Preparing WL-LSMS for First Principles Thermodynamics Calculations on Accelerator and Multicore Architectures

OLCF
Oak Ridge Leadership Computing Facility

Don Nicholson

Oak Ridge National Laboratory

Markus Eisenbach

Oak Ridge National Laboratory

Motivation

- Density Functional Calculations have proven to be a useful tool to study the ground state of many materials.

- For finite temperatures the situation is less ideal; one is often forced to rely on model calculation with parameters either fitted to first principles calculations or experimental results.

- Fitting to models is especially unsatisfactory in inhomogeneous systems, nanoparticles or other systems where the model parameters could vary significantly from one site to another.

Solution:
Combine First Principles calculations with statistical mechanics methods

Team

Oak Ridge National Laboratory
Markus Eisenbach, Don Nicholson, Junqi Yin, Khorgolkhuu Odbadrakh, Ying Wai Li)

University of Tennessee
Aurelian Rusanu

Florida State University
Gregory Brown

Pittsburgh Supercomputing Center
Yang Wang

University of Georgia
David Landau, Dilina Perera

This work was sponsored in parts by the US DOE Offices of Advanced Scientific Computing Research and Basic Energy Sciences and by the Center for Nanophase Material Sciences, the Center for Defect Physics, an Energy Frontier Research Center funded by the US DOE Office of Basic Energy Sciences. This research used resources of the Oak Ridge Leadership Computing Facility at ORNL, which is supported by the US DOE, Office of Science.

Thermodynamic Observables

- **Thermodynamic observables are related to the partition function Z and free energy F**

$$Z(\beta) = \sum_{\{\xi_i\}} e^{-\beta H(\{\xi_i\})}$$
$$F(T) = -k_B T \ln Z(1/k_B T)$$

- **If we can calculate Z(β) thermodynamic observables can be calculated as logarithmic derivatives.**

Wang-Landau Method

- Conventional Monte Carlo methods calculate expectation values by sampling with a weight given by the Bolzmann distribution
- In the Wang-Landau Method we rewrite the partition function in terms of the density of states which is calculated by this algorithm

$$Z(\beta) = \sum_{\{\xi_i\}} e^{-\beta H(\{\xi_i\})} = \int g(E) e^{-\beta E} dE$$

- To derive an algorithm to estimate g(E) we note that if randomly generated states are accepted with a probability proportional to 1/g(E) each energy interval is visited with the same frequency (flat histogram)

Metropolis Method

Metropolis et al, JCP **21**, 1087 (1953)

$$Z = \int e^{-E[\mathbf{x}]/k_B T} d\mathbf{x}$$

Compute partition function and other averages with configurations that are weighted with a Boltzmann factor

Sample configuration where Boltzmann factor is large.

1. Select configuration

$$E_i = E[\mathbf{x}_i]$$

2. Modify configuration (move)

$$E_f = E[\mathbf{x}_f]$$

3. Accept move with probability

$$A_{i\to f} = \min\{1, e^{\beta(E_i - E_f)}\}$$

Wang-Landau Method

Wang and Landau, PRL **86**, 2050 (2001)

$$Z = \int W(E) e^{-E/k_B T} dE$$

If configurations are accepted with probability 1/W all energies are visited equally (flat histogram) if W(E)=g(E).

1. Begin with prior estimate, eg $W(E) = 1$

2. Propose move, accepted with probability

$$A_{i\to f} = \min\{1, W(E_i)/W(E_f)\}$$

3. If move accepted increase DOS

$$W(E_f) \to W(E_f) \times f \qquad f > 1$$

4. Iterate 2 & 3 until histogram is flat

5. Reduce $f \to f = \sqrt{f}$ and go back to 1

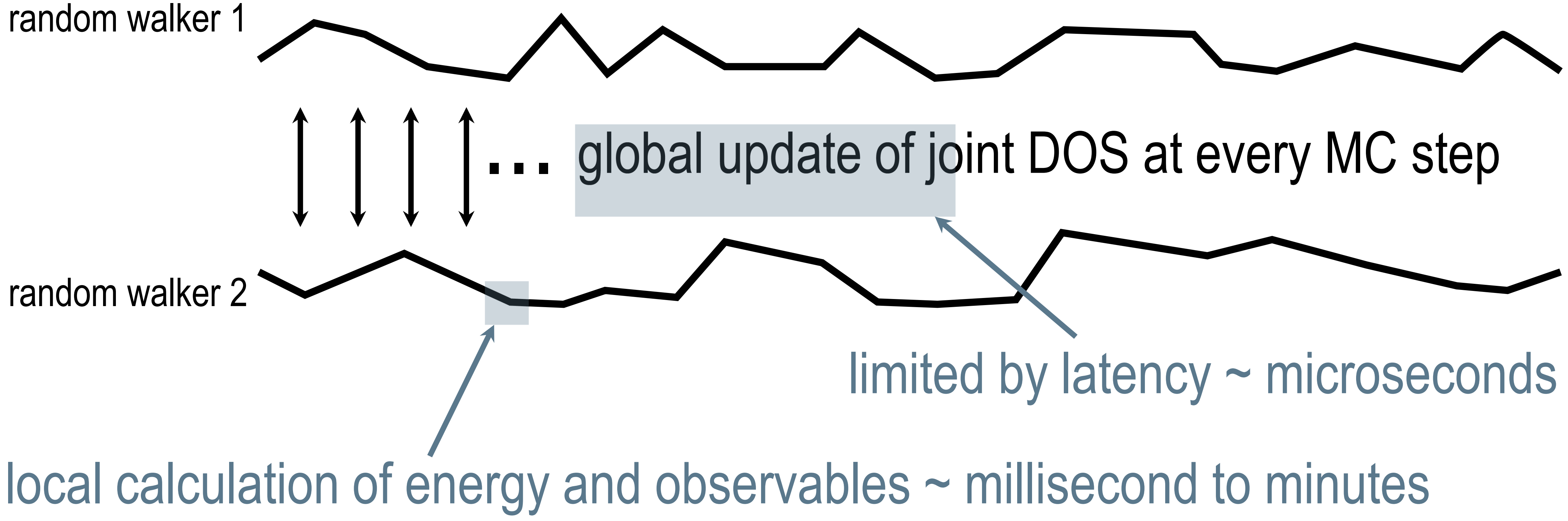
Not quite embarrassingly parallel

Metropolis MC acceptance:

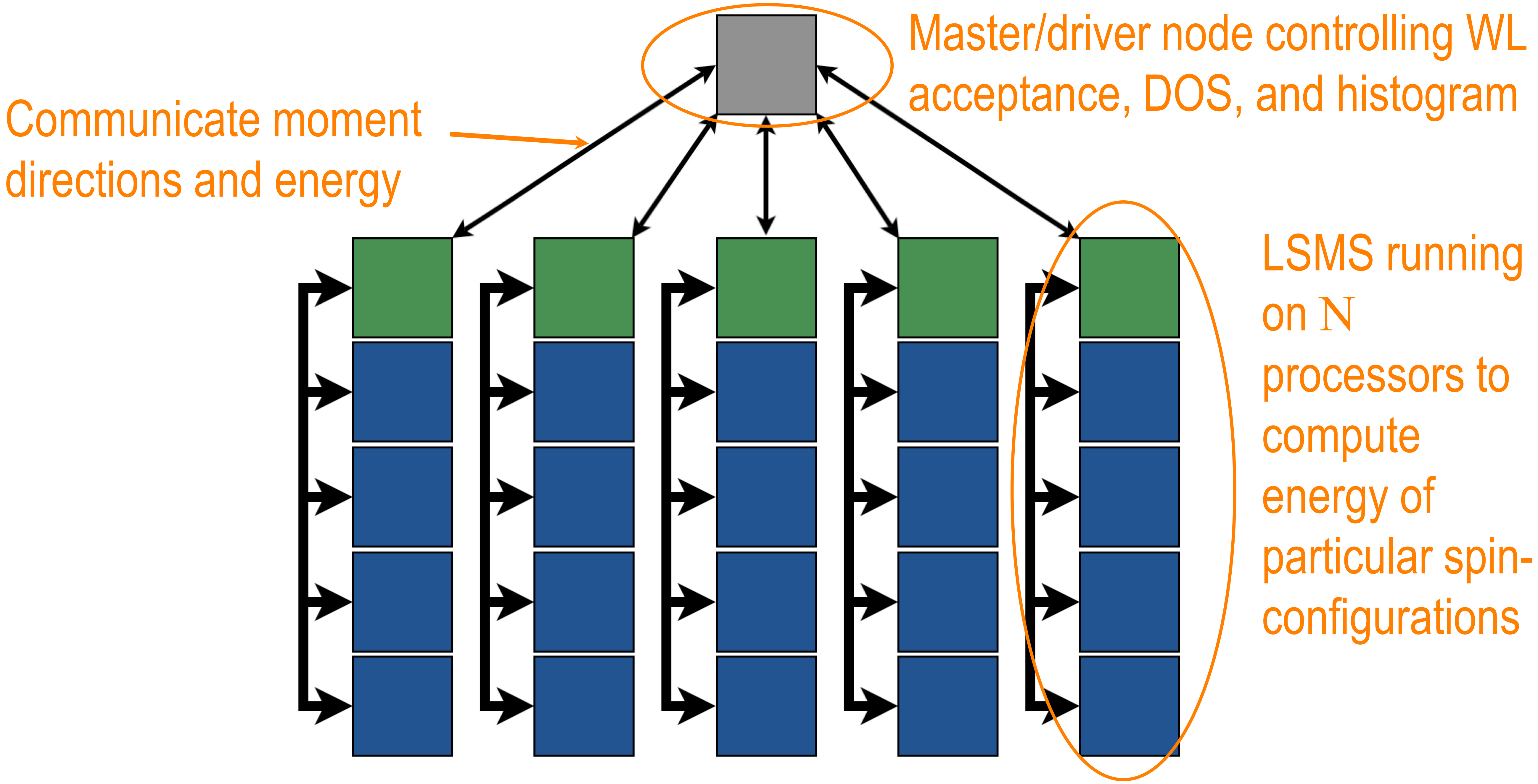
$$A_{i\to f} = \min\{1, e^{\beta(E_i - E_f)}\}$$

Wang-Landau acceptance:

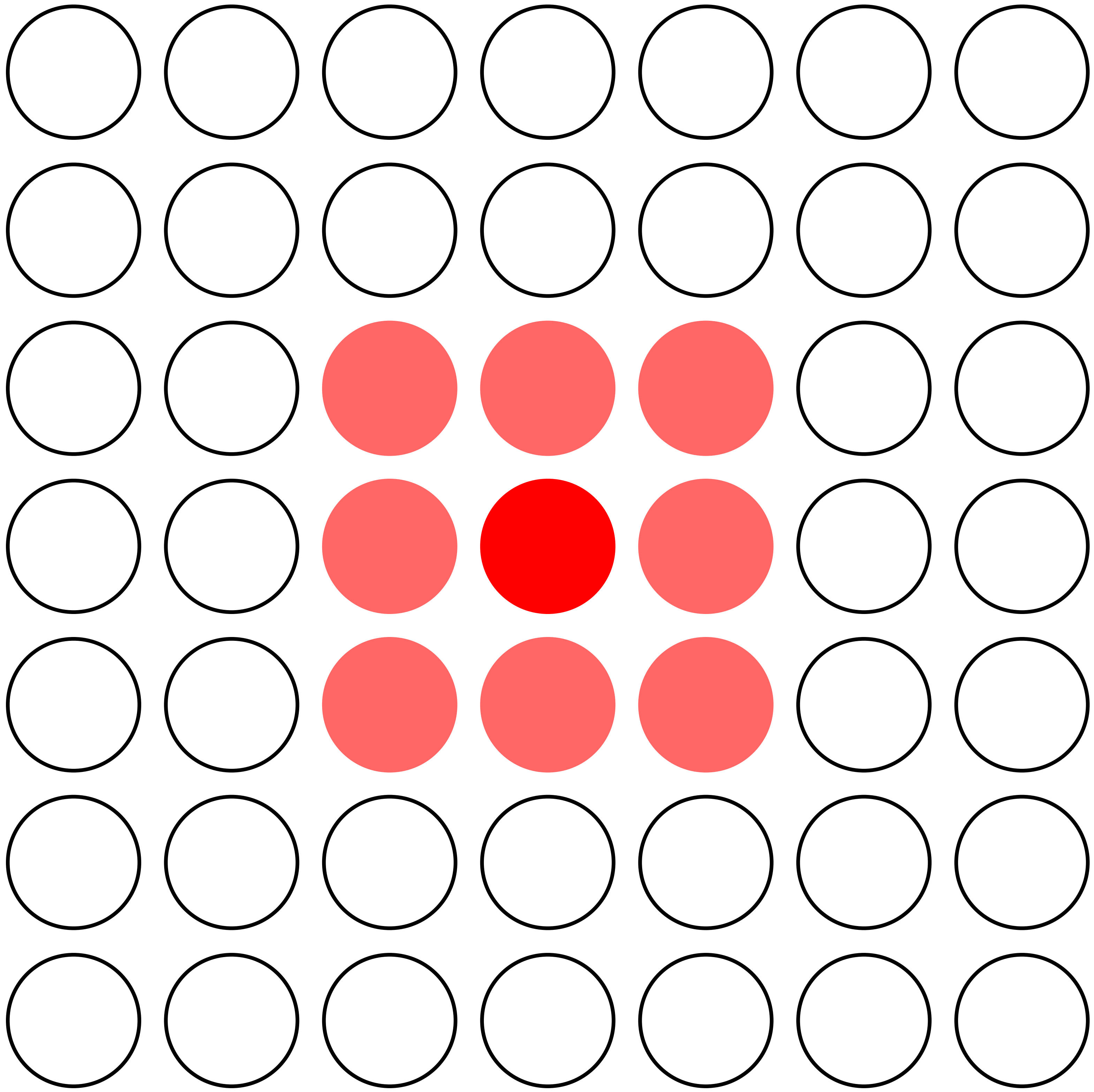
$$A_{i\to f} = \min\{1, e^{\alpha(w_\alpha(x_f) - w_\alpha(x_i))}\}$$


random walker 1
...
global update of joint DOS at every MC step
random walker 2
limited by latency ~ microseconds
local calculation of energy and observables ~ millisecond to minutes

Organization of the WL-LSMS code using a master-slave approach


Master/driver node controlling WL acceptance, DOS, and histogram
Communicate moment directions and energy
LSMS running on N processors to compute energy of particular spin-configurations

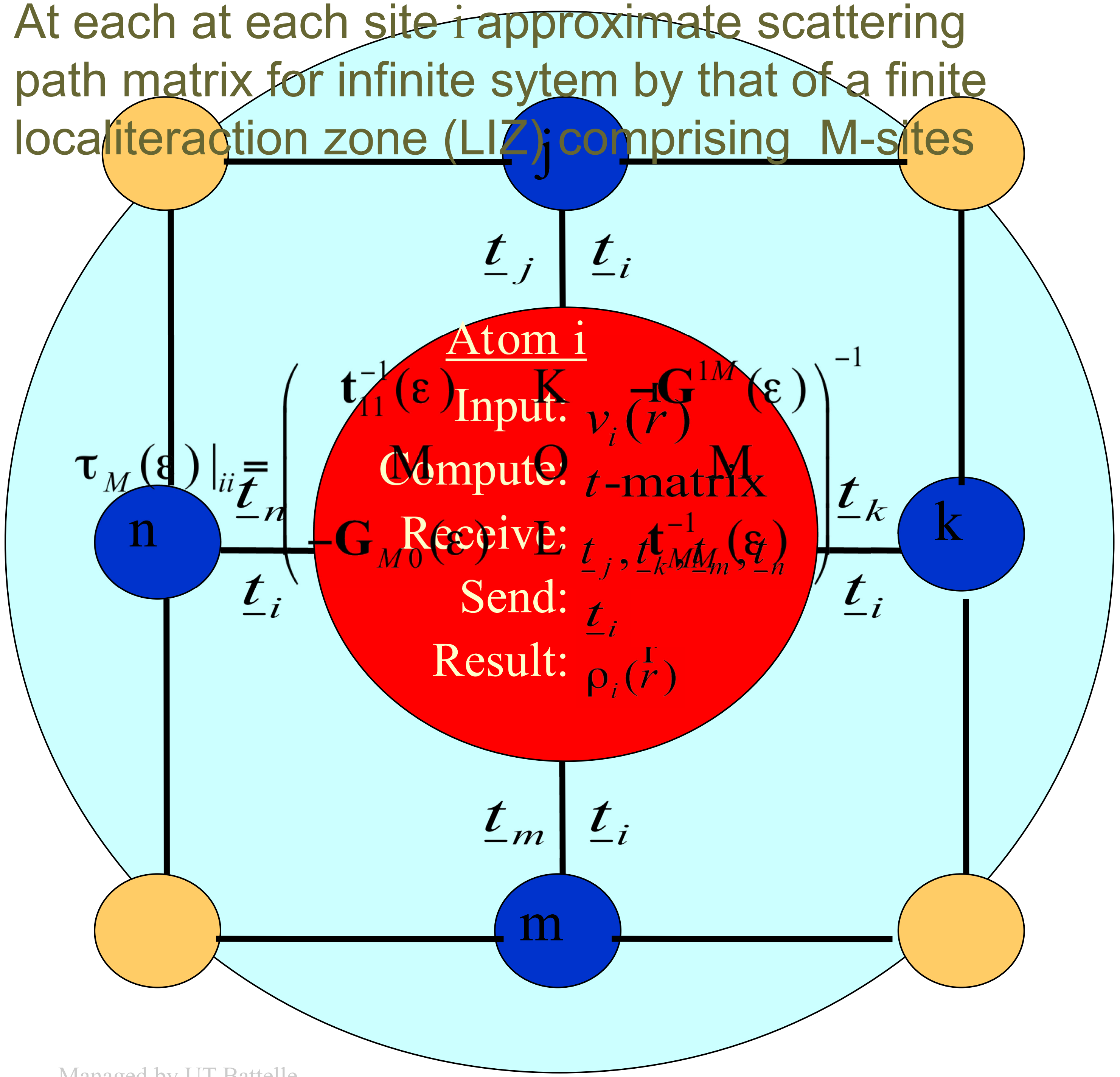
Nearsightedness and the locally self-consistent multiple scattering (LSMS) method



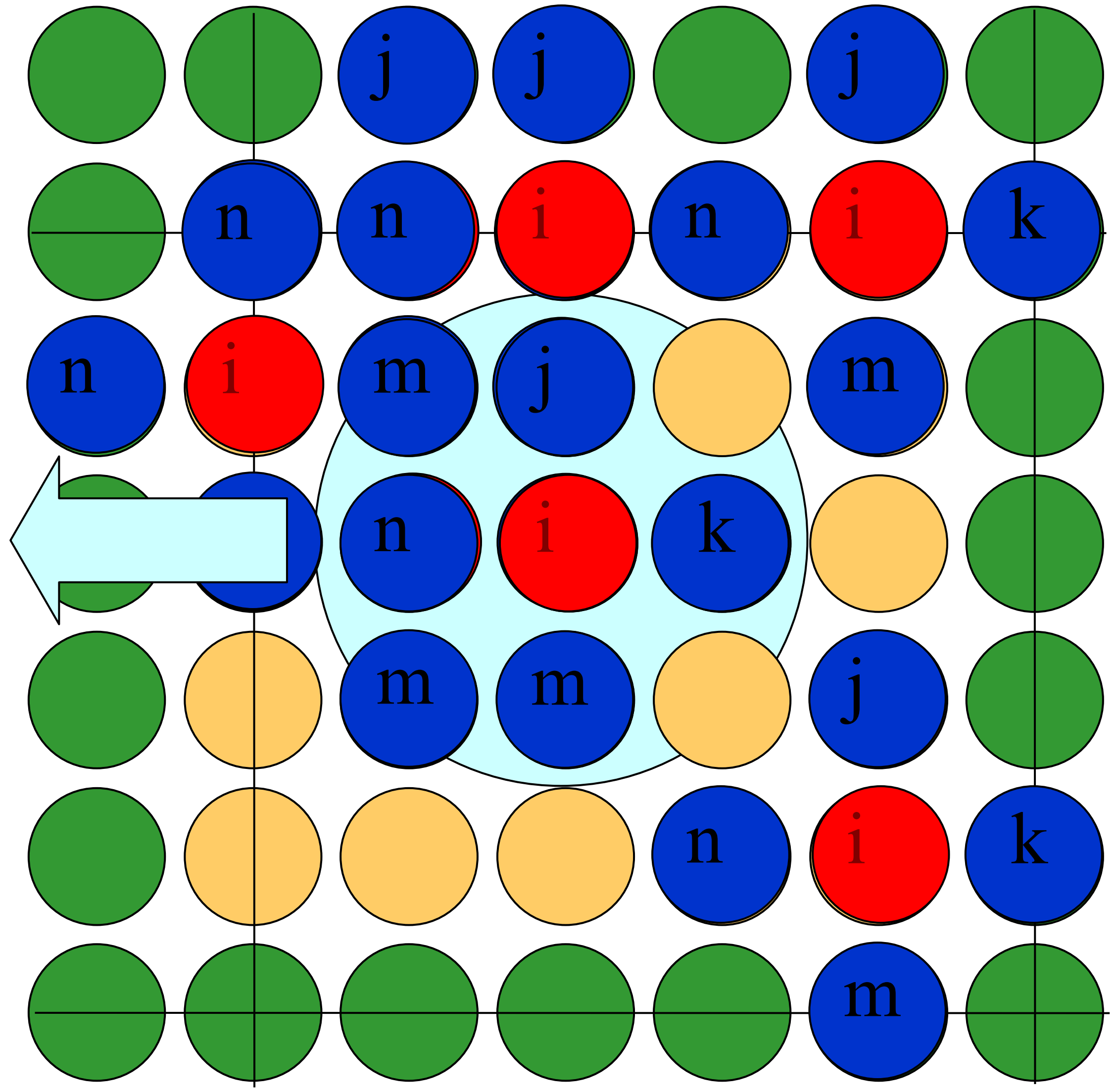
- Nearsightedness of electronic matter - Prodan & Kohn, PNAS **102**, 11635 (2005)
  - *Local electronic properties such as density depend on effective potential only at nearby points.*
- Locally self-consistent multiple scattering method - Wang et al., PRL **75**, 2867 (1995)
  - *Solve Kohn-Sham equation on a cluster of a few atomic shells around atom for which density is computed*
  - *Solve Poisson equation for entire system - long range of bare coulomb interaction*

Locally Self-consistent Multiple Scattering (LSMS) method

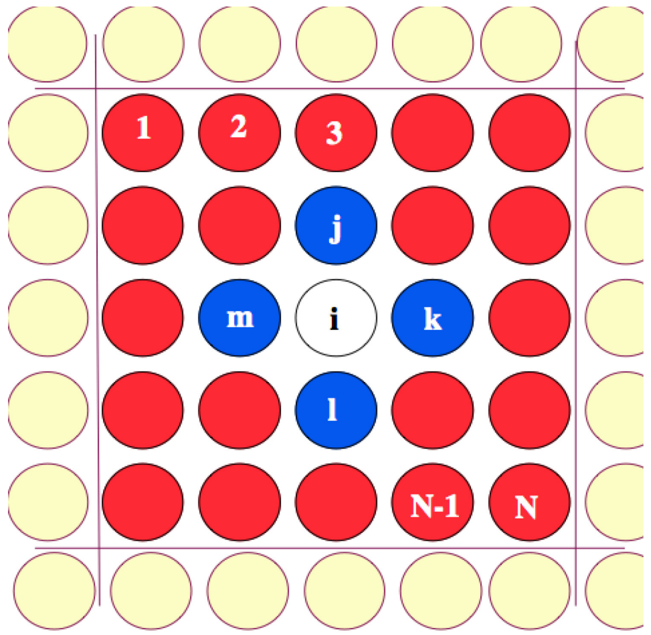
- Massively Parallel O[N] approach
- ➢ Approximate total electron density by sum of locally determined site densities
- ➢ At each at each site i approximate scattering path matrix for infinite sytem by that of a finite localiteraction zone (LIZ) comprising M-sites



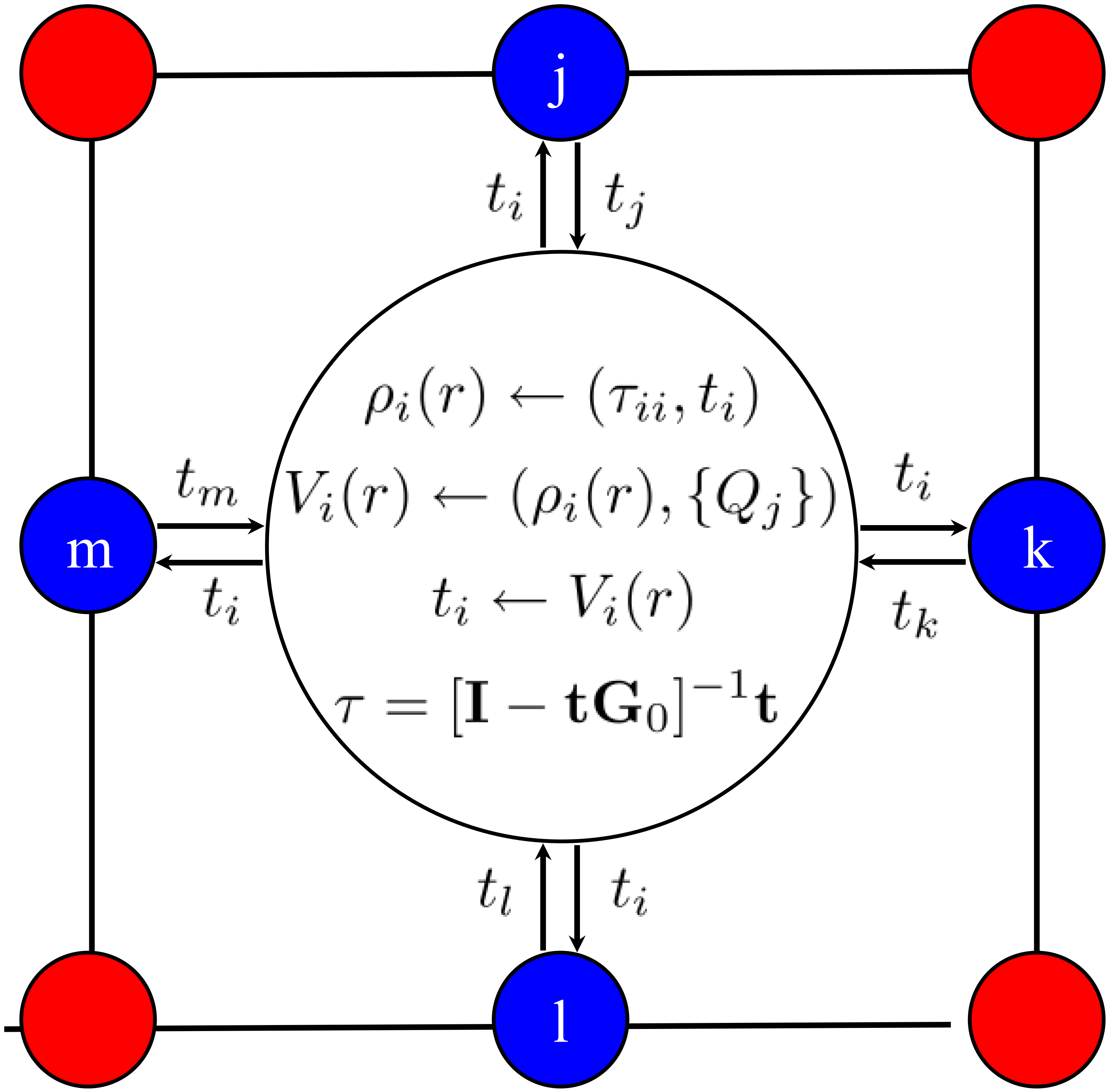
$$\tau_M(\varepsilon)|_{ii} = \begin{pmatrix} \mathbf{t}_{11}^{-1}(\varepsilon) & \mathrm{K} & -\mathbf{G}^{1M}(\varepsilon) \\ \mathrm{M} & \mathrm{O} & \mathrm{M} \\ -\mathbf{G}_{M0}(\varepsilon) & \mathrm{L} & \mathbf{t}_{MM}^{-1}(\varepsilon) \end{pmatrix}^{-1}$$



A parallel implementation and scaling of the LSMS method: perfectly scalable at high performance



- Need only block i of τ
- $\left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right)^{-1} = \left(\begin{array}{c|c} (A - BD^{-1}C)^{-1} & * \\ \hline * & * \end{array}\right)$
- Calculation dominated by ZGEMM
- Sustained performance similar to Linpack


j
m
k
l
t_i
t_j
t_m
t_i
t_i
t_k
t_l
t_i
$\rho_i(r) \leftarrow (\tau_{ii}, t_i)$
$V_i(r) \leftarrow (\rho_i(r), \{Q_j\})$
$t_i \leftarrow V_i(r)$
$\tau = [\mathbf{I} - \mathbf{t}\mathbf{G}_0]^{-1}\mathbf{t}$

Refactoring LSMS_1 to LSMS_3

- LSMS_1 assumes one atom / MPI rank
- But: This might not be ideal with current and future multicore CPU
- Highly impractical for accelerators (GPUs)
- Increase code flexibility adapt to new architectures and new physics
- Reduce the amount of code that needs to be rewritten
  (this is essentially a one person effort)
- LSMS_1:
  Fortran (mainly 77) for LSMS
  C++ for Wang-Landau

LSMS_3

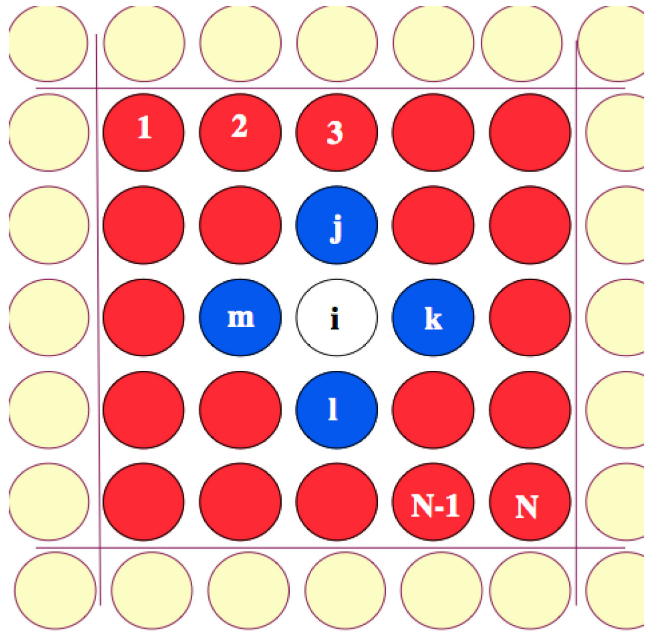
- Multiple atoms / MPI rank
- multithreading (OpenMP) in LSMS
- enable efficient use of accelerators
- New (less rigid) input file format
- Retain Wang-Landau part form LSMS_1
- LSMS_3:
  Top level routines and data structures: C++
  New communication routines: C++
  Many compute routines from LSMS_1: Fortran

```
  LSMSSystemParameters lsms;
  LSMSCommunication comm;
  CrystalParameters crystal;
  LocalTypeInfo local;

// Initialize communication and accelerator
// Read the input file

  communicateParameters(comm,lsms,crystal);

  local.setNumLocal(distributeTypes(crystal,comm));

  local.setGlobalId(comm.rank,crystal);

  buildLIZandCommLists(comm,lsms,crystal,local);

  loadPotentials(comm,lsms,crystal,local);

  setupVorpol(lsms,crystal,local,sphericalHarmonicsCoeficients);

  calculateCoreStates(comm,lsms,local);

  energyContourIntegration(comm,lsms,local);

  calculateChemPot(comm,lsms,local,eband);
```

Multiple Atoms / MPI rank

An important step to enable efficient use of multicore and accelerator architectures: Allow for more work / MPI rank!

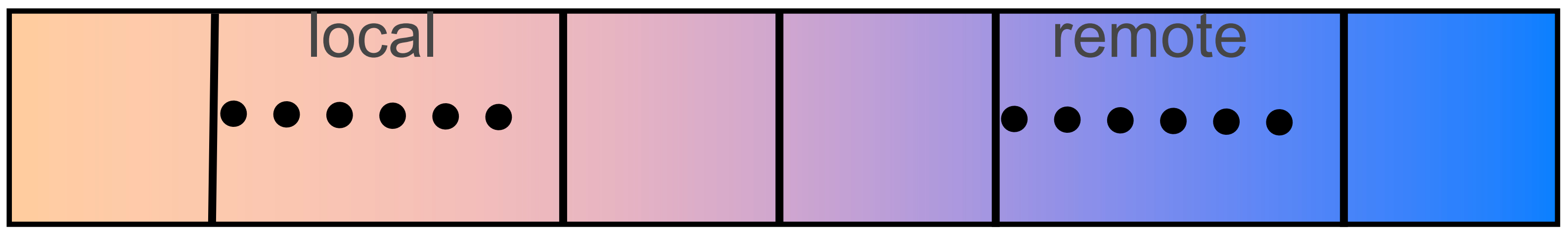
In LSMS: multiple atoms / MPI rank
    necessitates new communication pattern



```
For all atoms $i$ in crystal do
    Build the local interaction zone LIZ$i$ =
    {j|dist(x$i$,x$j$)<$r_{LIZ}$} of atom $i$
    for all atoms $j$ in LIZ$i$ do
        add atom $j$ to liat R$i$ of data to receive for
        atom I {tmatFrom}
        add atom $i$ to liat S$j$ of data to send from
        atom $j$ {tmatTo}
    end for
end for
remove duplicate entries from S$j$ and R$i$
```

Multiple Atoms / MPI rank

Matrix<Complex> tmatStore;



t matrices needed for building the local tau matrices

Building the tau matrices:

(1) Prepost receives for remote t matrices
(2) Loop over all local atom (OpenMP)
    calculate local t matrices
(3) Send local t matrices
(4) wait for completion of communication

```
expectTmatCommunication(comm,local);          (e.g. MPI_Irecv)

for(int i=0; i<local.atom.size(); i++)
  calculateSingleScattererSolution(lsms,local.atom[i],vr[i],energy,prel,pnrel, solution[i]);

sendTmats(comm,local);                        (e.g. MPI_Isend)
finalizeTmatCommunication(comm);              (e.g. MPI_Wait)
```

Calculating the tau matrix

```
for(int i=0; i<local.num_local; i++)
calculateTauMatrix(lsms,local,local.atom[i],energy,prel,tau_ii);
```

(1) For all local atoms (possibility for multithreading)
(a) build m matrix (m=I-tG) (multithreading or accelerator)
(b) invert m matrix (multithreading or accelerator)
(c)

$$\tau = \left[I - tG_0 \right]^{-1}$$

m has rank k * #LIZ and can be broken in k * k blocks m_{ij}

$$m_{ij} = I\delta_{ij} - t_i G_0^{ij}$$

only the diagonal block of the inverse corresponding to site *i=0* is needed

$$\tau_{00} = \left[I - tG_0 \right]_{00}^{-1}$$

Calculating G₀

R_{ij} describes the geometry of the system

G_0^{ij} blocks can be calculated independently: (L={l,m}), E complex

$$G^{ij}_{0,LL'}(E) = 4\pi i^{l-l'} \sum_{L''} C^{L}_{L'L''} D^{ij}_{L''}(E)$$

$$D^{ij}_{L}(E) = -i^{l+1}\sqrt{E}h_l(\sqrt{E}R_{ij})Y^*_L(\hat{R_{ij}})$$

$$Y_L = \sqrt{\frac{(2l+1)(l-|m|)!}{4\pi(l+|m|)!}}P_L(\cos\theta)e^{im\phi}$$

using Clebsch-Gordan coefficients $C^{L}_{L'L''}$,
spherical Hankel functions h_l,
and spherical harmonics Y_L

Note that all $G^{ij}_{0,LL'}$ can be calculated independently - high parallelism

$$R_{ij} = \sqrt{{R^x_{ij}}^2 + {R^y_{ij}}^2 + {R^z_{ij}}^2} \qquad \cos\theta = \frac{R^z_{ij}}{R_{ij}} \qquad \cos\phi = \frac{R^x_{ij}}{\sqrt{{R^x_{ij}}^2 + {R^y_{ij}}^2}}$$

Building m on the GPU

1) Allocate all the necessary memory (both for parameters that remain constant, such as atom position, as well as all the work space) at the beginning of the program - allocation of memory on GPUs as well as the allocation of pinned memory on the CPU is very expensive.

2) Build G0 on the GPU:

```
 makeBGijs_kernel<<<num_blocks,num_threads,sm.total,s>>>(...);
```

num_blocks: one block/atom in LIZ
num_threads: thread over l,l′
sm.total: shared memory size (depends on lmax)
s: we use multiple streams to allow the concurrent calculations for multiple atoms (one CUDA stream / OpenMP thread)

3) calculate 1-tG using zgemm_cublas

Block Inverse

The LSMS method requires only the first diagonal block of the inverse matrix

Recursively apply Schur complement

$$\left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right)^{-1} = \left(\begin{array}{c|c} (A - BD^{-1}C)^{-1} & * \\ \hline * & * \end{array}\right)$$

The block size is a performance tuning parameter:

- Smaller block size: less work
- Larger block size: higher performance of matrix-matrix multiply

Performance of LSMS dominated by double complex matrix matrix multiplication

ZGEMM

Main zblock_lu loop
BLAS: CPU, LAPACK: CPU

```
        n=blk_sz(nblk)
        joff=na-n
        do iblk=nblk,2,-1
        m=n
        ioff=joff
        n=blk_sz(iblk-1)
        joff=joff-n
c invert the diagonal blk_sz(iblk) x blk_sz(iblk) block
        call zgetrf(m,m,a(ioff+1,ioff+1),lda,ipvt,info)
c calculate the inverse of above multiplying the row block
c blk_sz(iblk) x ioff
        call zgetrs('n',m,ioff,a(ioff+1,ioff+1),lda,ipvt,
       &     a(ioff+1,1),lda,info)
        if(iblk.gt.2) then
        call zgemm('n','n',n,ioff-k+1,na-ioff,cmone,a(joff+1,ioff+1),lda,
       &     a(ioff+1,k),lda,cone,a(joff+1,k),lda)
        call zgemm('n','n',joff,n,na-ioff,cmone,a(1,ioff+1),lda,
       &     a(ioff+1,joff+1),lda,cone,a(1,joff+1),lda)
        endif
        enddo
        call zgemm('n','n',blk_sz(1),blk_sz(1)-k+1,na-blk_sz(1),cmone,
       &     a(1,blk_sz(1)+1),lda,a(blk_sz(1)+1,k),lda,cone,a,lda)
```

Main zblock_lu loop
BLAS: CPU
LAPACK: CPU

```
do iblk=nblk,2,-1
...

call zgetrf(…)
call zgetrs(…)

call zgemm(…)
call zgemm(…)

enddo

call zgemm(…)
```

Main zblock_lu loop – GGD BLAS: GPU (CUDA) LAPACK: GPU (CULA device API) (or libsci_acc)

```
call cublas_set_matrix(…)  (t only. m calculated on GPU)

do iblk=nblk,2,-1
...

call cula_device_zgetrf(…)
call cula_device_zgetrs(…)

call cublas_zgemm(…)
call cublas_zgemm(…)

enddo

call cublas_zgemm(…)

call cublas_get_matrix(…)   (tau_00 only)
```

WL-LSMS3

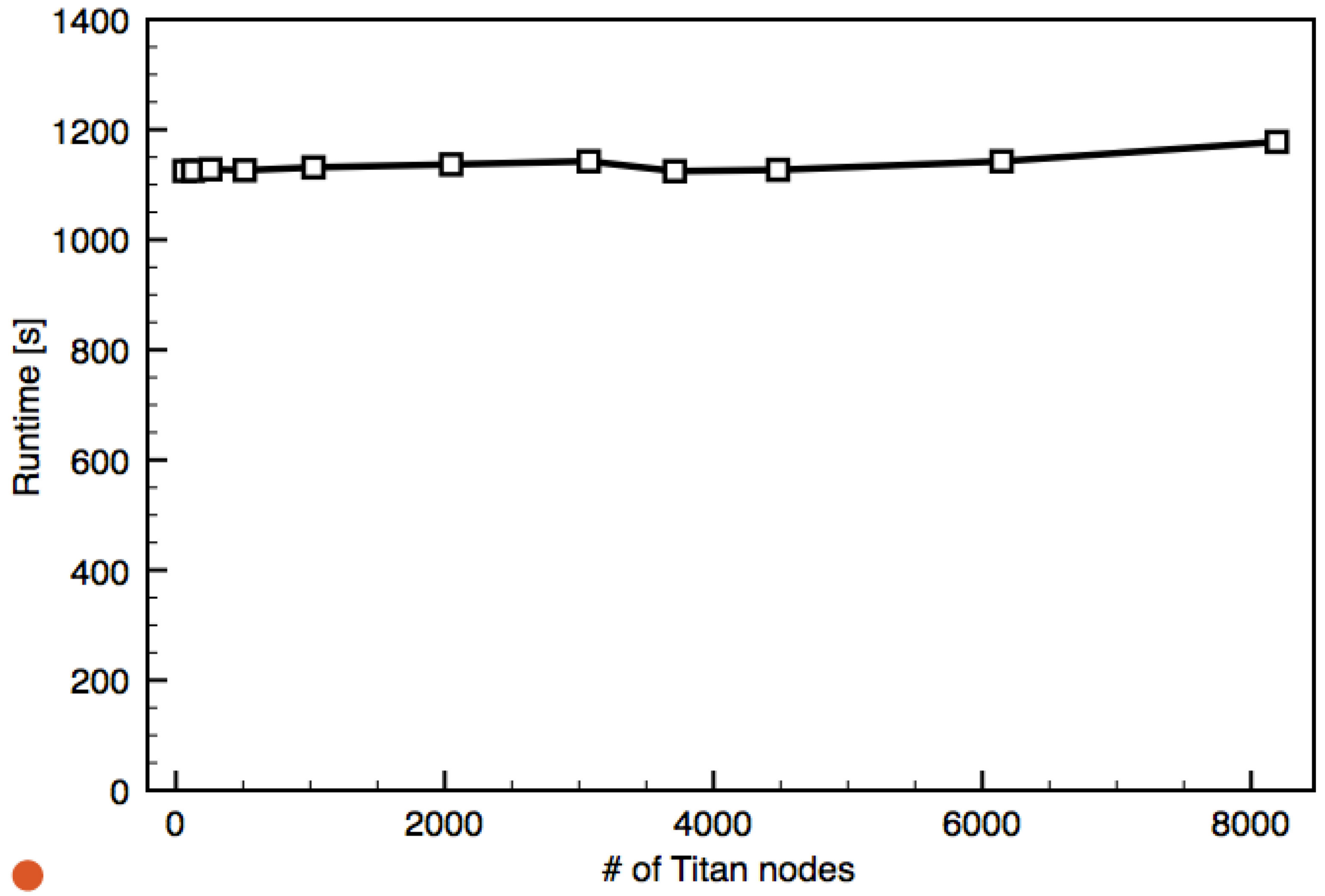
- First Principles Statistical Mechanics of Magnetic Materials
- Identified kernel for initial GPU work
  - zblock_lu (95% of wall time on CPU)
  - kernel performance: determined by BLAS and LAPACK: ZGEMM, ZGETRS, ZGETRF
- Preliminary performance of zblock_lu for 12 atoms/node of Jaguarpf or 12 atoms/GPU
  - For Fermi C2050, times include host-GPU PCIe transfers
  - Currently GPU node does not utilize AMD Magny Cours host for compute

	Jaguarpf node (12 cores AMD Istanbul)	Fermi C2050 using CUBLAS	Fermi C2050 using Cray Libsci
Time (sec)	13.5	11.6	6.4

Slide provided by Cray

Performance and scaling

- One node on Jaguar/Titan has a CPU theoretical peak of 140.8 GFlop/s
- LSMS achieves 95.3 GFlop/s per node with CPU only
- With Fermi GPUs it achieves 344 GFlop/s (3.61x speedup)
- Kepler sees further improved performance
  - Preliminary scaling results are near ideal: (fixed # of WL steps)



Conclusions

- Multithreading and Accelerators enable significantly reduced runtimes for first principles calculations
  - this leads to better statistics/faster convergence for Monte-Carlo
- Multiple acceleration strategies:
  - zblock_lu (most important kernel @ 95% runtime in CPU version of the code)
    - *accelerated by using vendor optimized libraries for matrix inversion and multiplication*
  - m Matrix construction (~30% of the remaining CPU runtime)
    - *accelerated using hand-coded CUDA and library provided matrix multiplication*
- Future work:
  - hybrid calculation (use both CPU and GPU simultaneously for calculations)
  - move more work to GPUs
    - *e.g. Green′s function calculation or single site solvers*

Single Site Wave Functions in Multiple Scattering Theory

In calculating the Green function, we need solutions, both regular and irregular, of the following single site Schrödinger equation, for atom *n* (=1, 2,…, *N*),

$$\left[-\nabla^2 + V_n(\vec{r}_n)\right]\phi_L^n(\vec{r}_n;\varepsilon) = \varepsilon\phi_L^n(\vec{r}_n;\varepsilon)$$

$$\left[-\nabla^2 + V_n(\vec{r}_n)\right]J_L^n(\vec{r}_n;\varepsilon) = \varepsilon J_L^n(\vec{r}_n;\varepsilon)$$

where the single scattering potential is $V_n(\vec{r}_n) = \begin{cases} V_{\text{LDA}}(\vec{r}), & \vec{r} \in \Omega_n; \\ 0, & \text{otherwise.} \end{cases}$

The boundary conditions are

regular solutions $\phi_L^n(\vec{r}_n;\varepsilon) \xrightarrow[r_n \to 0]{} j_l(\sqrt{\varepsilon} r_n) Y_{lm}(\hat{r}_n)$, and

irregular solutions $J_L^n(\vec{r}_n;\varepsilon) \xrightarrow[r_n \to R_C^n]{} j_l(\sqrt{\varepsilon} r_n) Y_{lm}(\hat{r}_n)$,

where R_C^n is the bounding sphere radius of Ω_n, and index $L = \{l, m\}$.

Solution of the Integral Equations

By breaking the single scattering potential into spherical and non-spherical components, such that

$$V_n(\vec{r}) = \tilde{V}_n(r) + \hat{V}_n(\vec{r}), \text{ where } \tilde{V}_n(r_n) \approx -2Z_n / r_n, \text{ as } r_n \to 0,$$

we obtain the single site solutions as

$$\phi_L^n(\vec{r}_n;\varepsilon) = \tilde{\phi}_l^n(r_n;\varepsilon) Y_{lm}(\hat{r}_n) + \hat{\phi}_L^n(\vec{r}_n;\varepsilon), \text{ with } \tilde{\phi}_l^n(r_n;\varepsilon) \xrightarrow[r_n \to 0]{} j_l(\sqrt{\varepsilon} r_n), \text{ and}$$

$$J_L^n(\vec{r}_n;\varepsilon) = \tilde{J}_l^n(r_n;\varepsilon) Y_{lm}(\hat{r}_n) + \hat{J}_L^n(\vec{r}_n;\varepsilon), \text{ with } \tilde{J}_l^n(r_n;\varepsilon) \xrightarrow[r_n \to R_C^n]{} j_l(\sqrt{\varepsilon} r_n).$$

Here $\tilde{\phi}_l^n(r_n;\varepsilon)$ and $\tilde{J}_l^n(r_n;\varepsilon)$ are the solutions of a radial differential equation which can be solved using a finite difference method. The non-spherical solutions are

$$\hat{\phi}_L(\vec{r}_n;\varepsilon) = \int_{r_n' \le r_n} K(\vec{r}_n, \vec{r}_n') \left[\hat{V}_n(\vec{r}_n') \tilde{\phi}_l(r_n';\varepsilon) Y_L(\hat{r}_n') + V_n(\vec{r}_n') \hat{\phi}_L(\vec{r}_n';\varepsilon) \right] d^3\vec{r}_n', \text{ and}$$

$$\hat{J}_L(\vec{r}_n;\varepsilon) = \int_{r_n \le r_n'} K(\vec{r}_n, \vec{r}_n') \left[\hat{V}_n(\vec{r}_n') \tilde{J}_l(r_n';\varepsilon) Y_L(\hat{r}_n') + V_n(\vec{r}_n') \hat{J}_L(\vec{r}_n';\varepsilon) \right] d^3\vec{r}_n',$$

where the kernel function ($r_n^> = \max\{r_n, r_n'\}$ and $r_n^< = \min\{r_n, r_n'\}$) is

$$K(\vec{r}_n, \vec{r}_n') = -\sqrt{\varepsilon} \sum_{L'} Y_{l'm'}(\hat{r}_n) \left[j_{l'}(\sqrt{\varepsilon} r_n^>) n_{l'}(\sqrt{\varepsilon} r_n^<) - n_{l'}(\sqrt{\varepsilon} r_n^>) j_{l'}(\sqrt{\varepsilon} r_n^<) \right] Y_{l'm'}^*(\hat{r}_n')$$

Computational Challenges

In the current implementation, we expand the single site solutions and the potential in spherical harmonics so that the integral equations become a set of coupled one dimensional integral equations.

- The equations are solved numerically on a logarithmic grid ($\approx$ 1000 points) along the radial direction and are solved iteratively ($>$ 10 iterations)
- The subscript index L of the single site solutions is usually cut off at 25, which corresponds to $l_{\max} = 4$, and the single site solutions for each L index are calculated independently.
- For each subscript index L of the single site solutions, if their spherical harmonic expansion is cut off at $l' = 8$, there are 81 ($= (8+1)^2$) coupled integral equations for regular and irregular solutions, respectively.