Acceleration optimization of real-time equilibrium reconstruction for HL-2A tokamak discharge control

Rui MA (马瑞), Fan XIA (夏凡), Fei LING (凌飞) and Jiaxian LI (李佳鲜)

Southwestern Institute of Physics, Chengdu 610041, People’s Republic of China

E-mail: marui@swip.ac.cn

Received 9 August 2017, revised 17 October 2017
Accepted for publication 17 October 2017
Published 4 January 2018

Abstract

Real-time equilibrium reconstruction is crucially important for plasma shape control in the process of tokamak plasma discharge. However, as the reconstruction algorithm is computationally intensive, it is very difficult to improve its accuracy and reduce the computation time, and some optimizations need to be done. This article describes the three most important aspects of this optimization: (1) compiler optimization; (2) some optimization for middle-scale matrix multiplication on the graphic processing unit and an algorithm which can solve the block tri-diagonal linear system efficiently in parallel; (3) a new algorithm to locate the X&O point on the central processing unit. A static test proves the correctness and a dynamic test proves the feasibility of using the new code for real-time reconstruction with 129 × 129 grids; it can complete one iteration around 575 μs for each equilibrium reconstruction. The plasma displacements from real-time equilibrium reconstruction are compared with the experimental measurements, and the calculated results are consistent with the measured ones, which can be used as a reference for the real-time control of HL-2A discharge.

Keywords: tokamak, equilibrium reconstruction, plasma displacement, GPU, CUDA

(Some figures may appear in colour only in the online journal)

1. Introduction

The toroidal and poloidal magnetic field configurations are the base of tokamak plasma operation. Optimum performance of a tokamak discharge requires accurate feedback control of many discharge parameters. As is well known, it is very difficult to obtain the shape information of plasma, such as shape and safety factor profile, directly. However, it can be evaluated from the available diagnostic data by solving the Grad–Shafranov equation, which describes the force balance of the tokamak equilibrium; it is called equilibrium reconstruction. There are many codes which are used, developed, and implemented on tokamak devices. The most frequently used code is EFIT [1, 2] which can reconstruct the plasma equilibrium offline. Its algorithm is computationally intensive such that it cannot be used directly for real-time discharge control. To compute discharge parameters in real time rapidly enough for tokamak feedback control, simpler algorithms implemented at most tokamak facilities [3–11] approximate the results of the full equilibrium reconstruction. For example, the P-EFIT has been used on EAST, which can complete a magnetic equilibrium reconstruction iteration in 300 μs with 65 × 65 grids. In addition, for plasma position and shape control, optical plasma boundary reconstruction has been developed on some devices by optical imaging diagnosis combined with OFIT/EFIT code [12–14].

At present, a real-time version of EFIT written in C programming language has been used by HL-2A; it scales the 1.92 m × 1.2 m rectangle plasma area by 33 × 33 grids and completes one iteration in 1 ms. In order to improve the accuracy, the equilibrium reconstruction will be performed by 129 × 129 grids. With the increase in the grid number, computation time increases more. However, it could not meet the real time requirement, so we must optimize the code. After analyzing the reconstruction algorithm, two methods have been used. First, massively parallel processing cores of
graphic processing units (GPUs) have been used efficiently for compute-intensive, highly parallel computation. Second, we have optimized the parts which could not be paralleled by the other algorithm, and these parts would still be executed on the central processing unit (CPU). After optimization, only 575 μs is needed for one-time complete iteration with a 129 \times 129 grid.

GPU has hundreds of processing units and high memory bandwidth, so it can achieve much stronger floating-point compute capability than CPU. More specifically, the GPU is especially well suited to address problems that can be expressed as data-parallel computations; the same program is executed on many data elements in parallel—with high arithmetic intensity—with the ratio of arithmetic operations to memory operations. The programming model assumes that the parallel computations execute on a GPU that operates as a co-processor to the host running the C program. The CPU works as a host. This method lets the CPU and GPU do collaborative work.

NVIDIA has developed compute unified device architecture (CUDA™) [15], a general purpose parallel computing platform and programming model that leverages the parallel compute engine in NVIDIA GPUs to solve many complex computational problems in a more efficient way than on a CPU. Developers can write GPU applications in standard languages like C, as a high-level programming language. A CPU. Developers can write GPU applications in standard languages like C, as a high-level programming language. A detailed description of CUDA™ can be found in the CUDA C PROGRAMMING GUIDE.

2. Basic procedure of equilibrium-reconstruction algorithm

The real-time equilibrium reconstruction code of HL-2A is evolved from the EFIT; its task is to compute the poloidal flux \( \varphi \) and the toroidal current density \( J_t \) distributions in the \((R, Z)\) plane. For the toroidal symmetry of a tokamak, ideal magnetohydrodynamic (MHD) equilibrium is described by the second-order partial differential Grad–Shafranov (G–S) equation for \( \varphi \):

\[
\Delta \varphi + \mu_0 R J_t = 0,
\]

Here, \( \varphi_p \) represents the flux created by the plasma current. The total poloidal flux is \( \varphi = \varphi_p + \varphi_{coil} \), where \( \varphi_{coil} \) is the poloidal flux generated by coils external to the plasma.

The toroidal current density profile consists of two terms: the plasma pressure \( P \) and the flux function of the poloidal plasma currents \( F \). It is represented as a linear combination of \( P \) and \( F \) through the following functions:

\[
J_t(R, \varphi; \alpha, \gamma) = R \left( P' \frac{\varphi}{\alpha} + \frac{\mu_0 FF' \varphi}{4\pi^2 R^2} \right) \] (2)

\[
P'(\varphi; \alpha_m) = \sum_{i=0}^{m-1} \alpha_i (\varphi^i - \varphi^m) \] (3)

\[
FF'(\varphi; \beta_n) = \sum_{i=0}^{n-1} \beta_i (\varphi^i - \varphi^n). \] (4)

Here, \( P' \) and \( FF' \) are functions of the normalized flux \( \varphi \) and the free parameters \( \alpha \), where \( \alpha = (\alpha_0, \beta) \), \( \varphi_p = (\varphi - \varphi_{axis})/(\varphi_{bdy} - \varphi_{axis}) \) is the flux normalized to the flux difference from the centre to the edge of the discharge, and \( \varphi_{axis} \) and \( \varphi_{bdy} \) are the fluxes at the magnetic axis and at the last closed flux surface, respectively. Because the current in the last closed flux surface is zero, the polynomial approximations of \( P \) and \( F \) are shown as expressions (3) and (4).

In the real-time algorithm, \( n \) and \( m \) are equal to 1. \( R \) and \( Z \) are the vector of values of the major radius at the grid points [16], so the plasma current at the grid elements could be represented as

\[
I_p = H \times \alpha = \left( R \varphi, \frac{1}{R} \varphi \right) \times (\alpha, \beta)^T. \] (5)

The operation \( \times \) between matrix or vectors in this paper indicates the general matrix product. The Green’s function coefficients relating current to the grid elements to signal at the diagnostics present as \( G_p \), which can be pre-computed. Then the diagnostic data resulting from the plasma current are

\[
G_p \times I_p = G_p \times H \times \alpha. \] (6)

The currents in the external poloidal field coils are also free parameters in the fitting problem. If \( G_c \) is represented as the Green’s function coefficients of external coils and the external diagnostics, the contribution by external coils is \( G_c \times I_e \), where \( I_e \) is the vector of external current sources. Thus, the total unknown vectors are \( X = (I_e, \alpha) \) for the fitting problem. If \( D \) represents the measured diagnostic data vector, the relationship between the diagnostic and unknown vector could be written as

\[
D = (G_c G_p \times H) \times (I_e \alpha). \] (6)

This is an over-determined system which could be solved through least-square best fit; then the plasma current profile could be computed using expression (5). Further, the total poloidal flux distribution could be obtained.

In the off-line equilibrium-reconstruction algorithm, the G–S equation is transformed into a block tri-diagonal linear system with a Dirichlet boundary condition by discrete difference. The numerical solution of the block tri-diagonal linear system is adopted by the Buneman’s method. The flux on the boundary grid points could be obtained as \( \varphi_p = G_p \times I_p \). This method is very time consuming, so in the real-time code we adopt another method to solve the equation.

Before the numerical solution, we need to determine the calculation area. Because the vacuum chamber of HL-2A and the position of the limiter are not coincident, the calculation region \((R \times Z)\) corresponds to \( R \in [1.05, 2.25] \) and \( Z \in [-0.96, 0.96] \), where \( R \) and \( Z \) are in meter. Figure 1 shows the schematic diagram of the meshed vacuum chamber, where we can find that many elements are out of the limiter. As shown in figure 1, the blue points are grid points inside the limiter while the light black ones are points outside the limiter. In order to reduce the computation time, only the grid points inside the limiter participate in calculation because their plasma current always is zero.

The basic flow path of the equilibrium reconstruction code could be written as:
(1) Compute the initial normalized flux.
(2) Compute the response matrix \((G_c G_p \times H)\).
(3) Compute the plasma current distribution.
(4) Solve the G–S equation, refresh the flux.
(5) Check the error. If not converged, go back to step 2 and continue; else calculate the plasma parameters.

Before the optimization, we counted the time consumption of every step. The following four parts are the most time consuming: computation of response matrix, flux on boundary and the plasma parameters, and numerical solution of the G–S equation. Therefore a high degree of parallelism must be achieved for each step.

3. Optimization of real-time equilibrium-reconstruction algorithm

In this paper, the main logical part of the code is still performed on the CPU. The optimization design follows the following principle: the algorithms that can be highly parallelized will be implemented in the GPU, and the parts that are not suitable for execution on GPU will be optimized by other algorithms.

3.1. CPU compiler optimization

The implementation efficiency on CPU is very low: one complete iteration requires about 76 ms. In order to make better use of CPU computing power, the modern compiler has a variety of optimization measures for the code. We use the Intel compiler here. Through the compiler options and the auto-vectorization, the time to complete one iteration will be reduced to 6 ms.

Compiler option optimization is to set the appropriate compiler option when the code is compiled, the advantage being that the compiler automatically analyzes the code without modifying it. Because the structure of the algorithm and the data dependence relation is opaque, only a very conservative optimization will be done to ensure the correctness. In order to lower the compilation difficulty and simplify the compilation process, the compiler implements a hierarchical package optimization option. The most common options include \(-O0, -O1, -O2, -O3, -Os\) and \(-O fast\). In the present calculation, the \(-O2\) is employed.

The most commonly used method of auto-vectorization is to achieve a number of floating-point calculations by accessing registers with multiple pipeline units. We use single instruction multiple data (SIMD) to improve the register utilization. For the parts in which SIMD optimization cannot be automatically carried out, we add the Intel compiler directive statement similar to openMP in the program. This way, the compiler can use SIMD in more parts.

3.2. Parallelization by GPU

After the optimization of the CPU compiler, the solution of the G–S equation becomes the most time-consuming part of the whole reconstruction process. The discrete difference of the G–S equation would create a block tri-diagonal linear system. We decouple the original linear system into an N independent tri-diagonal linear system by using discrete sine transform method \[17\]. Thus, we can use the GPU to solve the original problem in parallel; it takes the CPU 3030 \(\mu s\) when using Benemann’s method. With the following algorithm, however, this operation only takes the GPU 455 \(\mu s\). All tests are conducted on a workstation running Red Hat 6.7 with an Intel Xeon ®E5-2695 CPU and a NVIDIA Tesla k20X GPU.

3.2.1. Parallelization of flux calculation on boundary grid point. The flux on each grid boundary point would be obtained by the Green’s function method. There are 512 boundary grid points, so it is a multiplication of a 512-by-16641 matrix and a 16641-by-1 matrix. It takes the CPU 2030 \(\mu s\) when calculated by serial code. Because all plasma currents outside the plasma boundary are zeros, we only consider the points inside, which are shown in figure 1. Then the matrices are reduced to 512-by-5090 and 5090-by-1. We have tried to use the GPU-accelerated version of CUBLAS to accelerate, but the acceleration effect cannot reach the expected demand, because CUBLAS is mainly designed for large-scale computation and \(G_p \times I_p\) is only a middle-scale task. For this reason, a special parallel algorithm is designed based on the CUDA architecture \[18\].

In view of the hardware characteristics of the GPU, threads assigned to each block must be an integer multiple of...
32. Therefore, 30 columns and 30 rows of zeros are added to the matrices \( \mathbf{G}_p \) and \( \mathbf{I}_p \), respectively, before the computation. The calculation process is shown in figure 2. The GPU used in this paper is Tesla k20x which has thousands of computation cores. In order to improve the computational efficiency, it is necessary to consider its performance in its entirety. Two kernel functions have been used to calculate the grid boundary flux through two compressions. First, the original problem was divided into 40 parts, each holding 128 rows of the matrix \( \mathbf{G}_p \). When implemented in the GPU, each part would correspond with a ‘block’ and 512 threads would be assigned to every block. Then every thread would multiply the element on each row of matrix \( \mathbf{G}_p \) and the corresponding current value of matrix \( \mathbf{I}_p \), and finally the 128 column values are accumulated. After that process, the original problem was transformed into calculating the sum of elements on every row of a 512-by-40 matrix. These calculations are implemented in the first kernel function. Because there is no global synchronization in CUDA, a new kernel function was used to accumulate the elements. The matrix is processed using 16 blocks to obtain the final result. There are 32 threads which correspond with one row in every block. Finally, we obtained a vector with 512 elements, which is the flux on the grid boundary. Using this algorithm, the flux computation could be completed in 130 \( \mu \)s, which include the time consumed by data transmission from GPU to CPU.

In HL-2A, there are 5 poloidal coil currents, plasma current, 4 flux loops, and 18 magnetic probes used for plasma control. Therefore, the dimension of the response matrix is \( 5090 \times 28 \). The acceleration of the response matrix multiplications used a similar strategy, with some modification made according to the matrix dimensions.

\[ \begin{bmatrix} U_{j-3} & U_{j-2} & U_{j-1} & U_j & U_{j+1} & U_{j+2} & U_{j+3} \\ [j-2] & 0 \lambda' & 0 & d_{j-1} \\ [j-1] & b'_{j-1} & 0 & \lambda' & 0 & d_{j-1} \\ [j] & b'_{j1} & 0 & \lambda' & 0 & d_{j+1} & s_j' \\ [j+1] & b'_{j+1} & 0 & \lambda' & 0 & d_{j+1} & s_{j+1}' \\ [j+2] & b'_{j+2} & 0 & \lambda' & 0 & d_{j+1} & s_{j+2}' \end{bmatrix} \]

**Figure 3.** Schematic diagram of GPU parallel solution for G–S equation.

**Figure 4.** Basic flow path of the cyclic reduction method.

bottleneck and introduce parallelism at each step of the computation. The original problem is decoupled using the discrete sine transform (DST) method. After a DST and a transpose, it is transformed into 127 independent triangular linear systems with size 127, which could be solved on the GPU in parallel.

There would be 127 blocks on the GPU, in which a triangular linear system could be solved as shown in figure 3. A triangular linear system would correspond with fluxes on one row of the meshed area. The element on every column of the row would be obtained in a thread, so that 127 threads would be assigned in every block. Many methods are used to solve the triangular linear equation group; here we used the cyclic reduction method.

Figure 4 shows the basic flow path of the cyclic reduction method; it is to eliminate the non-diagonal elements of the former, and after the \( 2^{n-1} \) row by the diagonal element, until all non-diagonal elements are zeros. For the \( 127 \times 127 \) coefficient matrix, the equation group could be decoupled into a linear diagonal system after seven iterations. Then we can obtain the fluxes on every row directly. Each iteration has the same rule. For the \( n \)th elimination, when the row number is less than \( 2^{n-1}+1 \), only nonzero elements on the right side of the diagonal need to be eliminated by the diagonal element of the next \( 2^n-1 \) row. Similarly, only elements which are on the left side of the diagonal need to be eliminated by the diagonal element of the former \( 2^n-1 \) row when the row number is more than \( 127-2^{n-1} \). Thus, the eliminations of nonzero elements in the row before \( 2^{n-1}+1 \) are assigned to the threads between \((0, 2^{n-1})\). The eliminations for the row after the \( 127-2^{n-1} \) are assigned to the threads between \((127-2^{n-1}, 127)\). For the \( 2^{n-1} \)th to the \((127-2^{n-1}-1)\)th rows, all nonzero elements need to be eliminated. These computations are assigned to the threads between \((2^{n-1}, 127-2^{n-1})\). Because the mesh is fixed, the coefficient matrix could be pre-computed. The coefficient matrix of each reduction could be transmitted to the GPU before reconstruction. Thus, in the process of real-time reconstruction, only the right side of the equation group needs to be calculated in every-time iteration. In order to reduce the
amount of data and shorten the GPU addressing time, the coefficient matrix only stores the non-zero data of each line, and so the size of the coefficient matrix is reduced to 127 × 3. With the algorithm just described, solving the block tri-diagonal equation group only takes the GPU 340 μs.

Solution of the block tri-diagonal equation group by P-EIFT takes only 40 μs, because most parts of P-EIFT are parallelized by GPU. This approach avoids a large amount of data exchange between CPU and GPU, which is the bottleneck of CPU and GPU communication.

3.3. Algorithm optimization of CPU

In the off-line EFIT, the X point is located using Bicubic Spline interpolation, which needs to calculate the interpolation coefficient matrix. Each value in the matrix is related to the information on the grid point. This method could not be paralleled in theory and takes the CPU about 2500 μs for one time slice. In addition, the interpolation coefficient matrix of each time slice must be recalculated. A new strategy based on biquadratic interpolation is adopted in this paper, which can satisfy the requirement of accuracy by 129 × 129 grids.

The poloidal magnetic field \( B_p \) at the X point is equal to zero. It could be represented as

\[
B_p^2 = B_r^2 + B_z^2 = \frac{1}{R^2} \left( \frac{\partial \psi}{\partial R} \right)^2 + \frac{1}{R^2} \left( \frac{\partial \psi}{\partial Z} \right)^2 = 0
\]

(7)

where \( B_r \) and \( B_z \) are rough values since the gradient of \( \psi \) is computed with a finite difference using the flux value of adjacent grid points. Thus, the forward and backward differences of \( B_r \) and \( B_z \) meet the following expression [20]:

\[
B_r^+ \cdot B_z^- < 0, \quad B_r^- \cdot B_z^+ < 0.
\]

(8)

This criterion method used to find the X point position and compute its magnetic flux. The basic flow path of this strategy is as follows: \( B_r^+ \cdot B_z^- \)

(1) Define a computing box in which the X point is most likely to exist. Compute forward and backward differences of all grid points in the \( R \) and \( Z \) directions. The point \( A \) which satisfies expression (8) is the nearest grid point to the X point.

(2) Bring point \( A \) and five other grid points around it into the biquadratic interpolation polynomial, which is represented as

\[
\psi(R, Z) = a_0 R^2 + a_1 Z^2 + a_2 RZ + a_3 R + a_4 Z + a_5.
\]

(9)

Then we can get \( a_0-\alpha_5 \).

(3) Having the polynomial coefficients, the position \( (R, Z) \) of the X point could be obtained by the following expressions:

\[
B_r = -\frac{1}{R} \frac{\partial \psi}{\partial Z} = 0,
\]

\[
B_z = -\frac{1}{R} \frac{\partial \psi}{\partial R} = 0.
\]

(10)

Then the flux at the X point would be determined by bringing \((R, Z)\) into expression (9).

The method to find the magnetic axis is similar to the one used for the X point. However, their computation regions are different, which can be shown in figure 5. When implemented in CUDA, each part would correspond with a ‘block’, which is a thread union of the CUDA program model. Then each thread would execute the judgment of expression (8) on every grid point in the box. The strategy to determine the boundary flux is similar to the one used by P-EIFT. After locating the X point, the poloidal flux on the X points would be compared with the maximum poloidal flux of selected limiter points. If the former is bigger, it was a divertor discharge and the poloidal flux at that X point would be taken as the boundary flux. Otherwise, it was a limiter discharge, and the maximum flux on that limiter point would be taken as the boundary flux. Because only the normalized flux would be used in real-time control discharge, it does not need to calculate the detailed positions of the points on the plasma boundary. After this adjustment, this operation only takes the CPU 10 μs compared to the old algorithm, which takes 2900 μs to complete the computation for one slice.

4. Benchmark test

Static and dynamic benchmark tests are carried out to prove the accuracy of the code. Two tests are performed to compare the equilibrium with offline EFIT; the first is a slice equilibrium test and the other is a discharge pulse on HL-2A test continuously.

![Figure 5. Box1 and box2 are the scope of the X point and magnetic axis location.](image)
4.1. Static test

In this test, the main purpose is to verify the correctness and convergence of the code through comparing the reconstruction result with the offline EFIT results. The input measured data is a set of measurement data on the HL-2A magnetic diagnostics, which is created by offline EFIT using a fixed boundary equilibrium calculation mode.

Figure 6(b) shows the convergence rate of the computation. It is so quick that the convergent error drops to less than 5.0E–3 after about 14 iterations, which indicates a high-quality equilibrium reconstruction. The poloidal flux computed by real-time code is plotted in figure 6(a) and compared with the flux computed by offline EFIT. They almost coincide with each other, and the shape error is less than 0.5 mm.

In a real discharge experiment, the measurement data of the magnetic diagnostic has some uncertainty [21–24]. To simulate a real situation, 3% random error is added into the input signal and then we reconstruct the magnetic field configuration to verify the correctness of the code. As figure 7 shows, the shape error is less than 5 mm, which has no significant effect on the reconstruction. All these results show that the real-time equilibrium reconstruction code has a fast convergence speed and good stability.

4.2. Dynamic test

The static test has proved the correctness of the real-time equilibrium reconstruction code. However, 13–14 iterations, which take 575 μs to complete computation, are required to obtain an accurate-enough equilibrium result. This means that the time consuming of one slice will rise to a few milliseconds, which still does not meet the requirement of real-time. Thus, a strategy similar with P-EFIT is adopted, whose basic thought is to use the equilibrium result of the last time-slice as the initial input for the next computation. Only a one-time iteration is conducted for each slice. After this operation, the real-time requirements are satisfied.

In order to verify the feasibility of this strategy, another benchmark test is performed using the magnetic measurement data of HL-2A. The measurement data of shot 27 500 is used as the input data. This discharge has a standard down single-null divertor configuration, and the duration time is around 1900 ms. We reconstruct equilibrium configurations after \( t = 400 \) ms, when
the plasma has reached the steady-state situation. Then we compare the result with the convergence result reconstructed by offline EFIT. Figure 8(a) shows the reconstruction results at \( t = 400 \text{ ms} \), equivalent to the first iteration. The shape error rises to 5 mm because only one-time iteration is conducted. Figure 8(b) shows the results at \( t = 500 \text{ ms} \), equivalent to the 100th iteration. When reconstructed at \( t = 1000 \text{ ms} \), the shape error of the plasma center is about 1 cm, but the plasma boundary is accurate even with higher than 3% noise existing on the measurements. In addition, horizontal and vertical shift control are important in real-time control of HL-2A plasma discharge. In figure 9, horizontal and vertical displacement computed from real-time equilibrium reconstruction code are compared with experimental measurement values. The calculated values are consistent with the measured ones, which can be used as a reference for the real-time control of HL-2A discharge.

5. Summary and prospects

We have presented the execution details of the four most important aspects of acceleration optimization of equilibrium reconstruction in this paper. The real-time equilibrium reconstruction code is nearly identical to a standard equilibrium reconstruction technique, and proves to be feasible to use in real time for HL-2A identification of tokamak discharge parameters. We take 900 time slices as a statistical sample. With \( 129 \times 129 \) grid number, two polynomial coefficients, and 28 magnetic diagnostics, the code could complete a full iteration in 610 \( \mu \text{s} \) for all the slices in the sample. Among them, 80% of slices could be completed between 570 and 580 \( \mu \text{s} \).

We have presented an efficient CUDA implementation solution method, but most of the computing time in the code is still spent in solving the Poisson equation. In the future, this part will be accelerated continuously by other parallel methods used on more powerful GPUs. The computation region of HL-2M is always larger in design, and the magnetic diagnostics are about three times as much as HL-2A. This will lead to a more complex algorithm and increase in computation time. Therefore, we will design a new real-time equilibrium reconstruction system which is more suitable for HL-2M.

Acknowledgments

This work is supported in part by National Natural Science Foundation of China under Grant Nos. 11505050 and 11775071, and by the ITER-CN under Grant No. 2014GB107003.

References

[23] Shen B et al 2009 Fusion Eng. Des. 84 19