Exceptional service in the national interest

Sandia National Laboratories

MiniMD based on KokkosArray

Christian Trott, H. Carter Edwards, Daniel Sunderland

SAND 2012-9336 C

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

Outline

Sandia National Laboratories

- Molecular Dynamics
- MiniMD
- Data management
- Integration - A simple kernel
- Temperature calculation - A simple reduction
- Force calculation – Conditional reduction
- Force calculation – Use a few CUDA intrinsics
- Neighborlist build - Specialize for CUDA
- Performance portability

Molecular Dynamics

Sandia National Laboratories

- Solve Newtons equations for ***N*** particles:

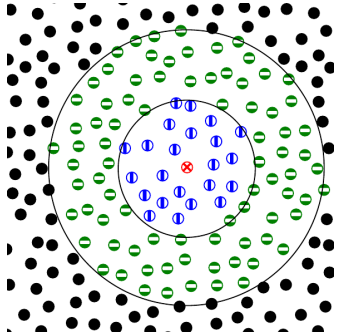
$$\frac{d^2 x_i}{dt^2} = m_i F_i$$

- Force calculation with simple Lennard Jones model:

$$F_i = \sum_{j, r_{ij} < r_{cut}} 6\varepsilon \left[\left(\frac{\varsigma}{r_{ij}} \right)^7 - 2 \left(\frac{\varsigma}{r_{ij}} \right)^{13} \right]$$

- Avoid loop over ***N*** particles with NeighborLists => O(***N***) problem

```
pos_i = pos[i];
for( jj = 0; jj < num_neigh[i]; jj++) {
  j = neighs[i][jj];
  r_ij = pos_i – pos[j]; //random read 3 floats
  if ( |r_ij| < r_cut ) {
    f_i += 6*e*( (s/r_ij)^7 – 2*(s/r_ij)^13 )
  }
}
f[i] = f_i;
```



- Typical numbers: ***N*** = 100k / node; Neighbors: 40
- Sparse memory access moderately compute bound

MiniMD

Sandia National Laboratories

- Part of Mantevo Suite (mantevo.org)
- MiniApp for LAMMPS (lammps.sandia.gov)
  - Test new ideas / programming models before implementing into production code
- Variants for MPI+OpenMP, MPI+OpenCL and MPI+KokkosArray
- 4k lines of code
- Code split into classes:
  - Integrate: main integration loop
  - Force{_LJ/_EAM}: actual force calculation
  - Neighbor: neighbor list construction
  - Comm: communication between MPI process
  - Thermo: calculates thermo dynamic output
- Following slides, try to show real code sometimes simplified

Data management

Sandia National Laboratories

- Data types:

```
typedef KokkosArray::View<double*[3] ,KokkosArray::LayoutRight,device_type>
tvector_2d ;

typedef tvector_2d::HostMirror  tvector_2d_host ;
```

- Atom::growarray() --- allocation

```
x = (double **) realloc_2d_double_array(x,nmax,3,3*nold);
```

becomes

```
tvector_2d xnew("X",nmax);                  //create new array
deep_copy_grow(xnew,x);                     //copy old array to new
x=xnew;                                     //automatically deletes old allocation
h_x = KokkosArray::create_mirror_view(x);   //create host variant of x {no-op on host}
```

- Atom::upload() / download() --- data transfer between host and device

```
KokkosArray::deep_copy(x,h_x);

KokkosArray::deep_copy(h_x,x);
```

No-op if h_x and x are the same

Integration (i) – a simple kernel

- Split function looping over variables into: (i) loop body function, (ii) functor calling loop body function, (iii) function submitting functor

```
class Integrate {
 public:
  ...
  void initialIntegrate();
  ...
 private:
  double **x, **v, **f;
  int nlocal;
}
```

```
class Integrate {
 public:
  ...
  void initialIntegrate();
  ...
 private:
  tvector_2d x,v,f;
  int nlocal;
  friend class InitialIntegrateFunctor;
  InitialIntegrateFunctor* f_initialIntregrate;
  KOKKOS_INLINE_FUNCTION
  void initialIntegrateItem(int i);
}

struct InitialIntegrateFunctor
{
  //required
  typedef tvector_2d::device_type device_type ;

  Integrate c;
  KOKKOS_INLINE_FUNCTION
  void operator()( const int i) const
  {
    c.initialIntegrateItem(i);
  }
};
```

Integration (ii) – a simple kernel

```
void Integrate::initialIntegrate() {
  #pragma omp for
  for(int i = 0; i<nlocal; i++) {
    v[i][0] += dtforce*f[i][0];
    v[i][1] += dtforce*f[i][1];
    v[i][2] += dtforce*f[i][2];
    x[i][0] += dt*v[i][0];
    x[i][1] += dt*v[i][1];
    x[i][2] += dt*v[i][2];
  }
}
```

```
void Integrate::initialIntegrate() {
  f_initialIntegrate->c=*this;

  KokkosArray::parallel_for(nlocal,
     *f_initialIntegrate);
}

KOKKOS_INLINE_FUNCTION
void Integrate::initialIntegrateItem(int i) {
  v(i,0) += dtforce*f(i,0);
  v(i,1) += dtforce*f(i,1);
  v(i,2) += dtforce*f(i,2);
  x(i,0) += dt*v(i,0);
  x(i,1) += dt*v(i,1);
  x(i,2) += dt*v(i,2);
}
```

Temperature calculation – a simple reduction

Sandia National Laboratories

```
double Thermo::temperature() {
  f_temp->c = *this;
  t_act = KokkosArray::parallel_reduce(nlocal,*f_temp);
  return t_act*t_scale;
}

KOKKOSARRAY_INLINE_FUNCTION void Thermo::temperatureItem(const int &i, double &
temp) const {
  double vx,vy,vz;
  vx = v(i,0);  vy = v(i,1);  vz = v(i,2);
  temp += (vx*vx + vy*vy + vz*vz)*mass;
}


struct TemperatureFunctor {
  typedef tvector_2d::device_type  device_type ;    //required
  typedef double    value_type;                      //required: what is the reduction type
  Thermo c;

  KOKKOSARRAY_INLINE_FUNCTION
  void operator()( const int i, value_type & temp) const   {
    c.temperatureItem(i,temp);
  }

  //two mandatory functions explaining how to initialize and how to reduce two values
  KOKKOSARRAY_INLINE_FUNCTION static void init( value_type & update) { update = 0; }
  KOKKOSARRAY_INLINE_FUNCTION static void join( volatile value_type & update ,
                    const volatile value_type & source ) {
    update += source ;
  }
};
```

Force calculation (i) – conditional reduction

Sandia National Laboratories

- Problem: Occasional energy calculation alongside forces
- Goal: use same kernel, but only do reduction if requested
- Old kernel:

```
void Force::compute_fullneigh() {
  for (int i = 0; i < nlocal; i++) {
    const double xtmp = x[i][0];
    const double ytmp = x[i][1];
    const double ztmp = x[i][2];
    double fix = 0; double fiy = 0; double fiz = 0;

    for (int k = 0; k < numneigh[i]; k++) {
      const int j = neighbors[k];
      const double delx = xtmp - x[j][0];
      const double dely = ytmp - x[j][1];
      const double delz = ztmp - x[j][2];
      const double rsq = delx*delx + dely*dely + delz*delz;
      if (rsq < cutforcesq) {
        const double sr2 = 1.0/rsq;
        const double sr6 = sr2*sr2*sr2;
        const double force = sr6*(sr6-0.5)*sr2;
        fix += delx*force; fiy += dely*force; fiz += delz*force;

        if(evflag) energy += sr6*(sr6-1.0); //conditional reduction
      }
    }
    f[i][0] += fix;  f[i][1] += fiy;  f[i][2] += fiz;
  }
}
```

Force calculation (ii) – conditional reduction

Sandia National Laboratories

- Force Kernel:

```
template<int EVFLAG>
KOKKOSARRAY_INLINE_FUNCTION
double ForceLJ::compute_fullneighItem(const int & i) const {
  double energy = 0;
  const double xtmp = x(i,0);  //read from KokkosArray View
  const double ytmp = x(i,1);
  const double ztmp = x(i,2);
  double fix = 0; double fiy = 0; double fiz = 0;

  for (int k = 0; k < numneigh[i]; k++) {
    const int j = neighbors[k];
    const double delx = xtmp - x(j,0);
    const double dely = ytmp - x(j,1);
    const double delz = ztmp - x(j,2);
    const double rsq = delx*delx + dely*dely + delz*delz;
    if (rsq < cutforcesq) {
      const double sr2 = 1.0/rsq;
      const double sr6 = sr2*sr2*sr2;
      const double force = sr6*(sr6-0.5)*sr2;
      fix += delx*force; fiy += dely*force; fiz += delz*force;

      if(EVFLAG) energy += sr6*(sr6-1.0); //reduction property
    }
  }
  f(i,0) += fix;  f(i,1) += fiy;  f(i,2) += fiz;
  return energy;
}
```

Force calculation (iii) – conditional reduction

Sandia National Laboratories

- Functor and definition, overload operator to provide interface for **parallel_for** and **parallel_reduce:**

```
class Integrate {
  ....
  ForceComputeFullneighFunctor* f_compute_fullneigh;

  template<int EVFLAG> KOKKOSARRAY_INLINE_FUNCTION
  double compute_fullneighItem(const int & i) const;
}

struct ForceComputeFullneighFunctor {
  typedef tvector_2d::device_type                    device_type ;
  typedef double    value_type;
  ForceLJ c;

  KOKKOSARRAY_DEVICE_FUNCTION
  void operator()( const int i) const { c.compute_fullneighItem<0>(i); }

  KOKKOSARRAY_DEVICE_FUNCTION
  void operator()( const int i, value_type & energy) const {
     energy += c.compute_fullneighItem<1>(i);
  }

  KOKKOSARRAY_DEVICE_FUNCTION
  static void init( volatile value_type & update) { update = 0; }

  KOKKOSARRAY_DEVICE_FUNCTION static void join( volatile value_type & update ,
                    const volatile value_type & source )
  { update += source; }
};
```

Force calculation (iv) – conditional reduction

Sandia National Laboratories

- Calling function: use **parallel_for** or **parallel_reduce** with the same functor

```
void ForceLJ::compute_fullneigh(Atom &atom, Neighbor &neighbor)
{
  x = atom.x;
  f = atom.f;
  nlocal = atom.nlocal;
  nmax = atom.nmax;
  numneigh=neighbor.numneigh;
  neighbors=neighbor.neighbors;
  maxneighs=neighbor.maxneighs;

  // clear force on own atoms
  ForceZeroFunctor f_forceZero;
  f_forceZero.f=f;
  KokkosArray::parallel_for(nlocal,f_forceZero);
  device_type::fence();

  f_compute_fullneigh->c = *this;
  if(evflag)
    energy = KokkosArray::parallel_reduce(nlocal,*f_compute_fullneigh);
  else
    KokkosArray::parallel_for(nlocal,*f_compute_fullneigh);
  device_type::fence();
}
```

Force calculation – use a few CUDA intrinsics

Sandia National Laboratories

- How to use Texture Cache for random access of position data

```
#if __CUDA_ARCH__
  #define c_x(a,b) tex1Dfetch_f1(lj_x_tex,3*a+b)
#else
  #define c_x(a,b) x(a,b)
#endif

template<int EVFLAG>
KOKKOSARRAY_INLINE_FUNCTION
double ForceLJ::compute_fullneighItem(const int & i) const
{
  const int numneighs = numneigh[i];
  const double xtmp = c_x(i,0);
  const double ytmp = c_x(i,1);
  const double ztmp = c_x(i,2);
  ....
}
```

- Attention: it is **not** enough to guard with custom defines such as I_COMPILE_FOR_CUDA
- The compiler invokes multiple stages and will compile for both host and CUDA, global defines exist in both!
- Plan: support Texture access transparently as a KokkosArray View property

Neighborlist build – Specialize for CUDA

Sandia National Laboratories

- Problem: GPU needs a significantly different algorithm, which includes usage of CUDA only features such as Shared Memory and block level synchronization
- Solution: special CUDA kernels which will not be cross compiled

```
struct NeighborBuildFunctor {
  typedef tvector_2d::device_type  device_type ;
  Neighbor c;
  KOKKOSARRAY_INLINE_FUNCTION void operator()( const int i) const {
  #if DEVICE==2
     c.build_ItemCuda(i);
  #else
     c.build_Item(i);
  #endif
  }
};
```

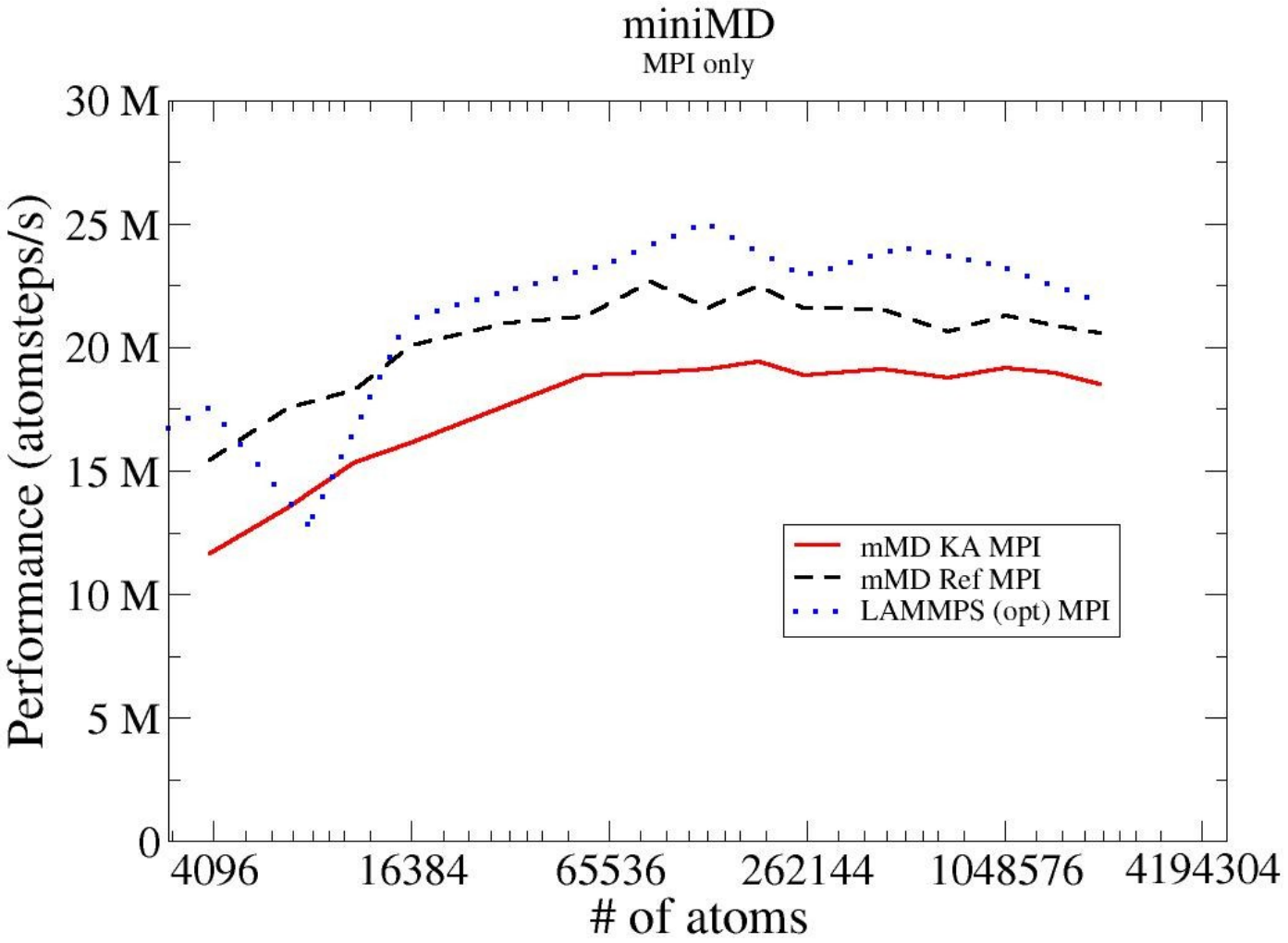
```
#if DEVICE==2
extern __shared__ double sharedmem[];
__device__ inline void Neighbor::build_ItemCuda(const int & ii) const {
  int ibin = blockIdx.x*gridDim.y+blockIdx.y;
  double* other_x = sharedmem;
  int* other_id = (int*) &other_x[3*blockDim.x];

  int bincount_current = bincount[ibin];
  int i = threadIdx.x<bincount_current?bins[ibin*atoms_per_bin+threadIdx.x]:-1;
  double xtmp = x(i,0);
  other_x[threadIdx.x] = xtmp;
  ....
}
#endif
```

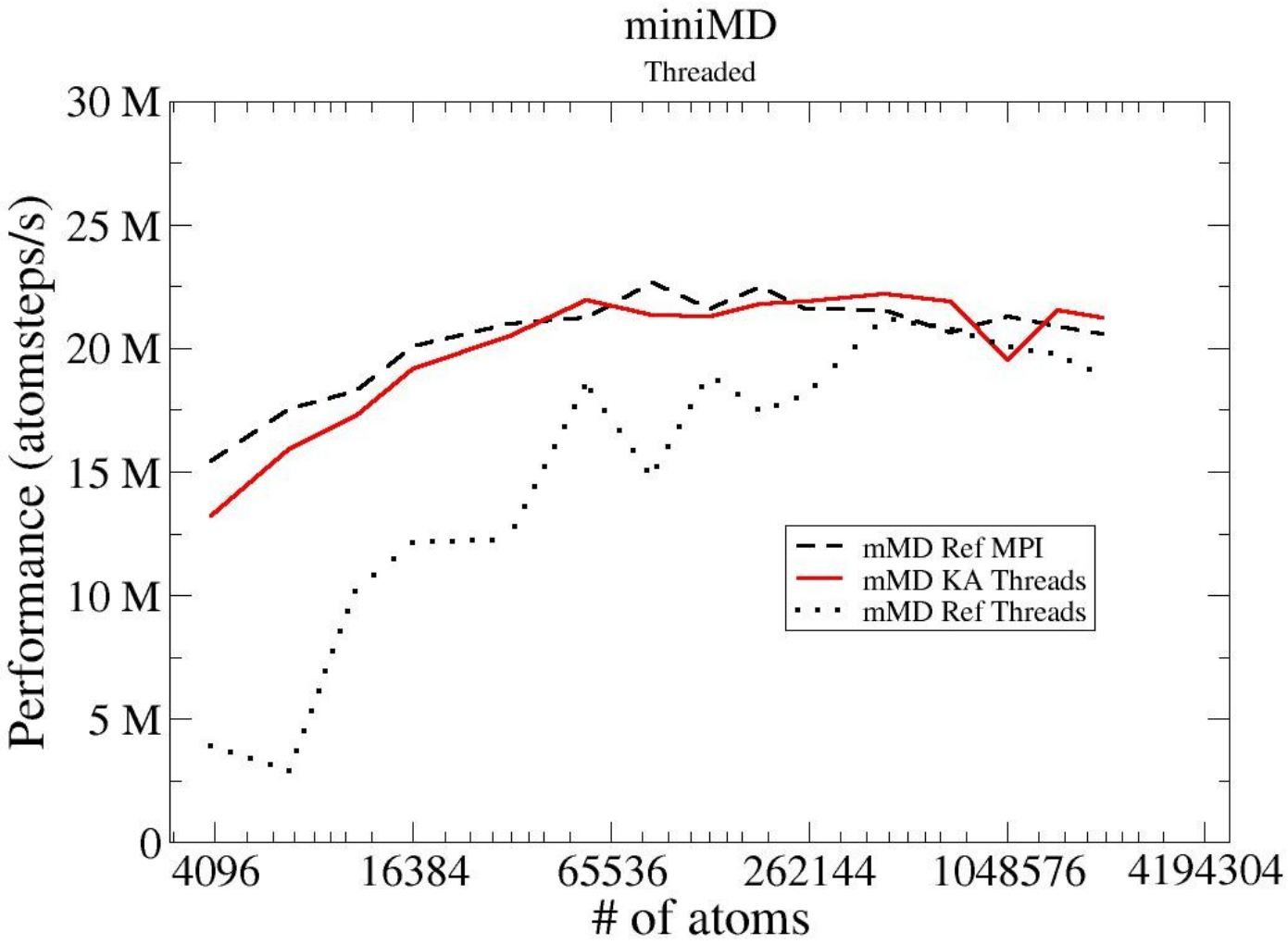
Performance Portability (i)

Sandia National Laboratories

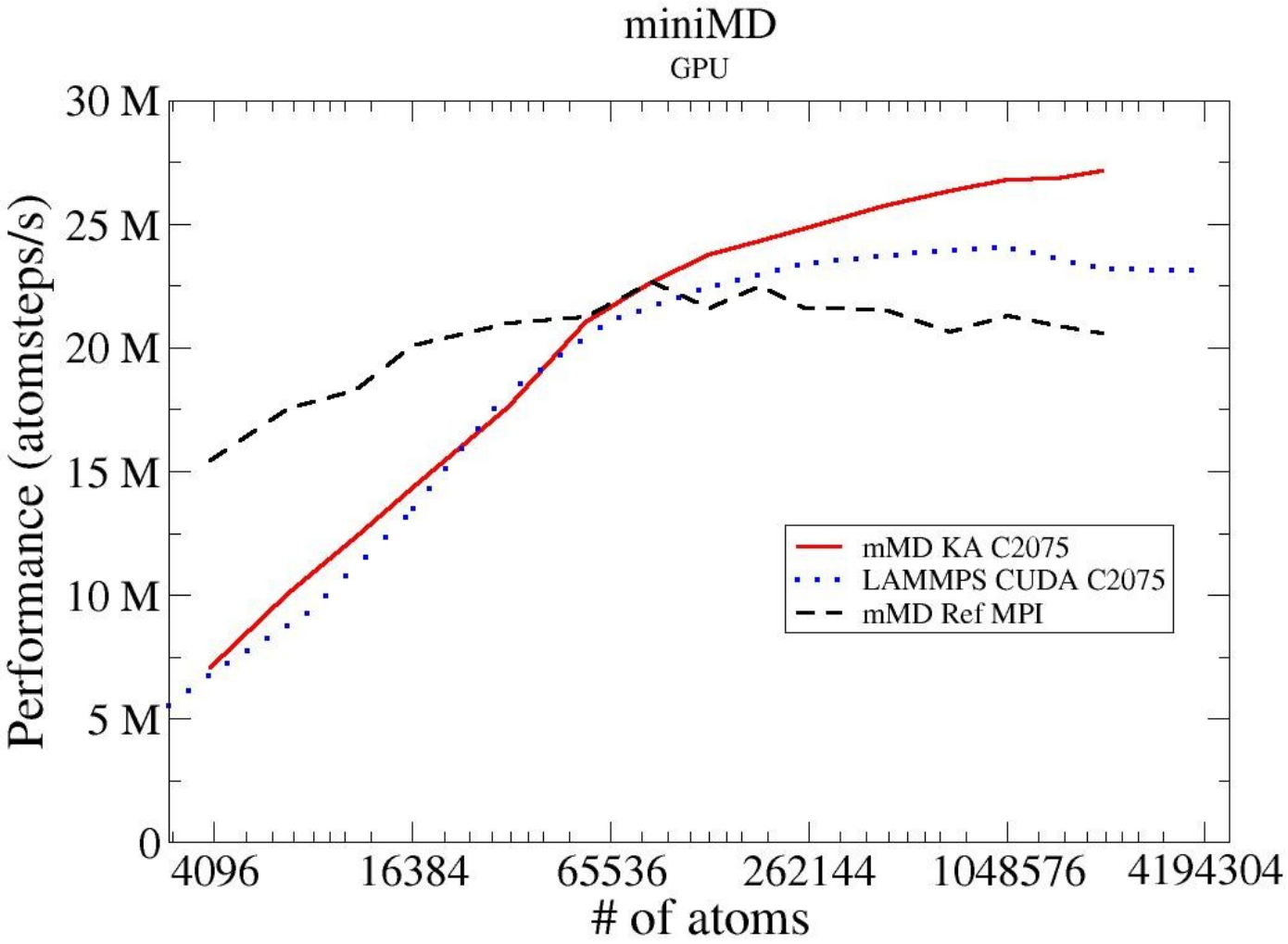
Node level performance: dual Sandy Bridge 16 cores @ 2.6GHz / C2075



Performance Portability (ii)



Performance Portability (iii)



Summary

Sandia National Laboratories

- MiniMD is performance portable with KokkosArray
  - Equivalent to pure CUDA
  - Better than OpenMP implementation
  - <10% hit on MPI only without threading
- Code complexity just slightly increased vs MPI/OpenMP implementation
  - Much less complicated then OpenCL / CUDA implementation
- More future proof then other programming models
  - New backends through KokkosArray and not in production code
  - Simple change of data layout without rewriting kernels
- Not talked about: Out of Bounds check with traceback in debug mode at modest performance hit – very helpful if problem does not occur for small problem-sizes/early in the run

Sandia National Laboratories

Thanks!

Look at mantevo.org for source code!*

crtrott@sandia.gov

*soon: release due before SC12, but mail me for code now if you want to.