A Hardware-Accelerated Steady State Power-Flow Solver

Daniel Chavarría-Miranda
Applied Computer Science

David Chassin
Energy Technology Development

Pacific Northwest National Laboratory (PNNL)

Introduction

- The power-flow problem is a fundamental calculation in power systems
  - It determines the voltage magnitude and phase angle at each bus i power grid under steady state conditions
- The preferred strategy for computing a solution is to use a Newton-Raphson iterative scheme
  - Advantage: quadratic convergence
  - Disadvantages:
    - problems handling “flat-start” initial conditions (zero angle, one-per-un magnitude)
    - requires computing a Jacobian on every iteration (global computation)

Introduction (cont.)

- Another options is to use a Gauss-Seidel iterative scheme
  - Advantages:
    - no problem handling “flat-start” conditions
    - computation for each bus in the grid requires only “local” information
      - Makes it easily parallelizable
  - Disadvantage: slower convergence rate
- Gauss-Seidel iterative formulation:

$$V_k = \frac{P_k + jQ_k}{Y_{kk}\overline{V}_k} - \sum_{n=1, n\neq k}^{N} V_n \frac{Y_{kn}}{Y_{kk}}$$

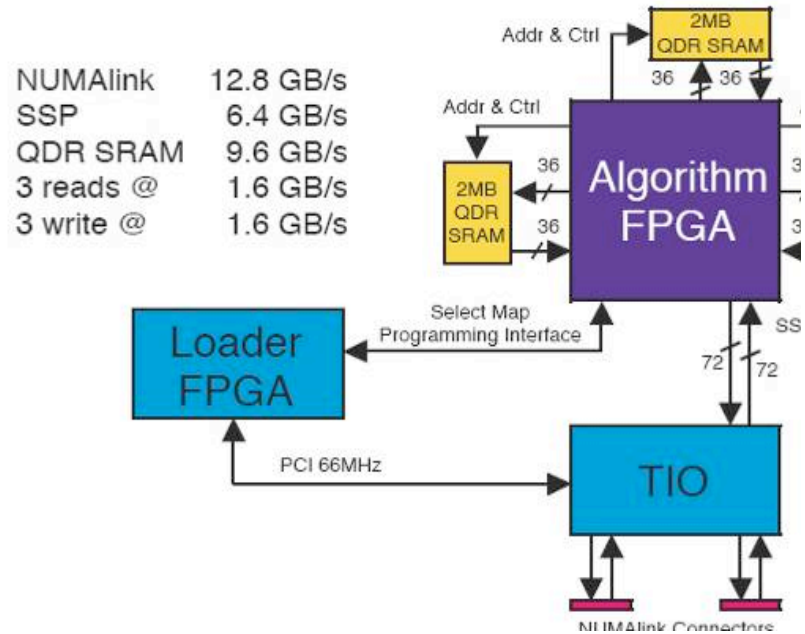
  - where V_k is the complex voltage for bus k, P_k is the real power at b k, Q_k is the reactive power at bus k and Y_{kn} is the admittance value between buses k and n.

Outline

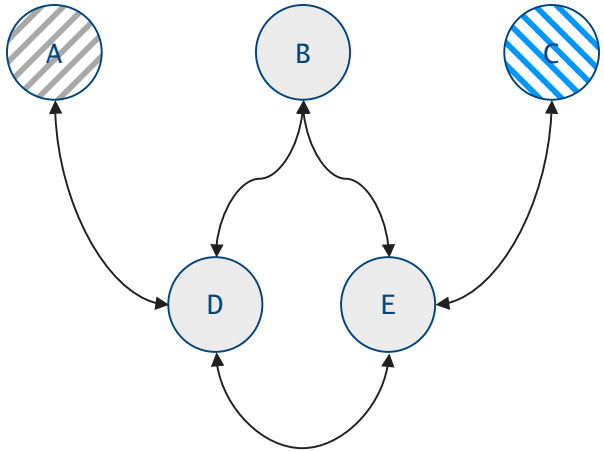
- Introduction
- Design & Implementation of a Gauss-Seidel Solver
  - Platform
  - Algorithms & Data Structures
  - Operators
  - Computational Structure
- Preliminary Results
- Conclusions & Future Work

Platform

- SGI RASC system:
  - SGI Altix 350 + Virtex II 6000 FPGA
  - FPGA board is connected to the high-speed NUMAlink fabric
  - Accessible to all the Itanium 2 General Purpose Processors (GPP) o the system, however the FPGA board cannot access system memor
  - SGI provides a software API to access the FPGA board's memory and registers
  - SGI provides prebuilt cores to control 3 2MB banks of SRAM and to interface with the system



Algorithms & Data Structures



- Prototype solves a 5-bus system with 1 swing bus (A), 1 voltage-controlled bus (C), and 3 regular (load) buses (B, D,
  - Swing bus is a reference bus with magnitude = 1.0 and phase angl 0.0
  - Voltage-controlled bus: voltage magnitude is fixed, phase angle ca vary
  - Regular (load) buses: voltage magnitude and angle can vary

Algorithms & Data Structures (cont.)

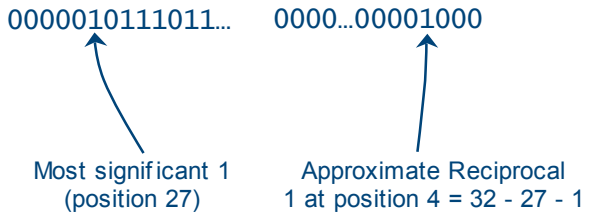
- Compile-time inputs to the prototype are:
  - Real and reactive power
  - Reactive power limits
  - Admittance matrix
  - Initial complex voltages for each bus
- In general, the admittance matrix (*Y*) has a sparse structure that corresponds to the grid topology
  - However, for simplifying the implementation we have used a dense representation (at the cost of some extra storage)

Operators

- We have implemented our protoype using Celoxica's Hand C
  - Provides a fixed-point operator library (+, -, *, /, <, >, etc.)
- All data elements are represented as signed 32-bit fixed-po quantities (16:16)
  - Complex numbers use two 32-bit quantities for their real and imaginary parts
- Built a library of complex number operators based on fixed-point primitives
- Also built our own fixed-point division operator because Handel-C's operator did not satisfy timing requirements
  - It was trying to do 32-bit fixed-point division in a single clock cycle

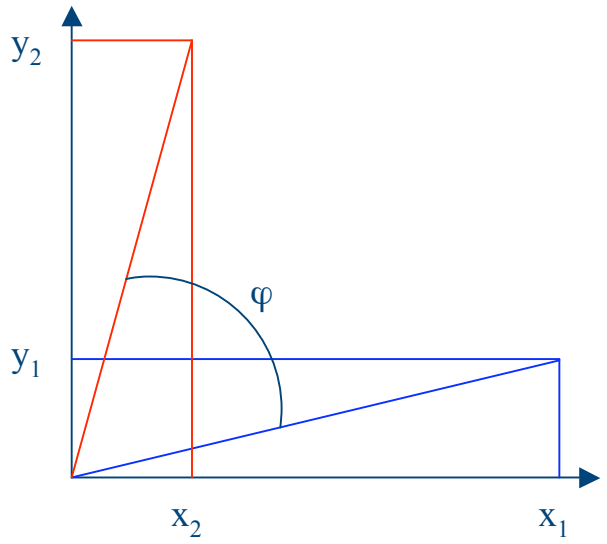
Operators (cont.)

- Division operator is based on a Newton-Raphson iterative scheme
  - First, compute the reciprocal of the divisor, then multiply by the dividend
  - Initial approximation of the reciprocal is done by finding the positior the most significant 1 in the binary representation of the divisor and "complementing" the position
  - This fraction is an upper bound for the reciprocal of the divisor



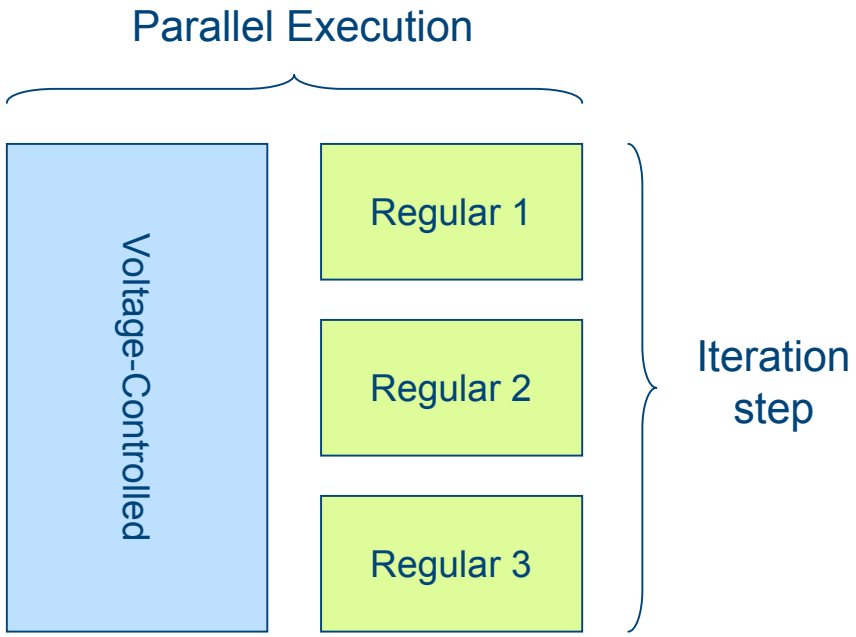
Operators (cont.)

- We also implemented a CORDIC library to compute sines and cosines
  - used in conversions between rectangular and polar complex numbe representations
- Extended the idea behind CORDIC to compute exponential as well



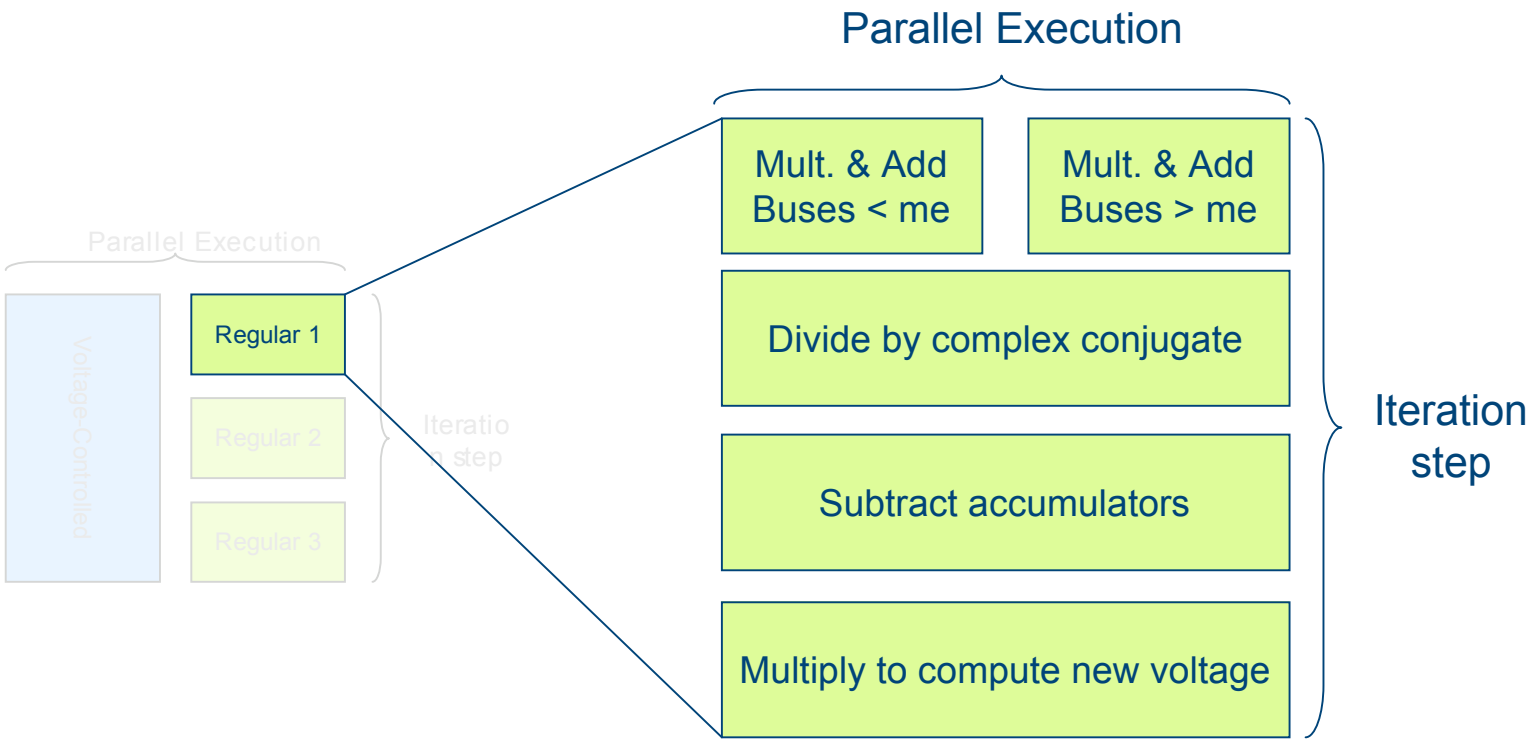
Computational Structure

- The implementation uses an iterative flow inside the FPGA
  - Initial values are loaded from precomputed ROM tables
  - In each iteration, the voltages for the regular (load) buses are computed in parallel with respect to the voltage-controlled bus



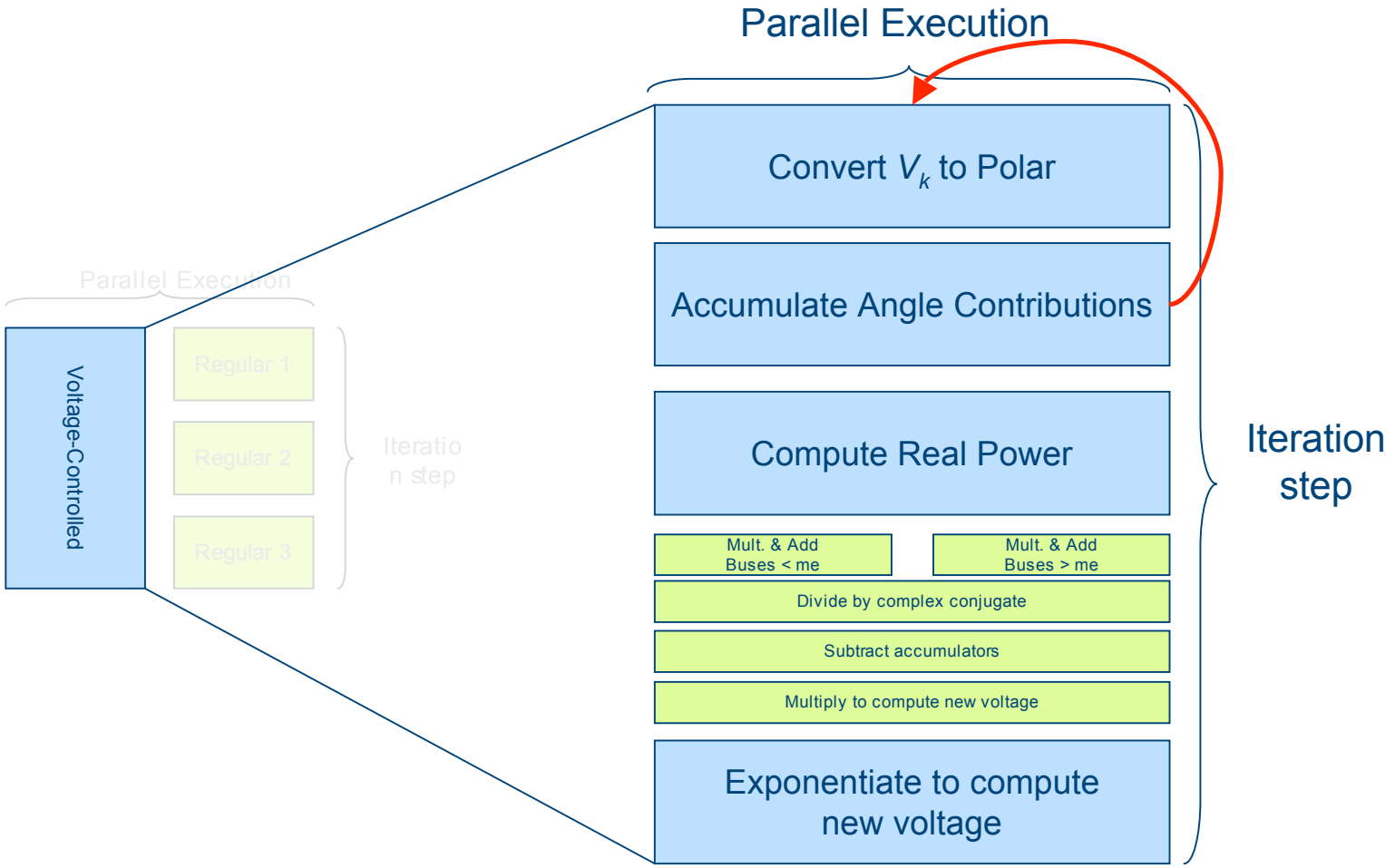
Computational Structure (cont.)

- Computation for a regular bus:



Computational Structure (cont.)

- Computation for a voltage-controlled bus:



Outline

- Introduction
- Design & Implementation of a Gauss-Seidel Solver
  - Platform
  - Algorithms & Data Structures
  - Operators
  - Computational Structure
- ⌦ Preliminary Results
- Conclusions & Future Work

Preliminary Results

Implementation	*Execution Time*	*FPGA Speedup*
Floating-point SW (simple)	3680μs	4.50
Fixed-point SW (simple)	2658μs	3.25
Fixed-point HW (@100 MHz)	818μs	1.00
Floating-point SW (optimized)	380μs	0.46

- Software versions execute on a 1.5 GHz Itanium 2 process of the Altix
- Fixed-point versions are accurate to one fractional digit

Analysis of Results

- One iteration of a regular bus takes 2.13μs (@100 MHz)
  - Fewer opportunities for optimization
- One iteration of a voltage-controlled bus takes 8.09μs
  - More opportunities for optimization
- Operators are not yet pipelined
  - They are “staged” in Handel-C, behave like a software subroutine c
- Simplifying the computation may enable executing at 200MHz (halve the execution time)

Conclusions

- Pros:
  - Prototype of a complex, non-linear solver for steady-state estimatio
    - Complex arithmetic operators & transcendental functions
  - Developed entirely in a higher-level hardware language (Handel-C)
  - Reasonable performance for the clock frequency achieved (100 MH
- Cons:
  - Performance is not yet competitive with software
  - Complex computation hinders higher degrees of parallelism and higher clock frequencies
    - Voltage-controlled path is quite complex

Future Work

- Need to scale the solver to realistic problem sizes (10,000 buses)
  - Requires handling sparse representations for the constant admittan data
  - Higher overheads in software for sparse representations might give an edge to the FPGA approach
- Need to improve performance
  - Optimize computational paths (regular & voltage-controlled buses)
  - Increase parallelism (more buses computed simultaneously)
  - Optimize & pipeline operators to overlap computation of multiple bu
    - Could help with achieving higher clock frequencies