RTCSA 2012

Performance Comparisons of Parallel Power Flow Solvers on GPU System

Chunhui Guo[1], Baochen Jiang[1], Hao Yuan[1], Zhiqiang Yang[1], Li Wang[2], Shangping Ren[2]

[1]Shandong University at Weihai, China
[2]Illinois Institute of Technology, USA

Outline

Power Flow

- Describe steady state of a power system
- Importance
  - optimize real-time control of running power systems
  - provide essential information for designing new power systems
  - provide basics for other power system analysis
- Calculation
  - involve thousands of equations
- Goal
  - increase computation speed

Parallel Computing

- Common approaches
  - multi-threading
  - parallel machines
  - distributed systems
- Disadvantages of these approaches
  - special hardware support
  - high cost
  - limited speed improvement

Parallel Computing on GPU

- GPU (Graphics Processing Unit)
  - high computing efficiency
  - low price
  - widely used in many fields
  - CUDA (Compute Unified Device Architecture)
- Current parallel power solvers on GPU
  - Newton method, Jacobi method
- What's missing
  - comparison among different parallel solvers
- Our work
  - parallelize and compare three common power flow solvers

Power Flow Model

For a power system with *n* independent buses, the power equations of bus *i* are:

$$P_i = \sum_{k=1}^{n} \left| V_i V_k Y_{ik} \right| \cos(\theta_{ik} + \delta_k + \delta_i) \qquad (1)$$

$$Q_i = -\sum_{k=1}^{n} \left| V_i V_k Y_{ik} \right| \sin(\theta_{ik} + \delta_k + \delta_i) \qquad (2)$$

- i, k :bus number
- P :real power
- Q :reactive power
- $|V|$:voltage magnitude
- δ :voltage angle
- $|Y_{ik}|$:magnitude of admittance between bus *i* and bus *k*
- θ_{ik} :angle of admittance between bus *i* and bus *k*

Power Flow Model

$$P_i = \sum_{k=1}^{n} \left| V_i V_k Y_{ik} \right| \cos(\theta_{ik} + \delta_k + \delta_i) \qquad (1)$$

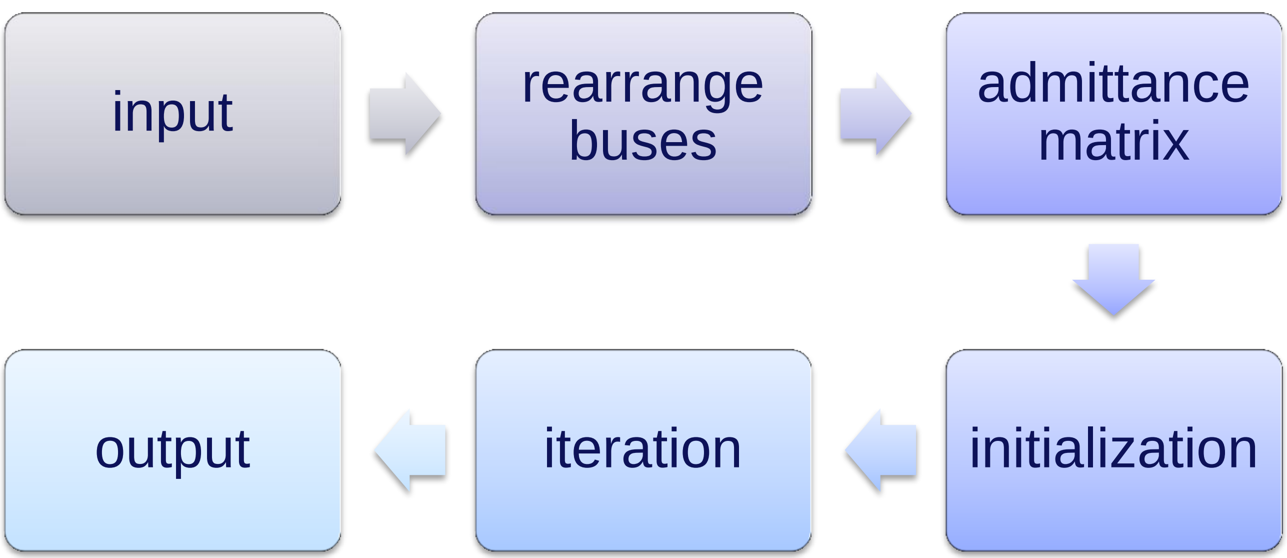
$$Q_i = -\sum_{k=1}^{n} \left| V_i V_k Y_{ik} \right| \sin(\theta_{ik} + \delta_k + \delta_i) \qquad (2)$$

- Equation (1) and (2)
  - non-linear
  - both $|Y_{ik}|$ and θ_{ik} are known
  - in P , Q , $|V|$ and δ , two variables are known
  - solvable

In order to calculate power flow, we need to solve the non-linear equations which consist of equation (1) and (2).

Power Flow Solver

- Calculation method
  - Gauss-Seidel solver
  - Newton-Raphson solver
  - P-Q decoupled solver
- Calculate steps



Power Flow Solver

- Gauss-Seidel solver
  - use the latest iteration value
- Newton-Raphson solver
  - transform non-linear equations to linear equations by Taylor series
  - coefficient matrix of linear equations (Jacobian matrix) needs to be recalculated in each iteration
  - **polar form** and rectangular form
- P-Q decoupled solver
  - simplified version of Newton-Raphson solver
  - use imaginary part of bus admittance to replace Jacobian matrix
  - coefficient matrix of linear equations remains unchanged

Speedup Analysis

- We use the multiplication number to estimate the computation cost and does not consider the communication cost between CPU and GPU.
- The speedup is sequential computation cost divided by parallel computation cost.
- For a power system with *n* buses, theoretical speedups are

Power Flow Solver	Speedup
Gauss-Seidel Solver	$0.2n$
Newton-Raphson Solver	$2n$
P-Q Decoupled Solver	$0.4n$

Parallelization

- Two problems
  - Which operations to parallelize ?
  - How to parallelize ?
- Parallelization operations
  - bus admittance matrix computation
  - iteration process
- parallelization steps

allocate GPU memory → copy original data from CPU to GPU → call kernel to process data → copy result data from GPU to CPU → release GPU memory

Gauss-Seidel Iteration

- Gauss-Seidel iterative format

$$V_i^{(k+1)} = \frac{1}{Y_{ii}}\left(\frac{P_i - jQ_i}{\overset{*}{V}_i^{(k)}} - \sum_{j=1}^{i-1} Y_{ij} V_j^{(k+1)} - \sum_{j=i+1}^{n} Y_{ij} V_j^{(k)}\right) \qquad (3)$$

$$Q_i^{(k)} = -\mathrm{Im}[\overset{*}{V}_i^{(k)} (\sum_{j=1}^{i-1} Y_{ij} V_j^{(k+1)} + \sum_{j=i}^{n} Y_{ij} V_j^{(k)})] \qquad (4)$$

- Parallelization operations
  - summation operations in equation (3) and (4)

Newton-Raphson Iteration

- Parallelization operations
  - Jacobian matrix computation
  - **linear equations solver**
- Jacobian matrix computation

$$J = \begin{bmatrix} H & N \\ K & L \end{bmatrix} \tag{5}$$

P-Q Decoupled Iteration

- Parallelization operations
  - **linear equations solver**

Linear Equations Solver

- Gaussian elimination method
  - **forward elimination**
  - back substitution
- Augmented matrix

$$A=\begin{bmatrix} a_{11} & \cdots & a_{1k} & \cdots & a_{1n} & a_{1,n+1} \\ \vdots & \ddots & & & \vdots & \vdots \\ a_{k1} & & a_{kk} & & a_{kn} & a_{k,n+1} \\ \vdots & & & \ddots & \vdots & \vdots \\ a_{n1} & \cdots & a_{nk} & \cdots & a_{nn} & a_{n,n+1} \end{bmatrix} \qquad (6)$$

- *k*th forward elimination step

$$a_{kj}=a_{kj}/a_{kk},(j=k+1\sim n+1) \qquad (7)$$

$$a_{ij}=a_{ij}-a_{ik}*a_{kj},(i=k+1\sim n,\ j=k+1\sim n+1) \qquad (8)$$

Gaussian Forward Elimination (1)

❖ Kernel to process equation (7)

$$a_{kj} = a_{kj} / a_{kk}, (j = k+1 \sim n+1) \quad (7)$$

Algorithm 1 GAUSS ELIMINATION CUDA KERNEL A

Input: Augmented matrix in GPU memory: $augMatrixGPU$, number of rows in matrix $augMatrixGPU$: n, the Gauss forward elimination step: k.

$i \leftarrow blockIdx.x \times blockDim.x + threadIdx.x$
$j \leftarrow blockIdx.y * blockDim.y + threadIdx.y$
if $i == k$ and $j > k$ and $j < n+1$ and $augMatrixGPU[k \times (n+1) + k] \neq 0.0$ **then**
  $augMatrixGPU[k \times (n+1) + j] \leftarrow augMatrixGPU[k \times (n+1) + j] / augMatrixGPU[k \times (n+1) + k]$
end if

Gaussian Forward Elimination (2)

❖ Kernel to process equation (8)

$$a_{ij} = a_{ij} - a_{ik} * a_{kj}, (i = k+1 \sim n, j = k+1 \sim n+1) \qquad (8)$$

Algorithm 2 GAUSS ELIMINATION CUDA KERNEL B

Input: Augmented matrix in GPU memory: $augMatrixGPU$, number of rows in matrix $augMatrixGPU$: n, the Gauss forward elimination step: k.

1: $i \leftarrow blockIdx.x * blockDim.x + threadIdx.x$
2: $j \leftarrow blockIdx.y * blockDim.y + threadIdx.y$
3: **if** $i > k$ and $i < n$ and $j > k$ and $j < n+1$ and $augMatrixGPU[k \times (n+1) + k] \neq 0.0$ **then**
4: $\quad augMatrixGPU[i \times (n+1) + j] \leftarrow augMatrixGPU[i \times (n+1) + j] - augMatrixGPU[i \times (n+1) + k] \times augMatrixGPU[k \times (n+1) + j]$
5: **end if**

Gaussian Forward Elimination (3)

❖ Main process

Algorithm 3 GAUSS FORWARD ELIMINATION

Input: Augmented matrix in GPU memory: $augmentMatrix$, number of rows in matrix $augmentMatrix$: n.

$cudaMalloc((void^{**})\&aguMatrixGPU, sizeof(float) \times n \times (n+1))$

$cudaMemcpy2D(aguMatrixGPU, sizeof(float) \times (n+1), aguMatrix, sizeof(float) \times (n+1), sizeof(float) \times (n+1), n, cudaMemcpyHostToDevice)$

$dim3\ blockDim(22, 22)$

$dim3\ gridDim((n+blockDim.x-1)/blockDim.x, (n+1+blockDim.y-1)/blockDim.y)$

for $k \leftarrow 0$ to $n-1$ **do**

    $GaussKernelA <<< gridDim, blockDim >>> (aguMatrixGPU, n, k);$

    $GaussKernelB <<< gridDim, blockDim >>> (aguMatrixGPU, n, k);$

end for

$cudaMemcpy2D(aguMatrix, sizeof(float) \times (n+1), aguMatrixGPU, sizeof(float) \times (n+1), sizeof(float) \times (n+1), n, cudaMemcpyDeviceToHost)$

$cudaFree(aguMatrixGPU)$

Performance Evaluation

- Experiment platform
  - host: Intel i3-2100 CPU(3.10GHz) & 2G RAM
  - device: Nvidia GeForce GTS450 GPU(192 CUDA cores & 1G RAM)
  - software: Windows 7, CUDA 4.0
- Experiment power systems

System	Bus Count	Branch Count
IEEE9	9	9
IEEE30	30	41
IEEE118	118	186
IEEE300	300	357
Shandong	974	1449

Experiment Result (1)

- Gauss-Seidel solver

System	CPU Runtime (s)	GPU Runtime (s)	Speedup
IEEE9	0.0001	0.3276	0.0003
IEEE30	0.002	0.7051	0.0028
IEEE118	0.023	3.2963	0.007
IEEE300	0.3428	7.2992	0.047
Shandong	1.2147	19.603	0.062

Experiment Result (2)

- Newton-Raphson solver

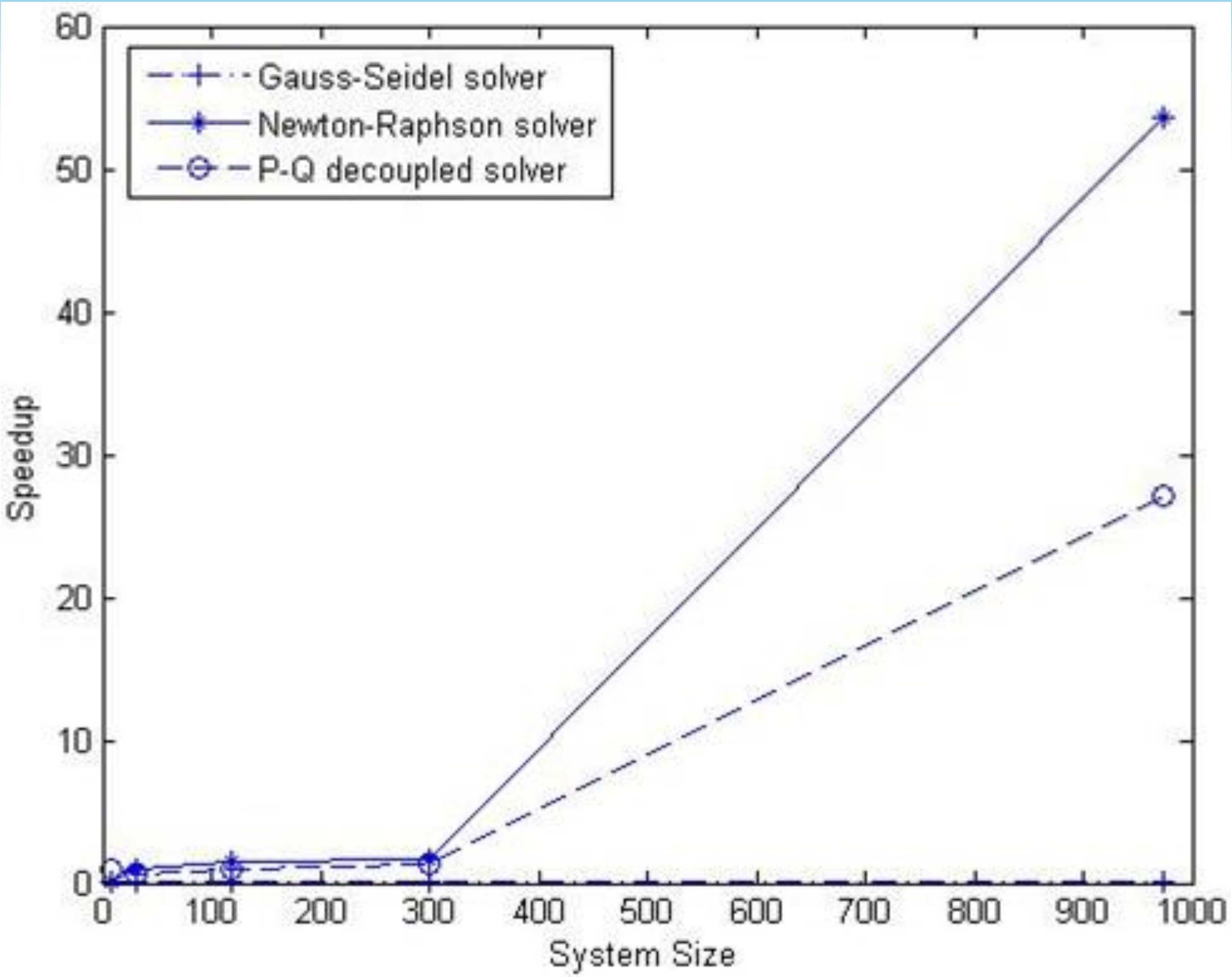
System	CPU Runtime (s)	GPU Runtime (s)	Speedup
IEEE9	0.0015	0.0094	0.1596
IEEE30	0.0098	0.0094	1.0426
IEEE118	0.3132	0.1997	1.5684
IEEE300	4.689	2.6848	1.7465
Shandong	583.831	10.881	53.656

Experiment Result (3)

- P-Q decoupled solver

System	CPU Runtime (s)	GPU Runtime (s)	Speedup
IEEE9	0.0047	0.0047	1.0
IEEE30	0.0081	0.0125	0.648
IEEE118	0.1137	0.117	0.9718
IEEE300	1.5107	1.1606	1.3017
Shandong	148.974	5.5068	27.0527

Result Analysis



Conclusion

- Parallelize three power flow solvers on GPU
  - bus admittance matrix computation
  - iteration process
- Compare speedup of three parallel power flow solvers
  - Newton-Raphson solver: best
  - P-Q decoupled solver: middle
  - Gauss-Seidel solver: worst

Future Work

- Improve speedup
- Reduce computation time
- Study different applications
- …

Thank You !

Q & A

Chunhui Guo   chunhui.guo@hotmail.com