mathbf{v}_1$

6. $\mathbf{p}_2^{(k+1)} = \mathbf{v}_2 - \text{prox}_{\sigma^{-1}\|\cdot\|_{2,1}}(\mathbf{v}_2)$

7. $\mathbf{p}_3^{(k+1)} = \mathbf{p}_3^{(k)} + \boldsymbol{\zeta}^{(k+1)} - \boldsymbol{\eta}$

8. $\bar{\mathbf{p}}_j^{(k+1)} = \mathbf{p}_j^{(k+1)} + \left(\mathbf{p}_j^{(k+1)} - \mathbf{p}_j^{(k)}\right), \quad j = 1, 2, 3.$

In Steps 1–3 we perform projections onto the hypercube $[0, \nu]^n \subset \mathbb{R}^n$, the closed half-space $V_n := \{\boldsymbol{\zeta} \in \mathbb{R}^n : \langle \mathbf{1}_n, \boldsymbol{\zeta} \rangle \leq n\}$, and the epigraph of φ from (3), respectively. In Step 6 the coupled soft shrinkage with threshold σ^{-1} is performed (see, [4] for more details). The remaining steps can be computed in a straightforward way.

From the computational point of view, the effect of the blur operator H on a given vector, in Steps 1 and 3, can be performed by a matrix by matrix multiplication. In our experiments, we used blur operator corresponding to a convolution with a Gaussian kernel with the standard deviation $s = 1.3$. Then the blur kernels are represented as symmetric banded matrices with bandwidth $\lfloor 3s \rfloor + 1 = 4$. For smaller values of the standard deviation, the corresponding blur kernels have smaller bandwidth.

In summary, when conceptualizing Algorithm 1, from the point of view of key computational operations, it consists of matrix by matrix multiplication in steps 1 and 3, Newton's method in step 3, scalar multiplication and vector sums in all steps, and a TV semi-norm calculation in the coupled soft shrinkage, performed in step 6.

3. Algorithm parallelization

Numerical results presented in [5] show heavy dependence between the convergence rate of the algorithm and the image size and the choice of the input parameters (ρ, σ). On the other hand, typically, the CT data consists of hundreds of high resolution (e.g. size 1446×1840) 2D radiographic images. Thus, for all practical purposes, the sequential realization of the algorithm cannot be used for real-time 3D volume reconstruction. Meanwhile, the least-squares data fidelity and the component-wise epigraphical projections allow for complete splitting of the pixel data, and fully parallelized implementation of the algorithm. However, an open question remains: what level parallel efficiency can be actually obtained when a parallelized version of the algorithm in question is executed on today's (2016) parallel computers. In this context, taking into account current state-of-the-art in parallel computing, we consider multi-node and multi-core parallelizations. It should be noted that, while we recognize that CPU's may be also applied to the problem at hand, in our initial investigations, we have decided to focus our attention only on multi-node and multi-core parallelization.
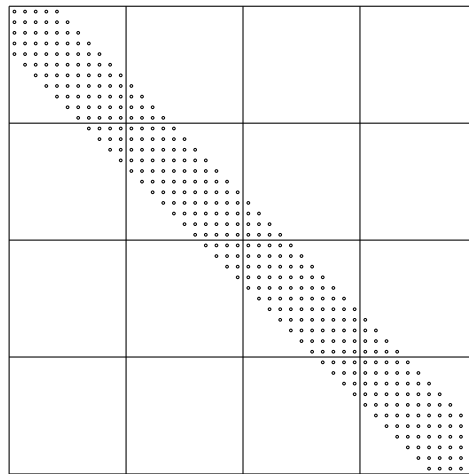
Fig. 1. The block form of the banded matrix.

3.1. Multi-thread-based parallel implementation

There exist two possible ways to compute the matrix by matrix and matrix by vector multiplication on computers with multi-core processors. The first one is to use a multi-threaded library, such as the Engineering and Scientific Subroutine Library on the IBM computers (ESSL, see http://www-03.ibm.com/systems/software/essl/index.html) or the Intel Math Kernel Library on Intel-based clusters (MKL, see http://software.intel.com/en-us/articles/intel-mkl/). The other possible approach is to use OpenMP. Obviously, Intel MKL and the IBM ESSL provide comprehensive functionality support for BLAS (level 1, 2, and 3) and LAPACK linear algebra routines, offering vector, vector–matrix, and matrix–matrix operations. However, our parallel implementation of multiplication of a symmetric band matrix by a matrix requires multiplication of a symmetric band matrix by a matrix and multiplication of triangular matrix by a matrix. Here, it should be stressed that there is no subroutine available in BLAS designed solely to multiply a band-matrix by a matrix. Therefore, we use BLAS subroutines DSBMV and DTRMM (see [10]) that perform matrix–vector operation with symmetric band matrix and matrix–matrix operation with upper or lower triangular matrix.

In the current parallel implementation, we use the multi-threaded library for the matrix by matrix multiplication and OpenMP for the matrix by vector multiplication and all vector operations. We have compiled our code using the OpenMP library and linked it to the multi-threaded library for the parallelization on a multi-core node of the computer system.

3.2. Multi-node implementation

It is a well-known fact that to achieve an efficient parallel, multi-node implementation of any algorithm, one has to minimize the communication between nodes. To this effect, we partition the image into m rectangles so that each rectangle contains approximately MN/m pixels. We map all pixels from a given rectangle into a single computing node. In this way, the Newton's method is performed in parallel for each pixel and does not need any communication. The same is true for the scalar multiplication and vector sums.

Observe that computation of the discrete gradient operator requires only M values from the next node (denoted by $+$ in Fig. 2). If the bandwidth of the blur kernel is 4, the matrix by matrix multiplication requires communication of only $4M$ values from both neighbor nodes. The values from the neighbor nodes, needed on the node P_1, are denoted by circles in Fig. 2. In the parallel implementation of the multiplication of the banded matrix by the dense matrix, the processor multiplies the diagonal block of the banded matrix by the part of the dense matrix located on the same node and the off-diagonal blocks by the vectors received from the neighbor nodes; see, the block form of the banded matrix represented in Fig. 1. To further improve performance, in our implementation, we have used overlapping of computation and local communication.

Observe that only the computation of the projection onto V_n requires a single global communication. Here, the MPI function MPI_Allreduce was used to compute the inner product $\langle \mathbf{1}_n, \boldsymbol{\zeta} \rangle$.

4. Numerical experiments

4.1. Images used in the experiments

To demonstrate the performance of the parallel denoising algorithm we have used the following images. The first two images, namely 'brain' (184×140) and 'cameraman' (256×256), were obtained from the set of MATLAB's demo images. The two images were blurred by a matrix corresponding to a convolution with a Gaussian kernel, with the standard deviation
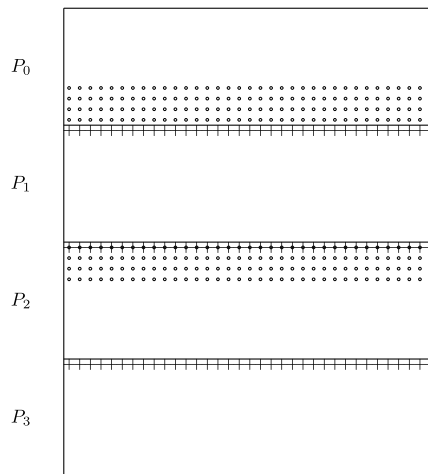
Fig. 2. Data distribution on four processors.

Table 1
Information about available hardware on the four computer systems.

Computer system	Number of processors per node	Processor	Number of cores per processor	Maximal number of threads per node	Memory per node in GB
IBM Blue Gene/P	1	PowerPC 450	4	4	4
HPCG	2	Intel Xeon X5560	4	16	24
Avitohol	2	Intel Xeon E5-2650 v2	8	32	64
MareNostrum	2	Intel Xeon E5-2670	8	32	32, 64, 128

Table 2
Compilers and libraries used on the four computer systems.

IBM Blue Gene/P	HPCG	Avitohol	MareNostrum
Compiler			
IBM XL C Compiler	Intel C Compiler		
09.00.0000.0013	15.0.3	16.0.2	16.0.1
MPI			
MPICH2	Intel MPI		
1.1	5.1.0.079	5.1.3.181	5.1.2.150
BLAS			
Engineering and Scientific Subroutine Library	Intel Math Kernel Library		

equal to 1.3. We have tested the parallel algorithm also on images obtained from the Tomograph XTH 225 Compact industrial CT scanning. The images have size 723×920 or 1446×1840 pixels. Finally, we have applied the proposed algorithm to a "transposed image", with size 1840×1446 pixels, which is equivalent to a parallel algorithm where the image is divided into vertical strips. For each image, the parameters of the primal–dual algorithm (ρ, σ) were tuned, in order to achieve good convergence of the algorithm. Specifically, well tuned parameters mean that a smaller total number of iterations will be performed. However, it should be stressed that the parallel performance of the algorithm, during each iteration, remains unchanged.

4.2. Experimental setup

Let us now report on the experiments performed with the parallel implementation of the algorithm. A portable parallel code was designed and implemented in C. As outlined above, the hybrid parallelization is based on joint application of the MPI and the OpenMP standards [11–14]. Furthermore, we use BLAS subroutines DSBMV and DTRMM for the matrix by vector, and the matrix by matrix multiplication, respectively.

The parallel code has been tested on cluster computer systems HPCG and Avitohol, at the Advanced Computing and Data Centre of the Institute of Information and Communication Technologies of the Bulgarian Academy of Sciences, on a supercomputer MareNostrum at the Barcelona Supercomputing Center, and on the IBM Blue Gene/P machine at the HPC Center of the West University of Timisoara (UVT). Tables 1 and 2 summarize the basic information about hardware, compilers and libraries used on the four computer systems. More details about each machine can be found in the next subsection.

Table 3
Average execution time for 50000 iterations of the algorithm on a single node of the IBM Blue Gene/P.

M	N	k		
		1	2	4
184	140	1968.89	1047.00	541.85
256	256	5447.13	2789.52	1426.96
723	920	31873.15	16468.24	8576.65
1446	1840	203484.30[a]	86774.35[a]	44157.63
1840	1446	174638.73[a]	87828.33[a]	44657.68

[a] The execution time for 50000 iterations exceeds the wall clock limit in the LoadLeveler (24 h). Thus, the reported average time is an interpolation of the execution time for 10000, 20000, and 40000 iterations.

Table 4
Average execution time for 50000 iterations of the algorithm on multiple nodes of the IBM Blue Gene/P using four OpenMP threads per node.

Nodes	$M \times N$				
	184×140	256×256	723×920	1840×1446	1446×1840
2	300.72	752.11	4522.52	22073.92	22541.97
3	202.92	497.99	3041.56	12524.39	12973.71
4	172.05	408.81	2340.84	11058.03	11369.30
5	134.12	305.26	1870.56	7557.76	7755.82
6	118.50	265.00	1575.69	6315.71	6584.73
8	108.71	227.07	1196.17	5567.35	5740.86
16	77.38	139.30	620.24	2798.61	2876.55
32	62.63	100.08	324.90	1079.46	1109.36

In our experiments, times were collected using the MPI provided timer, and we report the average time from multiple runs. In what follows, we report the average elapsed time T_p (in seconds), when using m MPI processes and k OpenMP threads, where $p = m \times k$, the parallel speed-up $S_p = T_1/T_p$, and the parallel efficiency $E_p = S_p/p$. During the numerical experiments, we have tested the parallel algorithm on one node for the number of OpenMP threads from 1 to the maximal available number of threads. On many nodes we report the results for the number of OpenMP threads equal to the number of cores per node.

4.3. Experimental results on individual computer systems

Tables 3 and 4 present times collected on the IBM Blue Gene/P supercomputer. For our experiments we used the BG/P machine located at the HPC Center of the West University of Timisoara (UVT). The supercomputer consists of 1024 compute nodes with quad core PowerPC 450 processors (running at 850 MHz). Each node has 4 GB of RAM. For the point-to-point communications a 3.4 Gb 3D mesh network is used (for more details about the machine, see http://hpc.uvt.ro/infrastructure/bluegenep/). In our experiments, to compile the code we have used the IBM XL C compiler and compiled the code with the following options: "-O5 -qstrict -qarch=450d -qtune=450 -qsmp=omp". To use the BLAS subroutines, we linked our code to the multi-threaded ESSL.

Tables 5 and 6 show the results collected on the HPCG cluster. The HP Cluster Platform Express 7000 consists of 36 blades BL 280c, with dual Intel Xeon X5560 processors (for the total of 288 cores and 576 computational threads). Each processor runs at 2.8 GHz. Processors within each blade share 24 GB of RAM, while nodes are interconnected with non-blocking DDR Interconnection via the Voltaire Grid director 2004, with latency 2.5 μs and bandwidth 20 Gbps (see also http://www.hpc.acad.bg/). We used an Intel C compiler, and compiled the code with the option "-O3 -openmp". To use the BLAS subroutines, we linked our code to the Intel Math Kernel Library (MKL).

The supercomputer system Avitohol is built with HP Cluster Platform SL250S GEN8. It has 150 servers, and two Intel 8-core Intel Xeon E5-2650 v2 8C processors and two Intel Xeon Phi 7120P coprocessors per node. Each processor runs at 2.6 GHz. Processors within each node share 64 GB of memory. Nodes are interconnected with a high-speed InfiniBand FDR network (see also http://www.hpc.acad.bg/). We used the Intel C compiler, and compiled the code with the options "-O3 -openmp". To use the BLAS subroutines, we linked our code to the optimized Intel MKL.

Finally, Tables 9 and 10 show the results collected on the MareNostrum, the most powerful supercomputer in Spain. It has 3,056 compute nodes, and two Intel SandyBridge-EP E5-2670/1600 20M 8-core processors per node. Each processor runs at 2.6 GHz. Processors within each node share 32, 64, or 128 GB of memory. Nodes are interconnected with a high-speed InfiniBand FDR10 network (see also http://www.bsc.es/marenostrum-support-services/mn3). When running our code on the MareNostrum, we used the Intel C compiler, and compiled the code with the options "-O3 -openmp". To use the BLAS subroutines, we linked our code to the optimized Intel MKL.

Table 5
Average execution time for 50000 iterations of the algorithm on a single node of the HPCG.

M	N	k				
		1	2	4	8	16
184	140	111.56	75.45	44.53	23.77	29.06
256	256	331.97	190.61	108.49	56.75	67.06
723	920	2889.83	1632.87	951.92	693.91	759.44
1446	1840	6033.53	3592.76	3002.91	1969.94	2089.67
1840	1446	6059.99	3663.30	2781.52	1903.07	2107.49

Table 6
Average execution time for 50000 iterations of the algorithm on many nodes of the HPCG using eight OpenMP threads.

M	N	Nodes					
		2	3	4	5	6	8
184	140	18.04	13.14	12.70	11.20	10.20	10.24
256	256	40.17	30.67	24.97	21.84	19.52	16.56
723	920	418.70	290.21	153.89	117.64	87.10	76.06
1446	1840	1405.22	973.61	782.80	622.49	487.14	358.27
1840	1446	1421.73	985.70	778.57	624.65	492.22	366.34

Table 7
Average execution time for 50000 iterations of the algorithm on a single node of the Avitohol.

M	N	k				
		1	2	4	8	16
184	140	183.41	107.48	59.37	33.95	24.90
256	256	513.03	276.88	143.40	75.31	41.44
723	920	2865.78	1471.72	763.39	424.63	278.96
1446	1840	10626.91	5210.13	2687.58	1479.57	971.47
1840	1446	10569.33	5243.83	2735.01	1516.87	986.93

Since each computer allows to use different number of nodes and/or different number of threads per node, it is difficult to make direct comparisons between them. As a matter of fact, such comparison was not the goal of our work. Nevertheless, it is easy to see that the Blue Gene is the least powerful among the four machines used in our experiments. This is related not only to the age of this machine, but also to the fact that, even when it was just completed, it had relatively slow processors. At the same time the Avitohol and the MareNostrum are the two most powerful machines, with only a minimal difference between them.

4.4. Parallel performance characteristics on individual machines

The average execution time on the four parallel systems is shown in Tables 3–10. The number of iterations in the algorithm is fixed to 50000. We have to mention, again, that this number of iterations does not allow to measure the execution time of the algorithm for the largest CT images on the IBM supercomputer, when using less than four OpenMP threads, because of the wall clock limit in the batch system established in the computer center where it is located. Instead, we have calculated an estimate for the execution time based on the measurement for the smaller number of iterations.

The speed-up obtained on the four parallel systems is reported in Tables 11–14. Again, we note that we could not measure the execution time on a single IBM core and in Table 11 we reported the speed-up per node as the ratio of the execution time on a single node over the time on multiple nodes. Table 11 shows good efficiency even for the small size 'cameraman' image on up to 8 nodes and for the bigger CT images on up to 32 nodes. Here, a super-linear speed-up is observed on 32 nodes, which is caused by the well-known effect resulting from decreasing the required memory per node and thus improving the memory management. This effect has been observed also in our earlier experiments run on that machine (see, for instance, [15]). The efficiency of the IBM Blue Gene/P for small problems is also in agreement with our earlier experiments on these supercomputers (for multiple cases of solving partial differential equations in 2D and 3D). It is a result of a somewhat unbalanced architecture of the Blue Gene, where an extremely efficient network is combined with "relatively slower" processors.

Tables 12–14 show the speed-up obtained on the Linux clusters with the Intel processors. It can be seen that for the "brain" image, the parallel efficiency is above 50% only when using up to 8 threads. For the "cameraman" image the efficiency is above 50%: on up to 8 cores of the HPCG, on up to 32 cores of the Avitohol, and on up to 16 cores of the MareNostrum.

Table 8
Average execution time for 50000 iterations of the algorithm on many nodes of the Avitohol using 16 OpenMP threads.

M	N	Nodes						
		2	3	4	5	6	8	16
184	140	18.72	21.67	20.40	17.20	16.10	15.13	12.39
256	256	23.61	21.96	19.54	16.80	17.10	16.36	12.36
723	920	132.98	83.35	67.33	54.74	48.03	38.07	22.25
1446	1840	496.31	328.66	238.78	191.22	145.56	96.19	56.52
1840	1446	507.98	329.54	241.45	194.15	149.02	97.00	52.64

Table 9
Average execution time for 50000 iterations of the algorithm on a single node of the MareNostrum.

M	N	k				
		1	2	4	8	16
184	140	158.14	92.11	51.92	31.65	28.74
256	256	433.95	236.08	123.95	69.94	44.10
723	920	2493.16	1274.59	657.51	391.40	254.83
1446	1840	9679.08	5013.61	2560.18	1569.86	1058.26
1840	1446	9686.76	4972.72	2580.64	1584.19	1072.96

Table 10
Average execution time for 50000 iterations of the algorithm on many nodes of the MareNostrum using 16 OpenMP threads.

M	N	Nodes						
		2	3	4	5	6	8	16
184	140	21.38	21.95	22.00	20.78	19.73	20.29	18.41
256	256	25.74	22.16	20.26	21.93	22.18	21.18	19.22
723	920	104.39	67.89	56.98	44.10	43.37	36.13	24.29
1446	1840	523.40	337.08	249.46	181.76	146.33	107.59	58.32
1840	1446	527.92	343.06	253.77	184.71	149.12	107.71	57.02

Table 11
Speed-up on the IBM Blue Gene/P.

M	N	Nodes							
		2	3	4	5	6	8	16	32
184	140	1.80	2.67	3.15	4.04	4.57	4.98	7.00	8.65
256	256	1.90	2.87	3.49	4.67	5.38	6.28	10.24	14.26
723	920	1.90	2.82	3.66	4.59	5.44	7.17	13.83	26.40
1446	1840	2.00	3.53	3.99	5.84	6.99	7.93	15.78	40.91
1840	1446	1.98	3.44	3.93	5.76	6.78	7.78	15.52	40.26

Table 12
Speed-up on HPCG.

M	N	$m \times k$								
		2	4	8	16	24	32	40	48	64
184	140	1.71	3.12	5.37	8.15	9.79	11.93	13.53	13.89	14.25
256	256	1.88	3.57	6.37	10.55	13.30	17.73	19.93	22.24	23.65
723	920	1.77	3.04	4.16	6.90	9.96	18.78	24.56	33.17	37.99
1446	1840	1.80	2.92	4.17	7.27	9.49	12.21	14.97	19.02	23.28
1840	1446	1.82	2.88	4.19	7.20	9.84	12.42	15.00	19.06	23.15

The parallel performance is much better for the (largest) CT images. Here, for instance, the obtained efficiency on 16 nodes of the Avitohol is 78%. As it could be expected, from the point of view of parallel speed-up the HPCG is less efficient than the IBM Blue Gene/P. But if we compute and compare the speed-up per node (as we did for the IBM supercomputer), for the CT images, we obtain speed-up 17.19 (18.15) on 16 nodes of Avitohol (MareNostrum). It shows that the newer Linux clusters have faster processors as well as more efficient network than the older IBM machine.

Table 13
Speed-up on Avitohol.

$m \times k$	$M \times N$				
	184×140	256×256	723×920	1840×1446	1446×1840
2	1.71	1.86	1.95	2.05	2.03
4	3.09	3.59	3.75	3.98	3.89
8	5.42	6.82	6.75	7.18	6.97
16	7.41	12.38	10.27	10.94	10.71
32	9.80	21.73	21.55	21.41	20.81
48	8.55	23.36	34.38	32.33	32.07
64	9.08	26.26	42.56	44.50	43.77
80	10.66	30.54	52.35	55.57	54.44
96	11.39	30.00	59.67	73.01	70.92
128	12.12	31.35	75.28	110.48	108.97
256	14.80	41.49	128.78	188.01	200.80

Table 14
Speed-up on MareNostrum.

$m \times k$	$M \times N$				
	184×140	256×256	723×920	1840×1446	1446×1840
2	1.72	1.84	1.96	1.93	1.95
4	3.05	3.50	3.79	3.78	3.75
8	5.00	6.20	6.37	6.17	6.11
16	5.50	9.84	9.78	9.15	9.03
32	7.40	16.86	23.86	18.49	18.35
48	7.20	19.58	36.69	28.71	28.24
64	7.19	21.42	43.72	38.80	38.17
80	7.97	21.22	56.48	55.98	57.69
96	8.02	19.56	57.43	66.15	64.96
128	7.79	20.49	68.94	89.97	89.94
256	9.38	24.77	102.55	174.46	186.89

Finally, in Fig. 3, we represent execution time of the hybrid code on all four systems. Here, it can be seen that one core of the HPCG system is the fastest but, for large CT images, the Avitohol is the most efficient computer architecture for the parallel realization of the discussed algorithm.

5. Concluding remarks

The aim of this paper was to study the efficiency of the parallel implementation of a novel image restoration algorithm. The proposed algorithm solves an Anscombe-transformed constrained optimization problem, based on the Least Squares techniques. The parallel implementation of the algorithm admits complete splitting of the pixel data and allows for their independent treatment within each iteration. Following the current state-of-the-art, in our parallel implementation, we have combined the MPI and the OpenMP standards, to capture the parallelism available within computational nodes and on the level of a multi-node system. We report the performance of the implemented algorithm on four distinct parallel computer systems. Obtained results indicate that the proposed approach shows very good performance when restoring large-scale CT images on a supercomputer as well as on Intel-processor-based clusters. Because of that, we significantly increase the application value of the algorithm, which up to this moment, was used only in a theoretical framework.

Taking into account the fact that, nowadays, image processing (like the one considered here — for the CT images) often involves GPU accelerators, porting the proposed code to such environment could be of value. At the same time, appearance on the market (April of 2016) of the next generation of the Intel's Xeon Phi, the Knights Landing processor, with 72 powerful processing nodes per an "integrated processor" raises questions concerning the best pathway to the future of high performance realization of the proposed algorithm. Therefore, first, we plan to gain access to a machine with Knights Landing processors and evaluate the efficiency of parallelization of the proposed algorithm in this environment.
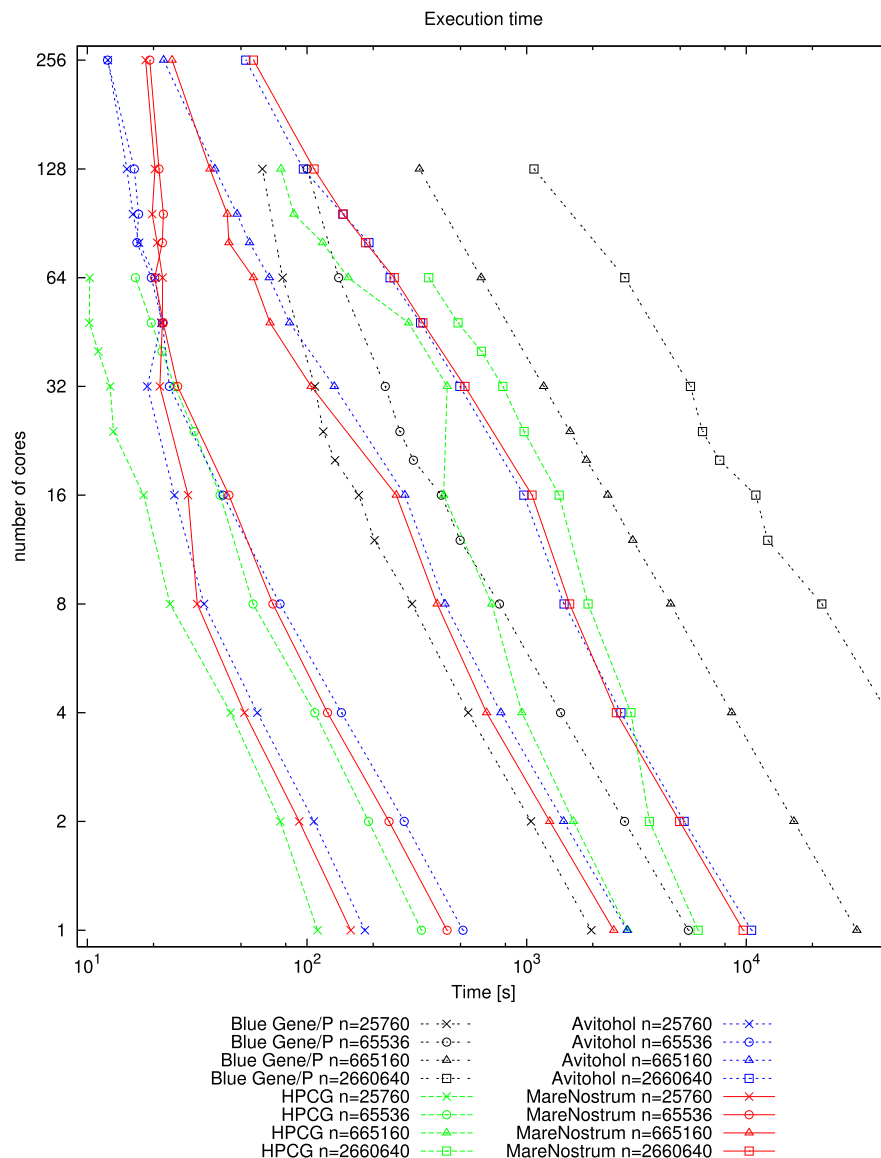
Acknowledgments

Execution time

Fig. 3. Execution time on the four computer systems.

References

[1] S. Kullback, R.A. Leibler, On information and sufficiency, Ann. Math. Statist. 22 (1) (1951) 79–86. http://dx.doi.org/10.1214/aoms/1177729694.
[2] F.J. Anscombe, The transformation of Poisson, binomial and negative-binomial data, Biometrika 35 (1948) 246–254.
[3] L.I. Rudin, S. Osher, E. Fatemi, Nonlinear total variation based noise removal algorithms, Physica D 60 (1992) 259–268.
[4] R. Ciak, B. Shafei, G. Steidl, Homogeneous penalizers and constraints in convex image restoration, J. Math. Imaging Vision 47 (3) (2013) 210–230.
[5] S. Harizanov, J.-C. Pesquet, G. Steidl, Epigraphical projection for solving least squares Anscombe transformed constrained optimization problems, in: A. Kuijper, K. Bredies, T. Pock, H. Bischof (Eds.), Scale-Space and Variational Methods in Computer Vision, in: Lecture Notes in Computer Science, vol. 7893, Springer, 2013, pp. 125–136.
[6] F. Murtagh, J.-L. Starck, A. Bijaoui, Image restoration with noise suppression using a multiresolution support, Astron. Astrophys. Suppl. Ser. 112 (1995) 179–189.
[7] M. Mäkitalo, A. Foi, Optimal inversion of the Anscombe transformation in low-count Poisson image denoising, IEEE Trans. Image Process. 20 (1) (2011) 99–109.
[8] A. Chambolle, T. Pock, A first-order primal–dual algorithm for convex problems with applications to imaging, J. Math. Imaging Vision 40 (1) (2011) 120–145.
[9] G. Chierchia, N. Pustelnik, J.-C. Pesquet, B. Pesquet-Popescu, Epigraphical projection and proximal tools for solving constrained convex optimization problems, Signal Image Video Process. 9 (8) (2015) 1737–1749.
[10] L.S. Blackford, A. Petitet, R. Pozo, K. Remington, R.C. Whaley, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, et al., An updated set of basic linear algebra subprograms (BLAS), ACM Trans. Math. Software 28 (2) (2002) 135–151.
[11] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J. Dongarra, MP: The Complete Reference, Scientific and Engineering Computation Series, The MIT Press, Cambridge, Massachusetts, 1997, second printing.
[12] D. Walker, J. Dongarra, MPI: a standard message passing interface, Supercomputer 63 (1996) 56–68.

[13] R. Chandra, R. Menon, L. Dagum, D. Kohr, D. Maydan, J. McDonald, Parallel Programming in OpenMP, Morgan Kaufmann, 2000.
[14] B. Chapman, G. Jost, R. Van Der Pas, Using OpenMP: Portable Shared Memory Parallel Programming, Vol. 10, MIT press, 2008.
[15] I. Lirkov, M. Paprzycki, M. Ganzha, S. Sedukhin, P. Gepner, Performance analysis of a scalable algorithm for 3D linear transforms, in: M. Ganzha, L. Maciaszek, M. Paprzycki (Eds.), Proceedings of the 2014 Federated Conference on Computer Science and Information Systems, in: Annals of Computer Science and Information Systems, vol. 2, IEEE, 2014, pp. 613–622.