CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2010; **00**:1–17
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

Scaling the SIESTA Magnetohydrodynamics Equilibrium Code

Sudip K. Seal*, Kalyan S. Perumalla* and Steven P. Hirshman†

*Computational Sciences and Engineering Division, †Fusion Energy Division
Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA
{*sealsk, perumallaks, hirshmansp*}*@ornl.gov*

SUMMARY

We report the results of a scaling effort that increases both the speed and resolution of the SIESTA magnetohydrodynamic equilibrium code. SIESTA is capable of computing three-dimensional plasma equilibria with magnetic islands at high spatial resolutions for toroidally confined plasmas. Starting with a small-scale parallel implementation, we identified scale-dependent bottlenecks of the code and developed scalable alternatives for each performance-significant functionality, cumulatively improving both its runtime speed (on the same number of processors) as well as its scalability (across larger number of processors) by an order of magnitude. The net outcome is an improvement in speed by over ten-fold, utilizing a few thousand processors, enabling SIESTA to simulate high spatial-resolution scenarios in under an hour for the first time.

1. INTRODUCTION

Magnetohydrodynamic (MHD) equilibrium codes are important for a wide range of plasma applications. They are used for optimizations and data analyses related to design of tokamaks and stellarators [18], reconstruction of plasma states from experimental data [13, 7] and initialization of extended MHD time-dependent codes [14, 17, 3]. SIESTA (Scalable Iterative Equilibrium Solver for Toroidal Applications, [10]) is an iterative MHD equilibrium solver [4, 8] that enables the exploration of a wide range of new scenarios to be simulated in support of all the aforementioned areas.

Next generation plasma applications, such as those for the International Thermonuclear Experimental Reactor (ITER), require MHD simulations at unprecedented spatial resolutions. Today's supercomputers have the computational capability of delivering MHD simulations with significantly better accuracy and speed than before. But, a highly sophisticated framework such as SIESTA, developed over several years, was not originally designed to exploit such large-scale parallelism. To gainfully utilize SIESTA's capabilities, execution of its core iteration scheme must be capable of sufficiently high speed to enable scientists to simulate a large number of scenarios within a given amount of wall clock time. The original implementation of SIESTA was limited in this aspect in both speed (taking many hours per run on large runs) as well as spatial resolution scale (runtime increasing much more than logarithmically with problem dimensions), warranting improvement and optimization of its parallel execution capabilities. Rewriting SIESTA for improved scaling is certainly one way to meet the scaling needs, though an extremely expensive one in terms of manpower and development time. Instead, introducing a series of efficient, non-intrusive optimizations offers an alternative approach to accelerating SIESTA's performance and improving its scalability. In this paper, we adopt the latter approach.

Existing equilibrium codes like VMEC [11] and EFIT [13] depend on an underlying assumption of the existence of nested magnetic surfaces to achieve high performance. SIESTA avoids such restrictions, albeit at somewhat higher computational cost. Other iterative solvers – such as the PIES code [15] – exist but use very different solution methods. SIESTA departs from previous iterative fluid solvers by using an accurate physics-based preconditioner to accelerate the convergence of the solution of the linear ideal MHD equations (which are expansions of the nonlinear force $\mathbf{F}_{MHD}$ around an instantaneous plasma state not yet in equilibrium). As usual, preconditioning transforms the linearized system into one with more favorable numerical convergence properties (smaller spectral radius or condition number) for the iterative procedure. SIESTA uses the nested magnetic flux surfaces computed by the VMEC [11] code to provide both an optimized set of quasi-polar background coordinates and an initial guess for the iterative procedure. The spectral nature of SIESTA's formulation provides a natural way to implement conservative forms of the evolution equations for the contravariant components of the magnetic field and the plasma pressure. It departs from the BETAS [1] code which uses an inverse representation with an integrable Hamiltonian perturbation to describe a single magnetic island. SIESTA is not limited to such integrable (single resonant surface) situations and only uses a nested inverse representation for the background coordinate system.

SIESTA executes multiple complex computational kernels that exhibit a wide range of domain-specific and parameter-dependent execution behaviors. Here, we report results from improvements which show dramatic increases in both the execution speed of SIESTA simulations, as well as the problem sizes that can be handled by SIESTA. Starting with an original version developed for small-scale parallel execution, we make a series of improvements for efficient large-scale parallel execution. The complex nature of the code makes detection of performance bottlenecks very challenging. We address this by combining performance profiling, improved process synchronization and customized implementations of scalable and efficient parallel solvers. Altogether, these improvements yield a net increase in SIESTA's speed by an order of magnitude (compared to its initial small-scale parallel implementation), and demonstrate scaling to a few thousands of processor cores.

Section 2 describes the notation and terminology used in this paper. A brief background of the physics behind the computations is described in the Section 3. The computational flow of execution that is at the core of SIESTA is presented in Section 4. Section 5 through Section 8 describe the optimizations that have been performed over the original version of SIESTA. The net performance gain in the capabilities of SIESTA due to these optimizations is discussed in Section 9. Conclusions are presented in Section 10.

2. TERMINOLOGY AND NOTATION

The original small-scale parallel implementation of SIESTA is referred to as the *unoptimized* version while the version resulting from the optimization and scaling efforts described in this paper is referred to as *optimized*. The subscripts u and o are used to refer to runtimes resulting from executing the unoptimized or optimized versions of SIESTA, respectively. We avoid using the term *speedup* which is conventionally used to compare runtimes on larger number of processors with that on a (usually fixed) smaller number of processors. Instead, we define a more relevant measure here called *performance gain* as the ratio of the runtime of an unoptimized computational kernel to that of its optimized version running on the *same* number of processors P. In the context of a block-tridiagonal preconditioner matrix, to be introduced soon, the number of block rows will be denoted by N and the block size by M. Further, for ease of presentation, we will assume $N = 2^n$, and the number of processors $P = 2^q$, where n and q are non-negative integers (although SIESTA's implementation does not have these restrictions).

All experimental results reported in this paper are based on executions on a Cray XT4 machine with a quad-core 2.3 GHz single socket AMD Opteron processor (Budapest) and 8 GB of memory in each compute node. The nodes are connected via a high-bandwidth SeaStar interconnect. The software was implemented in Fortran 95 with Message Passing Interface (MPI) for inter-processor

DOI: 10.1002/cpe

communication. All block-level matrix-matrix and matrix-vector multiplications as well as block-level matrix inversions were carried out using BLAS routines optimized for best performance on Cray systems and belonging to the Cray Scientific Libraries, LibSci.

3. BACKGROUND

The MHD energy W of a stationary plasma (velocity $\mathbf{v} = 0$) with magnetic field $\mathbf{B}$ and pressure p is:

$$W = \int \left[\frac{B^2}{2\mu_0} + \frac{p}{\gamma - 1} \right] dV, \tag{1}$$

where the integral is over the plasma volume V and μ_0 is the permeability of free space. Taking the time derivative of the above equation and using Maxwell's equations (see [10] for details) to substitute for the appropriate physical observables yields the following MHD energy principle:

$$\frac{\partial W}{\partial t} = - \int \left[\mathbf{v} \cdot (\mathbf{J} \times \mathbf{B} - \nabla p) - \eta J^2 \right] dV, \tag{2}$$

where $\mathbf{v}$ is the velocity field, $\mathbf{J} = (\nabla \times \mathbf{B})/\mu_0$ is the current density and η is the resistivity. For sufficiently small resistivity ($\eta \rightarrow 0$) and treating $\mathbf{v}$ as a variational parameter, an equilibrium state ($\partial W/\partial t$=0) is reached where W becomes quasi-stationary when the following ideal MHD force balance is satisfied:

$$\boldsymbol{\mathcal{F}} \equiv \mathbf{J} \times \mathbf{B} - \nabla p = 0. \tag{3}$$

3.1. MHD Equilibrium Principle in Curvilinear "Flux" Coordinates

The flux coordinate system is defined in terms of three independent coordinate variables, namely, the poloidal angle u, the toroidal angle v and the radial flux coordinate s. For stellarator-symmetric plasmas, this coordinate system maps to the cylindrical coordinate system (R, ϕ, Z) as follows:

$$R = \sum_{m,n} R_{mn}(r) \cos(mu + nv) \ , \ Z = \sum_{m,n} Z_{mn}(r) \sin(mu + nv) \ , \ \phi = v \ , \ \text{ where } r = \sqrt{s}.$$

Here, R_{mn} and Z_{mn} are Fourier coefficients. In the flux coordinate system, the covariant components of the magnetic field B_i are linearly related to the contravariant components B^i by the metric tensor elements g_{ij} as follows:

$$B_i = g_{ij} B^j \ , \ g_{ij} = R_i R_j + Z_i Z_j + \delta_{vv} R^2,$$

where the subscripts on R and Z denote partial derivatives ($R_r \equiv \partial R/\partial r$, etc) and summation on repeated indices is implied. Minimization of the MHD energy (see Eqn (1)) in the flux coordinate system can be shown (see [10]) to be equivalent to the following minimization principle:

$$\delta W = - \int \mathcal{P}^{ij} \mathcal{F}_i \mathcal{F}_j dV \leq 0,$$

where

$$\mathcal{F}_i = \mu_0^{-1} B^j \left(\frac{\partial B_i}{\partial x_j} - \frac{\partial B_j}{\partial x_i} \right) - \frac{\partial p}{\partial x_i},$$

and $\boldsymbol{\mathcal{P}}$ is any positive definite matrix.

Prepared using cpeauth.cls
DOI: 10.1002/cpe

3.2. Preconditioner

Let the Hessian matrix operator $\mathbf{H}$ be defined as:

$$H_{ij} = \frac{\partial \mathcal{F}_i}{\partial \xi_j}, \tag{4}$$

where $\boldsymbol{\xi}$ is the displacement vector in flux coordinates. It has been shown [10] that the steepest descent method to minimize the preceding MHD energy can be greatly accelerated by choosing $\boldsymbol{\mathcal{P}}$ at the k^{th} iteration to be a physics-based preconditioner of the following form:

$$\mathcal{P}^k_{ij} = (-\widetilde{H}^k_{ij} + \lambda^k I_{ij})^{-1}, \tag{5}$$

where λ^k is an eigenvalue shift at iteration number k and

$$\widetilde{\mathbf{H}}^k = x\mathbf{H}^k_{approx} + (1-x)\mathbf{H}^k_{exact} \tag{6}$$

is the effective Hessian matrix ($0 \leq x \leq 1$). $\mathbf{H}^k_{approx}$ is a negative self-adjoint approximation of the exact Hessian for the system when it is *far from equilibrium*. It can be shown to have the following form:

$$\mathbf{H}^k_{approx}(\boldsymbol{\xi}^k) = \mu_0^{-1}\nabla \times [\nabla \times (\boldsymbol{\xi}^k \times \mathbf{B}^k)] \times \gamma\nabla(p\nabla \cdot \boldsymbol{\xi}^k) \tag{7}$$

The shift parameter λ^k is added to the diagonal elements to remove the null space of the Hessian operator. Its value is reduced based on a pseudo-transient continuous method [12] as equilibrium is approached:

$$\lambda^k \sim \lambda_0 L^k \ , \ L^k = \sqrt{|F^k_{MHD}|^2/F^0_{MHD}|^2}.$$

The blending parameter x is initially chosen to be equal to 1 and approaches 0 as an MHD equilibrium state is reached.

4. EXECUTION FLOW OF SIESTA

From the computational point of view, during each iteration k of a SIESTA simulation, a set of *non-linear* force equations $\boldsymbol{\mathcal{F}}^k = 0$ is solved. Numerically, this non-linear set of equations is approximated by a Taylor expansion of Eqn (3) as:

$$\boldsymbol{\mathcal{F}}^k = \mathbf{F}^k + \mathbf{H}^k \cdot \boldsymbol{\xi}^k + O(\xi^2) = 0. \tag{8}$$

Here, $\mathbf{F}^k$ is the residual nonlinear MHD force at iteration k, $\boldsymbol{\xi}$ is a displacement vector and $\mathbf{H}^k$ is the Hessian matrix defined as $H^k_{ij} = \partial \mathcal{F}^k_i / \partial \xi_j$. Ignoring the second-order terms, the original problem translates to numerically solving the following approximately linear set of equations:

$$\mathbf{H}^k \cdot \boldsymbol{\xi}^k = -\mathbf{F}^k \tag{9}$$

in each iteration k. To ensure that the higher-order non-linear terms can indeed be ignored, a small positive parameter λ is added to the diagonal elements of $\mathbf{H}$ to suppress large null-space eigenvectors. This yields the preconditioner $\boldsymbol{\mathcal{P}}$:

$$\boldsymbol{\mathcal{P}}^k = (-\mathbf{H}^k + \lambda^k \mathbf{I})^{-1}. \tag{10}$$

From Eqn (9), it follows that:

$$\boldsymbol{\mathcal{P}}^k \cdot \mathbf{H}^k \cdot \boldsymbol{\xi}^k = -\boldsymbol{\mathcal{P}}^k \cdot \mathbf{F}^k. \tag{11}$$

DOI: 10.1002/cpe

Algorithm 1 SIESTA's Execution Flow

1: initialize simulation parameters
2: **while** convergence criterion is not met, begin iteration k **do**
3: compute the Hessian matrix, $\mathbf{H}^k$
4: check self-adjointness of Hessian
5: construct the preconditioner $\boldsymbol{\mathcal{P}}^k = (-\mathbf{H}^k + \lambda^k \mathbf{I})^{-1}$
6: **while** GMRES convergence criterion is not met **do**
7: solve $\boldsymbol{\xi}^k \approx \boldsymbol{\mathcal{P}}^k \cdot \mathbf{F}^k$
8: **end while**
9: add resistive perturbations to $\mathbf{B}$
10: **end while**

As long as λ^k is small ($0.0001 \leq \lambda^0 \leq 1$ are typical values with $\lambda^k \rightarrow 0$ for $k \gg 1$), $\boldsymbol{\mathcal{P}}^k \approx -(\mathbf{H}^k)^{-1}$ and the following approximate solution is computed:

$$-\boldsymbol{\mathcal{P}}^k \cdot \mathbf{F}^k = \boldsymbol{\mathcal{P}}^k \cdot \mathbf{H}^k \cdot \boldsymbol{\xi}^k \approx -(\mathbf{H}^k)^{-1} \cdot \mathbf{H} \cdot \boldsymbol{\xi}^k = -\boldsymbol{\xi}^k. \tag{12}$$

Thus, in iteration k, the numerically computed approximate solution is $\boldsymbol{\xi}^k \approx \boldsymbol{\mathcal{P}}^k \cdot \mathbf{F}^k$ which is then improved using the GMRES (Generalized Minimal Residual) method [16].

Algorithm 1 summarizes the execution flow of SIESTA. Most of the steps in the algorithms are self-descriptive. Step 4 of the algorithm will be discussed shortly. Note that SIESTA's execution is highly non-uniform in terms of the inner and outer iterations – the number of iterations of the inner loop is not the same for every iteration of the outer loop. Profiling the original parallel version of SIESTA revealed that constructing the preconditioner was computationally one of the most expensive steps. Computing the preconditioner is a two-step process. First, the Hessian has to be computed, and then the augmented Hessian (see Eqn (10)) has to be inverted, both in parallel.

5. COMPUTING AND PARTITIONING THE HESSIAN MATRIX

Because of the second-order coupling in radius of the linearized force equations, $\mathbf{H}$ is block-tridiagonal. It consists of an $N \times N$ array of blocks where each block is an $M \times M$ array. The block size and the number of block rows, are both related to the spatial resolution of the simulation. $M = 3(2\mathcal{N}+1)(\mathcal{M}+1)$ where $(2\mathcal{N}+1)$ is the total number of toroidal Fourier modes. The toroidal mode number $n \in [-\mathcal{N}, \mathcal{N}]$. $(\mathcal{M}+1)$ is the total number of poloidal modes. The poloidal mode number $m \in [0, \mathcal{M}]$. The factor of 3 accounts for the three spatial components of the independent displacement vector. SIESTA constructs the blocks of the Hessian matrix one column (of length M) at a time using a mesh coloring scheme. For each column, three radial sweeps compute the full radial dependence of the Hessian.

Thus, based on the above description, each column in a block of the Hessian matrix can be uniquely identified by its global block row number, $i \in [1, N]$, its block type, $j \in [L, D, U]$ and its column position within the block, $k \in [1, M]$. The computational domain can, therefore, be thought of as a columnar space of $M(3N-2)$ columns, each tagged by an integer triplet, (i, j, k). The -2 term arises from the fact that the first and the last block row each have only two blocks. In its original form, SIESTA's parallel column generation algorithm partitioned the columnar space amongst the participating processors (ranks) in a column cyclic pattern. This is illustrated in Fig. 1(a) where the cyclic column partitioning shown is for an example with $N = P = 4$ and $M = 8$. Whereas this partitioning of the columnar space was suitable for the original parallel block matrix factorization kernel, the goal of this work is to relieve SIESTA's performance bottleneck posed by this very same kernel. As will be discussed soon, this is achieved by implementing and integrating a scalable block factorization algorithm based on cyclic reduction. On the other hand, the partitioning expected by a cyclic reduction-based algorithm is one in which the first $\frac{N}{P}$ block rows are mapped to the first rank, the second $\frac{N}{P}$ block rows to the second rank and so on. An example is shown in Fig. 1(b). A remapping kernel is, therefore, necessary to translate the original to the target domain partitioning.

Prepared using cpeauth.cls
DOI: 10.1002/cpe

There are two considerations of a remapping algorithm. First, even as the columns are being produced based on a source pattern, the columns need to be sent to the appropriate destination processors so that they conform to the target domain decomposition. Secondly, since this repartitioning is executed multiple number of times inside the second loop, its runtime must be as small as possible compared to the total runtime.

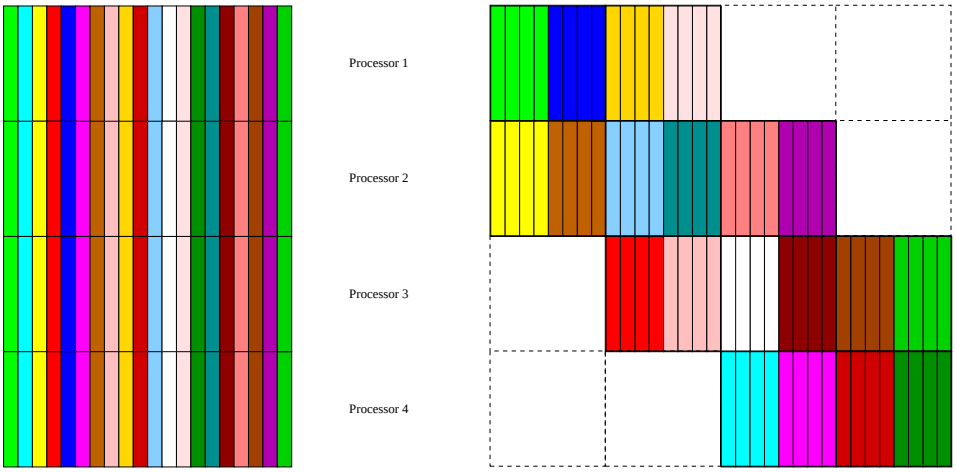
A relatively straightforward way to implement a repartitioning algorithm is to buffer the columns into appropriate messages for destination processors and ship them out using one collective call such as an $Allgather$ primitive after all local columns have been generated. This approach has two drawbacks, namely: (a) for realistic problem parameters, a very large memory buffer is needed to store the impending message, and (b) using collective calls blocks all the participating processors resulting in poor scalability.

In our implementation, only a small memory buffer is used; the task of remapping is achieved by adopting a multiple producers-consumers approach. All columns are, therefore, sent to the destination processor as they are produced, and the destination processor consumes them as they become available in the network buffer. Deadlock conditions that are absent in small-scale scenarios potentially arise when the Hessian size increases. To avoid deadlocks, care is taken to relieve incoming network buffers by using non-blocking primitives to receive and accept any incoming data even while new columns are being generated. The advantage of this approach is that very little amount of memory is required and processors can overlap communication with computation.

The overhead of repartitioning the columnar space across processors clearly adds to the time t spent within the Hessian construction kernel. However, this added time spent in repartitioning is more than compensated by a vastly reduced total runtime. In other words, using the notation established in Section 2, although $t_u < t_o$, $T_u \gg T_o$. Let $f_u = t_u/T_u$ be the percentage of time spent in the Hessian construction routine in the unoptimized version of SIESTA. The corresponding percentage of time spent in the optimized version f_o needs to be normalized to the same total runtime for comparison and is, therefore, defined as:

$$f_o = t_o/T_o = (t_o * (T_u/T_o))/T_u = x_{uo}(t_o/T_u)\,, \text{ where } x_{uo} = T_u/T_o.$$

Table I shows that the extra cost incurred in repartitioning the computational domain is compensated for by an order of magnitude reduction in the total runtime.



(a) Original Partitioning. (b) Target Partitioning.

Figure 1. An example remapping for the case of $N = 4$, $M = 8$ and $P = 4$ where each color within a processor represents an integer triplet (i, j, k). For general values of M, N, and P, the partitions are not all equal and care has to be taken to accommodate differing sized partitions.

DOI: 10.1002/cpe

P	N	$\frac{N}{P}$	t_u	t_o	T_u	T_o	f_u	f_o
64	64	1	1.43	1.93	69.37	21.89	2.10%	8.80%
128	128	1	1.55	2.04	211.95	30.22	0.70%	6.80%
256	256	1	1.87	2.40	832.80	61.29	0.20%	3.90%
512	512	1	4.99	6.17	2062.72	178.91	0.20%	3.40%
1024	1024	1	21.17	25.50	7786.86	749.87	0.03%	3.40%
64	128	2	3.00	3.90	144.10	30.44	2.10%	12.80%
128	256	2	3.62	4.50	453.99	62.06	0.80%	7.30%
256	512	2	5.03	6.25	1976.33	179.45	0.30%	3.50%
512	1024	2	21.18	25.80	7325.54	729.35	0.30%	3.50%
1024	2048	2	52.66	68.60	16422.56	1816.48	0.30%	3.80%

Table I. Runtime overhead of repartitioning during Hessian construction when $M = 234$.

6. CONSTRUCTING THE PRECONDITIONER

Within each iteration k of the outer loop in Algorithm 1, a preconditioner $\boldsymbol{\mathcal{P}}$ is constructed based on the Hessian $\mathbf{H}$ computed as described above. Since the Hessian is block-tridiagonal, the block factorization procedure to construct $\boldsymbol{\mathcal{P}}$ can be formalized by considering the i^{th} block row of the block-tridiagonal system $\mathbf{H}x = b$:

$$L_i x_{i-1} + D_i x_i + U_i x_{i+1} = b_i \ , \ 1 < i < N, \tag{13}$$

where L_i, D_i and U_i denote the lower, main and upper diagonal blocks, respectively, in block row i of $\mathbf{H}$, and b_i is the i^{th} block of the right hand side vector.

6.1. Accelerated Thomas Algorithm

The unoptimized version of SIESTA used the Thomas algorithm [19] to construct $\boldsymbol{\mathcal{P}}$ as follows. From Eqn (13), the equation for x_i is used to eliminate x_i in the equation for x_{i+1} leading to the following recursion relation:

$$x_i = -\Delta_i(U_i x_{i+1} + \beta_i), \tag{14}$$

where $\Delta_i = (D_i - L_i\Delta_{i-1}U_{i-1})^{-1}$ and $\beta_i = b_i - L_i\beta_{i-1}$ with the starting values $\Delta_0 = \beta_0 = 0$. The recurrence is carried out until the boundary at $i = N$ at which point the boundary condition $U_N = 0$ is used to initiate the backward solving process using the above recursion relation to iterate backwards from $i = N$ to $i = 1$. Once Δ_i is determined and stored for a given matrix P, the relation for β_i can be iterated for multiple right hand side (RHS) vectors and the back substitution carried out to obtain the solution for each RHS without the need for further matrix inversions or matrix-matrix products. Originally, SIESTA used ScaLAPACK [2] to parallelize the matrix inversions, matrix-matrix and matrix-vector operations at the block level within the Thomas algorithm. This is a sub-optimal parallel approach as it does not take advantage of the block-tridiagonal structure of the matrices and, hence, does not scale with N.

6.2. Cyclic Reduction Based Parallel Solver

For greater scalability, a redesigned parallel solver called BCYCLIC [9], specialized for block-tridiagonal matrices, was integrated with SIESTA. It is based on a cyclic reduction algorithm in which block rows with even indices $i = 2k$ $(i \leq k \leq N/2)$ in Eqn (13) are eliminated in terms of odd indices using:

$$x_{2k} = \widetilde{b}_{2k} - \widetilde{L}_{2k}x_{2k-1} - \widetilde{U}_{2k}x_{2k+1} \tag{15}$$

where $\widetilde{b}_{2k} = D_{2k}^{-1}b_{2k}$, $\widetilde{L}_{2k} = D_{2k}^{-1}L_{2k}$, $\widetilde{U}_{2k} = D_{2k}^{-1}U_{2k}$. The boundary conditions on the block matrices are $L_1 = U_N = 0$. A similar equation for the odd indices $i = 2k - 1$ $(i \leq k \leq N/2)$ can

DOI: 10.1002/cpe

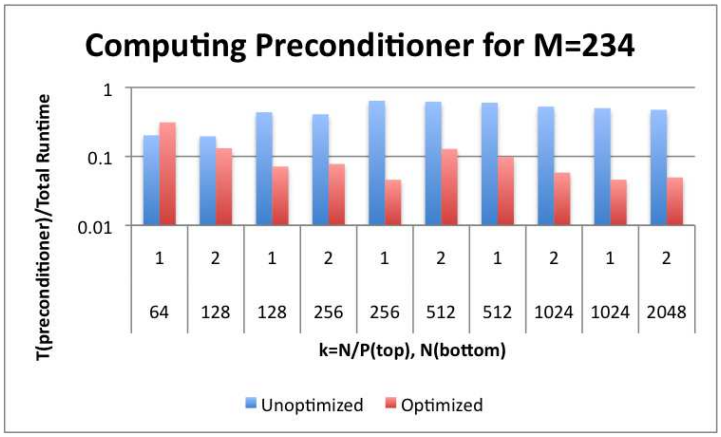


Figure 2. Performance gain in constructing the preconditioner $\mathcal{P}$ using a cyclic reduction-based algorithm. The times above include the overhead of remapping the columnar space, as described in the previous section. Note that the top and bottom labels of the horizontal axis refer to the granularity ($\frac{N}{P}$) and number of block rows (N), respectively.

be written and then Eqn (15) used to eliminate the even indexed terms. This yields:

$$\widetilde{L}_{2k-1}x_{2k-3} + \widetilde{D}_{2k-1}x_{2k-1} + \widetilde{U}_{2k-1}x_{2k+1} = \widetilde{b}_{2k-1} \tag{16}$$

where:

$$\begin{aligned}
\widetilde{D}_{2k-1} &= D_{2k-1} - L_{2k-1}\widetilde{U}_{2k-2} - U_{2k-1}\widetilde{L}_{2k}, \\
\widetilde{L}_{2k-1} &= -L_{2k-1}\widetilde{L}_{2k-2}, \\
\widetilde{U}_{2k} &= -U_{2k-1}\widetilde{U}_{2k}, \\
\widetilde{b}_{2k-1} &= b_{2k-1} - L_{2k-1}\widetilde{b}_{2k-2} - U_{2k}\widetilde{b}_{2k}.
\end{aligned}$$

Note that Eqn (16) is structurally the same as Eqn (13), with the number of new indices $2j' = 2k - 1$ and $2j' \pm 1 = 2k - 1 \pm 2$ now reduced by half. This step is recursively applied $n = \log_2 N$ times until only a single equation remains which is trivially solved for x_1 using the appropriate boundary conditions. This solution initiates a backward solve phase in which the recursion tree is traversed in the reverse direction. At each recursive step during this backward traversal, the even indexed unknowns are computed by substituting the now known odd-indexed values.

Performance gains delivered by BCYCLIC over the original ScaLAPACK based solver when constructing the preconditioner are shown in Fig. 2. Note the log-log scale. The time to construct $\mathcal{P}$, denoted by $T(preconditioner)$, shown in this figure includes the repartitioning overhead incurred in laying out the columnar space across processors in the manner expected by a cyclic reduction-based algorithm (see Fig. 1(b)). Clearly, the advantage of using a parallel solver customized for block-tridiagonal systems far surpasses the disadvantage of incurring the remapping overhead.

7. ACCELERATING GMRES

The GMRES (Generalized Minimal Residual) method [16] is used in the inner loop in Algorithm 1 to refine the initial solution guess $x = \mathcal{P}(\mathbf{b})$, where $\mathbf{b} = \mathcal{F}$. Note that the execution flow of SIESTA is such that the Hessian matrix $\mathbf{H}$ (or the preconditioner $\mathcal{P}$) is *computed* once for each iteration of the outer loop but *used* in every iteration of the inner loop. SIESTA uses the CERFACS [5] "reverse communication" implementation of GMRES with the right-hand preconditioner option.

DOI: 10.1002/cpe

7.1. Improving the Efficiency of Iterative Solutions

In its original parallel version, after constructing a preconditioner, SIESTA computed a solution of $\boldsymbol{\xi}^k \approx \boldsymbol{\mathcal{P}}^k \cdot \mathbf{F}^k$ in each iteration of the inner loop by using the LU decomposition method for general distributed matrices. It did so by making calls to PDGETRF, PDGETRS and DGSUM2D routines in ScaLAPACK. Unlike the cyclic reduction-based algorithm implemented in BCYCLIC, a ScaLAPACK based general approach is inefficient as it does not take advantage of the special block diagonal structure of the preconditioner.

BCYCLIC was implemented as a two-phase algorithm to best suit the execution flow of SIESTA. In the first phase, referred to as the *ForwardSolve* phase, BCYCLIC constructs the preconditioner $\boldsymbol{\mathcal{P}}$ as described in the previous section. In the second phase, referred to as the *BackwardSolve* phase, BCYCLIC explicitly computes a solution of the block-tridiagonal system for a given right hand side in parallel. The number of *BackwardSolve* phases typically exceeds the number of *ForwardSolve* phases by one to two orders of magnitude due to the iterative nature of the GMRES procedure. See Table II for an example.

P	N	ForwardSolves	BackwardSolves
64	64	6	348
64	128	6	352
128	128	6	352
128	256	6	363
256	256	6	363
256	512	7	472
512	512	7	472
512	1024	11	1099
1024	1024	11	1099
1024	2048	11	1123

Table II. Number of iterations in the outer and inner loops of Algorithm 1 when $M = 234$.

At the end of the *ForwardSolve* phase, a local portion of x is available at processor 0. This value is then propagated to processor $P/2 - 1$ which in turn uses it for back substitution to evaluate its local portion of x. The newly computed values of x are then similarly propagated by traversing the recursion tree in the backward direction from the bottom-most level to the top-most level. Once the back substitution traverses $\lg P$ levels from the leaf level of the recursion tree, the algorithm enters its final compute intensive opposite to that of the first $\lg(\frac{N}{P})$ steps for the *ForwardSolve* phase. All the while, new values are computed at each level by back-substituting with values computed in the previous level. Communication of matrices is required only for the boundary blocks. Finally, when the top level is reached, the solution of the system of equations is complete. Since no matrix inversion or matrix-matrix multiplications are involved in the *BackwardSolve* phase, this phase can be shown to be $O(M)$ faster than the *ForwardSolve* phase. Performance advantage over the original ScaLAPACK based implementation of the backward solve iterations is shown in Fig. 3 for the example shown in Table II.

7.2. Improving the Efficiency of Krylov Space Generation

An integral component of the GMRES iterative procedure that computes the solution of $\mathbf{H} \cdot \boldsymbol{\xi} = \mathbf{F}$ is the generation of the n^{th} Krylov subspace, K_n. The n^{th} Krylov subspace for this problem is

$$K_n = \text{ span } \left\{\boldsymbol{\xi}, \mathbf{H}\boldsymbol{\xi}, \mathbf{H}^2\boldsymbol{\xi}, \cdots, \mathbf{H}^{n-1}\boldsymbol{\xi}\right\}$$

In its original version, SIESTA reused two subroutines to generate the Krylov subspace. It used an inverse Fourier subroutine to convert the displacement vector to position space. The resulting displacement vector $\boldsymbol{\chi}$, in position space, was used as an input to a subroutine that directly computed $f(\boldsymbol{\chi})$, where $f(y) = \mathbf{H}y$, which was then transformed back into the Fourier space using another call

DOI: 10.1002/cpe

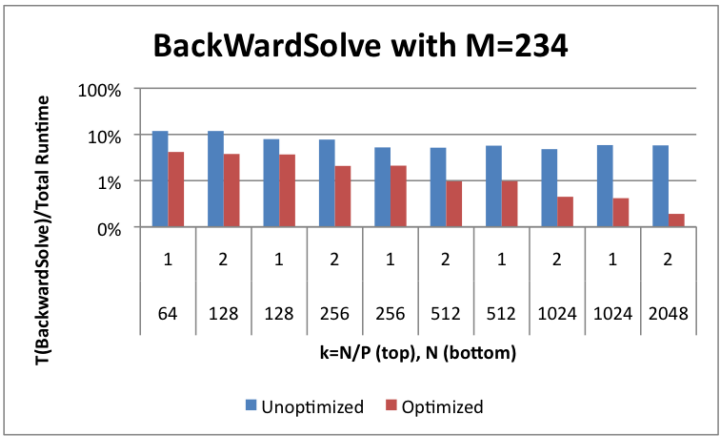


Figure 3. Performance gain from the BackwardSolve phase of the cyclic reduction algorithm. The top and bottom labels of the horizontal axis refer to the granularity ($\frac{N}{P}$) and number of block rows (N), respectively.

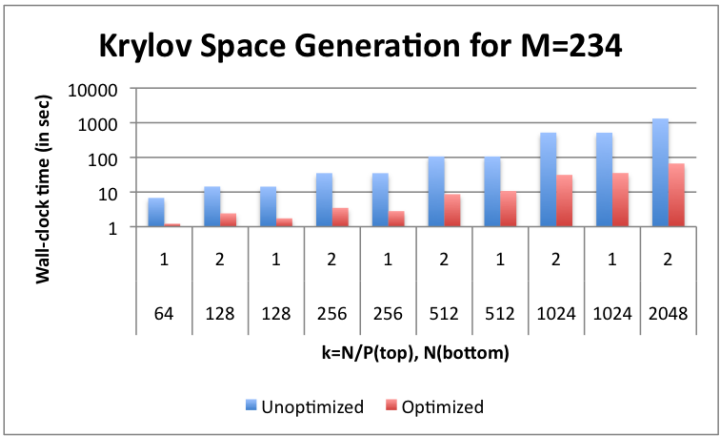


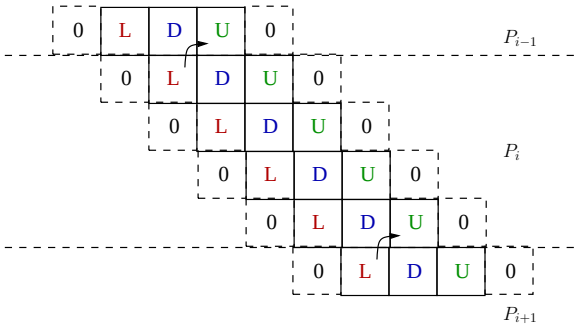
Figure 4. Performance gain from optimizing the Krylov subspace generation procedure. The top and bottom labels of the horizontal axis refer to the granularity ($\frac{N}{P}$) and number of block rows (N), respectively.

to the Fourier routine. This pseudo-spectral technique was repeated to compute each member of the set $\{\boldsymbol{\xi}, \mathbf{H}\boldsymbol{\xi}, \mathbf{H}^2\boldsymbol{\xi}, \cdots, \mathbf{H}^{n-1}\boldsymbol{\xi}\}$. Since $\mathbf{H}$ is a sparse matrix containing a linear (in the problem size NM) number of non-zero elements, it was carried out serially on each processor to avoid the overhead of inter-processor communications and data movement. However, computation costs begin to grow with increasing problem sizes, quickly degrading SIESTA's performance on large numbers of processors used for larger problem sizes.

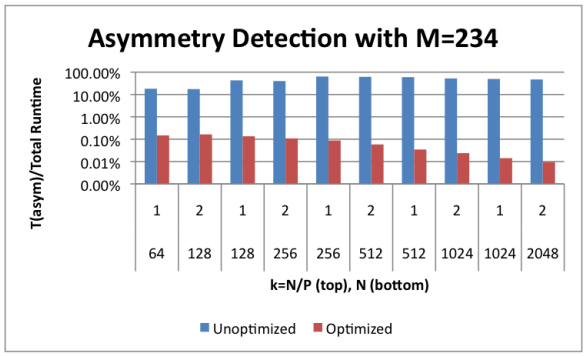
A key advantage of the repartitioning (see Section 5) carried out before constructing the preconditioner, is that it correctly partitions the columns of $\boldsymbol{\mathcal{P}}$ across processors for efficient use by the GMRES routine. From Eqn (10), it is clear that, to compute $\mathbf{H}$ from $\boldsymbol{\mathcal{P}}$, only the added perturbation to the diagonal elements need to be subtracted out. In this optimized version of SIESTA, the diagonal perturbation is stored during the Hessian computation phase and subtracted out in the GMRES routine to regain the distributed form of the Hessian $\mathbf{H}$. Based on this observation, a highly efficient parallel matrix-vector subroutine was implemented to generate the n^{th} Krylov subspace with almost no additional inter-processor data movement overhead.

Performance gains from the aforementioned optimized Krylov subspace generation is shown in Fig. 4. The advantage of the new approach is clear from the order of magnitude runtime improvements.

DOI: 10.1002/cpe



(a)



(b)

Figure 5. (a) The local portion of the asymmetry index can be computed without any additional communication once the lower block diagonal matrix of the first block row of the next rank is made available. (b) Performance gain in Hessian asymmetry detection due to the optimized kernel. Note the log scale along the vertical axis. The top and bottom labels of the horizontal axis refer to the granularity ($\frac{N}{P}$) and number of block rows (N), respectively.

8. HESSIAN ASYMMETRY DETECTION

As the plasma state approaches equilibrium in the outer loop of Algorithm 1, the Hessian matrix approaches self-adjointness. SIESTA computes a measure of the self-adjointness of $\mathbf{H}$ every time it is generated, to monitor that the simulation has not strayed from its path to equilibrium. This is accomplished by computing an asymmetry index defined by:

$$\delta = \sqrt{||\mathbf{H} - \mathbf{H}^T||^2 / ||\mathbf{H}||^2}, \tag{17}$$

where $||\mathbf{B}|| = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |b|_{ij}^2}$ is the Frobenius norm of the $m \times n$ matrix B. In the original version of SIESTA, this asymmetry index was computed using calls to the ScaLAPACK routines PDLANGE and PDGEADD. The routine PDLANGE returns the Frobenius norm of a distributed matrix while the routine PDGEADD adds (or subtracts) two distributed matrices, both of which use a set of Basic Linear Algebra Communication Subprograms (BLACS) for communication.

To gainfully exploit the special block-tridiagonal structure of the Hessian, we optimized SIESTA to take advantage of the already distributed Hessian available after step 3 in Algorithm 1. Recall that after repartitioning the columnar space, as described in Section 5, the N block rows of the matrix $\mathbf{H}$ are already partitioned across P processors such that the i^{th} set of $\frac{N}{P}$ block rows are owned by rank i. Given this partitioning scheme of the Hessian matrix, it is *not* necessary to compute $\mathbf{H}^T$ explicitly. The local part of $\mathbf{H}^T$ can be generated by simply assigning $D_i \leftarrow D_i^T$, $L_i \leftarrow U_{i-1}^T$ and $U_i \leftarrow L_{i+1}^T$ for all $1 \leq i \leq \frac{N}{P}$. Thus, if each process with rank $i > 0$ (the first rank is rank 0)

DOI: 10.1002/cpe

sends the lower diagonal block matrix of its first block row to the process with rank $i-1$, then the local portion of both the numerator and the denominator in Eqn (17) can be computed without any further communication. See Fig. 5(a). The partial sums can then be aggregated on each processor using an *AllGather* operation to make the global value of the asymmetry index available in each processor. Thus, an extremely scalable and efficient parallel kernel results as an additional advantage of repartitioning the columnar space for the cyclic reduction algorithm.

Runtime overhead of Hessian asymmetry detection in the original code is not noticeable on smaller processor counts but it quickly surpasses other computational costs as the number of cores increases to thousands. This is because a ScaLAPACK-based implementation, although simpler in practice, does not take advantage of the special structure of the Hessian or the simplifying properties of the asymmetry index computation in Eqn (17). This optimization delivers performance gains by multiple orders of magnitude for this kernel, as demonstrated in Fig. 5(b) in which the percentage of time spent in asymmetry detection, $T(asym)$, is compared to the total runtime of SIESTA.

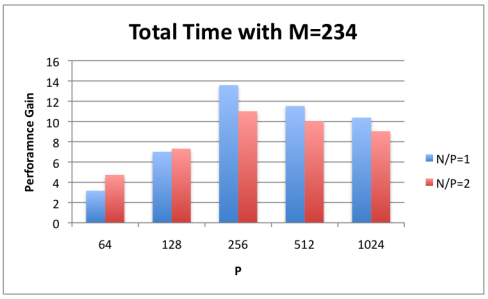
9. OVERALL PERFORMANCE AND RESULTS

A series of optimizations in key kernels of SIESTA was described in the previous sections. The net performance improvement of SIESTA can be measured (and demonstrated) using two common parallel performance metrics, namely, *weak (or iso-granular)* scaling and *strong* scaling [6]. In weak scaling, total run times on varying number of processors while maintaining a fixed ratio of the problem size to processor counts are compared. Strong scaling, on the other hand, compares runtimes in which the problem size is held constant while varying the processor count. Performance gain (see Section 2 for the definition of performance gain) is used to compare weak and strong scaling of the unoptimized (T_u) and optimized (T_o) runtimes.

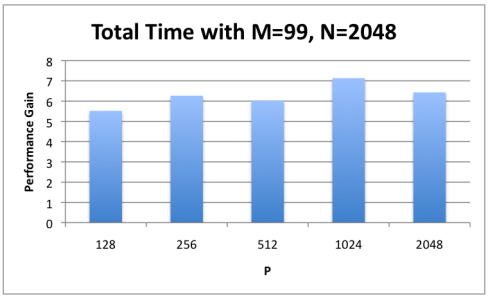
P	N	Unoptimized	Optimized	Gain
512	512	9775.39	969.04	10.09
512	1024	$>$ 24000.00	2395.90	$>$ 10.02
1024	1024	25557.35	3213.71	7.95
1024	2048	$>$ 30600.00	3874.86	$>$ 7.90

Table III. Runtimes (in sec) with $M = 459$. Runtimes which exceeded the alloted machine times are indicated accordingly with lower bounds on the resulting gains.

Weak scaling performance gains are shown in Fig. 6(a) with block size $M = 234$ for two different granularities, namely, $\frac{N}{P} = 1$ and $\frac{N}{P} = 2$. When $\frac{N}{P} = 1$ and $M = 234$, the optimized version of SIESTA runs three times faster than the unoptimized version on 64 processors. This performance gain increases as the number of processors is increased while keeping the granularity constant.



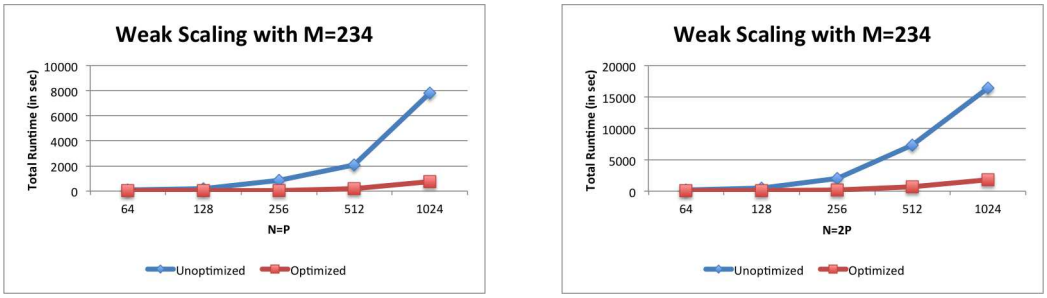
(a) Weak scaling with $M = 234$.



(b) Strong scaling with $M = 99$ and $N = 2048$.

Figure 6. Weak (iso-granular) and strong scaling of optimized SIESTA. Performance gain is defined as T_u/T_o for both weak and strong scaling executions.

DOI: 10.1002/cpe



(a) Weak scaling with $M = 234$ and $N = P$.

(b) Weak scaling with $M = 234$ and $N = 2P$.

Figure 7. Weak (iso-granular) scaling of unoptimized and optimized SIESTA.

When the number of processors increases to 256, the optimized version of SIESTA runs *over thirteen times faster* than the unoptimized version. The performance gain continues to remain an order of magnitude higher even as the number of processors is increased to 1024. Even with larger granularity ($\frac{N}{P} = 2$ is shown in Fig. 6(a)), the performance advantage increases to over ten times as the number of processors increases from 64 to 256. Performance gains begin to diminish with increasing number of processors, as can be expected. There is a point of diminishing return, which depends on both the value of the block size and the number of row blocks.

Strong scaling performance gains are shown in Fig. 6(b) with block size $M = 99$ on up to $P = 2048$. The optimized version of SIESTA runs 5 to 7 times faster than the unoptimized version when the number of processors vary by over an order of magnitude between 128 to 2048. As can be inferred from Fig. 6(a), this performance gain will only be more pronounced as the block size M increases. For example, when $M = 459$, the optimized version of SIESTA runs over 8 times faster compared to the unoptimized version with $N = 2048$ and $P = 1024$ (see Table III) but 7 times faster when $M = 99$ (see Fig. 6(b)).

Weak scaling of the total runtimes is presented in Fig. 7 for $M = 234$. Fig. 7(a) and Fig. 7(b) differ only in the granularity of the problem, as indicated. In both cases, the optimized version of SIESTA clearly exhibits superior weak scaling. Strong scaling of the total runtimes, presented in Fig. 8 for $M = 99$ and $N = 2048$, shows that as P approaches N, optimized SIESTA scales better than the unoptimized version in the strong sense, despite a decreasing self-relative strong speedup of both versions. The speedup decreases because of incremental optimization, benefits of which are not uniformly manifested in the parallel execution flow of the entire code. That this happens despite excellent strong scaling of the optimized versions of the computationally intensive components, as presented in the earlier sections, is an indication that performance bottlenecks have shifted from the original time-consuming linear algebra routines to other portions of the original code which still retain vestiges of sequentiality. Elimination of these bottlenecks is a focus of the next phase of optimizations.

The most important outcome of this scaling effort is providing SIESTA with the capability to execute runs that are almost an order of magnitude faster than previously possible. For example, the runtime of a configuration of current interest with $M = 459$ and $N = 1024$ is reduced from over 25,000 seconds down to less than 3,300 seconds (see Table III). Even larger gains are expected for larger M and/or N.

Application of the accelerated SIESTA code to a low pressure stellarator equilibrium plasma is shown in Fig. 9 and Fig. 10 for two M values (the variation with the number of radial points is small for this case). In Fig. 9, the pressure contours are plotted in flux-coordinate space for two toroidal cross sections $v = 0$ and $v = \pi/N_{fp}$, where $N_{fp} = 5$ is the number of field periods for this plasma. The three different color levels shown correspond to significant changes in the pressure. The most significant change occurs in the island contours inside the dominant resonance around radius $s = 0.55$ where $\iota/N_{fp} = 1/6$ ($m = 6$ islands). As M (the mode number product) increases, the amplitude of the variation of the pressure inside the islands is reduced but an oscillatory component

DOI: 10.1002/cpe

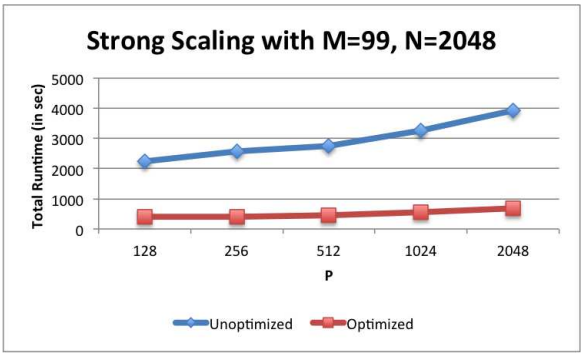


Figure 8. Strong scaling of SIESTA.

develops. This can be clearly seen in Fig. 10 where the local pressure as a function of radius is plotted for vertical cuts (in Fig. 9) at $u = 0$. In the $v = 0$ plane, that cut goes through the "O" point (widest point) of the island, while for $v = \pi/N_{fp}$, the cut is through the "X" point and shows very little departure of the pressure from its original (no island) profile. Thus, the poloidal variation of the island couples with the radial variation to give a slow oscillatory convergence to an eventually flattened pressure profile inside the island as M increases. With $N = P = 256$, the unoptimized and optimized run times for the experiments corresponding to Fig. 9(a) and Fig. 9(b), are 2,852 seconds and 241 seconds, respectively. For the experiments with $M = 528$, corresponding to Fig. 9(c) and Fig. 9(d), the unoptimized and optimized run times are 4,602 seconds and 540 seconds with $N = P = 256$. The performance gain due to the optimizations is almost an order of magnitude in both cases.

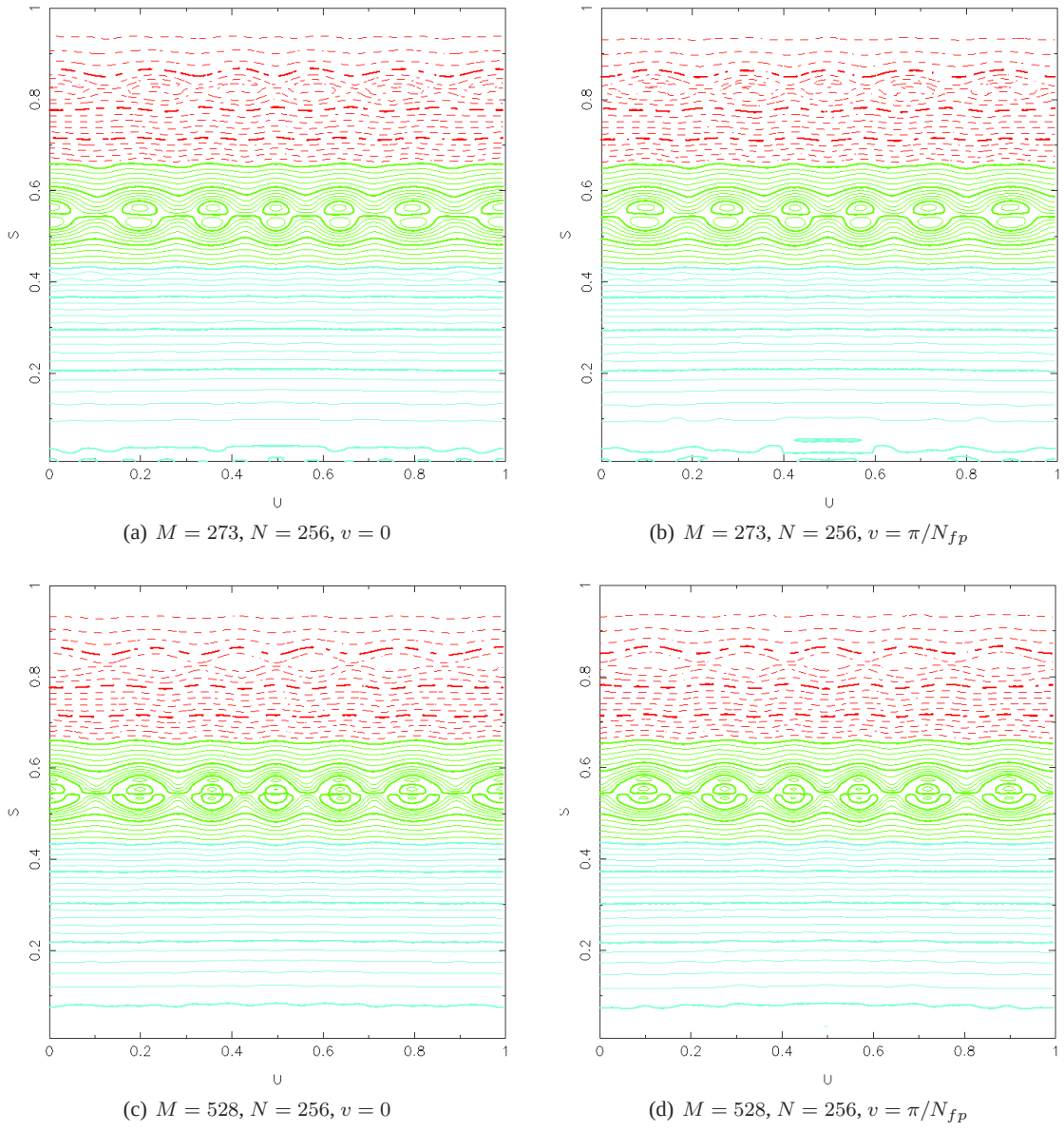
For every experiment reported here, the results obtained from the optimized version of SIESTA were fully verified for correctness. The original unoptimized version was used as the baseline. For each experiment, irrespective of whether the unoptimized or the optimized versions was used, the force residual was at the level of 10^{-19}, ensuring convergence of the iterations.

10. CONCLUSION AND FUTURE WORK

Developed over several years, SIESTA is a computationally complex magnetohydrodynamics equilibrium code. To extend its capabilities to meet the demands of new plasma applications (such as those for ITER), the code needs to scale to thousands of processors. Rewriting a sophisticated code such as SIESTA to be able to scale to that many processors is an extremely time-consuming and expensive enterprise. Instead, taking an alternative approach, this paper describes a sequence of optimizations which were implemented to deliver a dramatic improvement in SIESTA's runtime. Starting with the original small-scale parallel implementation, runtime profiling of SIESTA was used to identify major bottlenecks. Each of these bottlenecks was relieved through a combination of efficient algorithmic redesign and careful re-assessment of pre-existing computational strategies. Both were possible due to a new parallel domain decomposition scheme that allowed multiple elegant optimizations based on the special structure of the underlying system of equations. Together, these non-intrusive optimizations improved SIESTA's runtime by over ten-fold without any significant modification to the original code while scaling it to 2,048 processors.

Further improvements are indeed possible. Though the values of N (which limit the maximum number of processors P that can be used to maintain a minimum of unit granularity of the block row to processor mapping) used in this paper are adequate for current plasma applications, significantly higher values of N as well as M are of interest in the evaluation of next generation plasma applications (in which, for example, the corresponding N and/or M are as large as 10,000 or more). In fact, core counts larger than N can be used if the extra cores are deployed to parallelize block level linear algebra operations, particularly for larger block size M. This translates to the potential of

DOI: 10.1002/cpe



(a) $M = 273$, $N = 256$, $v = 0$

(b) $M = 273$, $N = 256$, $v = \pi/N_{fp}$

(c) $M = 528$, $N = 256$, $v = 0$

(d) $M = 528$, $N = 256$, $v = \pi/N_{fp}$

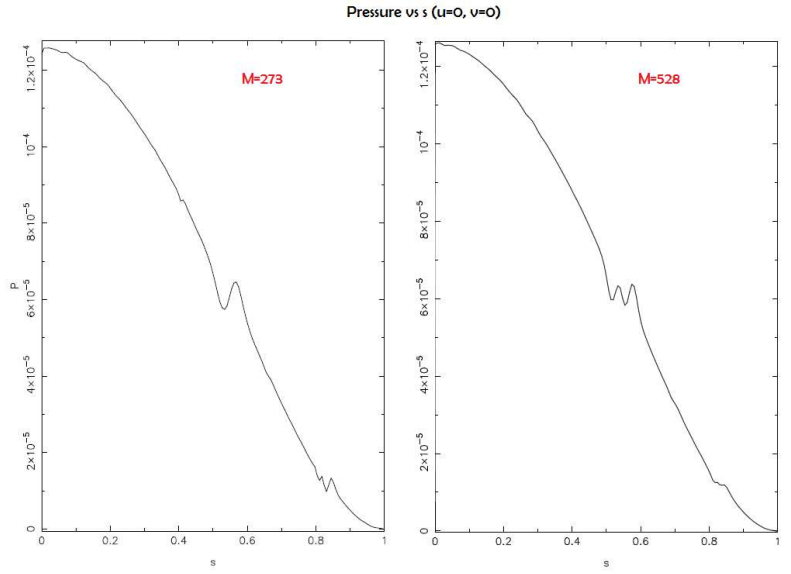
Figure 9. Pressure contours for a variety of poloidal and toroidal resolutions. The subscript N_{fp} refers to the number of field periods of the plasma.

testing our non-intrusive optimizations on much larger number of processor cores (e.g., $P = 10,000$ for simple optimizations, or even greater values of P and larger values of M). Evaluation of SIESTA at these scales, with particular emphasis on multithreading-based optimizations to exploit multi-core systems, is of interest in the near future.

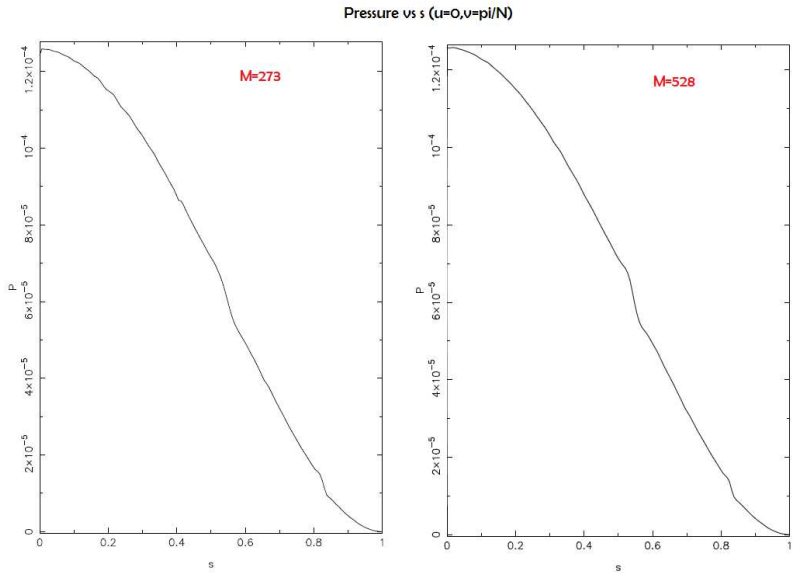
ACKNOWLEDGEMENTS

This paper has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the U.S. Department of Energy. Accordingly, the United States Government retains and the publisher,

DOI: 10.1002/cpe



(a)



(b)

Figure 10. Radial variation of the pressure profile at $u = 0$ for (a) $v = 0$, through the "O" point of the dominant $m = 6$ island and (b) $v = \pi/N_{fp}$, through the "X" point.

REFERENCES

1. O. Betancourt. BETAS, A Spectral Code for Three-dimensional Magnetohydrodynamic Equilibrium and Nonlinear Stability Calculations. *Communications of Pure and Applied Mathematics*, 41(5):551, 1988.

DOI: 10.1002/cpe

2. L. S. Blackford, J. Choi, A. Cleary, E. D'Azeuedo, J. Demmel, I. Dhillon, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK User's Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.
3. L. Chacon. An Optimal, Parallel, Fully Implicit Newton-Krylov Solver for Three-dimensional Viscoresistive Magnetohydrodynamics. *Physics of Plasmas*, 15:056103, 2008.
4. R. Chodura and A. Schluter. A 3D code for MHD Equilibrium and Stability. *Journal of Computational Physics*, 41:68, 1981.
5. V. Fraysse, L. Giraud, S. Gratton, and J. Langou. A Set of GMRES Routines for Real and Complex Arithmetics on High Performance Computers. *ACM Transactions on Mathematical Software*, 31(2):228–238, 2005.
6. A. Grama, G. Karypis, V. Kumar, and A. Gupta. *Introduction to Parallel Computing*. Addison Wesley, 2 edition edition, 2003.
7. J. D. Hanson, S. P. Hirshman, S. F. Knowlton, L. L. Lao, E. A. Lazarus, and J. M. Shields. V3FIT: A Code for Three-dimensional Equilibrium Reconstruction. *Nuclear Fusion*, 49:075031, 2009.
8. T. Hayashi and et al. . In *Plasma Phys. Control. Fusion*, page 29, 1993.
9. S. P. Hirshman, K. S. Perumalla, V. E. Lynch, and R. Sanchez. BCYCLIC: A Parallel Block Tri-diagonal Matrix Cyclic Solver. *Journal of Computational Physics*, 229:6392–6404, 2010.
10. S. P. Hirshman, R. Sanchez, and C. R. Cook. SIESTA: A Scalable Iterative Equlibrium Solver for Toroidal Applications. *Physics of Plasmas*, 18:062504, 2011.
11. S. P. Hirshman and J. C. Whitson. Steepest-descent Moment Method for Three-dimensional Magnetohydrodynamic Equilibria. *Physics of Fluids*, 26(12):3553, 1983.
12. C. T. Kelley and D. E. Keyes. Convergence Analysis of Pseudo-Transient Continuation. *SIAM Journal of Scientific Computing*, 35(2):508–523, 1998.
13. L. L. Lao, H. S. John, R. Stambaugh, A. Kellman, and W. Pfeiffer. Reconstruction of Current Profile Parameters and Plasma Shapes in Tokamaks. *Nuclear Fusion*, 25:1611, 1985.
14. W. Park, E. V. Belova, G. Y. Fu, X. Z. Tang, H. R. Strauss, and L. E. Sugiyama. Plasma Simulation Studies using Multilevel Physics Models. *Physics of Plasmas*, 6:1796, 1999.
15. A. H. Reiman and H. S. Greenside. Calculation of Three-dimensional MHD Equilibria with Islands and Stochastic Regions. *Computational Physics Communications*, 43:157, 1986.
16. Y. Saad and M. H. Shcultz. GMRES: A Generalized Minimal Residual Algorithm for Solving Non-symmetric Linear Systems. *SIAM J. Sc. Statist. Comput.*, 7:856–869, 1986.
17. C. R. Sovenic, A. H. Glasser, T. A. Gianakon, D. C. Barnes, R. A. Nebel, S. E. Kruger, D. D. Schnack, S. J. Plimpton, A. Tarditi, M. S. Chu, and the NIMROD Team. Nonlinear Magnetohydrodynamics Simulation using high-order Finite Elements. *Journal of Computational Physics*, 195:355, 2004.
18. D. Spong, S. Hirshman, L. Berry, J. Lyon, R. Fowler, D. Strickler, M. Cole, B. Nelson, D. Williamson, A. Ware, D. Alban, R. Sanchez, G. Fu, D. Monticello, W. Miner, and P. Valanju. Physics Issues of Compact Drift Optimized Stellarators. *Nuclear Fusion*, 41:771, 2001.
19. L. H. Thomas. Elliptic Problems In Linear Difference Equations Over A Network . *Watson Sci. Comput. Lab. Rep., Columbia University*, 1949.

DOI: 10.1002/cpe