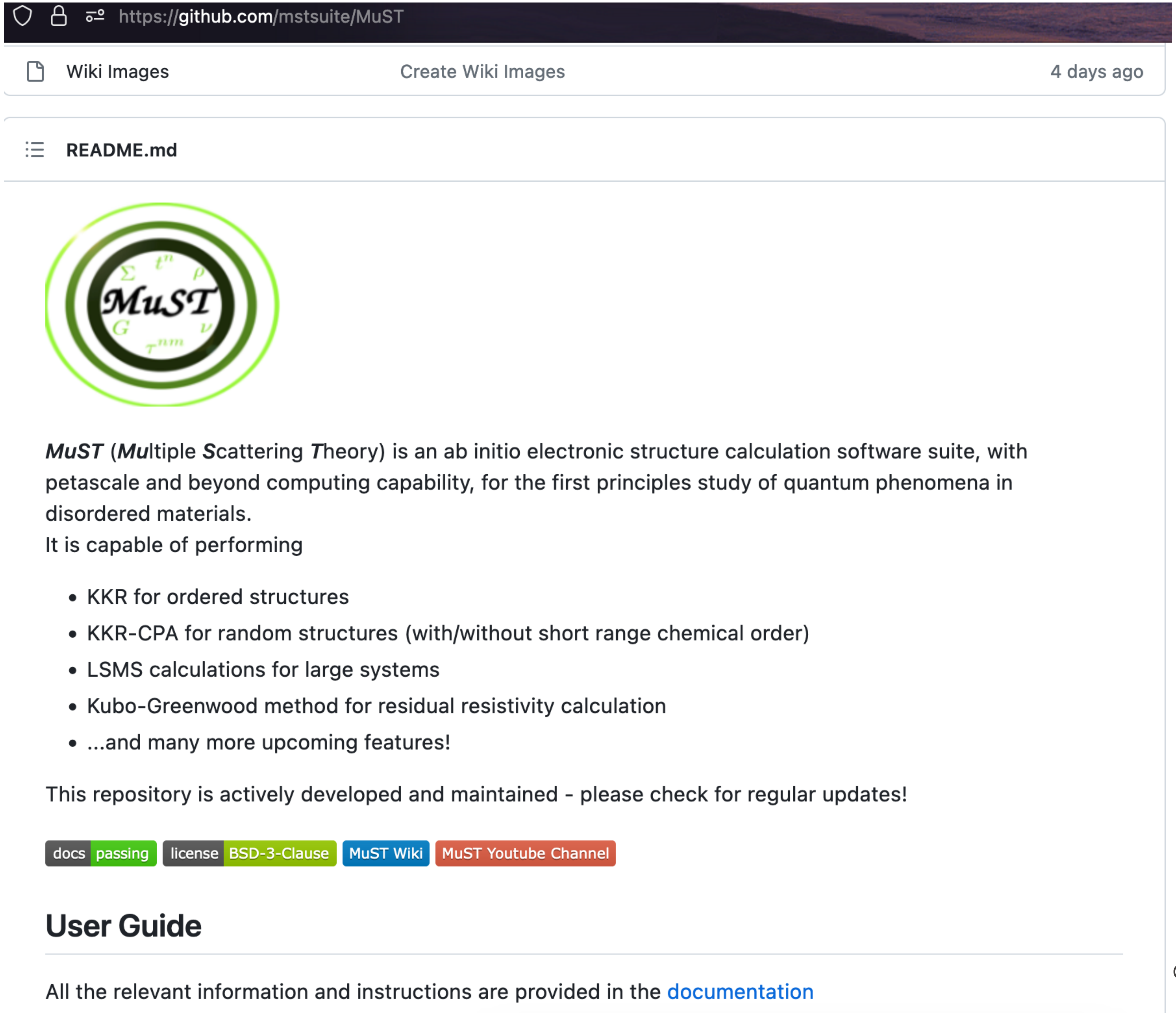
High performance GPU accelerated MuST software

Xiao Liang, PhD

Pittsburgh Supercomputing Center

April 19, 2023


https://github.com/mstsuite/MuST

Wiki Images Create Wiki Images 4 days ago

README.md

MuST (***Mu***ltiple ***S***cattering ***T***heory) is an ab initio electronic structure calculation software suite, with petascale and beyond computing capability, for the first principles study of quantum phenomena in disordered materials.
It is capable of performing

- KKR for ordered structures
- KKR-CPA for random structures (with/without short range chemical order)
- LSMS calculations for large systems
- Kubo-Greenwood method for residual resistivity calculation
- ...and many more upcoming features!

This repository is actively developed and maintained - please check for regular updates!

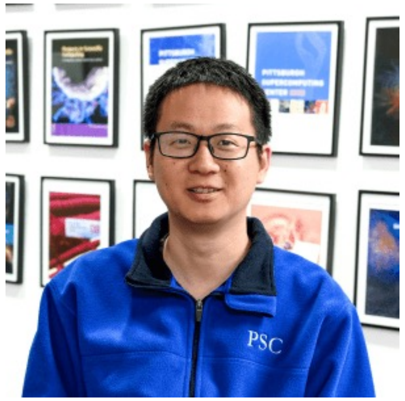
docs passing license BSD-3-Clause MuST Wiki MuST Youtube Channel

User Guide

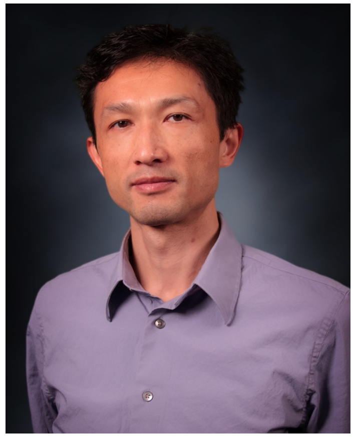
All the relevant information and instructions are provided in the documentation

PSC

TACC


Xiao Liang

Yang Wang


Hang Liu

Ed Hanna

Derek Simmel

This work was supported by the National Science Foundation through the OAC-2139536 Characteristic Science Applications for the Leadership Class Computing Facility award. The MuST package is the product of an open source project supported in part by NSF Office of Advanced Cyberinfrastructure and the Division of Materials Research within the NSF Directorate of Mathematical and Physical Sciences under award number 1931367, 1931445, and 193152

Outline:

- DFT basics
- Green’s function approach (KKR method)
- Locally Self-consistent Multiple Scattering (LSMS) method
- Computational challenge: matrix inverse
- GPU acceleration: code review
- Benchmark systems and results
- Run GPU accelerated MuST on Bridges-2 @ PSC
- Outlook

Outline:

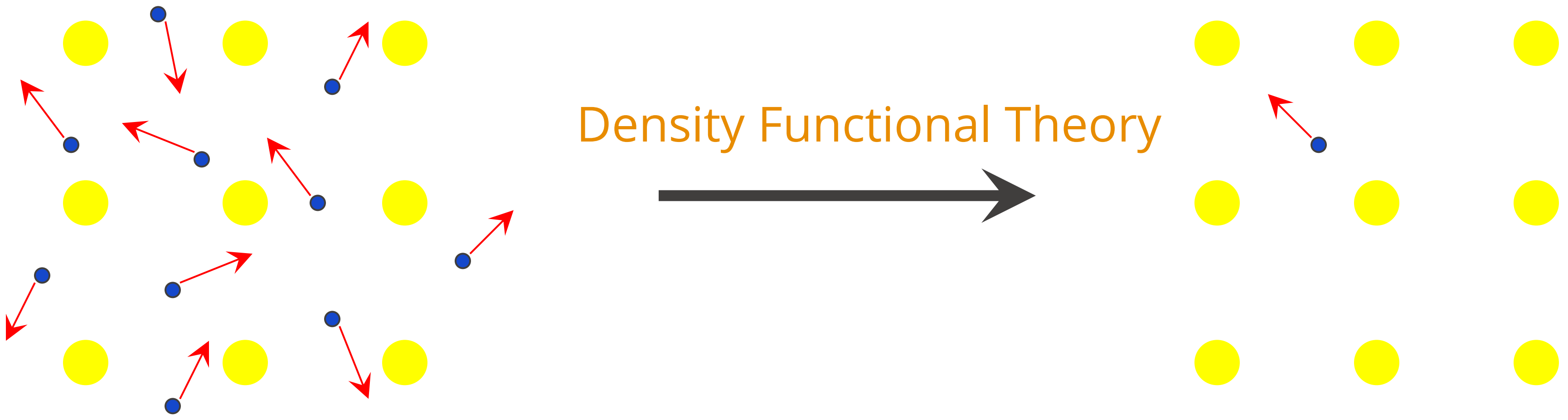
- DFT basics
- Green’s function approach (KKR method)
- Locally Self-consistent Multiple Scattering (LSMS) method
- Computational challenge: matrix inverse
- GPU acceleration: code review
- Benchmark systems and results
- Run GPU accelerated MuST on Bridges-2 @ PSC
- Outlook

Quantum Mechanical Approach to Solid State Materials

Electron: ● Nucleus: ●

Many-electron problem

One-electron problem



electron-electron interaction

electron-nucleus interaction

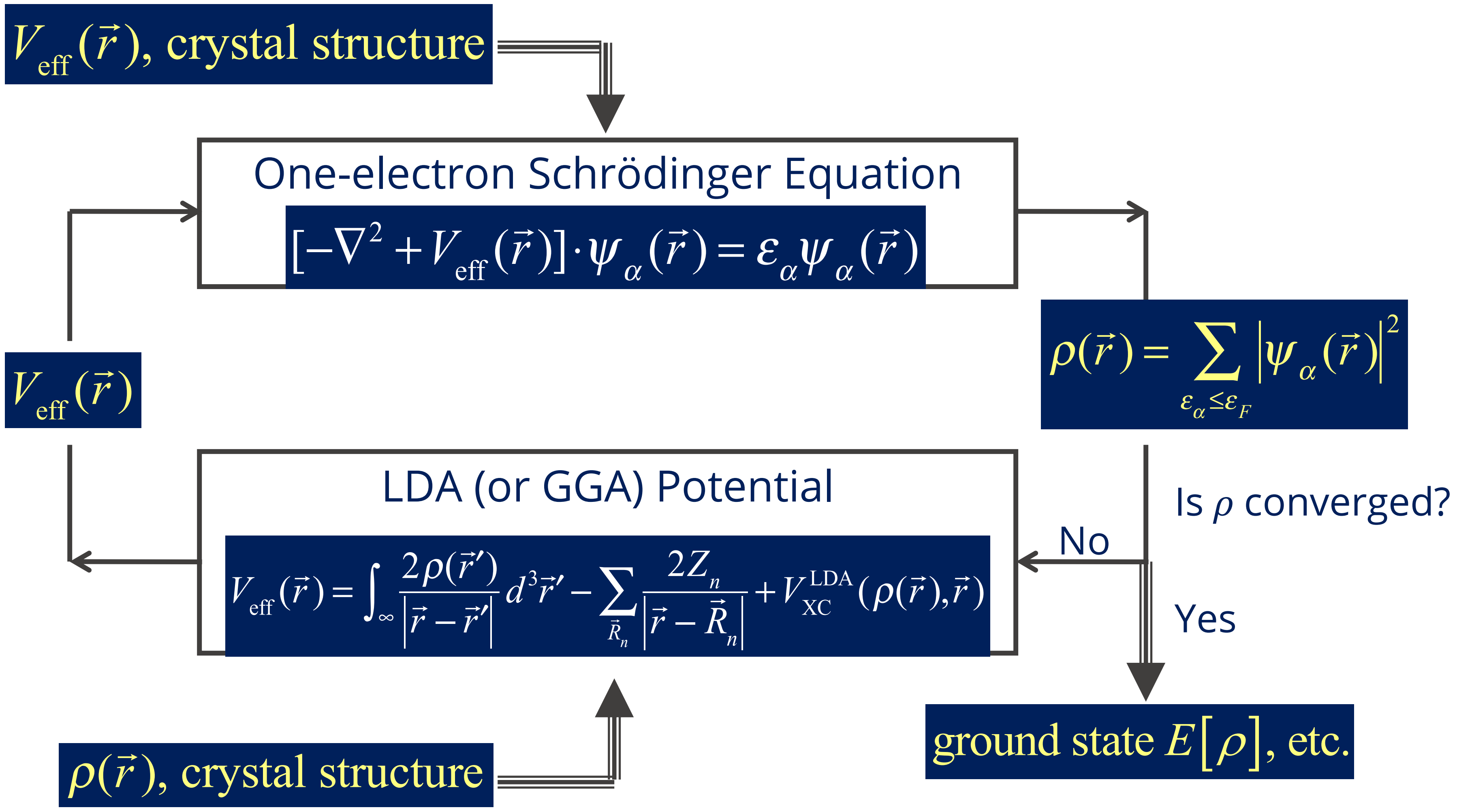
many-electron Schrödinger equation

non-interacting electrons move in an effective potential: $V_{\text{eff}}[\rho]$

one-electron Schrödinger equation

$$\left(-\frac{\hbar^2}{2m_e}\nabla^2 + e^2\int_\infty \frac{\rho(\vec{r}\,')}{|\vec{r}-\vec{r}\,'|}d^3\vec{r}\,' - e^2\sum_{\vec{R}_n}\frac{Z_n}{|\vec{r}-\vec{R}_n|} + V_{\text{xc}}[\rho]\right)\Psi_\alpha(\vec{r}) = \varepsilon_\alpha\Psi_\alpha(\vec{r})$$

The Self-consistent Process in an *Ab Initio* Electronic Structure Calculation



$V_{\text{eff}}(\vec{r})$, crystal structure

$$[-\nabla^2 + V_{\text{eff}}(\vec{r})]\cdot\psi_\alpha(\vec{r}) = \varepsilon_\alpha \psi_\alpha(\vec{r})$$

$$\rho(\vec{r}) = \sum_{\varepsilon_\alpha \le \varepsilon_F} \left|\psi_\alpha(\vec{r})\right|^2$$

$V_{\text{eff}}(\vec{r})$

$$V_{\text{eff}}(\vec{r}) = \int_\infty \frac{2\rho(\vec{r}\,')}{|\vec{r}-\vec{r}\,'|} d^3\vec{r}\,' - \sum_{\vec{R}_n} \frac{2Z_n}{|\vec{r}-\vec{R}_n|} + V_{\text{XC}}^{\text{LDA}}(\rho(\vec{r}),\vec{r})$$

ground state $E[\rho]$, etc.

$\rho(\vec{r})$, crystal structure

$$E[\rho] = \sum_{\varepsilon_\alpha \le \varepsilon_F} \varepsilon_\alpha - \int_\infty \frac{\rho(\vec{r})\rho(\vec{r}\,')}{|\vec{r}-\vec{r}\,'|} d^3\vec{r} d^3\vec{r}\,' - \int_\infty V_{\text{XC}}^{\text{LDA}}(\rho(\vec{r}),\vec{r})\rho(\vec{r}) d^3\vec{r} + E_{\text{XC}}^{\text{LDA}}[\rho]$$

Outline:

- DFT basics
- Green’s function approach (KKR method)
- Locally Self-consistent Multiple Scattering (LSMS) method
- Computational challenge: matrix inverse
- GPU acceleration: code review
- Benchmark systems and results
- Run GPU accelerated MuST on Bridges-2 @ PSC
- Outlook

Single site case

	Schrodinger Eq.	Green's function
Free particle:	$\hat{H}_0\|\varphi\rangle$=E $\|\varphi\rangle$	$\hat{G}_0 = (E - \hat{H}_0)^{-1}$
Particle in one atom site potential V :	$(\hat{H}_0 + V)\|\psi\rangle$=E $\|\psi\rangle$	$\hat{G} = (E - \hat{H}_0 - V)^{-1}$

Wave-function is obtainable: $|\psi\rangle = (\mathrm{I} + \hat{G}\mathrm{V})|\varphi\rangle$ $\Rightarrow$ Charge density

$|\psi\rangle = (\mathrm{I} + \hat{G}_0\mathrm{T})|\varphi\rangle$ T: single site scattering matrix

Obtain charge density without wave-function :

Green's function definition: $$G(r, r';\ E) = \sum_n \frac{\langle r|n\rangle\langle n|r'\rangle}{E - E_n + i\eta}$$

Density is $$\rho(r) = -\frac{2}{\pi} Im \int_{-\infty}^{E_F} G(r, r; E)dE$$

Multi-site case

Green's function:

$$G(\mathbf{r}_n, \mathbf{r}_n; \epsilon) = \sum_{L,L'} Z_L^n(\mathbf{r}_n; \epsilon) \tau_{LL'}^{nn}(\epsilon) Z_{L'}^{n*}(\mathbf{r}_n; \epsilon) - \sum_L Z_L^n(\mathbf{r}_n; \epsilon) J_L^{n*}(\mathbf{r}_n; \epsilon)$$

Dyson equation: $\underline{\tau}(E) = \left[\underline{t}^{-1}(E) - \underline{g_0}(E)\right]^{-1}$

$$\tau\text{-matrix: } \underline{\tau}^{nn}(\varepsilon) = \begin{bmatrix} \underline{t}_1^{-1}(\varepsilon) & -\underline{g}_{12}(\varepsilon) & \cdots & -\underline{g}_{1N}(\varepsilon) \\ -\underline{g}_{21}(\varepsilon) & \underline{t}_2^{-1}(\varepsilon) & \cdots & -\underline{g}_{2N}(\varepsilon) \\ \vdots & \vdots & \ddots & \vdots \\ -\underline{g}_{N1}(\varepsilon) & -\underline{g}_{N2}(\varepsilon) & \cdots & \underline{t}_N^{-1}(\varepsilon) \end{bmatrix}_{nn}^{-1}$$

Matrix rank is: 1, no spin: $N(l_{max} + 1)^2$ 2, spin: $2N(l_{max} + 1)^2$

Differences (Advantages) comparing to directly solving KS Eq. :

1. No pseudopotentials required
2. No wave-function normalization and orthogonalization
3. No Hamiltonian diagonalization
4. Can be used to study random alloys (combining with CPA)

Brief summary on KKR method:

1. Solve single-site solution Z and J. single-site scattering matrix (t) and free particle propagator (g0).

2. Build KKR matrix.

3. Invert KKR matrix, obtain multiple-scattering matrix (\tau)

4. Construct Green's function with \tau, Z and J

5. Obtain charge density through Green's function

6. Obtain effective potential through charge density (LDA or GGA)

7. Go to DFT self-consistent

The Self-consistent Process in the Green function based *Ab initio* Electronic Structure Calculation

Atomic units:

$m_e = 1/2$

$\hbar = 1$

$\mu_B = e/c$

$e^2 = 2$

$V_{\text{eff}}(\mathbf{r})$, crystal structure

Green function of the Kohn-Sham Equation

$$G(\mathbf{r}_n,\mathbf{r}_n;\varepsilon) = \sum_{L,L'} Z_L^n(\mathbf{r}_n;\varepsilon)\tau_{LL'}^{nn}(\varepsilon)Z_{L'}^{n*}(\mathbf{r}_n;\varepsilon) - \sum_{L} Z_L^n(\mathbf{r}_n;\varepsilon)J_L^{n*}(\mathbf{r}_n;\varepsilon)$$

$$\rho(\mathbf{r}) = \rho_{\text{core}}(\mathbf{r}) - \frac{2}{\pi}\,\text{Im}\int_{\varepsilon_b}^{\varepsilon_F} G(\mathbf{r},\mathbf{r};z)\,dz$$

Is density converged?

No

Yes

ground state E, etc.

LDA (or GGA) Potential

$$V_{\text{eff}}(\mathbf{r}) = \int_\infty \frac{2\rho(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|}d^3\mathbf{r}' - \sum_{\mathbf{R}_n}\frac{2Z_n}{|\mathbf{r}-\mathbf{R}_n|} + V_{\text{XC}}[\rho(\mathbf{r})]$$

$V_{\text{eff}}(\mathbf{r})$ $\leftarrow$ A mixing scheme applied here

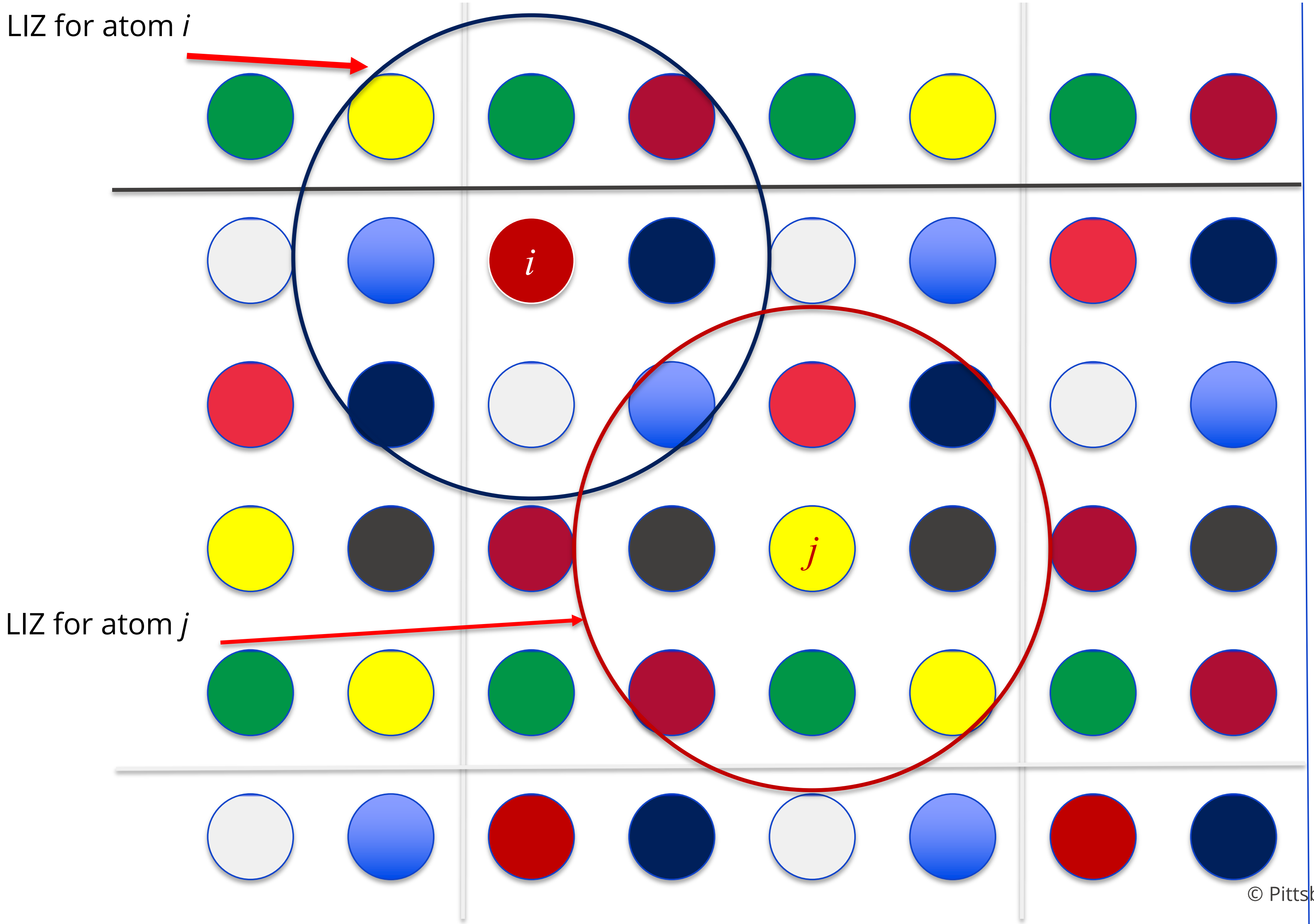
$$E[\rho] = \int_{-\infty}^{\varepsilon_F}\varepsilon\rho(\varepsilon)\,d\varepsilon - \int_\infty \frac{\rho(\mathbf{r})\rho(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|}d^3\mathbf{r}\,d^3\mathbf{r}' - \int_\infty V_{\text{XC}}(\mathbf{r})\rho(\mathbf{r})\,d^3\mathbf{r} + E_{\text{XC}}[\rho]$$

Outline:

- DFT basics
- Green’s function approach (KKR method)
- Locally Self-consistent Multiple Scattering (LSMS) method
- Computational challenge: matrix inverse
- GPU acceleration: code review
- Benchmark systems and results
- Run GPU accelerated MuST on Bridges-2 @ PSC
- Outlook

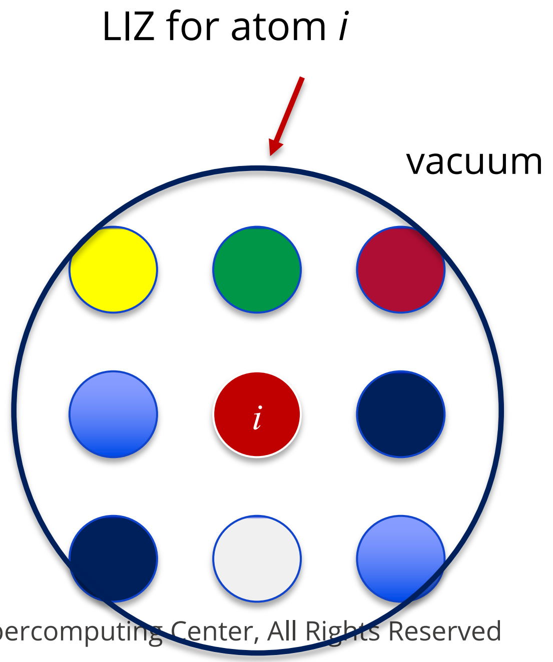
$$G(\mathbf{r}_n, \mathbf{r}_n; \epsilon) = \sum_{L,L'} Z_L^n(\mathbf{r}_n; \epsilon)\tau_{LL'}^{nn}(\epsilon) Z_{L'}^{n*}(\mathbf{r}_n; \epsilon) - \sum_{L} Z_L^n(\mathbf{r}_n; \epsilon) J_L^{n*}(\mathbf{r}_n; \epsilon)$$

Locally Self-consistent Multiple Scattering (LSMS) Method



The LIZ cluster with *M* atoms around each site is considered being embedded in vacuum. The *τ*-matrix for site *i* is given by

$$\underline{\tau}^{11}(\varepsilon) = \begin{bmatrix} \underline{t}_1^{-1}(\varepsilon) & \cdots & -\underline{g}_{1M}(\varepsilon) \\ \vdots & \ddots & \vdots \\ -\underline{g}_{M1}(\varepsilon) & \cdots & \underline{t}_M^{-1}(\varepsilon) \end{bmatrix}_{11}^{-1}$$



Outline:

- DFT basics
- Green’s function approach (KKR method)
- Locally Self-consistent Multiple Scattering (LSMS) method
- Computational challenge: matrix inverse
- GPU acceleration: code review
- Benchmark systems and results
- Run GPU accelerated MuST on Bridges-2 @ PSC
- Outlook

Matrix inverse multi-core acceleration

$$\begin{bmatrix}1 & 0 & 0\\ 3 & 1 & 0\\ 1 & \frac{3}{2} & 1\end{bmatrix}\begin{bmatrix}y_1\\ y_2\\ y_3\end{bmatrix}=\begin{bmatrix}1\\ 5\\ 10\end{bmatrix}$$

$$\mathrm{L}\vec{Y}=\vec{M}$$

$$\begin{bmatrix}1 & 1 & 1\\ 0 & -2 & -6\\ 0 & 0 & 3\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix}=\begin{bmatrix}1\\ 2\\ 6\end{bmatrix}$$

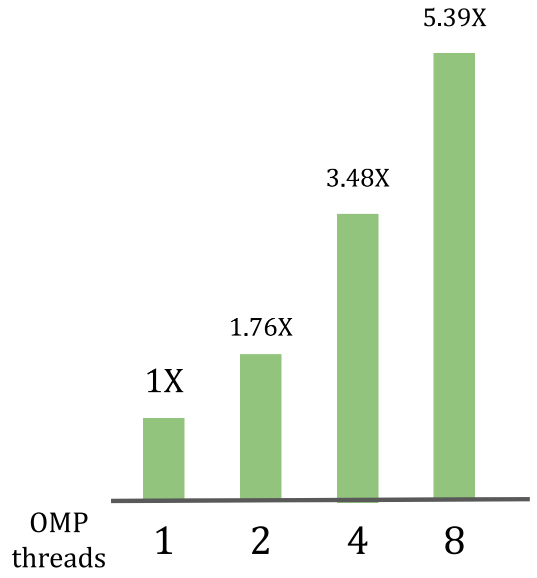
$$\mathrm{U}\vec{X}=\vec{N}$$

Solving X: $A\vec{X}=\vec{B}$ through LU decomposition: 1, $A=LU$
2, $LU\vec{X}=\vec{B}$
3, $L\vec{Y}=\vec{B}$
4, $\mathrm{U}\vec{X}=\vec{Y}$

Solving X: $\begin{bmatrix}A_{00} & A_{01}\\ A_{10} & A_{11}\end{bmatrix}\begin{bmatrix}X_{00} & X_{01}\\ X_{10} & X_{11}\end{bmatrix}=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$ is:

Solving $\begin{bmatrix}A_{00} & A_{01}\\ A_{10} & A_{11}\end{bmatrix}\begin{bmatrix}X_{00}\\ X_{10}\end{bmatrix}=\begin{bmatrix}1\\ 0\end{bmatrix}$ and $\begin{bmatrix}A_{00} & A_{01}\\ A_{10} & A_{11}\end{bmatrix}\begin{bmatrix}X_{01}\\ X_{11}\end{bmatrix}=\begin{bmatrix}0\\ 1\end{bmatrix}$

Intel MKL threads many-core acceleration ratio:



Outline:

- DFT basics
- Green’s function approach (KKR method)
- Locally Self-consistent Multiple Scattering (LSMS) method
- Computational challenge: matrix inverse
- GPU acceleration: code review
- Benchmark systems and results
- Run GPU accelerated MuST on Bridges-2 @ PSC
- Outlook

Offloading matrix inverse on GPUs

$$G(\mathbf{r}_n, \mathbf{r}_n; \epsilon) = \sum_{L,L'} Z_L^n(\mathbf{r}_n; \epsilon)\tau_{LL'}^{nn}(\epsilon)Z_{L'}^{n*}(\mathbf{r}_n; \epsilon) - \sum_{L} Z_L^n(\mathbf{r}_n; \epsilon)J_L^{n*}(\mathbf{r}_n; \epsilon)$$

Block LU on CPU: N^2

Full inverse: N^3 However much faster on GPU

Recursive Block Inverse Technique

Since the LSMS method requires only the first diagonal block of the inverse matrix, we perform block inverse recursively:

$$\left[\begin{array}{c|c} \underline{A}_{11} & \underline{A}_{12} \\ \hline \underline{A}_{21} & \underline{A}_{22} \end{array}\right]^{-1} = \left[\begin{array}{c|c} \left[\underline{A}_{11} - \underline{A}_{12}\underline{A}_{22}^{-1}\underline{A}_{21}\right]^{-1} & * \\ \hline * & * \end{array}\right]$$

$$\Downarrow$$

$$\left[\underline{A}_{11} - \underline{A}_{12}\underline{A}_{22}^{-1}\underline{A}_{21}\right]^{-1} = \left[\begin{array}{c|c} \underline{B}_{11} & \underline{B}_{12} \\ \hline \underline{B}_{21} & \underline{B}_{22} \end{array}\right]^{-1} = \left[\begin{array}{c|c} \left[\underline{B}_{11} - \underline{B}_{12}\underline{B}_{22}^{-1}\underline{B}_{21}\right]^{-1} & * \\ \hline * & * \end{array}\right]$$

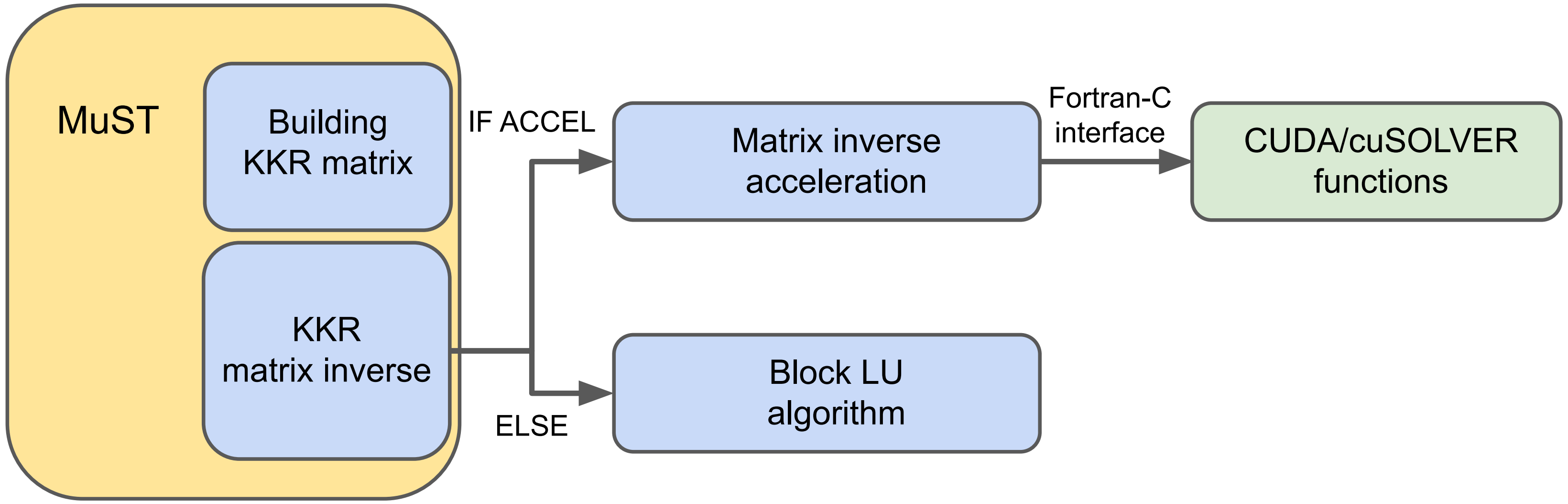
$$\Downarrow$$

$$\vdots$$

The block size is a performance tuning parameter

Performance of LSMS is dominated by double complex matrix matrix multiplication (**zgemm** in BLAS)

Offloading matrix inverse on GPUs (Code structure)



Fortran:

```
subroutine a(x1)
double complex:: x1(*)
call matrix_inverse_cuda(x1)
end subroutine a
```

CUDA in C:

```
extern “C”
void matrix_inverse_cuda_(double complex *x1)
{
    cudaMalloc;
    cudaMemcpy(… , HostToDevice);
    cusolverDnZgetrf; #LU decomposition
    cusolverDnZgetrs; #solve linear equation
    cudaMemcpy(… , DeviceToHost);
    cudaFree;
}
```

Outline:

- DFT basics
- Green’s function approach (KKR method)
- Locally Self-consistent Multiple Scattering (LSMS) method
- Computational challenge: matrix inverse
- GPU acceleration: code review
- Benchmark systems and results
- Run GPU accelerated MuST on Bridges-2 @ PSC
- Outlook

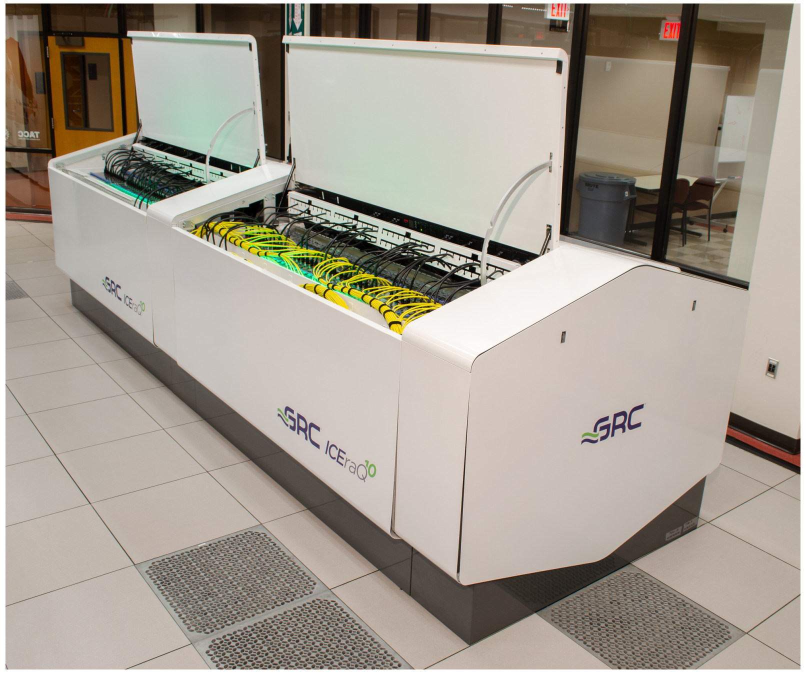
GPU systems

GPUs support double precision (FP64) : NVIDIA V100, A100; AMD Instinct …

1. Lonestar6 @ TACC

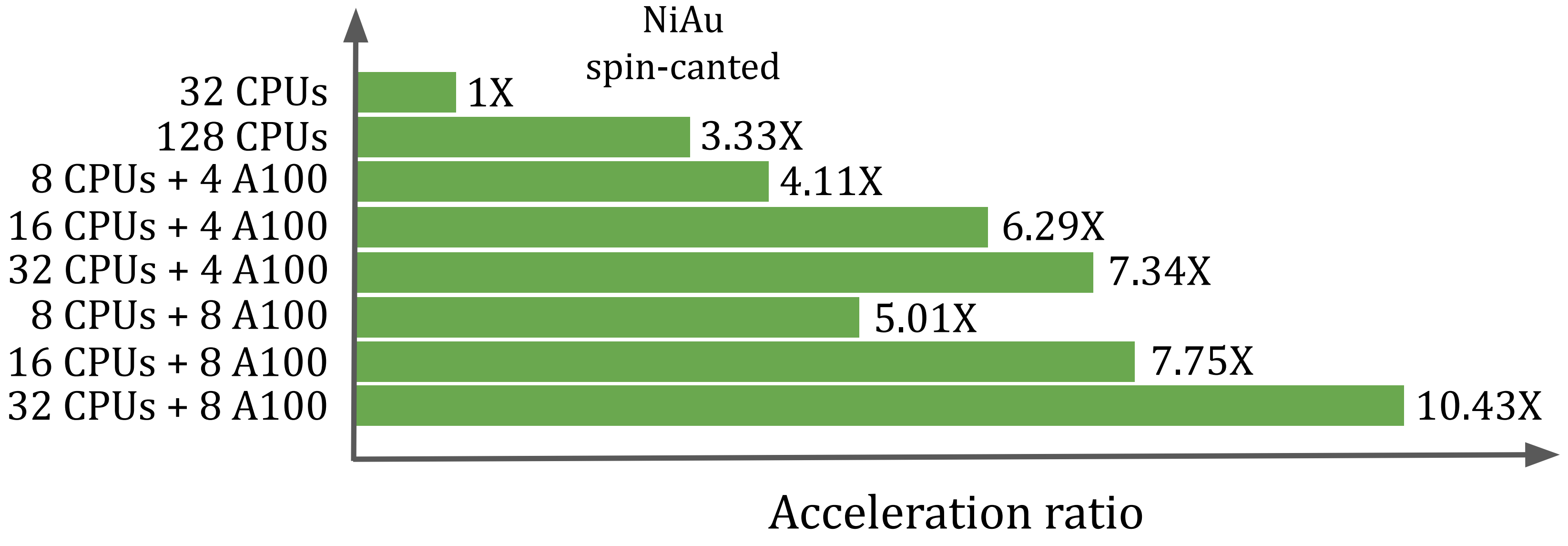
2. Bridges2 @ PSC

3. BIL cluster @ PSC



Acceleration ratio on one computing node

Testing Case: NiAu alloy totally 64 atoms; KKR matrix size: 12450 x 12450



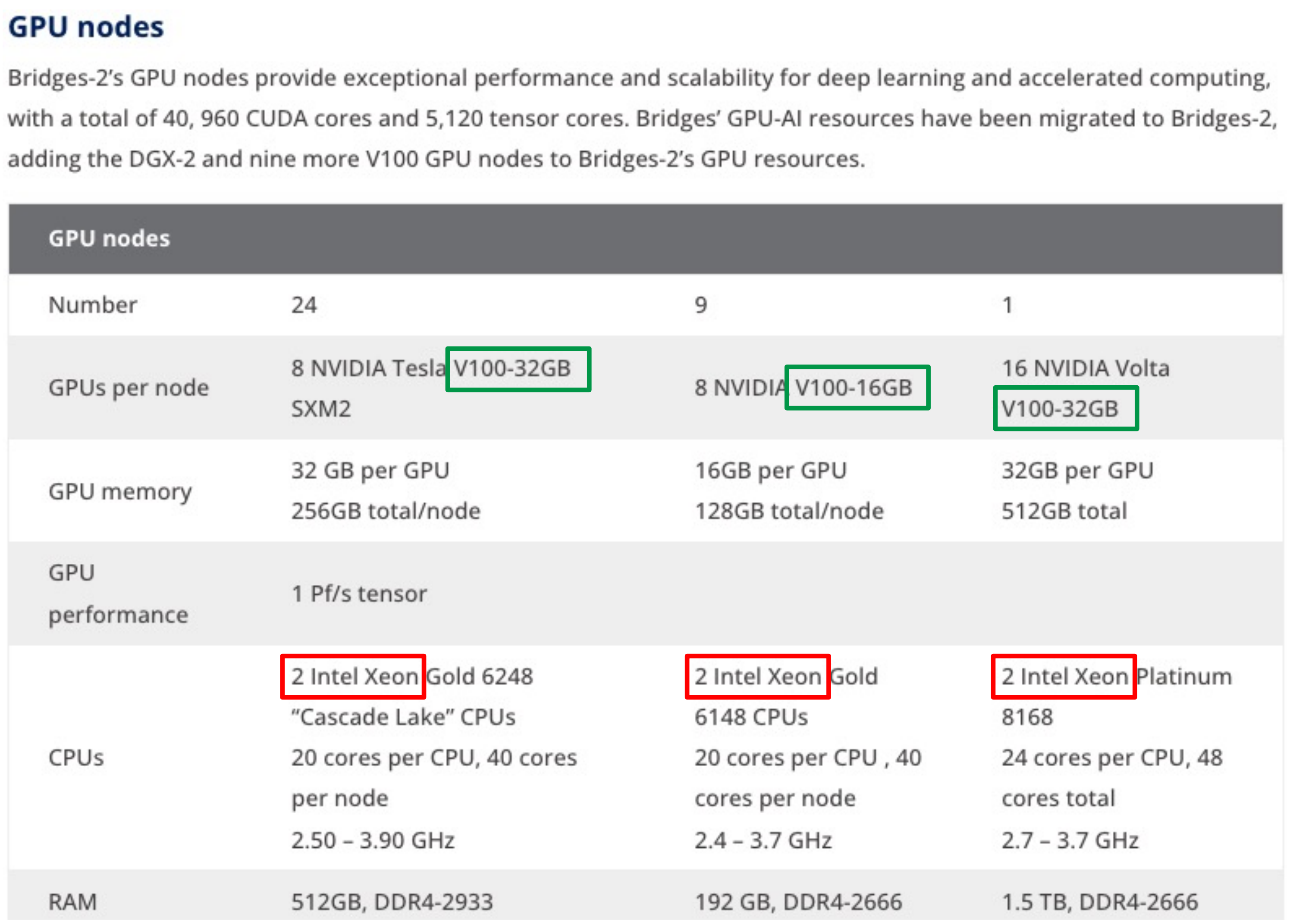
	GPU number/MPI rank number	8	16	32
NiAu spin-canted (on PSC-BIL 8-A100-80G)	2	5528(0.101)	4543(0.108)	
	4	4618(0.104)	3022(0.117)	2589(0.134)
	8	3790(0.194)	2452(0.129)	1821(0.148)

MPI rank number (=CPU core number)	
32	19002
64	11069
128	5695

Outline:

- DFT basics
- Green’s function approach (KKR method)
- Locally Self-consistent Multiple Scattering (LSMS) method
- Computational challenge: matrix inverse
- GPU acceleration: code review
- Benchmark systems and results
- Run GPU accelerated MuST on Bridges-2 @ PSC
- Outlook

Setting up environments on Bridges-2



GPU nodes

Bridges-2's GPU nodes provide exceptional performance and scalability for deep learning and accelerated computing, with a total of 40, 960 CUDA cores and 5,120 tensor cores. Bridges' GPU-AI resources have been migrated to Bridges-2, adding the DGX-2 and nine more V100 GPU nodes to Bridges-2's GPU resources.

GPU nodes			
Number	24	9	1
GPUs per node	8 NVIDIA Tesla V100-32GB SXM2	8 NVIDIA V100-16GB	16 NVIDIA Volta V100-32GB
GPU memory	32 GB per GPU 256GB total/node	16GB per GPU 128GB total/node	32GB per GPU 512GB total
GPU performance	1 Pf/s tensor		
CPUs	2 Intel Xeon Gold 6248 "Cascade Lake" CPUs 20 cores per CPU, 40 cores per node 2.50 – 3.90 GHz	2 Intel Xeon Gold 6148 CPUs 20 cores per CPU , 40 cores per node 2.4 – 3.7 GHz	2 Intel Xeon Platinum 8168 24 cores per CPU, 48 cores total 2.7 – 3.7 GHz
RAM	512GB, DDR4-2933	192 GB, DDR4-2666	1.5 TB, DDR4-2666

Setting up environments:

1. MPI:
module load openmpi/4.1.1-gcc8.3.1

2. CUDA runtime:
module load cuda/11.7.1

3. Intel MKL:
module load mkl/2020.4.304

Download, compile and run

1. Download the latest code:

“git clone https://github.com/mstsuite/MuST.git”

2. Compile:

demo architecture file at :
“architecture/Bridges2-linux-gnu-mkl-cuda”

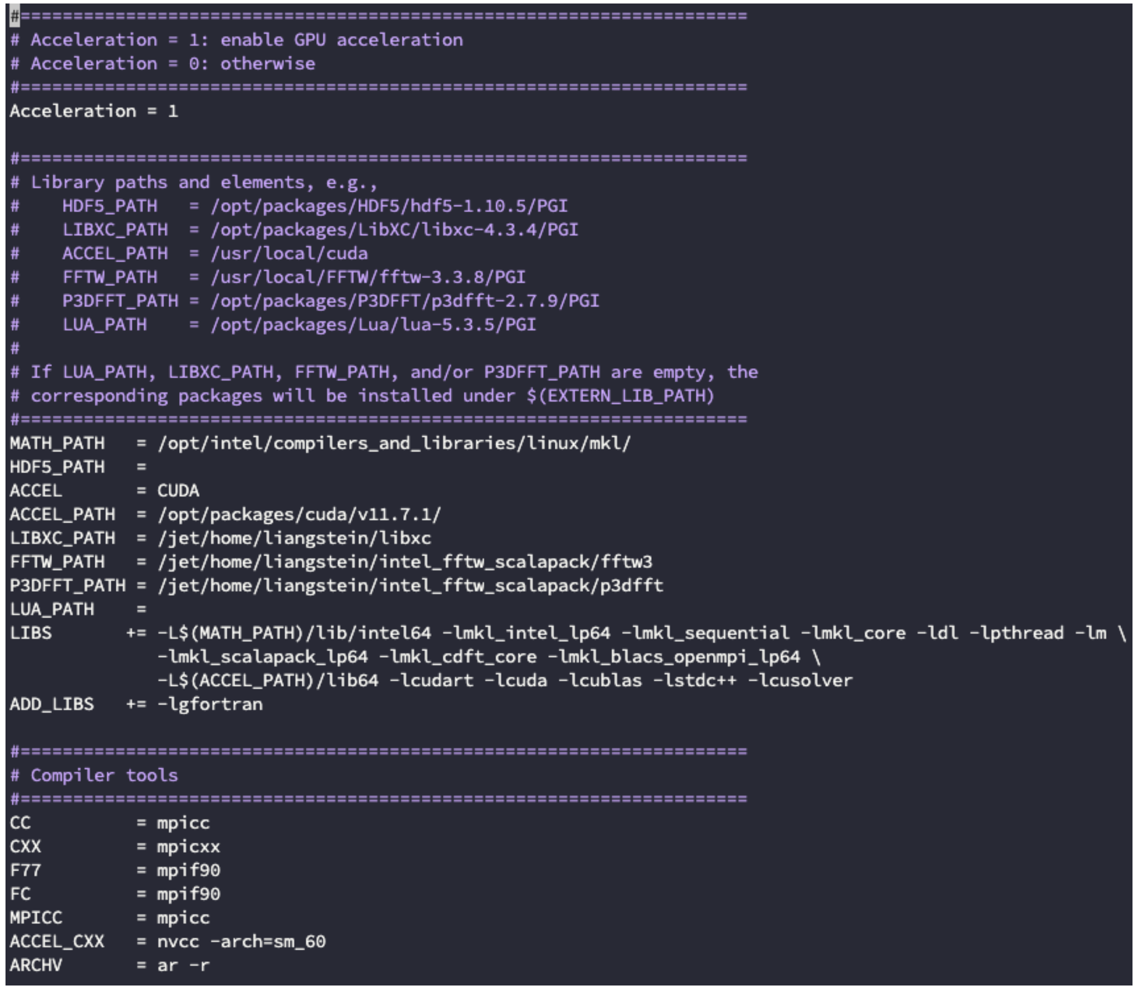
Download, compile and run

1. Download the latest code:

“git clone https://github.com/mstsuite/MuST.git”

2. Compile:

demo architecture file at :
“architecture/Bridges2-linux-gnu-mkl-cuda”



```
#======================================================================
# Acceleration = 1: enable GPU acceleration
# Acceleration = 0: otherwise
#======================================================================
Acceleration = 1

#======================================================================
# Library paths and elements, e.g.,
#     HDF5_PATH    = /opt/packages/HDF5/hdf5-1.10.5/PGI
#     LIBXC_PATH   = /opt/packages/LibXC/libxc-4.3.4/PGI
#     ACCEL_PATH   = /usr/local/cuda
#     FFTW_PATH    = /usr/local/FFTW/fftw-3.3.8/PGI
#     P3DFFT_PATH = /opt/packages/P3DFFT/p3dfft-2.7.9/PGI
#     LUA_PATH     = /opt/packages/Lua/lua-5.3.5/PGI
#
# If LUA_PATH, LIBXC_PATH, FFTW_PATH, and/or P3DFFT_PATH are empty, the
# corresponding packages will be installed under $(EXTERN_LIB_PATH)
#======================================================================
MATH_PATH    = /opt/intel/compilers_and_libraries/linux/mkl/
HDF5_PATH    =
ACCEL        = CUDA
ACCEL_PATH   = /opt/packages/cuda/v11.7.1/
LIBXC_PATH   = /jet/home/liangstein/libxc
FFTW_PATH    = /jet/home/liangstein/intel_fftw_scalapack/fftw3
P3DFFT_PATH = /jet/home/liangstein/intel_fftw_scalapack/p3dfft
LUA_PATH     =
LIBS        += -L$(MATH_PATH)/lib/intel64 -lmkl_intel_lp64 -lmkl_sequential -lmkl_core -ldl -lpthread -lm \
               -lmkl_scalapack_lp64 -lmkl_cdft_core -lmkl_blacs_openmpi_lp64 \
               -L$(ACCEL_PATH)/lib64 -lcudart -lcuda -lcublas -lstdc++ -lcusolver
ADD_LIBS    += -lgfortran

#======================================================================
# Compiler tools
#======================================================================
CC           = mpicc
CXX          = mpicxx
F77          = mpif90
FC           = mpif90
MPICC        = mpicc
ACCEL_CXX    = nvcc -arch=sm_60
ARCHV        = ar -r
```

Download, compile and run

3. A demo for GPU acceleration at location:

“Benchmark/CoCrFeMnNi/MT/u56_CUDA”

Submission example:

```
“sbatch -p GPU -t 48:0:0 -n 40 --gpus=v100-32:8 bash_script.sh”
```

In bash_script.sh:

```
#!/bin/bash
mpirun -n 40 ~/bind_MPI_to_GPU.sh ~/mst2_cuda < i_mst
```

Outline:

- DFT basics
- Green’s function approach (KKR method)
- Locally Self-consistent Multiple Scattering (LSMS) method
- Computational challenge: matrix inverse
- GPU acceleration: code review
- Benchmark systems and results
- Run GPU accelerated MuST on Bridges-2 @ PSC
- Outlook

Outlook

1. Build KKR matrix on GPU, reduce data transfer time
2. Multi-GPU matrix inverse, enable larger unit cell size
3. Other types of acceleration cards
4. Full potential calculation speed up

Thanks for your attention