Multiscale Universal Interface
A Concurrent Framework for Coupling Heterogeneous Solvers

YU-HANG TANG, SHUHEI KUDO, XIN BIAN, ZHEN LI, GEORGE KARNIADAKIS
Brown University

CSRC Workshop on LAMMPS for Nonequilibrium Systems, Sep 24, 2015
Multiscale Simulations by Domain Decomposition

- Each solver handles a subdomain and use the other as boundary
Diversity in Current Coupling - I

- Equation
  - Newton’s
  - Schrödinger's
  - etc.

- Discretization / Geometry

- Time stepping: uniform, staggered, variable
Example: Grafted Surface
Example: Grafted Surface

SPH

DPD

$V_x$

$y$

Analytical
DPD
SPH

$\text{SPH}$

$\text{DPD}$

RECEIVE/SEND
SEND/RECEIVE

$0.0$ $0.4$ $0.8$ $1.2$ $1.6$ $2.0$

$t = 10000$
$t = 2000$
$t = 1000$
$t = 500$
$t = 200$
$t = 100$

09/24/2015

YU-HANG TANG @ BROWN U | CSRC WORKSHOP ON LAMMPS FOR NONEQUILIBRIUM SYSTEMS
Example: Conjugate Heat Transfer

- **FEM**: Heat equation
- **eDPD**: Navier-Stokes + Heat equation
Example: Conjugate Heat Transfer
Diversity in Concurrent Coupling - II

- Solver: C, C++, Fortran, Python, ...
- Scheme
- Parallelization: Serial, OpenMP, MPI, ...

- Existing solutions largely rely on embedding or metacode

- The majority of existing code
  - was not developed to be coupled
  - need refactoring/invasive development

YU-HANG TANG @ BROWN U | CSRC WORKSHOP ON LAMMPS FOR NONEQUILIBRIUM SYSTEMS
Multiscale Universal Interface (MUI)

**IS**

- A plug-and-play platform for testing ideas on multiscale coupling.
- A communication layer for multi-solver information exchange.
- A header-only C++ library that can be dropped into existing codes easily.

**IS NOT**

- A specific coupling method that dictates which and how physical quantities get coupled.
- A driver/wrapper that requires the exposure of certain programming interfaces from the solver.
Workflow Overview

pressure[double] → push → UNIFACE → mpi → mpi → UNIFACE → tcp → tcp → fetch

flux[int] → UNIFACE

sampler1, sampler2, sampler3

UNIFACE

9/24/2015

YU-HANG TANG @ BROWN U | CSRC WORKSHOP ON LAMMPS FOR NONEQUILIBRIUM SYSTEMS
Abstraction: Data points

- Data point := ( location, type, value )

- Location: Vector Expression Templates
  - arbitrary dimension
  - real/complex coordinate
  - automatic SIMDization

- Value: arbitrary type
  - C++ templates
  - Type list metaprogramming
Abstraction: Sampling

- Texture Sampler
  - Hardware-implemented
  - Interpolate continuous color surface from discrete pixels

- Data sampler
  - C++ functors
  - Can implement any interpolation

...
Sampler Design

template<typename O_TP, typename I_TP=O_TP, typename CONFIG=default_config>
class sampler_gauss {
public:
  using OTYPE      = O_TP;
  using ITYPE      = I_TP;
  using REAL       = typename CONFIG::REAL;
  using INT        = typename CONFIG::INT;
  using point_type = typename CONFIG::point_type;

  sampler_gauss( REAL r_, REAL h_ ) :
    r(r_), h(h_),
    nh(std::pow(2*PI*h,-0.5*CONFIG::D)) {} 

template<template<typename,typename> class CONTAINER>
inline OTYPE filter( point_type focus,
                      const CONTAINER<ITYPE,CONFIG> &data_points ) const {
  REAL wsum = 0;
  OTYPE vsum = 0;
  for(INT i = 0 ; i < data_points.size() ; i++) {
    auto d = (focus-data_points[i].first).normsq();
    if ( d < r*r ) {
      REAL w = nh * std::exp( (-0.5/h) * d );
      vsum += data_points[i].second * w;
      wsum += w;
    }
  }
  return vsum / wsum;
}
Parallelization: MPI MPMD

- Solvers compiled separately, runs concurrently
- MPMD syntax: mpirun -np N1 solver1 : -np n2 solver2
- URI: protocol://domain/interface
  - Use hash function to digitize the string
- Fetch method thread-safe
- TCP communicator in progress
Time coherence

- MUI does not implicitly enforce solvers synchronization
  - In addition: time stepping may not align
Time coherence

- Time frames
  - points of same timestamp merged as frames
  - tagged by timestamp
  - sampling performed on frames

- Chrono sampler
  - Interpolate spatial results from time frames
Selective Communication

- By default MUI broadcast all data points to all peer ranks
  
  - $O(N^2)$ messages!

- In many situations the interpolation algorithm is local

- Regions of interest
  
  - Hint for MUI to cut unnecessary messages
  
  - Arbitrary Boolean operations of boxes, spheres, and points
  
  - Use validity period for moving boundary
Example Revisited: Grafted Surface

/G******* DPD **********/
For t = 0:dt:T
  For each particle i
    If WithinSendRegion(i)
      MUI::Push("v_x", coord[i], vel_x[i])
      MUI::Commit(t)
  
  Force Eval, Integrate...

  t_{SPH} = Floor(t,50dt)
  For each particle i
    If WithinSendRegion(i)
      MUI::Push("v_x", coord[i], vel_x[i])
      MUI::Commit(t)
  
  Force Eval, Integrate...

  t_{DPD} = Floor(t,50dt)
  For each particle i
    If WithinSendRegion(i)
      MUI::Push("v_x", coord[i], vel_x[i])
      MUI::Commit(t)
  
  Force Eval, Integrate...

/******* SPH **********/
For t = 0:50dt:T
  For each particle i
    If WithinSendRegion(i)
      MUI::Push("v_x", coord[i], vel_x[i])
      MUI::Commit(t)
  
  Force Eval, Integrate...

  For each particle i
    If WithinReceiveRegion(i)
      S_s = Quintic(r_{DPD}, h_{DPD})
      S_t = AverageOver(50dt)
      v_x[i] = MUI::Fetch("v_x", coord[i], t_{DPD}, S_s, S_t)
      MUI::Forget(t)

#include "fix.h"
#include <mui/mui.h>

class FixMUI : public Fix {
public:
    FixMUI(class LAMMPS *, int, char **);
    virtual ~FixMUI() {
        if ( interface ) delete interface;
    }
    int setmask() {
        return POST_INTEGRATE | END_OF_STEP;
    }
    virtual void post_integrate();
    virtual void end_of_step();

protected:
    mui::uniface3d *interface;
    double send_upper, send_lower;
    double recv_upper, recv_lower;
    double sample_rc;
};

FixMUI::FixMUI(LAMMPS *lmp, int narg, char **arg)
    nevery = 1;
}

// the PUSH part
void FixMUI::post_integrate()
{
    for (int i = 0; i < atom->nlocal; i++) {
        if ( interface ) delete interface;
    }
    int setmask() {
        return POST_INTEGRATE | END_OF_STEP;
    }
    virtual void post_integrate();
    virtual void end_of_step();

protected:
    mui::uniface3d *interface;
    double send_upper, send_lower;
    double recv_upper, recv_lower;
    double sample_rc;
};

void FixMUI::end_of_step()
{
    double t = update->ntimestep * update->dt;
    interface->commit( t );
    interface->barrier( t-1 );
    interface->forget( t-1 );
}

#include "fix.h"
#include <mui/mui.h>

class FixMUI : public Fix {
public:
    FixMUI(class LAMMPS *, int, char **);
    virtual ~FixMUI() {
        if ( interface ) delete interface;
    }
    int setmask() {
        return POST_INTEGRATE | END_OF_STEP;
    }
    virtual void post_integrate();
    virtual void end_of_step();

protected:
    mui::uniface3d *interface;
    double send_upper, send_lower;
    double recv_upper, recv_lower;
    double sample_rc;
};

FixMUI::FixMUI(LAMMPS *lmp, int narg, char **arg)
    nevery = 1;
}

// the PUSH part
void FixMUI::post_integrate()
{
    for (int i = 0; i < atom->nlocal; i++) {
        if ( interface ) delete interface;
    }
    int setmask() {
        return POST_INTEGRATE | END_OF_STEP;
    }
    virtual void post_integrate();
    virtual void end_of_step();

protected:
    mui::uniface3d *interface;
    double send_upper, send_lower;
    double recv_upper, recv_lower;
    double sample_rc;
};

void FixMUI::end_of_step()
{
    double t = update->ntimestep * update->dt;
    interface->commit( t );
    interface->barrier( t-1 );
    interface->forget( t-1 );
}
Thank you!

Reference & Acknowledgement

Yu-Hang Tang, Shuhei Kudo, Xin Bian, Zhen Li, George Em Karniadakis, Multiscale Universal Interface: A Concurrent Framework for Coupling Heterogeneous Solvers, Journal of Computational Physics, manuscript in revision.

This work was supported by the Department of Energy (DoE) Collaboratory on Mathematics for Mesoscopic Modeling of Materials (CM4). Simulations were carried out at the Oak Ridge Leadership Computing Facility through the Innovative and Novel Computational Impact on Theory and Experiment program at Oak Ridge National Laboratory under project BIP102 and BIP118.
Language compatibility

- MUI is written in C++11
  - C and Fortran wrapper included

- Compatible compilers

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Version</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCC</td>
<td>4.8.3</td>
<td>-std=c++11</td>
</tr>
<tr>
<td>Clang</td>
<td>3.5.0</td>
<td>-std=c++11</td>
</tr>
<tr>
<td>Intel C++</td>
<td>15.0</td>
<td>-std=c++11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/Qstd=c++11</td>
</tr>
<tr>
<td>NVCC</td>
<td>7.0</td>
<td>-std=c++11</td>
</tr>
</tbody>
</table>
Example Revisited: Heat Transfer

- FEM: Heat equation
- eDPD: Navier-Stokes + Heat equation

```plaintext
/******* eDPD ********/
For t = 0:dt:T
  t_{FEM} = Floor(t, 10dt)
  For each particle i
    If WithinCutoffOfCylinder(i)
      S_s = Linear
      S_t = ExactTime
      T_{wall} = MUI::Fetch("T", coord[i], t_{FEM}, S_s, S_t)
      q = CalculateFlux(T[i], T_{wall})
      MUI::Push("q", coord[i], -q/Cv)
      MUI::Commit(t)
    If t % 10dt = 0 then MUI::Forget(t-10dt)
  Force Eval, Integrate...

/******* FEM *******/
For t = 0:10dt:T
  For each boundary vertex i
    MUI::Push("T", coord[i], T[i])
    MUI::Commit(t)
  For each boundary vertex i
    S_s = VoronoiMean(Vertices)
    S_t = SumOver(10dt)
    f[i] = MUI::Fetch("q", coord[i], t, S_s, S_t)
    MUI::Forget(t)
  Solve for Next Step
```

09/24/2015

YU-HANG TANG @ BROWN U | CSRC WORKSHOP ON LAMMPS FOR NONEQUILIBRIUM SYSTEMS