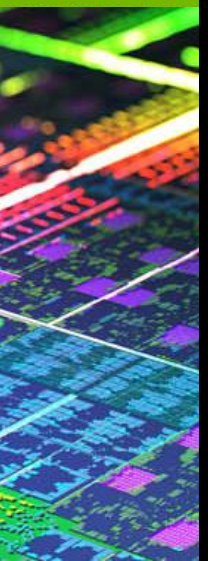
Jacobi-Davidson Eigensolver in Cusolver Library

Lung-Sheng Chien, NVIDIA
lchien@nvidia.com



Outline

- CuSolver library
  - cuSolverDN: dense LAPACK
  - cuSolverSP: sparse LAPACK
  - cuSolverRF: refactorization
- Jacobi-Davidson eigenvalue solver
- Batch sparse QR
- Conclusions

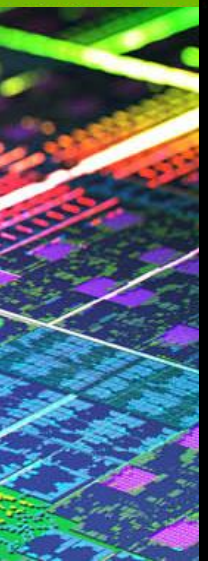
CuSolver library

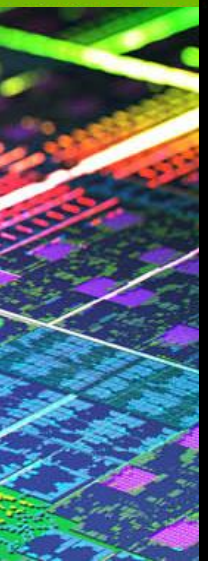
- CuSolverDN: dense LAPACK
  - linear solver: QR, Choleksy and LU with partial pivoting
  - bidiagonalization
  - symmetric eigenvalue solver (syev)

- CuSolverSP: sparse LAPACK
  - linear solver: QR, Choleksy and LU with partial pivoting
  - batch linear solver: QR
  - eigenvalue solver: shift-inverse, Jacobi-Davidson

- CuSolverRF: LU without pivoting

- Only available in CUDA 7.0





Outline

- CuSolver library
- Jacobi-Davidson eigenvalue solver
- Batch sparse QR
- Conclusions

Jacobi-Davidson eigenvalue solver

- Given a *m*-by-*m* Hermitian matrix *A*, find *p* eigen-pairs near target τ
- Ritz pair on subspace (cuSolverDN, steqr)

$$(V^H A V)s = \theta(V^H V)s$$

- Residual evaluation (cuSparse, csrmm)

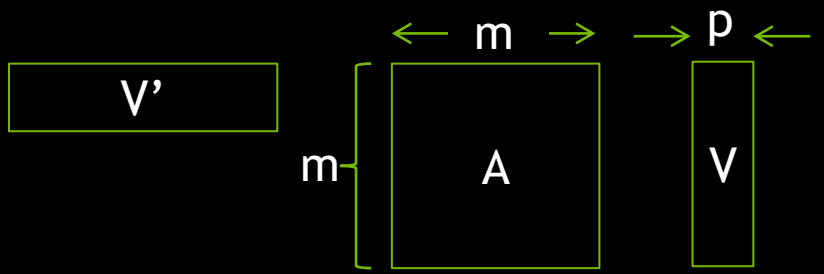
$$r_j = Au_j - \theta_j u_j \quad \text{for } j = 1{:}\,p$$

- Update search space (cuSolverSP, batch QR)

$$v_j = \left(A - \theta_j\right)^{-1} u_j \quad \text{for } j = 1{:}\,p$$

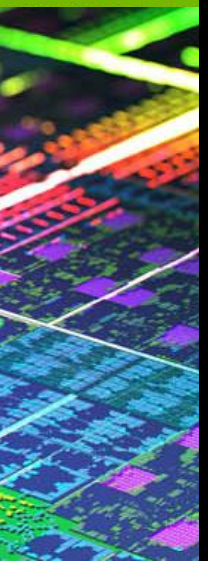
$$w_j = \left(A - \theta_j\right)^{-1} r_j \quad \text{for } j = 1{:}\,p$$

- Orthogonalization of V (dense QR)



Complexity analysis

- Ritz pair computation, $(V^H AV)s = \theta(V^H V)s$
  - tridiagonalization needs O(p^3)
- Residual evaluation $r_j = Au_j - \theta_j u_j$ for $j = 1{:}p$
  - matrix-vector multiplication needs O($m\frac{nnz}{m}p$)
- Update search space, $v_j = \left(A - \theta_j\right)^{-1} u_j$ and $w_j = \left(A - \theta_j\right)^{-1} r_j$
  - depends on nnz(Q + R), 2D 5-point stencil needs O($10m^{1.5}$) to do QR [1]
- Orthogonalization: O(mp^2)
- Typical size of *p* is 100, and typical size of *m* is 1 million, so QR dominants the computation, need a good way to perform QR *p* times, i.e. batchQR



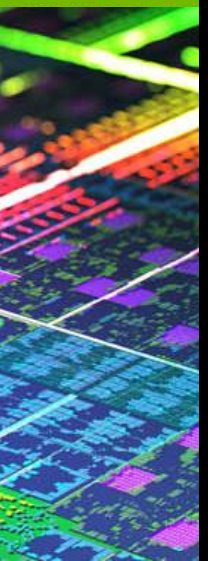
Design spec

- Ritz pair computation
  - for typical size p, O(p^3) cannot saturate a GPU, CPU is good enough

- Residual evaluation
  - memory bound and easy to achieve maximum bandwidth for regular matrix, a good fit on GPU

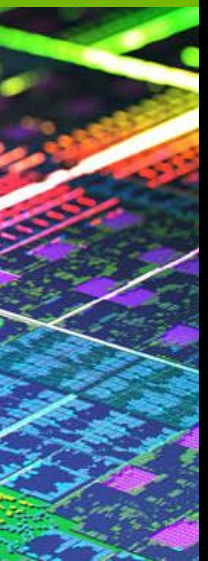
- Update search space
  - symbolic analysis of QR: only done once on CPU (not bottleneck)
  - numerical factorization of QR
    cuSolverSP provides CPU path (openmp) which is good for single QR; however GPU is better for batch QR because of high bandwidth

- Orthgonalization: computational-bound, GPU is preferred



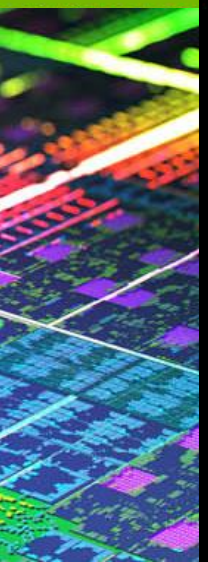
Challenges

- Huge zero fill-in makes QR a non-feasible solution
- Exact inversion is not necessary, instead, choose a “preconditioner” M such that |A-M| is much smaller than |M|, or M is just a block diagonal of A
- If M is a block diagonal of A, then 1) parallelism improved by (m/blockSize) times, and 2) zero fill-in of QR goes down dramatically
- The cuSolverSP can count “nonzeros of QR” in O(|A|), so the user can measure zero fill-in of different M and pick up the best one



Outline

- CuSolver library
- Jacobi-Davidson eigenvalue solver
- Batch sparse QR
- Conclusions



Batch sparse QR

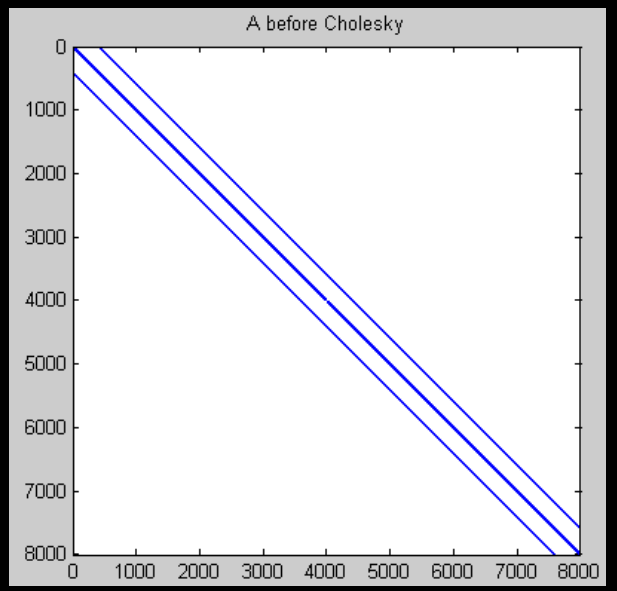
- Solve ($A_j x_j = b_j$) for $j = 1{:}\,N$, each A_j has the same sparsity pattern
- Symbolic analysis is done for SINGLE matrix
- Numerical factorization is memory-bound, limited by parallelism and memory efficiency
- Batch operation improves both parallelism and memory efficiency

$$factorization: Q_j A_j = R_j$$
$$projection: y_j = Q_j b_j$$
$$backward\ substitution: x_j = R_j \backslash y_j$$

Jacobi-Davidson: $A_j = (A - \theta_j)$

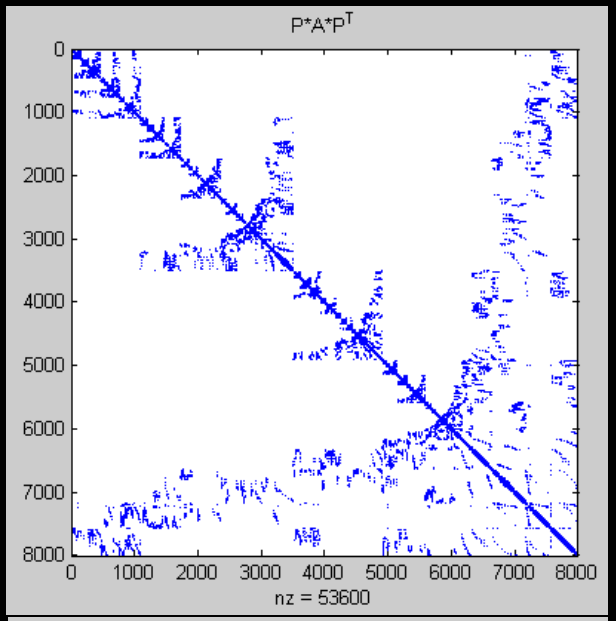
Effect of zero fill-in

Original A

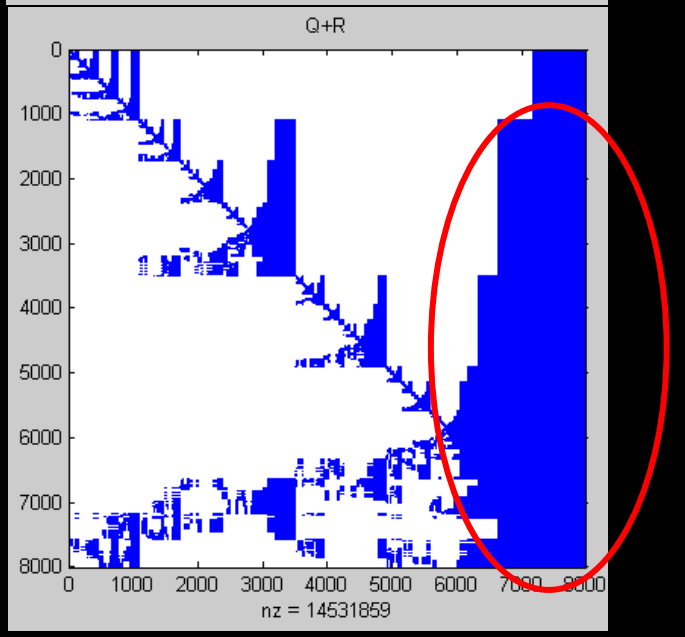


7-point stencil

colamd(A)



QR(colamd(A))



huge zero fill-in

Benchmark: spd matrices

- From Florida Matrix Collection except Laplacian operator
- Laplacian operator is standard Finite Difference 5-pt and 7-pt with Dirichlet boundary condition
- nnzA is # of nonzeros of A
- nnzG is # of nonzero of Q+R after *colamd*
- levels is # of levels in QR

matrix name	m	nnzA	nnzG	nnzA/m	nnzG/m	levels
mhd4800b.mtx	4800	16160	39436	3.4	8.2	1198
LF10000.mtx	19998	59990	159966	3.0	8.0	19998
1138_bus.mtx	1138	2596	12307	2.3	10.8	91
Chem97ZtZ.mtx	2541	4951	89086	1.9	35.1	101
nos6.mtx	675	1965	25495	2.9	37.8	183
ex33.mtx	1733	11961	100945	6.9	58.2	618
lap2D_5pt_n100.mtx	10000	29800	949109	3.0	94.9	786
plat1919.mtx	1919	17159	223337	8.9	116.4	788
bcsstk34.mtx	588	11003	110472	18.7	187.9	588
bcsstk26.mtx	1922	16129	258294	8.4	134.4	636
gyro_m.mtx	17361	178896	2852731	10.3	164.3	1201
bodyy4.mtx	17546	69742	2573516	4.0	146.7	1475
bodyy5.mtx	18589	73935	2882458	4.0	155.1	1389
aft01.mtx	8205	66886	1743949	8.2	212.5	1920
minsurfo.mtx	40806	122214	5864828	3.0	143.7	2094
Muu.mtx	7102	88618	1867034	12.5	262.9	1640
nasa1824.mtx	1824	20516	445042	11.2	244.0	714
ex9.mtx	3363	51417	1171575	15.3	384.4	1864
Kuu.mtx	7102	173651	4735207	24.5	666.7	7102
s1rmt3m1.mtx	5489	112505	1941004	20.5	353.6	3211
lap3D_7pt_n20.mtx	8000	30800	5301344	3.9	662.7	2166
bcsstk13.mtx	2003	42943	1326440	21.4	662.2	1492
sts4098.mtx	4098	38227	3378440	9.3	824.4	2202
Pres_Poisson.mtx	14822	365313	8218300	24.6	554.5	13870
shallow_water1.mtx	81920	204800	14965626	2.5	182.7	2925
finan512.mtx	74752	335872	22439963	4.5	300.2	3018

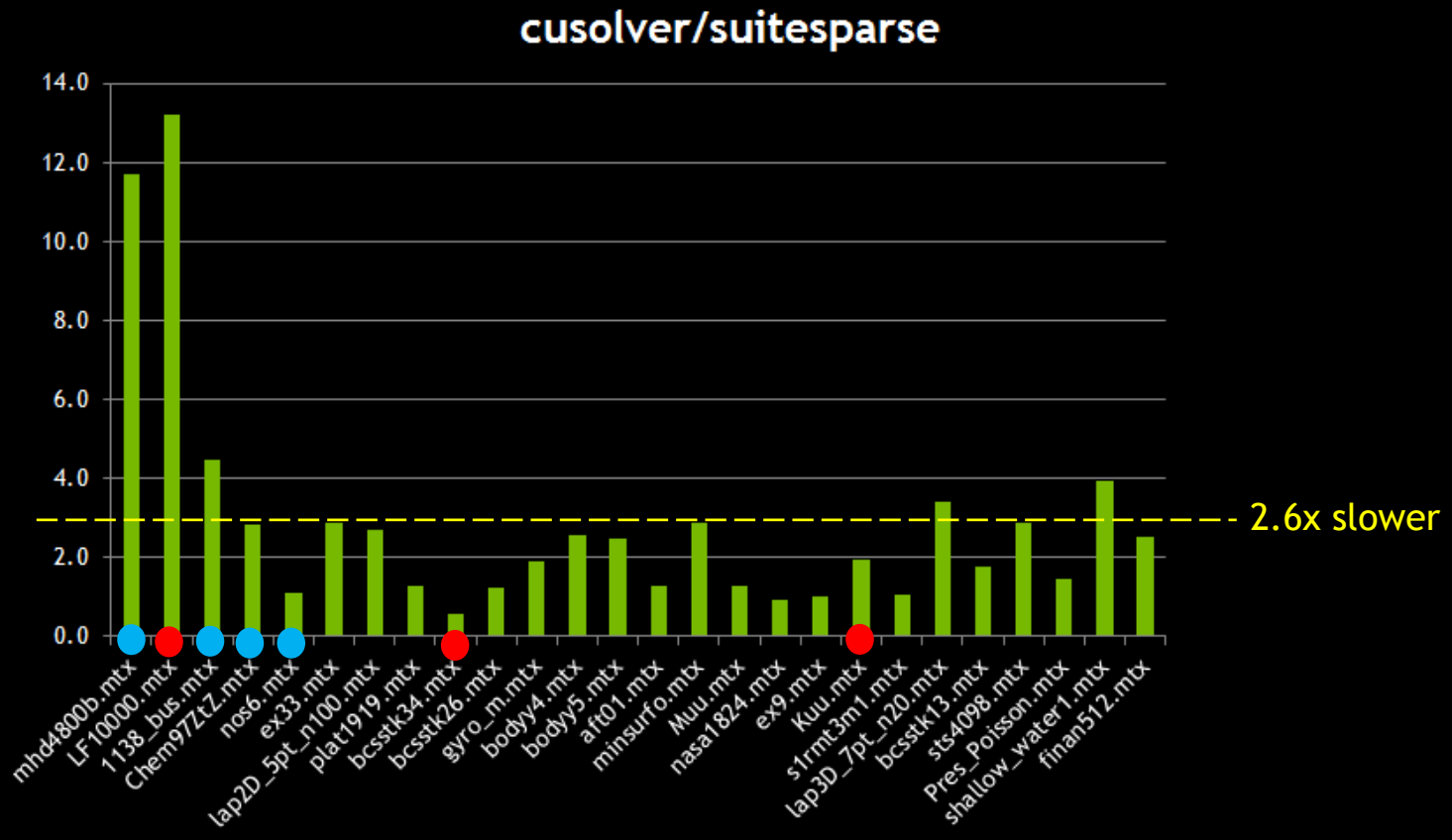
http://www.cise.ufl.edu/research/sparse/matrices/

CuSolver QR (GPU) versus SPQR (SuiteSparse, CPU)

- CuSolver QR is 2.6x slower than SPQR in average
- CuSolver QR is left-looking non-supernodal approach, may be much slower if there are many big super nodes.

GPU: K40
CPU: i7-3930K CPU @ 3.20GHz

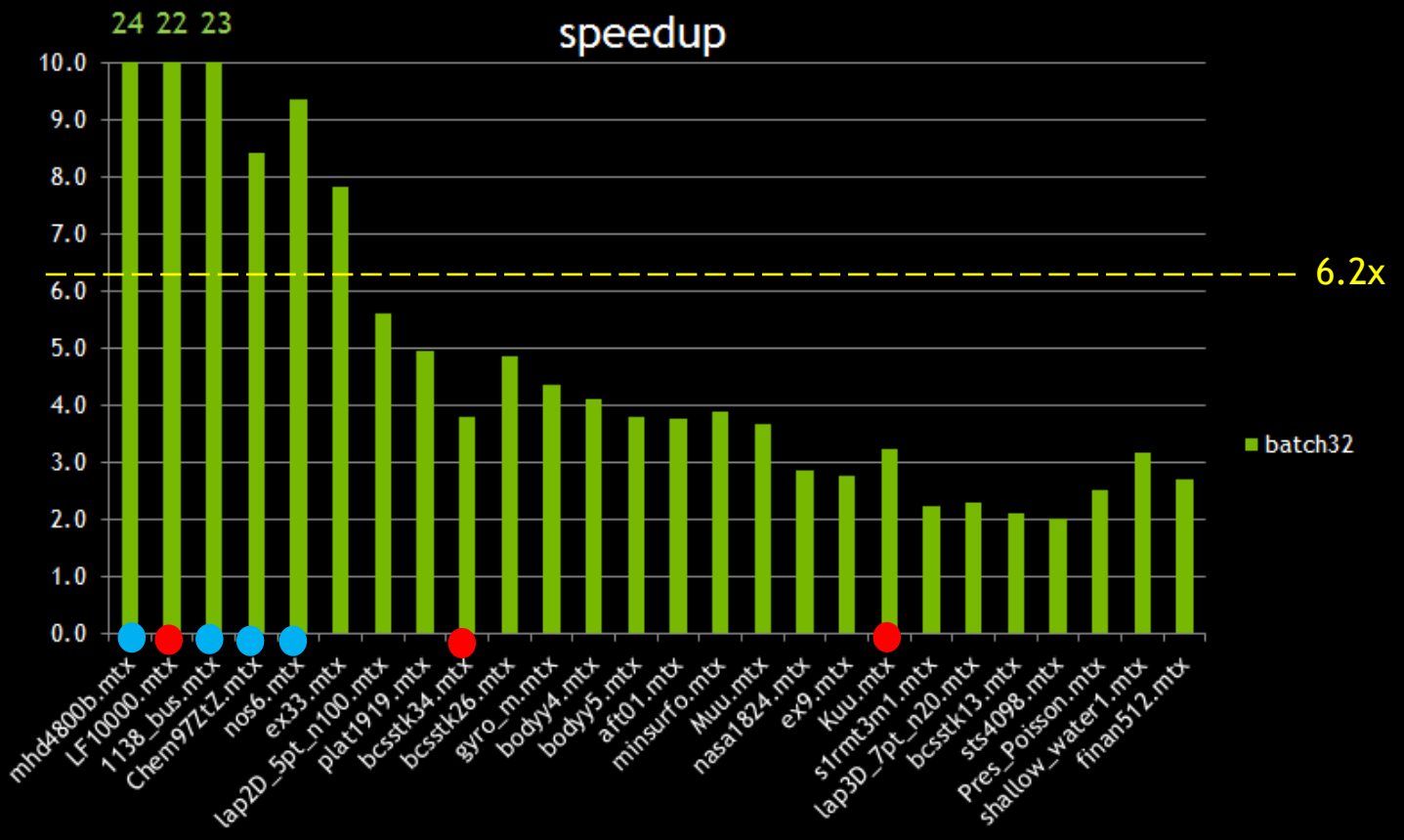
SuiteSparse-3.6.0
/SPQR/Demo
/qrdemo.c



Batch QR versus single QR

- speedup = T(batchSize * QR) / T(batchQR)
- The speedup starts from 2x, up to 24x
  - performance comes from memory efficiency (no extra parallelism for batch32)

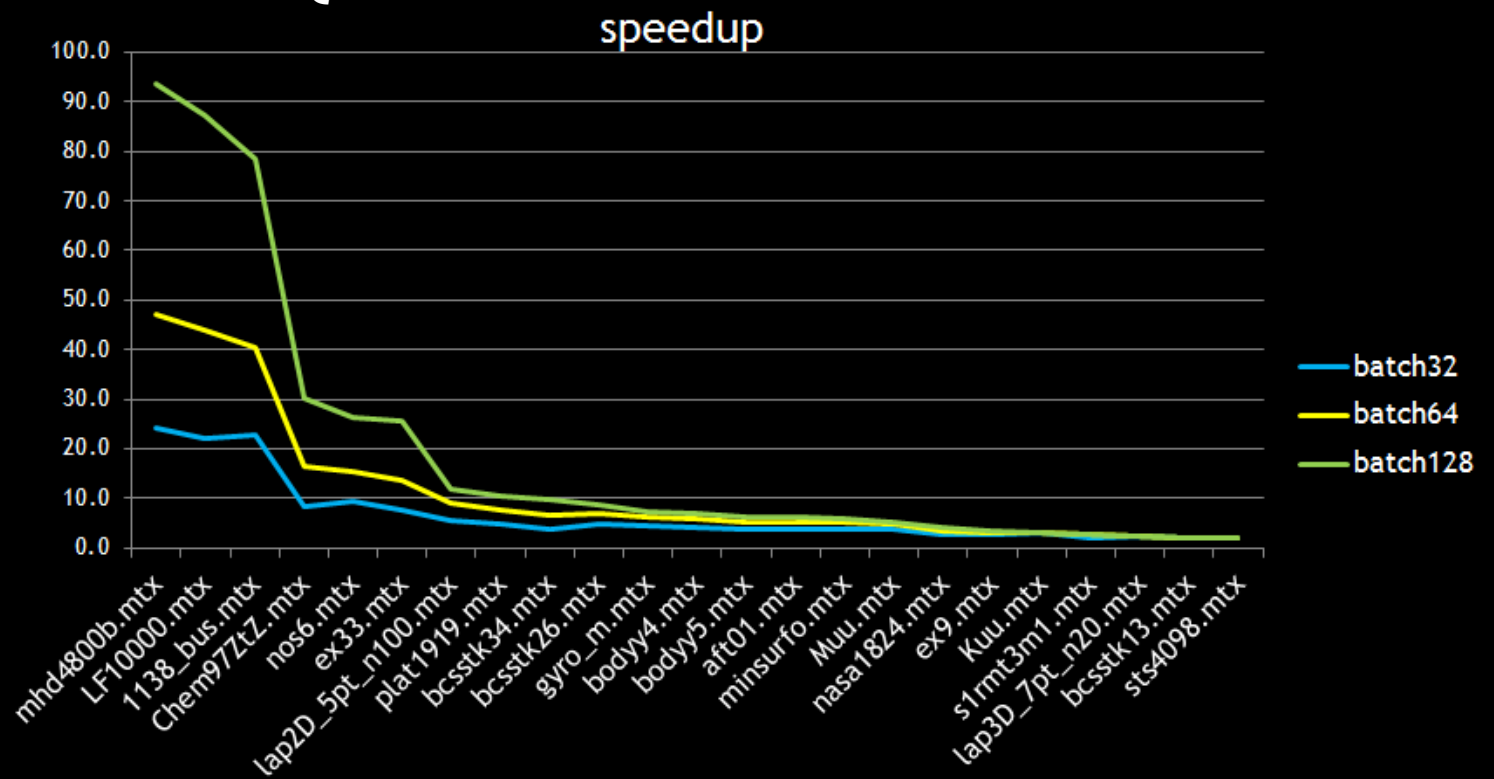
GPU: K40



performance versus batchSize

- speedup = T(batchSize * QR) / T(batchQR)
- If bandwidth is not saturated, the more batch size, the more performance
- If zero fill-in is large (rightmost matrices), batch QR does not scale
- Batch QR is 6x faster than SPQR

GPU: K40

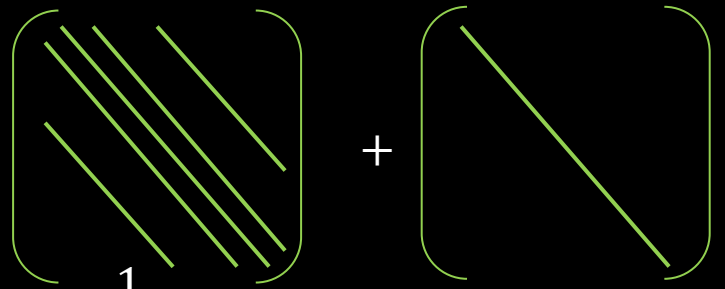


Example: Hydrogen atom

- Eigenvalue problem

$$H\psi = E\psi$$

- Hamiltonian $H = -\frac{1}{2}\Delta + V_{ext}$ is composed of kinetic energy, external potential $V_{ext} = \frac{-1}{R}$
- Kinetic energy operator is 7-point stencil

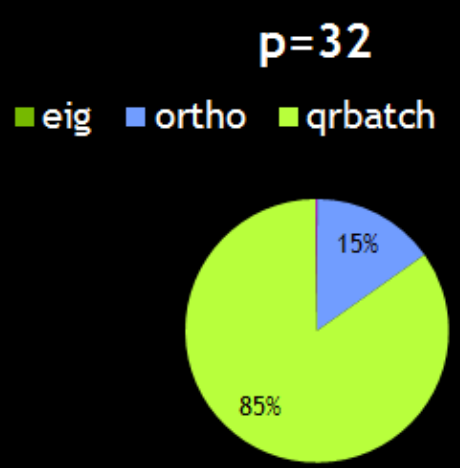
$H =$  $+$

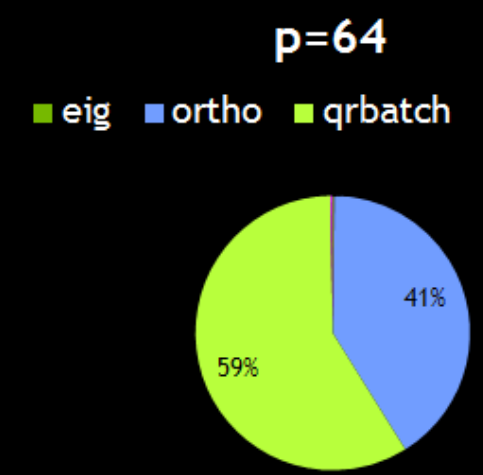
$-\frac{1}{2}\Delta$ V

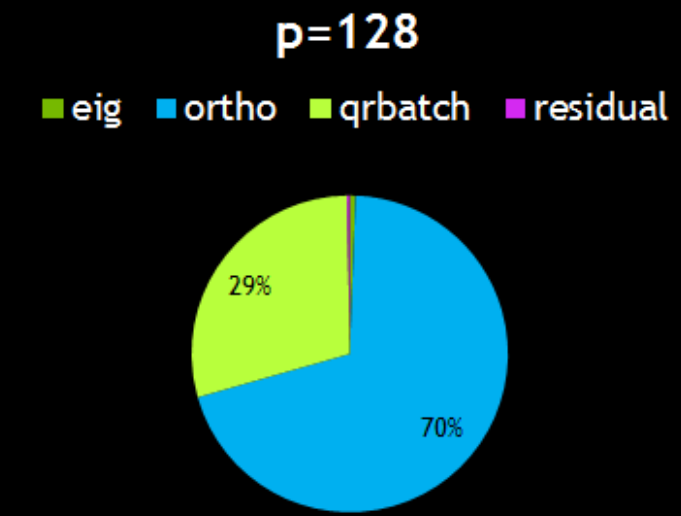
Objective: find 10 eigenvalues near $\tau = -15$

Preconditioner: Block diagonal

- $A = -\frac{1}{2}\left(D_x^2 \oplus I \oplus I + I \oplus D_y^2 \oplus I + I \oplus I \oplus D_z^2\right) + V$
- $M = -\frac{1}{2}\left(D_x^2 \oplus I \oplus I + I \oplus D_y^2 \oplus I + I \oplus I \oplus \mathbf{diag}(D_z^2)\right) + V$
- A is 32768 x 32768, nnz(A) = 223232, nnz(M)=159744
- nnz(Q+R) = 1773056

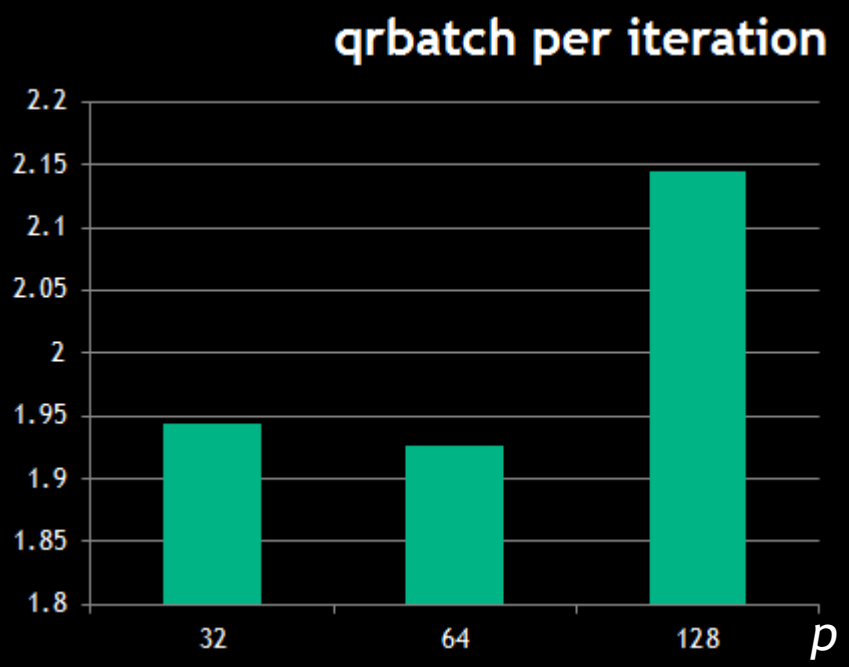


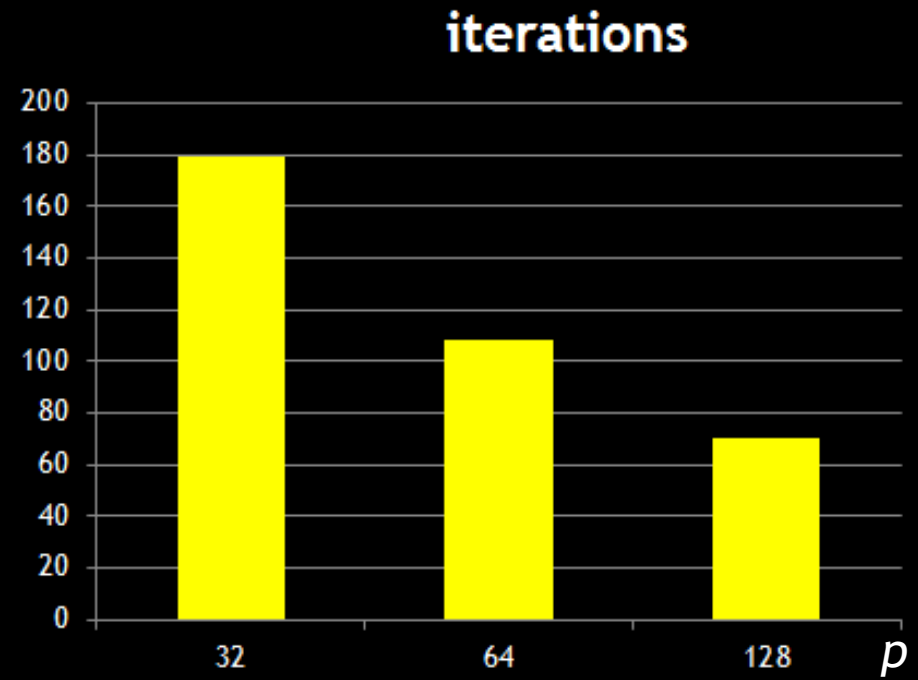


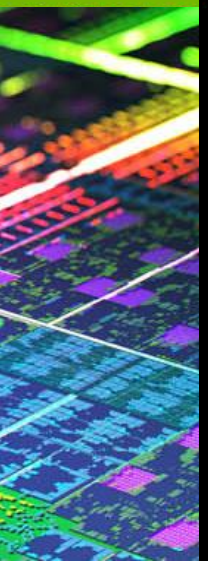


Performance

- QR batch has the same runtime for p=32, 64 and 128
- Bigger p, less iterations and faster convergence

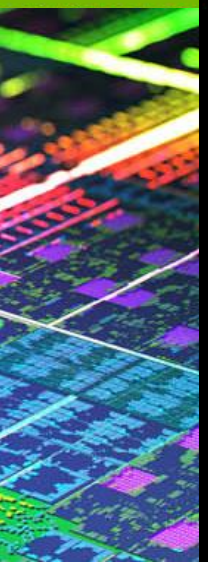






Conclusions

- CuSolver provides basic supports on linear solver and eigenvalue solver
  - CuSolverDn is better than CPU for large matrix
  - CuSolverSP provides both CPU and GPU implementations, the performance depends on sparsity pattern of the matrix
- Batch sparse QR is efficient in subspace eigenvalue solver
- zero fill-in can affect scalability of batch QR, it is necessary to choose a proper preconditioner M (for example, block diagonal of A)
- Symmetric eigenvalue solver (dense and sparse) is on the roadmap



Thank you !

- [1] Alan George, Joseph W. Liu, Computer Solution of Large Sparse Positive Definite Systems, Prentice-Hall series in computational mathematics
- [2] umfpack, cholmod, KLU, http://www.cise.ufl.edu/research/sparse/