Journal Pre-proof

The inherent overlapping in the parallel calculation of the Laplacian

J. Sánchez-Curto, P. Chamorro-Posada

PII: S1877-7503(23)00005-4
DOI: https://doi.org/10.1016/j.jocs.2023.101945
Reference: JOCS 101945

To appear in: *Journal of Computational Science*

Received date : 1 March 2022
Revised date : 30 November 2022
Accepted date : 12 January 2023

Please cite this article as: J. Sánchez-Curto and P. Chamorro-Posada, The inherent overlapping in the parallel calculation of the Laplacian, *Journal of Computational Science* (2023), doi: https://doi.org/10.1016/j.jocs.2023.101945.

The inherent overlapping in the parallel calculation of the Laplacian

J. Sánchez-Curto[a,*], P. Chamorro-Posada[a]

[a]*Departamento de Teoría de la Señal y Comunicaciones e Ingeniería Telemática, Universidad de Valladolid, ETSI Telecomunicación, Paseo Belén 15, 47011 Valladolid, Spain*

Abstract

A new approach for the parallel computation of the Laplacian in the Fourier domain is presented. This numerical problem inherits the intrinsic sequencing involved in the calculation of any multidimensional Fast Fourier Transform (FFT) where blocking communications assure that its computation is strictly carried out dimension by dimension. Such data dependency vanishes when one considers the Laplacian as the sum of n independent one-dimensional kernels, so that computation and communication can be naturally overlapped with nonblocking communications. Overlapping is demonstrated to be responsible for the speedup figures we obtain when our approach is compared to state-of-the-art parallel multidimensional FFTs.

Key words: Laplacian, overlapping, nonblocking communications

1. Introduction

The Laplacian is a widely used differential operator. In fluid mechanics, for instance, the three dimensional Laplacian accounts for the diffusion of the particles in a fluid, so that it is an essential term in the Navier-Stokes equations [1]. In the study of semiconductors the diffusion of carriers is also ruled by the Laplacian [2]. Another field of physics where the Laplacian plays an essential role is in electrodynamics where the wave equation always includes the Laplacian accounting, this time, for the wave diffraction in space. In the

*Corresponding author
Email address: julsan@tel.uva.es (J. Sánchez-Curto)

field of digital image processing, the Laplacian is a widely used sharpening technique to detect edges and, in turn, enhance image perception [3].

The Laplacian is usually found as a term in Partial Differential Equations (PDEs) which can be solved, among others, by spectral methods [4]. They provide the spatial evolution of the system while the time evolution is preserved, thus turning PDEs into Ordinary Differential Equations (ODEs). In terms of numerics, this offers a dual way of computing space and time, i.e. a time-marching approach based on finite differences for time and a spectral method for space [4]. Focusing our interest on wave propagation, this strategy can be also applied to the study of propagation of quasi-stationary beams, where the role of time is replaced by the longitudinal coordinate that gives the evolution of the beam along the direction of propagation [5].

When Fourier series are used in spectral methods, the Laplacian is numerically computed in the Fourier domain by means of the Fast Fourier Transform (FFT) [1] which has been essential in the field of numerical signal processing since its publication in 1965 [6]. The search of improvements in its performance linked to the reduction in the number of FLOPS was the origin of subsequent works, where alternative solutions to the 2-radix approach, such as the split-radix [7, 8] or the prime factor [9, 10] algorithms were presented. An excellent review of three decades of literature can be found in [11], where any reader can get more insight all relevant aspects of the FFT computation. A second milestone in the development of the FFT was the FFTW library [12], since it showed that the FFT performance is tightly related to an optimal use of hardware resources rather than the strict count of FLOPS of any implementation. Its subsequent success among the scientific community was based on the auto-tuning process that, fully transparent to the the programmer, guaranteed for each computer the optimal implementation.

It is precisely in the parallel computation of the FFT where this connection between hardware and performance can be found since the early years of the supercomputing era. Any kind of parallel system in the market received its corresponding FFT implementation that was designed taking into account the supercomputer architecture. From the pioneering vector supercomputers [13, 14] or hypercubes machines [15–17] to the more recent shared memory systems, such as GPUs [18–20], hundreds of FFT designs and implementations have been reported (see [21] and references therein). Nevertheless, in the search of portable solutions to be run on heterogeneous distributed memory systems, parallel FFTs libraries (see [22] and references therein)

have been developed based on the message passing paradigm.

In this work, however, we do not present yet another contribution on parallel FFTs, since we focus on a different problem, which is the computation of the Laplacian as a whole, i.e. the kernel composed by forward and backward parallel multidimensional FFTs separated by the Laplacian in the Fourier domain. This kernel, usually embedded in the parallel computation of large-scale simulations in the field of fluid mechanics [23], has not deserved any particular attention, so that efficient parallel fluid solvers have exclusively focused on the design of novel parallel FFT libraries [24]. On the other hand, we show here that the kernel posses an intrinsic data independency that can be exploited to overlap computation and communication tasks. Traditionally, parallel multidimensional FFTs are based on the transpose method [25] that strictly alternates local computation stages and communication tasks (data transposes) in a divide-and-conquer strategy. Blocking communications assure that one dimension is computed when only the previous one is complete. This inherent sequencing can be removed when one faces the computation of the Laplacian not as one n-dimensional problem, but as n one-dimensional kernels. Since no data dependency exists between them, the computation of one kernel can take place while the result of a different one is currently being sent. Computation and communications tasks can be naturally overlapped with nonblocking communications.

The nonblocking facilities of parallel libraries were initially proposed to enhance the performance of certain numerical problems in the pioneering works of Hoefler [26–28]. As regards the FFT, most studies targeted at the three dimensional problem (3D FFT), which was considered as an excellent test-bed for exploiting the potential benefits of overlapping, since at least two communications steps are sandwiched among highly demanding computations. In [29], for instance, existing FFTs parallel libraries, such as P3DFFT [30], were redesigned to accommodate the potential overlapping associated to the computation of successive FFTs. In the search of data independency that allows overlapping, some authors have proposed a pipeline technique, i.e. chopping local data into smaller blocks, so that a sub-block can be sent while other is still being processed [31, 32]. Other works, however, have shown that the mere replacement of the nonblocking capabilities of the Message Passing Interface (MPI) standard [33, 34] by other alternatives [35] can enhance the FFT performance. The nonblocking approach has been also incorporated in more specific problems, such as the study of turbulence in fluid dynamics, where overlapping was shown to speedup the numerical solution of Pseudo-

Spectral Simulations [36]. More recently, auto-tuning techniques, which can automatically locate and maximize the overlapping of communication and computation [37, 38], have been also applied to the FFT case [39]. As far as we know, the intrinsic overlapping embedded in the computation of the Laplacian has not been reported in the literature before and constitutes the basic motivation of our work.

This paper is structured as follows. In Sec. 2 we develop our proposal for the two-dimensional case highlighting the differences with the classical approach based on state-of-the-art FFTs. Since overlapping plays a major role we briefly present for this particular problem a simple model for the overlapping that help us understand the performance results. These are displayed in Sec. 3 where series of tests are carried out for different problem sizes and number of cores. Data are analyzed and discussed in Sec. 4 where speedup figures are shown and explained in terms of the theory. Section 5 summarizes the main conclusions and presents the guidelines for future work.

2. A nonblocking approach for the Laplacian

The Laplacian of a two dimensional (2D) function $u(x, y)$ can be calculated in the Fourier domain taking into account the differentiation property of the Fourier transform

$$\nabla^2 u(x,y) \overset{\mathfrak{F}}{\leftrightarrow} -(\Omega_1^2 + \Omega_2^2) U(\Omega_1, \Omega_2) \tag{1}$$

where $U(\Omega_1, \Omega_2)$ accounts for the 2D continuous Fourier Transform of $u(x, y)$, i.e. $u(x, y) \overset{\mathfrak{F}}{\leftrightarrow} U(\Omega_1, \Omega_2)$, while Ω_1 and Ω_2 are the continuous frequencies. In the discrete domain, the numerical computation of the Laplacian can be expressed as a two-step procedure where a 2D forward Discrete Fourier Transform (DFT) is first computed

$$U[k_1, k_2] = \sum_{n_2=0}^{N_2-1} \left(\sum_{n_1=0}^{N_1-1} u[n_1, n_2] \omega_{N_1}^{-n_1 k_1} \right) \omega_{N_2}^{-n_2 k_2}, \tag{2}$$

to be multiplied by $H[k_1, k_2] = -(k_1^2 + k_2^2)$ and converted back to the discrete domain by means of a 2D backward DFT

$$y[n_1, n_2] = \frac{1}{N} \sum_{k_2=0}^{N_2-1} \left(\sum_{k_1=0}^{N_1-1} H[k_1, k_2] U[k_1, k_2] \omega_{N_1}^{n_1 k_1} \right) \omega_{N_2}^{n_2 k_2}. \tag{3}$$

In Eq. (2) $u[n_1, n_2]$ is a 2D array that results from the sampling of $u(x, y)$ over a finite square with $N = N_1 \times N_2$ data points, so that n_1 and n_2 are integers verifying $0 \leq n_1 < N_1$ and $0 \leq n_2 < N_2$. In Eq. (3) $y[n_1, n_2]$ contains the 2D Laplacian, k_1 and k_2 are the discrete frequencies and $\omega_{N_j} = e^{i2\pi/N_j}$ for $j = 1, 2$.

Our approach is based on the special form of $H[k_1, k_2]$ that can be considered as a particular case of those functions verifying $H[k_1, k_2] = H[k_1]+H[k_2]$. Taking into account the linear property of the DFT, the substitution of Eq.(2) in Eq.(3) turns the right side of Eq. (3) into

$$\frac{1}{N}\sum_{k_2=0}^{N_2-1}\left(\sum_{k_1=0}^{N_1-1} H[k_1]\left\{\sum_{n_2=0}^{N_2-1}\left(\sum_{n_1=0}^{N_1-1} u[n_1, n_2]\omega_{N_1}^{-n_1k_1}\right)\omega_{N_2}^{-n_2k_2}\right\}\omega_{N_1}^{n_1k_1}\right)\omega_{N_2}^{n_2k_2} \quad + \\ \frac{1}{N}\sum_{k_2=0}^{N_2-1}\left(\sum_{k_1=0}^{N_1-1} H[k_2]\left\{\sum_{n_2=0}^{N_2-1}\left(\sum_{n_1=0}^{N_1-1} u[n_1, n_2]\omega_{N_1}^{-n_1k_1}\right)\omega_{N_2}^{-n_2k_2}\right\}\omega_{N_1}^{n_1k_1}\right)\omega_{N_2}^{n_2k_2}. \quad (4)$$

In Eq. (4), the forward and backward transforms along one dimension cancel each other since $H[k_1]$ and $H[k_2]$ do not depend on k_2 and k_1, respectively, so that Eq. (3) can be rewritten as

$$\begin{aligned} y[n_1, n_2] &= y_c[n_1, n_2] + y_r[n_1, n_2] \\ &= \frac{1}{N_1}\sum_{k_1=0}^{N_1-1}\left(H[k_1]\sum_{n_1=0}^{N_1-1} u[n_1, n_2]\omega_{N_1}^{-k_1n_1}\right)\omega_{N_1}^{k_1n_1} \\ &+ \frac{1}{N_2}\sum_{k_2=0}^{N_2-1}\left(H[k_2]\sum_{n_2=0}^{N_2-1} u[n_1, n_2]\omega_{N_2}^{-k_2n_2}\right)\omega_{N_2}^{k_2n_2}, \end{aligned} \quad (5)$$

where $y_c[n_1, n_2]$ and $y_r[n_1, n_2]$ account for the 1D kernel along the columns and rows of the matrix, respectively. Eq. (5) simply represents the calculation of $\nabla^2 = \partial^2/\partial x^2 + \partial^2/\partial y^2$ in the Fourier domain where each dimension is independently calculated. The 1D kernel involves, in the case of $y_r[n_1, n_2]$, N_1 forward and backward DFTs of length N_2 where a point-to-point complex multiplication by $H[k_2]$ is sandwiched between them. An analogous definition addresses $y_c[n_1, n_2]$. The four nested sums involved in Eqs. (2) and (3) are reduced to two in Eq. (5), thus turning a single 2D problem into two

1D problems. Only the explicit sum of $y_r[n_1, n_2]$ and $y_c[n_1, n_2]$ makes the Laplacian depend on both partial results. Of course, this procedure can be extended to $n > 2$ dimensions.

The number of FLOPS in Eq.(2) and Eq. (3) when compared to Eq.(5) is, in both cases, $\mathcal{O}(N \log N)$. Nevertheless, the potential benefits of Eq. (5) are revealed when one evaluates them in terms of parallel computing.

2.1. Parallel computing evaluation

The parallel computation of the Laplacian according to Eqs. (2) and (3) is illustrated in Fig. 1(a) where a slab (1D) distribution of processors is assumed, so that rows are locally stored and columns are distributed among $P = 6$ processors. Figure 1(a) shows the timeline corresponding to all the operations involved, i.e. 2D forward and backward FFTs separated by a point-to-point multiplication that represents the Fourier transformation of the Laplacian (solid black circles in the middle of the figure). Each 2D FFT involves the three operations enclosed by the horizontal parentheses, i.e. the computation of forward 1D FFTs to the rows, a data transposition and the computation of forward 1D FFTs to data that were initially spread out among processors. The final transpose is saved as the Fourier transformation of the Laplacian is transposed [40], so that the backward 2D FFTs starts with the computation of the columns, instead of rows. One can proceed to compute a new dimension when only the previous one is completed. This imposes a strict sequentially between computation and communication which is guaranteed by blocking communications.

The solution we propose in Eq. (5) is represented in Fig. 1(b) where the two 1D kernels of Eq. (5) are represented. Each 1D kernel involves forward and backward 1D FFTs separated by the point-to-point multiplication by $H[k_1]$ or $H[k_2]$ (black solid circles in the middle of the two kernels). An initial double data distribution allocates N_1/P rows and N_2/P columns in each processors, and both kernels are locally computed after that. The solely scattering of N_1/P rows of a global array among processors would be inefficient in this case, since a subsequent data transpose should follow the initial scattering to get the columns in each processor. Two consecutive collective communications would be necessary on data which remain the same.

Figure 1 also shows that a data movement is necessary to transpose the result of one 1D kernel and perform the final sum in Eq. (5). This data movement and the computation of the second 1D kernel can take place simultaneously, since no data dependency exists. Communication and computation

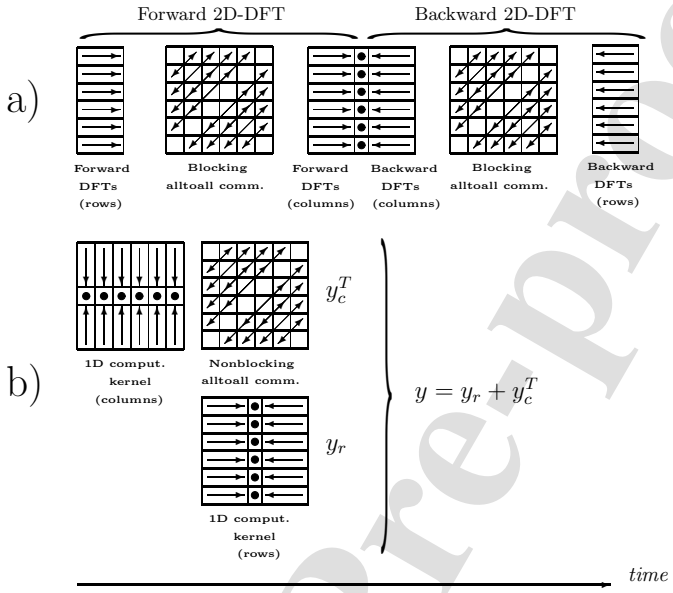

Figure 1: Blocking (a) and nonblocking (b) approaches for the parallel computation of the Laplacian in the Fourier domain. The vertical arrows in the bottom picture represent the kernel computation along the columns which are also local to each processor.

tasks can be overlapped based on the use of nonblocking communications, as it is illustrated in Fig. 1(b). The overlapped solution arises naturally from Eq. (5), in contrast to other nonblocking approaches for the FFT [32] where overlapping is obtained by the segmentation of large tasks. As the timeline of Fig. 1 suggests, our approach should lead to a reduction in the elapsed time when compared to the general approach represented in Fig. 1(a).

2.2. A simple model for overlapping

The impact of overlapping on the elapsed time for this particular problem can be predicted based on a simple model we present in Fig. 2. The number of processors P that a parallel system dedicates to solve this problem and the time involved in its solution are displayed in the x and y axis, respectively. We decompose the elapsed time $t_{elapsed}$ (solid black line) as the sum of two complementary contributions, i.e. computation t_{comp} and communication t_{comm} which are represented with dashed black lines in Fig. 2. In this case,

we assume that $t_{comp} \sim P^{-1}$ while $t_{comm} \sim P^{\gamma}$ with $\gamma > 1$ depending on the number of cores per node [41].

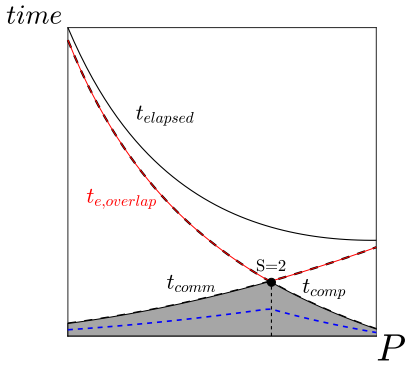

Figure 2: Theoretical enhancement of overlapping for this particular problem. Elapsed time for a solution with (red) or without (black) overlapping. Maximum theoretical speedup is achieved when computation and communication times are the same.

The grey area under both lines represents the potential overlapping, so that $t_{overlap} = min\{t_{comm}, t_{comp}\}$. The elapsed time for a solution with overlapping $t_{e,overlap} = t_{elapsed} - t_{overlap} = max\{t_{comp}, t_{comm}\}$ is represented with a red solid line which is superimposed on the dotted black one. The red line shows a minimum when $t_{comm} = t_{comp}$, thus obtaining the maximum theoretical speedup $S = t_{elapsed}/t_{e,elapsed} = 2$. On the other hand, in those systems when $t_{comm} \gg t_{comp}$ or vice versa, overlapping will not provide any significant improvement. Such is the case, for instance, of GPUs where communication time prevails over computation time in large-scale FFTs [42].

The model of Fig. 2 represents an ideal situation where we have assumed that the whole code of any numerical problem can be overlapped and all system resources could be scheduled to guarantee that communication and computation demands can be fulfilled simultaneously. A more realistic analysis should take into account these issues, as well as hardware characteristics such as the cache size. The real overlapping must thus lie beyond the theoretical one, as it is represented in Fig. 2 by the dotted blue line inside the grey area.

3. Performance results

Table 1 illustrates the code that develops the scheme shown in Fig. 1(b). While the lines in red italic font correspond to the inter-core explicit communication, the instructions in blue regular font account for the computation of the two 1D kernels, the final sum of Eq. (5) and the local data movements that must take place before and after each transmission to complete a parallel transpose [25]. Although these intra-core data movements are essential in the communication process, it can not be overlapped with the explicit communication. The code sandwiched between the nonblocking functions (in red italic font) constitute the overlapped section of our approach.

```
rows ← global                          /* local data allocation of N1/P rows*/
columns ← global                       /* local data allocation of N2/P columns */
                                       /* 1D kernel:  */
  fftwexecute(plan1,columns)           /* N2/P forward FFTs of N1 points */
  columns ← H[k1]· columns             /* point-to-point multiplication */
  fftwexecute(plan2,columns)           /* backward FFTs */

columnsT← columns                      /* local data movement before sending */
MPI_Ialltoall(columnsT,...,columns,req) /* nonblocking sending of data */
                                       /* 1D kernel:  */
  fftwexecute(plan3,rows)              /* N1/P forward FFTs of N2 points */
  rows ← H[k2]· rows                   /* point-to-point multiplication */
  fftwexecute(plan4,rows)              /* backward FFTs */

MPI_Wait(req)
columnsT ← columns                     /* local data movement after reception */
rows ← rows + columnsT                 /* y ← yr + yc */
```

Table 1: Code implementing Eq. (5) with nonblocking communications.

The implementation of our proposal has been written in `C` and communication issues have relied on the MPI library [43]. We have employed the nonblocking version of the all-to-all communication (`MPI_Ialltoall`) and the `FFTW` library [12] to perform the forward and backward FFTs involved in the 1D kernel. The data allocation resources of the `FFTW-MPI` have also been used to provide the distribution of data not only among the columns, but also the rows of a global array.

The local data movements before and after the all-to-all communication are essentially data block transposes that can be very time demanding and dominate the overall time for large problem sizes in small scale systems [32]. This crucial issue arises when one faces the design of any multidimensional FFT, but data transpose is by itself a subject in parallel computing [25, 44] and deserves a great attention when large amount of data are involved [45].

The access to non-contiguous large amount of data is prone to cause cache misses [46] which are specially harmful in terms of performance. Cache-oblivious algorithms have been proposed to minimize such effect regardless of the cache size [47]. They propose data reordering based on Morton order [48] and have have shown their effectiveness not only on data transpose and linear algebra problems [49], but also in the design of collective MPI-based communications [50]. In our work, instead of a naive approach for data transpose, we have used cache-oblivious algorithms [47] where the optimum sub-block or matrix size has been experimentally chosen.

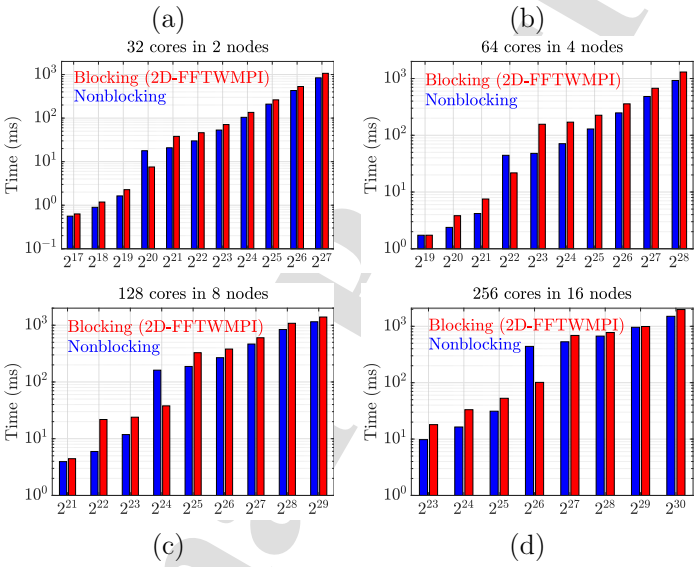

Figure 3: Elapsed time of our proposal (blue) and the conventional scheme (red) for different problem sizes. The number of cores is 32 (a), 64 (b), 128 (c) and 256 (d). Speedup figures are displayed in Fig. 6.

We compare our implementation with the code summarized in Tab. 2 that shows the computation of the Laplacian based on the classical approach represented by Eqs. (2) and (3). The computation of 2D FFTs relies on the `FFTW-MPI` [12] that constitutes an excellent reference for 2D problems like this, where only a 1D (slab) data distribution among processors is reasonable. This library has also developed its own specific blocking all-to-all

functions that replace the standard MPI counterpart, so that the communication process can also be tuned [12]. As it can be seen in Tab. 2, we call the `FFTW-MPI` functions with the appropriate flags [12] in order to save innecessary data transposes.

```
plan1T=fftw_mpi_plan_dft_2d(...,FFTW_MPI_TRANSPOSED_OUT);   /*plan for a 2D forward FFT */
plan2T=fftw_mpi_plan_dft_2d(....,FFTW_MPI_TRANSPOSED_IN);   /*plan for a 2D backward FFT */

rows ← global                  /* local data allocation of N_1/P rows */

  fftwexecute(plan1T)          /* 2D forward FFTs of N points without final transpose */
  plan2T ← H^t[k_1,k_2]·plan1T /* multiplication by the transposed Laplacian */
  fftwexecute(plan2T)          /* 2D backward FFTs of N points to transposed data */
```

Table 2: Code implementing Eqs. (2) and (3) with blocking communications.

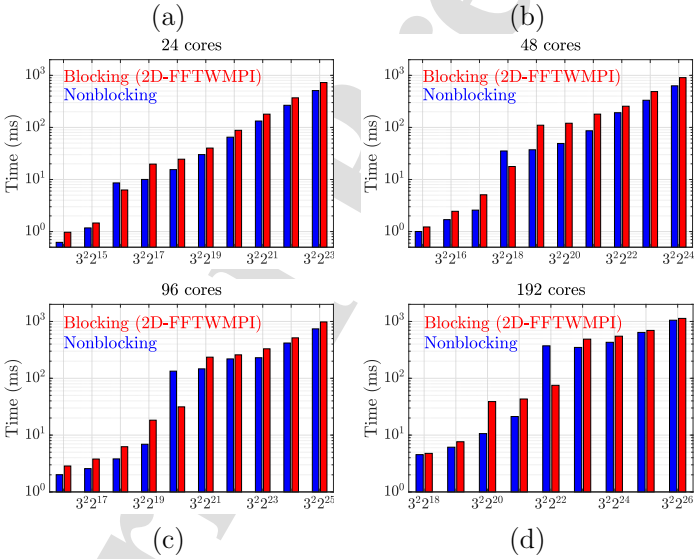

Figure 4: Elapsed time of our proposal (blue) and the conventional scheme (red) for different problem sizes. The number of cores is 24 (a), 48 (b), 96 (c) and 192 (d). Speedup figures are displayed in Fig. 7.

Our work has been carried out at the *Castilla y León supercomputing centre, SCAYLE*. We have access to the Haswell cluster which is composed of 114 nodes and each node has 2 octa-core Intel Xeon E5-2630 v3 (Haswell)

processors running at 2.40 GHz. Each node has a 32 GB RAM memory and are interconnected through an Infiniband FDR network at 56 Gb/s. Series of tests have been carried out to compare the performance of the codes described in Tabs. 1 and 2.

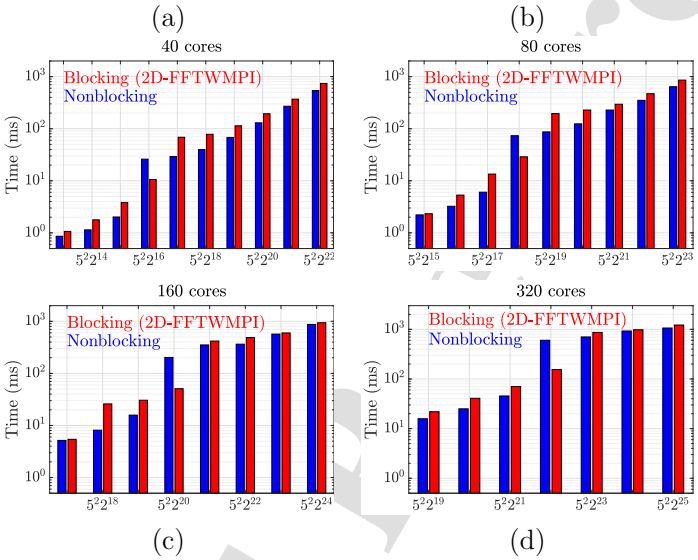

Figure 5: Elapsed time of our proposal (blue) and the conventional scheme (red) for different problem sizes. The number of cores is 40 (a), 80 (b), 160 (c) and 320 (d). Speedup figures are displayed in Fig. 8.

Figures 3, 4 and 5 show the elapsed time of both solutions for different problem sizes $N = N_1 \times N_2$ and number of cores. In Fig. 3 we have assumed that the number of cores P is a power of two, so that $N = 2^k$ where $N_1 = N_2$ if k is even, and $N_1 = 2N_2$ otherwise. Nevertheless, this condition on P is very restrictive, since most computing systems hardly scale as a power of two. We have thus performed tests for ordinary scenarios where P decomposes as prime factors different from two. Since the number of data points in the matrix must verify $N/P^2 \in Z$ [12], non-power-of-two Discrete Fourier Transforms (DFTs) must be employed, which always has a negative impact on performance. To minimize this effect, we choose P containing one single prime factor different from two. As one can see in Fig. 4, while the number of cores evolves as $P = 3 \cdot 2^m$ with $m = 3, 4, 5, 6$, the number of data points

is $N = 3^2 2^k$ with k assuming similar values as those of Fig. 3. A similar argument is valid for Fig. 5, where we have replaced $P = 3$ by $P = 5$.

Elapsed times approximately span over three orders of magnitude, so that a logarithmic plot has been used in the y-axis. The time represented in the y-axis is the mean value that results from calculating the Laplacian 50 times, in order to assure the `FFTW` has finished the warm-up iterations and has chosen a definitive and optimal solution among all possibilities tested. Since the number of nodes we can access in the cluster is limited, we have employed the maximum number of cores per node (16 cores) in order to study the scalability of our solution.

4. Discussion

With the exception of one case we comment below, our nonblocking solution exhibits a better performance than the blocking approach based on the 2D-FFTWMPI. Since the computational load of both solutions is quite similar and relies on the same `FFTW` library, the enhancement lies on the communications side. All pictures in Figs. 3, 4 and 5 also show that our design scales with the number of cores.

All plots in Figs. 3, 4 and 5 undergo a significant fall in performance of both solutions that does not takes place simultaneously. Our solution dramatically gets worse when $N/P^2 = 2^{10}$, while the 2D-FFTWMPI replicates such behaviour when $N/P^2 = 2^{11}$. As it is confirmed in the tests that only evaluate communication time (see Fig. 9), it is a communication issue. Intranode data communication, that prevails in our tests, is performed through memory copies of data on a shared memory space [50, 51] which may eventually get full. Larger memory demands are fulfilled through cache misses which have a very negative impact on the performance, thus masking any other possible improvement [46]. This can also explain why our solution get worse first when compared to the 2D-FFTWMPI. Our proposal doubles the amount of local data, thus increasing the memory demands when compared to a conventional solution.

4.1. Speedup

The logarithmic plots of Sec. 3 provide a general overview of the performance of both solutions, but they do not allow us to quantify the speedup. Speedup figures from Figs. 3, 4 and 5 are plotted in Figs. 6, 7 and 8, respectively. We have highlighted in red the exact value in each series where the

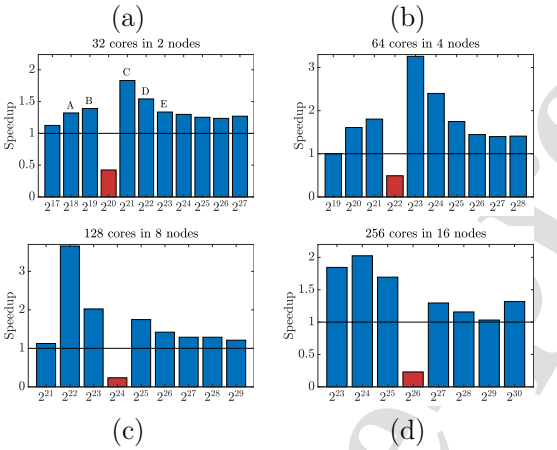

Figure 6: Speedup figures for the cases of Fig. 3.

2D-FFTWMPI behaves better than our solution. The maximum values are slightly greater than 2 in certain cases because our solution not only benefits from overlapping, but also from the replacement of one data transpose by a double initial data allocation. When both factors occur simultaneously, one gets such figures.

4.2. Analysis of results

In order to explain such figures, we have carried out a second series of tests that analyse the code in Tab. 1 evaluating separately computation and communication. Results are displayed in Fig. 9 where we have interpolated the data series with a 1:8 ratio to focus on the shape of the curves. Although we have analysed the four cases of Fig. 3, we only show the case of 32 cores since the behaviour is quite similar for a different number of cores. The overall elapsed, computation and communication times are represented with black squares, blue diamonds and red points, respectively. We have also added a magenta line with triangles which is the sum of the blue and red ones, so that the difference between the black and magenta lines represents the time saved by overlapping. This difference is more significant in the five cases labelled with upper-case letters in Fig. 9 that correspond to the bars of Fig. 6(a) where speedup reaches the largest values. Overlapping is thus responsible for these speedup figures.

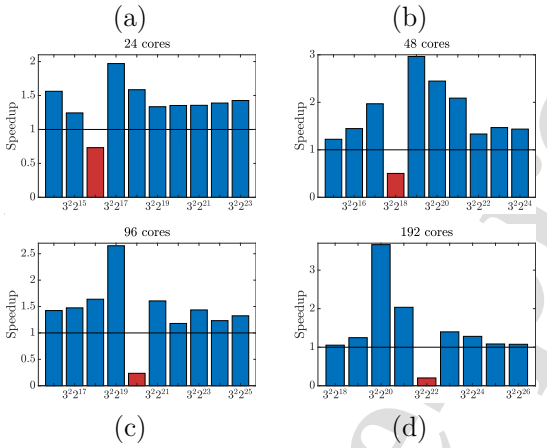

Figure 7: Speedup figures for the cases of Fig. 4.

The prediction of the theoretical model of Fig. 2 reinforces this idea. According to the model, speedup is maximum when computation and communication times are comparable. This is precisely what the labeled cases in Fig. 9 show, where the blue and red lines cross or have a very similar value.

Figure 9 also shows that elapsed time is dominated by communications or computation depending on the the problem size. While computation time evolves according to the computational load $\mathcal{O}(N \log N)$, communication time is severely affected by the cache miss we have analysed before. For large values of N, the black and magenta lines are superimposed and the potential benefit of overlapping vanishes. This was also predicted by the model of Fig. 2 since one enters a regime where $t_{comp} \gg t_{comm}$.

5. Conclusions and future work

This work has presented a new approach for the parallel computation of the Laplacian in the Fourier domain. In contrast to the classical approach based on computing successive multidimensional FFTs where blocking communications separate local computations, we have presented a solution that computes the Laplacian as n distinct one dimensional kernels. Communication and computation are overlapped by means of nonblocking communications, thus removing the inherent sequentially of the classical approach. This

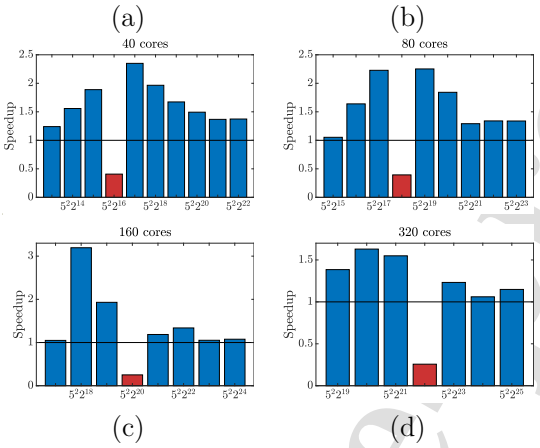

Figure 8: Speedup figures for the cases of Fig. 5.

higher level of parallelism has led to a general better performance with the exception of very particular cases. This work also shows how the benefits of overlapping are limited to scenarios where computation and communications times are comparable. This balance will depend on the problem size, parallel resources and hardware characteristics of the nodes and it should be an essential programmer task to find it out and work around it.

Our work has presented the performance results of the 2D Laplacian based on a slab (1D) decomposition for a small-scale cluster. The extension of the problem to the 3D case depends on the cluster size. Rewriting the 3D Laplacian as $\nabla^2 = \nabla_T^2 + \partial^2/\partial z^2$ where T denote the two transverse coordinates (x and y) one has a straightforward mapping for the slab decomposition which is suitable for a small-scale cluster. The expected speedup should not be very different from the figures we have presented in this work. However, in the case we had access to a large-scale cluster, a pencil (2D) decomposition should be considered instead, like in other similar 3D problems such as the 3D FFT [52]. An estimation of the speedup in such scenario can be calculated based on the code percentage that can be overlapped. While the ratio between the number of kernel computations and communication steps is 2:1 in the 2D case (see the code in Table 1), it would turn into a 3:2 ratio in the 3D case. Communication demands grow in relation to the computational kernels as the number of dimensions goes up, thus increasing the amount

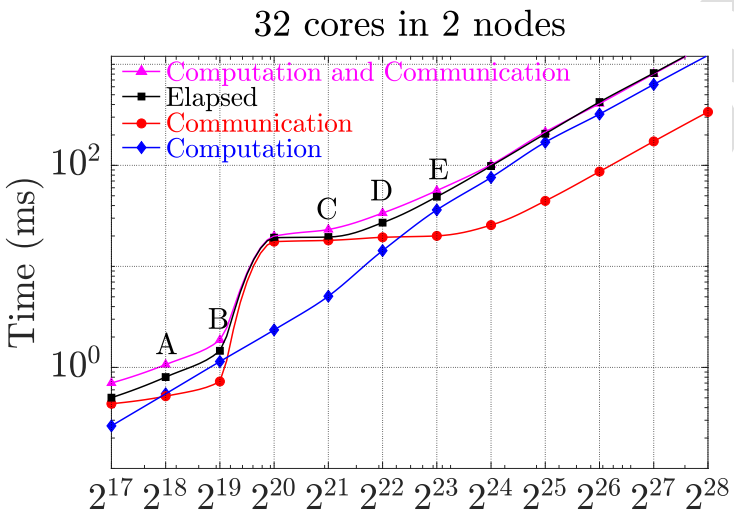

Figure 9: Elapsed (black squares), communication (red points) and computation (blue diamonds) times. Magenta triangles represent the the sum of the blue and red ones.

of code that is prone to be overlapped. As a result, higher speedup figures should be expected for the 3D Laplacian on a pencil decomposition.

This work was supported by the Consejería de Educación, Junta de Castilla y León [grant number VA296P18].

References

[1] C. Canuto, M. Hussaini, A. Quarteroni, T. Zhang, Spectral Methods in Fluid Dynamics, Springer, Berlin, 1988.

[2] E. M. Wright, W. J. Firth, I. Galbraith, Beam propagation in a medium with a diffusive Kerr-type nonlinearity, J. Opt. Soc. Am. B 2 (2) (1985) 383–386. doi:https://doi.org/10.1364/JOSAB.2.000383.

[3] R. C. Gonzalez, R. E. Woods, Digital Image Processing, Prentice Hall, Upper Saddle River, N.J., 2008.

[4] J. P. Boyd, Chebyshev and Fourier spectral methods, Courier Corporation, 2001.

[5] P. Chamorro-Posada, G.S. McDonald, G.H.C. New, Non-paraxial beam propagation methods, Optics Communications 192 (1) (2001) 1–12. doi:https://doi.org/10.1016/S0030-4018(01)01171-3.

[6] J.W. Cooley, J.W. Tukey, An Algorithm for the Machine Calculation of Complex Fourier Series, Mathematics of Computation 19 (1965) 297–301.

[7] P. Duhamel, H. Hollmann, Split radix FFT algorithm, Electronics Letters 20 (1984) 14–16(2).

[8] H. Sorensen, M. Heideman, C. Burrus, On Computing the Split-Radix FFT, IEEE Transactions on Acoustics, Speech, and Signal Processing 34 (1) (1986) 152–156. doi:10.1109/TASSP.1986.1164804.

[9] S. Winograd, On Computing the Discrete Fourier Transform, Mathematics of Computation 32 (141) (1978) 175–199.

[10] H. Silverman, An introduction to programming the Winograd Fourier transform algorithm (WFTA), IEEE Transactions on Acoustics, Speech, and Signal Processing 25 (2) (1977) 152–165. doi:10.1109/TASSP.1977.1162924.

[11] C. van Loan, Computational Frameworks for the Fast Fourier Transform, Society for Industrial and Applied Mathematics SIAM, Philadelphia PA, 1992.

[12] M. Frigo, S. G. Johnson, The Design and Implementation of FFTW3, Proceedings of the IEEE 93 (2) (2005) 216–231. doi:https://doi.org/10.1109/JPROC.2004.840301.

[13] P. N. Swarztrauber, FFT algorithms for vector computers, Parallel Computing 1 (1) (1984) 45–63. doi:https://doi.org/10.1016/S0167-8191(84)90413-7.

[14] D. A. Carlson, Ultrahigh-performance FFTs for the CRAY-2 and CRAY Y-MP supercomputers, The Journal of Supercomputing 6 (1992) 107–116. doi:https://doi.org/10.1007/BF00129773.

[15] P. N. Swarztrauber, Multiprocessor FFTs, Parallel Computing 5 (1) (1987) 197–210. doi:https://doi.org/10.1016/0167-8191(87)90018-4.

[16] C. Tong, P. N. Swarztrauber, Ordered Fast Fourier Transforms on a Massively Parallel Hypercube Multiprocessor, Journal of Parallel and Distributed Computing 12 (1) (1991) 50–59. doi:https://doi.org/10.1016/0743-7315(91)90028-8.

[17] P. N. Swarztrauber, S. W. Hammond, A comparison of optimal FFTs on torus and hypercube multicomputers, Parallel Computing 27 (6) (2001) 847–859. doi:https://doi.org/10.1016/S0167-8191(00)00107-1.

[18] K. Moreland, E. Angel, The FFT on a GPU, in: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, 2003, pp. 112–119.

[19] A. Nukada, S. Matsuoka, Auto-tuning 3-D FFT library for CUDA GPUs, in: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, 2009, pp. 1–10. doi:10.1145/1654059.1654090.

[20] A. Nukada, K. Sato, S. Matsuoka, Scalable multi-GPU 3-D FFT for TSUBAME 2.0 Supercomputer, in: SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, 2012, pp. 1–10. doi:10.1109/SC.2012.100.

[21] O. Ayala, L.-P. Wang, Parallel implementation and scalability analysis of 3D Fast Fourier Transform using 2D domain decomposition, Parallel Computing 39 (1) (2013) 58–77. doi:https://doi.org/10.1016/j.parco.2012.12.002.

[22] L. Dalcin, M. Mortensen, D. E. Keyes, Fast parallel multidimensional FFT using advanced MPI, Journal of Parallel and Distributed Computing 128 (2019) 137–150. doi:https://doi.org/10.1016/j.jpdc.2019.02.006.

[23] M. Yokokawa, K. Itakura, A. Uno, T. Ishihara, Y. Kaneda, Tflops Direct Numerical Simulation of Turbulence by a Fourier Spectral Method on the Earth Simulator, in: SC'02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing, IEEE, 2002, pp. 50–50.

[24] A. G. Chatterjee, M. K. Verma, A. Kumar, R. Samtaney, B. Hadri, R. Khurram, Scaling of a Fast Fourier Transform

and a pseudo-spectral fluid solver up to 196608 cores, Journal of Parallel and Distributed Computing 113 (2018) 77–91. doi:https://doi.org/10.1016/j.jpdc.2017.10.014.

[25] A. Grama, A. Gupta, G. Karypis, V. Kumar, Introduction to Parallel Computing, Addison Wesley, Harlow, England, 2003.

[26] T. Hoefler, A. Lumsdaine, W. Rehm, Implementation and performance analysis of non-blocking collective operations for MPI, in: SC'07: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, IEEE, 2007, pp. 1–10. doi:https://doi.org/10.1145/1362622.1362692.

[27] T. Hoefler, P. Kambadur, R. L. Graham, G. Shipman, A. Lumsdaine, A case for standard non-blocking collective operations, in: Proceedings of the 14th European Conference on Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer-Verlag, Berlin, Heidelberg, 2007, p. 125–134.

[28] T. Hoefler, P. Gottschling, A. Lumsdaine, Leveraging non-blocking collective communication in high-performance applications, in: Proceedings of the Twentieth Annual Symposium on Parallelism in Algorithms and Architectures, SPAA '08, Association for Computing Machinery, New York, NY, USA, 2008, p. 113–115. doi:10.1145/1378533.1378554. URL https://doi.org/10.1145/1378533.1378554

[29] K. Kandalla, H. Subramoni, K. Tomko, et al., High-performance and scalable non-blocking all-to-all with collective offload on InfiniBand clusters: a study with parallel 3D FFT, Comput. Sci. Res. Dev. 26 (2011) 237. doi:https://doi.org/10.1007/s00450-011-0170-4.

[30] D. Pekurovsky, P3DFFT: A Framework for Parallel Computations of Fourier Transforms in Three Dimensions, SIAM Journal on Scientific Computing 34 (4) (2012) C192–C209. doi:https://doi.org/10.1137/11082748X.

[31] S. Song, J. K. Hollingsworth, Designing and Auto-Tuning Parallel 3-D FFT for Computation-Communication Overlap, in: Proceedings of the 19th ACM SIGPLAN symposium on Principles and practice of parallel programming, 2014, pp. 181–192.

[32] S. Song, J. K. Hollingsworth, Computation–communication overlap and parameter auto-tuning for scalable parallel 3-D FFT, Journal of Computational Science 14 (2016) 38–50. doi:https://doi.org/10.1016/j.jocs.2015.12.001.

[33] W. Gropp, E. Lusk, A. Skjellum, Using MPI: Portable Programming with the Message Passing Interface, The MIT Press, Cambridge, Massachussets, 1999.

[34] W. Gropp, E. Lusk, A. Skjellum, Using MPI-2: Advanced Features of the Message Passing Interface, The MIT Press, Cambridge, Massachussets, 1999.

[35] R. Nishtala, P. H. Hargrove, D. O. Bonachea, K. A. Yelick, Scaling Communication-Intensive Applications on BlueGene/P Using One-Sided Communication and Overlap, in: 2009 IEEE International Symposium on Parallel and Distributed Processing, 2009, pp. 1–12. doi:10.1109/IPDPS.2009.5161076.

[36] J. H. Göbbert, H. Iliev, C. Ansorge, H. Pitsch, Overlapping of communication and computation in nb3dfft for 3d fast fourier transformations, in: Jülich Aachen Research Alliance (JARA) High-Performance Computing Symposium, Springer, 2016, pp. 151–159.

[37] Y. Barigou, On Communication-Computation Overlap in High-Performance Computing, Master's thesis, University of Houston (May 2016).

[38] Y. Barigou, E. Gabriel, Maximizing Communication–Computation Overlap Through Automatic Parallelization and Run-time Tuning of Non-blocking Collective Operations, International Journal of Parallel Programming 45 (6) (2017) 1390–1416.

[39] D. Takahashi, Automatic Tuning of Computation-Communication Overlap for Parallel 1-D FFT, in: 2016 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC) and 15th International Symposium on Distributed Computing and Applications for Business Engineering (DCABES), 2016, pp. 253–256. doi:10.1109/CSE-EUC-DCABES.2016.193.

[40] R.C. Agarwal, F.G. Gustafson, M. Zubair, A high performance parallel algorithm for 1-D FFT, in: Proceedings of the 1994 Conference on Supercomputing, IEEE Computer Society Press, 1994, pp. 34–40.

[41] R. Kumar, A. Mamidala, D. K. Panda, Scaling alltoall collective on multi-core systems, in: 2008 IEEE International Symposium on Parallel and Distributed Processing, 2008, pp. 1–8. doi:https://doi.org/10.1109/IPDPS.2008.4536141.

[42] Y. Chen, X. Cui, H. Mei, Large-Scale FFT on GPU Clusters, in: Proceedings of the 24th ACM International Conference on Supercomputing, ICS '10, Association for Computing Machinery, New York, NY, USA, 2010, p. 315–324. doi:10.1145/1810085.1810128.

[43] M. Snir, S. Otto, S.-H. Lederman, D. Walker, J. Dongarra, MPI–The Complete Reference: Volume 1, The MPI Core, The MIT Press, Boston, Massachusetts, 1998.

[44] J. Choi, J. Dongarra, D. Walker, Parallel matrix transpose algorithms on distributed memory concurrent computers, Parallel Computing 21 (9) (1995) 1387–1405. doi:https://doi.org/10.1016/0167-8191(95)00016-H.

[45] J. J. Suh, V. Prasanna, An efficient algorithm for out-of-core matrix transposition, IEEE Transactions on Computers 51 (2002) 420–438. doi:https://doi.org/10.1109/12.995452.

[46] J. L. Hennessy, D. A. Patterson, Computer Architecture: A Quantitative Approach, 5th Edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2011.

[47] M. Frigo, C. E. Leiserson, H. Prokop, S. Ramachandran, Cache-Oblivious Algorithms, ACM Trans. Algorithms 8 (1) (2012) 1–22. doi:https://doi.org/10.1145/2071379.2071383.

[48] G. M. Morton, A computer orientated Geodetic Data Base; and a New Technique in File Sequencing, Tech. rep., IBM Ltd (1966).

[49] J. Thiyagalingam, O. Beckmann, P. H. J. Kelly, Is Morton layout competitive for large two-dimensional arrays yet?, Concurr. Comput.: Pract. Exper. 18 (11) (2006) 1509–1539. doi:https://doi.org/10.1002/cpe.1018.

[50] S. Li, Y. Zhang, T. Hoefler, Cache-oblivious MPI All-to-All communications based on morton order, IEEE Transactions on Parallel and Distributed Systems 29 (3) (2018) 542–555. doi:https://doi.org/10.1109/TPDS.2017.2768413.

[51] S. Ramos, T. Hoefler, Modeling communication in cache-coherent SMP systems: a case-study with Xeon Phi, in: HPDC 2013 - Proceedings of the 22nd ACM International Symposium on High-Performance Parallel and Distributed Computing, 2013, pp. 97–108. doi:https://doi.org/10.1145/2462902.2462916.

[52] D. Takahashi, An Implementation of Parallel 3-D FFT with 2-D Decomposition on a Massively Parallel Cluster of Multi-core Processors, in: Parallel Processing and Applied Mathematics, Springer Berlin Heidelberg, 2010, pp. 606–614. doi:https://doi.org/10.1007/978-3-642-14390-8_63.

- Inherent overlapping in the parallel calculation of the Laplacian
- Nonblocking communications benefit from data independency
- Nonblocking approaches lead to better performance results than blocking solutions

Author Biography

Julio Sanchez-Curto received his Ph.D. in 2009 from the University of Valladolid. He joined the Department of Signal Theory and Communications at the University of Valladolid, where he holds an Associate Professor position since 2018. His main research interests include microwave engineering, optical solitons, nonlinear interfaces and the development of parallel algorithms for nonlinear propagation problems.

Conflict of Interest

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: