Accelerating and Scaling Lanczos Diagonalization with GPGPU

Bill Brouwer, Filippo Spiga,
Pierre-Yves Taunay, Sreejith GJ

Nvidia GTC 2013

wjb19@psu.edu

Outline

- Introduction
- Applications
  - QE
  - FQHE
- Theory
  - Diagonalization
    - Lanczos algorithm
    - QR algorithm
- Implementation
  - CuBlas &MKL
  - Distributed GPUs
    - MPI + GPUdirect
- Results
- Conclusions

wjb19@psu.edu

Introduction

- Diagonalization or computation of eigenvalues is arguably the most important step in condensed matter calculations; generally arises from solution to Schrödinger's equation

- Myriad systems with different Hamiltonians *H* are studied, with a view to understanding the energy spectrum or range of values for ε_i where i ranges from 1 to *N* particles, levels, electrons etc

- New and fascinating science takes place with increasing *N*; GPUs give us the opportunity to scale and accelerate key algorithms in condensed matter and thus push the boundaries

- The Hamiltonian operator models the total energy for a particular physical model in an abstract way, whereas the spectrum represents predicted measurable (real) quantities

wjb19@psu.edu

Introduction

•Often *H* is incredibly complicated and one resorts to approximations; regardless one can expand the wavefunctions using a basis over a field with defined inner product, giving rise to an eigenvalue problem

•For symmetric (Hermitian) matrices, we can solve this by diagonalizing the Hamiltonian using the Lanczos algorithm.

•In this work we are concerned with two key application areas:

- The self-consistent field cycle steps with a plane wave DFT code, Quantum Espresso
  - Moderately large, dense matrices of order <<1e6
  - Generally need many eigenvalues & eigenvectors

- Producing spectra in studies of the Fraction Quantum Hall Effect
  - Massive, less dense matrices of order > 1e6
  - Generally need few eigenvalues & eigenvectors

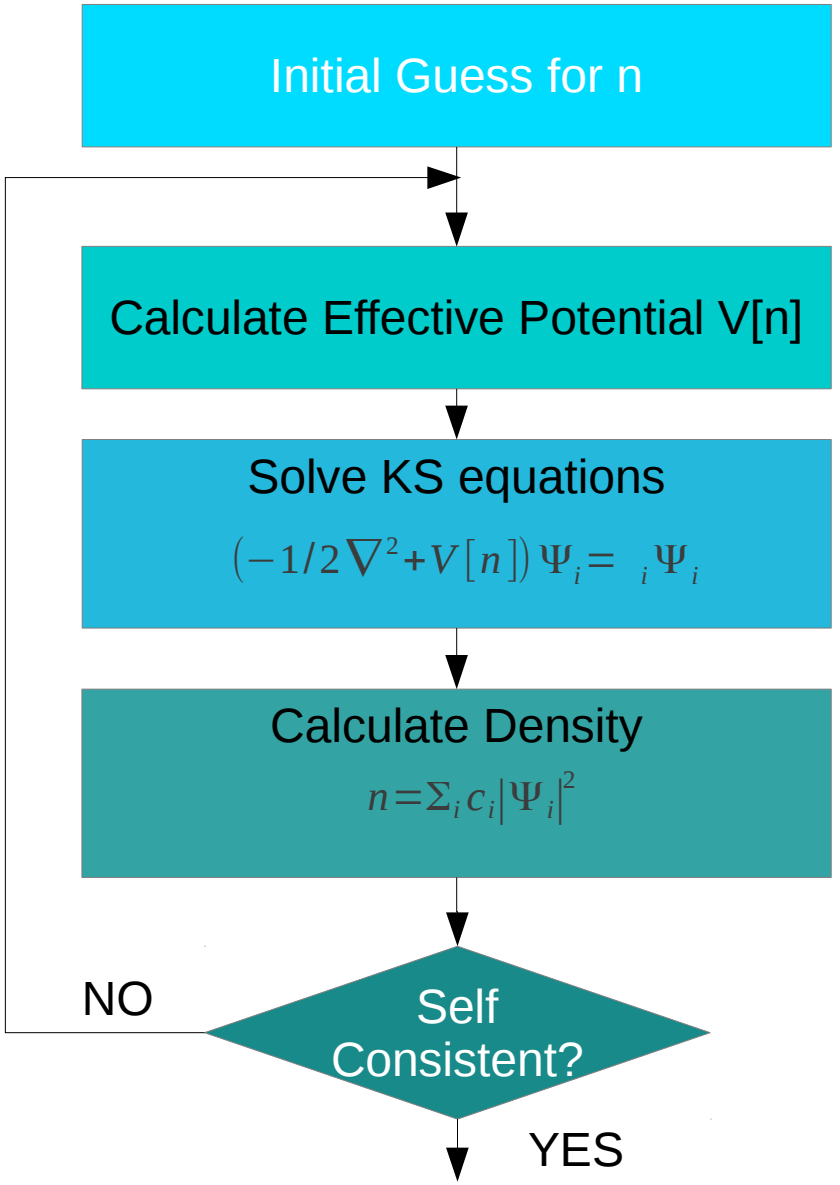
wjb19@psu.edu

Application : Quantum Espresso

- QE allows us to solve the ground state electronic structure problem in order to calculate and study material properties from first principles

- This includes solving the Kohn-Sham equations of Density Functional Theory (DFT), approximation to many body (electron) problem → diagonalization and FFT are bottlenecks for computation

- The current parallel Davidson algorithm for diagonalization is based on SCALAPACK (sans GPU)



wjb19@psu.edu

Application : Fractional Quantum Hall Effect

- FQHE gives a fascinating example of strongly correlated electronic system; many examples of FQHE are well understood (1/3,2/5,3/7 etc), many are not (5/2,12/5,6/13 etc)

- Computational study of small systems provides new insights into properties of the quantum state describing the phenomena – such as the quantum numbers of ground state, excitations

- Also helps in better testing of the theories

- Bottleneck step in numerical studies involve Lanczos diagonalization of large Hamiltonians

- Ability to diagonalize larger Hamiltonians immediately translates to far better understanding of the Physics

wjb19@psu.edu

Outline

- Introduction
- Applications
  - QE
  - FQHE
- Theory
  - Diagonalization
    - Lanczos algorithm
    - QR algorithm
- Implementation
  - CuBlas &Mkl
  - Distributed GPUs
    - MPI + GPUdirect
- Results
- Conclusions

wjb19@psu.edu

Theory

- Diagonalization by QR algorithm is expensive for even moderate N, since compute scales as $O(N^3)$

- In condensed matter, often we don't care about the whole spectrum since Probability(higher energies) < Probability(lower energies) cf Boltzmann distribution

- Lanczos (symmetric form of Arnoldi) is an iterative, power method that is less expensive than QR; storage costs can be kept to a minimum as well

- Worst calculation throughout iterations is GEMV
  - Based on earlier statement, often suffices to calculate just a few e-vectors/e-values
  - Can periodically restart from last calculated lanczos vector, throwing out earlier vectors in basis

wjb19@psu.edu

Basic Lanczos Algorithm

input : Hermitian $A \in \mathbb{C}^{N \times N}$
output: Matrix $T \in \mathbb{C}^{M \times M}$ w/ principal and sub-diagonals α, β

initialize;
$\mathbf{q} = \mathbf{x}/\|\mathbf{x}\|, Q_1 = [\mathbf{q}]$;
$\mathbf{r} = A\mathbf{q}$, $\alpha_1 = \mathbf{q} * \mathbf{r}$;
$\mathbf{r} = \mathbf{r} - \alpha_1 \mathbf{q}$;
for $j \leftarrow 1$ **to** M **do**
    $\mathbf{v} = \mathbf{q}$, $\mathbf{q} = \mathbf{r}/\beta_{j-1}$, $Q_j = [Q_{j-1}, \mathbf{q}]$;
    $\mathbf{r} = A\mathbf{q} - \beta_{j-1}\mathbf{v}$;
    $\alpha_j = \mathbf{q} * \mathbf{r}$;
    $\mathbf{r} = \mathbf{r} - \alpha_j \mathbf{q}$;
    $\beta_j = \|\mathbf{r}\|$;
end

wjb19@psu.edu

Lanczos Algorithm : Other steps

- Lanczos vectors q aren't exactly orthogonal; schemes abound for correcting, at this stage code monitors the dot product of $q_{j-1} * q_j$ and periodically applies Gram-Schmidt throughout iterations

- The Lanczos algorithm creates a tridiagonal matrix T, with principle α_0 and minor $\beta_{-1} = \beta_{+1}$ diagonals, which must be solved for the (Lanczos) eigenvalues and eigenvectors

- These are the same e-values as those for A, *however eigenvectors for A if required need to be reconstructed from the Lanczos vectors*

- We could use a GPU version of the QR algorithm for the final tridiagonal matrix (details in another talk), however the implicit tridiagonal algorithm is ~ $O(N^2)$ so for now this final step is performed strictly on CPU

wjb19@psu.edu

(Implicit) Tri-Diagonal QR Algorithm

```
while ( m > 1){
     // calculate wilkinson shift mu
     mu = f(alpha_m,alpha_m-1,beta_m);

     // apply to first beta,alpha elements

     x = alpha[0]-mu;
     y = beta[0];
     for k=1:m=1

          //calculate givens rotations eg., see Bindel et al
          if (m>2)
               [c,s] = givens(x,y)
          else
          // For the last iteration, we need to diagonalize the remaining 2x2 exactly
               th      = 0.5*atan2(2*beta[0], (alpha[1] - alpha[0]));
               s       = sin(th);
               c       = cos(th);
          end

          // update alpha & beta
          alpha_k+1 = g(alpha_k,beta_k,c,s);
          beta_k+1  = h(alpha_k,beta_k,c,s);

          // update e-vectors Q if necessary
          Q_new = Q_old * [c s; -s c];
     end

     if (tol) m--;
end
```

wjb19@psu.edu

Outline

- Introduction
- Applications
  - QE
  - FQHE
- Theory
  - Diagonlization
    - Lanczos algorithm
    - QR algorithm
- Implementation
  - CuBlas &Mkl
  - Distributed GPUs
    - MPI + GPUdirect
- Results
- Conclusions

wjb19@psu.edu

Implementation

- Single node version of basic Lanczos is comprised of MKL or cublas routines

- New API as of cuda 4.0 (use header cublas_v2.h), can still use legacy

- Introduction of handle allows for easier use of threads

- Scalars can be passed by reference on host or device, allows for asynchronous operation of routines

- CublasSetAtomicsMode()
  - *Several routines have alternate implementations that allow for the use of atomics*
  - Faster, but potential numerical differences between results

wjb19@psu.edu

Implementation

- **Routines used from cublas in basic Lanczos:**
- cublas<t>nrm2()
  - Euclidean norm of vector
- cublas<t>gemv()
  - Matrix vector/multiply, r update
- cublas<t>gemm()
  - For orthogonalization

- **Currently working to incorporate:**
- cublas<t>symv()*
  - Symmetric matrix/vector multiply
- cublas<t>hemv()*
  - Hermitian matrix/vector multiply
- cublas<t>hbmv()
  - Hermitian banded matrix/vector multiply
- cublas<t>hpmv()
  - Hermitian packed matrix/vector

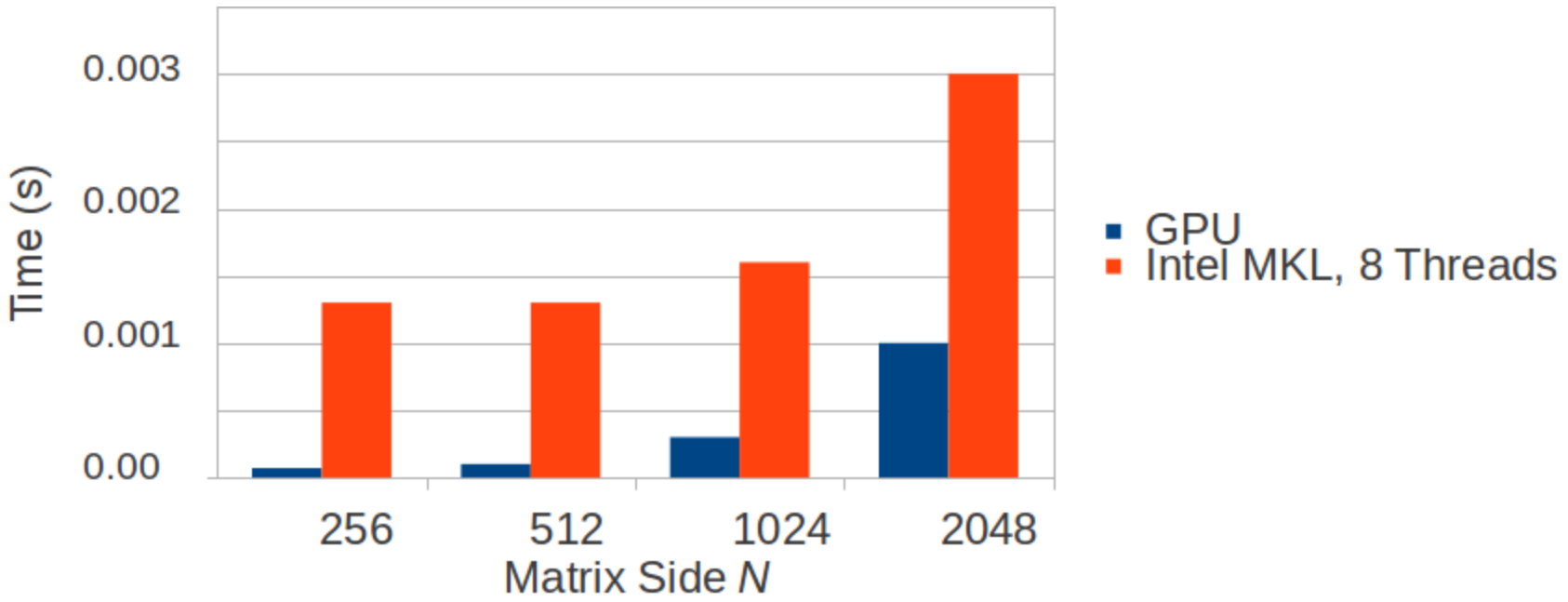
*Faster implementation using atomics that can be enabled with cublasSetAtomicsMode()

wjb19@psu.edu

Implementation

•GPU (M2090) provides excellent results for basic Lanczos; as much as 3x vs 8 threads running MKL routine (Intel x5550)

•Obviously need to transfer data across PCIe + network in MPI enabled Lanczos will degrade this performance



wjb19@psu.edu

GPUDirect

•In moving to distributed memory/MPI, we can take advantage of GPUDirect → improved interactions with third party devices, in particular PCIe & Infiniband

•Under MPI, IB and GPU drivers share the same pinned memory (since GPUDirectv2/CUDA 4.0)

•MPI calls can use device pointers directly; must be 64 bit/ RH Linux Kernel >= 2.6.18 and NVIDIA Driver >= v270 (*Earlier kernels may work with patch, but communication probably routed through host ala GPUDirectv1)

•As of GPUDirectv3/CUDA 5.0, RDMA has been introduced improving communication between GPUs on network even further (*Need appropriate IB driver)

•Support has existed in OpenMPI since 1.7(?), although must still build from trunk (at time of talk/writing this); many (but not all) MPI calls are supported (see refs at end of talk)

wjb19@psu.edu

GPUDirect : Tuning

•Build openMPI from trunk with cuda support `--with-cuda(=DIR)` ; link against a `--enable-debug` build & check operation of application with gdb etc

```
1138    in common_cuda.c
(gdb) bt
#0  mca_common_cuda_is_gpu_buffer (pUserBuf=0x200000200) at
common_cuda.c:1138
#1  0x00002b0cc6b4ac42 in mca_cuda_convertor_init (convertor=0xd2dd50,
pUserBuf=0x200000200) at opal_datatype_cuda.c:57
#2  0x00002b0cc6b37c1f in opal_convertor_prepare_for_recv
(convertor=0xd2dd50, datatype=0x613460, count=32, pUserBuf=0x200000200)
    at opal_convertor.c:548
```

```
/* If not enabled, then nothing else to do */
    if (!opal_cuda_enabled) {
        return;
    }
    if (ftable.gpu_is_gpu_buffer(pUserBuf)) {
        convertor->cbmemcpy = (memcpy_fct_t)&opal_cuda_memcpy;
        convertor->flags |= CONVERTOR_CUDA;
    }
}
```

wjb19@psu.edu

GPUDirect : Operation

- Use `cudaMallocHost` for MPI+GPU buffers, make sure pinned memory is supported by device

```
struct cudaDeviceProp p;
cudaGetDeviceProperties(&p,0);
int support = p.canMapHostMemory;

if(support == 0){
    fprintf(stderr,"%s does not support mapping host memory\n",hostname);
    MPI_Finalize();
    exit(1);
}


cerror =
cudaHostAlloc((void**)&r,sizeof(cuDoubleComplex)*matSize,cudaHostAllocMapped);

cerror = cudaHostGetDevicePointer(&d_r,r,0);
```

wjb19@psu.edu

GPUDirect : Benchmarking

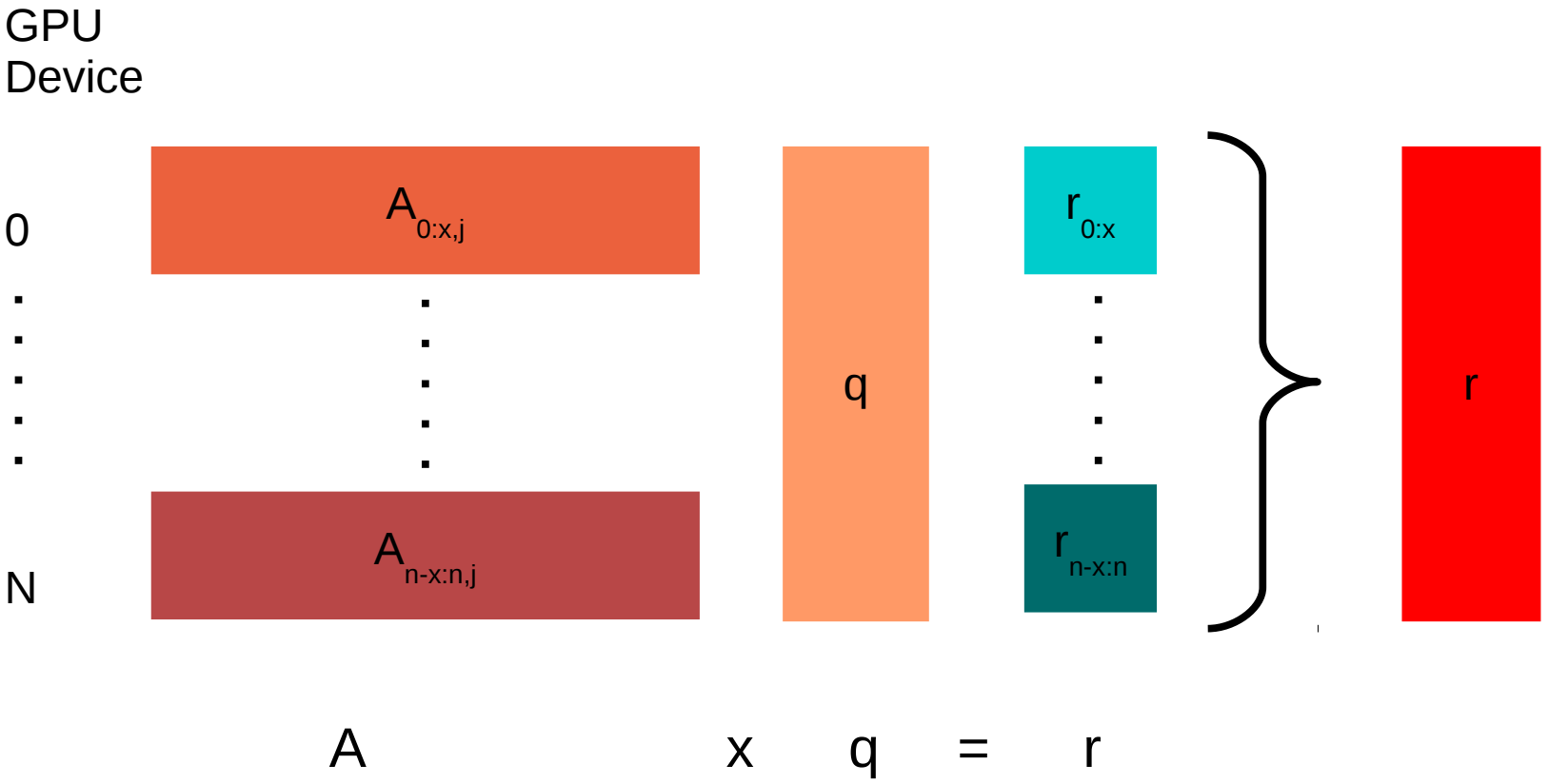
- Initial results encouraging for performing network transfers on simple payloads between devices:

```
[wjb19@lionga scratch]$ mpicc -I/usr/global/cuda/4.1/cuda/include
-L/usr/global/cuda/4.1/cuda/lib64 mpi_pinned.c -lcudart

[wjb19@lionga scratch]$ qsub test_mpi.sh
2134.lionga.rcc.psu.edu

[wjb19@lionga scratch]$ more test_mpi.sh.o2134
Process 3 is on lionga7.hpc.rcc.psu.edu
Process 0 is on lionga8.hpc.rcc.psu.edu
Process 1 is on lionga8.hpc.rcc.psu.edu
Process 2 is on lionga8.hpc.rcc.psu.edu
Host->device bandwidth for process 3: 2369.949046 MB/sec
Host->device bandwidth for process 1: 1847.745750 MB/sec
Host->device bandwidth for process 2: 1688.932426 MB/sec
Host->device bandwidth for process 0: 1613.475749 MB/sec
MPI send/recv bandwidth: 4866.416857 MB/sec
```

wjb19@psu.edu

Parallel Strategy for Lanczos

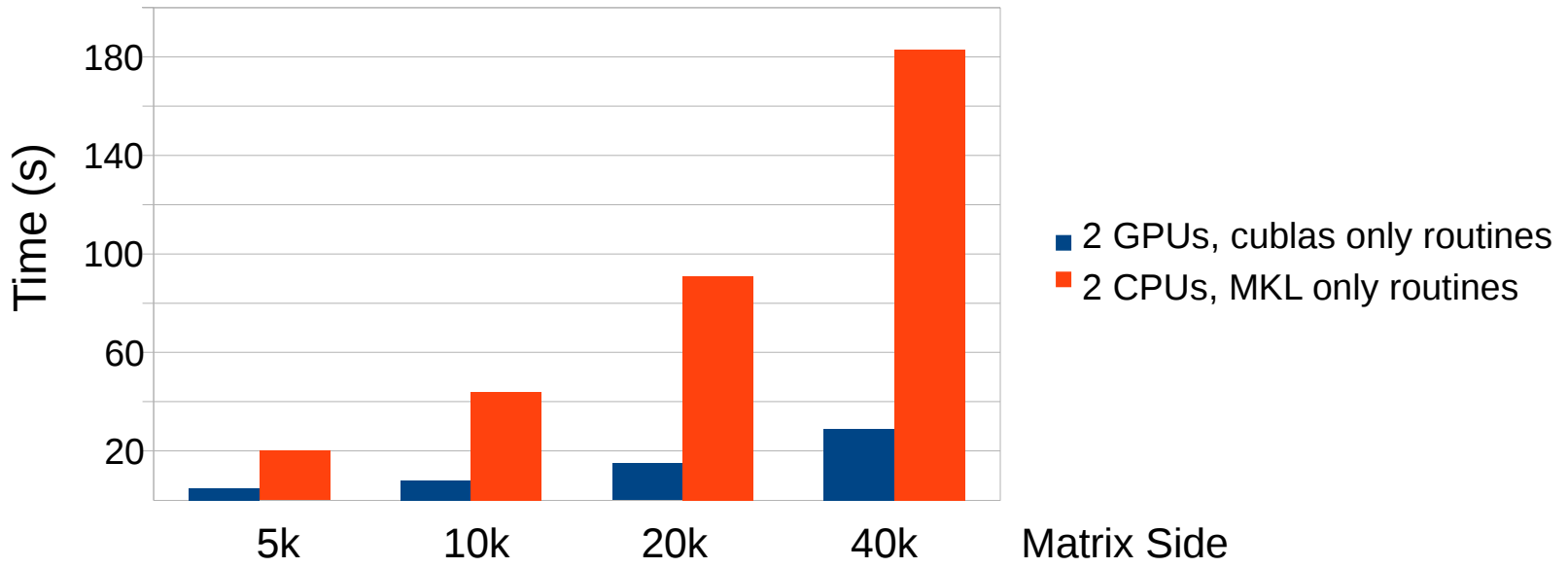
- Simply stripe *A* across GPUs; each calculates a portion of the *r* update (line 7 of algorithm) and then `MPI_Allgather` collects to each



wjb19@psu.edu

Parallel Results

- Using two devices, GPU (M2070) is ~5-6x over CPU (Intel x5550), when running Lanczos with limited (re)orthogonalization
- Dense matrices, 256 eigenvalues calculated, `gemv` for r update



wjb19@psu.edu

Conclusions/Directions

- Initial results promising ~ 24x single node, 5-6x in parallel, over using similar routines on CPUs, scaling to more cores needs some work

- Need to handle compression particularly for FQHE application, currently investigating CUSP for sparse matrices

- Introducing more sophisticated tiling to allow use of symmetric cublas routines w/ atomics

- Better approaches to maintaining orthogonality
  - advantage of using Gram Schmidt is BLAS type ops which map well to GPU

- How does this compare to other approaches in electronic structure eg., subspace filtering?

- Shortly will integrate and test with QE

wjb19@psu.edu

References

- NETLIB http://www.netlib.org/
- MKL http://software.intel.com/en-us/intel-mkl
- Implicitly restarted Arnoldi/Lanczos Methods for Large Scale Eigenvalue Calculations (D. Sorensen)
- Numerical Methods for Large Eigenvalue Problems (Y. Saad)
- Matrix Computations, Golub & Van Loan
- Linear Algebra Lecture Notes of Peter Arbenz (ETH)
- Mellanox/GPUdirect : http://www.mellanox.com/pdf/whitepapers/TB_GPU_Direct.pdf
- OpenMPI/CUDA details http://www.open-mpi.org/faq
- GPUDirect https://developer.nvidia.com/gpudirect
- Quantum Espresso http://www.quantum-espresso.org/
- Numerical methods for electronic structure calculations of materials (Saad et al)
- Quantum Algorithms for Complex Materials (XSEDE 2012) https://www.xsede.org/documents/234989/378230/chelikowsky_xsede.pdf

Acknowledgements

- Hewlett Packard for a Donation of 50 M2070 devices
- Jason Holmes & Penn State Research Computing http://rcc.its.psu.edu/
- TACC/Stampede http://www.tacc.utexas.edu/resources/hpc/#stampede
- XSEDE https://www.xsede.org/

wjb19@psu.edu