

SANDIA REPORT

SAND2013-10575

Unlimited Release

Printed December, 2013

QCAD Simulation and Optimization of Semiconductor Double Quantum Dots

E. Nielsen, X. Gao, I. Kalashnikova, R. P. Muller, A. G. Salinger, R. W. Young

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.



Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: <http://www.osti.gov/bridge>

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: <http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online>



QCAD Simulation and Optimization of Semiconductor Double Quantum Dots

E. Nielsen, X. Gao, I. Kalashnikova, R. P. Muller, A. G. Salinger, R. W. Young

Abstract

We present the Quantum Computer Aided Design (QCAD) simulator that targets modeling quantum devices, particularly silicon double quantum dots (DQDs) developed for quantum qubits. The simulator has three differentiating features: (i) its core contains nonlinear Poisson, effective mass Schrodinger, and Configuration Interaction solvers that have massively parallel capability for high simulation throughput, and can be run individually or combined self-consistently for 1D/2D/3D quantum devices; (ii) the core solvers show superior convergence even at near-zero-Kelvin temperatures, which is critical for modeling quantum computing devices; (iii) it couples with an optimization engine Dakota that enables optimization of gate voltages in DQDs for multiple desired targets. The Poisson solver includes Maxwell-Boltzmann and Fermi-Dirac statistics, supports Dirichlet, Neumann, interface charge, and Robin boundary conditions, and includes the effect of dopant incomplete ionization. The solver has shown robust nonlinear convergence even in the milli-Kelvin temperature range, and has been extensively used to quickly obtain the semiclassical electrostatic potential in DQD devices. The self-consistent Schrodinger-Poisson solver has achieved robust and monotonic convergence behavior for 1D/2D/3D quantum devices at very low temperatures by using a predictor-correct iteration scheme. The QCAD simulator enables the calculation of dot-to-gate capacitances, and comparison with experiment and between solvers. It is observed that computed capacitances are in the right ballpark when compared to experiment, and quantum confinement increases capacitance when the number of electrons is fixed in a quantum dot. In addition, the coupling of QCAD with Dakota allows to rapidly identify which device layouts are more likely leading to few-electron quantum dots. Very efficient

QCAD simulations on a large number of fabricated and proposed Si DQDs have made it possible to provide fast feedback for design comparison and optimization.

Acknowledgment

The work described in this report was significantly aided by helpful discussions with Brian Adams, Malcolm Carroll, Nathan Bishop, Mark Friesen, Michael Lilly, Michael Phipps, Rajib Rahman, Mike Stopa, Laura Swiler, and Jason Verley, and by project management help from Robert Hoekstra and John Aidun.

This work was funded under LDRD Project Number 151297 and Title "Integrated Nano- and Quantum Electronic Device Simulation Toolkit".

Contents

1	Introduction	15
2	QCAD Software Structure	17
3	QCAD Semiclassical Poisson Solver	21
	Carrier Statistics	21
	The Reference Potential	24
	Incomplete Ionization	25
	Boundary Conditions	27
4	QCAD Schrodinger-Poisson Solver	33
	Schrodinger Solver	33
	Self-Consistent Schrodinger-Poisson Solver	36
	Quantum Electron Density	36
	Self-Consistency	39
	Validation Example	42
5	QCAD User Guide	47
	Introduction	47
	What is QCAD?	47
	How do I get QCAD?	48
	How do I run QCAD?	48
	Input files	48
	Main XML input file specification	49

The Skeleton	49
The Problem list	50
The Poisson Problem and Schrodinger Problem lists	51
Parameters and Responses	56
The Discretization list	70
The Piro list	71
Material file specification	72
Mesh file specification	77
Output Files	77
QCAD’s “multi-shot” mode	77
How to run QCAD in “multi-shot” mode	78
DAKOTA input file examples	79
Simple examples	80
2D Poisson problem	80
2D Schrodinger problem	81
6 QCAD Developer Guide	83
Albany and Trilinos Primer	83
Trilinos Overview	83
Model Evaluators	83
Albany Overview	84
Albany::SolverFactory	84
Albany::Application	84
Problems	85
Evaluators	85
Responses	85
QCAD code structure	86

Overview	86
Problems	86
Poisson Problem	86
Schrodinger Problem	87
Mesh Regions	87
Responses	88
Field Integral	88
Field Average	88
Field Value	88
Center Of Mass	89
Save Field	89
Region Boundary	89
Saddle Value	89
Coupled Solvers: QCAD::Solver	90
Integrated P-S Solver: QCAD::CoupledPoissonSchrodinger	91
Known limitations / possible extensions	92
7 QCAD Performance Studies	93
Introduction	93
Devices Considered	93
Meshes Considered (and Meshing Methodology)	94
Architectures Considered	96
Preconditioners Considered	96
Mesh Convergence Study	99
Ottawa Flat 270 Geometry	99
Mosdot3D Geometry	101
Preconditioner Performance Studies	104

Ottawa Flat 270 Geometry	104
Mosdot3D Geometry	107
Scalability Studies	109
References	111

List of Figures

2.1	Schematic of structure of different QCAD solvers, Albany, and, various solver and optimization packages.	18
2.2	We can form sheets (“e-”) of electrons at a MOS interface using an accumulation gate, using a positive (“+”) voltage. By introducing additional depletion gates with negative (“-”) voltage, we can deplete most of this sheet, leaving puddles that form quantum dots.	18
2.3	Different shapes of depletion gates. Each colored in the left, middle, and right figures is a metal or polysilicon gate that can be set to a different voltage to form a quantum dot.	19
2.4	Optimization of the Ottawa Flat 270 structure. The left figure shows the depletion gate configuration for the Ottawa Flat 270 structure, and the right figure shows the resulting optimization using the Nonlinear Poisson solver, with a variety of constraints detailed in the text.....	20
3.1	(Color online) Fermi-Dirac integral of 1/2 order in logarithmic scale. Black curves are the asymptotic expansions, $\exp(x)$ for $x \rightarrow -\infty$, and $\frac{4x^{3/2}}{3\sqrt{\pi}}$ for $x \rightarrow +\infty$. The red solid curve labeled as Approximate is obtained using Eq. (3.7), while the blue dashed curve labeled as Numerical uses the numerical integration method in Ref. [?]. The red solid and blue dashed curves do not show any visible difference, and agree well with the asymptotic forms for large $ x $, whereas they differ significantly from the asymptotic forms in the region of $x \in (-4, 4)$	23
3.2	Schematics of the band structure of a MOS-type device under zero bias.	25
3.3	The derivation of the Robin BC parameters used to set a value of the potential and a nearby surface charge on a single surface. This diagram shows a 1D cut along the direction \hat{n} normal to a surface element. ϕ_0 , σ_s , and d are given values, and ϕ is the variable being solved. The derivative ϕ' is along the direction normal to the surface. Combining the equations yields $\epsilon_s \phi' = \epsilon_s \frac{\phi_0 - \phi}{d} - q\sigma_s$, which takes the form of a Robin boundary condition.	31

4.1	(Color online) (a) Wave functions and energies obtained from the QCAD Schrodinger solver for a 1D parabolic potential well. (b) Analytic wave functions and energies for the same potential well. All the wave functions are scaled by the same factor for easy visualization. It is clear that QCAD wave functions and energies agree very well with the analytic results.	34
4.2	(Color online) (a) Lowest three wave functions and energies obtained from the QCAD Schrodinger solver for the 1D finite potential well with $m_b^* = m_w^* = 0.067m_0$ where m_0 is the free electron mass. (b) Same as (a) except $m_b^* = m_0$ and $m_w^* = 0.067m_0$. All wave functions are scaled by the same factor for easy visualization.	35
4.3	(Color online) Δ_2 -valley lowest four subband wave functions and energies in a 1D MOS Si capacitor at $T = 50$ K and $V_g = 3$ V obtained from QCAD (a) and from SCHRED (b). The Si/SiO ₂ interface is located at $x = 0$. The solid and dashed curves are obtained without and with the exchange-correlation effect, respectively. The subband energies in [meV], referenced from the Fermi level and including the V_{xc} effect, are denoted by E1 <i>i</i> , where the “1” indicates the Δ_2 -valley and <i>i</i> indexes the subband (SCHRED’s labeling convention). For comparison, the corresponding energies without V_{xc} are -72.54, 26.71, 90.73, 144.69 for QCAD, and -71.76, 26.12, 89.22, 142.27 for SCHRED.	43
4.4	(Color online) Schematic diagram of the simulated 2D structure with all dimensions given in nm. The blue 2D contour in the Si quantum region shows the Δ_2 -valley lowest subband wave function obtained from QCAD without the V_{xc} effect at $T = 10$ K, $V_{g1} = 0.8$ V, and $V_{g2} = 3.5$ V with all voltages referred to flat band. The dash line denotes the $y = -2$ nm location.	44
4.5	(Color online) Δ_2 -valley lowest five subband wave functions along the $y = -2$ nm dash line in Fig. 4.4. These wave functions agree very well with Fig. 4(a) in Ref. [?].	45
4.6	(Color online) Δ_2 -valley lowest three subband energies as a function of integrated electron density in the Si quantum region. Black curves plot the data extracted from Fig. 3 in Ref. [?], while the red squares are the data obtained from QCAD. The energies are with respect to the Fermi level which is set to 0.	45
4.7	(Color online) Convergence behavior of the self-consistent QCAD S-P solver for the 1D MOS Si capacitor and the 2D gate-induced Si quantum wire devices.	46
7.1	Device Geometries Considered.	94
7.2	Mesh refinement.	95
7.3	Illustration of first-order (TETRA4) and second-order (TETRA10) tetrahedral finite elements.	96

7.4	Mesh quality (in “scale” metric) for Ottawa Flat 270 geometry	100
7.5	Convergence plot for TETRA4 finite elements for Ottawa Flat 270 geometry . .	101
7.6	Convergence plot for various finite elements for Mosdot3D geometry	104
7.7	Ifpack vs. ML preconditioners with ML settings A	105
7.8	Ifpack vs. ML preconditioners with ML settings B	106
7.9	Ifpack vs. ML preconditioners with ML settings C	107
7.10	Weak scalability plot for TETRA4 elements + ML preconditioner (16, 128, 1024 processors on Red Sky)	110

Chapter 1

Introduction

The next generation of semiconductor devices will have to confront quantum mechanical effects. These include both phenomena to be avoided, like gate leakage, as well as new behavior that can be harnessed, like entanglement. Few-electron nanodevices have been developed to use entanglement in quantum computing and sensing beyond the traditional quantum limits, but the resulting entangled device states are extremely sensitive to atomic scale effects such as surface roughness that are not traditionally considered in nanoelectronics modeling.

We have developed a robust and efficient simulator that solves for semiclassical and self-consistent quantum electrostatic potential, single- and multi-electron wave functions and energies at near-zero temperatures, and that can be used to simulate and optimize many different quantum dot structures very efficiently and provide fast feedback on which device layouts are more likely to lead to few-electron behavior. Although there have been numerous simulation papers [?, ?, ?, ?, ?, ?] on quantum dots in literature, they focus on addressing different issues than what we were trying to resolve. Existing commercial [?] and academic [?] device simulators either target room-temperature and many-electron devices, whereas our applications require temperatures close to zero Kelvin and one/few-electron devices, or target simple and few geometries, whereas our DQD devices have very complex three-dimensional (3D) shapes and can have many different layouts due to the inherent large design space of DQDs. And they often have license and platform restrictions which seriously limit simulation efficiency. The Sandia Quantum Computer Aided Design (QCAD) project is developing an integrated open-source toolkit that serves as the simulator of imperative need, by addressing the challenges associated with modeling realistic DQDs, including complex geometries, many device layouts, low temperature operation, and 3D quantum confinement effects, to accelerate the development of few-electron DQD qubit work at Sandia. The QCAD toolkit leverages a number of Sandia-developed software programs [?], including the Trilinos suite, the Albany code [?], the Dakota toolbox, and the Cubit geometry and meshing tool.

The QCAD simulator [?, ?] is built upon the Albany code [?] and contains three core modules of Poisson (P), Schrodinger (S), and Configuration Interaction (CI) solvers. These physical solvers can be run individually or combined self-consistently (i.e., self-consistent S-P and S-P-CI solvers) for simulating arbitrary 1D/2D/3D quantum devices made from multiple different materials. They have demonstrated fast and robust convergence behavior even at very low temperatures. Furthermore, very high simulation throughput has been

achieved by using a combination of pre- and post-processing scripting, automated structure creation and meshing, distributed parallel computing capability and resources. For example, we can typically obtain simulation results for dozens of experimental DQDs overnight, compared to several days of simulation on one structure using a commercial TCAD tool in the past. The QCAD solvers enable us to compute capacitances of experimental importance such as quantum dot-to-gate capacitances. Comparison of calculated capacitances with measurement provides certain insight regarding the shape and location of a dot and possible locations of defect charges. In addition, by comparing the results obtained using the semiclassical P solver and the self-consistent quantum S-P solver respectively, we can investigate how quantum spatial confinement influences the capacitances. We observed that quantum confinement enhances the dot-to-gate capacitances when the number of electrons stays fixed in a quantum dot. Another powerful component of QCAD is its coupling with the optimization driver Dakota, which enables optimization of gate voltages in many DQD devices to achieve multiple design targets simultaneously, and helps to identify which device designs are more likely exhibiting few-electron quantum dot behavior.

Chapter 2

QCAD Software Structure

The QCAD toolkit leverages a number of Sandia-developed software programs [?], including the Trilinos suite, the Albany code [?], the Dakota toolbox, and the Cubit geometry and meshing tool to simulate the electronic structure of quantum dots to determine their utility as qubits in quantum computing devices. A schematic of the structure of the different software components is shown in figure 2.1.

Quantum dots are a region of a semiconductor where the local electrostatics allow a “puddle” of electrons to form, typically near a semiconductor-insulator interface. We often use a silicon metal-oxide-semiconductor (MOS) system, with an additional level of gates in the insulator to deplete the sheet into the puddles that form the quantum dot, as shown in figure 2.2. The depletion gates themselves, shown in cross-section in figure 2.2, have considerable structure when shown in a top view, as in figure 2.3. The quantum effects we wish to use to form qubits are most pronounced with few numbers of electrons, and a major challenge is to design robust enough systems that can form few-electron dots, which often involves modifying the shapes of the gates and the spacings between different layers.

As figure 2.1 shows, we are primarily interested three different solvers. An important component of all solvers is a linear Poisson’s equation solver that determines the electrostatic potential that results from the gate voltages and other device parameters. The Nonlinear Poisson solver uses a series of call to the linear Poisson solver to treat electrons semiclassically, that is, as classical particles that obey quantum (Fermi-Dirac) statistics. The Schrodinger-Poisson solver uses a similar set of calls, but performs a fully quantum mechanical solution to the electrons by solving a one-particle Schrodinger-Equation. The Configuration Interaction solver takes single-particle solutions from the Schrodinger-Poisson solver and determines multi-electron solutions.

Several aspects of the Albany framework make it straightforward to implement a physics package. A heat-flow problem had already been implemented, and thus the basics of solving the linear Poisson equation had already been implemented, and this code could serve as a model for the solvers we would add to QCAD. The ability to code physics equations easily with automatic differentiation, which was something that we did not anticipate would be of much use, turned out to make a major difference, as it meant that, for the most part, we could code only the basic physics equations, without needing to derive, code, or debug the derivatives, yet still having full access to a wide range of solvers that offered robust convergence. In much the same way, parallel code was obtained with no additional effort;

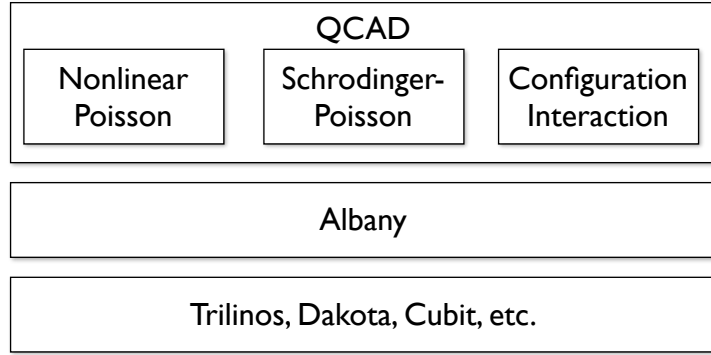


Figure 2.1. Schematic of structure of different QCAD solvers, Albany, and, various solver and optimization packages.

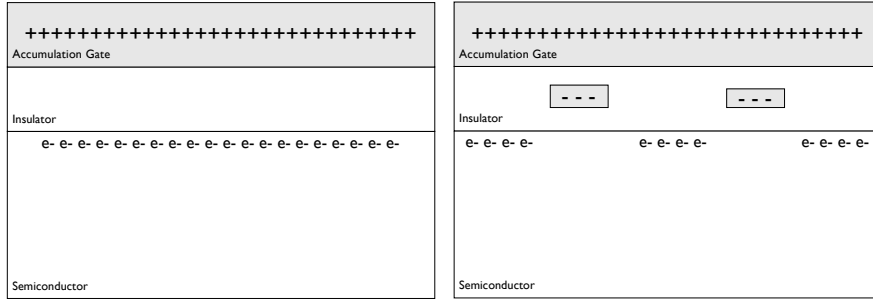


Figure 2.2. We can form sheets (“e-”) of electrons at a MOS interface using an accumulation gate, using a positive (“+”) voltage. By introducing additional depletion gates with negative (“-”) voltage, we can deplete most of this sheet, leaving puddles that form quantum dots.

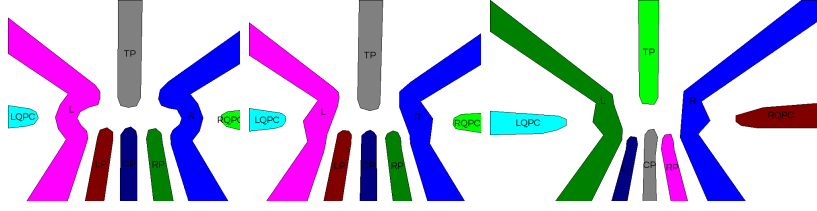


Figure 2.3. Different shapes of depletion gates. Each colored in the left, middle, and right figures is a metal or polysilicon gate that can be set to a different voltage to form a quantum dot.

we programmed serial versions of our methods, and the Albany libraries and solvers insured that we obtained highly parallel applications.

Another key element to the choice of the Albany framework was the additional packages that it presented to us. Albany had a sophisticated finite-element analysis capability, with a variety of different elements present. We used the Cubit [?] solid modeler and mesher to build our geometries and meshes, and Albany was able to import these structures using the ExodusII file format. Many different solvers in the Trilinos package [?] are available. Finally, the Dakota optimization package [?], via the TriKota interface, is available and provided a variety of sophisticated optimization options that have been extremely useful in optimizing sophisticated targets (see below).

An example of the type of optimization we performed is shown in figure 2.4. We wished to optimize a quantum dot containing two electrons, with tunable tunnel barriers in and out of the dot region, between the left and right electrons of the dot, and with the channels on the sides that are used as electrometers also having tunable tunnel barriers. We optimized the voltages on all gates, with the left/right symmetry in the gate voltages imposed as an additional constraint. The right side of figure 2.4 shows the resulting electron density computed using the Nonlinear Poisson solver. The red field in the background is the “sheet” of electrons, and the blue gates are the depleted images of the depletion gates. The quantum dot itself is the narrow curved region underneath the gate labeled TP at the right.

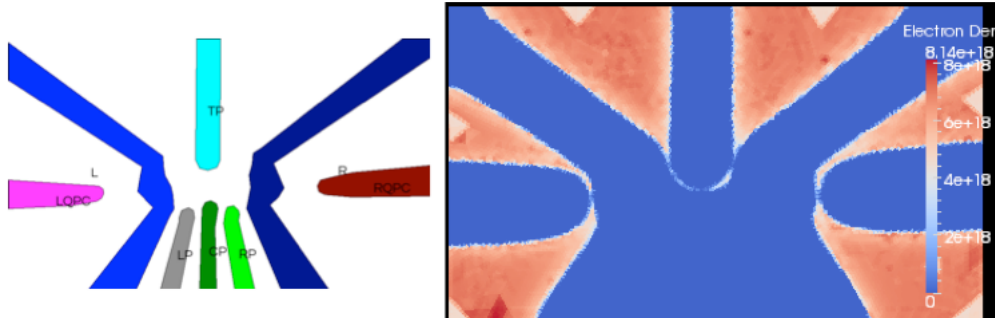


Figure 2.4. Optimization of the Ottawa Flat 270 structure. The left figure shows the depletion gate configuration for the Ottawa Flat 270 structure, and the right figure shows the resulting optimization using the Nonlinear Poisson solver, with a variety of constraints detailed in the text.

Chapter 3

QCAD Semiclassical Poisson Solver

The well-known Poisson equation in a bulk semiconductor is given by

$$-\nabla \cdot (\epsilon_s \nabla \phi) = q(p - n + N_D^+ - N_A^-), \quad (3.1)$$

where ϕ is the electrostatic potential, ϵ_s is the static permittivity, q is the elementary charge, n and p are the electron and hole concentrations respectively, N_D^+ and N_A^- are ionized donor and acceptor concentrations respectively. Note that the form of the left hand side in Eq. (3.1) allows ϵ_s to have spatial dependence.

Carrier Statistics

n and p are given by carrier statistics for bulk (spatially unconfined) semiconductors. Both Maxwell-Boltzmann (MB) and Fermi-Dirac (FD) statistics are implemented in QCAD. For the MB statistics, n and p take the exponential forms,

$$\begin{aligned} n &= N_C \exp\left(\frac{E_F - E_C}{k_B T}\right), \\ p &= N_V \exp\left(\frac{E_V - E_F}{k_B T}\right), \end{aligned} \quad (3.2)$$

where k_B is the Boltzmann constant, T is the lattice temperature, E_C and E_V are the conduction and valence band edge respectively, and E_F is the extrinsic Fermi level (more details on E_C , E_V , and E_F are given in Sec. 3). For the FD statistics, n and p are expressed in terms of the Fermi-Dirac integrals (see Appendix A for the derivation),

$$\begin{aligned} n &= N_C \mathcal{F}_{1/2}\left(\frac{E_F - E_C}{k_B T}\right), \\ p &= N_V \mathcal{F}_{1/2}\left(\frac{E_V - E_F}{k_B T}\right). \end{aligned} \quad (3.3)$$

N_C and N_V are effective density of states (DOS) in the conduction and valence band, respectively. Assuming parabolic band structure, we have

$$\begin{aligned} N_C &= 2 \left(\frac{m_n^* k_B T}{2\pi\hbar^2} \right)^{3/2}, \\ N_V &= 2 \left(\frac{m_p^* k_B T}{2\pi\hbar^2} \right)^{3/2}, \end{aligned} \quad (3.4)$$

where \hbar is the reduced Planck constant, m_n^* and m_p^* are respectively the electron and hole DOS effective mass including all equivalent band minima. For bulk silicon, there are six equivalent conduction minima, and the valence band minimum is degenerate including heavy hole and light hole bands at the Γ valley, hence

$$\begin{aligned} m_n^* &= 6^{2/3} (m_l m_t^2)^{1/3}, \\ m_p^* &= \left(m_{hh}^{3/2} + m_{lh}^{3/2} \right)^{2/3}, \end{aligned} \quad (3.5)$$

with m_l , m_t , m_{hh} , and m_{lh} being the electron longitudinal, electron transverse, heavy hole, and light hole effective mass, respectively.

The $\mathcal{F}_{1/2}(x)$ function in Eq. (3.3) is the Fermi-Dirac integral of 1/2 order and is defined as [?]

$$\mathcal{F}_{\frac{1}{2}}(x) = \frac{2}{\sqrt{\pi}} \int_0^\infty \frac{\sqrt{\varepsilon} d\varepsilon}{1 + \exp(\varepsilon - x)}. \quad (3.6)$$

Although the closed form of this integral can be formally expressed by the polylogarithm function[?] or by a complete expansion discussed in Ref. [?], the polylogarithm function and the complete expansion involve summations of infinite series. Hence in practice, one has to either use certain approximation to obtain a computable analytic expression, or use numerical integration techniques[?, ?]. There have been a few approximate analytic forms proposed in literature [?, ?, ?] that offer relatively simple expressions and sufficient accuracy. Among them, the approximate expression in Ref. [?] takes a single simple form and provides a relative error less than 0.4% for $x \in (-\infty, +\infty)$, hence has been widely used in the device modeling community [?]. The expression in Ref. [?] is given as

$$\begin{aligned} \mathcal{F}_{\frac{1}{2}}(x) &\approx \left(e^{-x} + \frac{3\sqrt{\pi}}{4} v^{-3/8} \right)^{-1}, \\ v &= x^4 + 50 + 33.6x(1 - 0.68 \exp[-0.17(x+1)^2]). \end{aligned} \quad (3.7)$$

The asymptotic expansion at $x \rightarrow -\infty$ leads to $\mathcal{F}_{1/2}(x) = \exp(x)$, implying that Eq. (3.3) becomes equivalent to Eq. (3.2), which is the case for non-degenerate semiconductors where

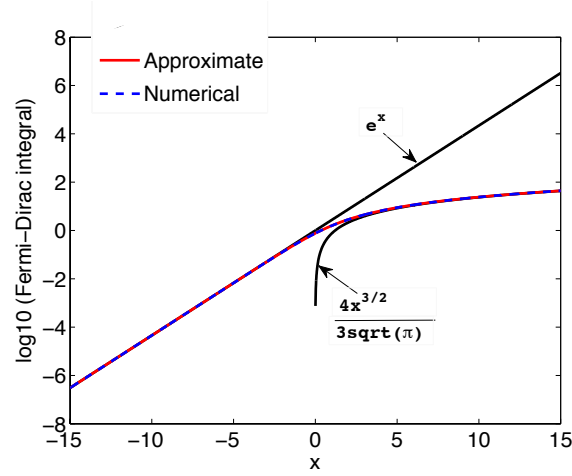


Figure 3.1. (Color online) Fermi-Dirac integral of 1/2 order in logarithmic scale. Black curves are the asymptotic expansions, $\exp(x)$ for $x \rightarrow -\infty$, and $\frac{4x^{3/2}}{3\sqrt{\pi}}$ for $x \rightarrow +\infty$. The red solid curve labeled as Approximate is obtained using Eq. (3.7), while the blue dashed curve labeled as Numerical uses the numerical integration method in Ref. [?]. The red solid and blue dashed curves do not show any visible difference, and agree well with the asymptotic forms for large $|x|$, whereas they differ significantly from the asymptotic forms in the region of $x \in (-4, 4)$.

$E_F \ll (E_C - [\text{a few } k_B T])$ for n , and $E_F \gg (E_V + [\text{a few } k_B T])$ for p . The asymptotic form at $x \rightarrow +\infty$ is $\mathcal{F}_{1/2}(x) = \frac{4x^{3/2}}{3\sqrt{\pi}}$, which corresponds to the strongly degenerate case near 0 K, where $E_F \gg (E_C + [\text{a few } k_B T])$ for n (i.e., the Fermi level is located within the conduction band), and $E_F \ll (E_V - [\text{a few } k_B T])$ for p (i.e., the Fermi level is inside the valence band). Figure 3.1 shows a comparison of the 1/2-order Fermi-Dirac integral calculated by different methods. It is clear that the approximate expression in Eq. (3.7) produces visually the same result as the numerical approach [?], and follows the proper asymptotic forms for large $|x|$. In the small $|x|$ regime, neither of the asymptotic forms is valid. Since semiconductor DQD qubits are currently operated in this regime (corresponding to very low temperatures, mK to a couple of K), it is important to adopt a sufficiently accurate evaluation of the Fermi-Dirac integral. Due to the good accuracy and simplicity of Eq. (3.7), we implemented this form in QCAD for the FD statistics. In the actual implementation, we approximate $\mathcal{F}_{1/2}(x)$ by e^x for $x < -50$ to avoid numerical instability caused by the e^{-x} term in Eq. (3.7). Such large negative values of x can occur at very low temperatures, and this approximation results in no discernible loss of accuracy, as shown in Fig. 3.1.

The Reference Potential

Before solving the Poisson Eq. (3.1) for the electrostatic potential ϕ , one needs to relate ϕ to the band energies of the materials making up the device. One requirement for the electrostatic potential is that it must be continuous everywhere in a device. For a homo-junction device such as a PN silicon diode, E_C , E_V , and E_i (the intrinsic Fermi level) as functions of position are parallel to each other and continuous across the device, so it is natural to choose $-q\phi = E_i$, i.e., to solve for the inverse of intrinsic Fermi level. In an arbitrary hetero-junction structure, however, E_C , E_V , and E_i could all be discontinuous. Figure 3.2 shows a schematic of the band structure of a MOS-type device under zero bias illustrating the discontinuity of E_C and E_V . What is always continuous in arbitrary homo- and hetero-junction devices is the vacuum level indicated as E_0 in Fig. 3.2. Therefore, we choose ϕ to satisfy [?]

$$-q(\phi - \phi_{ref}) = E_0 = E_C + \chi, \quad (3.8)$$

where ϕ_{ref} is a constant reference potential and χ is the electron affinity of a material. This choice implies that we are solving for the inverse of the vacuum level shifted by a constant value. While in theory different ϕ_{ref} values only change the resulting solution, ϕ , by a constant offset, in practice they can lead to different numerical convergence behavior during simulation. A good choice of ϕ_{ref} that has shown numerical robustness in devices containing silicon is to select as the reference potential the intrinsic Fermi level of silicon relative to the vacuum level, i.e., $q\phi_{ref} = E_0 - E_i(\text{Si})$. For a more detailed explanation of band diagrams schematics such as that in Fig. 3.2, we refer the reader to Ref. [?].

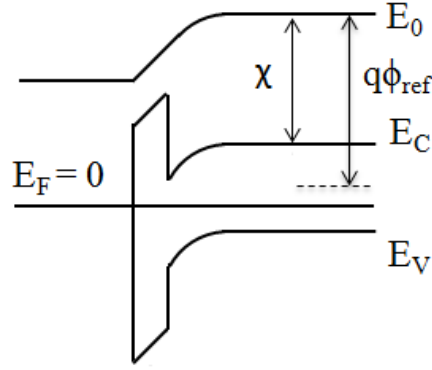


Figure 3.2. Schematics of the band structure of a MOS-type device under zero bias.

With the choice of ϕ in Eq. (3.8), we can rewrite E_C and E_V as

$$\begin{aligned} E_C &= -q(\phi - \phi_{ref}) - \chi, \\ E_V &= -q(\phi - \phi_{ref}) - \chi - E_g, \end{aligned} \quad (3.9)$$

with E_g being the band gap of a material. Then the n and p in Eq. (3.1) becomes a function of ϕ only, i.e., $n(\phi)$ and $p(\phi)$, as χ and E_g are material-dependent parameters and known for most semiconductors.

As QCAD does not solve the carrier transport (e.g., drift-diffusion) equations, all calculations must assume that thermal equilibrium (zero current flow) has been attained. The Fermi level, E_F , is taken to be a constant throughout any electrically-connected region of a device. The value of this constant is set by the voltages applied to the device. For example, if a voltage V_{sub} is applied to the substrate (right side) of the device in Fig. 3.2, E_F will become $-qV_{sub}$. The band structure shown in Fig. 3.2 corresponds to a MOS-type structure with metal gate (left side). If the structure instead had a highly-doped semiconductor gate (e.g., n^+ polysilicon gate) with applied voltage V_g , E_F in the gate region would be $-qV_g$ while E_F in the substrate region remains at $-qV_{sub}$.

Incomplete Ionization

When impurities are introduced into the semiconductor crystals, depending on the impurity energy level and the lattice temperature, not all dopants are necessarily ionized, especially at very low lattice temperatures (where DQD qubits are commonly operated).

The ionized concentration for donors and acceptors is given by [?]

$$\begin{aligned} N_D^+ &= \frac{N_D}{1 + g_D \exp\left(\frac{E_F - E_D}{k_B T}\right)}, \\ N_A^- &= \frac{N_A}{1 + g_A \exp\left(\frac{E_A - E_F}{k_B T}\right)}, \end{aligned} \quad (3.10)$$

where E_D is the donor energy level, g_D is the donor ground state degeneracy factor, E_A is the acceptor energy level, and g_A is the acceptor ground state degeneracy factor. g_D is equal to 2 because a donor level can accept one electron with either spin or can have no electron when filled. g_A is equal to 4 because in most semiconductors each acceptor level can accept one hole of either spin and the impurity level is doubly degenerate as a result of the two degenerate valence bands (heavy hole and light hole bands) at the Γ point.

To write N_D^+ and N_A^- as a function of ϕ , we need to rewrite $E_F - E_D = E_F - E_C + E_C - E_D = E_F - E_C + E_d$ with E_d being the donor ionization energy, and $E_A - E_F = E_A - E_V + E_V - E_F = E_a + E_V - E_F$ with E_a being the acceptor ionization energy. The most common donors in bulk Si are phosphorus (P) and arsenic (As), which have ionization energies of $E_d = 46$ meV and 54 meV respectively[?]. The most common acceptor dopant in bulk Si is boron (B), which has $E_a = 44$ meV[?].

Substituting Eq. (3.9) and $E_F - E_D$, $E_A - E_F$ into Eq. (3.10), we get [?]

$$\begin{aligned} N_D^+ &= \frac{N_D}{1 + g_D \exp\left(\frac{E_F + E_d - q\phi_{ref} + \chi + q\phi}{k_B T}\right)}, \\ N_A^- &= \frac{N_A}{1 + g_A \exp\left(\frac{-E_F + E_a + q\phi_{ref} - \chi - E_g - q\phi}{k_B T}\right)}. \end{aligned} \quad (3.11)$$

With these expressions, N_D^+ and N_A^- also become a function of ϕ , i.e., $N_D^+(\phi)$ and $N_A^-(\phi)$. Hence, the entire right hand side (RHS) of Eq. (3.1) can be written as a nonlinear function of ϕ . Applying integration by parts and divergence theorem, we then rewrite the equation into the finite element (FE) weak form,

$$\begin{aligned} &\int \epsilon_s \nabla \phi \cdot \nabla w d\Omega - \int_{\Gamma} \epsilon_s \nabla \phi \cdot \hat{\eta} w d\Gamma \\ &- \int q[p(\phi) - n(\phi) + N_D^+(\phi) - N_A^-(\phi)] w d\Omega = 0, \end{aligned} \quad (3.12)$$

where w is the FE nodal basis function and the second term is a line integral over the simulation domain boundary with $\hat{\eta}$ being the unit normal vector of the surface element $d\Gamma$. The weak form is discretized using the Trilinos/Intrepid library, and the resulting discrete

equation is solved by a nonlinear Newton solver also in Trilinos. Both the discretization library and the Newton solver were made directly available to QCAD through the Albany framework (cf. Fig. 2.1).

Boundary Conditions

An essential ingredient to the formulation of a differential equation are boundary conditions (BCs). QCAD supports three types of BCs: Dirichlet, Neumann, and Robin BCs. We will next discuss the implementation of these types in turn.

Dirichlet BCs are divided into two cases: (1) setting a voltage on the surface of a metallic region that borders insulator, and (2) setting a voltage on the surface of an Ohmic contact region which borders semiconductor. Case (1) is used for gate electrodes in field effect transistor (FET)-like devices, and the Dirichlet BC value ϕ_{ins} on the bordering insulator(s) is given by the simple expression

$$\phi_{ins} = V_g - \frac{\Phi_m - q\phi_{ref}}{q}, \quad (3.13)$$

with V_g being the applied gate voltage and Φ_m being the metal work function.

In the second case (used for Ohmic contacts in semiconductors), the potential on the bordering semiconductor surfaces is computed assuming thermal equilibrium and charge neutrality at the contacts. The calculation depends on carrier statistics and dopant ionization. For MB statistics, the charge neutrality $n + N_A^- = p + N_D^+$ condition leads to

$$\begin{aligned} & N_C \exp\left(\frac{E_F + q\phi - q\phi_{ref} + \chi}{k_B T}\right) + N_A^- \\ &= N_V \exp\left(\frac{-E_F - q\phi + q\phi_{ref} - \chi - E_g}{k_B T}\right) + N_D^+. \end{aligned} \quad (3.14)$$

With complete ionization of dopants (i.e., $N_A^- = N_A$ and $N_D^+ = N_D$), the potentials at n-type and p-type Ohmic contacts are respectively given by

$$\begin{aligned} \phi_{ohm}^n &= \frac{q\phi_{ref} - \chi}{q} + \frac{k_B T}{q} \ln\left(\frac{N_D}{N_C}\right) + V_a, \\ \phi_{ohm}^p &= \frac{q\phi_{ref} - \chi - E_g}{q} - \frac{k_B T}{q} \ln\left(\frac{N_A}{N_V}\right) + V_a, \end{aligned} \quad (3.15)$$

where V_a is an externally applied voltage. When including incomplete ionization effect of

dopants, we have for n-type and p-type semiconductors respectively,

$$\begin{aligned} N_C \exp\left(\frac{E_F - E_C}{k_B T}\right) &= \frac{N_D}{1 + 2 \exp\left(\frac{E_F - E_D}{k_B T}\right)}, \\ N_V \exp\left(\frac{E_V - E_F}{k_B T}\right) &= \frac{N_A}{1 + 4 \exp\left(\frac{E_A - E_F}{k_B T}\right)}. \end{aligned} \quad (3.16)$$

Let us denote $y_n = \exp(\frac{E_F - E_D}{k_B T})$ and $y_p = \exp(\frac{E_A - E_F}{k_B T})$. Then, by the definitions of E_a and E_d , we obtain the identities, $\exp(\frac{E_F - E_C}{k_B T}) = y_n \exp(\frac{-E_d}{k_B T})$ and $\exp(\frac{E_V - E_F}{k_B T}) = y_p \exp(\frac{-E_a}{k_B T})$. Substituting the identities into Eq. (3.16), we obtain

$$\begin{aligned} y_n &= -\frac{1}{4} + \frac{1}{4} \left[1 + \frac{8N_D}{N_C} \exp\left(\frac{E_d}{k_B T}\right) \right]^{1/2}, \\ y_p &= -\frac{1}{8} + \frac{1}{8} \left[1 + \frac{16N_A}{N_V} \exp\left(\frac{E_a}{k_B T}\right) \right]^{1/2}. \end{aligned} \quad (3.17)$$

From the definitions of y_n and y_p , the use of $-qV_a = E_F$, and Eq. (3.9), we can obtain the potentials that include dopant incomplete ionization effect at the n-type and p-type Ohmic contacts respectively as,

$$\begin{aligned} \phi_{ohm}^n &= \frac{-E_d + q\phi_{ref} - \chi}{q} + \frac{k_B T}{q} \ln(y_n) + V_a, \\ \phi_{ohm}^p &= \frac{-E_a + q\phi_{ref} - \chi - E_g}{q} - \frac{k_B T}{q} \ln(y_p) + V_a. \end{aligned} \quad (3.18)$$

At very low temperatures, the exponential terms in Eq. (3.17) could blow up numerically. To avoid numerical instability in QCAD, we approximate the $\ln(y_n)$ and $\ln(y_p)$ terms for very low temperatures as,

$$\begin{aligned} \ln(y_n) &= \frac{1}{2} \ln\left(\frac{N_D}{2N_C}\right) + \frac{E_d}{2k_B T}, \\ \ln(y_p) &= \frac{1}{2} \ln\left(\frac{N_A}{4N_V}\right) + \frac{E_a}{2k_B T}. \end{aligned} \quad (3.19)$$

For FD statistics and assuming complete ionization of dopants, we have

$$\begin{aligned} N_C \mathcal{F}_{\frac{1}{2}}\left(\frac{E_F - E_C}{k_B T}\right) &= N_D \text{ for n-type ,} \\ N_V \mathcal{F}_{\frac{1}{2}}\left(\frac{E_V - E_F}{k_B T}\right) &= N_A \text{ for p-type .} \end{aligned} \quad (3.20)$$

To solve for the potentials at Ohmic contacts, in principle, we need to numerically solve Eq. (3.20) as the Fermi-Dirac integral does not have an analytic result. In QCAD, we use an approximate expression for the inverse of the 1/2 order Fermi-Dirac integral, that is, given $u = \mathcal{F}_{1/2}(\eta)$, η is computed as [?],

$$\begin{aligned} \eta &= \frac{-\ln(u)}{u^2 - 1} + \frac{\nu}{1 + (0.24 + 1.08\nu)^{-2}}, \\ \nu &= \left(\frac{3\sqrt{\pi}u}{4}\right)^{2/3}. \end{aligned} \quad (3.21)$$

This approximation has an error of less than 0.6% for the entire η range. Using this expression, we obtain the BC potentials as,

$$\begin{aligned} \phi_{ohm}^n &= \frac{q\phi_{ref} - \chi}{q} + \frac{k_B T}{q} \eta + V_a, \\ \phi_{ohm}^p &= \frac{q\phi_{ref} - \chi - E_g}{q} - \frac{k_B T}{q} \eta + V_a, \end{aligned} \quad (3.22)$$

with η given in Eq. (3.21) where $u = N_D/N_C$ for n-type and $u = N_A/N_V$ for p-type. For the case of FD statistics and incomplete ionization, there exists no approximate analytic expressions for the BC potentials, and one has to solve a non-trivial nonlinear equation if want to be very accurate. In QCAD, we approximate this case using MD statistics with incomplete ionization and utilize the BC potentials given in Eq. (3.18).

Neumann BCs in finite element methods are used to specify how “flux” is conserved across boundaries. By default, all boundaries that are not given any other type of boundary conditions, assume implicit Neumann BCs which preserve the flux. In the case of the Poisson equation, the flux is $\epsilon_s \nabla \phi \cdot \hat{\eta}$, where $\hat{\eta}$ is the unit normal of the boundary surface. Thus, by default (i.e. when no other boundary condition is specified), $\epsilon_s \nabla \phi \cdot \hat{\eta} = 0$ on outer boundaries of the finite element mesh and $\epsilon_{s1} \nabla \phi_1 \cdot \hat{\eta}_1 = \epsilon_{s2} \nabla \phi_2 \cdot \hat{\eta}_2$ on internal boundaries. These two conditions are automatically satisfied in the finite element framework by setting the $\int \epsilon_s \nabla \phi \cdot \hat{\eta} d\Gamma$ term to 0 in Eq. (3.12)

QCAD has the ability to specify non-flux-conserving Neumann BCs on specific boundaries such that the difference between the fluxes on either side of the boundary are equal to some specified constant value. Written mathematically, $(\epsilon_{s2} \nabla \phi_2 - \epsilon_{s1} \nabla \phi_1) \cdot \hat{\eta} = q\sigma_s$, where $\hat{\eta}$ is

the unit normal vector of the interface pointing from material 2 to 1 and σ_s is the specified constant. Physically, σ_s is a surface charge density located at the boundary. Note that when $\sigma_s = 0$ (i.e. no surface charge), the boundary condition reduces to the default flux-conserving condition. Within the finite element discretization in QCAD, this type of Neumann BC is implemented in the integral form

$$\int_{\Gamma_{cbc}} (\epsilon_{s2} \nabla \phi_2 - \epsilon_{s1} \nabla \phi_1) \cdot \hat{\eta} w d\Gamma_{cbc} = \int_{\Gamma_{cbc}} q \sigma_s w d\Gamma_{cbc}. \quad (3.23)$$

A major shortcoming of the Neumann BCs is their inability to characterize surface charge on (or extremely close to) an interface whose voltage is set by a Dirichlet BC. This is due to the simple fact that specifying both Dirichlet and Neumann BCs on the same surface overdetermines the problem. Yet, this is essentially what is needed to model a layer of charge that is stuck to one of the conducting gates (often polysilicon) used to control a device. One way around this technical difficulty is to place a layer of very thin finite-element cells around the charged gate and set a Neumann BC on the new surface lying a small distance away from the gate itself. This approach, however, suffers due to the thin finite elements adversely affecting convergence and their being hard to create in the first place. Instead, we use what are called Robin boundary conditions[?] to address the issue of charged gates. Robin BCs are similar to Neumann BCs but allow the flux at a surface to depend on the value of the solution (in this case the potential) there. Specifically, the Robin BC for an internal surface element can be written $(\epsilon_{s2} \nabla \phi_2 - \epsilon_{s1} \nabla \phi_1) \cdot \hat{\eta} = C + \alpha(\phi_2 - \phi_1)$, where C and α are fixed constants. At an external surface, we have $\epsilon_s \nabla \phi \cdot \hat{\eta} = C + \alpha \phi$. We would like to roll into a single boundary condition, a Dirichlet condition at one surface followed by a Neumann boundary condition at a parallel surface lying a very small distance away from the first surface. This can be done at an external surface using $C = \epsilon_s \phi_0 / d - q \sigma_s$ and $\alpha = \epsilon_s / d$, as shown in Fig. 3.3, which places a surface charge of $q \sigma_s$ a distance d away from a point at which ϕ is pinned to ϕ_0 . We somewhat arbitrarily choose $d = 10$ nm, which is much smaller than any of the mesh features (for semiclassical Poisson simulations) in our devices of interest. Robin BCs are enforced in an integral form, similar to that of Neumann BCs (cf. Eq. 3.23).

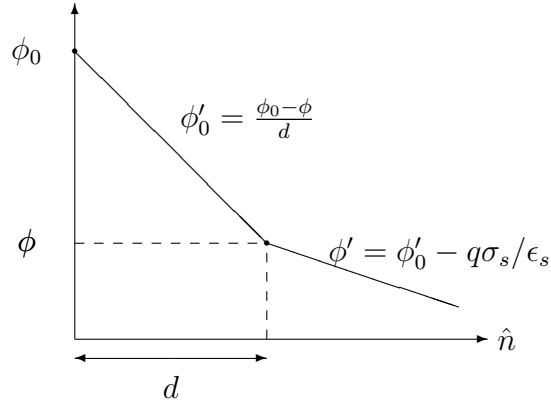


Figure 3.3. The derivation of the Robin BC parameters used to set a value of the potential and a nearby surface charge on a single surface. This diagram shows a 1D cut along the direction \hat{n} normal to a surface element. ϕ_0 , σ_s , and d are given values, and ϕ is the variable being solved. The derivative ϕ' is along the direction normal to the surface. Combining the equations yields $\epsilon_s \phi' = \epsilon_s \frac{\phi_0 - \phi}{d} - q\sigma_s$, which takes the form of a Robin boundary condition.

Chapter 4

QCAD Schrodinger-Poisson Solver

Schrodinger Solver

The time-independent single-particle effective mass Schrodinger equation takes the form of

$$-\frac{\hbar^2}{2} \nabla \left(\frac{1}{m^*} \nabla \psi(\mathbf{r}) \right) + V(\mathbf{r}) \psi(\mathbf{r}) = E \psi(\mathbf{r}). \quad (4.1)$$

The FE weak form of the equation is

$$\begin{aligned} & \frac{\hbar^2}{2m^*} \left(\int \nabla \psi \cdot \nabla w d\Omega - \int_{\Gamma} \nabla \psi \cdot \hat{\eta} w d\Gamma \right) \\ & + \int V \psi w d\Omega - \int E \psi w d\Omega = 0 \end{aligned} \quad (4.2)$$

The weak form is discretized by the FE method and the resulting eigenvalue problem $[H][\psi] = [E][\psi]$ is solved by the Trilinos eigensolver package called Anasazi [?].

The Schrodinger solver supports two types of boundary conditions: Dirichlet and Neumann. For Dirichlet boundaries, $\psi = 0$. All other boundaries excluding Dirichlet are treated as Neumann BCs which, require $\frac{1}{m^*} \nabla \psi \cdot \hat{\eta} = 0$ on outer boundaries, and $\frac{1}{m^*} \nabla \psi \cdot \hat{\eta}$ being continuous (i.e., flux conservation) across material interfaces on internal boundaries. As in the Poisson solver, Neumann BCs are automatically satisfied in the FE framework by setting $\int_{\Gamma} \nabla \psi \cdot \hat{\eta} w d\Gamma = 0$ in Eq. (4.2). (The ability to set non-flux-conserving Neumann boundary conditions is absent in the Schrodinger solver since it would have no application in our work.)

Figure 4.1 shows a comparison between the QCAD Schrodinger solver and analytic results of the lowest six wave functions and energies for a 1D parabolic potential well. The QCAD and analytic results are in excellent agreement. The solver was also applied to 2D and 3D infinite potential wells. The obtained wave functions and energies were compared with the analytic results and excellent agreement was also observed.

One of the advantages of using the FE discretization over the finite difference discretization is that the continuities of ψ and $\frac{1}{m^*} \nabla \psi$ across heterojunctions are automatically satisfied in the former case, whereas they have to be explicitly enforced in the latter case. Specifically,

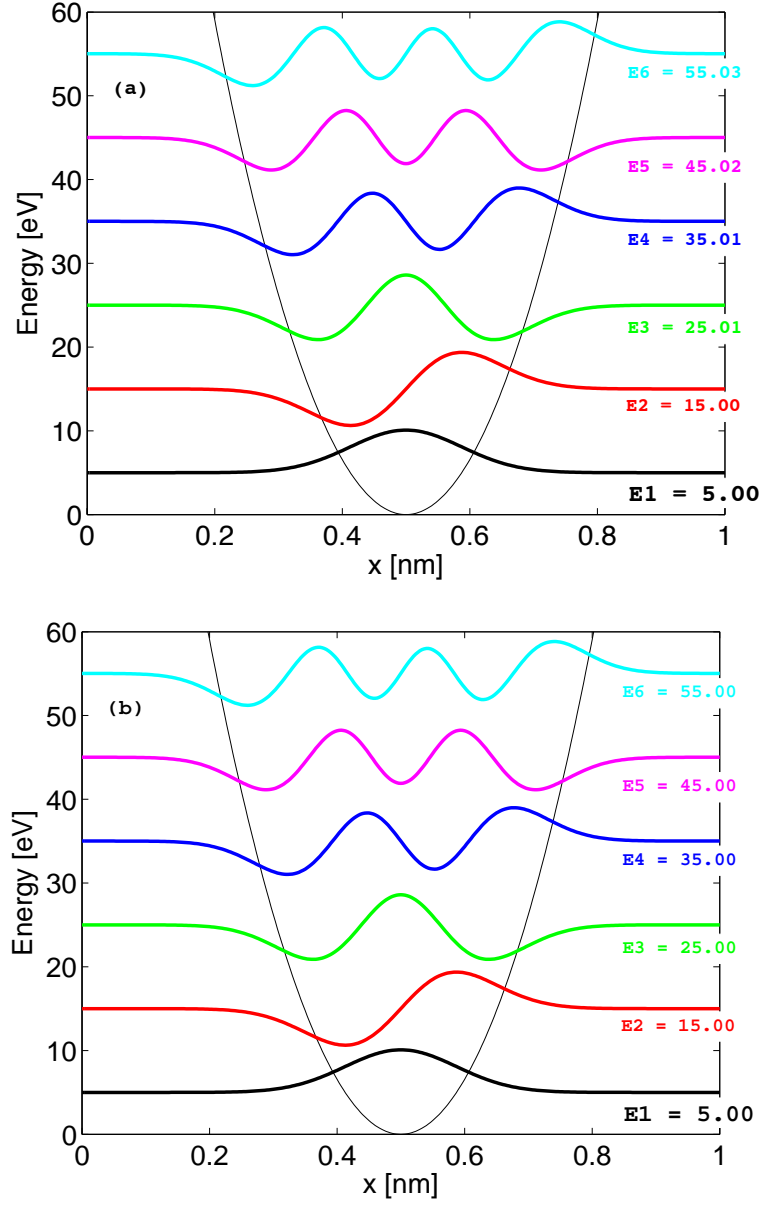


Figure 4.1. (Color online) (a) Wave functions and energies obtained from the QCAD Schrodinger solver for a 1D parabolic potential well. (b) Analytic wave functions and energies for the same potential well. All the wave functions are scaled by the same factor for easy visualization. It is clear that QCAD wave functions and energies agree very well with the analytic results.

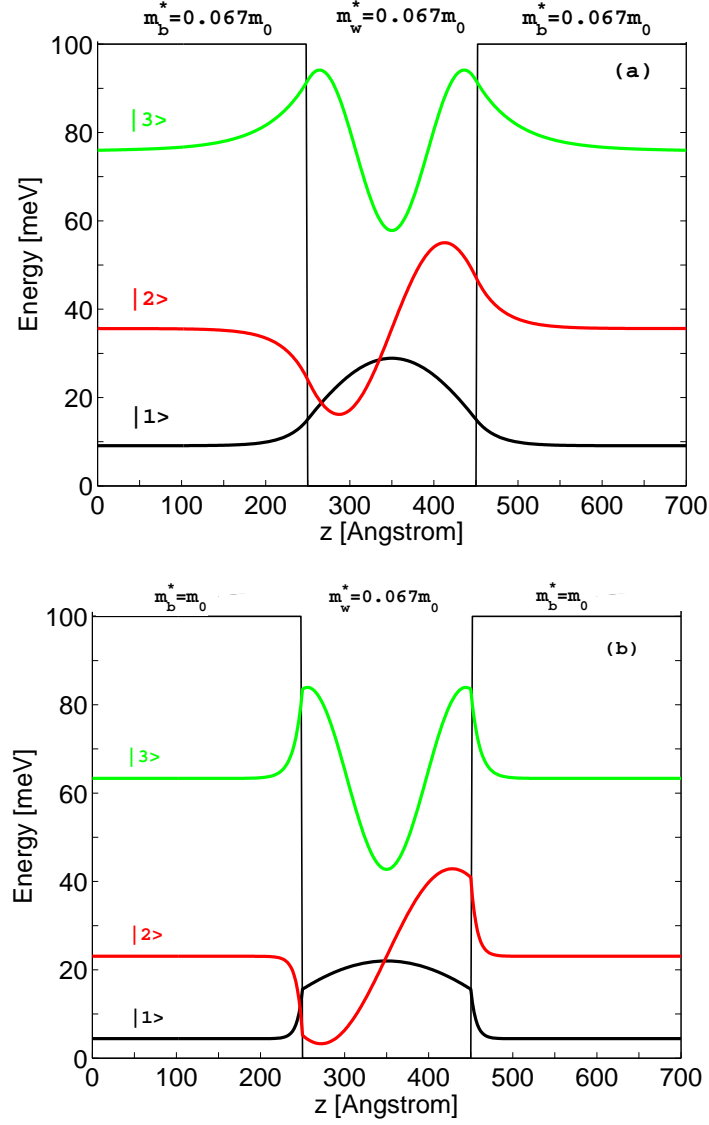


Figure 4.2. (Color online) (a) Lowest three wave functions and energies obtained from the QCAD Schrodinger solver for the 1D finite potential well with $m_b^* = m_w^* = 0.067m_0$ where m_0 is the free electron mass. (b) Same as (a) except $m_b^* = m_0$ and $m_w^* = 0.067m_0$. All wave functions are scaled by the same factor for easy visualization.

when going from a homojunction device to a heterojunction device, the QCAD Schrodinger solver does not require any code change except setting the proper effective masses for the materials used. As an example, consider a 1D finite potential well that has a width of 20 nm, a potential height of 100 meV, and can have different effective masses for the well and barrier. Figure 4.2(a) shows the lowest three wave functions and energies obtained from QCAD for a homojunction device with the same effective mass for the well and barrier. In this case, the wave functions and their first derivatives are all continuous across the junctions. The wave functions and energies agree very well with the corresponding analytic results in Figure 2.15, Ref. [?]. When the well and barrier have different effective masses, as shown in Fig. 4.2(b) for a heterojunction device, the wave functions are still continuous across the junctions, but their first derivatives are discontinuous due to the difference in effective masses.

Self-Consistent Schrodinger-Poisson Solver

In realistic quantum devices such as DQDs, we can divide the entire structure (relatively large) into semiclassical and quantum regions. These regions are chosen such that in semiclassical regions, solving the nonlinear Poisson equation alone is often sufficient to obtain a good estimate of electrostatics, whereas in quantum regions, the Poisson and Schrodinger equations need to be coupled self-consistently for electrons (we focus on electrons only as they are used for qubit operation). The coupled two equations take the following form

$$\begin{aligned} -\nabla \cdot (\epsilon_s \nabla \phi) &= q[p(\phi) - n(E_i, \psi_i) + N_D^+(\phi) - N_A^-(\phi)], \\ \frac{-\hbar^2}{2} \nabla \cdot \left(\frac{1}{m^*} \nabla \psi_i \right) + V(\phi, n) \psi_i &= E_i \psi_i, \end{aligned} \quad (4.3)$$

where the electron density n becomes a function of the i th energy level E_i and the envelope wave function ψ_i of the Schrodinger equation, while the potential energy V is a function of ϕ and n . The general expression for $n(E_i, \psi_i)$ is given by $\sum_i N_i |\psi_i|^2$, where the N_i term takes different expressions depending on confinement dimensionality.

Quantum Electron Density

In Si quantum devices, when we focus on those devices where the Si/other material (e.g., Si/SiO₂) interfaces are parallel to the [100] plane, the six equivalent conduction band minima of the bulk silicon are split into two groups due to the breaking of crystallographic symmetry, widely known as Δ_4 (fourfold degeneracy) and Δ_2 (double degeneracy) valleys, with Δ_2 valleys are lower in energy. At low temperatures, especially the operating temperatures for DQD qubits which are in the mK to a few Kelvin range, only the Δ_2 valleys are occupied by electrons; therefore, we consider the Δ_2 valleys only for Si devices in QCAD. Due to the ellipsoidal energy surfaces at the Δ_2 minima, the electron effective mass in the Schrodinger

equation is different from that used in computing the electron density, and it depends on the confinement direction and the number of confined directions.

Two un-confined dimensions. In 1D-confined devices such as a 1D Si MOS capacitor, electrons are spatially confined in one direction (assumed x direction in QCAD) but free to move in the y and z directions. The Si/SiO₂ interface is in the $y-z$ plane and perpendicular to the longitudinal axis of the Δ_2 valleys. The coupled Schrodinger equation is 1D and given by

$$-\frac{\hbar^2}{2} \frac{d}{dx} \left(\frac{1}{m_t^*} \frac{d\psi_i(x)}{dx} \right) + V(\phi, n) \psi_i(x) = E_i \psi_i(x). \quad (4.4)$$

where m_t^* is the electron longitudinal effective mass of silicon. From Appendix A, the volume electron density n is computed as

$$\begin{aligned} n(E_i, \psi_i) &= \sum_i N_i |\psi_i|^2 = \sum_i n_{2D,i} |\psi_i|^2 \\ &= \sum_i \left(2 \frac{m_t^* k_B T}{\pi \hbar^2} \ln \left[1 + \exp \left(\frac{E_F - E_i}{k_B T} \right) \right] |\psi_i|^2 \right), \end{aligned} \quad (4.5)$$

where m_t^* is the electron transverse effective mass of silicon and 2 accounts for the double degeneracy of the Δ_2 valleys. $|\psi_i(x)|^2$ is spatially normalized to 1, i.e., $\int |\psi_i(x)|^2 dx = 1$, and hence has the unit of 1/length. When $E_F > E_i$, the $\exp[(E_F - E_i)/(k_B T)]$ term in Eq. (4.5) can numerically go to infinity for very small T (mK to a few K), which can cause numerical instability. To avoid such problems, when the argument $(E_F - E_i)/(k_B T)$ is relatively large (e.g., > 100), we replace the $\ln[1 + \exp((E_F - E_i)/(k_B T))]$ term with $(E_F - E_i)/(k_B T)$.

One un-confined dimension. We next consider devices, such as quantum wire structures, where electrons are confined along two dimensions (assumed x and y directions in QCAD) and are free to move in the z direction (the wire direction). The Si/SiO₂ interface is perpendicular to the y axis and also the longitudinal axis of the Δ_2 valleys. The coupled Schrodinger equation is 2D and given by

$$\begin{aligned} &-\frac{\hbar^2}{2} \frac{\partial}{\partial x} \left(\frac{1}{m_t^*} \frac{\partial \psi_i(x, y)}{\partial x} \right) - \frac{\hbar^2}{2} \frac{\partial}{\partial y} \left(\frac{1}{m_l^*} \frac{\partial \psi_i(x, y)}{\partial y} \right) \\ &+ V(\phi, n) \psi_i(x, y) = E_i \psi_i(x, y). \end{aligned} \quad (4.6)$$

From Appendix A, the volume electron density n is computed as

$$\begin{aligned} n(E_i, \psi_i) &= \sum_i N_i |\psi_i|^2 = \sum_i n_{1D,i} |\psi_i|^2 \\ &= \sum_i \left[2 \left(\frac{2m_t^* k_B T}{\pi \hbar^2} \right)^{\frac{1}{2}} \mathcal{F}_{-\frac{1}{2}}(\eta_F) |\psi_i|^2 \right], \end{aligned} \quad (4.7)$$

where $\mathcal{F}_{-\frac{1}{2}}(\eta_F)$ is the Fermi-Dirac integral of -1/2 order. It is computed by using the approximate analytic expressions in Ref. [?], which have a very small error less than 0.001% in the entire η_F range. $|\psi_i(x, y)|^2$ is spatially normalized to 1, i.e., $\int \int |\psi_i(x, y)|^2 dx dy = 1$, and hence has the unit of $1/\text{length}^2$.

Zero un-confined dimensions. In devices such as quantum dot structures, electrons are spatially confined in all three directions and there are no (zero) dimensions in which they are free to move. The Si/SiO₂ interface is perpendicular to the z direction and also the longitudinal axis of the Δ_2 valleys. The coupled Schrodinger equation is 3D and given by

$$\begin{aligned} & -\frac{\hbar^2}{2} \frac{\partial}{\partial x} \left(\frac{1}{m_t^*} \frac{\partial \psi_i(x, y, z)}{\partial x} \right) - \frac{\hbar^2}{2} \frac{\partial}{\partial y} \left(\frac{1}{m_t^*} \frac{\partial \psi_i(x, y, z)}{\partial y} \right) \\ & - \frac{\hbar^2}{2} \frac{\partial}{\partial z} \left(\frac{1}{m_l^*} \frac{\partial \psi_i(x, y, z)}{\partial z} \right) \\ & + V(\phi, n) \psi_i(x, y, z) = E_i \psi_i(x, y, z). \end{aligned} \quad (4.8)$$

The volume electron density n is computed as

$$n(E_i, \psi_i) = \sum_i N_i |\psi_i|^2 = \sum_i \left[\frac{4}{1 + \exp(\frac{E_i - E_F}{k_B T})} |\psi_i|^2 \right], \quad (4.9)$$

where 4 accounts for the double degeneracy of the Δ_2 valleys and that of the spin. $\psi_i(x, y, z)$ is normalized to 1 in the 3D quantum domain, and has the unit of $1/\text{length}^3$. When $E_i > E_F$, the $\exp[(E_i - E_F)/(k_B T)]$ term in Eq. (4.9) can blow up numerically. To avoid such problem, when $(E_i - E_F)/(k_B T)$ is relatively large (e.g., > 100), we replace the $[1 + \exp(\frac{E_i - E_F}{k_B T})]^{-1}$ term with $\exp(\frac{E_F - E_i}{k_B T})$.

All the above derivations are also applicable to other devices where the semiconductors have a single conduction band minimum located at the Γ valley such as GaAs-based devices, except that the valley degeneracy is 1 and a single electron effective mass is used in all the equations.

Next we discuss the potential energy term $V(\phi, n)$ in the coupled Schrodinger equation. It takes the form of

$$V(\phi, n) = q\phi_{ref} - \chi - q\phi + V_{xc}(n), \quad (4.10)$$

where $V_{xc}(n)$ is the exchange-correlation correction due to the Pauli exclusion principle in real many-electron systems. For the $V_{xc}(n)$ term, we use the well-known local density parameterization suggested by Hedin and Lundqvist [?] that has also been widely used by other

authors [?, ?, ?]. It is given as

$$\begin{aligned}
V_{xc}(n) &= \frac{-q^2}{4\pi^2\epsilon_s} [3\pi^2 n(\mathbf{r})]^{\frac{1}{3}} \left[1 + 0.7734x \ln \left(1 + \frac{1}{x} \right) \right], \\
x &= \frac{1}{21} \left(\frac{4\pi n(\mathbf{r}) b^3}{3} \right)^{-\frac{1}{3}}, \\
b &= \frac{4\pi\epsilon_s \hbar^2}{m_{xc}^* q^2}.
\end{aligned} \tag{4.11}$$

Since this parameterization requires a scalar effective mass m_{xc}^* as input, we use an average mass for Si as suggested in Ref. [?],

$$\frac{1}{m_{xc}^*} = \frac{1}{3} \left(\frac{1}{m_l^*} + \frac{2}{m_t^*} \right). \tag{4.12}$$

Self-Consistency

The Schrodinger (S) and Poisson (P) equations in Eq. (4.3) have strong nonlinear coupling. They need to be solved self-consistently by certain iterative numerical schemes. Various iteration schemes [?, ?, ?, ?, ?, ?, ?, ?] have been proposed and used over the past few decades. Among them, three are notable: the under-relaxation method, the damped Newton method, and the predictor-corrector approach.

The under-relaxation scheme [?, ?] (also called convergence-factor or simple average method) solves the S and P equations in succession, and under-relaxes the electron density n or the electrostatic potential ϕ for the k th S-P outer iteration, using either a pre-set constant or an adaptively determined relaxation parameter $w^{(k)}$ (see the references for details). The advantage of this method is its simplicity. Its weakness is that the relaxation parameter $w^{(k)}$ is not known in advance and needs to be dynamically but heuristically readjusted during the course of iterations; if $w^{(k)}$ is too large, the iteration loop cannot reach convergence, whereas, if $w^{(k)}$ is too small, it takes too many iteration steps to achieve convergence.

The damped Newton method [?, ?] also solves the S and P equations in succession, but uses a damped Newton method [?] for the outer S-P iteration. Specifically, this approach starts from an initial guess $\phi^{(0)}$, solves the Schrodinger eigenvalue problem, computes quantum electron density $n^{(k)}$ according to section 4, and then solves a linear P equation, obtained by linearizing the Poisson equation in Eq. (4.3) according to the Newton method [?] with an approximate Jacobian matrix; the potential $\phi_{out}^{(k)}$ from the linearized P equation is used to obtain $\phi_{in}^{(k+1)} = \phi_{in}^{(k)} + w^{(k)}(\phi_{out}^{(k)} - \phi_{in}^{(k)})$, which is then input to the S equation, and the procedure continues until self-consistency is reached. Here, the $w^{(k)}$ damping parameter is not heuristic, but can be determined by the selection algorithm of the damped Newton method (cf. Ref. [?]). k represents the k th Newton iteration and also the k th outer S-P iteration. The Jacobian matrix must be approximated because the quantum electron density n

does not have explicit dependence on the potential ϕ . The approximate Jacobian matrix is obtained by simply assuming a semi-classical electron density expression in the P equation. The approach has shown reasonably robustness [?], however, because of the approximate nature of the Jacobian, it often takes many Newton iterations (e.g., 50) to achieve sufficient self-consistent accuracy (e.g., ϕ is converged within 0.01 meV).

It is well-known that the under-relaxation [?], damped Newton [?], and other similar iteration schemes [?, ?] do not necessarily lead to convergence, or take too many iterations to achieve it. These schemes have been used mostly in 1D S-P problems and in only a limited number of 2D applications, and one may rightly expect that they would have much more difficulty in achieving convergence in 3D S-P problems (e.g. in quantum dots). The key reason for the instability of these methods is that they do not physically address the strong nonlinear coupling between the S and P equations. In 1997 Trellakis et. al [?] proposed the predictor-corrector (p-c) iteration scheme based on a perturbation argument. Due to its solid physical groundings, the p-c method has shown fast and robust convergence behavior [?, ?, ?], and has been widely used in 2D and 3D simulations of various quantum semiconductor devices [?, ?, ?]. Given its excellent track record, we implemented this p-c method in QCAD for the self-consistent S-P loop.

The key feature of the p-c method is that it partially decouples the S and P equations by moving most nonlinearities into the nonlinear Poisson equation

$$-\nabla \cdot (\epsilon_s \nabla \phi) = q[p(\phi) - \tilde{n}(\phi) + N_D^+(\phi) - N_A^-(\phi)], \quad (4.13)$$

where $\tilde{n}(\phi)$ is an approximate expression for the quantum electron density $n(E_i, \psi_i)$, which has an explicit dependence on the potential ϕ (note the exact quantum density $n(E_i, \psi_i)$ does not have explicit dependence on ϕ). The nonlinear Poisson equation can be solved by a Newton method (the predictor step). The predicted result for \tilde{n} and ϕ from this equation is then corrected in an outer iteration step by the solution of Schrodinger equation (the corrector step).

The approximate quantum density \tilde{n} is obtained by using the first-order perturbation theory and the derivative property of Fermi-Dirac integrals [?]. The resulting \tilde{n} expression is the same as the exact quantum density n given in Section 4, except that the argument in the Fermi-Dirac integral is modified to include an explicit dependence on ϕ . For 1D-confined Si devices, \tilde{n} is given by

$$\begin{aligned} \tilde{n}(\phi) = & \sum_i \left(2 \frac{m_i^* k_B T}{\pi \hbar^2} |\psi_i^{(k)}|^2 \right. \\ & \times \ln \left[1 + \exp \left(\frac{E_F - E_i^{(k)} + q(\phi - \phi^{(k)})}{k_B T} \right) \right] \Bigg), \end{aligned} \quad (4.14)$$

where the superscripts (k) denote quantities obtained in the previous k th outer S-P iteration

step (hence they are known quantities). For 2D-confined Si devices,

$$\begin{aligned} \tilde{n}(\phi) = & \sum_i \left[2 \left(\frac{2m_i^* k_B T}{\pi \hbar^2} \right)^{\frac{1}{2}} |\psi_i^{(k)}|^2 \right. \\ & \left. \times \mathcal{F}_{-\frac{1}{2}} \left(\frac{E_F - E_i^{(k)} + q(\phi - \phi^{(k)})}{k_B T} \right) \right]. \end{aligned} \quad (4.15)$$

For 3D-confined Si devices,

$$\tilde{n}(\phi) = \sum_i \left[\frac{4 |\psi_i^{(k)}|^2}{1 + \exp \left(\frac{E_i^{(k)} - E_F - q(\phi - \phi^{(k)})}{k_B T} \right)} \right]. \quad (4.16)$$

Note that there is a minus sign in the $q(\phi - \phi^{(k)})$ term in Eq. (4.16). In principle, once the self-consistent S-P loop is converged, this term should be numerically zero, which might suggest that the sign shall not matter. However, our experience with QCAD is that the minus sign is very important for the 3D-confined case to achieve self-consistent convergence; if we used a plus sign here, the outer S-P loop ran into numerical oscillations.

The self-consistent p-c procedure in QCAD is done in the following steps.

(1) Solve the semiclassical nonlinear Poisson equation, Eq. (3.1), using the Newton solver in Trilinos [?], to obtain an initial potential $\phi^{(0)}$ and compute the initial total potential energy $V^{(0)}$ without the exchange-correlation correction V_{xc} .

(2) Solve the coupled Schrodinger equation for the k th ($k \geq 1$) S-P iteration step,

$$-\frac{\hbar^2}{2} \nabla \left(\frac{1}{m^*} \nabla \psi^{(k)} \right) + V^{(k-1)} \psi^{(k)} = E^{(k)} \psi^{(k)},$$

to obtain $E^{(k)}$ and $\psi^{(k)}$ (performed using an eigensolver available in Trilinos).

(3) Solve the coupled nonlinear Poisson equation with the approximate quantum electron density $\tilde{n}^{(k)}(\phi^{(k)}; \phi^{(k-1)}, E^{(k)}, \psi^{(k)})$,

$$-\nabla \cdot (\epsilon_s \nabla \phi^{(k)}) = q[p(\phi^{(k)}) - \tilde{n}^{(k)} + N_D^+(\phi^{(k)}) - N_A^-(\phi^{(k)})],$$

using the Trilinos Newton solver to obtain the updated potential $\phi^{(k)}$, and compute $\tilde{n}^{(k)}$ and $V^{(k)}$ including $V_{xc}(\tilde{n}^{(k)})$. Note we want to use the latest electron density to compute V_{xc} for good convergence.

(4) Check if $\|\phi^{(k)} - \phi^{(k-1)}\| < \delta$ for $k \geq 2$ everywhere in the device, with δ being a pre-defined tolerance often chosen as 1×10^{-5} V; if not, repeat steps (2) to (4).

It is clear from the above procedure that there is no under-relaxation step between two S-P iterations and the outer iteration reduces to a simple alternation between solving S

and P equations. In addition, the Newton Jacobian matrix for the nonlinear P equation, Eq. (4.13), can be found analytically, avoiding the necessity of using an approximate Jacobian matrix in the damped Newton iteration scheme [?]. In terms of code implementation, the p-c method was very straightforward to implement in QCAD within the Albany framework. Here we emphasize that, because of the automatic differential capability in Trilinos, the Newton Jacobian matrix is computed directly by the code, and we do not need to derive the Jacobian matrix. More details on the implementation and the Albany code structure are found in Ref. [?].

Validation Example

To validate the self-consistent S-P solver, we performed simulations on two structures and compared with other simulation results. The first one is a 1D MOS Si capacitor with 4-nm oxide and $5 \times 10^{17} \text{ cm}^{-3}$ p-substrate doping. Figure 4.3 compares the Δ_2 -valley lowest four wave functions and energies in the capacitor obtained from QCAD and SCHRED [?] at $T = 50 \text{ K}$, the lowest temperature allowed by SCHRED. (SCHRED is a 1D self-consistent Poisson-Schrodinger solver for MOS capacitors available on www.nanohub.org.) There are two simulation differences between QCAD and SCHRED: (i) QCAD applies the S-P solver to both Si and SiO_2 regions, leading to slight wave function penetration in the oxide ($x < 0$) as seen in Fig. 4.3a, while SCHRED assumes $\psi = 0$ at the Si/ SiO_2 interface; (ii) QCAD considers the two Δ_2 valleys only, whereas SCHRED includes both the two Δ_2 and the four Δ_4 valleys. A typical effective mass of $0.5m_0$ is often assumed [?, ?] for SiO_2 , where m_0 is the free electron mass. To minimize the wave function difference near the Si/ SiO_2 interface due to the different boundary conditions imposed by the two tools, we used $0.005m_0$ as the SiO_2 effective mass for the QCAD simulations. The choice of setting $m_{ox}^* = 0.005m_0$ is because, at the Si/ SiO_2 interface, QCAD applies the flux conservation condition of $\frac{1}{m_{ox}^*} \cdot \frac{d\psi}{dx}|_{ox} = \frac{1}{m_{si}^*} \cdot \frac{d\psi}{dx}|_{si}$, and in order to make ψ at the interface as close to 0 as possible (to be more consistent with SCHRED), we need small $\frac{d\psi}{dx}|_{ox}$, meaning small m_{ox}^* . At $T = 50 \text{ K}$, we expect that ignoring the higher energy Δ_4 valleys produces negligible effect on the Δ_2 -valley results, as only the Δ_2 -valley lowest subband is occupied by electrons at this low temperature. As expected, the wave functions and energies in Fig. 4.3 show excellent agreement between QCAD and SCHRED. The results also indicate that in this device, the exchange-correlation potential V_{xc} significantly increases the subband energy separation due to the many-body interaction (e.g., the separation between the lowest two subbands is increased from $26.71+72.54 = 99$ to $45.72+74.5 = 120 \text{ meV}$ when including V_{xc}), and it also somewhat compresses the wave functions as seen by the differences between the dash and solid curves in the figure, which agree with the observations in Ref. [?].

The second example is a gate-induced Si quantum wire structure from Ref. [?]. Figure 4.4 shows the schematic diagram of the simulated 2D structure. For simulation purpose, the device is divided into quantum and semiclassical regions. The quantum regions include the 15-nm thick Si quantum and the 4-nm thick SiO_2 quantum regions denoted in the figure, where the self-consistent S-P solver is applied. The 15-nm and 4-nm were chosen such that

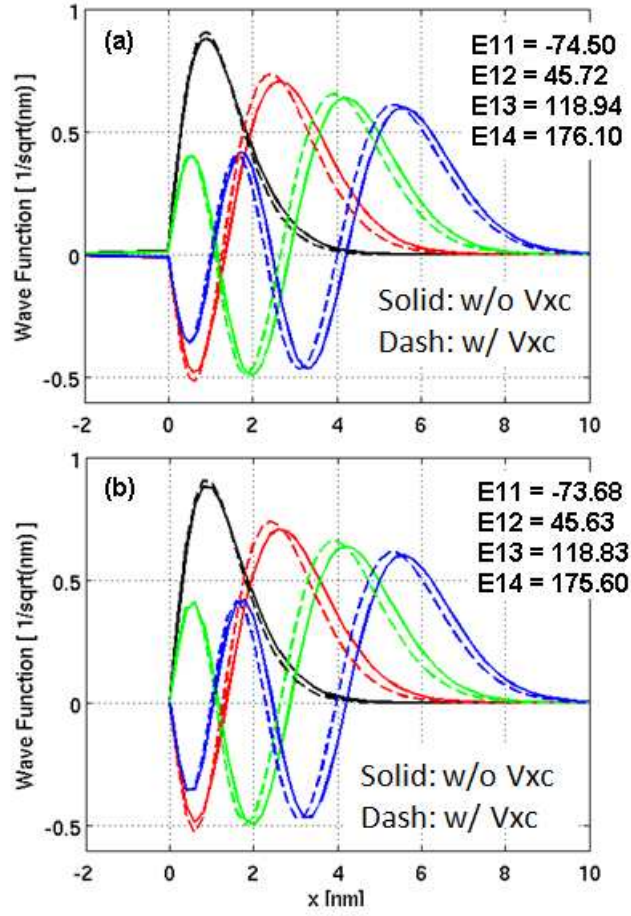


Figure 4.3. (Color online) Δ_2 -valley lowest four subband wave functions and energies in a 1D MOS Si capacitor at $T = 50$ K and $V_g = 3$ V obtained from QCAD (a) and from SCHRED (b). The Si/SiO₂ interface is located at $x = 0$. The solid and dashed curves are obtained without and with the exchange-correlation effect, respectively. The subband energies in [meV], referenced from the Fermi level and including the V_{xc} effect, are denoted by $E1i$, where the “1” indicates the Δ_2 -valley and i indexes the subband (SCHRED’s labeling convention). For comparison, the corresponding energies without V_{xc} are -72.54, 26.71, 90.73, 144.69 for QCAD, and -71.76, 26.12, 89.22, 142.27 for SCHRED.

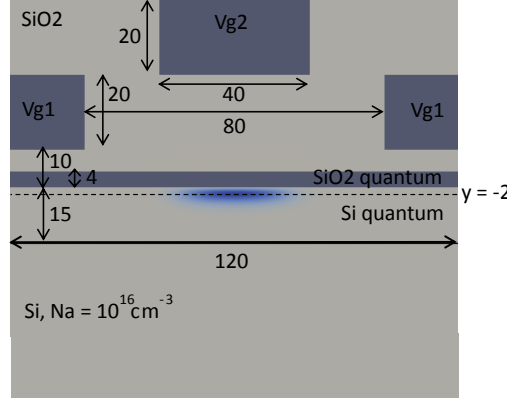


Figure 4.4. (Color online) Schematic diagram of the simulated 2D structure with all dimensions given in nm. The blue 2D contour in the Si quantum region shows the Δ_2 -valley lowest subband wave function obtained from QCAD without the V_{xc} effect at $T = 10$ K, $V_{g1} = 0.8$ V, and $V_{g2} = 3.5$ V with all voltages referred to flat band. The dash line denotes the $y = -2$ nm location.

the wave functions are essentially 0 at the boundaries of the quantum and non-quantum regions. The remaining Si and SiO₂ regions are treated as semiclassical, that is, only the Poisson equation with semiclassical carrier density is solved at each S-P iteration. The gate V_{g2} induces electrons in the Si quantum region, while the V_{g1} gates are used to deplete electrons, hence an effective quantum wire is formed with the wire direction perpendicular to the 2D plane. The blue 2D contour in the Si quantum region shows the Δ_2 -valley lowest subband wave function obtained from QCAD without the effect of V_{xc} at $T = 10$ K, $V_{g1} = 0.8$ V, and $V_{g2} = 3.5$ V with all voltages referred to flat band. The peak of the wave function is located around the $y = -2$ nm dash line (the $y = 0$ location is at the Si-quantum/SiO₂-quantum interface). The lowest five subband wave functions along the $y = -2$ nm line are given in Fig. 4.5, which agree very well with Fig. 4(a) in Ref. [?]. Given $T = 10$ K and $V_{g1} = 0.8$ V, we also performed QCAD S-P simulations for a range of V_{g2} voltages, integrated the electron density in the Si quantum region for each V_{g2} , and then plotted the subband energies as a function of integrated electron density to compare with the results in Ref. [?]. Figure 4.6 compares the Δ_2 -valley lowest three subband energies as a function of integrated electron density in the Si quantum region between QCAD and the reference. The agreement between them is very good considering the fact that the reference used a different S-P iteration scheme and did not mention if a fixed interface charge was used or not (no fixed charge was used in QCAD simulations).

For these 1D and 2D examples, their S-P convergence behavior from QCAD are plotted in Fig. 4.7, where the vertical axis is the maximum potential error between two outer iterations in the entire device including the semiclassical region (note that the corresponding

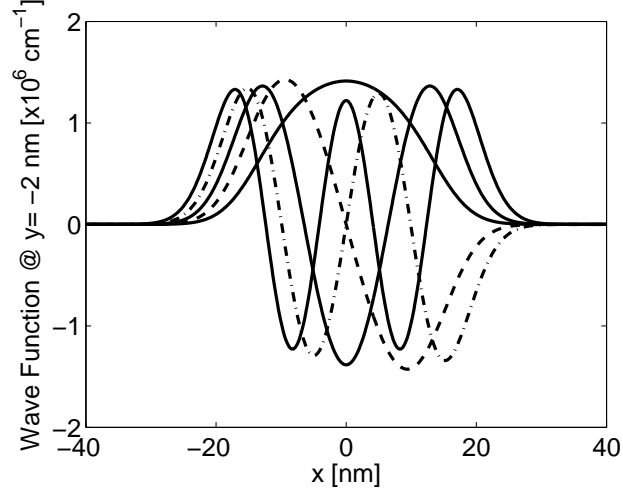


Figure 4.5. (Color online) Δ_2 -valley lowest five subband wave functions along the $y = -2$ nm dash line in Fig. 4.4. These wave functions agree very well with Fig. 4(a) in Ref. [?].

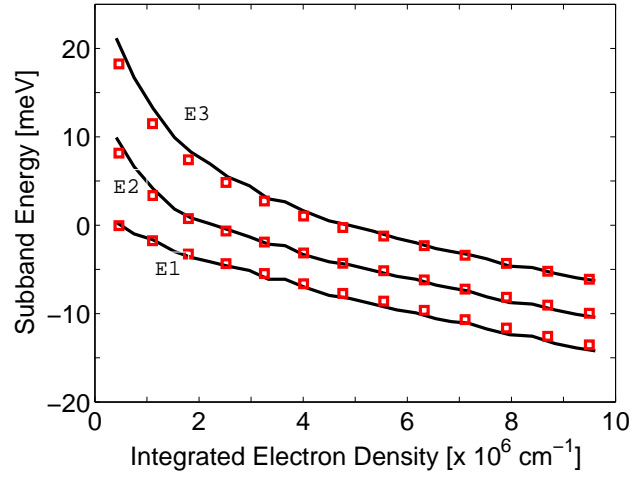


Figure 4.6. (Color online) Δ_2 -valley lowest three subband energies as a function of integrated electron density in the Si quantum region. Black curves plot the data extracted from Fig. 3 in Ref. [?], while the red squares are the data obtained from QCAD. The energies are with respect to the Fermi level which is set to 0.

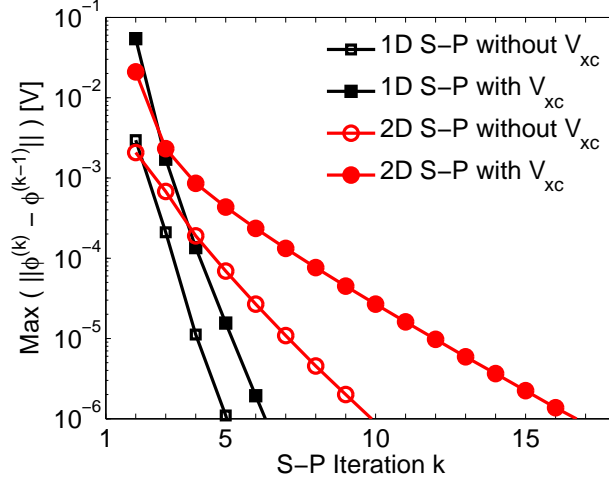


Figure 4.7. (Color online) Convergence behavior of the self-consistent QCAD S-P solver for the 1D MOS Si capacitor and the 2D gate-induced Si quantum wire devices.

convergence are not available from either the referenced tool or paper). We see that the predictor-corrector approach for the S-P iteration leads to fast and monotonic convergence in the two closed quantum systems; and the inclusion of the V_{xc} effect requires more iteration steps because of the stronger coupling that V_{xc} introduces between the P and S equations.

Chapter 5

QCAD User Guide

Introduction

What is QCAD?

The Quantum Computer Aided Design toolkit is a finite-element based tool designed to simulate semiconductor quantum devices. Such devices typically have nanometer length scales and operate at milli-Kelvin temperatures. QCAD can be used to solve problems in one, two, or three dimensions, and has several modes of operation:

- **Semiclassical mode.** The non-linear Poisson equation obtained by making the Thomas-Fermi approximation, which relates conduction band energy relative to the Fermi level to electron density, is solved, resulting in a “semiclassical” electron density and electric potential throughout the simulated domain.
- **Schrodinger mode.** The time-independent Schrodinger equation is solved, yielding the single particle eigen-energies and eigen-states of a given potential. The potential can be supplied externally, or can be given as a mathematical expression (limited parsing capability).
- **Poisson-Schrodinger mode.** The simulation domain is divided into “quantum” and “non-quantum” regions. A self-consistent solution is found for the electron density and potential by alternately solving Poisson and Schrodinger equations and iterating until convergence. The Poisson equation results from the Thomas-Fermi approximation in the non-quantum region and a quantum density, proportional to the sum of squared single particle wave functions over occupied states, in the quantum region, and the Schrodinger equation is solved only within the quantum region. Finding self-consistent solutions may be accelerated by using a coupled Poisson-Schrodinger solver, which solves the Poisson and Schrodinger equations simultaneously using an iterative non-linear method.
- **Schrodinger-CI mode.** After solving the Schrodinger equation as in “Schrodinger mode”, Slater-determinants of the resulting single-particle states are used to form a

many-particle basis, and the many-electron Schrodinger equation is solved using a configuration interaction (CI) method.

- **Poisson-Schrodinger-CI mode.** Coupling the above two modes together, this mode iteratively converges single-particle electron wave functions and then applies a configuration interaction algorithm to compute many-electron wave functions.

The QCAD tool is a single executable, and reads input from one or several XML file(s). The format of these files depends upon the mode of operation and on the problem specifics, and is the focus of this user guide.

A finite-element mesh specifying the discretization of the problem domain must be given in all cases. Though it can be supplied in several formats, by far the most flexible and consequently most utilized format is an Exodus (*.exo) mesh. This mesh format is native to the meshing tool Cubit, and it is our desire that other common mesh file formats will be supported in the future.

How do I get QCAD?

Right now, unless you work at Sandia, you probably can't. We're working on it.

How do I run QCAD?

The executable that runs QCAD is called **Albany**. You run QCAD by supplying a single command argument to Albany – the name of the main XML input file. For example:
`/path/to/Albany input.xml`

Input files

To run a QCAD simulation, at least one and usually several input files are required. There are three distinct types of QCAD input files:

1. **“Main XML” input file:** This type of input file defines which type of problem should be solved and how it should be setup. The main XML file contains most if not all of the run-specific information, such as the particular boundary conditions used, what quantities to compute, and what solver parameters to use. It also references the next two types of input files.
2. **“Materials XML” input file:** Also known as a “material database file”, this input file specifies the material properties to be used in regions of the simulation domain.

3. **“Exodus mesh” input file:** An Exodus-format binary file which describes how the simulation domain is discretized into finite elements. When using multiple processors, the a single Exodus mesh file can be split into several smaller files (the number equal to the number of processors) to specify how the mesh is divided among the processors. (Alternatively, QCAD can divide a single-file mesh among many processors internally.)

We will next describe each of these types of input files in more detail.

Main XML input file specification

The Skeleton

All QCAD XML input files consist of `<ParameterList name="LIST_NAME"> ...`
`</ParameterList>` blocks which contain sub-`ParameterList` block and

```
<Parameter name="PARAM_NAME" type="PARAM_TYPE" value="PARAM_VALUE" />
```

tags. No other XML tags are used. The main XML input file takes a similar format for all of QCAD’s modes of operation. Its general structure is given in listing 5.1. Note that this listing does not show all available parameter lists, but is meant to convey the essential skeleton of structure in a QCAD input file.

Listing 5.1. General structure of a QCAD input file.

```
<ParameterList>
  <ParameterList name="Problem">

    <!-- Root Parameters -->

    <ParameterList name="Poisson Problem">
      <!-- Poisson Parameters -->
    </ParameterList>

    <ParameterList name="Schrodinger Problem">
      <!-- Schrodinger Parameters -->
    </ParameterList>
  </ParameterList>

  <ParameterList name="Discretization">
    <!-- Discretization Parameters -->
  </ParameterList>

  <ParameterList name="Piro">
    <!-- Solver parameters -->
  </ParameterList>
</ParameterList>
```

As can be seen from the listing, the high-level structure of consists of a single root list with three major sub-lists:

Sub-list name	Description
Problem	specifies which QCAD problem is solved, and most of the problem- (as distinguished from the <i>solver</i> -) parameters.
Discretization	specifies how the domain of the problem is divided into finite elements, i.e. “discretized”.
Piro	specifies details regarding how the problem described in the Problem sub-list is solved. In short, the solver-parameters. Most of these parameters are usually taken from a “Piro defaults” file specified in the Problem sub-list, in which case only solver parameters which deviate from those defaults need to be specified here.

In most cases default values can be used for all of the Piro parameters, and the **Piro** list can be omitted all together, leaving just two main sub-lists. The following sections describe the contents of each of the main sub-lists in detail.

The Problem list

The Problem list, as its name implies, specifies which QCAD problem is being solved. All QCAD problems can be viewed as a set of one or more pieces, with each piece being one of the following types:

- **Poisson** - solves the Poisson equation, possibly taking as input quantum wave-functions and eigen-energies from a previous Schrodinger solution.
- **Schrodinger** - solves the Schrodinger equation (an eigenproblem), possibly taking as input a potential generated by a previous Poisson solution.
- **Configuration Interaction (CI)** - solves a many-particle Schrodinger equation using a basis of many-particle states constructed from a set of single-particle states and energies.
- **Integrated Poisson-Schrodinger** - solves the Poisson and Schrodinger equations simultaneously by interpreting the Schrodinger equation for each desired eigenvector as a separate differential equation.

Since each type of piece typically takes a different set of parameters, the Problem list contains two major sub-lists named **Poisson Problem** and **Schrodinger Problem** which hold the

parameters relevant to any Poisson and/or Schrodinger pieces being solved. Parameters common to both Poisson- and Schrodinger-type pieces are contained under the Problem list directly, along with those relevant to Configuration Interaction pieces. (CI pieces take relatively few parameters, and so they do not merit their own sub-list). Integrated Poisson-Schrodinger pieces are given parameters from both the Poisson Problem and Schrodinger Problem sub-lists, and therefore do not require their own sub-list. Table 5.1 lists the most important Problem-list parameters. Next, the Poisson Problem and Schrodinger Problem sub-lists are explained in detail.

The Poisson Problem and Schrodinger Problem lists

The Poisson Problem and Schrodinger Problem lists (sub-lists of the Problem list) contain those parameters specific to the Poisson and Schrodinger pieces of a given QCAD problem, respectively. Sometimes one of these two lists is not needed and can be omitted (for example, for Poisson-type problems there are no Schrodinger pieces and the Schrodinger Problem sub-list can be omitted). The main purpose of the parameters under these lists is to define boundary conditions, formal parameters, responses, and an problem-specific options. Formal parameters and responses will be covered later in section 5, so the focus of this section will be on specifying boundary conditions and problem-specific options. Both the Poisson Problem and Schrodinger Problem lists specify boundary conditions in the same way, except for the name of the degree-of-freedom (DOF) contained in some of the parameter names. The following text describes how to set each of the three possible types of boundary conditions in QCAD. The placeholder *dof* should be substituted with **Phi** in the Poisson case and with **psi** in the Schrodinger case.

Dirichlet Boundary Conditions are defined using the **Dirichlet BCs** sub-list of the Poisson or Schrodinger problem lists. The **Dirichlet BCs** list contains any number of **double**-type parameters with names of the form “**DBC on NS *myNodesetName* for DOF *dof***”. The placeholder *myNodesetName* is the name of a nodeset defined in the exodus mesh. This type of parameter sets a Dirichlet boundary condition on the nodeset with value equal to the value of this parameter is the voltage (assumed to be given in volts). In a Schrodinger Problem, the values of Dirichlet boundary conditions should almost always be equal to zero.

Neumann Boundary Conditions are defined using the **Neumann BCs** sub-list of the Poisson or Schrodinger problem lists. The **Neumann BCs** list contains any number of **Array(double)**-type parameters with names of the form “**NBC on SS *mySidesetName* for DOF *dof* set scaled jump**”. The placeholder *mySidesetName* is the name of a sideset defined in the exodus mesh. The value of this type of parameter is a 1-element double-precision array that specifies a the magnitude of a jump in the derivative of the problem solution at the sideset along the direction normal to the sideset. In the Poisson problem, the double-precision value specifies a surface charge concentration in units of 10^{11}cm^{-2} . For example, to place a fixed surface charge of $4 \times 10^{11}\text{cm}^{-2}$

Table 5.1. Parameters of the main Problem list

Name	Type	Description
Solution Method	string	a necessary parameter that should always be set to “QCAD Multi-Problem”. This marks the input file as a QCAD problem, distinguishing it from other types of problems the Albany executable is capable of solving (provided it was build with other packages enabled).
Name	string	<p>the type and dimension of the problem being solved, also referred to as the mode in which QCAD is operating. Options are (where $x = 1, 2, 3$ depending on the dimension):</p> <ul style="list-style-type: none"> • Poisson xD • Schrodinger xD • Schrodinger CI xD • Poisson Schrodinger xD • Poisson Schrodinger CI xD
Length Unit In Meters	double	the length unit used in the mesh. For instance, $1e - 6$ for microns.
Energy Unit In Electron Volts	double	the energy unit to output quantities like the potential, conduction band, eigenvalues, etc. in. Note that boundary condition voltages are always given in volts regardless of what this parameter is set to. For instance $1e - 3$ for milli-electron-volts.
Temperature	double	the system temperature in Kelvin. Not needed for Schrodinger or Schrodinger CI problems.
MaterialDB Filename	string	the filename of a materials database file (see section 5 on material files).
Piro Defaults Filename	string	the filename of an XML file containing default parameters for the Piro sublist. The QCAD installation comes with a <code>default_piro_params.xml</code> file, which lists parameters appropriate to most QCAD problems, and thus in most cases it is appropriate to reference that file here.

on a sideset, the correct parameter value is $\{4.0\}$. Even though Neumann boundary conditions are available in Schrodinger problems, there are rarely used.

Robin Boundary Conditions are defined using the same **Neumann BCs** sub-list as used for the Neumann boundary conditions. To define Robin boundary conditions, use parameter names of the form “NBC on SS *mySidesetName* for DOF *dof* set robin”. The placeholder *mySidesetName* is the name of an *external* (i.e. on the edge of the mesh) sideset defined in the exodus mesh. The value of this type of parameter is a 3-element double-precision array. In the Poisson problem it specifies (in this order): the value of the solution a distance d from the sideset along it’s outward pointing normal vector, the dielectric constant ϵ divided by the distance d , and finally the surface charge, in units of 10^{11}cm^{-2} on the sideset. For example, to set the value of the solution to equal 1 volt a distance 10nm away from a surface charge of $2 \times 10^{11}\text{cm}^{-2}$ in silicon dioxide (dielectric constant 3.9), the correct boundary condition for a mesh given in microns is $\{1.0, 390, 2.0\}$. The 390 comes from writing d as 0.01 microns and dividing $\epsilon/d = 3.9/0.01 = 390$. Even though Robin boundary conditions are available in Schrodinger problems there is no known reason to use them.

Poisson-specific options are contained in the **Poisson Source** sub-list of the **Poisson Problem** list. In this list the user may set several parameters, though in most cases the default values are desired and the Poisson Source sub-list is omitted. Below is a list of the most useful parameters:

Name	Type	Description
Factor	double	an overall factor by which to multiply the RHS of the Poisson equation (i.e. the Poisson source term).
Device	string	a string specifying the device to be simulated. The default value of “elementblocks” means the device is set by the supplied mesh and the materials database file. However, there are a few other options used for testing that can be entered here.
user-defined	double	charge parameter names can be defined in the materials database XML file and a value can be set to those names here. (See section 5, which describes the materials database.)

Additionally, the Poisson Source list allows sub-lists named “**Mesh Region x** ”, where x is an integer index starting at zero. These sub-lists contain parameters defining a region of the mesh as well as a value which scales the RHS of the Poisson equation but only within the specified region. A RHS scaling is set using the double-type **Factor Value** parameter, and the mesh region is specified using the parameters in Table 5.2 (not all of which need to be specified). We note that these parameters are used throughout QCAD to

define regions of the mesh, and in particular are used by several responses which can act only on a portion of the mesh (see section 5).

Schrodinger-specific options are contained in the **Potential** sub-list of the **Schrodinger Problem** list. Here the user specifies what potential is used in the Schrodinger equation. This list is almost always supplied by the user in the **Schrodinger** or **Schrodinger CI** modes, whereas in calculation modes where Schrodinger and Poisson pieces are coupled (e.g. **Poisson Schrodinger** mode) the default behavior of using the Poisson piece's conduction band as the Schrodinger potential is desired. The available parameters in the **Potential** list are:

Name	Type	Description
Type	string	specifies the type of potential, or where to get it from. Allowed values are Parabolic , Infinite Wall , Finite Wall , String Formula , and From State . The parabolic potential is centered at 0.5 (in all dimensions) and has a confinement energy of E0 . The finite wall is a barrier of height E0 that in one dimension occupies x-coordinates between zero and Barrier Width and Barrier Width + Well Width and $2*\text{Barrier Width} + \text{Well Width}$. In two and three dimensions the well lies between Barrier Width and Barrier Width + Well Width in each dimension. The infinite wall type just sets the potential to zero, which leaves only the infinite walls at the edges of the mesh.
E0	double	a generic energy scale parameter whose meaning varies based on the value of the Type parameter. For a Parabolic potential, this is the confinement energy, and for Finite Wall this is the wall height.
Formula	string	if Type equals "String Formula" this parameter specifies an analytic expression that is evaluated to compute the potential. Basic mathematical operations are available. For example, valid formula strings include: $10*((x-0.5)^2 + (y-0.5)^2)$ and $-1/(x^2 + y^2 + z^2)^{0.5}$.
Barrier Width	double	the width of the barrier (applicable in the Finite Wall type only)
Well Width	double	the width of the well (applicable in the Finite Wall type only)
Scaling Factor	double	a multiplicative scaling applied to the potential.

Table 5.2. XML parameters available for specifying a mesh region.

Name	Type	Description
Element Block Name	string	restrict the mesh region to within an element block.
Element Block Names	string	restrict the mesh region to within several element blocks. Names of blocks are comma-delimited.
Quantum Element Blocks Only	bool	whether the region should be restricted to “quantum” element blocks (those with “quantum” equal to true in the materials database).
x min	double	restrict the x -coordinate of the mesh region
x max		
y min	double	restrict the y -coordinate of the mesh region
y max		
z min	double	restrict the z -coordinate of the mesh region
z max		
Level Set Field Name	string	if restricting the region based on the value of a field, the name of the “level-set field”.
Level Set Field Minimum	double	a minimum value for the level-set field, such that any cell with an average value less than this value is not included in the mesh region.
Level Set Field Maximum	double	a maximum value for the level-set field, such that any cell with an average value greater than this value is not included in the mesh region.

Parameters and Responses

Up to this point we have mentioned the formal inputs and outputs of a QCAD run termed “parameters” and “responses”. We will refer to the former as “formal parameters”, since the term “parameter” can also refer to XML entries which use the `<Parameter>` tag. Formal parameters, which can be thought of as the run-time inputs of a QCAD application, are completely different and distinct from the `<Parameter>` parameters of XML files.

The user specifies the formal parameters and responses through XML `Parameters` tags within `ParameterList` blocks named `Parameters` and `Response Functions`, respectively. It is likely to be intuitive why the user would want to specify responses, since this is essentially telling QCAD how to post-process the solution once the problem has been solved. For example, specifying that you would like QCAD to output the integral of the electron density over a region of the mesh is to specify a QCAD response. Why the user should need to specify `Parameters`, on the other hand, is more subtle.

Parameters Typically one thinks of the voltages (i.e. boundary conditions) set on various parts of the mesh as input “parameters” to a QCAD simulation. While boundary conditions as specified in the appropriate lists (described above) are indeed inputs to the QCAD run, and while they can be *made into* formal parameters, they are not by default formal parameters. So what’s the point of these formal parameters anyway? The answer is twofold: 1) QCAD automatically differentiates of the all the responses with respect to all of the formal parameters, and 2) formal parameters are exposed to DAKOTA when running a QCAD simulation inside of DAKOTA. (As both of these items are related to more advanced topics, new users need not bother themselves with formal parameters.) Thus, if a given boundary condition, say the voltage on Gate1, is listed as a formal parameter, the QCAD will compute the derivative of the responses with respect to this voltage. If one of those responses is the electron charge in a region, then the derivative of this response with respect to the voltage on Gate1 is a capacitance. Furthermore, if the voltage on Gate1 is a formal parameter, it can be optimized or swept using DAKOTA. If neither of these behaviors is desired, then there is no need to include a quantity (in this case a boundary condition) as a formal parameter. In addition to boundary conditions, other quantities can be made into formal parameters. These include charge parameters (defined in the materials XML file), mesh region factors (defined in the Poisson Source list), and scaling factors (Poisson source term or Schrodinger potential term scaling). The names of formal charge parameters are defined in the material database file. The names of mesh region factors are `Mesh Region Factor x` , where x is the index matching the mesh region sub-list of the `Poisson Source` list. Additional factors are `Poisson Source Factor` (multiplies RHS of Poisson equation), `Schrodinger Potential Scaling Factor` (multiplies the potential term of the Schrodinger equation), and `Schrodinger Potential E0` (sets the E0 Schrodinger potential parameter).

Every possible formal parameter has a name. The `Parameters` sub-lists contain one integer parameter named `Number`, which specifies the number of parameters, and that many `Parameter x` string parameters which specify the names of all the formal parameters. Thus,

Table 5.3. Field names available in Poisson (P) and Schrodinger (S) pieces

Field Name	P/S	Description
Solution	P	solution to Poisson’s equation (reference pt determined by material parameters)
Conduction Band	P	conduction band energy
Valence Band	P	valence band energy
Poisson Source	P	right-hand-side of Poisson equation
Electric Potential	P	the same as solution, but with reference subtracted out
Electron Density	P	density of electrons in cm^{-3}
Hole Density	P	density of holes in cm^{-3}
Ionized Dopant	P	density of ionized dopants in cm^{-3}
V	S	potential energy used in Schrodinger equation

in order to bestow formal parameter-hood upon a quantity one only needs to add its name as the value of one of the **Parameter** x parameters within the **Parameters** sub-list of either the **Poisson Problem** or **Schrodinger Problem** lists (whichever is appropriate).

Responses Responses are defined within the **Response Functions** sub-lists of the **Poisson Problem** and **Schrodinger Problem** lists. Each **Response Functions** sub-list contains one integer parameter named **Number**, which specifies the number of responses, and that many **Response** x string parameters, and optionally **ResponseParams** x sub-lists, which specify the type-names and further options of each of the responses, respectively. Many of the responses deal with one or more “fields”, which are named quantities that take values at each node (or quadrature) point of the mesh. The available field names for the Poisson and Schrodinger solvers are given in Table 5.3 for reference. Below is a list of each available type of response and its options. Each item heading specifies a possible value of a **Response** x parameter, and “options” refer to the parameter entries of the corresponding **ResponseParams** x list).

- **Center Of Mass** computes the center of mass of a field within a region. Computes 4 response values containing the x , y , and z coordinates of the center of mass and a dummy value of 1.0. Options are:

Name	Type	Description
Field Name	string	the “mass” field on which to compute the center of mass.
<i>Mesh Region Parameters</i>	–	any of the parameters in Table 5.2, which describe the region of the mesh over which to compute the center of mass.

- **Field Average** computes the average of a field within a region. Computes a single response value. Options are:

Name	Type	Description
Field Name	string	the field with which to compute the average value.
<i>Mesh Region Parameters</i>	–	any of the parameters in Table 5.2, which describe the region of the mesh over which to compute the average value.

- **Field Integral** computes the integral of a field within a region. Computes a single response value. Options are:

Name	Type	Description
Field Name	string	the field to integrate, when integrating a single field.
Field Name Im	string	if integrating a complex field, the field containing the imaginary part of the field to integrate, when integrating a single field.
Conjugate Field	bool	whether or not to conjugate the field before integrating, when integrating a single field.
Field Name x	string	where x is an integer index, starting at zero. Specifies the x -th term in a product of fields to integrate.
Field Name Im x	string	where x is an integer index, starting at zero. Specifies the imaginary part of the x -th term in a product of (complex) fields to integrate.

Name	Type	Description
Conjugate Field x	bool	whether or not to conjugate the x -th field before integrating a product of fields.
Return Imaginary Part	bool	whether or not to return (as the response value) the imaginary or real part of the specified integral.
Integrand Length Unit	string	specifies the (linear) length unit of the integrand. Allowed values are <code>m</code> (meters), <code>cm</code> (centimeters), <code>um</code> (microns), <code>nm</code> (nanometers), and <code>mesh</code> (the same units as the mesh). The default is centimeters.
Positive Return Only	bool	if “true” and integral is zero or negative, instead of returning the value of the integral return a huge number (like 10^{100}). Default is “false”. This option is used when optimizing a non-negative quantity.
<i>Mesh Region Parameters</i>	–	any of the parameters in Table 5.2, which describe the region of the mesh to integrate over.

- **Field Value** computes the value of a “return field” and the position where a second “op field” takes its maximum or minimum within a region. Computes 5 response values containing the value of the return field at the extremum, the value of the op field at the extremum, and the x , y , and z coordinates of the extremum. Options are:

Name	Type	Description
Operation	string	which extremum to find. Allowed values are <code>Minimize</code> and <code>Maximize</code> .
Operation Field Name	string	field on which to find extremum, if a scalar field.
Operation Vector Field Name	string	field on which to find extremum, if a vector field. In this case, the magnitude of the vector field, projected according to the “Operate on ...” options below is minimized or maximized.
Return Field Name	string	field whose value at the extremum should be returned as the response value, if a scalar field.
Return Vector Field Name	string	field whose magnitude at the extremum should be returned as the response value, if a vector field.
Operate on x-component	bool	whether to include the x-component contribution to the magnitude of a vector field being maximized or minimized. Default is “true”.
Operate on y-component	bool	whether to include the y-component contribution to the magnitude of a vector field being maximized or minimized. Default is “true”.
Operate on z-component	bool	whether to include the z-component contribution to the magnitude of a vector field being maximized or minimized. Default is “true”.
<i>Mesh Region Parameters</i>	–	any of the parameters in Table 5.2, which describe the region of the mesh to search for an extremum.

- **Region Boundary** computes the bounding box of a mesh region and saves its limits in a text file. Computes a single dummy response value (always equal to zero). Options are:

Name	Type	Description
<i>Mesh Region Parameters</i>	–	any of the parameters in Table 5.2, which describe the region of the mesh whose limits should be computed.
Output Filename	string	the name of the file into which the limits of the mesh region should be written.

- **Saddle Value** computes the value of a “return field” at the saddle point of another field. This response uses a modified nudged elastic band algorithm to locate the saddle point, and must be given starting and ending points and/or element blocks and a gradient field of the field for which the saddle point is being found. Computes 5 response values. Since there are so many options available for this response, we list them by categories. General options are:

Name	Type	Description
Field Name	string	the (scalar) field on which to find a saddle point.
Field Scaling Factor	double	scaling factor for the field. Default is -1.0 , as usually the Poisson potential is used here.
Field Gradient Name	string	the (vector) field giving the gradient of the field specified, possibly after scaling.
Field Gradient Scaling Factor	double	a scaling factor applied to the gradient field given to obtain the actual required gradient. Default is -1.0 , as usually the Poisson potential gradient is used here.
Return Field Name	string	the (scalar) field whose value at the saddle point should be returned as the first response value. If unspecified, the field used to find the saddle point will also be used as the return field. If set to the special value “current”, then a Green’s function based 1D current calculation is performed along the saddle path after it has been found.

Name	Type	Description
Return Field Scaling Factor	double	a scaling factor to be applied to the return field value before returning it. Default is 1.0.
Image Point Size	double	the size of the image points, in mesh units (quadrature point values contribute to image points via Gaussian weights around the actual image point. This specified the size of the Gaussian). Default is 0.01.
Number of Image Points	int	the number of image points to use in the nudged-elastic band (NEB) algorithm for finding the saddle point. Usually between 3 and 11. Default is 10.
Number of Final Points	int	Number of “final points”, which are the points at which the saddle point path is evaluated at the end of the NEB algorithm. By default, the same as the number of image points.
Max Time Step	double	maximum time step used in the NEB algorithm. Default is 1.0.
Min Time Step	double	minimum time step used in the NEB algorithm. Default is 0.002.
Maximum Iterations	int	maximum iterations (i.e. number of time steps) of the NEB algorithm. Default is 100.
Backtrace After Iteration	int	after this iteration, enable backtracing to enforce monotonic convergence. Default is no backtracing (disabled).
Convergence Tolerance	double	convergence tolerance of the NEB algorithm. Default is 10^{-5} .
Min Spring Constant	double	minimum spring (elastic band) constant of the NEB algorithm. Default is 1.0.
Max Spring Constant	double	maximum spring (elastic band) constant of the NEB algorithm. Default is 1.0.
Output Filename	string	name of file into which final saddle point results are written.
Debug Filename	string	name of file into which debugging information is written.

Name	Type	Description
Append Output	bool	whether or not to append or overwrite to the Output Filename. Default is “false”.
Climbing NEB	bool	whether or not to employ the “climbing” version of the NEB algorithm. Default is “true”.
Anti-Kink Factor	double	a factor, which if set to a positive value, penalizes kinks in the elastic band. Default is 0 (no kink suppression).
Aggregate Worksets	bool	if true, all workset data is aggregated at the onset of the algorithm. This requires more memory, but reduces the need for MPI communication and thereby speeds up the algorithm. Default is “false”.
Adaptive Image Point Size	bool	whether the image point size is allowed to vary. Default is “false”.
Adaptive Min Point Weight	double	if adaptive image point size is enabled, the minimum total weight (roughly the number of strongly contributing quadrature point value) allowed before enlarging the image point size. Default is 5.
Adaptive Max Point Weight	double	if adaptive image point size is enabled, the maximum total weight (roughly the number of strongly contributing quadrature point value) allowed before shrinking the image point size. Default is 10.
Lock to z-coord	double	if given, all the image points in the NEB method are restricted to having the specified z-coordinate.
z min	double	the minimum z-coordinate allowed in image points.
z max	double	the maximum z-coordinate allowed in image points.
Begin Point	Array(double)	the beginning location of the NEB algorithm’s elastic band.
Begin Element Block	string	the beginning element block of the NEB algorithm’s elastic band. The actual beginning point is found dynamically by minimizing the value of the saddle point field within the element block.
Begin Polygon	sub-list	the beginning element block of the NEB algorithm’s elastic band. The actual beginning point is found dynamically by minimizing the value of the saddle point field within the polygon.

Name	Type	Description
End Point	Array(double)	the ending location of the NEB algorithm's elastic band.
End Element Block	string	the ending element block of the NEB algorithm's elastic band. The actual ending point is found dynamically by minimizing the value of the saddle point field within the element block.
End Polygon	sub-list	the ending element block of the NEB algorithm's elastic band. The actual ending point is found dynamically by minimizing the value of the saddle point field within the polygon.
Percent to Shorten Begin	double	percentage to shorten the elastic band from its specified beginning location. Default is 0.0.
Percent to Shorten End	double	percentage to shorten the elastic band from its specified ending location. Default is 0.0.
Saddle Point Guess	Array(double)	an initial guess at the saddle point's location. The initial elastic band will be routed from the beginning point to the ending point through this point, if provided.
Debug Mode	int	debug verbosity level. Default is 0.

Options regarding the use of a level-set method, which is runs after the elastic band method and helps to even more precisely locate the saddle point are:

Name	Type	Description
Levelset Radius	double	radius levelset post-analysis. If greater than zero, after the NEB algorithm finds a saddle point, a level-set saddle-point-finding algorithm is performed in the mesh region that lies within the specified radius of the NEB's saddle point. This algorithm works by effectively filling the region with water until two pools join into one. Default is 0.0, which disables the level-set algorithm.
Levelset Field Cutoff Factor	double	this factor multiplied by the maximum field difference within the region gives a cutoff factor for level-set algorithm. When points are being considered for addition into existing "pools", only points within this energy from the current "water level" (i.e. energy) are considered. The default is 1.0, which means all points are considered. Setting this parameter to a value less than one will speed up the algorithm but may cause it to miss a point that should be added to a pool if that point lies on a strong local gradient.
Levelset Distance Cutoff Factor	double	this factor, multiplied by the average linear dimension of cells within the level-set region, gives a distance cutoff which sets the maximum distance for which two points are considered "neighbors", so that they will be considered part of the same "pool" if their field values are both below the "water level". Default is 1.0.
Levelset Minimum Pool Depth Factor	double	this factor, multiplied by half the difference between the NEB's saddle value and the minimum field value within the level-set region, set the minimum depth a pool must have before it is considered a "deep pool". The level-set method succeeds in finding a saddle point when two deep pools join into one. Default is 1.0.

Lastly, options dealing with the Green's Function method used to compute current through a saddle path (after it has been found) are:

Name	Type	Description
GF-CBR Method Grid Spacing	double	spacing, in mesh units, between the points to be used by the Greens Function - Contact Block Reduction method (for computing the current through the saddle path). Default is 0.0005, (0.5 nanometers if the mesh is in microns).
GF-CBR Method Vds Sweep	bool	whether a source-drain voltage sweep should be performed. Default is "false".
GF-CBR Method Vds Initial Value	double	initial value of source-drain voltage sweep. Default is 0.0.
GF-CBR Method Vds Final Value	double	final value of source-drain voltage sweep. Default is 0.0.
GF-CBR Method Vds Steps	int	number of steps in source-drain voltage sweep. Default is 0.
GF-CBR Method Eigensolver	string	the eigensolver used by the Greens Function - Contact Block Reduction method. Allowed values are <code>tq12</code> or <code>Anasazi</code> . Default is <code>tq12</code> , which should almost always be used.
GF-CBR Method Energy Cutoff Offset	double	an additive factor that raises the default maximum energy eigenvalue considered by the GF-CBR method. By default, the GF-CBR solves for eigenstates with eigenvalues up to $\max(\mu_S, \mu_D) + 20k_B T$, where μ_S and μ_D are the chemical potentials of the source and drain, respectively, and T is the temperature. By adding an offset to this value, more eigenstates will be solved for, giving a possibly more accurate calculation of the current but requiring more compute time. Units are electron volts. Default value is 0.5.

- **Save Field** saves a field into a state which is optionally included in the output Exodus mesh file. Computes a single dummy response value (always equal to zero). Options are:

Name	Type	Description
Field Name	string	field to save, if a scalar field.
Vector Field Name	string	field to save, if a vector field.
Vector Operation	string	what operation to perform on a vector field before saving. Allowed values are <code>magnitude</code> , <code>xyMagnitude</code> , <code>xzMagnitude</code> , <code>yzMagnitude</code> , <code>magnitude2</code> , <code>xyMagnitude2</code> , <code>xzMagnitude2</code> , <code>yzMagnitude2</code> , <code>xCoord</code> , <code>yCoord</code> , and <code>zCoord</code> . Default is <code>magnitude</code> .
State Name	string	name of state to save field into (shows up as this name in an Exodus file). Defaults to the same name as the field.
Output to Exodus	bool	whether or not field should be written to Exodus file. Default is “true”.
Output Cell Average	bool	whether or not to output a single averaged value per cell, or output the value of each quadrature point independently (resulting in multiple values per cell, and not as easy to visualize). Default it “true”.
Memory Placeholder Only	bool	if “true”, this response doesn’t actually do anything except reserve the memory for the specified state. This option is only used internally within QCAD. Default is “false”.

- **Solution Average** computes the average of the solution over the entire mesh. Computes a single response value. This response has no options and thus has no accompanying `ResponseParams` sub-list.

Parameter and Response Manipulation In addition to the `Parameters` and `Response Functions` sub-lists of the `Poisson Problem` and `Schrodinger Problem` lists, the user may also specify `Parameters` and `Response Functions` sub-lists of the root `Problem` list. These “master” parameter and response lists can be used select and manipulate how the parameters of the `Poisson Problem` and `Schrodinger Problem` lists get exposed as parameters of the overall QCAD simulation, and how the responses of the `Poisson Problem` and `Schrodinger`

Mode	Names of Pieces	Comments
Poisson	Poisson*	—
Schrodinger	Schrodinger*	—
Poisson Schrodinger	InitPoisson, Poisson*, Schrodinger, PoissonSchrodinger*	—
Schrodinger CI	Schrodinger, CoulombPoisson, CoulombPoissonIm	—
Poisson Schrodinger CI	InitPoisson, Poisson, Schrodinger, CIPoisson*, PoissonSchrodinger, CoulombPoisson, CoulombPoissonIm, NoChargePoisson, DeltaPoisson	—

Table 5.4. The names of the internal “pieces” of a QCAD simulation when run in each of the possible modes. Bracketed terms are only sometimes present, based on the input parameters. Asterisks mark the “default” pieces, which are used when no master parameter and/or response lists are given. Grayed items have parameters and responses that are no likely to be useful to the user.

Problem lists are passed through to become responses of the overall QCAD simulation. When a master parameter list is not supplied, the parameters of the default piece of the problem are exposed. When a master response list is not supplied, the responses of the default piece of the problem are passed through.

The structure of a master parameters list is that same as those of the Poisson Problem and Schrodinger Problem lists, except that the allowed names of the formal parameters take a special form. The value of a **Parameter** x XML parameter in the master **Parameters** list can in general be any number of copies of a function call, written as ‘ $fn(a,b,\dots)>$ ’ followed by an array-like reference to one or more of the formal parameters within a named piece of the QCAD problem, written as *ProblemPart*[*index*]. When the QCAD problem is given a (formal) parameter value, it manipulates that value using each of the function calls, then sets the indexed (formal) parameter(s) of the specified piece of the QCAD problem with the manipulated value.

The only currently available function name (fn) is **scale**, which takes a single argument and multiplies the parameter value by this argument. The available *ProblemPart* names depend on the type of QCAD problem being solved, but typically include one or both

of **Poisson** and **Schrodinger**. In square brackets, *index* can be a single index, *min:max* specifying all the indices from *min* to *max*-1, or even a comma-separated list of single- or range-type entries. Note that indices are 0-based, so that the first parameter of the Poisson part is referenced as **Poisson**[0]. Here are a few examples of allowed master (formal) parameter names:

- **Poisson**[0] - set the first (formal) parameter of the Poisson part of the problem.
- **Schrodinger**[0:3] - set the first three (indices 0, 1, and 2) parameters of the Schrodinger part.
- **scale**(10)>**Poisson**[0,2] - multiply the parameter by 10, then set the first and third (indices 0 and 2) parameters of the Poisson part.

The structure of a master response list is that same as those of the Poisson Problem and Schrodinger Problem lists, except that the allowed names of the responses take a special form and there are no **ResponseParams** *x* sub-lists. The value of a **Response** *x* parameter in the master response list is in general of the form $fn(x_1, \dots, x_n)$, where each x_i can be either a double-precision value (e.g. 2.1) or a reference to a response of some part of the problem, written *ProblemPart*[*index*]. The available names and syntax for *ProblemPart* and *index* are the same as for the formal parameter names described above, with the additional option of specifying **Eigenvalue** as the *ProblemPart* in problems involving Schrodinger parts. It should be noted that that *index* indexes response values, not entire responses. For example, if the first Response of the Poisson part is a Field Value response (specified in the **Poisson Problem's Response Functions** sub-list), then **Poisson**[0] through **Poisson**[4] refer to the five response values computed by the single Field Value response. The response values generated by a master response correspond to the function evaluated at the response values or fixed values given as function parameters. Some functions may compute multiple values, in which case that master response computes multiple values (similar to a Field Value response). In many cases, there is no need for any manipulation of the responses, and syntax is simplified to just *ProblemPart*[*index*]. Allowed function names *fn* are:

Function Signature	Description
<code>dist(x_1, x_2)</code>	computes the distance between points in 1D. Returns a single double-precision value.
<code>dist(x_1, y_1, x_2, y_2)</code>	computes the distance between points in 2D. Returns a single double-precision value.
<code>dist($x_1, y_1, z_1, x_2, y_2, z_2$)</code>	computes the distance between points in 3D. Returns a single double-precision value.
<code>scale(s_1, s_2)</code>	multiplies its two arguments together and returns the result (a single double-precision value).
<code>divide(s_1, s_2)</code>	divides s_1/s_2 and returns the result (a single double-precision value).
<code>nop(...)</code>	takes any number of arguments and echos them as return values. Thus, this function returns as many double-precision values as it has arguments. Note that the simplified syntax above just uses <code>nop</code> as the function name implicitly.
<code>DgDp(<i>ProblemPart</i>, <i>pIndex</i>, <i>gIndex</i>)</code>	returns the derivative of the <i>ProblemPart</i> part's <i>gIndex</i> -th response value with respect to its <i>pIndex</i> -th parameter as a single double-precision value. This response function is special in that its first argument is the un-indexed name of a part of the problem. Note that <i>gIndex</i> is an index of response values, so that responses that produce multiple values (e.g. the Field Value response) need to be considered accordingly.

The Discretization list

This list specifies how the QCAD problem's domain is discretized into finite elements, usually by referencing an Exodus-formal mesh file. Alternatively, a standard mesh (called an STK mesh in QCAD) in 1D, 2D, or 3D can be generated without a mesh file by setting the appropriate parameters within this list. The important parameters and their types for the Discretization list are:

Name	Type	Description
Method	string	what type of input mesh is being used. For Exodus meshes, use <code>Ioss</code> . For “standard” meshes in 1D, 2D, or 3D, use <code>STK1D</code> , <code>STK2D</code> , or <code>STK3D</code> , respectively.
Exodus Input File Name	string	only applicable to the <code>Ioss</code> method. The path and filename to the input Exodus-format mesh file.
Exodus Output File Name	string	The path and filename to the output Exodus-format mesh file. Note that this parameter is available for all values of the Method parameter.
Use Serial Mesh	bool	only applicable to the <code>Ioss</code> method. When true, an exodus mesh will be read in from the single .exo file named as the input file, even when running on multiple processors. When false and running on $N > 1$ processors, QCAD will try to read in files that have the same names as the specified input exodus file but end in “. $N.x$ ”, where x runs from 0 to $N - 1$.
1D Elements	int	The number of elements along the first dimension of an STK mesh.
1D Scale	double	The scaling of elements along the first dimension of an STK mesh.
2D Elements	int	The number of elements along the second dimension of an STK mesh.
2D Scale	double	The scaling of elements along the second dimension of an STK mesh.
3D Elements	int	The number of elements along the third dimension of an STK mesh.
3D Scale	double	The scaling of elements along the third dimension of an STK mesh.

The Piro list

The Piro sub-list contains under it all solver-related parameters organized into a complicated tree based on the Trilinos (the library QCAD is based on) package names of the com-

ponents being used. Currently this User Guide does not provide details about setting these parameters, and the reader is referred to the Trilinos documentation (see trilinos.sandia.gov). If the **Piro Defaults Filename** parameter under the Problem list is set, then QCAD takes the Piro parameter list found in the given filename as an initial (default) set of Piro parameters, and any parameters listed in the input file's Piro list override those in the default list. If no values need to be overridden, the Piro list can be omitted altogether. This allows the user to specify a typical or common set of solver parameters in a separate XML file which can be re-used in many input XML files. Moreover, the QCAD installation comes with a **default.piro.params.xml** file containing parameters appropriate to most QCAD problems. This file allows users to start with a good typical set of parameters and only tweak them when necessary. Finally, comments in the **default.piro.params** file provide brief documentation of what the different solver parameters do.

Material file specification

The material database file, or “materials XML” file, is an XML-formatted file that specifies material properties and associates them with mesh entities such as element blocks, node-sets, and sidesets. The basic structure of a materials database file consists of four major sections (each contained in a parent **ParameterList** XML block):

Materials defines material names and properties. Each material is specified by a named sub-list of the parent Materials section list. Every material sub-list must have a parameter named **Category** with value **Semiconductor**, **Insulator**, or **Metal**. Based upon this categorization, the material sub-list is expected to have other certain parameters defined.

In **Semiconductor** material lists, one must define:

Name	Type	Description
Permittivity	double	the relative permittivity of the material.
Conduction Band Minimum	string	the shapes of the conduction band minima. Allowed value are Delta2 Valley (applicable to Silicon), and Gamma Valley (applicable to GaAs).
Number of conduction band min	int	number of conduction band minima. Equal to 2 for MOS 2DEGs (not 6, since vertical confinement splits valleys) and 1 for GaAs.
Number of valence band max	int	number of valence band maxima.
Longitudinal Electron Effective Mass	double	in units of the electron rest mass.
Transverse Electron Effective Mass	double	in units of the electron rest mass.
Light Hole Effective Mass	double	in units of the electron rest mass.
Heavy Hole Effective Mass	double	in units of the electron rest mass.
Electron DOS Effective Mass	double	in units of the electron rest mass. Includes valley degree of freedom.
Hole DOS Effective Mass	double	in units of the electron rest mass. Includes valley degree of freedom.
Electron Affinity	double	in units of electron volts.
Zero Temperature Band Gap	double	in units of electron volts.
Band Gap Alpha Coefficient	double	in units of eV/K.
Band Gap Beta Coefficient	double	in units of Kelvin.
Reference Temperature	double	in Kelvin. The reference temperature for computing N_c and N_v .

In **Insulator** material lists, one must define:

Name	Type	Description
Permittivity	double	the relative permittivity of the material.
Number of conduction band min	int	number of conduction band minima. Usually set equal to 1. Used when the insulator is included in the quantum region of a mesh.
Longitudinal Electron Effective Mass	double	in units of the electron rest mass. Used when the insulator is included in the quantum region of a mesh. Usually assumed equal to 0.5 in insulators.
Transverse Electron Effective Mass	double	in units of the electron rest mass. Used when the insulator is included in the quantum region of a mesh. Usually assumed equal to 0.5 in insulators.
Band Gap	double	in units of electron volts.
Electron Affinity	double	in units of electron volts.

In **Metal** material lists, one must define:

Name	Type	Description
Permittivity	double	the relative permittivity of the material.
Work Function	double	in units of electron volts.

ElementBlocks defines the properties of the different possible element blocks that appear in mesh files. Information about each element block is given in a sub-list of the element block’s name beneath the parent ElementBlocks list. Typical values for an element block list are the following:

Name	Type	Description
material	string	the name of the material the element block is made of. Must be the name of a material list under the Materials parent list.
Dopant Type	string	the type of dopant, if any. Allowed values are Donor and Acceptor . Only applicable in semiconductors.
Doping Profile	string	the doping profile. Currently, only Constant is allowed.
Doping Value	double	the concentration of dopants in cm^{-3} (always a positive number).
Doping Parameter Name	string	a user-defined name that may be listed as a formal parameter of a Poisson problem piece. If this parameter is set, then Doping Value should not be, and vice versa.
Charge Value	double	the density of fixed charge (can be positive or negative) within the element block. Only applicable in insulators.
Charge Parameter Name	string	a user-defined name that may be listed as a formal parameter of a Poisson problem piece. If this parameter is set, then Charge Value should not be, and vice versa.
Charge Parameter Scaling	double	a scaling factor applied to the value of the Charge Parameter Name formal parameter to get the actual density of fixed charge within the element block. Only applicable when Charge Parameter Name is given.
quantum	bool	whether the element block is tagged as part of the “quantum region” of a mesh.

Element block lists may also override any of the material parameters found in a Material list. When QCAD looks for a property of an element block, such as the permittivity, it looks first in the element block’s sub-list for a Permittivity parameter and then in the Material list named by the element block’s “material” parameter. Indeed, there is no need to specify a an element block’s material if all of the material properties needed

are given in the element block's list itself.

NodeSets defines the properties of the different possible nodesets that appear in mesh files. Information about each nodeset is given in a sub-list of the nodeset's name beneath the parent NodeSets list. Typical values for a nodeset list are the following:

Name	Type	Description
<code>material</code>	<code>string</code>	the name of the material the nodeset's nodes belong to. If the nodes are along an interface between materials, the material that is logically setting the boundary condition should be specified. For instance, if the nodeset is used to set the voltage on an aluminum electrode, and its nodes border aluminum and silicon dioxide, aluminum should be listed as the nodeset's material. The value must be the name of a material list under the Materials parent list.
<code>elementBlock</code>	<code>string</code>	the name of the element block the nodeset's nodes belong to. If the nodes are along an interface between element blocks, the element block that is logically setting the boundary condition should be specified. For instance, if the nodeset is used to set the voltage on an p+ ohmic contact, and its nodes border the "pPlusOhmic" and "substrate" element blocks, "pPlusOhmic" should be listed as the nodeset's element block. Typically this parameter is used instead of specifying the name of a material directly, and is the only option when the relevant doping information is contained in an element block specification.

SideSets defines the properties of the different possible sidesets that appear in mesh files. Information about each sideset is given in a sub-list of the sideset's name beneath the parent SideSets list. The same values used for a nodeset lists apply to sideset lists, and we refer the reader to the NodeSets section above.

In addition to these lists, within the root **ParameterList** of the materials XML file one must define the two string parameter **Reference Material** and **Quantum Material** as the names of existing materials. The Reference Material parameter sets the reference potential to be used in solving Poisson problems. The reference material is typically taken to be

the material of the bulk of the device, and should be the material that contains the quantum electrons, if any. The Quantum Material parameter is used in Poisson Schrodinger modes, and should be the material containing quantum electrons. Thus, ideally the Reference Material and Quantum Material will be the same.

Mesh file specification

QCAD uses mesh files in the Exodus format. While any tool capable of generating Exodus-format meshes should work, QCAD has been tested using meshes created by Sandia National Labs CUBIT software package. The main requirements for creating a valid QCAD mesh are that the mesh be completely divided into named element blocks (that is, no part of the mesh is not contained in a named element block), and that nodesets and/or sidesets can be defined at locations where boundary conditions exist.

Output Files

In addition to outputting progress and response information to the console, QCAD typically writes one or more Exodus-format mesh files upon completion which contain computed field such as potentials and wave-functions. The names of the output mesh file is specified in the Discretization section of the input XML file (see section 5). In order to view the output Exodus files, the writers recommend using the Paraview application (www.paraview.org). In addition to the output mesh file, some QCAD responses (namely, the Region Boundary and Saddle Value responses) optionally write data to text files specified in their parameter lists.

QCAD’s “multi-shot” mode

Any of the QCAD simulation modes can be run either in “single-shot” mode, in which the problem described in the input XML file is run once using the values given in the input file (e.g. the boundary conditions), or in “multi-shot” mode, in which the problem described in the input XML file is run multiple times for different sets of formal parameters (for a discussion of formal parameters, see section 5). How many times the QCAD problem is solved and for which formal parameters is determined by the Sandia-developed DAKOTA package, and depends on what type of multi-shot simulation is desired. When QCAD is run in multi-shot mode, one can think of the simulation as being composed of a DAKOTA wrapper around a “QCAD core” which solves the QCAD problem given a set of formal parameters. The DAKOTA wrapper has its own input file (distinct from the input XML file), and calls the QCAD core (usually) multiple times as it follows an algorithm specified in its input file. A few of the algorithms, or “methods” that the DAKOTA wrapper is capable of performing are:

- **Vector parameter study:** the QCAD core is run for a series of parameters which lie along a line segment in (multi-dimensional) parameter space. The initial and final parameter points are given, as well as the number of intervals.
- **Multi-dimensional parameter study:** the QCAD core is run for a series of parameters which form a multi-dimensional grid in parameter space. The minimum and maximum values for each independent parameter are given, as well as the number of intervals along each parameter direction.
- **Non-linear least squares parameter optimization:** a number of target response values are given, and the QCAD core is run with varying parameters in order to minimize a sum-of-squares expression where each term is the difference between an actual response value and its target. In addition to the target values, an initial point in parameter space and weights for each sum-of-squares term are given.

DAKOTA is capable of many more methods that we do not list here, and we refer the interested reader to the DAKOTA Users guide (available at dakota.sandia.gov).

How to run QCAD in “multi-shot” mode

Running QCAD in multi-shot mode is identical to running QCAD in single-shot mode apart from a few exceptions:

1. **DAKOTA input file:** A DAKOTA input file must be created, which specifies what method DAKOTA uses (e.g. whether a vector study or parameter optimization is performed). See section 5 below for example DAKOTA input files.
2. **QCAD input XML file:** An `Analysis` block needs to be added to the QCAD input XML file under the “Piro” sub-list that specifies the DAKOTA input file as well as an output file. Specifically, this list has the form:

```
<ParameterList name="Analysis">
  <Parameter type="string" name="Analysis Package" value="Dakota"/>
  <ParameterList name="Dakota">
    <Parameter type="string" name="Input File" value="dakota_input_file.txt"/>
    <Parameter type="string" name="Output File" value="dakota_output_file.txt"/>
  </ParameterList>
</ParameterList>
```

and should be placed inside the `Piro` list (a `Piro` list may need to be added if it doesn't already) of the the QCAD input XML file.

3. **Executable is AlbanyAnalysis:** Finally, the QCAD simulation must be run using the `AlbanyAnalysis` executable, rather than the `Albany` executable. The executable's command line arguments are the same, so for example, the command for running a multi-shot QCAD simulation could be `/path/to/AlbanyAnalysis input.xml`.

DAKOTA input file examples

This section contains example DAKOTA input files for using vector study and parameter optimization methods. In both cases, the user specified the number of parameters, and for parameter optimizations the user specifies the number of responses. When required, these numbers must match the number of formal parameters and responses of the QCAD core, i.e., the numbers in the “master” lists directly under the **Problem** list (cf. section 5). For more details regarding DAKOTA input files, see the DAKOTA Users Guide.

Vector study example The following listing is an example DAKOTA input file for a vector study QCAD run.

```
method,                                # THESE ARE COMMENTS; don't include in actual input file
  vector_parameter_study                # Name of the method to use
  final_point = 1.0 2.0                 # final parameter values
  num_steps = 3                         # number of intervals

variables,
  continuous_design = 2                 # number of parameters
  cdv_descriptors 'p1' 'p2'            # labels for these parameters (optional)
  initial_point 0.0 1.0                # initial parameter values

interface,                             # the 'interface' section is always the same
  direct                               # for QCAD simulations. Don't worry about
  analysis_driver = 'Albany_Dakota'    # what these lines mean.
  evaluation_servers = 1

responses,
  num_objective_functions = 3           # the number of responses
  response_descriptors = 'r1' 'r2' 'r3' # labels for these responses
  no_gradients                    # since this is a vector study, we don't need
  no_hessians                     # gradients or hessians
```

Non-linear least squares parameter optimization example The following listing is an example DAKOTA input file for a parameter optimization QCAD run.

```
method,
  nl2sol                                # non-linear least squares method
  max_iterations = 100                  # maximum iterations before giving up
  convergence_tolerance = 1.0e-5       # convergence tolerance for sum-of-squares expression

variables,
  continuous_design = 2                 # number of parameters
  cdv_lower_bounds 2.0 0.0              # lower bounds of parameters
  cdv_upper_bounds 10.0 5.0            # upper bounds of parameters
  initial_point 5.0 2.0                # initial parameter point
  cdv_descriptors 'p1' 'p2'            # labels for parameters
```

```

interface,                                # the 'interface' section is always
  direct                                  # the same for QCAD simulations
  analysis_driver = 'Albany_Dakota'
  evaluation_servers = 1

responses,
  num_least_squares_terms = 2             # number of responses
  response_descriptors = 'r1' 'r2'        # labels for responses
  least_squares_weights = 1.0 1.0         # weights for least squares terms
  analytic_gradients              # tells DAKOTA that QCAD supplies analytic derivatives
  no_hessians                      # but does not supply hessian matrices
  least_squares_data_file = 'dak.data' freeform # dak.data is a data file containing
                                              # the target response values, one per line.

```

Simple examples

2D Poisson problem

Below is a simple input file for running a semiclassical Poisson simulation of a device given by the 2-dimensional mesh file “myMesh.exo” and placing the output in “output.exo”.

Listing 5.2. Example 2D Poisson QCAD input file.

```

<ParameterList>
  <ParameterList name="Problem">
    <Parameter name="Solution Method" type="string" value="QCAD Multi-Problem" />
    <Parameter name="Name" type="string" value="Poisson 2D" />

    <Parameter name="Length Unit In Meters" type="double" value="1e-6"/>
    <Parameter name="Temperature" type="double" value="300"/>
    <Parameter name="MaterialDB Filename" type="string" value="materials.xml"/>
    <Parameter name="Piro Defaults Filename" type="string" value="default_piro_params.xml"/>

  <ParameterList name="Parameters" />
  <ParameterList name="Response Functions" />

  <ParameterList name="Poisson Problem">
    <ParameterList name="Dirichlet BCs">
      <Parameter name="DBC on NS substrate for DOF Phi" type="double" value="0" />
      <Parameter name="DBC on NS gate for DOF Phi" type="double" value="+1" />
    </ParameterList>
  </ParameterList>
</ParameterList>

<ParameterList name="Discretization">
  <Parameter name="Method" type="string" value="Ioss" />
  <Parameter name="Exodus Input File Name" type="string" value="myMesh.exo" />
  <Parameter name="Exodus Output File Name" type="string" value="output.exo" />
</ParameterList>

```



```
</ParameterList>
```

This input file specifies that QCAD should solve a two-dimensional (semiclassical) Poisson problem. The length units of the mesh are microns ($1\text{e} - 6$ meters) and the energy unit is the default of electron volts. The associated material database XML file is “materials.xml”. There are no Parameters or Responses specified. Under the Poisson Problem list, boundary conditions (*e.g.*, applied voltages) are specified, in this case two Dirichlet-type boundary conditions on the nodesets named “substrate” and “gate”, which are present in the input mesh file, myMesh.exo. There is no Piro sub-list, which means that the solver parameters are taken entirely from the set of default values in “default_piro_params.xml” (provided by QCAD). In most cases, these default values are fine.

2D Schrodinger problem

Below is a simple input file for running a semiclassical Poisson simulation of a device given by the 2-dimensional mesh file “myMesh.exo” and placing the output in “output.exo”.

Listing 5.3. Example 2D Schrodinger QCAD input file.

```
<ParameterList>
  <ParameterList name="Problem">
    <Parameter name="Solution Method" type="string" value="QCAD Multi-Problem" />
    <Parameter name="Name" type="string" value="Schrodinger 2D" />

    <Parameter name="Length Unit In Meters" type="double" value="1e-6"/>
    <Parameter name="Energy Unit In Electron Volts" type="double" value="1e-3"/>
    <Parameter name="MaterialDB Filename" type="string" value="materials.xml"/>
    <Parameter name="Piro Defaults Filename" type="string" value="default_piro_params.xml"/>

    <Parameter name="Number of Eigenvalues" type="int" value="5"/>
    <Parameter name="Only solve schrodinger in quantum blocks" type="bool" value="false"/>

  <ParameterList name="Parameters" />
  <ParameterList name="Response Functions" />

  <ParameterList name="Schrodinger Problem">

    <ParameterList name="Potential">
      <Parameter name="Type" type="string" value="String Formula" />
      <Parameter name="Formula" type="string" value="10*((x-0.5)^2 + (y-1.0)^2)" />
      <Parameter name="Scaling Factor" type="double" value="1.0" />
    </ParameterList>

    <ParameterList name="Dirichlet BCs">
      <Parameter name="DBC on NS boundary for DOF psi" type="double" value="0" />
    </ParameterList>

  </ParameterList>
```

```

</ParameterList>

<ParameterList name="Discretization">
  <Parameter name="1D Elements" type="int" value="50"/>
  <Parameter name="1D Scale" type="double" value="1.0"/>
  <Parameter name="2D Elements" type="int" value="50"/>
  <Parameter name="2D Scale" type="double" value="2.0"/>
  <Parameter name="Method" type="string" value="STK2D"/>
  <Parameter name="Exodus Output File Name" type="string" value="output.exo"/>
</ParameterList>

</ParameterList>

```

This example solves the time-independent Schrodinger equation on a 2D rectangular mesh with a centered parabolic potential. The lowest five eigen-energies are found in units of meV, and the mesh coordinates are in units of microns. Setting “Only solve schrodinger in quantum blocks” to false means that QCAD pays no attention to the “isQuantum” parameter of an element block in the materials XML file, and simply solves the Schrodinger equation on the entire mesh. The 2-dimensional mesh is generated internally (there’s no Exodus input file). It has 50 intervals between 0 and 1.0 in the x-dimension and 50 intervals between 0.0 and 2.0 in the y-dimension.

Chapter 6

QCAD Developer Guide

Albany and Trilinos Primer

Trilinos Overview

Trilinos is the (large) library of tools and functionality underlying QCAD. It contains everything from common data types (e.g. Vectors) to sophisticated solvers. Listed below are some of the Trilinos components most used by QCAD.

- **NOX** Non-linear solver.
- **Belos, AztecOO** Linear solvers.
- **Anasazi** Eigensolver.
- **Teuchos** Utility objects, e.g. ParameterLists.
- **Intrepid** Automatic differentiation.
- **LOCA** Continuation analysis.
- **Phalanx** Fields, field manager, and evaluators.
- **Piro** Packages Trilinos solvers into ModelEvaluator interfaces.
- **Stratimikos** Gives a united interface to linear solvers.
- **Epetra** vector objects with MPI support.

Model Evaluators

The `EpetraExt::ModelEvaluator` Trilinos class is central to Albany and QCAD. `EpetraExt::ModelEvaluator` defines a very generic interface for “evaluating” some type of model. Essentially, a model evaluator is a black box with a specific interface for passing data in and getting data back out. The inputs are a number of Epetra vectors called “parameters”, and the outputs are another set of Epetra vectors called “responses”.

Albany Overview

Albany is the name of a finite-element code framework which sits on top of Trilinos and makes it easier to rapidly develop finite-element applications. To the outside world, an Albany application looks like a `ModelEvaluator`, which is essentially a black box with a specific interface for passing data in and getting data back out. The Albany framework supplies much of the innards required to make a black box that solves some set of equations on a finite element mesh. By using the Albany framework, much of the structure of QCAD is inherited from Albany, and it is thus a basic understanding of Albany is essential to QCAD development.

Albany::SolverFactory

An `Albany::SolverFactory` object creates a solver object, which presents itself as a `ModelEvaluator`. To initialize the solver object, the solver factory typically creates an `Albany::Application` object and wrapper `Albany::ModelEvaluator` object and passes this model evaluator to the solver object. When the solver object is asked to evaluate itself (via its `evalModel(...)` function), it evaluates (usually repeatedly) the `Albany::ModelEvaluator` object that was passed to it upon initialization.

Albany::Application

The `Albany::Application` class is central to an Albany application (i.e. a finite-element based solver which uses the Albany framework). Although it is not a `ModelEvaluator`, it contains most of the code required to implement a model evaluator. `Albany::ModelEvaluator` (which is a `ModelEvaluator`) is a thin wrapper around `Albany::Application`.

Upon initialization, an `Albany::Application` instance creates via an `Albany::ProblemFactory` object an instance of an `Albany::AbstractProblem`-derived class (which class is chosen based on the “name” parameter in the Problem sub-list of the input XML file). This “problem class” is created by the Albany user and essentially defines the set of equations to solve and the set quantities to compute on the finite-element mesh. Specifically, the problem class specifies which evaluators are created. After the problem creates its evaluators, the `Albany::Application` instance hands them to multiple field managers, which perform the “fill” operations that ultimately compute the residual, jacobian, etc. requested by calls to the wrapper `Albany::ModelEvaluator`’s `evalModel(...)` function.

Note that the `Albany::Application` class is fully implemented and is not (typically) modified when creating a project (like QCAD) that utilizes the Albany framework. Instead, the project adds new problem classes and evaluators which dictate what set of equations is being solved.

Problems

We refer to C++ classes derived from the abstract base class `Albany::AbstractProblem` as “problem classes”, and such classes define what sort of problem is to be solved on the finite element mesh setup by the Albany framework. The problem class specifies the number of equations, the names of the “degrees of freedom”, and is responsible for constructing the evaluator objects used to ultimately evaluate the residual of the equations being solved (see the `constructEvaluators` function, which is indirectly called by the `Albany::AbstractProblem` function `buildEvaluators`). The problem class essentially orchestrates the creation of a set of evaluators as one would build a tower out of blocks. Upon construction, some evaluators take a set of parameters which can specify which quantities it should act upon, allowing the problem designer to use a evaluator “block” in a flexible way.

Evaluators

The term “evaluator” refers to a C++ class (or instance of such a class) derived from `PHX::Evaluator`. We will use the term “field” to refer to a named set of values on the finite-element mesh, either at all nodal points or at all quadrature points. Fields can be either of vector or scalar type. Evaluator objects *evaluate*, that is, compute, one or several fields given (possibly) other fields as input. This is done by calling the evaluator’s `evaluateFields(...)` function. An evaluator specifies which fields it depends upon (i.e. which need to be evaluated before it’s `evaluateFields(...)` is called) and which fields it computes. One should think of an evaluator as a logical part of a computation, and thus think of breaking up a large calculation (e.g. of the residual of an equation) into logical units each corresponding to a separate evaluator. This is what is done within a problem class when it constructs the set of evaluators needed to compute the residuals of equations. A `PHX::FieldManager` object is used to evaluate a list of evaluators, and determines an ordering (if one exists) for evaluating the evaluators such that all dependencies are met. Instances of the `Albany::Application` class contain field manager objects which are used to evaluate, for instance, the residual or Jacobian of the equations at hand.

Responses

The term “response” refers generally to some kind of output within the Albany framework, borrowing from the `ModelEvaluator` terminology although it’s precise meaning can depend on its context (which can be confusing). An `Albany::Application` object creates a number of “response functions”, which are objects derived from `Albany::AbstractResponseFunction`. Which response functions are created is specified in the input XML file. Each of the response functions can be asked to evaluate itself, an operation which performs a separate field manager fill, to produce some number of numerical (double-precision) values termed “response values”. Each response function belongs to a response vector (also specified in the input file), and the contents of a response vector are given by concatenating all of the response values

from the responses belonging to that vector. Each of these response vectors correspond to a “response” in the ModelEvaluator sense (i.e. if a ModelEvaluator says it supports 2 responses, it means 2 response vectors).

Response functions can also generate output by writing field data to the Albany::StateManager object contained in the Albany::Application object. There are some general purpose “stock” response functions included in the Albany framework, and projects that require more customized output are able to implement new responses for this purpose.

QCAD code structure

Overview

This section gives a description of the code added to (or on top of, if you like) the Albany framework as part of the QCAD project. The two fundamental QCAD simulation types, “Poisson” and “Schrodinger”, are implemented as problem classes in the standard way within Albany. The composite simulation types, “Poisson-Schrodinger”, “Schrodinger-CI”, and “Poisson-Schrodinger-CI” are implemented via QCAD::Solver objects. A QCAD::Solver object is not a Problem object but a Model Evaluator, and thus the implementation of the composite simulation types falls somewhat outside the usual method implementing a new type of problem/simulation in Albany. Similarly, the integrated Poisson-Schrodinger solver (which optionally runs at the end of a Poisson-Schrodinger type simulation) is implemented with a Model Evaluator class, QCAD::CoupledPoissonSchrodinger, rather than a Albany::AbstractProblem derived class.

Separate from the Problems and Model Evaluators used to setup the systems of physics equations to be solved, the QCAD project adds a number of Response classes to the code base. These responses were designed solely for QCAD, but may have broader application. The Response classes are for the most part small chunks of straightforward code, with the exception of the “SaddleValue” response.

In the sections that follow, we describe the two problem classes first, then the response classes, and finally the model evaluator classes used in composite-type simulations.

Problems

Poisson Problem

The QCAD::PoissonProblem class sets up the solution of Poisson’s equation.

$$\nabla^2 \phi = \rho_{cl}(\phi) + \rho_{quantum}(\Psi_i, E_i) \quad (6.1)$$

The evaluation of the equation is broken into evaluators as follows:

- QCAD::PoissonResid evaluates the entire residual: LHS - RHS.
- QCAD::PoissonSource evaluates the source term (RHS) of the Poisson equation, and also scales itself according to the units of ϕ on the LHS. Most of the heavy lifting is done by this evaluator.
- QCAD::Permittivity evaluates the permittivity, ϵ .

Schrodinger Problem

The QCAD::SchrodingerProblem class sets up the solution of the Schrodinger equation.

$$\mathcal{H}\Psi = E\Psi \quad (6.2)$$

Because this is an eigenvalue equation, it can not be solved using the usual way of minimizing a residual. Anasazi is an eigensolver in Trilinos, but instead of calling Anasazi directly, QCAD makes use of LOCA to solve the eigenproblem. LOCA is a general package for parameter continuation, that is, evaluating a model repeatedly as a single parameter changes. After evaluating the model LOCA performs a stability analysis on time-dependent problems, that is, it solves the eigenproblem $dF(X)/dX\vec{v} = \lambda dF(X)/d\dot{X}\vec{v}$. For example, for the heat equation, where $F(T) = dT/dt - d^2T/dx^2$, LOCA would solve a finite element version of $d^2/dx^2\vec{v} = \lambda\vec{v}$. The sign of each eigenvalue determines whether the given eigenvector mode is stable or unstable (in time). In the case of the Schrodinger Problem, we don't need any parameter continuation, and so LOCA is told to step at most a single time. The residual that is computed is $F(Psi) = d\Psi/dt - H * \Psi$, and the initial guess of $\Psi = 0$ solves the residual minimization immediately, leaving only the eigensolve portion, $H * \vec{v} = \lambda M * \vec{v}$, where M is the mass matrix. Thus, through the use of LOCA, the Schrodinger problem can be cast in the usual compute-a-residual form, though this form is somewhat non-intuitive (e.g. it includes time derivatives when the Schrodinger equation being solved does not).

The evaluation of the equation is broken into evaluators as follows:

- QCAD::SchrodingerResid evaluates the entire residual
- QCAD::SchrodingerPotential evaluates the potential term of the Schrodinger Hamiltonian, unless it is directly imported from an auxiliary vector, in which case other evaluators import the potential.

Mesh Regions

The QCAD::MeshRegion class encapsulates a set of cells, or “region” , of a finite element mesh. There are several ways to define the region, including the use of static coordinate

limits, element blocks, or by thresholding a field. In the end, the MeshRegion object can be queried as to whether a set of coordinates lies inside its region or not, and consolidates this logic. Several responses utilize MeshRegion objects to determine which areas of the mesh to act upon, and the PoissonSource evaluator uses MeshRegion objects to implement scaling factors that only apply to portions of the mesh (“Mesh Region” parameters of the Poisson Problem).

Responses

Here we give a synopsis of each of the Response classes added by QCAD, and any relevant code comments.

Field Integral

The Field Integral response integrates a specified field or product of fields, possibly complex, over a region of the mesh. The region is specified using a MeshRegion object, and conjugation can be applied to complex fields. This response computes a single double-precision quantity. If the integral is complex, a response parameter indicates whether the real or imaginary part should be returned.

Field Average

The Field Average response is similar to the Field Integral response except it divides the result by the volume of the region being averaged over. Computes a single double-precision quantity. Useful, for example, for extracting the average of the conduction band within a small region of the mesh centered about a point of interest.

Field Value

The Field Value response performs some operation, namely the minimization of maximization of a field on a MeshRegion. This response computes the value of the field being minimized/maximized at the extrema, the value of the x, y, and z, coordinates of the extrema, and also the value of a “return field” at the extrema, which need not be the field that is minimized/maximized. Thus, this response computes five double-precision quantities that are output in the order:

1. Extrema value of the “return field”
2. Extrema value of the field being operated on
3. x-coordinate of extrema

4. y-coordinate of extrema
5. z-coordinate of extrema

This response allows one to compute, for instance, the value of the conduction band at the maximum of the electron density within a region.

Center Of Mass

The Center Of Mass response computes the average coordinate within a MeshRegion weighted by a specified field (i.e. the center of mass, with a specified field being the “mass”). This response computes 4 double-precision values; the first three are the x, y, and z coordinates of the center of mass, and the final value is a dummy value always equal to one (1.0). This last value is needed to keep track of the volume divide by in order to compute the center of mass coordinates. The Center of Mass response could be used, for example, to find the approximate position of a quantum dot by taking the center of mass of the electron density.

Save Field

The Save Field response, when evaluated, saves a specified field into an Albany::StateManager state. The memory for these states is owned by the Discretization object, and take values on the the cell quadrature points. When saving a vector field, the Save Field response can compute various operations such as taking the magnitude of the vector and pulling out a single component. This response computes a single dummy double-precision value which is always equal to zero.

Region Boundary

The Region Boundary response computes the smallest box containing all the cells of a specified MeshRegion and outputs the minimum and maximum coordinate values to a specified text file. This response computes a single dummy double-precision value which is always equal to zero. **;- Check this**

Saddle Value

The Saddle Value response executes a nudged elastic band algorithm to find the saddle point in a specified field given beginning and ending points or regions. It then, optionally, performs a 1D Greens function transport solution using the value of the specified field along the saddle path. There are many options that define the behavior of this very complicated

response, and these are detailed in the QCAD User’s Guide. A central challenge in implementing this response is that the saddle point finding algorithm needs access to the entire region being considered, which cannot be assumed to lie on a single processor. Thus, there must be MPI communication to update the position of the elastic band.

Coupled Solvers: QCAD::Solver

In addition to the individual Poisson and Schrodinger problems, there is a need to solve problems in which one iterates between a Poisson solution and Schrodinger solution. The problem classes (derived from Albany::AbstractProblem) are not intended to contain sub-problems, or sub-solvers for that matter, and thus creating a problem class to handle simulations which couple Poisson and Schrodinger problems is not appropriate. Instead, something more akin to a numerical solver (like NOX), which presents itself as a ModelEvaluator is needed. The QCAD::Solver class is a EpetraExt::ModelEvaluator-derived class which essentially wraps around multiple sub-Albany::Application objects. When a QCAD::Solver object is constructed is creates via Albany::SolverFactory ModelEvaluators for the *solvers* of its sub-Albany::Applications. When a QCAD::Solver is evaluated (by calling evalModel(...)) it calls evalModel on these solver objects, which solve the corresponding sub-problems. The sequence of creating a solver factory and then evaluating the solver is similar to the flow in `Main_Solve.cpp`.

Ideally communication between different Albany::Applications is performed through the ModelEvaluator’s inputs (parameters) and outputs (responses). Parameters and responses can be “scalar”, meaning a double-precision value that does not correspond to any particular mesh nodes, or “distributed”, meaning the response is akin to a field which consists of a double-precision value at each mesh node. At the time of this writing, the Albany framework has support for scalar parameters and responses, and distributed responses, but not distributed parameters. As such, within QCAD::Solver inter-Application communication of distributed quantities is done in a more roundabout way (communication of scalars is done using the Model Evaluator interface). Distributed quantities are saved in as states (within Albany::StateManager). Hooks have been added into Albany::StateManager that allow QCAD::Solver to import States from an external container into the StateManager’s states prior to running the evalModel(...) corresponding to the associated solver and Albany::Application. Since the states hold field data at mesh quadrature points, this method of saving and importing states does not work for nodal quantities. Nodal quantities can be imported using “Auxiliary vectors”, which are also managed by the StateManager but are distinguished from states in that they hold nodal quantities and are not partitioned by workset. Eigenvectors and eigenvalues are communicated using the Eigendata structure, again managed by Albany::StateManager. It should be noted that the eigenvectors stored in the Eigendata structure are in the *overlapped* distribution, that is, nodes along processor boundaries are duplicated on each bordering processor so that for every cell owned by a processor, all the nodes for that cell are also owned by the processor.

As part of running an algorithm that involves multiple coupled Albany::Application ob-

jects, the QCAD::Solver class performs a substantial amount of input file processing which consists of a lot of monkeying with Teuchos::ParameterList objects to take a single user-given input file and create input parameter lists for each separate Albany::Application.

A final feature of QCAD::Solver is its ability to act as a switchboard for parameters and responses. The user can specify how the parameters input to QCAD::Solver (as a ModelEvaluator) get processed into parameters for the contained Albany::Application objects, and likewise how the responses coming from the the contained Application objects get processed into responses of the QCAD::Solver object. Thus, there is substantial logic to support this parameter and response manipulation. The ability to perform this manipulation is valuable even on its own, and as such the QCAD::Solver can be used to solve standalone Poisson or Schrodinger simulations. For example, it may be desirable to tie two of the Dirichlet boundary condition parameters of a Poisson problem into a single parameter so that Dakota can vary the two simultaneously.

From a broad perspective, QCAD::Solver unifies QCAD, as all types of QCAD simulations can be run using a QCAD::Solver object to evaluate one or more contained Albany::Application (or AlbanyCI) objects as needed. The QCAD::Solver input file format can be used to run any QCAD simulation, and in a very flexible way with regard to what parameters are exposed and responses are returned.

Integrated P-S Solver: QCAD::CoupledPoissonSchrodinger

In addition to the typical iteration between Poisson and Schrodinger steps to solve a coupled Poisson-Schrodinger system, QCAD is able to solve the Poisson and Schrodinger equations simultaneously in an “integrated mode”. In this mode, the Poisson equation (including quantum source terms), N copies of the Schrodinger equation $\mathcal{H}\Psi_i = E_i\Psi_i$, and N normalization equations $\int |\Psi_i|^2 dV = 1$ (for $i = 1 \dots N$) are taken together as a system of equations and solved using standard residual minimization techniques with finite elements. This method of solving a coupled Poisson-Schrodinger problem is not meant to replace the iterative method in which the Schrodinger step uses an actual eigensolver, but to complement it by taking over after the iterative method is near convergence. It is expected that the integrated mode, if initialized with a near-converged solution, will arrive at a completely converged solution more quickly than the iterative algorithm alone would have.

Integrated-mode Poisson-Schrodinger simulations are carried out by the QCAD::CoupledPoissonSchrodinger class, which like QCAD::Solver is a ModelEvaluator. A QCAD::CoupledPoissonSchrodinger object holds an Albany::Application object for a Poisson and a Schrodinger problem, but unlike QCAD::Solver it creates these Application objects directly (not via Albany::SolverFactory). When a QCAD::CoupledPoissonSchrodinger object is evaluated, it computes the requested quantities (e.g. residual, jacobian, etc) by dividing up the quantities into pieces corresponding to the Poisson, Schrodinger, and normalization equations and uses the contained Application objects to fill the appropriate pieces. Whereas the QCAD::Solver class more analogous to a Solver (like NOX), the

QCAD::CoupledPoissonSchrodinger class is more like an Albany::ModelEvaluator that gets called by a solver to compute certain quantities.

The Epetra_Map (specifies how to divide an Epetra_Vector among multiple processors) for the QCAD::CoupledPoissonSchrodinger ModelEvaluator consists of $N+1$ concatenated copies of the non-overlapped discretization map, i.e. how the mesh nodes are divided among the processors, plus an additional N double-precision quantities that are divided as evenly as possible among the processors. Creating this “combined map” and logically separating a vector with the combined map into Poisson, Schrodinger, and normalization parts is a central part of what QCAD::CoupledPoissonSchrodinger does. It then uses its member Albany::Applications to fill the relevant parts of a vector (e.g. for the residual) or matrix (e.g. for the jacobian), as well as hand-coded analytic derivatives for parts of the jacobian matrix. Supporting classes QCAD::CoupledPSJacobian and QCAD::CoupledPSPreconditioner encapsulate the code for building the Jacobian matrix and preconditioner, both as Epetra_Operators. Computing the Jacobian as an Epetra_Operator as opposed to a Crs_Matrix, and computing all quantities piece-wise allows QCAD to take advantage of the repetitive structure of the problem and save on computation time and space.

A final support class is QCAD::CoupledPSObserver, which takes care of outputting the combined solution vector as a Poisson potential and N eigenvectors to exodus nicely.

Known limitations / possible extensions

- **No magnetic field support.** Since Albany does not support complex numbers, QCAD contains no support for magnetic fields. This should be a straightforward addition after the migration of Albany from Epetra to Tpetra (templated).

Chapter 7

QCAD Performance Studies

Introduction

Since QCAD's inception, the QCAD computational suite has been adopted by analysts to help with the design of actual quantum devices to be constructed in a lab. The fact that these analysts require timely and accurate predictions to aid their designs provides motivation to study and improve the code's performance, robustness and scalability, as well as to validate the solutions computed within the code.

The present section summarizes the results of some numerical studies aimed to address the following questions:

1. Do the solutions computed within the QCAD computational suite converge with successive mesh (h -) refinement?
2. Are higher-order finite elements viable for this application? That is, do higher-order elements deliver a more accurate solution at a lower computational cost than their lower-order counterparts?
3. Which choice of preconditioner within the Trilinos **Ifpack** (ILU) [?] and **ML** (algebraic multi-grid) [?] packages minimizes the total solve time of the linear systems ($\mathbf{Ax} = \mathbf{b}$) that arise from the discretization of the governing equations?
4. How well does the QCAD code scale with respect to problem size?

The discussion herein is an extension of earlier analysis summarized in [?], and may assist a user of the QCAD computational suite in selecting the best settings for a given QCAD run.

Devices Considered

In the present document, attention is restricted to two device designs:

- The Ottawa Flat 270 device.

- The Mosdot3D device device.

The former design is a modification of the original Ottawa device design [?]. It is a rather complex geometry with intricate features (Figure 7.1(a)), and can hence be difficult to mesh. This design is considered due to its simplicity: the relevant geometry is simply a three-dimensional (3D) box (Figure 7.1(b)). Because of its simplicity, the Mosdot3D device can be meshed with a variety of elements (tetrahedra, hexahedra), and quality meshes are guaranteed.

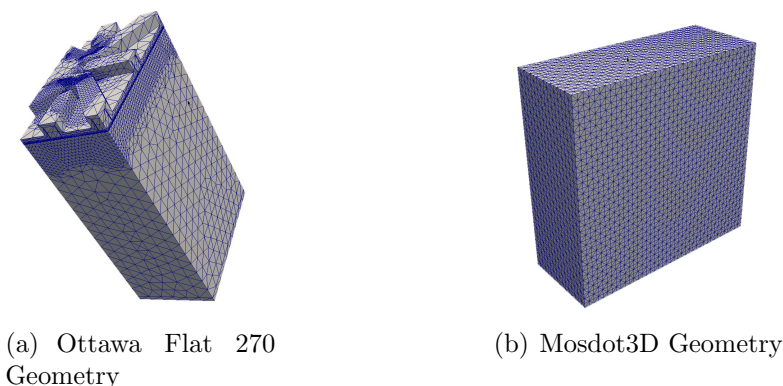


Figure 7.1. Device Geometries Considered

For both problems, the governing partial differential equation (PDE) is the non-linear Poisson equation, which describes the large population of atoms/molecules of which a device is composed (for a discussion of the Poisson physics, the reader is referred to Chapter 3 of this report). The Schrodinger and coupled Poisson-Schrodinger physics were not considered in the performance studies undertaken during the time of the QCAD project.

Meshes Considered (and Meshing Methodology)

The QCAD devices of interest (listed above) are meshed using the CUBIT meshing tool [?]. In general, four-node tetrahedral finite elements (referred to as **TETRA4** elements) are used (Figure 7.3(a)). Although CUBIT has the capability to create hexahedral meshes (as the name “CUBIT” suggests), “real” device geometries, such as the Ottawa Flat 270 device, are in general too complex to mesh using hexahedra. Simpler devices such as the Mosdot3D device can be meshed using hexahedral elements.

For our convergence studies, finer meshes were generated through a successive mesh refinement of an original “coarse” mesh, consisting of ≈ 1 million four-node tetrahedral elements for the Ottawa Flat 270 device, and 3200 four-node tetrahedral elements for the Mosdot3D device. This successive mesh refinement was achieved in CUBIT using the command:

```
refine volume all numsplit N
```

where N is the level of successive mesh refinement (e.g., if $N = 1$ each element is refined once in each direction, if $N = 2$ each element is refined twice in each direction, etc; Figure 7.2). From basic finite element theory, four-node tetrahedral finite elements are expected to converge at a rate of two in the continuous L^2 norm [?].

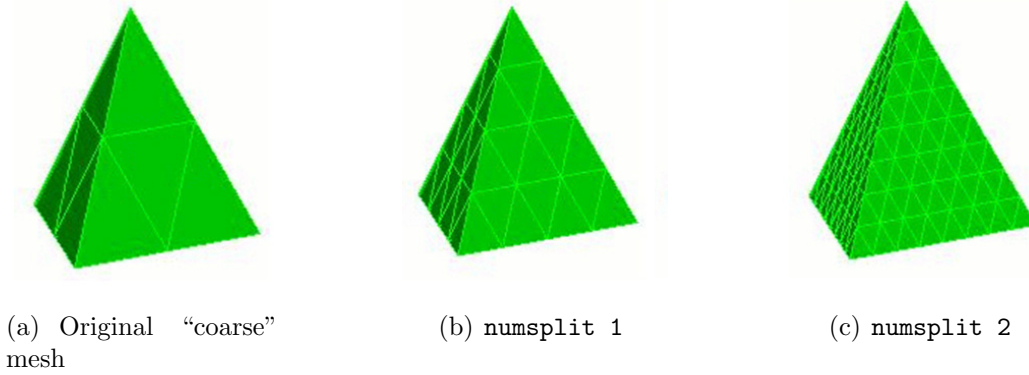


Figure 7.2. Mesh refinement

In addition to performing mesh refinement, the CUBIT meshing tool can also be used to generate higher-order finite element meshes, e.g. second-order ten-node tetrahedral finite element meshes (Figure 7.3(b)). These elements can be created using the command

```
block all element type TETRA10
```

and have a theoretical convergence rate of three in the continuous L^2 norm [?]. They are referred to herein as **TETRA10** elements.

As stated earlier, simpler devices such as the Mosdot3D device can be meshed using hexahedral elements. Hence, in addition to meshing the Mosdot3D device geometry with **TETRA4** and **TETRA10** elements, this device geometry is meshed also using first-order eight-node hexahedral elements (referred to as **HEX8** elements) and second-order twenty-seven node hexahedral elements (referred to as **HEX27** elements). These hexahedral meshes are successively refined in a similar manner to their tetrahedral counterparts. The convergence rates of the **HEX8** and **HEX27** finite elements are two and three respectively in the continuous L^2 norm [?]. Although hexahedral elements have the same convergence rates as their tetrahedral counterparts, they are expected in general to produce a more accurate solution for a fixed mesh resolution.

The consideration of higher-order elements, namely the **TETRA10** and **HEX27** elements, is motivated by the recent paper [?], where a huge benefit was seen in using higher-order finite

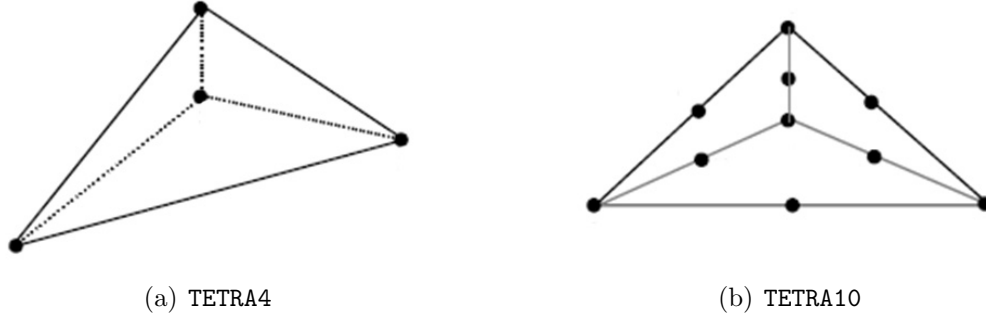


Figure 7.3. Illustration of first-order (TETRA4) and second-order (TETRA10) tetrahedral finite elements

elements for Schrodinger-Poisson problems similar to those implemented within the QCAD computational suite.

Architectures Considered

All runs were performed on the 160-TFlop Red Sky cluster at Sandia.

Preconditioners Considered

For the preconditioner performance studies, the relative performance of several different preconditioners available through the Trilinos **Ifpack** and **ML** packages is evaluated. Twelve basic preconditioner types are considered: nine **Ifpack** preconditioners and three **ML** preconditioners (Table 7.1). The **Ifpack** preconditioners are effectively ILU preconditioners, and differ in the **overlap** and **level-of-fill** options. The **ML** preconditioners are algebraic multi-grid preconditioners based on three default preconditioner types available in the **ML** package: **SA** (classical Smoothed Aggregation), **DD** (classical smoothed aggregation based on two-level Domain Decomposition), and **DD-ML** (three-level algebraic Domain Decomposition). For a detailed discussion of these **Ifpack** and **ML** options, the reader is referred to the **Ifpack** and **ML** user guides, [?] and [?] respectively.

Table 7.1. Summary of preconditioners evaluated

Preconditioner #	Type	Parameters
1	ifpack	overlap = 0, level-of-fill = 0
2		overlap = 1, level-of-fill = 0
3		overlap = 2, level-of-fill = 0
4		overlap = 0, level-of-fill = 1
5		overlap = 1, level-of-fill = 1
6		overlap = 2, level-of-fill = 1
7		overlap = 0, level-of-fill = 2
8		overlap = 1, level-of-fill = 2
9		overlap = 2, level-of-fill = 2
10	ML	default type = SA
11		default type = DD
12		default type = DD-ML

By perusing the ML users' guide [?], the reader may observe that the ML preconditioner package has a number of options and parameters that may be specified through an input file (i.e., over-written from the default settings). In an effort to optimize the performance of the ML preconditioners, it is worthwhile to explore several of these options. To this effect, three variants of the ML preconditioners introduced in Table 7.1, referred to as A, B and C, are considered. The parameter lists for these preconditioner options are summarized in Table 7.2 for the specific case of an SA default preconditioner. The C variant preconditioner employs the matrix repartitioning option available through the Trilinos **Zoltan** package. Essentially, repartitioning uses information about the mesh coordinates to perform dynamic load-balancing of coarse-level matrices in the multigrid preconditioner. With repartitioning, message passing latency on the coarse level can be improved, and the well-known problem of the coarsening rate dropping as the number of unknowns per processor becomes small can be avoided. Providing the user with the option to select an ML preconditioner with **Zoltan** repartitioning required some non-trivial new development within Albany. Functions that identify and communicate the (x, y, z) coordinates of a problem's underlying mesh have been added to existing Albany classes.

Table 7.2. Summary of ML settings evaluated (for example of default values: SA)

ML settings A

```
<ParameterList name="ML">
  <Parameter name="Base Method Defaults" type="string" value="none"/>
    <ParameterList name="ML Settings">
      <Parameter name="default values" type="string" value="SA"/>
      <Parameter name="smoother: type" type="string" value="Chebyshev"/>
      <Parameter name="smoother: pre or post" type="string" value="both"/>
      <Parameter name="coarse: type" type="string" value="Amesos-KLU"/>
    </ParameterList>
  </ParameterList>
```

ML settings B

```
<ParameterList name="ML">
  <Parameter name="Base Method Defaults" type="string" value="none"/>
    <ParameterList name="ML Settings">
      <Parameter name="default values" type="string" value="SA"/>
      <Parameter name="smoother: type" type="string" value="Chebyshev"/>
      <Parameter name="smoother: pre or post" type="string" value="both"/>
      <Parameter name="coarse: type" type="string" value="Amesos-KLU"/>
      <Parameter name="coarse: max size" type="int" value="512"/>
      <Parameter name="aggregation: type" type="string" value="Uncoupled-MIS"/>
    </ParameterList>
  </ParameterList>
```

ML settings C

```
<ParameterList name="ML">
  <Parameter name="Base Method Defaults" type="string" value="none"/>
    <ParameterList name="ML Settings">
      <Parameter name="default values" type="string" value="SA"/>
      <Parameter name="smoother: type" type="string" value="Chebyshev"/>
      <Parameter name="smoother: pre or post" type="string" value="both"/>
      <Parameter name="coarse: type" type="string" value="Amesos-KLU"/>
      <Parameter name="coarse: max size" type="int" value="512"/>
      <Parameter name="repartition: enable" type="int" value="1"/>
      <Parameter name="repartition: partitioner" type="string" value="Zoltan"/>
      <Parameter name="repartition: Zoltan dimensions" type="int" value="3"/>
      <Parameter name="repartition: max min ratio" type="double" value="1.3"/>
      <Parameter name="repartition: min per proc" type="int" value="1000"/>
    </ParameterList>
  </ParameterList>
```

Mesh Convergence Study

We first perform a mesh convergence study on the two device designs considered: the Ottawa Flat 270 device and the Mosdot3D device. Two metrics are used to study mesh convergence: the mean value of the solution, and a field integral of the solution (the integral of the solution over the domain). The former quantity is effectively an $L^1(\Omega)$ norm, and the latter is effectively an $L^2(\Omega)$ norm. From basic finite element theory, the expected convergence rates in these norms for four-node tetrahedral and eight-node hexahedral finite elements are one and two respectively; for ten-node tetrahedral and twenty-seven-node hexahedral finite elements, they are two and three respectively [?]. Since an analytical form of the exact solution to this problem is not available, relative errors are measured with respect to a converged reference solution, computed numerically on a fine mesh. Given u_{ref} , the computed solution on the reference mesh (in this case, the mean value of the solution or the field integral), the relative error is computed as:

$$\epsilon_{ref} = \frac{|u_{ref} - u_N|}{|u_{ref}|}, \quad (7.1)$$

where u_N is the solution computed on a mesh of N elements.

Ottawa Flat 270 Geometry

For the Ottawa Flat 270 convergence study, three four-node tetrahedral and two ten-node tetrahedral meshes of the geometry are created. These mesh resolutions are summarized in Table 7.3.

Table 7.3. Ottawa Flat 270 Meshes Considered

	Refinement	# Noes in Mesh	# Elements in Mesh	# Procs for Run	# Nodes on Red Sky for Run
TETRA4	numsplit 0	160,669	935,424	16	2
	numsplit 1	1,262,489	7,483,392	128	16
	numsplit 2	10,031,281	59,867,136	1024	128
TETRA10	numsplit 0	1,262,489	935,424	128	32
	numsplit 1	10,031,281	7,483,392	1024	256

The number of processors for each run was computed such that each processor had approximately 10,000 nodes. Note that the TETRA4 + numsplit N+1 mesh has the same number

of nodes as the `TETRA10 + numsplit N` mesh, for $N = 0, 1, \dots$. For all runs, `Num Blocks` and `Maximum Iterations` for the Linear solver (Belos) were both set to 500. Hexahedral meshes were not considered for this example, as it is not possible to mesh the device of interest with hexahedral elements.

First, we check the quality of the meshes considered, as mesh quality can affect solution convergence. Figure 7.4 depicts the quality of the elements in the `TETRA4 + numsplit 0` and `TETRA4 + numsplit 1` meshes in the CUBIT “scale” metric. If this value is < 0.2 , the element is considered “bad” [?]. It can be seen from this figure that the meshes have a number of elements of poor quality. The mesh quality does not seem to improve with mesh refinement or mesh smoothing.

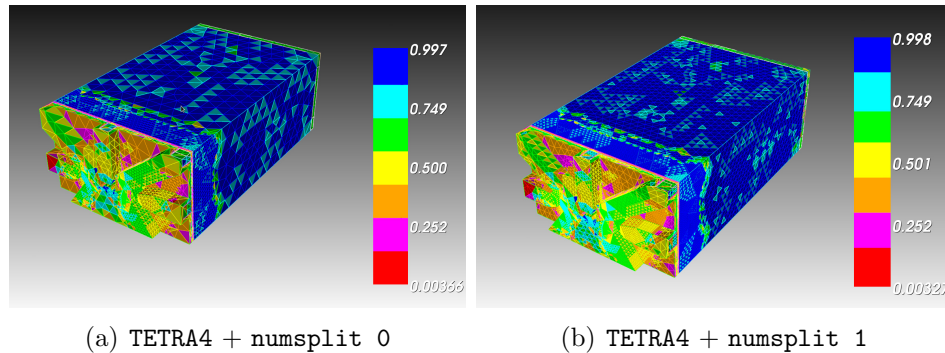


Figure 7.4. Mesh quality (in “scale” metric) for Ottawa Flat 270 geometry

Despite the presence of “bad” elements in the meshes considered, we undertake a mesh convergence study for this problem. Table 7.4 gives the values of two quantities of interest (responses) used to evaluate mesh convergence: the field integral of the left dot electrons and the field integral of the right dot electrons. The `TETRA10 + numsplit 1` result is missing from the table, as the nonlinear solver failed to converge for this mesh. Once the quantities of interest (the field integrals) are computed for each mesh, the relative error in each solution is computed using the formula (7.1), with the `TETRA4 + numsplit 2` solution taken to be the reference solution u_{ref} . The relative error as a function of the mesh size is plotted on a log-log scale in Figure 7.5 for the `TETRA4` finite elements. The convergence rates are reasonably close to the theoretical convergence rate of two for the finite elements considered. The convergence rates for the `TETRA10` finite elements is not reported, as there are not enough data points for the `TETRA10` meshes to compute these rates.

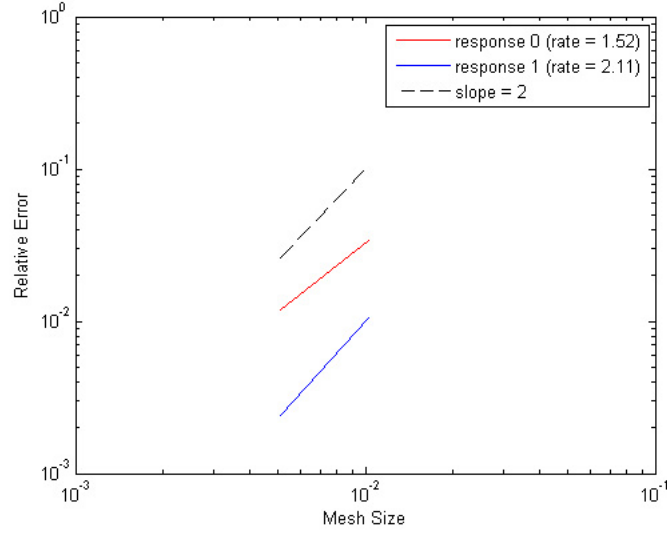


Figure 7.5. Convergence plot for TETRA4 finite elements for Ottawa Flat 270 geometry

Table 7.4. Mesh Convergence of Field Integral Quantities for Ottawa Flat 270 Geometry

	Refinement	Field Integral (Left Dot Electrons)	Field Integral (Right Dot Electrons)
TETRA4	numsplit 0	19.0776943	17.30564924
	numsplit 1	19.9838443	17.5311494
	numsplit 2	19.75007	17.488771
TETRA10	numsplit 0	20.205781	18.1760936
	numsplit 1	—	—

Mosdot3D Geometry

We now undertake a mes mesh convergence study similar to the one performed above for a simpler geometry, the Mosdot3D geometry. The reason for this study is two-fold. Since the

geometry is a simple box, it can be meshed with tetrahedral as well as hexahedral elements (unlike the Ottawa Flat 270 geometry, which is too complex to mesh with hexahedral elements). Moreover, since the geometry is simple, the mesh quality is expected to be perfect. Hence, any effects of bad mesh quality on convergence are not present in this problem.

The meshes considered for the Mosdot3D convergence study are summarized in Table 7.5.

Table 7.5. Mosdot3D Meshes Considered

	Refinement	# Nodes in Mesh	# Elements in Mesh	# Procs for Run	# Nodes on Red Sky for Run
TETRA4	numsplit 0	3969	19,200	1	1
	numsplit 1	28,577	153,600	8	1
	numsplit 2	216,513	1,228,800	64	8
	numsplit 3	1,684,865	9,830,400	512	64
HEX8	numsplit 0	3969	3200	1	1
	numsplit 1	28,577	25,600	8	1
	numsplit 2	216,513	204,800	64	8
	numsplit 3	1,684,865	1,638,400	512	64
TETRA10	numsplit 0	28,577	19,200	8	1
	numsplit 1	216,513	153,600	64	8
	numsplit 2	1,684,865	1,228,800	512	64
	numsplit 3	13,292,289	9,830,400	4096	512
HEX27	numsplit 0	28,577	25,600	8	1
	numsplit 1	216,513	25,600	64	8
	numsplit 2	1,684,865	204,800	512	64
	numsplit 3	13,292,289	1,638,400	4096	512

The number of processors for each run was computed such that each processor had approximately 3200 nodes. Note that the TETRA4 + numsplit N meshes have the same number of nodes as the TETRA10 + numsplit N+1 meshes, the HEX8 + numsplit N meshes, and the HEX27 + numsplit N+1 meshes, for $N = 0, 1, \dots$. For all runs, Num Blocks and

Maximum Iterations for the Linear solver (Belos) were both set to 200.

For the Mosdot3D problem, convergence in two metrics is studied: the solution average and the field integral of the electron density in the quantum region. Table 7.6 gives the values of these quantities for each of the meshes considered. Unlike for the Ottawa Flat 270 problem, all runs (on all meshes) converged. Once the quantities of interest (the solution average and field integral) are computed for each mesh, the relative error in each solution is computed using the formula (7.1), with the `HEX 8 + numsplit 3` solution taken to be the reference solution u_{ref} . The relative error as a function of the mesh size is plotted on a log-log scale in Figure 7.6 for all finite elements considered. In this convergence rate plot, the mesh size is approximated as $(\text{number of elements})^{-1/3}$.

Table 7.6. Mesh Convergence of Solution Average and Field Integral Quantities for Mosdot3D Geometry

	Refinement	Solution Average	Field Integral (Electron Density in Quantum Region)
TETRA4	<code>numsplit 0</code>	0.28286192	−5411.6888096
	<code>numsplit 1</code>	0.24704725	−6019.4785176
	<code>numsplit 2</code>	0.226695582	−6226.973865
	<code>numsplit 3</code>	0.215715908	−6307.89499814
HEX8	<code>numsplit 0</code>	0.290733103	−5997.995616
	<code>numsplit 1</code>	0.250036118	−6225.725794
	<code>numsplit 2</code>	0.227867438	−6306.879351
	<code>numsplit 3</code>	0.2161816919	−6339.327812
TETRA10	<code>numsplit 0</code>	0.2500988	−6182.004445
	<code>numsplit 1</code>	0.2281012094	−6293.594787
	<code>numsplit 2</code>	0.2162818224	−6337.624098
	<code>numsplit 3</code>	0.210178835316	−6353.88902243
HEX27	<code>numsplit 0</code>	0.2518402396	−6331.245184
	<code>numsplit 1</code>	0.228988126	−6355.0088013
	<code>numsplit 2</code>	0.2166345454	−6362.7661636
	<code>numsplit 3</code>	0.2098942876	−6364.0844277

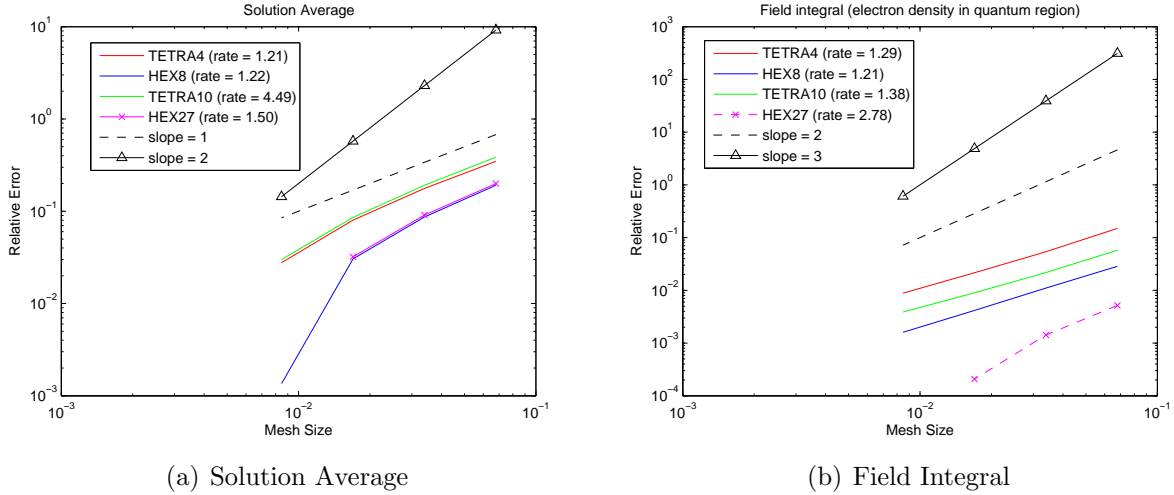


Figure 7.6. Convergence plot for various finite elements for Mosdot3D geometry

The reader's attention is drawn to the field integral convergence plot (Figure 7.6(b)), as the field integral is an integrated quantity and therefore close to an L^2 norm, the common norm in which convergence is studied. The reader may observe by studying Figure 7.6(b) that the HEX elements produce a more accurate solution than their TETRA counterparts of comparable convergence order. The TETRA4 and TETRA10 elements are converging at a slower rate than their theoretical rates (two and three in the L^2 norm, respectively). The HEX8 element is also not achieving its theoretical convergence rate of two. All three elements, the TETRA4, TETRA10 and HEX8 are converging at similar rates. The HEX27 element comes closest to its theoretical convergence rate of three.

Preconditioner Performance Studies

We now perform some preconditioner performance studies on the Ottawa Flat 270 and Mosdot3D problems. The results of these studies are aimed to aid the QCAD user in selecting the best preconditioner to use for a given QCAD run.

Ottawa Flat 270 Geometry

Figures 7.7–7.9 depict the Belos total linear solve times and total preconditioner creation times for the nine Ifpack preconditioners and the three ML preconditioners summarized in Table 7.1.

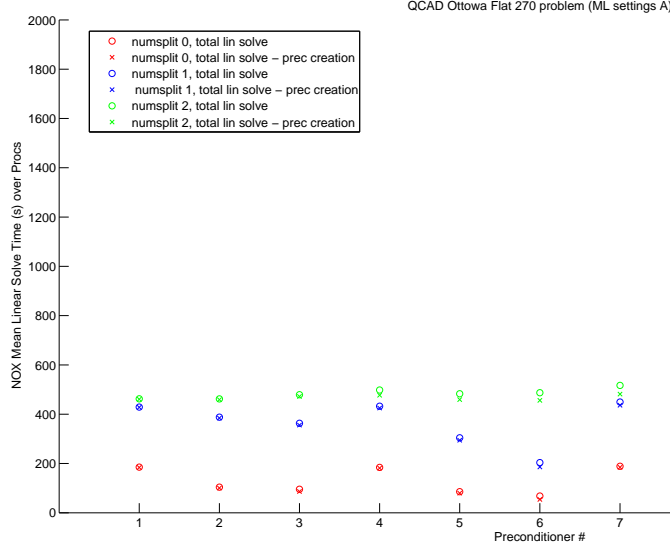


Figure 7.7. Ifpack vs. ML preconditioners with ML settings A

The ML preconditioners in Figure 7.7 have the ML settings A; the ML preconditioners in Figure 7.8 have the ML settings B; the ML preconditioners in Figure 7.9 have the ML settings C. With settings that are effectively the default ML settings (settings A), the ML preconditioners are outperformed by the Ifpack preconditioners by a large margin on the finest mesh considered (Table 7.7). The performance of the ML preconditioners improves when the `aggregation: type` is changed to `Uncoupled-MIS` (settings B); however the ML preconditioners still do not outperform the Ifpack preconditioners on the finest mesh (Figure 7.8). Inspection of the verbose output from the ML package suggested that the situation may be improved by introducing `Zoltan` repartitioning based on nodal coordinate, and the Albany code base was modified to allow this option, as discussed above. The reader may observe an extraordinary speedup in the total linear solve and preconditioner creation times for the ML preconditioners with repartitioning (settings C) (Figure 7.9).

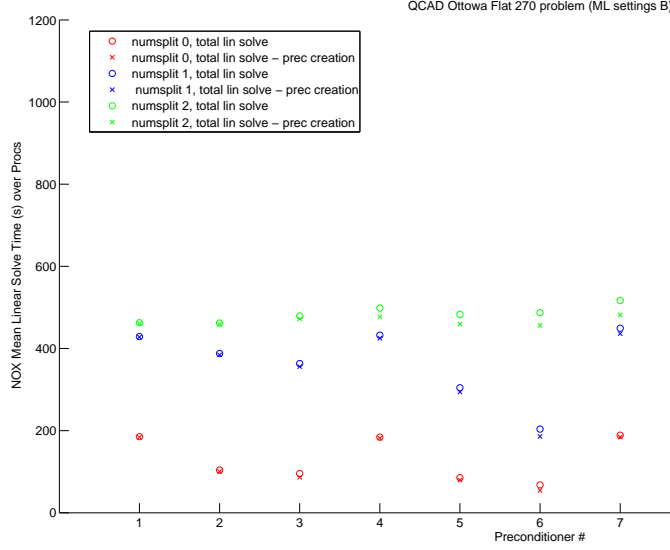


Figure 7.8. Ifpack vs. ML preconditioners with ML settings B

With settings C, the ML preconditioners achieve a factor speedup of more than two relative to the **Ifpack** preconditioners for *all* mesh resolutions considered. More specifically, for a problem discretized by ≈ 10 million tetrahedral elements, run on 128 processors on the Red Sky cluster:

- The linear solves were 2.4–5.45 times faster with an ML preconditioner plus **Zoltan** repartitioning (settings C) compared to an **Ifpack** preconditioner.
- The linear solves with an ML preconditioner plus **Zoltan** repartitioning (settings C) were ≈ 2 times faster than with a “black box” ML preconditioner (settings A).

For a problem discretized by ≈ 60 million tetrahedral elements, run on 1024 processors on the Red Sky cluster:

- The linear solves were 2.4–2.79 times faster with an ML preconditioner plus **Zoltan** repartitioning (settings C) compared to an **Ifpack** preconditioner.
- The linear solves with an ML preconditioner plus **Zoltan** repartitioning (settings C) were 9.5 times faster than with a “black box” ML preconditioner (settings A)

The ML preconditioner option with **Zoltan** repartitioning (settings C) is therefore recommended for all problem sizes.

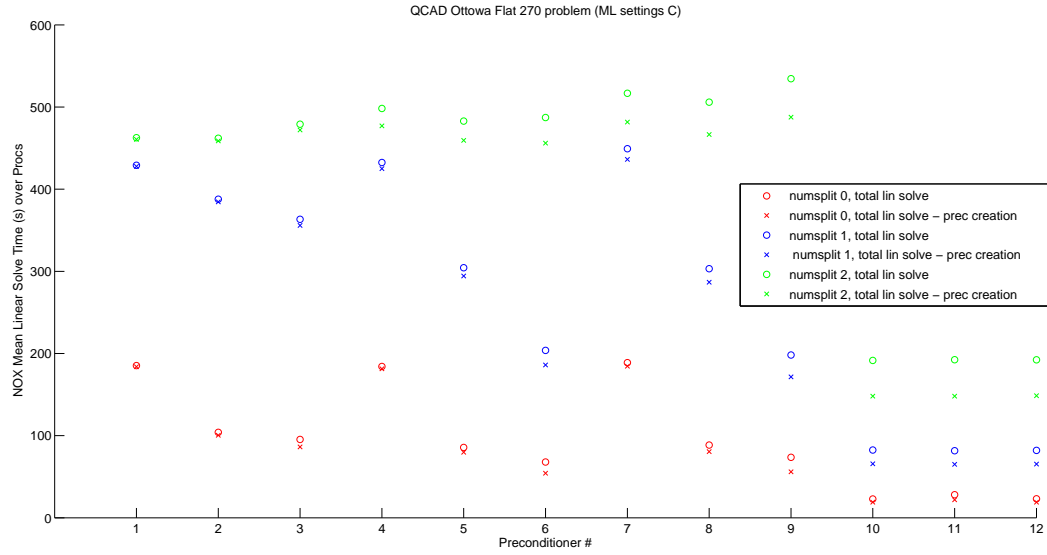


Figure 7.9. Ifpack vs. ML preconditioners with ML settings C

Mosdot3D Geometry

A preconditioner performance study is now performed on the Mosdot3D geometry. The objective of this study is to evaluate numerically the performance of various preconditioners when higher-order finite elements (TETRA10 and HEX8 elements) are employed. Only one mesh resolution is considered, the mesh resolution for which the discretization has 28,577 nodes. For brevity, the Ifpack preconditioners #1-4 and #7 are omitted from this study. The ML preconditioners #11-12 are also omitted from this study, as the performance of these preconditioners was similar to the performance of the ML preconditioner #10. Similarly, ML settings A and B are omitted from the study.

Tables 7.7–7.10 show the total linear solve, the total preconditioner creation and total Albany times for each of the runs, in addition to the number of unconverged linear solves and the number of nonlinear iterations required.

Table 7.7. Mosdot3D preconditioner performance study:
TETRA4 elements

Preconditioner #	5	6	8	9	10
Total Linear Solve Time	8.280	2.120	2.894	16.73	1.999
Total Preconditioner Creation Time	7.53	0.9731	2.247	7.36	1.447
# Unconverged Linear Solves	0	0	0	0	0
Total Albany Time	23.35	21.44	16.89	55.61	11.0
# Nonlinear Iterations	7	7	7	7	7

Table 7.8. Mosdot3D preconditioner performance study:
HEX8 elements

Preconditioner #	5	6	8	9	10
Total Linear Solve Time	1.699	3.320	1.956	3.619	0.7974
Total Preconditioner Creation Time	1.035	2.574	1.38	3.026	0.3861
# Unconverged Linear Solves	0	0	0	0	0
Total Albany Time	75.383	2.868	5.732	7.229	4.366
# Nonlinear Iterations	7	7	7	7	7

Table 7.9. Mosdot3D preconditioner performance study:
TETRA10 elements

Preconditioner #	5	6	8	9	10
Total Linear Solve Time	12.21	50.23	12.88	61.52	2.458
Total Preconditioner Creation Time	8.672	46.99	9.783	57.13	1.414
# Unconverged Linear Solves 0	0	0	0	0	
Total Albany Time	18.93	55.16	30.25	66.66	8.354
# Nonlinear Iterations	7	7	7	7	7

Table 7.10. Mosdot3D preconditioner performance study:
HEX27 elements

Preconditioner #	5	6	8	9	10
Total Linear Solve Time	30.31	35.23	35.79	208.2	22.20
Total Preconditioner Creation Time	21.99	23.89	27.39	194.4	0.715
# Unconverged Linear Solves	1	1	1	1	1
Total Albany Time	33.38	43.9	40.0	211.0	27.57
# Nonlinear Iterations	8	8	8	8	8

The **ML** preconditioner gives the shortest total Albany time for all elements considered. It is curious that this is the case for the higher-order **TETRA10** and **HEX28** elements, since the **ML** preconditioners have not been designed to work with these elements out of the box.

Scalability Studies

Finally, we provide some weak scalability results for the Ottawa Flat 270 problem, the larger of the problems considered here (and hence the more amenable to scalability studies). Figure 7.10 shows a weak scalability plot for the **TETRA4** elements with an **ML** preconditioner (Preconditioner #10 with settings C). The three data points included are the 16, 128 and 1024 processor Red Sky runs (see Table 7.3). For perfect weak scalability, all curves in the figure should be flat (i.e., have slope equal to zero). The figure suggests good weak scalability in the finite element assembly, but not the linear solve.

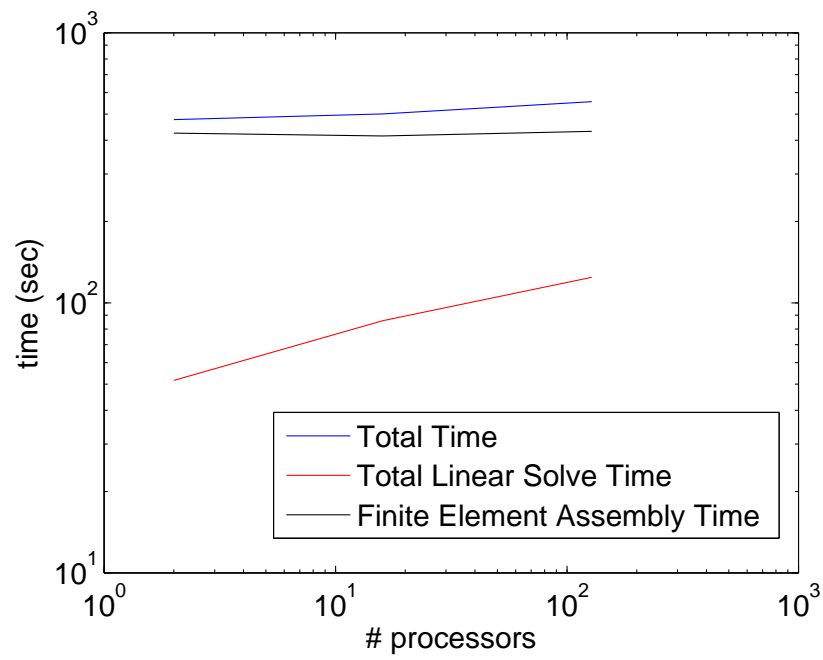


Figure 7.10. Weak scalability plot for TETRA4 elements + ML preconditioner (16, 128, 1024 processors on Red Sky)

DISTRIBUTION:

1	MS 1322	Richard P. Muller, 1425
1	MS 1415	Erik Nielsen, 1121
1	MS 9159	Raymond Tuminaro, 1121
1	MS 0899	Technical Library, 9536 (electronic copy)
1	MS 0359	D. Chavez, LDRD Office, 1911

