The Parallel Waveform IBiCG Technique for Transient Simulation of Semiconductor Devices

Laurence Tianruo Yang
Department of Computer Science, St. Francis Xavier University, P.O. Box 5000, Antigonish, B2G 2W5, NS, Canada

Abstract

In this paper, we mainly study the parallelization aspects of the accelerated waveform relaxation algorithms for the transient simulation of semiconductor devices on parallel distributed memory computers since these methods are competitive with standard point-wise methods on serial machines, but are significantly faster on parallel computers. Here we are proposing an improved parallel version of the Biconjugate gradient method (IBiCG) combining elements of numerical stability and parallel algorithm design, for solving the resulting sequence of time-varying sparse linear differential-algebraic initial-value problems (IVP) arising at each linearization step with waveform Newton. The algorithm is derived such that all inner products, matrix-vector multiplications and vector updates of a single iteration step are independent and communication time required for inner product can be overlapped efficiently with computation time of vector updates. Therefore, the cost of global communication on parallel distributed memory computers can be significantly reduced. The resulting IBiCG algorithm maintains the favorable properties of the Lanczos process while not increasing computational costs. Experimental results carried out on Parsytec massively parallel systems with regards to the comparison with other accelerated approaches such as convolution SOR and waveform GMRES techniques on waveform relaxation algorithm and pointwise methods are described as well.

1 Introduction

The computational expense and growing importance of performing semiconductor device transient simulation, along with the increasing availability of parallel computers, suggest that parallel algorithms should be developed and used for these simulation problems. Although several approaches have been shown that SIMD type machines are effective for device transient simulation [13, 27], it is more important to develop the parallel algorithms on MIMD type machines, such as distributed memory computers, since these are becoming increasingly more popular and cost-effective.

The most efficient serial algorithm for device transient simulation is the pointwise Newton-GMRES algorithm. In this algorithm, block-Jacobi preconditioned GMRES [24] is used to solve the linear systems arising at each Newton iteration of each timestep of an implicit integration formula applied to (1). The pointwise Newton-GMRES method requires communication due to inner products for each GMRES iteration. On distributed memory computers, the matrices and vectors are distributed over the processors, so that even when the matrix operations can be implemented efficiently by parallel operations, we cannot avoid the global communication required for inner product computations. Usually the performance of this method on this kind of architecture is always limited because of the global communication required for the inner products while the matrix-vector multiplication only requires local communication, i.e. communication with only a few processors. In a word, these global communication costs become relatively more and more important when the number of the parallel processors is increased and thus they have the potential to affect the scalability of the algorithm in a negative way [8, 9]. Although many of these communication operations can be overlapped with local computation [1, 9, 29, 30, 31, 32], the communication latency is still so large that it cannot be hidden by the relatively small amount of computation done at a single timestep. To achieve high performance on parallel distributed memory computers, it is critical that a numerical algorithm avoids frequent global communications between processors due to inner products on distributed memory computers. The waveform relaxation (WR) approach to solve time-dependent problems is such a method, because in parallel waveform relaxation, iterates are communicated between processors only after having been computed over a time in-
terval [15, 20, 28].

As with any iterative schemes, efficiency of the waveform relaxation depends on rapid convergence, and there are several approaches to accelerated waveform relaxations, including multigrid [16], Krylov-subspace [17, 18], and convolution SOR techniques [21, 22]. In [23], they mainly focus on one method, namely waveform GMRES (WGMRES), an extension of the generalized minimum residual algorithm (GMRES) [24], of the Krylov-subspace methods to accelerate the convergence of waveform relaxation algorithms. The GMRES method yields an optimal reduction of the residual norm for a given number of iterations. The method requires the computation of an orthogonal basis of the Krylov subspace. For this, a number of inner products, increasing with the iteration count, needs to be computed. On massively parallel computers, the basic time-consuming computational kernels of the iterative schemes such as GMRES are usually: inner products, vector updates, and matrix-vector products. In many situations, especially when matrix operations are well-structured, these operations are suitable for implementation on vector and share memory parallel computers [10]. But for parallel distributed memory machines, although they have successfully avoided by using waveform relaxation method accelerated by waveform GMRES technique instead of pointwise Newton-GMRES, they still can not avoid the global communication required for inner product during waveform GMRES technique. The performance of their approach on this kind of architecture is still limited.

In this paper, we will mainly focus on another Krylov subspace method, namely, the Biconjugate gradient algorithm [14, 11], for large and sparse linear systems with unsymmetric coefficient matrices.

Recently, Bücke et al. [6, 7] propose a new modified parallel version of the BiConjugate Gradient (BiCG) method. The algorithm is derived that both generated sequences of Lanczos vectors are scalable and is reorganized without changing the numerical stability so that there is only a single global synchronization point per iteration. Therefore, the cost of global communication on parallel distributed memory computers can be reduced. Based on their similar ideas, we propose a new improved two-term recurrences Lanczos process without look-ahead as the underlying process for the new Improved BiConjugate Gradient (IBiCG) method. The algorithm is reorganized without changing the numerical stability so that all inner products, matrix-vector multiplications and vector updates of a single iteration step are independent, and subsequently communication time required for inner product can be overlapped efficiently with computation time of vector updates.

Therefore, the cost of global communication on parallel distributed memory computers can be significantly reduced. The resulting IBiCG algorithm maintains the favorable properties of the Lanczos process while not increasing computational costs. Based on the proposed method, several experimental results are carried out on Pansytec massively parallel systems in order to compare with other accelerated approaches, such as convolution SOR and waveform GMRES techniques, on waveform relaxation algorithm and pointwise methods.

The paper is organized as follows. In section 2, we will describe shortly some background about device transient simulation. Then we will present briefly the accelerated waveform relaxation method with several acceleration techniques such as convolution SOR and waveform GMRES methods in section 3. Then the corresponding proposed efficient parallel waveform IBiCG will also be described in section 4. In section 5, the parallel implementation details including data distribution and communication scheme are presented. Finally in section 6, the comparison of parallel performance of proposed method with other accelerated approaches such as convolution SOR and waveform GMRES techniques on waveform relaxation algorithm and pointwise methods are described as well.

2 Device Transient Simulation

Charge transport within a semiconductor device is assumed to be governed by the poission equation, and the electron and hole continuity equations [2, 26]. Given a two-dimensional rectangular discretization mesh, the device equation system is typically discretized with finite-difference formula applied to the Poission equation, and an exponentially fit finite-difference formula applied to the continuity equations (the Scharfetter-Gummel method [26]). On an N-node mesh, this spatial discretization yields a sparse differential-algebraic initial-value problem (IVP) consisting of 3N equations in 3N unknowns, denoted by

\[
\begin{align*}
F_1(u(t), n(t), p(t)) &= 0, \\
\frac{d}{dt} p(t) + F_2(u(t), n(t), p(t)) &= 0, \\
\frac{d}{dt} p(t) + F_3(u(t), n(t), p(t)) &= 0,
\end{align*}
\]

where

\[
F_i(u(0), n(0), p(0)) = 0, \quad i = 1, 2, 3
\]

and \( t \in [0, T] \), and \( u(t), n(t), p(t) \) are vectors of normalized potential, electron concentration, and hole concentration, respectively. Here \( F_1, F_2 \) and \( F_3 \) are specified component-wise as:

\[
F_{i1}(u_i, n_i, p_i, u_j) =
\]
\[
\frac{kT}{q} \sum_j d_{ij} \varepsilon_{ij} (u_i - u_j) - qA_i (n_i - n_i + N_{D_i} - N_{A_i}),
\]
\[
F_2(u_i, n_i, u_j, n_j) = \frac{kT}{qA_i} \sum_j d_{ij} \mu_{s_{ij}} [n_i B(u_i - u_j) - n_j B(u_j - u_i)] + R_i,
\]
\[
F_3(u_i, n_i, u_j, n_j, p_j) = \frac{kT}{qA_i} \sum_j d_{ij} \mu_{s_{ij}} [n_i B(u_i - u_j) - p_j B(u_i - u_j)] + R_i.
\]

The sums above are taken over the silicon nodes \( j \) adjacent to node \( i \). For each node \( j \) adjacent to node \( i \), \( L_{ij} \) is the distance from node \( i \) to node \( j \), \( d_{ij} \) is the length of the side of the Voronoi box that encloses node \( i \) and bisects the edge between node \( i \) and \( j \), and \( \varepsilon_{ij}, \mu_{s_{ij}} \) and \( \mu_{p_{ij}} \) are the dielectric permittivity, electron and hole mobility, respectively, on the edge between nodes \( i \) and \( j \). The Bernoulli function, \( B(x) = x/(e^x - 1) \), is used to exponentially fit potential variation to electron and hole concentration variations, and efficiently upwinds the current equations.

3 Accelerated Waveform Methods

Usually we can solve the differential-algebraic equation (DAE) system (1) by discretizing the system in time with a low-order implicit integration method. The standard approach used to solve the resulting sequence of non-linear algebraic systems is to use some variants of Newton’s method and/or relaxation [2, 19]. As we mentioned above, this approach is very limited on parallel distributed memory computers with a high communication costs.

A more effective approach is to solve (1) on parallel distributed memory computers is to use waveform relaxation (WR) to decompose the DAE system into subsystem before time discretization [15]. The WR algorithm has several computational advantages such as avoiding factoring large and sparse matrices, exploiting multi-rate behavior, using different timesteps to resolve different solution components, and finally well suitable for parallel implementations because of a low global communication/computation ratio. However we apply to solve the device equation system (1), the WR algorithm converge slowly unless acceleration techniques, such as convolution SOR, or waveform GMRES, are used.

Convolusion SOR (CSOR) is a generalized waveform extension of the well-known successive overrelaxation (SOR) method to accelerate the convergence of relaxation methods for solving linear system of equations [33]. The CSOR method can be described shortly as follows. We mainly consider the problem of numerically solving the linear initial-value problem:

\[
\frac{d}{dt} x(t) + A x(t) = b(t), \quad x(0) = x_0,
\]

where \( A \in \mathbb{R}^{n \times n}, b(t) \in \mathbb{R}^n \) is given for all \( t \in [0, T] \), and \( x(t) \in \mathbb{R}^n \) is to be computed. Detailed descriptions about CSOR are presented in [23]. In generally, in iteration \( k + 1 \), each waveform \( x^{k+1}_n \) is computed as in ordinary Gauss-Seidel WR, and then is moved slightly in iteration direction by convolution with CSOR parameter, namely function \( \omega(t) \). The convolution allows the CSOR method to correctly account for the frequency-dependence of the spectrum of the Gauss-Jacobi WR operator by, in effect, using a different SOR parameter for each frequency.

Our attention is focus on another efficient acceleration technique, namely waveform BiCG for solving (2). With the splitting like \( A = M - N \), the basic iteration can be expressed as follows:

\[
\frac{d}{dt} x^{k+1}(t) + M x^{k+1}(t) = N x^k(t) + b(t).
\]

The solution \( x \) to (2) is thus a fixed point of the WR algorithm [18, 21], satisfying the integral operator equation

\[
(I - K)x = \psi. \tag{3}
\]

Here we follow the notations described in [23] where \( K \) is defined by

\[
(K x)(t) = \int_0^t e^{(s-t) M} N x(s) ds,
\]

and

\[
\psi(t) = e^{-t M} x(0) + \int_0^t e^{(s-t) M} b(s) ds.
\]

Krylov subspace methods can be used to accelerate the convergence of WR, but as \( K \) is not self-adjoint, a variant suitable for non-self-adjoint operators must be used [18, 23]. In [23], Reichelt et al. have used waveform GMRES (WGRMES), an extension of the generalized minimum residual algorithm to successfully accelerate the convergence on MIMD parallel computers such as Intel iPSC/860. However, on distributed memory computers, the performance of their approach on this kind of architecture is still very limited because they still can not avoid the large global communication required for inner product during waveform GMRES accelerated technique. In next subsections, we will shortly introduce the BiCG method and its background. Here we use \( A \) to express the coefficient matrix instead of \( I - K \) throughout the left of paper.
4 The Parallel IBiCG Method

4.1 The Improved Lanczos Process

Although Lanczos used the similar technique as the coupled two-term recurrence in the early of 1950’s, the majority of papers are dealing with the three-term recurrences process, until, recently, Freund et al. [12] reused this idea to improve numerical stability. They showed that, the latter variant of the Lanczos process, are numerically more stable. That is why we pursue further on this unsymmetric Lanczos process with two-recurrences as the underlying process of the BiCG method.

Recently, Bücke et al. [5] propose a new parallel version of the quasi-minimal residual (QMR) method based on the coupled two-term recurrences Lanczos process without look-ahead strategy. The algorithm is derived that both generated sequences of Lanczos vectors are scaled to unit length and there is only one single global synchronization point per iteration. Based on their similar ideas, we are able to (re)schedule the operations of in the Lanczos process, in such a way that the numerical stability are maintained and all inner products and matrix-vector multiplications of a single iteration step are independent and subsequently communication time required for inner product can be overlapped efficiently with computation time. The framework of this improved Lanczos process based on two-term recurrences is described in Algorithm 1:

Algorithm 1 Improved Lanczos Process

\begin{enumerate}
\item \( p_0 = q_0 = u_0 = 0, \gamma_1 = (\tilde{w}_1, \tilde{v}_1), \xi_1 = (\tilde{w}_1, \tilde{w}_1), s_1 = A^T\tilde{w}_1, \)
\item \( \rho_1 = (\tilde{w}_1, \tilde{v}_1) \epsilon_1 = (s_1, \tilde{v}_1), \mu_1 = 0, \eta = \epsilon_1 / \rho_1; \)
\item \( \text{for } n = 1, 2, \ldots \text{ do} \)
\item \( q_n = \frac{1}{\eta_n} s_n - \frac{\rho_n \mu_n}{\eta_n} q_{n-1}; \)
\item \( \tilde{w}_{n+1} = q_n - \rho_n \tilde{w}_n; \)
\item \( s_{n+1} = A^T \tilde{w}_{n+1}; \)
\item \( \tau_n = A\tilde{y}_n; \)
\item \( u_n = \frac{1}{\tau_n} s_n - \mu_n u_{n-1}; \)
\item \( \tilde{v}_{n+1} = u_n - \rho_n \tilde{v}_n; \)
\item \( \tilde{v}_{n+1} = u_n - \rho_n \tilde{v}_n; \)
\item \( \gamma_{n+1} = (\tilde{v}_{n+1}, \tilde{v}_{n+1}); \)
\item \( \xi_{n+1} = (\tilde{w}_{n+1}, \tilde{v}_{n+1}); \)
\item \( \rho_{n+1} = (\tilde{w}_{n+1}, \tilde{v}_{n+1}); \)
\item \( \eta_{n+1} = (s_{n+1}, \tilde{v}_{n+1}); \)
\item \( \mu_{n+1} = \frac{\rho_{n+1} \eta_{n+1}}{\gamma_{n+1} \eta_{n+1}}; \)
\item \( \epsilon_{n+1} = (s_{n+1}, \tilde{v}_{n+1}); \)
\item \( \text{end for} \)
\end{enumerate}

In the following, we consider the parallelism of the operations in a single iteration step implemented as follows:

- The inner products of a single iteration step (11), (12), (13) and (14) are independent.
- The matrix-vector multiplications of a single iteration step (6) and (7) are independent.
- The communications required for the inner products (11), (12), (13) and (14) can be overlapped with the update for \( p_n \) in (10).

Therefore, the cost of communication time on parallel distributed memory computers can be significantly reduced.

4.2 The improved BiCG method

The improved Lanczos process now is used as a major component to a Krylov subspace method for solving a system of linear equations

\[ Ax = b, \quad \text{where } A \in \mathbb{R}^{n \times n}, x, b \in \mathbb{R}^n. \] (4)

In each step, it produces approximation \( x_n \) to the exact solution of the form

\[ x_n = x_0 + V_n z_n, \] (6)

where \( V_n \) is generated by the improved unsymmetric Lanczos process, and \( z_n \) is determined by the property described later.

For improved Lanczos process, the \( n \)-th iteration step generates

\[ V_{n+1} = [v_1, v_2, \cdots, v_{n+1}] \] and \( P_n = [p_1, p_2, \cdots, p_n], \)

that are connected by

\[ P_n = V_n U_n^{-1}, \quad AP_n = V_{n+1} L_n, \] (7)

where \( L_n \) and \( U_n \) are the leading principal \((n + 1) \times n\) and \( n \times n \) submatrices of the bidiagonal matrices \( L \) and \( U \) generated by the improved Lanczos process. Note that \( L_n \) has full rank since we assume no breakdown occurs. The setting of \( y_n = U_n z_n \) can be used to reformulate the iterate in terms of \( y_n \) instead of \( z_n \) giving

\[ x_n = x_0 + P_n y_n. \] (8)
The corresponding residual vector in term of $y_n$ is obtained by the above scenario, namely

$$r_n = b - Ax_n = r_0 - V_{n+1} L_n y_n$$

(9)

$$= V_{n+1}(\gamma_1 e_1^{(n+1)} - L_n y_n),$$

where the improved Lanczos process starts with $v_1 = \frac{1}{\|r_0\|_2} r_0$ and $e_1^{(n+1)} = (1, 0, \ldots, 0)^T$. By using and properly reorganizing the improved Lanczos process to generate the Krylov subspace and fixing $y_n$, and so implicitly $z_n$ by $y_n = U_n z_n$, we can derive the iterative method which results the algorithm described in Algorithm 2. Due to the limited space, we only can describe the sketch of the algorithm. Note that the underlying Lanczos process operates with unit scaling of both sequences of Lanczos vectors, and is explicitly presented in [4]. In this case, the residual computation simplifies to

$$\|r_n\| = \|\gamma_n \kappa_n\|.$$

(10)

Correspondingly, we can use it as a simple stopping criterion.

If looking carefully the the above iterative process for the solution of linear systems, you can see that Algorithm 2 is just a new variant of the BiConjugate Gradient method (BiCG) [11, 14]. The detailed description and corresponding discussion can be found in [6, 7].

Under the assumptions, the Improved BiConjugate gradient (IBiCG) method using improved Lanczos process as underlying process can be efficiently parallelized as follows:

- The inner products of a single iteration step (16), (17), (18), and (19) are independent.
- The matrix-vector multiplications of a single iteration step (11) and (12) are independent.
- The vector updates (13), (14) and (15) are independent.
- The vector updates (9) and (10) are independent.
- The communications required for the inner products (16), (17), (18) and (19) can be overlapped with the update for $p_n$ in (20).

Therefore, the cost of communication time on parallel distributed memory computers can be significantly reduced.

5 Parallel Implementation

5.1 Data Distribution

For large and sparse matrices, if you are working on different computer system architectures or dealing with different algorithms or data, the efficient storage schemes should be considered differently. In this paper, we decide to use one of the most common format called CRS format (compressed row storage). The main reason behind is that this type of storage scheme is very suitable for both regularly and irregularly structured large and sparse matrices. The detailed description can be found in the literature. Briefly speaking, the nonzeros of large and sparse matrix are stored in row-wise in three one-dimensional arrays. The values of nonzeros are contained in array value. The corresponding column indices are contained in array colInd. The elements of rowInd point to the position of the beginning of each row in value and colInd.

In order to efficiently parallelize the IBiCG algorithm, in particular, on a distributed memory architecture, we first need to decide the data distribution of matrix and vector arrays, hopefully optimally, to each processor and then determine an efficient communication scheme by taking into account different sparsity patterns, not only for matrix-vector multiplication but also for inner products, to minimize the overall execution time. In this paper, we will mainly follow the approach has been used in [3] for data distribution and communication scheme which do not require any knowledge about the matrix sparsity pattern. Also the communication scheme are automatically determined.
by the analysis of the indices of the non-zero matrix elements.

In the following part, we will use the same notations introduced in [3] for the short illustrations. Let $n_k$ and $e_k$ denote the number of rows and no-zeros of processor $k$, where $k = 0, \ldots, p - 1$, respectively. $e$ and $n$ are the total number of corresponding numbers. $g_k$ is the index of the first row of processor $k$, and $z_i$ is the number of non-zeros of row $i$. Easily we can get the following relations: $n = \sum_{k=0}^{p-1} n_k, e = \sum_{k=0}^{p-1} e_k, g_k = 1 + \sum_{i=0}^{k-1} n_i, e_k(g_k, n_k) = \sum_{i=g_k}^{i=n_k} z_i$. Based on the analysis, the total costs of each iteration can be described as $c_1 s e + c_2 n + c_3$ where the first term corresponds to the number of operations for $s$ matrix-vector multiplications, the second term corresponds to the number of vector updates. Since we are mainly dealing with large and sparse matrices, the constants can be neglected. Now we can estimate the contribution of the operations executed on processor $k$ to the total number of operations by

$$\zeta \approx \frac{c_1 s e_k + c_2 n_k}{c_1 s e + c_2 n} = \frac{s e_k + \xi n_k}{s e + \xi n},$$

where $\xi = c_2 / c_1$ depends on both the iterative methods and also the processor architecture. Ideally, the computational load balance should be distributed in such a way that each processor only gets $p$-th fraction for the total number of operations. Based on this, we can use the following strategy to distribute the rows of the matrix and the vector components [3]:

$$n_k = \begin{cases} \min\{1 \leq i \leq \frac{n - \sum_{i=0}^{k-1} n_i - q}{p - k + 1} & \frac{s e_i + \xi n_i}{s e + \xi n} \geq \frac{1}{p} \}, \\
\frac{n}{p} - \sum_{i=0}^{k} n_i & k = q + 1, \ldots, p - 1 \end{cases}$$

Since our main target is large and sparse matrices and we assume $p < n$, the relation $q = p - 1$ or $q + 1 = p - 1$ always hold. It can be shown that for $\xi = c_2 / c_1 \to 0$, each processor will get nearly the same number of non-zeros which means that the execution time of the vector updates in negligible with the execution of matrix-vector multiplications. It also can be shown that for $\xi = c_2 / c_1 \to \infty$ each processor will get nearly the same number of rows which means that the execution time of the matrix-vector multiplications only contribute to a very small part of the total execution time.

5.2 Communication schemes

After the discussion of data distribution, we also need to investigate a suitable communication scheme by preprocessing the distributed column index arrays for efficient matrix-vector multiplications since on a distributed memory systems, its computation requires communication due to the partial vector on each processor. Similarly we will use the approach proposed in [3] for our communication schemes.

If we decide to implement the matrix vector multiplication row-wisely, components of the vector $x$ of $y = Ax$ are communicated. We firstly analyze the arrays colInd on each processor to determine which elements result in access to non-local data. Then, the processors exchange information to decide which local data must be sent to which processors. Based on the above analysis, we will reorder these two arrays colInd and value in such a way that the data that results in access to processor $l$ is collected in block $l$, called local block. The motivation behind this reordering is to perform computation and communication overlapped. The elements of block $l$ succeed one another row-wise with increasing column index per row. The detailed description can be found in [3].

For the parallel implementation of this operation, each processor executes asynchronous receive-routines to receive necessary non-local data. Then all components of the vector $x$ that are needed on other processors are sent asynchronously. After the required data are available, each processor will perform operations with it local block. After that, as soon as non-local data arrive, processor continue the matrix vector operation by accessing the elements of the corresponding blocks. It will be repeated until the whole operation is completed. According to the communication scheme described, the communication and computation are performed overlapped so that waiting time can be reduced.

6 Experimental Results

The three different MOS devices ldd, soi and kar described in Table 1 are used to construct six simulation examples described in [23], each device being subjected to either a drain voltage pulse with the gate held high (the D examples), or a gate voltage pulse with the drain held high (the E examples). The (karG64 and soiG64) examples are constructed by refining the meshes of karG and soiG to contain 64 vertical lines. Zero-volt Dirichlet boundary conditions are imposed by ohmic contacts at the source and along the bottom of the substrate and Neumann reflecting boundary conditions are imposed along the left and right sides. The description of device and illustration of the drain-driven karD example are described in Figure 1 and Table 1 which are taken from [23].

We have compared our approach with parallelized pointwise Newton/GMRES, WRN [25], WN/WGMRES, and WRN with CSOR acceleration. The backward Euler method with 256 fixed timesteps is used for all experiments, on a simulation interval of 51.2 or 51.2 picoseconds. Although the use of global uniform timesteps precludes multirate integration, it
also simplifies the problem of load-balancing. The convergence criterion for all experiments is that the maximum relative error of any terminal current over the simulation interval be less than $10^{-4}$. The initial guess for WRN and for the accelerated waveform methods is produced by performing 16 or 32 WR iterations beginning with flat waveform extended from the initial conditions. Table 2 shows a relatively complete comparison of the execution times in wall clock seconds required to complete a transient simulation of the \textit{karD}, \textit{soiG}, \textit{lddD} and \textit{karG64} examples using WRN, WRN with CSOR acceleration, pointwise Newton/GMRES, WN/WGMRES and WN/IBiCG respectively on Parsytec massively parallel distributed memory computer. From the experimental results, we can see clearly that pointwise-GMRES does exhibit some speedup when parallelized, but the speedup flattens out after four processors. Compared with that of original WRN method, the execution time of WRN with acceleration technique CSOR is obviously shorter. WN/WGMRES approach propoed in [23] is better than WRN with CSOR acceleration technique. Our approach need shortest execution times to complete a transient simulation which is consistent with our expectations where the inner products, even matrix-vector multiplications are totally independent and therefore can be executed simultaneously.

References

Table 2. Experimental results on different simulation examples

<table>
<thead>
<tr>
<th>Device</th>
<th>( \text{kav}_D )</th>
<th>( \text{so}_G )</th>
<th>( \text{ld}_D )</th>
<th>( \text{kav}_G64 )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pointwise-GMRES</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1466.48</td>
<td>1</td>
<td>82.12</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>884.20</td>
<td>2</td>
<td>400.26</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>358.40</td>
<td>4</td>
<td>130.38</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>396.42</td>
<td>8</td>
<td>137.62</td>
<td>8</td>
</tr>
<tr>
<td>16</td>
<td>688.66</td>
<td>16</td>
<td>148.35</td>
<td>16</td>
</tr>
<tr>
<td><strong>WRN</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4876.88</td>
<td>4</td>
<td>1148.59</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>2346.48</td>
<td>8</td>
<td>623.47</td>
<td>8</td>
</tr>
<tr>
<td>16</td>
<td>1246.75</td>
<td>16</td>
<td>408.21</td>
<td>16</td>
</tr>
<tr>
<td><strong>WRN-CSOR</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>920.43</td>
<td>4</td>
<td>233.28</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>472.31</td>
<td>8</td>
<td>124.65</td>
<td>8</td>
</tr>
<tr>
<td>16</td>
<td>242.14</td>
<td>16</td>
<td>58.33</td>
<td>16</td>
</tr>
<tr>
<td><strong>WN/WGMRES</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>632.45</td>
<td>4</td>
<td>202.35</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>360.28</td>
<td>8</td>
<td>114.38</td>
<td>8</td>
</tr>
<tr>
<td>16</td>
<td>190.42</td>
<td>16</td>
<td>77.46</td>
<td>16</td>
</tr>
<tr>
<td><strong>WN/IBiCG</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>223.57</td>
<td>4</td>
<td>77.18</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>108.69</td>
<td>8</td>
<td>43.96</td>
<td>8</td>
</tr>
<tr>
<td>16</td>
<td>59.46</td>
<td>16</td>
<td>24.59</td>
<td>16</td>
</tr>
</tbody>
</table>


