Fast parallel Grad–Shafranov solver for real-time equilibrium reconstruction in EAST tokamak using graphic processing unit*

Yao Huang(黄耀)[1], Bing-Jia Xiao(肖炳甲)[1,2], and Zheng-Ping Luo(罗正平)[1,†]

[1] *Institute of Plasma Physics, Chinese Academy of Sciences, Hefei* 230031, *China*
[2] *School of Nuclear Science & Technology, University of Science & Technology of China, Hefei* 230027, *China*

(Received 22 February 2017; revised manuscript received 1 April 2017; published online 20 June 2017)

To achieve real-time control of tokamak plasmas, the equilibrium reconstruction has to be completed sufficiently quickly. For the case of an EAST tokamak experiment, real-time equilibrium reconstruction is generally required to provide results within 1ms. A graphic processing unit (GPU) parallel Grad–Shafranov (G-S) solver is developed in P-EFIT code, which is built with the CUDA™ architecture to take advantage of massively parallel GPU cores and significantly accelerate the computation. Optimization and implementation of numerical algorithms for a block tri-diagonal linear system are presented. The solver can complete a calculation within 16 μs with 65×65 grid size and 27 μs with 129×129 grid size, and this solver supports that P-EFIT can fulfill the time feasibility for real-time plasma control with both grid sizes.

Keywords: tokamak, Grad–Shafranov equation, equilibrium reconstruction, GPU parallel computation

PACS: 52.55.Fa, 52.55.–s, 52.65.Kj **DOI:** 10.1088/1674-1056/26/8/085204

1. Introduction

Plasma equilibrium reconstruction is an important tool for tokamak data analysis. Based on experimental measurements, the plasma shape and current profile can be evaluated. The real-time plasma equilibrium reconstruction is routinely used for the plasma control in takamak operation.[1,2] The essential requirement for optimum performance in plasma feedback control is to obtain the accurate and fast equilibrium reconstruction results. For the toroidal symmetry of a tokamak, ideal magnetohydrodynamic (MHD) equilibrium is described by G-S equation.

$$\Delta^*\psi = -\mu_0 R J_\phi, \quad J_\phi = RP'(\psi) + \frac{\mu_0 FF'(\psi)}{4\pi^2 R}. \tag{1}$$

Here, ψ is the poloidal magnetic flux per radian of the toroidal angle ϕ enclosed by a magnetic surface, $F = 2\pi RB_\phi/\mu_0$ is the poloidal current function, B_ϕ is the toroidal magnetic field, and $\Delta^* = R^2\nabla\cdot(\nabla/R^2)$. Various equilibrium reconstruction codes which solve G-S equation numerically have been developed and implemented on different tokamaks. One of the most widely used codes is EFIT,[3–5] which reconstructs equilibrium by solving the G-S equation while approximately conserving experimental measurements. In EFIT, J_ϕ is represented as a linear combination of a number N_P and N_F of basis functions.

$$P'(\psi) = \sum_{n=1}^{N_P} \alpha_n y_n(\psi), \quad FF'(\psi) = \sum_{n=1}^{N_F} \beta_n y_n(\psi). \tag{2}$$

The value of the linear parameters α_n and β_n are fitted by using available measurements and constraints. After obtaining the plasma current J_ϕ, the poloidal flux ψ can then be calculated by solving Eq. (1) with the finite-difference method.

The numeric solution of Eq. (1) is one of the most computationally expensive parts of the equilibrium reconstruction algorithm. The efficient numerical solver for Eq. (1) based on CPU parallel computation and the codes called GPEC[6] and LIUQE[7] have been developed and implemented in ASDEX-U.[8] Additionally in Ref. [9], a GPU parallel version EFIT code for magnetic reconstructions, P-EFIT has been developed which significantly accelerates the whole computation process. The P-EFIT efficiently takes advantage of massively parallel GPU cores and significantly accelerates the EFIT reconstruction algorithms. This has already been successfully implemented for plasma control in EAST.[10]

In this paper, we present a numerically optimized solver for Eq. (1) based on GPU parallel computation and CUDA architecture,[11] which is one of the most important parts of P-EFIT. Through carefully designing parallel algorithms, the G-S solver can complete the job within 16 μs with 65×65 grid size and 27 μs with 129×129 grid size. P-EFIT can complete one typical equilibrium reconstruction iteration in about 155 μs with 65×65 grid size and 274 μs with 129×129 grid size, these results show good capability for real-time plasma configuration feedback control and potential plasma current profile control with high spatial resolution. The detailed algorithms of GPU parallel G-S solver are described in Section 2. Actual numerical implementation, computational performance and discussion on the relevance of the solver for real-time plasma equilibrium calculations are provided in Section 3. A

*Project supported by the National Magnetic Confinement Fusion Research Program of China (Grant No. 2014GB103000), the National Natural Science Foundation of China (Grant No. 11575245), and the National Natural Science Foundation of China for Youth (Grant No. 11205191).

†Corresponding author. E-mail: zhpluo@ipp.ac.cn

© 2017 Chinese Physical Society and IOP Publishing Ltd

summary and the outlook on further development of this work are discussed in Section 4.

2. Numerical algorithm details

Equation (1) is, in mathematical terms, an elliptic differential equation or Poisson equation, when plasma current J_ϕ is available. Based on the algorithms described by Buneman[12] and Buzbee,[13] Hagenow,[14] a fast Poisson solver is developed for solving Eq. (1) with Dirichlet boundary conditions. To create a Dirichlet boundary condition, flux values at the boundary of the meshed area must be calculated through a pre-computed Green function matrix multiplied by the plasma current vector. Then the finite difference method is used and Eq. (1) is transformed into a block tri-diagonal equation (Eq. (3)) which can be solved by Buneman's solver or other solvers.

$$\begin{aligned}
&\begin{bmatrix} A & -d_1 I & & & & \\ -b_2 I & A & -d_2 I & & & \\ & -b_3 I & A & -d_3 I & & \\ & & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & -d_{M-1} I \\ & & & & -b_M I & A \end{bmatrix} \begin{bmatrix} \Psi_1 \\ \Psi_2 \\ \Psi_3 \\ \vdots \\ \Psi_N \end{bmatrix} \\
&= \begin{bmatrix} S_1 + b_1 \Psi_0 \\ S_2 \\ S_3 \\ \vdots \\ S_M + d_M \Psi_{M+1} \end{bmatrix}, \\
c &= \left(\frac{\Delta R}{\Delta Z}\right)^2, \quad 1 + \frac{\Delta R}{2R_i} = b_i, \quad 1 - \frac{\Delta R}{2R_i} = d_i, \\
A &= \begin{bmatrix} 2(1+c) & c & & \\ -c & 2(1+c) & & \\ & & \ddots & \\ & & -c & 2(1+c) \end{bmatrix}, \\
S_i &= \begin{bmatrix} (\Delta R)^2 \mu_0 j_{i,1} + c\psi_{i,0} \\ (\Delta R)^2 \mu_0 j_{i,2} \\ \vdots \\ (\Delta R)^2 \mu_0 j_{i,N} + c\psi_{i,N+1} \end{bmatrix}, \quad \Psi_i = \begin{bmatrix} \psi_{i,1} \\ \psi_{i,2} \\ \vdots \\ \psi_{i,N} \end{bmatrix}, \qquad (3)
\end{aligned}$$

where ΔR and ΔZ are the widths of grid in the r and z directions, R_i is the coordinate of each grid in the r direction, $j_{m,n}$ and $\psi_{m,n}$ are the value of plasma current density and magnetic flux on each grid, respectively.

The main idea of our approach is to exploit massively parallel GPU cores when solving Poisson's differential equation. The algorithms in Refs. [6] and [15] have adopted the way to combine the fast Fourier transform (FFT) which essentially decouples the original block tri-diagonal system in one of the two grid dimensions and the solution of the resulting set of tri-diagonal systems of equations, so that each independent system can be solved in parallel. However, because of the GPU's hundreds of cores and high bandwidth memory, matrix multiplication can be calculated in parallel efficiently.[8,9] When the rectangle grid is settled, the matrix A is also settled. The eigenvalue decomposition Eq. (4), rather than the FFT, can be directly used to decouple the original problem.

$$\begin{aligned}
A &= Q\Lambda Q^{\mathrm{T}}\Lambda = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_M \end{bmatrix}, \\
\lambda_i &= c\left(2 - 2\cos\frac{\mathrm{i}\pi}{m+1}\right) + 2, \\
Q &= \sqrt{\frac{2}{m+1}} \begin{bmatrix} \sin\frac{\pi}{m+1} & \sin\frac{2\pi}{m+1} & \cdots & \sin\frac{m\pi}{m+1} \\ \sin\frac{2\pi}{m+1} & \sin\frac{4\pi}{m+1} & \cdots & \sin\frac{2m\pi}{m+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sin\frac{m\pi}{m+1} & \sin\frac{2m\pi}{m+1} & \cdots & \sin\frac{m^2\pi}{m+1} \end{bmatrix}. \qquad (4)
\end{aligned}$$

After multiplying both sides of Eq. (3) by the Q^{T} matrix as shown in Eq. (5) and making a transposition as shown in Eq. (6), the initial block tri-diagonal system is transformed into a block diagonal system which has M tri-diagonal systems with rank $M \times M$ as shown in Eq. (7). Each tri-diagonal system can be solved in parallel on GPU.

$$\begin{aligned}
&-b_i\Psi_{i-1} + A\Psi_i - d_i\Psi_{i+1} = S_i, \\
&-b_i Q^{\mathrm{T}}\Psi_{i-1} + \Lambda Q^{\mathrm{T}}\Psi_i - d_i Q^{\mathrm{T}}\Psi_{i+1} = Q^{\mathrm{T}}S_i, \qquad (5) \\
&Q^{\mathrm{T}}\Psi_i = \hat{\Psi}_i, \quad Q^{\mathrm{T}}S_i = \hat{g}_i, \\
&\tilde{\Psi}_j = (\hat{\Psi}_{1,j}, \hat{\Psi}_{2,j}, \hat{\Psi}_{3,j}, \ldots, \hat{\Psi}_{M,j})^{\mathrm{T}}, \\
&\tilde{g}_j = (\hat{g}_{1,j}, \hat{g}_{2,j}, \hat{g}_{3,j}, \ldots, \hat{g}_{M,j})^{\mathrm{T}}, \qquad (6) \\
&\begin{bmatrix} T_1 & & & \\ & T_2 & & \\ & & \ddots & \\ & & & T_N \end{bmatrix} \begin{bmatrix} \tilde{\Psi}_1 \\ \tilde{\Psi}_2 \\ \vdots \\ \tilde{\Psi}_N \end{bmatrix} = \begin{bmatrix} \tilde{g}_1 \\ \tilde{g}_2 \\ \vdots \\ \tilde{g}_N \end{bmatrix}, \\
&T_j = \begin{bmatrix} \lambda_j & -d_1 & & \\ -b_2 & \lambda_j & \ddots & \\ & \ddots & \ddots & -d_{M-1} \\ & & -b_M & \lambda_j \end{bmatrix}. \qquad (7)
\end{aligned}$$

After $\tilde{\Psi}_i$ is solved, another transposition to obtain $\hat{\Psi}_i$ followed by multiplying $\hat{\Psi}_i$ by Q is performed to obtain the final result Ψ_i as shown in the following Eq. (8):

$$\begin{aligned}
&\hat{\Psi}_j = (\tilde{\Psi}_{j,1}, \tilde{\Psi}_{j,2}, \tilde{\Psi}_{j,3}, \ldots, \tilde{\Psi}_{j,M})^{\mathrm{T}}, \\
&\Psi_i = Q\hat{\Psi}_i. \qquad (8)
\end{aligned}$$

Table 1 shows the flow chart of GPU parallel G-S solver's numeric algorithm.

Table 1. Structure of numeric algorithm of GPU parallel G-S solver.

Step 1	Eigenvalue decomposition (multiplication of Q matrix);
Step 2	Matrix transposition of matrix $\hat{g}_i$;
Step 3	Solve block tri-diagonal linear system (matrix T_j inverse);
Step 4	Matrix transposition of matrix $\tilde{\Psi}_j$;
Step 5	Inverse eigenvalue decomposition (multiplication of Q^{T} matrix).

3. Implementation and computational performance

3.1. Implementation with CUDA architecture

In CUDA architecture, a multithreaded program is partitioned into blocks of threads that are executed independently of each other. Each block shared memory, which is visible to all threads of the block, is much faster than the global memory that all blocks and threads can access. There are standard linear algebra routines in the CUDA library, but the size of the matrix in equilibrium reconstruction algorithms is too small to achieve good performance by using them directly. When developing G-S solver in P-EFIT with CUDA architecture, we need to design customized parallel algorithms, arrange GPU cores into blocks and threads, consider how to take full advantage of hundreds of GPU cores and high speed memory (e.g., shared memory), and minimize the communication and synchronization between different cores. Each parallel algorithm needs to be carefully designed through combining the needs of the various numerical algorithms and the GPU capacity.

3.1.1. Eigenvalue and inverse eigenvalue decomposition

The eigenvalue decomposition and inverse eigenvalue decomposition (Steps 1 and 5 in Table 1) consist mainly of M matrix-vector multiplications with rank $M \times M$. The M matrix-vector multiplications ($Q^{\mathrm{T}}S_i$ and $Q\hat{\psi}_i$) can be transformed into a matrix-matrix multiplication ($Q^{\mathrm{T}}\overline{\overline{S_i}}$ and $Q\overline{\overline{\hat{\psi}}}_i$) with rank $M \times M$. For matrix–matrix multiplication, the simplest parallel strategy has one thread computing one element in the resulting matrix by using $M \times M$ GPU cores. However, this strategy cannot achieve good performance because each element in Q^{T}, $\overline{\overline{S_i}}$, Q, and $\overline{\overline{\hat{\psi}}}_i$ needs to be read from the GPU global memory M times. The time spent to access these data is the main bottleneck of parallel computation.

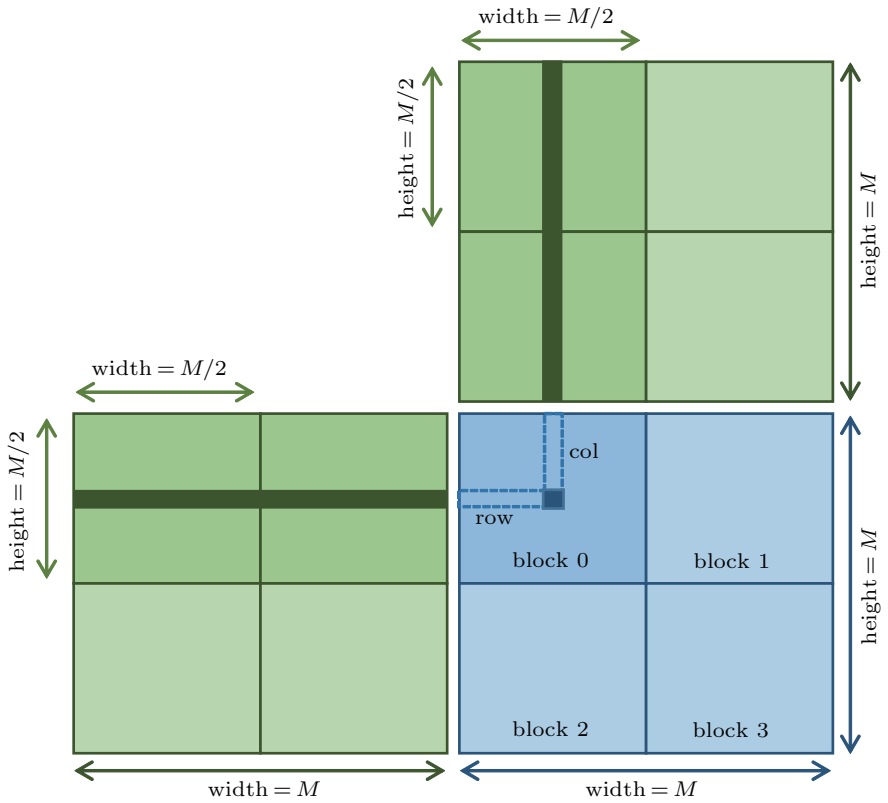


Fig. 1. (color online) Matrix division and block distribution in multiplication of two green matrices with rank $M \times M$.

By dividing the matrices (Q^{T}, $\overline{\overline{S_i}}$, Q, $\overline{\overline{\hat{\psi}}}_i$) into small pieces and distributing GPU threads into blocks based on the sizes of small pieces, each block computes a small piece of the matrix-matrix multiplications. For example, if the original matrix with rank $M \times M$ is divided into 4 small pieces of the same size, four blocks and $M \times M/4$ threads in each block are distributed as shown in Fig. 1. Each thread in each block calculates an element in the small piece of the matrix-matrix multiplications. Each element in each of Q^{T}, $\overline{\overline{S_i}}$, Q, and $\overline{\overline{\hat{\psi}}}_i$ only needs to be read from the GPU global memory four times, then the small pieces of matrix are stored in the shared memory of

each block. By this strategy, the time spent to access the data is reduced significantly and the demand for GPU cores remains the same. Finally, the eigenvalue and inverse eigenvalue decomposition are calculated efficiently on GPU.

3.1.2. Tri-diagonal solver

Solving independent block tri-diagonal linear system can be parallelized with coarse and fine grain strategy. M blocks are distributed to solve M independent tri-diagonal linear systems T_j's, which is coarse-grained parallelization. However, the most challenging part is to solve the tri-diagonal linear system in parallel. The Gaussian elimination method which is used to solve tri-diagonal linear systems is a typical serial algorithm. To make the equation easier to express, each independent tri-diagonal linear system T_j can be rewritten as shown in Eq. (9). During the elimination, every accumulation step has to use the results from the prior step as shown in Eq. (10).

$$\begin{bmatrix} b_1 & c_1 & & \\ a_1 & b_2 & \ddots & \\ & \ddots & \ddots & c_{n-1} \\ & & a_{n-1} & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix}, \tag{9}$$

$$\begin{aligned}
&b'_i = b_i - k_{i-1}c_{i-1},\\
&d'_i = d_i - k_{i-1}d'_{i-1},\\
&k_{i-1} = (a_{i-1}/b'_{i-1}),\\
&x_n = d'_n/b'_n,\\
&x_i = d'_i/b'_i - m_i x_{i+1},\\
&m_i = c_i/b'_i,
\end{aligned} \tag{10}$$

$$\begin{aligned}
&d'_1 = d_1,\\
&d'_2 = d_2 - k_1 d'_1,\\
&d'_3 = d_3 - k_2 d'_2 = d_3 + (-k_2)d_2 + (-k_2)(-k_1)d_1,\\
&\vdots\\
&d'_n = d_n + (-k_{n-1})d_{n-1} + (-k_{n-1})(-k_{n-2})d_{n-2} + \cdots\\
&\qquad + \prod_{i=1}^{n-1}(-k_i)d_1,
\end{aligned} \tag{11}$$

$$\begin{aligned}
&x_n = d'_n/b'_n,\\
&x_{n-1} = d'_{n-1}/b'_{n-1} - m_{n-1}d'_n/b'_n,\\
&x_{n-2} = d'_{n-2}/b'_{n-2}\\
&\qquad + (-m_{n-2})d'_{n-1}/b'_{n-1} + (-m_{n-2})(-m_{n-1})d'_n/b'_n,\\
&\vdots\\
&x_1 = d'_1/b'_1 + (-m_1)d'_2/b'_2 + (-m_1)(-m_2)d'_3/b'_3 + \cdots\\
&\qquad + \prod_{i=1}^{n-1}(-m_i)d'_n/b'_n.
\end{aligned} \tag{12}$$

When the rectangle grid is settled, the coefficients, such as k_{i-1}, m_i, and b' in Eq. (10), can be calculated before hand, only coefficients d'_i and results x_i need to be solved. As shown in Eqs. (11) and (12), the value of $d'_{i-1}(x_{i+1})$ is needed when calculating $d'_i(x_i)$. If distributing M threads in each block to directly accumulate the coefficients d'_i and the results x_i as shown in Eqs. (11) and (12), M iterations are needed in the M thread, the M thread also completes the calculation in the $M-1$ thread, thus this parallel strategy cannot achieve good computational performance. The parallel data method called 'prefix sum' in Ref. [16] is adopted in calculating the coefficients d'_i and the results x_i. As a pseudo-code shown in Fig. 2, the original M iterations in the sequence of accumulations can be done in $\log_2 M$ iterations.

for j=1 **to** $\log_2 M$ **do**

  for all M **threads in parallel do**

    if $m \geq 2^j$ **then**

$$d[m] = k[m-1] \times d[m-2^{j-1}] + d[m]$$
$$k[m-1] = k[m-1] \times k[m-1-2^{j-1}]$$

    end if

  end do

end do

Fig. 2. Pseudo-code of GPU parallel tri-diagonal solver algorithms, the original M iterations in the sequence of accumulations can be done in $\log_2 M$ iterations.

With coarse-grained parallelization for solving each T_j tri-diagonal linear system independently and fine-grained parallelization for solving every T_j tri-diagonal linear system with parallelized Gaussian elimination method, the calculation of Step 3 in Table 1 can be parallelized efficiently.

3.1.3. Matrix transposition

The matrix regrouping in Eq. (6) is actually matrix transposition. By using the massive GPU cores to access GPU global memory and using the coalescence access technique[9] to make full use of the high bandwidth of GPU global memory, this data manipulation can be done in a very short time.

3.2. Benchmark results and computational performance

Our main hardware platform for performing the benchmarks is a Linux workstation with one Intel(R) Xeon(R) CPU E5-2667 v3 3.20 GHz and one NVIDIA Pascal TITAN X GPU card with using single float precision. Unless otherwise stated, all the timing results in the following sections are assumed to come from the tests on this hardware platform. The numerical grids with 65×65 and 129×129 are the main cases under consideration. The corresponding 65×65 spatial resolution is generally regarded as being sufficiently accurate for plasma shape control, whereas the finer 129×129 spatial resolution is sufficient when extended to plasma current profile control. Our primary goal is that both spatial resolutions can satisfy the time feasibility requirements in real-time reconstruction for plasma discharge control.

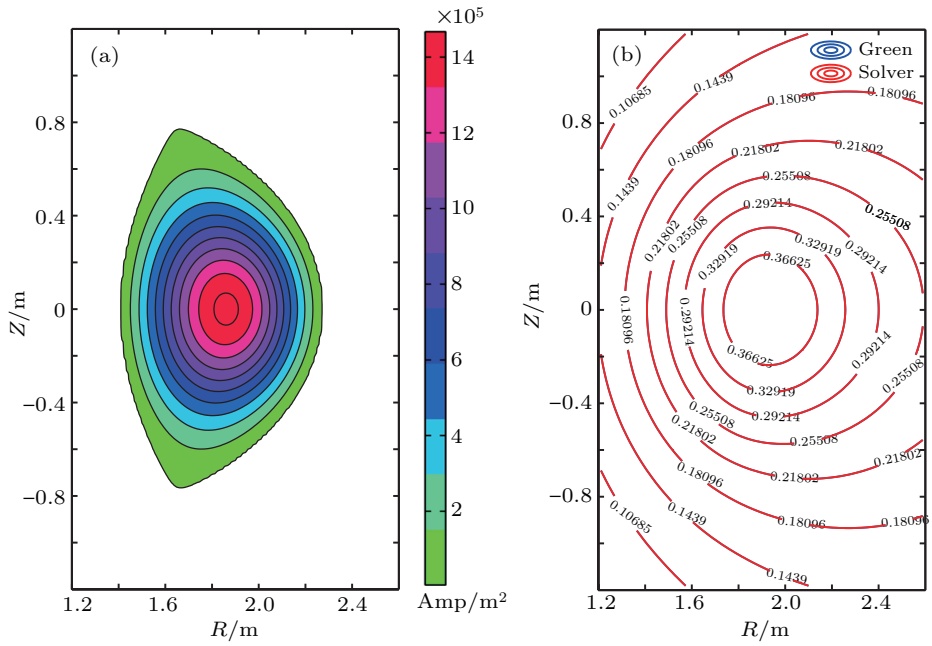


Fig. 3. (color online) (a) Plasma current density distribution of typical EAST double-null configuration and (b) comparison between magnetic flux surface from Green's function (blue lines) and that from GPU parallel G-S solver (red lines) contributed by plasma current shown in Fig. 3(a), and their difference is so small that blue and red lines coincide with each other.

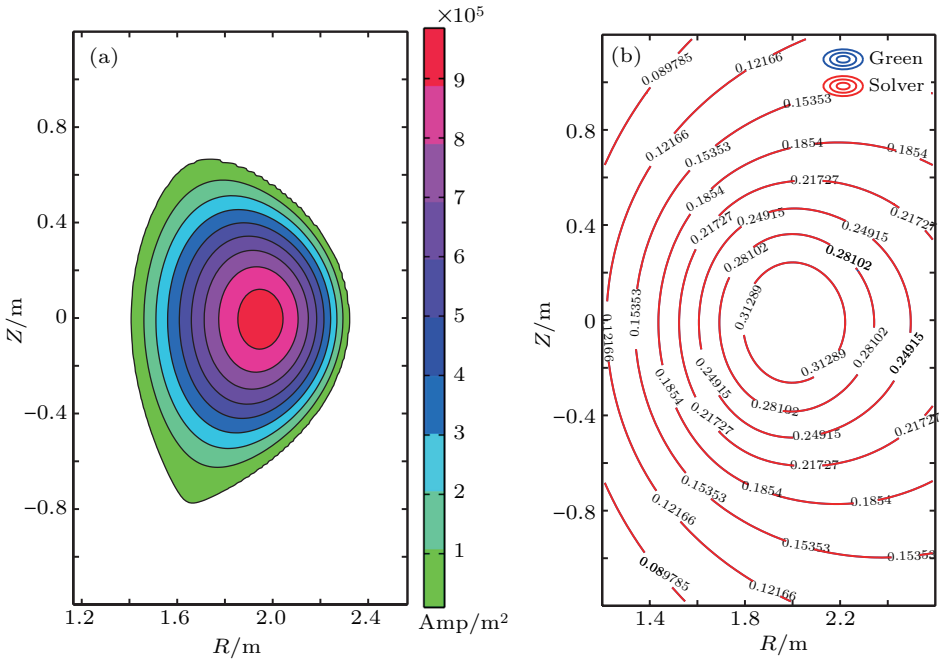


Fig. 4. (color online) (a) Plasma current density distribution of typical EAST low-single-null configuration and (b) comparison between magnetic flux surface from Green's function (blue lines) and that from GPU parallel G-S solver (red lines) contributed by plasma current shown in Fig. 4(a), and their difference is so small that blue and red lines coincide with each other.

The correctness of the GPU parallel G-S solver in P-EFIT can be indirectly proved by P-EFIT equilibrium reconstruction results in Ref. [9] and P-EFIT implementation in plasma control in Ref. [10]. Here, GPU parallel Grad–Shafranov solver results are shown in Figs. 3–5. Figures 3(a)–5(a) show the plasma current density distributions of typical EAST double-null, low-single-null and upper-single-null configurations with a 65×65 grid as the input. Figures 3(b)–5(b) show the comparisons between the magnetic flux distribution results from the GPU parallel G-S solver and the results from directly multiplying the plasma current by Green's function.[17] The results from the two accord with each other well, with the maximum relative error being less than 10^{-8}. During the equilibrium reconstruction iterations, it is generally required that magnetic flux distribution from the last two successive iterations should have small relative difference, i.e., less than 10^{-4}, to converge. Thus, our approach is accurate enough. It is necessary to mention that the magnetic flux distribution result in Figs. 3(b)–5(b) should be added to the magnetic flux contributed by the external poloidal field coils to obtain the final magnetic flux distribution result.

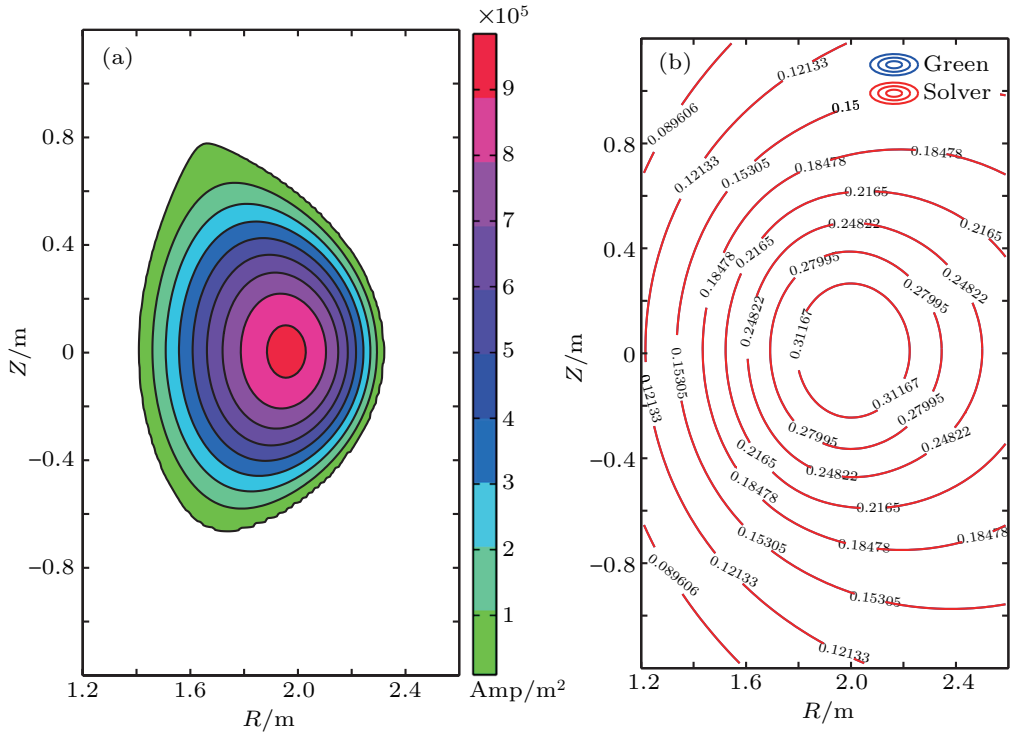


Fig. 5. (color online) (a) Plasma current density distribution of typical EAST upper-single-null configuration and (b) comparison between magnetic flux surface from Green's function (blue lines) and that from GPU parallel G-S solver (red lines) contributed by plasma current shown in Fig. 5(a), and their difference is so small that blue and red lines coincide with each other.

More benchmarks are conducted to test the timing results of the GPU parallel G-S solver as shown in Table 2. The solver can give out the result in 0.016 ms with a 65×65 grid and 0.027 ms with 129×129 grid. The time taken by each step increases when the grid is finer. Considering the grid size and the consuming time, the solver has its peak computational performance with the 129×129 grid size. When the grid size is 257×257, the eigenvalue decomposition and tri-diagonal solver need 257×257 GPU cores and a large number of shared memories in blocks. Therefore, the computational needs of the parallel algorithms go well beyond TITAN X GPU capacity. At the same time, when transposing the matrix with size $M \times M$, the data transferring in GPU global memory increases 16 times when the grid size increases from 65×65 to 257×257, the computational cost also increases about 16 times. That is why the G-S solver spends much more time with the 257×257 grid size.

Table 2. Timing results of each part in GPU parallel G-S solver for different grid sizes, twice eigenvalue decomposition and consuming time of matrix transposition steps.

	Gird					
	65×65		129×129		257×257	
	T/μs	%	T/μs	%	T/μs	%
Eigenvalue decomposition	8.1	50.6	12.3	45.5	42.4	39.6
Matrix transposition	1.7	10.6	6.5	24.1	26.7	24.9
Tri-diagonal solver	6.2	38.8	8.2	30.4	37.9	35.5
Solver	16	100	27	100	107	100

3.3. Relevance for equilibrium reconstruction

Although the GPU parallel G-S solver would not bring better reconstruction quality, the main advantage of this solver is the acceleration of the computation with high spatial resolution to satisfy the requirement for real-time plasma control. High spatial resolution such as the 129×129 grid is needed to perform a detailed equilibrium calculation with more diagnostics such as internal plasma current profile diagnostics and kinetic diagnostics.[18] The machine cycle time of EAST's plasma control system is typically 0.1 ms,[19,20] but the response time of EAST's poloidal field power supply is about 1 ms.[21] It is generally demanded that plasma equilibrium reconstruction should provide results for plasma shape control per 1ms or less. As one of the important and time consuming parts in the whole equilibrium reconstruction algorithm, the G-S solver should complete the job in much less time.

The GPU optimized equilibrium reconstruction code P-EFIT[9,10] is based on the EFIT framework but takes advantage of massively parallel GPU cores to significantly accelerate the computation. In a typical equilibrium reconstruction calculation in the case of EAST, there is a total of $N_M = 74$ magnetic diagnostic signals and the corresponding response matrix $\overline{\overline{R}}$ has dimensions $N_M \times (N_P + N_F + N_{ext})$, where $N_P + N_F$ is the total number of basis functions employed and N_{ext} is the number of external coils, which are fitted to magnetic data. The G-S solver refreshes poloidal flux on grid (Δ^* inversion) after obtaining the plasma current distribution. In addition to the Grad–Shafranov solver, the P-EFIT includes and implements all the algorithms in equilibrium reconstruction by GPU parallel computation, such as the calculation of the response matrix, least square fitting, plasma boundary searching, and global parameter (β_p, li) calculation. It should be noted that after a converged equilibrium solution ψ in Eq. (1) is obtained, additional post-processing computations are required to finally compute global parameters and deliver control errors for the plasma control system; the optimization of these parts is already included in P-EFIT such as the detection of contours in the plasma boundary and flux surface as well as the integrals computation to obtain β_p, li, and the safety factor profile. These detailed parallel algorithms can be found in Ref. [8] and the flow chart of P-EFIT is shown in Fig. 6.

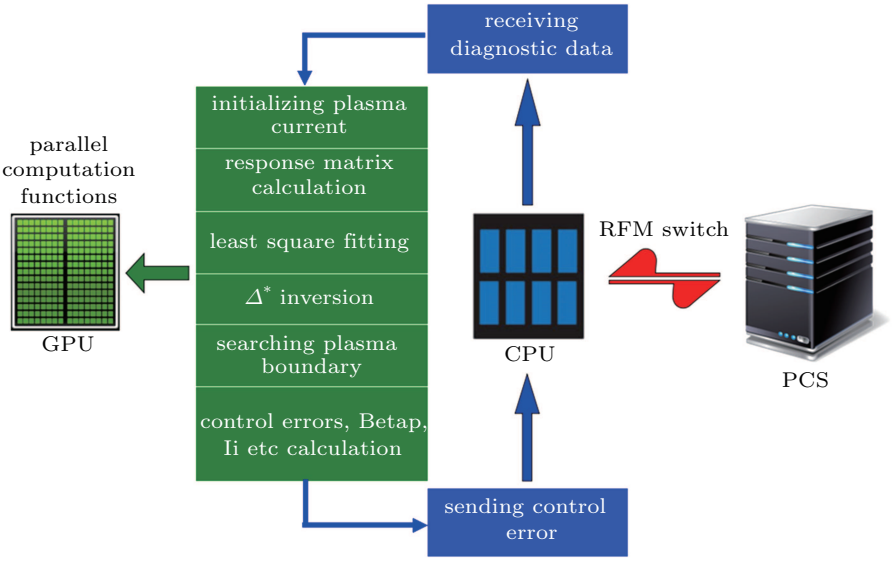


Fig. 6. (color online) Flow chart of P-EFIT: Green parts are for equilibrium calculations which are processed on GPU, and blue parts are for the data interface which is processed on CPU. PCS stands for plasma control system. RFM switch is a high speed reflective memory network transferring data between GPU server and PCS server.

To achieve the computational performance, each part of the equilibrium reconstruction algorithm needs to be carefully designed. In Table 3 summarized is the time needed to complete each part in the P-EFIT code. Finally, the P-EFIT can complete the whole iteration in about 155 μs with 65×65 grid size and 274 μs with 129×129 grid size. For the 129 ×129 grid size, the P-EFIT can also fulfill the requirement for real-time control, and finer spatial resolution could provide more accurate results when extending to plasma current profile reconstruction with more diagnostics. For now, all the parts of the P-EFIT code are using single floating precision, because GPU single floating precision computational capacity is much better than GPU double floating precision computational capacity. Although for numerical computation, double floating precision is usually required to guarantee precision in a complicated calculation. In real-time equilibrium reconstruction, single floating precision has been proved to be accurate enough.[7,9,10]

Another benchmark is performed to compare the GPU parallel G-S solver with P-EFIT computational performance on different hardware platforms. Here, the GPU parallel G-S solver and P-EFIT process the same task but in different hardware environments with using single floating precision. As shown in Table 4, with the more powerful GPU, the computational performance gain from more powerful hardware, the consuming time of G-S solver and one typical equilibrium reconstruction iteration decreases.

Table 3. Consuming time of each part in P-EFIT per typical equilibrium reconstruction iteration.

	Grid			
	65×65		129×129	
	T/μs	%	T/μs	%
Grad–Shafranov solver	16	16.7	27	9.9
Response matrix calculation	25	14.9	57	20.8
Least square fitting	31	18.5	32	11.7
Searching plasma boundary	22	13.0	35	12.8
Global parameters calculation	29	17.9	54	19.7
Others	32	19.0	69	25.2
Total	155	100	274	100

Table 4. GPU parallel G-S solver and P-EFIT computational performances on different hardware platforms.

Hardware platform	Grid			
	65×65		129×129	
CPU + GPU	G-S solver/μs	Iteration/μs	G-S solver/μs	Iteration/μs
Xeon E5-2670 v2 + Tesla K20	37	312	91	643
Xeon E5-2690 v2 + Tesla K40	32	282	73	558
Xeon E5-2667 v3 + Pascal TITAN X	16	155	27	274

The P-EFIT takes advantage of massively parallel GPU cores to significantly accelerate the EFIT reconstruction algorithms and has been successfully implemented for plasma control in EAST by using external magnetic data with a 65×65 grid size.[9,22] Besides the RT-EFIT,[2] the P-EFIT provides another routine real-time plasma equilibrium reconstruction method, which has better spatial resolution and customized modules for plasma control in EAST. The P-EFIT supports a snowflake configuration, the MIMO plasma shape control experiments[23] in EAST, and has some preliminary results in DIII-D.[24]

4. Conclusions

In this paper, we present a GPU parallel optimized G-S solver in P-EFIT code. Based on CUDA architecture, parallel numerical algorithms are designed to achieve good computational performance. Finally, the GPU parallel G-S solver can complete the job within 16 μs with 65×65 grid size and 27 μs with 129×129 grid size, which means this solver allows P-EFIT to fulfill the time feasibility requirements for real-time plasma control with both grid sizes. Further work will focus on developing more flexible parallel algorithms so that P-EFIT could extend to including more diagnostics easily. On the other hand, the latest released GPU hardware has better computational capacity and different hardware architectures[25] which could mean that existing algorithms may be adjusted to achieve better computational performance.

References

[1] Blum J, Boulbe C and Faugeras B 2009 *J. Comput. Phys.* **231** 960

[2] Ferron J R, Walker M L, Lao L L, St. John H E, Humphreys D A and Leuer J A 2002 *Nucl. Fusion* **38** 1055

[3] Lao L L, John H S, Stambaugh R D, Kellman A G and Pfeiffer W 2011 *Nucl. Fusion* **25** 1611

[4] Lao L L, Ferron J R, Groebner R J, Howl W, John H S, Strait E J and Taylor T S 1990 *Nucl. Fusion* **30** 1035

[5] Lao L L, John H E S, Peng Q, Ferron J R, Strait E J, Taylor T S, Meyer W H, Zhang C and You K I 2005 *Fusion Sci. Technol.* **48** 968

[6] Rampp M, Preuss R, Fischer R, Hallatschek K and Giannone L 2012 *Fusion Sci. Technol.* **62** 409

[7] Moret J M, Duval B P, Le H B, Coda S, Felici F and Reimerdes H 2015 *Fusion Eng. Design* **91** 1

[8] Giannone L, Fischer R, Mccarthy P J, Odstrcil T, Zammuto I, Bock A, Conway G, Fuchs J C, Gude A, Igochine V, Kallenbach A, Lackner K, Maraschek M, Rapson C, Ruan Q, Schuhbeck K H, Suttrop W, Wenzel L and ASDEX Upgrade Teama 2015 *Fusion Eng. Design* **100** 519

[9] Yue X N, Xiao B J, Luo Z P and Guo Y 2013 *Plasma Phys. Control. Fusion* **55** 085016

[10] Huang Y, Xiao B J, Luo Z P, Yuan Q P, Pei X F and Yue X N 2016 *Fusion Eng. Design* **112** 1019

[11] NVIDIA 2016 CUDA C Programming guide v. 8.0

[12] Buneman O 1969 A compact non-iterative poisson solver *Computers*

[13] Buzbee B L, Dorr F W, George J A and Golub G H 1970 *Siam J. Num. Anal.* **8** 722

[14] Hagenow K V and Lackner K 1975 *Proc. 7th Conf. Numerical Simulation of Plasmas*, June 2–4, New York, USA, p. 140

[15] Yue X N, Xiao B J, Luo Z P 2013 *Comput. Sci.* **40** 21 (in Chinese)

[16] Hillis W D and Jr G L S 1986 *Communications of the Acm* **29** 1170

[17] Schill R A 2003 *IEEE Trans. Magn.* **39** 961

[18] Ren Q, Chu M S, Lao L L and Srinivasan R 2011 *Plasma Phys. Control. Fusion* **53** 095009

[19] Xiao B J, Humphreys D A, Walker M L, Hyatt A, Leuer J A, Mueller D, Penaflor B G, Pigrowski D A, Johnson R D, Welander A, Yuan Q P, Wang H Z, Luo J R, Luo Z P, Liu C Y, Liu L Z and Zhang K 2008 *Fusion Eng. Design* **83** 181

[20] Yuan Q P, Xiao B J, Luo Z P, Walker M L, Welander A S, Hyatt A, Qian J P, Zhang R R, Humphreys D A, Leuer J A, Johnson R D, Penaflor B G, Mueller D 2013 *Nucl. Fusion* **53** 043009

[21] Fu P, Liu Z Z, Gao G and Yang L 2010 *IEEE Conference on Industrial Electronics and Applications*, June 15–17, 2010, Taichung, Taiwan, Vol. 10, p. 457

[22] Xiao B J, Yuan Q P, Luo Z P, Huang Y, Liu L, Guo Y, Pei X F, Chen S L, Humphreys D A, Hyatt A, Mueller D, Calabróe G, Crisantie F and Albanesef R 2016 *Fusion Eng. Design* **112** 660

[23] Guo Y, Xiao B J, Liu L, Yang F, Wang Y H and Qiu Q L 2016 *Chin. Phys. B* **25** 115201

[24] Huang Y, Lao L L, Xiao B J, Luo Z P and Yue X N 2015 *57th Annual Meeting of the APS Division of Plasma Physics*, November 16–20, 2015, Savannah, USA, APS Meeting Abstracts

[25] Pascal architecture whitepaper, NVIDIA Tesla P100-The Most Advanced Data Center Accelerator Ever Built, http://www.nvidia.com/object/pascal-architecture-whitepaper.html