An Improved Eigensolver for Quantum-dot Cellular Automata Simulations

A. Taylor Baldwin, Jeffrey Will, Douglas Tougaw
Electrical and Computer Engineering
Valparaiso University
Valparaiso, Indiana
Aaron.Baldwin@valpo.edu

***Abstract*— The work in this paper describes the application of an optimized eigensolver algorithm to produce the kernel calculations for simulating quantum-dot cellular automata (QCA) circuits, an emerging implementation of quantum computing. The application of the locally optimal block preconditioned conjugate gradient (LOBPCG) method to calculate the eigenvalues and eigenvectors for this simulation was shown to exhibit a 15.6 speedup over the commonly used QR-method for a representative simulation and has specific advantages for the Hermitian, positive-definite, sparse matrices commonly encountered in simulating the Time-Independent Schrödinger equation. We present the computational savings for a simulation analyzing the effect of stray charges near a four-cell line of QCA cells with a single driver cell, and we discuss implications for wider application. We further discuss issues of problem preconditioning which are specific to QCA simulation when utilizing the LOBPCG method.**

Keywords—Performance Metrics; Quantum-dot Cellular Automata, Eigensolvers, LOBPCG, Simulation, Numeric Linear Algebra

I. INTRODUCTION

The following paper describes the application of an optimized eigensolver to a simulation which computes the effect of stray charges on a four-cell line of quantum-dot cellular automata. We first describe the fundamentals of quantum-dot cellular automata (QCA) and the methodology behind our simulation.

This is followed by a detailed description of the linear algebra behind the optimized eigensolver that was used within the simulation. This eigensolver uses the locally optimal block preconditioned gradient (LOBPCG) method to find the eigenvalues and eigenvectors needed for the QCA simulation. Our objective is to provide conclusive evidence that the LOBPCG method is a superior eigensolver for this computation when compared to the efficiency of traditional eigensolver algorithms.

A. Quantum-dot Cellular Automata Simulation

Quantum-dot Cellular Automata (QCA) utilize a nanoscale architecture based on placing four quantum dots, at the four corners of the square. Electrons are able to tunnel between adjacent pairs of these dots, but each dot can only be occupied by one electron (ignoring opposite spins) at a given time [1, 2, 3, 4, and 5]. Two low-energy ground states occur when the two electrons occupy diagonally opposite sites, either right-leaning or left-leaning, and these two states can then be encoded as a binary '1' or '0,' allowing quantum storage of binary information within a single QCA cell. Electrostatic interaction between neighboring QCA cells can then be used to create quantum logic circuits, much like the transistors of present silicon-based fabricated circuits [6, 7, 8, 9, 10, 11, 12, 13, 14, and 15]. The benefit of QCA cells comes from low switching times on the order of a few nanoseconds and the fact that QCA bistable elements operate at a tiny fraction of the power required by silicon-based transistors [16 and 17].

For the scope this paper, we consider and present a single possible QCA configuration consisting of four-site cells selected to match values used in canonical publications (cf. [1, 2, 3, 4, and 5]), but the calculations required for simulation are generalizable to a variety of QCA implementations. We consider a cell consisting of four quantum-dots at the corners of a square of diagonal 40 nm with an intercell spacing of 60 nm (See Figure 1.)

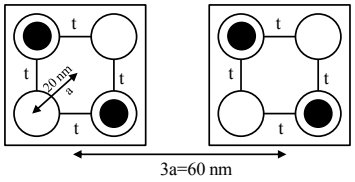

Figure 1. A pair of QCA cells using typical geometry. Each cell has four quantum-dots, with two electrons occupying each cell. Tunneling is allowed between sites, and Coulombic interactions within cells cause the electrons to achieve bistable ground states where electrons occupy opposite corners of the cell. Inter-cellular Coulombic interactions cause neighboring cells to have identical "leans" on the ground states, allowing propagation of information along a line of QCA cells.

Simulations of QCA devices require the consideration of the six possible configurations shown in Figure 2. Each configuration is represented by occupancy basis kets, also shown in the figure. The bistable ground states are represented by $|\phi_2\rangle = |1\ 0\ 1\ 0\rangle$ (binary 1) and $|\phi_5\rangle = |0\ 1\ 0\ 1\rangle$ (binary 0).

978-1-4799-1365-7/13/$31.00 ©2013 IEEE

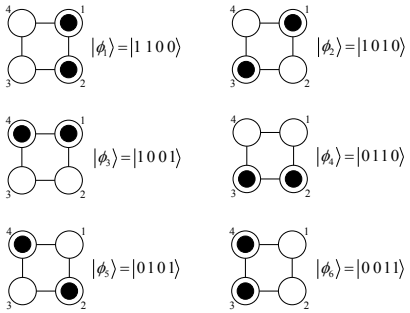

Figure 2. Six possible configurations of the two electrons, giving the basis kets for the single QCA cell. Configurations 2 and 5 with electrons occupying opposite corners are the two ground-state configurations and are encoded as a binary '1' and '0' respectively.

When considering a system with two cells, we must consider the direct product of the six basis states of the first cell with the six basis states of the second cell, which results in a full-basis set of 36 two-cell basis kets as shown schematically in Figure 3

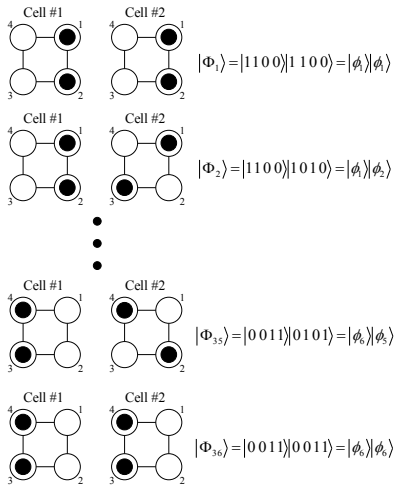

Figure 3. Sequence of 36 basis kets forming the full basis of a two-cell configuration from the direct product of two single-cell kets.

In like fashion, a circuit of three QCA cells would require 36 x 6 = 216 basis kets and a QCA cell system of N cells would require 6^N basis kets, each of which would be the direct product of the N single-cell basis kets. With useful QCA circuits typically consisting of 20 or more cells [7, 18, and 15], the computational demand for simulation rapidly escalates.

This paper focuses on improving the computation time for the stray-charge analysis of [19] which considers an input driver cell and both three and four additional driven cells, each contributing six candidates for permutation in the full basis ket. The Hamiltonian for these systems can be written in matrix form using the 216 or 1296 direct product basis kets. The Hamiltonian used for the systems is

$$H = \sum_i E_{0_i} n_i + \sum_{i>j} t_{i,j}\left(a_i^\dagger a_j + a_j^\dagger a_i\right) + \sum_{i>j} V_0 \frac{n_i n_j}{\left|\bar{R}_i - \bar{R}_j\right|} + \sum_{i,j} V_0 \frac{n_i q_j}{\left|\bar{R}_i - \bar{R}_j\right|}$$

where the first term represents the on-site energy to confine each electron to a quantum-dot, the second term defines the tunneling of adjacent sites within the cell, and the third represents Coulombic interaction of inter-cellular electrons within the cell. The fourth term accounts for "external" charges, including the two fixed charges of the driver cell, the single stray charge, and a layer of positive charges 10 nm directly below the sites comprising the QCA cells (representing the donor atoms of the electrons). The Hamiltonian must then be diagonalized to compute the eigenvalues and eigenvectors of the matrix in order to determine the energies and charge configurations that satisfy the Time-Independent Schrödinger Equation, shown in (1).

$$H\left|\Psi_i\right\rangle = E_i\left|\Psi_i\right\rangle \tag{1}$$

For the purposes of the simulation considered here, it is required to find the lowest energy state (ground state) of the system and the second-lowest energy state. These two energies correspond to the lowest two eigenvalues of the matrix. The difference between these two states is referred to as the "excitation energy" of the first excited state. The eigenvector corresponding to the ground state can be used to determine the expected value of the ground-state charge configuration, as shown in Equation (2)

$$\rho_i = \left\langle \Psi_0 \left| n_i \right| \Psi_0 \right\rangle \tag{2}$$

The four charge densities are subsequently used to determine the polarization of a cell, given in Equation (3)

$$P = \frac{\rho_1 + \rho_3 - \left(\rho_2 + \rho_4\right)}{\rho_1 + \rho_2 + \rho_3 + \rho_4} \tag{3}$$

where P takes on a value of +1 for electron occupation of sites 1 and 3 and a value of -1 for electron occupation in sites 2 and 4.

B. 1.2 Optimized Eigensolvers

An effective way to minimize the computation time interval for the calculation of the eigenvalues for large, symmetric, positive definite matrices is to exploit the specific characteristics of these matrices by choosing an appropriate eigensolver. The locally optimal block preconditioned conjugate gradient method (LOBPCG), is one such alternative [22]. This algorithm is specifically optimized for the calculation of large symmetric eigenvalue problems, based on a local preconditioned optimization of a three-term recurrence and through the use of the Raleigh-Ritz method, resulting in a basis on the Krylov subspace, as seen in [22]. The generalized eigenvalue problem is of the form $(A - \lambda \cdot B) = 0$ where A and B are real symmetric matrices and A is positive definite. (It is not necessary for matrix B to be positive definite). The eigenvalue problem begins its solution within LOBPCG with an approximated preconditioned matrix that contains n columns and m rows. This estimate should be similar to the eigenvectors that will be found in the solution. The value of n is the number of eigenvalues required in the solution and the column values contain the approximate preconditioned eigenvector estimates that are associated with each extreme eigenvalue.

The algorithm focuses on preconditioning the matrix, beginning with random approximations. This allows the LOBPCG algorithm to achieve superior convergence and has augmented the speed and accuracy of this eigensolver. The LOBPCG algorithm uses simultaneous block iterations that allow the computation of an entire subspace rather than a single eigenvector by itself. This allows the algorithm to work faster on parallel processing computers, and the convergence of the most extreme (largest and smallest) eigenvalues occurs more rapidly [22, 23, and 24]. The LOBPCG method utilizes the Rayleigh-Ritz procedure within the trial subspace mentioned earlier to iterate the program until the required eigenvalues and eigenvectors converge. The Rayleigh-Ritz method is a direct variational method in which the minimum of a functional represented on a normalized linear space is approximated with a linear combination that includes the elements within that space. The output of LOBPCG then provides the approximations of the corresponding number of the most extreme eigenvalues, the smallest or largest, and their corresponding eigenvectors, which are consistent with numerically computed eigenvalues found using other methods [22 and 25]. The process of the algorithm is outlined in Figure 4.

(1) Input: m starting vectors $x_1^{(0)}, \dots, x_m^{(0)}$, to compute: $Ax, Bx,$ and Tx for a given vector x, and and vector inner product (x, y).
(2) Start: select $\mu_j^{(0)}$, and set $p_j^{(0)} = 0, j = 1, \dots, m.$
(3) Iterate: For $i = 0, \dots,$ Until Converegence Do:
(4) $\mu_j^{(i)} \coloneqq \frac{x_j^{(i)}, Bx_j^{(i)}}{x_j^{(i)}, Ax_j^{(i)}}, j = 1, \dots, m;$
(5) $r_j \coloneqq Bx_j^{(i)} - \mu_j^{(i)} Ax_j^{(i)}, j = 1, \dots, m;$
(6) $w_j^{(i)} \coloneqq Tr_j, j = 1, \dots, m;$
(7) Use Rayleigh − Ritz method for pencil $B - \mu A$ on trial subspace. Span$\{w_1^{(i)}, \dots, w_m^{(i)}, x_1^{(i)}, \dots, x_m^{(i)}, p_1^{(i)}, \dots, p_m^{(i)}\}$;
(8) $x_j^{(i+1)} \coloneqq \sum_{k=1,\dots,m} a_k^{(i)} w_k^{(i)} + \gamma_k^{(i)} p_k^{(i)};$
(9) Output: the approximations $\mu_j^{(k)}$ and $x_j^{(k)}$ to the largest eigenvalues μ_j and corresponding eigenvectors, j = 1, … , m.

Figure 4. The linear algebraic process behind the LOBPCG algorithm found in [22].

The LOBPCG method compares equally if not better than several other methods that are used to calculate the eigenvalues of large matrices. The Davidson method [26], for example, converges at roughly the same rate as LOBPCG, but in terms of computation LOBPCG is significantly less expensive. The Jacobi-Davidson QR (JDQR) method for the solution of eigenpairs [20], as another example, is not as robust as LOBPCG because it fails to produce the eigenvalues and eigenvectors for larger matrices (greater than 108 x108), and is about half the speed of LOBPCG in typical cases. JDQR also fails to accurately compute eigenpairs when handling randomly generated initial guess matrices; a characteristic which LOBPCG handles well, as seen through tests where JDQR converges to the second smallest eigenpair instead of converging to the smallest one. The LOBPCG algorithm has been implemented in software on several platforms and computer languages. The LOBPCG function is quite versatile and can be manipulated to fit many specific characteristics of the matrix whose extreme eigenvalues and eigenvectors it is computing. The number of iterations that occur before convergence can be chosen to optimize computation time and accuracy of the eigenpairs. The precision or residual tolerance of the solution can be chosen as well within the function parameters. The software for this and other preconditioned eigensolvers can be found at [27]. Our research utilized a slightly modified version of the MATLAB implementation from [28].

The LOBPCG function has also been used in several other published applications, including research done with the Earth Simulator Supercomputer in Japan. The main research, highlighted in [29], that was performed on this supercomputer was a high-performance computation, with multi-billion cell matrices, to determine several exact numerical approaches to solving quantum many-body problems.

II. Applications of LOBPCG to QCA Simulation

The optimization for QCA simulation discussed in this paper focuses on optimizing the accuracy and speed of an algorithm that computes a full-basis calculation to determine the stray charge on an n-cell QCA line. The particular work performed focused on the optimization of the calculation of the two smallest eigenvalues of multiple 1296 x 1296 matrices. The purpose of this calculation is to simulate the effects of stray charge on a row of four quantum-dot cellular automata and a driver cell, similar to that done in [19] and [30].

Research primarily focused on the development and implementation of algorithms that provided more efficient and faster computation of the smallest eigenvalues of large matrices. Three linear algebraic methods were used to successfully compute eigenvalues for these large, symmetric, sparse, Hermitian matrices. They included three solvers for eigenvalues within the MATLAB environment: `eig()`, `eigifp()`, and `LOBPCG()`, discussed below

The default Matlab function for the computation of eigenvalues is `eig()`. This function utilizes the QR algorithm or (for the generalized problem) uses the QZ method. The QZ method uses a prior transformation to Hessenberg tri-diagonal form to find the eigenvalues. The QR algorithm uses a QR decomposition [20], where the matrix is written as a product of an orthogonal matrix and an upper triangular matrix, then the factors are multiplied in the reverse order, and the process continues until all the eigenvalues are calculated.

The function `eigifp()`, described in [21], uses a two-level iteration with a projection on Krylov subspaces generated by a shifted matrix A_{λ_k} in the inner iteration. Either the Lanczos or the Arnoldi algorithm is employed for the projection, and the preconditioner of the matrix uses Incomplete Lower-Upper (ILU) factorization. This method only provides us with accurate eigenvectors and the eigenvalues were too dissimilar from those found with the default Matlab function to use as an effective comparison to `eig()`.

The function `LOBPCG()`, the algorithm for which was discussed in the previous section, uses an iterative minimization of the generalized Rayleigh quotient to find the smallest eigenvalues of a Hermitian matrix. The MATLAB default eigenvalue calculator, the `eig()` function mentioned earlier, was replaced by `LOBPCG()`. The `eig()` function is a general-purpose function that is not optimized for specific types of matrices like the sparse, positive-definite, Hermitian matrices that are being dealt with in a QCA stray charge calculation. The `eig()` function uses the QR algorithm with some variants, depending on the settings, to compute the eigenvalues of a given matrix, and it returns all the eigenvalues of that matrix. The `LOBPCG()` function is called with the MATLAB code format found in the documentation of [28]:

```
[blockVectorX,lambda,failureFlag]=lobpcg(blockVectorY,
operatorA,residualTolerance,maxIterations,verbosityLevel)
```

The `LOBPCG()` function is calibrated in this statement to return three variables. The `blockVectorX` contains a matrix with the eigenvectors that correspond to the two smallest eigenvalues, or whatever *n* number of eigenvalues are necessary for the calculation. One important note is that within the LOBPCG library documentation it is suggested that users should not try to solve for more than twenty percent of the possible eigenvalues for a matrix. This is because the calculation decreases in accuracy and speed as more and more eigenpairs are being calculated. The `lambda` variable within the function contains a vector with the eigenvalues from the operand matrix. These values represent the smallest *n* eigenvalues of the operand matrix `operatorA`, which are sorted smallest to largest in the vector returned from the function after the calculation has been completed.

The `failureFlag` indicates the Boolean value returned from a statement which checks if there was enough iterations for the eigenvalues to converge to a specific value. Also, inside the function itself a wide variety of commands can be implemented. In this case `blockVectorY` contains an estimation matrix that should be reasonably similar to the eigenvectors contained within `blockVectorX`.

The `operatorA` statement contains the matrix whose eigenpairs are being found. In order to optimize the algorithm used by the `LOBPCG()` function the operator matrix should be a sparse, Hermitian, positive-definite, symmetric matrix. Our simulation is composed of calculations that require the diagonalization of a Hamiltonian matrix, `operatorA`, that is 6^4 by 6^4, or of size 1296 x 1296. This Hamiltonian, for the QCA application, represents a four-cell QCA line with a fixed driver cell as seen in Figure 5.

The `residualTolerance` and `maxIterations` variables control the residual tolerance of the eigenvalues and the maximum number of iterations that are completed to find the eigenvalues. When the tolerance is decreased and the iteration value is increased, both the speed of the calculation and the accuracy of the eigenvalues increase. However, as the residual tolerance decreases or the iteration value increases, the computation time of the simulation increases significantly.

The last setting is the verbosityLevel, which can be set to the values 0, 1, or 2, where each corresponding value controls the amount of printed info seen while the function is processing. The zero provides the least amount of feedback about the results of the algorithm. This value was used in this application because allowing the program to output information decreases the efficiency and speed of the optimized program.

III. Results and Analysis

The optimization provided a strong indication of higher efficiency in the computation of eigenvalues. This was shown through a decrease by a factor of 15.6 in the computation time. The results of this computation using LOBPCG can be seen in Figure 5. This can be compared to Figure 6, which is the same simulation performed using the eig() function to solve for eigenvalues instead, thus demonstrating the validity of the results with the improved eigensolvers.

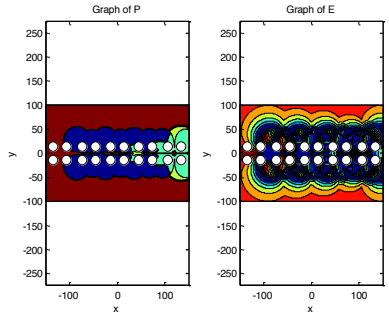

Figure 5. Excitation energy for the first excited state of a four-cell QCA line using LOBPCG is seen on the left. The color at each point represents the energy between the first excited state and the ground state when the stray charge is located at that point. The simulation is run for both polarities of input polarization, and the least successful outcome is shown for each point. The simulation is run for both values of input polarization, and the least successful outcome is shown.

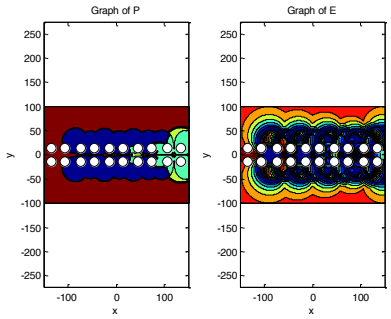

Figure 6. Excitation energy and polarization for the first excited state of a four-cell QCA line using EIG. The graph represents the same stray charge values as Figure 3.

The default MATLAB eigenvalue solver eig() was able to compute the stray charge of a four-cell QCA line in 33:22:00. This is in comparison to the function LOBPCG(), which was optimized for large, sparse, positive-definite matrices, like those used to simulate the stray charge. The LOBPCG() function was able to compute the same information in 02:08:00

as seen in Table 1. The eigifp() algorithm, which was mentioned earlier, solved the problem correctly but required a calculation time of 07:32:00, significantly slower than LOBPCG(). These results are summarized in Table 1.

The other values on Table 1 demonstrate the calculation time needed for each of the methods using different sizes of matrices. These matrices were similar to the ones that were built and solved in the stray charge full basis simulation but were larger sizes, and the values included are approximate. The full four cell simulation only used 1296 x 1296 matrices, but for a five cell simulation 7776 x 7776 matrices would be solved instead. As can be seen the runtime is always the least for LOBPCG no matter the size of the matrix. In addition, the LOBPCG eigensolver was the only solver capable of solving larger matrices on the order of 23000 x 23000. These initial tests provide compelling evidence that the LOBPCG algorithm is a very effective eigensolver replacement for QCA simulations.

TABLE 1. EIGENSOLVER RUNTIME

Eigenvalues of 7776*7776 Matrices	EIG	EIGIFP	LOBPCG
Calculation Time (hh:mm:ss)	(00:00:50)	(00:00:02)	(00:00:01)
Eigenvalues of 23k*23k Matrices	EIG	EIGIFP	LOBPCG
Calculation Time (hh:mm:ss)	N/A	N/A	(37:06:36)
Full Simulation w/ 135k 6^4*6^4 Matrices	EIG	EIGIFP	LOBPCG
Calculation Time (hh:mm:ss)	(33:20:00)	(07:30:00)	(02:08:00)

One important note, with reference to the residualTolerance and iterations parameters, within our simulation, was that increasing the iteration number from 35 to 50 in our simulation had the effect of increasing the computation time from 02:08:00 to 03:15:00. It is important to note also that as the iterations increase the accuracy does as well; however, at a certain point the accuracy is maximized, which in our case was the implementation of thirty-five iterations.

The computer used in these simulations was comprised of an Intel Core i7-2600K CPU running at 3.40 GHz on an Intel Z-68 express chipset motherboard (Gigabyte Z68XP-UD3) with 16 GB DDR3 dual-channel system memory and magnetic disk running 64-bit Windows 7 Professional. The version of MATLAB used was 7.12.0.635 R2011a 64-bit with the addition of MATLAB's parallel computing toolbox. Times reported for simulation are reported using equal CPU resources (four threads on two cores) for all cases of algorithm implementation.

A. Selection of Estimate Vectors

The QCA simulation we consider here requires the calculation of the lowest two eigenpairs of the (1296 x 1296) matrix representing the location of the stray charge. This must be done for each possible location in the 200 x 300 nm region of simulation. In our case, the space was divided into a 300 x 450 grid and thus the lowest two eigenpairs were calculated for a total of 135,000 1296 x 1296 matrices. In the application of the standard QR algorithm, we are not able to exploit any spatial correlation of an already-calculated solution for the matrix at location (m, n) to do a subsequent calculation for the matrix located at, say, (m+1, n) or (m, n+1). However, we can exploit the ability of the LOBPCG algorithm to accept an estimate vector as the seed of its solution, and it is the optimization of determining the estimate vector (blockVectorY), that we turn to now and subsequently analyze.

The initial estimate matrix used for the simulation was an educated guess that used ones and zeroes to populate a preconditioned hypothesis matrix for the eigenvectors that were related to the smallest two eigenvalues. After the eigenpair computation of the first matrix, the two eigenvectors from the answer are used to populate the blockVectorY parameter, called nextguess below,

```
[vecs1,vals1,failureFlag]=lobpcg(nextguess,
                    H_final1,1e-7,35,0)
```

and from that point on the previous eigenvectors are used as the guess matrix for all subsequent eigenpair calculations.

This was found to work well because the actual eigenvectors will be very similar to the ones that were calculated for the previous matrix. This method is much more efficient than using a random number generator to populate the guess eigenvector matrix. When the change was made to use the previous eigenvectors as the preconditioned guess matrix , the calculation time was reduced from 02:38:00 to 02:08:00. Also, the accuracy, when compared to the results of the eig() function, improved to 99.99%, where 99% of the eigenvalues of one method were within one percent of the value found using the other method. The total computational time of each algorithm is outlined in Figure 7 below.

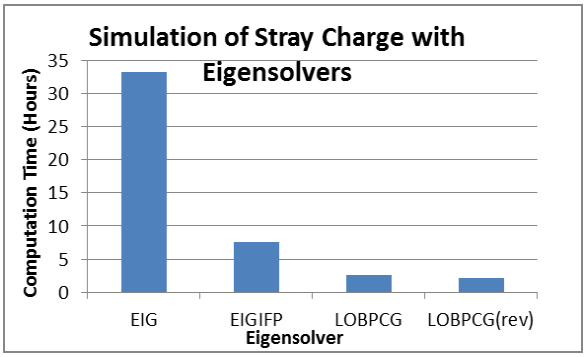

Figure 7. Simulation time for each of the eigensolvers, LOBPCG contains two different implementations. The first version of LOBPCG uses a randomly generated eigenvector guess matrix, while the revised version uses an estimate based on the previous eigenvectors that were found in the previous calculation.

The accuracy was verified using analysis that would calculate the absolute error between the eigenvalues of all the matrices generated in the simulation using eig() versus using LOBPCG. From this result it was concluded that the

difference between the eigenvalues of each method was greater than 1% different for 0.0118%of the data, which provides strong evidence for the accuracy of the LOBPCG method.

IV. Conclusions

We have demonstrated that the LOBPCG algorithm has clear advantages over the standard QR algorithm for calculating eigenvectors and eigenvalues in QCA simulation. In the stray-charge problem in particular, the time of calculation was reduced by a factor of 12.8 initially. After exploiting the special correlation found in the problem to improve the estimate vectors seeding the calculation, we further reduced the time for an overall speedup of 15.6.

The work herein has demonstrated the need for better eigensolvers for the QCA simulation community. Presently, common computation software used in the community of QCA simulation for the presented problem takes over 30 hours for the problem studied. It is hoped that the work presented here will motivate other QCA simulation researchers to apply the LOBPCG algorithm to their simulations and thereby move the research in QCA simulation forward by implementing an order-of-magnitude reduction in computation time.

References

[1] C. S. Lent, P. D. Tougaw, and W. Porod, "Bistable saturation in coupled quantum dots for quantum cellular automat," *Applied Physics Letters*, vol. 62, no. 7, pp. 714-716, Feb. 1993.

[2] C. S. Lent and P. D. Tougaw, "Lines of interacting quantum-dot cells: a binary wire," *Journal of Applied Physics*, vol. 74, no. 10, pp. 6227-6233, Oct. 1993.

[3] C. S. Lent, P. D. Tougaw, and W. Porod, "Bistable saturation in coupled quantum-dot cells," *Journal of Applied Physics*, vol. 74, no. 5, pp. 3558-3566 , Sept. 1993.

[4] C. S. Lent, P. D. Tougaw, W. Porod, and G. H. Bernstein, "Quantum cellular automata," *Nanotechnology*, vol. 4, no. 1, pp. 49-57, Jan. 1993.

[5] C. S. Lent and P. D. Tougaw, "A device architecture for computing with quantum dots," *Proceedings of the IEEE*, vol. 85, no. 4, pp. 541-557, Apr. 1997.

[6] A. Gin, S. Williams, H. Meng, and P.D. Tougaw, "Hierarchical design of quantum cellular automata," *Journal of Applied Physics*, vol. 85, no.7, pp. 3713-3720 , Apr. 1999.

[7] C. R. Graunke, D. I. Wheeler, P. D. Tougaw, and J. D. Will, "Implementation of a crossbar network using quantum-dot cellular automata," *IEEE Trans. Nanotechnol.*, vol. 4, pp. 435-440, July 2005.

[8] S. Hashemi, M. Tehrani, and K. Navi, "An efficient quantum-dot cellular automata full-adder," *Scientific Research and Essays*, vol. 7, no. 2, pp. 177-189, 2012.

[9] M. Hayati and A. Rezaei, "Design and Optimization of Full Comparator Based on Quantum-Dot Cellular Automata," *ETRI Journal*, vol. 34, no. 2, 2012.

[10] J. Janulis, P. D. Tougaw, S. Henderson, and E. Johnson, "Serial Bit Stream Analysis Using Quantum-Dot Cellular Automata," *IEEE Trans. Nanotechnol.*, vol. 3, pp. 158-164, Mar. 2004.

[11] R. Katti and S. Shrestha, "Novel asynchronous registers for sequential circuits with quantum-dot cellular automata," in *IEEE International Symposium on Circuits and Systems*, 2012.

[12] J. R. Pasky, L. Henry, and P. D. Tougaw, "Regular arrays of quantum-dot cellular automata macrocells," *Journal of Applied Physics*, vol. 87, no. 12, pp. 8604-8609 , June 2000.

[13] A. Shahidinejad et al, "A Novel Quantum-dot Cellular Automata XOR Design," *Advanced Materials Research*, pp. 622-623, 545-550, 2012.

[14] P. D. Tougaw and C. S. Lent, "Logical devices implemented using quantum cellular automata," *Journal of Applied Physics*, vol. 75, no. 3, pp. 1818-1825, Feb. 1994.

[15] J. D. Wood and P. D. Tougaw, "Matrix Multiplication Using Quantum-Dot Cellular Automata to Implement Conventional Microelectronics," *IEEE Trans. Nanotechnol.*, vol. 10, pp. 1036-1042, Sept. 2011.

[16] C. S. Lent, J. Timler, and P. D. Tougaw, "Quantum-dot cellular automata," in *Nanoelectronic Devices*, MIT Press, 2001.

[17] P. D. Tougaw and C. S. Lent, "Dynamic behavior of quantum cellular automata," *Journal of Applied Physics*, vol. 80, no. 8, pp. 4722-4736, Oct. 1996.

[18] B. Sen, M. Dutta, D. Saran, and B. Sikdar, "An efficient multiplexer in quantum-dot cellular automata," in *Progress in VLSI Design and Test*, 2012, pp. 350-351.

[19] M. LaRue, P. D. Tougaw, and J. Will, "Stray Charge in Quantum-dot Cellular Automata: A Validation of the Intercellular Hartree Approximation," *IEEE Trans. Nanotechnol., vol.* 2, pp. 225-233, 2013

[20] D. R. Fokkema, G. L. G. Sleijpen, and H. A. Van Der Vorst, "Jacobi-Davidson style QR and QZ algorithms for the reduction of matrix pencils," *SIAM Journal on Scientific Computing*, vol. 20, pp. 12-23, 1992.

[21] G. Golub, and Q. Ye, "An Inverse Free Preconditioned Krylov Subspace Method for Symmetric Generalized Eigenvalue Problems," *SIAM Journal on Scientific Computing*, vol. 24, pp. 312-334, 2004.

[22] A. V. Knyazev, "Toward The Optimal Preconditioned Eigensolver: Locally Optimal Block Preconditioned Conjugate Gradient Method," *SIAM Journal on Scientific Computing*, vol. 23, no. 2, pp. 517-541, 2001.

[23] A. V. Knyazev, "Preconditioned Eigensolvers—an Oxymoron?" *Electron. Trans. Numer. Anal.*, vol. 7, pp. 104-123, 1998.

[24] A. V. Knyazev, "Preconditioned Eigensolvers: Practical Algorithms. In Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guid," *SIAM*, Philadelphia, pp. 352-368, 2000.

[25] A. V. Knyazev and A. L. Skorokhodov, "Preconditioned Gradient-Type Iterative Methods In a Subspace for Partial Generalized Symmetric Eigenvalue Problems," *SIAM Journal of Nuerical Analysis*, vol. 31, pp. 1226-1239, 1994.

[26] R. B. Morgan. "Davidson's Method and Preconditioning for Generalized Eigenvalue Problems," *Journal of Computational Physics*, vol. 89, pp. 241-245, 1990.

[27] A. Knyazev. (2012, Sept. 13) *Preconditioned Conjugate Gradient Methods for Eigenproblems*. Available: http://math.ucdenver.edu/~aknyazev/software/CG/

[28] Mathworks. (2004). *lobpcgm - File Exchange*. (2012, October 31). Available: http://www.mathworks.com/matlabcentral/fileexchange/48-lobpcg-m

[29] S. Yamada et al., "High-performance computing for exact numerical approaches to quantum many-body problems on the earth simulator," in SC '06 Proceedings of the 2006 ACM/IEEE conference on Supercomputing. Article No. 47, doi:10.1145/1188455.1188504

[30] P. D. Tougaw and C. S. Lent, "The effect of stray charge on quantum cellular automata," *Japanese Journal of Applied Physics*, vol. 34, pp. 4373-4375, 1995.