Bull. Nov. Comp. Center, Num. Anal., 17 (2015), 61–67

Optimized parallel algorithm for solving the Poisson equation in non-stationary problems*

N.V. Snytnikov

Abstract. We compare two algorithms for solving the Poisson equation: the first one is based on domain decomposition with direct coupling of subdomains (DDCS) and the second one is based on multidimensional Fast Fourier Transform and data transposition (FFTT).

Results of the comparison made helped us to introduce several optimizations for the DDCS method and to develop a new method combining DDCS/FFTT, that is aimed at solving the 3D Poisson equation on grids with billions of nodes using thousands of processors.

1. Introduction

Simulating the dynamics of stars and gas in the galaxies [1–3] or circumstellar disks [4] requires to solve the 3D Poisson equation for a gravitational potential. It is necessary to use grids that could provide a fine spatial resolution (1024^3, 2048^3, and higher).

Such grids cannot be placed in the memory of a single processor. Therefore some kind of computational domain decomposition is needed. At the same time the decomposition must provide minimal communications between processors, because interconnections are considered as a main performance bottleneck in supercomputer simulations.

In [5, 6], we proposed an algorithm for solving 2D and 3D Poisson equations in the context of non-stationary problems (DDCS). It is based on coupling the adjacent subdomains using the single-layer potential method [7] and pre-computation of auxiliary values in the Fourier expansion of a single layer potential (that is similar to the idea proposed in [8]).

In this paper, we compare two algorithms for solving the 2D Poisson equation: the DDCS method and *de facto* a standard method of multidimensional Fast Fourier Transform with Transposition (FFTT). The latter one is simple, highly efficient and easy to implement. So, the comparison is useful because it helps one to identify possible issues with the DDCS and gives an idea of comparative performance measured in absolute units.

A detailed description of the DDCS algorithm can be found in [5, 6], therefore we do not discuss it here. In Section 2, we briefly describe the

*Supported by the RFBR under Grants 14-01-31088, 14-07-00241. Numerical experiments were conducted at the Siberian Supercomputer Center, Joint Supercomputer Center, and MSU Lomonosov supercomputer.

FFTT algorithm, discuss the comparison results for DDCS and FFTT, and propose several optimizations for DDCS. In Section 3, we outline a general scheme of the hybrid DDCS/FFTT method.

2. Comparison with a parallel FFT-based algorithm

The 2D FFTT algorithm was implemented in the following way. The computational domain Ω with $L_x \times L_y$ grid domation is divided in X direction into N equal subdomains Ω_n. One processor is assigned to treat each subdomain. So, the total number of processors is N.

FFTT Algorithm:

1. On each processor apply one-dimensional Fast Sine Transform to the grid density function in Y direction (FSTY).
2. Perform data transposition from Y to X direction: subdivide a grid function into blocks and transfer them to all the other processors. It corresponds to the MPI procedure MPI_Alltoall.
3. On each processor apply FSTX.
4. Multiply resulting values by corresponding values of wave numbers.
5. Apply the inverse FSTX.
6. Perform data transposition from X to Y direction.
7. Apply the inverse FTSY.
8. Finally, the resulting grid function of the gravitational potential defined on the subdomain Ω_n will be located in the memory of the corresponding processor.

The communication complexity of this algorithm is defined by MPI_Alltoall procedure. Each processor must send $L_y L_x/N$ real values and receive the same amount of data.

Test experiments were performed on MVS-100K at Joint Supercomputer Center (JSCC) and on the supercomputer at Siberian Supercomputer Center (SSCC) and on the MSU Lomonosov supercomputer. The results were similar and compatible for all the three supercomputers (thus, we present the measurement results only for SSCC). We used FFTW 3.1.4 library [11] for the Fast Fourier Transforms.

We split the total time of the FFTT algorithm to the two parts: T_{calc} (the time needed to perform direct and inverse FSTX and FSTY, serialize/deserialize data before and after transposing) and T_{comm} (the time needed to perform two calls of MPI_Alltoall).

The DDCS time was split to the three logical parts:

- T_{calc}: the time needed to solve the Poisson and the Laplace equations locally in a subdomain,
- T_{comm}: interprocessor communications — transfer data using the procedure MPI_SendRecv,
- T_{prop}: calculation of a single layer potential and boundary conditions for all the tree structure levels.

Tables 1 and 2 represent a weak scaling of the DDCS and the FFTT algorithms (where the number of grid nodes is proportional to the number of processors, i.e. amount of nodes per processor is kept the same).

Tables 3 and 4 represent a strong scaling for 16384×16384 and 65536×65536 problems (where the number of grid nodes is kept the same, while the number of processors is increasing).

Table 1. Weak scaling of DDCS and FFTT algorithms on supercomputer NKS-G6 SSCC: the computational domain and grid size are increasing in one direction proportional to the number of processors

N_p	$L_x \times L_y$	Solving time for $2048N_p \times 2048$ grid, s						
		DDCS				FFTT		
		T_{all}	T_{calc}	T_{comm}	T_{prop}	T_{all}	T_{calc}	T_{comm}
4	4096×2048	1.02	0.98	0.04	0.0004	0.55	0.49	0.06
8	8192×2048	0.98	0.97	0.01	0.0008	0.73	0.64	0.09
16	16384×2048	0.99	0.98	0.01	0.0011	0.80	0.65	0.15
32	32768×2048	0.98	0.97	0.01	0.0014	0.97	0.76	0.21
64	65536×2048	0.99	0.97	0.02	0.0017	1.17	0.91	0.25
128	131072×2048	1.07	0.98	0.09	0.0020	1.78	1.12	0.66
256	262144×2048	1.08	0.98	0.10	0.0023	2.25	1.27	0.97
512	524288×2048	1.07	0.97	0.09	0.0025	2.60	1.57	1.03

Table 2. Weak scaling of DDCS and FFTT algorithms on supercomputer NKS-G6 SSCC: the computational domain is increasing in two directions, but the total grid size is kept proportional to the number of processors

N_p	$L_x \times L_y$	Solving time for $2048\sqrt{N_p} \times 2048\sqrt{N_p}$ grid, s						
		DDCS				FFTT		
		T_{all}	T_{calc}	T_{comm}	T_{prop}	T_{all}	T_{calc}	T_{comm}
4	4096×4096	1.05	1.00	0.05	0.001	0.56	0.49	0.07
16	8192×8192	1.03	1.00	0.02	0.005	0.77	0.64	0.13
64	16384×16384	1.11	1.07	0.02	0.019	0.94	0.71	0.23
256	32768×32768	1.37	1.15	0.16	0.062	1.33	0.81	0.53

Table 3. Strong scaling of DDCS and FFTT algorithms on supercomputer NKS-G6 SSCC: the computational domain is kept the same (16384 × 16384) and the number of processors is increasing

N_p	# of nodes per proc.	Solving time for 16384 × 16384 grid, s						
		DDCS				FFTT		
		T_{all}	T_{calc}	T_{comm}	T_{prop}	T_{all}	T_{calc}	T_{comm}
16	16,777,216	5.74	5.66	0.07	0.01	3.50	2.97	0.53
32	8,388,608	2.32	2.20	0.10	0.02	1.83	1.39	0.43
64	4,194,304	1.11	1.07	0.02	0.02	0.94	0.71	0.23
128	2,097,152	0.62	0.53	0.07	0.02	0.56	0.35	0.21
256	1,048,576	0.34	0.26	0.05	0.03	0.32	0.18	0.14
512	524,288	0.20	0.13	0.04	0.03	0.37	0.09	0.28

Table 4. Strong scaling of DDCS and FFTT algorithms on supercomputer NKS-G6 SSCC: the computational domain is kept the same (65536 × 65536) and the number of processors is increasing

N_p	# of nodes per proc.	Solving time for 65536 × 65536 grid, s						
		DDCS				FFTT		
		T_{all}	T_{calc}	T_{comm}	T_{prop}	T_{all}	T_{calc}	T_{comm}
256	16,777,216	6.79	6.29	0.35	0.15	7.10	3.95	3.15
512	8,388,608	3.10	2.63	0.30	0.17	3.58	1.95	1.63

The analysis of the results brings about the following conclusions:

1. The time spent on communications for the DDCS algorithm (T_{comm}) is less (by several times) than the time spent on communications for the FFTT algorithm.

2. The total time for the DDCS algorithm (T_{all}) is not much better than the total time for FFTT. In some cases (like strong scaling on 16384 × 16384 grid) it is even worse unless it is launched on 512 processors. The explanation is the following: modern supercomputers (both their hardware and software) are good when providing fast communications for a moderate number of processors (less than 1024) as it used to be, so the calculation time for FFTT (T_{calc}) is comparable with the time communication (T_{comm}). However, the calculation time for the DDCS includes solving the Poisson equation two times (see a detailed description in [5]), so this time cannot be easily compensated.

3. The DDCS algorithm has an extremely good scalability in the case of increasing the computational domain in one direction (see Table 1). However, if the domain increases in two directions, the results does not look impressive (see Table 2). The reason is in the increasing amount of data (increasing L_y dimension) that should be transferred between subdomains in each SendRecv operation.

This means that the 2D version of the DDCS algorithm can be useful for a big number of processors for problems with elongated subdomains. While for the average-size problems it is still better to use the FFTT algorithm.

3. The optimized parallel algorithm for solving the 3D Poisson equation

Since our main interest is to solve the 3D Poisson equation, we have analyzed what optimizations could be applied to the algorithms described in [6]. We have measured the time headed for communications procedures for both DDCS and FFTT algorithms (i.e. MPI_AlltoAll and MPI_SendRecv operations) and compared their performance.

First of all, it turned out that in the case of using a big number of MPI processes (more than 256) independent pair-wise communications in the DDCS algorithm at the propagation stage in the bottom-up tree pass (see [6] for the tree structure description) have an unstable performance. If MPI processes participating in SendRecv communication are located at the same CPU node, then SendRecv procedure is fast. But if both MPI processes are far from each other, then communication time may be 10–50 times worse (e.g. 0.6 seconds against 0.02 seconds). In the ideal case, where each processor is connected with an other processor, it should not happen. But in the case of real supercomputers, the communications are performed in a more complicated way, so a lot of things depend on how hardware and software are organized. Thus, instead of independent SendRecv communications for each MPI process we reformulate the algorithm in such a way as to minimize the number of bottom-up SendRecv communications and instead we use top-down Broadcast communication. This gives a significant performance gain in reducing communication time (up to ten times).

The second interesting result of the analysis is how MPI_Alltoall communication procedure behaves when using a two-directional slab decomposition [9, 10]. We have found, that the performance of the two-directional decomposition (where 2 transpositions in 2 directions should be applied) does not work significantly better than a one-directional decomposition. The performance in most of the cases is even worse because of the fact that the first transposition is made inside a group of MPI processes that are located closely to each other (at the same nodes, or at connected nodes). But for the second transposition, the MPI group processes are very far from each other. So, the performance for the latter group may be 5–15 times worse.

These considerations lead to the following algorithm for solving the 3D Poisson equation that combines DDCS with FFTT. The computational domain Ω with $L_x \times L_y \times L_z$ grid dimensions is divided into $N = N_x \times N_z$ slab subdomains in X and Z directions. For each subdomain $\Omega_{n,m}$ we assign one processor $g_{n,m}$. Then we define larger subdomains $\Omega^n = \cup_m \Omega_{n,m}$ and

$\Omega_m = \cup_n \Omega_{n,m}$, and then create a group of processors $G^n = \sum_m g_{n,m}$ and $G^m = \sum_n g_{n,m}$ corresponding to the subdomains Ω^n and Ω_m.

Combined DDCS/FFTT Algorithm:

1. On each processor apply FSTY to the grid density function.
2. On each group G^n make a data transposition from Y to Z direction.
3. Apply the DDCS algorithm for each of the group G^m. We need to solve L_y independent 2D screened Poisson equations.
4. On each group G^n make a data transposition from Z to Y direction.
5. Apply the inverse FSTY.
6. Finally, each processor $g_{n,m}$ contains the grid functions of the gravitational potential for $\Omega_{n,m}$.

The test experiments with the grid 2048^3 and 1024 processors have shown that an appropriate choice of processor groups (like $N_x = 32$ and $N_y = 32$, where processors of the group G^n are located closer to each other in a hardware network) may significantly reduce the communication time (approximately by 10 times, from 7 seconds to 0.5 seconds) if compared with the FFTT-only version.

4. Conclusion

In this paper, we present the results of comparing the two algorithms for the Poisson equation solution. It turned out that for average-sized problems. the 2D version of FFTT algorithm is preferable, while the DDCS algorithm is useful for a big number of processors (thousands and even more) and elongated subdomains.

To solve the 3D Poisson equation we propose several optimizations and the new algorithm based on a combination of DDCS and FFTT.

References

[1] Vshivkov V.A., Snytnikov V.N., Snytnikov N.V. Simulation of three-dimensional dynamics of matter in gravitational field with the use of multiprocessor computer // Computational Technologies. — 2006. — Vol. 11, No. 2. — P. 15–27 (In Russian).

[2] Vshivkov V.A., Lazareva G.G., Snytnikov A.V., et al. Hydrodynamical code for numerical simulation of the gas components of colliding galaxies // Astrophysical J. Supplement Series. — 2011. — Vol. 194. — P. 1–12.

[3] Springel V., Yoshida N., White S.D.M. GADGET: a code for collisionless and gasdynamical cosmological simulation // New Astronomy. — 2001. — Vol. 6. — P. 79–117.

[4] Snytnikov V.N., Vshivkov V.A., Kuksheva E.A., et al. Three-dimensional numerical simulation of a nonstationary gravitating N-body system with gas // Astronomy Letters. — 2004. — Vol. 30. — P. 124–137.

[5] Snytnikov N.V. A parallel algorithm for solving 2D Poisson's equation in the context of nonstationary problems // Vychislitel'nye Metody i Programmirovanie. — 2015. — Vol. 16. — P. 39–51 (In Russian).

[6] Snytnikov N.V. Domain decomposition based on a direct method for solving the three-dimensional Poisson's equation in nonstationary astrophysical problems // Vychislitel'nye Metody i Programmirovanie. — 2015. — Vol. 16. — P. 94–98 (In Russian).

[7] Huang J., Greengard L. A fast direct solver for elliptic partial differential equations on adaptively refined meshes // SIAM J. Sci. Comput. — 2000. — Vol. 21. — P. 1551–1566.

[8] Terekhov A.V. Parallel dichotomy algorithm for solving tridiagonal system of linear equations with multiple right-hand sides // Parallel Computing. — 2010. — Vol. 36. — P. 423–438.

[9] Ayala O., Wang L.P. Parallel implementation and scalability analysis of 3D Fast Fourier Transform using 2D domain decomposition // Parallel Computing. — 2013. — Vol. 39. — P. 58–77.

[10] Duya T.V.T., Ozaki T. A decomposition method with minimum communication amount for parallelization of multi-dimensional FFTs // Computer Physics Communications. — 2014. — Vol. 185. — P. 153–164.

[11] Frigo M., Johnson S.G. FFTW Software. — http://www.fftw.org.