Parallel Molecular Dynamics with a Time-Reversible Nose-Hoover Thermostat on CPUs and GPUs

1.0

Generated by Doxygen 1.8.2

Mon Jan 13 2014 22:54:02

Contents

Chapter 1

CBEMDGPU

CBE MD on GPUs

by Nathan A. Mahynski, George A. Khoury, and Carmeline J. Dsilva

Our main report in located in the report/ directory (report.pdf).

Compiling

See main.cpp to set parameters which are documented by example in this file. Our code has the following dependencies:

1. Boost (http://www.boost.org/). Refer to Makefile for PATHTOBOOST variable which must be set. Note that compilation on TIGER is easy since you are allowed to link to other user's directories. George's installation is already provided in the Makefile.
2. Cuda toolkit 5.5 (module load module cudatoolkit/5.5.22 on TIGER)
3. Intel C++ Compilers (icpc) and OMP (module load openmpi/intel-12.1/1.4.5/64 on TIGER)
4. googletest (https://code.google.com/p/googletest/). Again George's GTEST_DIR directory (see Makefile) is provided to be linked against at compile time.

See FYI section below for more details. These must be taken care of before attempting to compile the code. Always use "make clean" before attempting a new compilation.

To compile the CPU version, type $ make MD which produces a binary called md

To compile the GPU version, type $ make -f Makefile_cuda which produces a binary called md

To compile tests, type $ make TESTS which produces a binary called tests

To compile the timing executable (used for CPU only scaling studies), type $ make TIMING which produces a binary called timing

To obtain the CUDA scaling results go to the cuda_timing/ directory, type $ make -f Makefile_cuda which produces a binary called cuda_timing

To compile the program that runs a simulation to compare with LAMMPS output, type $ make LMP_COMPARE which produces a binary called lmp_compare used to generate data at specific settings we also ran on LAMMPS with.

To compile the program that tests the NVE integrator, type $ make TEST_NVE which produces a binary called test_nve

In the Makefile, the PATHTOBOOST variable should point to the diretory where the C++ boost libraries are saved.

In the GTEST_DIR variable should point to the directory where the Google Tests libraries are saved.

Note that the Intel C++ compilers must be used; the GNU C++ compilers do not work with the OpenMP portion of our code (this is a known bug in the compiler, see FYI section).

Execution

Most binaries expects 4 input, the number of threads to use with OMP, the number of atoms, the skin radius for the cell/neighbor lists, and the number of steps to simulate. $ ./md nthreads natoms rs nsteps > log 2> err.

The exceptions are (1) tests which is simply executed as ./tests, and (2) test_nve and (3) lmp_compare which are executed as ./binary_name nthreads. However, the latter two are not of much interest; if you want to check the code is running just check to see if ./tests works.

This also produces a trajectory.xyz file which can be visualized with VMD (if you have it installed) $ vmd -xyz trajectory.xyz

Submission scripts for TIGER are included in the run_scaling.sh∗ files. The integer suffix (i.e. run_scaling.sh.1) refers to the number of threads OMP will use.

Explanation of main.cpp

How to use, change, and make your own in 10 steps.

1. Most binaries expects 4 input, the number of threads to use with OMP, the number of atoms, the skin radius for the cell/neighbor lists, and the number of steps to simulate. $ ./md nthreads natoms rs nsteps > log 2> err

1. The random number generator seed is then set manually to ensure that results are reproducible.

1. The user must then specify basic properties about the system. A systemDefinition object should be instantiated and then temperature, box size, particle mass, and pair potential cutoff should be specified

   $ systemDefinition a;

   $ a.setTemp(1.0);...

1. The system can then be initialized with the command initThermal or initRandom (see doxygen documentation for more details). For example, in the former:

   $ a.initThermal(nAtoms, Temp, rngSeed, separation);

1. Where "nAtoms" is the number of atoms, "separation" is the initial separation of the particles on the simple cubic lattice on which they are initialized, and the rest of the variables are self-explanatory. This command completely initializes the system for simulation.

1. Next, the pair potential should be specified. As an example in main.cpp, the preprocessor flag NVCC is used to select either the CPU version of the shifted lennard-jones potential (slj) or the GPU version (dev_slj). The Makefile (compare to Makefile_cuda) will define this if necessary and allows the code to flow naturally and work in both cases.

   $ pointFunction_t pp = slj;

   $ a.setPotential(pp);

1. If the GPUs are used (NVCC compiler flag is defined) the number of threads and blocks the GPU kernel will be invoked with must be set. This is done next.

1. Then additional arguments for the pair potential must be specified. In the case of the slj function, variables epsilon, sigma, delta, and ushift must be given. This is then set in the systemDefinition.

   $ std::vector <float> args(5);

   $ args[0] = 1.0; // epsilon

   $ args[1] = 1.0; // sigma

   $ args[2] = 0.0; // delta

   $ args[3] = 0.0; // ushift

   $ a.setPotentialArgs(args);

1. Afterwards, the integrator (ensemble) must be specified. In the case of the NVT ensemble where we are using the Nose-Hoover thermostat, the integrator is given by the object nvt_NH. This requires a damping constant, which for the slj potential should be about unity. Then the numerical timestep should be given, for the slj this is usually efficient around 0.005.

   $ nvt_NH integrate (1.0);

   $ integrate.setTimestep(timestep);

1. Finally the simulation is ready to iterate. A simple loop can be set to do this. An example for the case of the NVE ensemble is also provided in test_nve.cpp which can be compiled with make TEST_NVE (see test_nve.cpp)

FYI

Known bugs, etc.

There are a few known instances of bugs related to compiler options, etc.

1. Using icpc instead of g++ Our code uses OMP to parallelize many calculations. Specifically, the atoms member (vector) in the systemDefinition object must be shared often. However, because it is a member of a class g++ struggles to properly share this in memory. This is a known bug in the g++ compiler which the intel (icpc) version handles rigorously. As a result, our code will produce errors if compiled with the g++ compiler when more than 1 core is used. To use this on tiger the module openmpi/intel-12.1/1.4.5/64 should be loaded to make the icpc compiler accessible. This also provides the OMP libraries necessary.

1. Cuda toolkit 5.5 CUDA is a finicky tool. Different GPUs require different toolkits and versions to work properly. In fact, compilation may succeed with a bad version but the run time behavior produces unexpected (incorrect) results. For the K20 cards on tiger, the latest toolkit (v5.5) must be loaded. To do so, load the module cudatoolkit/5.5.22 before attempting to compile the program. The Makefile must include flags consistent with the GPUs version of CUDA (which on TIGER in 3.5) so the Makefile_cuda contains a flag "NVFLAGS = -gencode arch=compute_35,code=sm_35" for the .cu files. Furthermore, you will find that preprocessor flags NVCC and NOGPU are found throughout the code which act as switches to activate/deactivate GPU functionality throughout the compilation process.

1. Tiger GPUs Unfortunately it appears that the queueing system on tiger is having problems reserving all or some of the GPUs exclusively for single jobs by users. As a result, thrust (the GPU equivalent of the STL for C++) will have memory issues if one or more of the GPUs on a node are already in use. Because of the unusually high load on tiger over the past month, we have been unable to obtain good results on this cluster since usually this situation is encountered. Our private GPU cluster was used to obtian our results instead, though our Makefile_cuda is set so this should compile properly on tiger if you want to check.

1. Use of preprocessor flags The use of NVCC and NOGPU flags is crucial during compilation to ensure the correct header files are linked and compiled, and certain sections of code are ignored depending on the compiler. The Makefiles provided already handle this appropriately, but one should be aware of these details while reading the code.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Class Documentation

4.1 atom Struct Reference

Structure of an atom.

`#include <dataTypes.h>`

Public Attributes

- float3 pos
  Position.
- float3 vel
  Velocity.
- float3 acc
  Acceleration.

4.1.1 Detailed Description

Structure of an atom.

Definition at line 29 of file dataTypes.h.

The documentation for this struct was generated from the following file:

- /Users/nathanmahynski/CBEMDGPU/dataTypes.h

4.2 cellList_cpu Class Reference

`#include <cellList.h>`

Public Member Functions

- cellList_cpu (const float3 &box, const float rc, const float rs)
- void checkUpdate (const systemDefinition &sys)
  Check if the neighbor list requires updating.
- int cell (const float3 &pos)
  Calculate the cell in which a given coordinate is located.
- int head (const int cell) const

Return the first atom (aka 'head') of each cell.

- int list (const int index) const

  Iterates through a linked list of particles, returns the index of the next atom in line.

- std::vector< int > neighbors (const int cellID) const

  Returns the indices of a cell's neighboring cells.

Public Attributes

- int3 nCells

  Number of cells in each direction.

4.2.1 Detailed Description

Cell Lists

Author

Nathan A. Mahynski

Date

11/19/13

Maintains a linked list to track cells on the CPU

Definition at line 39 of file cellList.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 cellList_cpu::cellList_cpu (const float3 & *box*, const float *rc*, const float *rs*)

Cell Lists

Author

Nathan A. Mahynski

Date

11/19/13

Initialize a cell list

Parameters

in	*box*	Box size
in	*rc*	Cutoff radius
in	*rs*	Skin Radius

Definition at line 156 of file cellList.cpp.

References nCells.

```
                                                                    {
    if (rc < 0.0) {
    throw customException("Cutoff radius must be > 0");
    return;
```

```
}
rc_ = rc;
if (rs < 0.0) {
throw customException("Skin radius must be > 0");
return;
}
rs_ = rs;

box_ = box;

start_ = 1;
lcell_.x = 1.01*(rc+rs);
nCells.x = (int) floor (box.x/lcell_.x);
lcell_.x = (box.x/nCells.x);

lcell_.y = 1.01*(rc+rs);
nCells.y = (int) floor (box.y/lcell_.y);
lcell_.y = (box.y/nCells.y);

lcell_.z = 1.01*(rc+rs);
nCells.z = (int) floor (box.z/lcell_.z);
lcell_.z = (box.z/nCells.z);

if (lcell_.x <= (rc+rs) || lcell_.y <= (rc+rs) || lcell_.z < (rc+rs)) {
throw customException("Cell width must exceed sum of cutoff
   and skin radius");
return;
}

if (lcell_.x < 1) {
throw customException ("Box dimension x too small relative
   to skin and cutoff radius to use cell lists");
return;
}
if (lcell_.y < 1) {
throw customException ("Box dimension y too small relative
   to skin and cutoff radius to use cell lists");
return;
}
if (lcell_.z < 1) {
throw customException ("Box dimension z too small relative
   to skin and cutoff radius to use cell lists");
return;
}

if (nCells.x < 3 || nCells.y < 3 || nCells.z < 3) {
throw customException("Must be able to have at least 3 cells
   in each direction, change box size, skin, or cutoff radius");
return;
}

try {
head_.resize(nCells.x*nCells.y*nCells.z, -1);
} catch (std::exception &e) {
std::cerr << e.what() << std::endl;
throw customException ("Unable to initialize head for cell
   list");
return;
}
// build neighbors for each cell
neighbor_.resize(nCells.x*nCells.y*nCells.z);
for (unsigned int cellID = 0; cellID < nCells.x*nCells.y*nCells
  .z; ++cellID) {
    const int zref = floor(cellID/(nCells.x*nCells.y));
    const int yref = floor((cellID - zref*(nCells.x*nCells.y))/
  nCells.y);
    const int xref = floor(cellID - zref*(nCells.x*nCells.y) -
  yref*nCells.x);
    for (int dx = -1; dx <= 1; ++dx) {
    int xcell = xref + dx;
    if (xcell >= nCells.x) xcell = 0;
    if (xcell < 0) xcell = nCells.x-1;
    for (int dy = -1; dy <= 1; ++dy) {
        int ycell = yref + dy;
        if (ycell >= nCells.y) ycell = 0;
        if (ycell < 0) ycell = nCells.y-1;
        for (int dz = -1; dz <= 1; ++dz) {
        int zcell = zref + dz;
        if (zcell >= nCells.z) zcell = 0;
        if (zcell < 0) zcell = nCells.z-1;
        const int cellID2 = xcell + ycell*nCells.x + zcell*(nCells
  .x*nCells.y);
        neighbor_[cellID].push_back(cellID2);
        }
    }
    }
}
```

```
    for (unsigned int cellID = 0; cellID < nCells.x*nCells.y*nCells
      .z; ++cellID) {
    if (neighbor_[cellID].size() != 27) {
        throw customException ("Cell list initial build failed
       to find 27 neighbors (including self)");
        return;
    }
    }
}
```

4.2.3 Member Function Documentation

4.2.3.1 int cellList_cpu::cell (const float3 & *pos*)

Calculate the cell in which a given coordinate is located.

Calculates the cell (linear index of it) a position belongs to in a periodic box. The position does not need to be "inside" the box, boundary conditions are applied.

Parameters

in	*pos*	Position of atom

Returns

cell

Definition at line 252 of file cellList.cpp.

References nCells.

Referenced by checkUpdate(), and head().

```
                                        {
    float3 inBox = pbc(pos, box_);
    const int x = (int) floor(inBox.x/lcell_.x);
    const int y = (int) floor(inBox.y/lcell_.y);
    const int z = (int) floor(inBox.z/lcell_.z);
    return x + y*nCells.x + z*nCells.x*nCells.y;
}
```

4.2.3.2 void cellList_cpu::checkUpdate (const systemDefinition & *sys*)

Check if the neighbor list requires updating.

Checks to see if the cell list needs to be updated and if so, rebuilds Uses linked list, so algorithm in O(N)

Parameters

in	*sys*	System definition

Definition at line 266 of file cellList.cpp.

References systemDefinition::atoms, systemDefinition::box(), cell(), and systemDefinition::numAtoms().

Referenced by integrator::calcForce().

```
                                                    {
    int build = 0;
    if (start_) {
    const int natoms = sys.numAtoms();
    try {
            list_.resize(natoms, -1);
        } catch (std::exception& e) {
            std::cerr << e.what() << std::endl;
            throw customException ("Unable to initialize list
       for cell list");
```

```
            return;
        }
        try {
            posAtLastBuild_.resize(natoms);
        } catch (std::exception& e) {
            std::cerr << e.what() << std::endl;
            throw customException ("Unable to initialize
       position list for cell list");
            return;
        }
        start_ = 0;
        build = 1;
    } else {
        // check max displacements
        float drMax1_ = 0.0;
        float drMax2_ = 0.0;
        float3 dummy;
            for (unsigned int i = 0; i < sys.numAtoms(); ++i) {
                const float dr2 = pbcDist2 (sys.atoms[i].pos,
      posAtLastBuild_[i], dummy, sys.box());
                if (dr2 > drMax1_*drMax1_) {
                    drMax2_ = drMax1_;
                    drMax1_ = sqrt(dr2);
                } else if (dr2 > drMax2_*drMax2_) {
                    drMax2_ = sqrt(dr2);
                }
            }
        if (drMax1_+drMax2_ > rs_) {
            build = 1;
        } else {
            build = 0;
        }
    }

    if (build) {
        if (sys.numAtoms() != list_.size()) {
            throw customException ("Number of atoms in
       simulation has changed");
            return;
        }
        for (unsigned int i = 0; i < head_.size(); ++i) {
            head_[i] = -1;
        }
            for (unsigned int i = 0; i < sys.numAtoms(); ++i) {
                const int icell = cell(sys.atoms[i].pos);
                list_[i] = head_[icell];
                head_[icell] = i;
                posAtLastBuild_[i] = sys.atoms[i].pos;
            }
    }

    return;
}
```

The documentation for this class was generated from the following files:

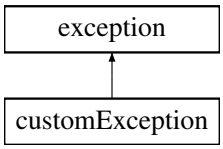
- /Users/nathanmahynski/CBEMDGPU/cellList.h
- /Users/nathanmahynski/CBEMDGPU/cellList.cpp

4.3 customException Class Reference

Error reporting.

`#include <common.h>`

Inheritance diagram for customException:



Public Member Functions

- const char * what () const throw ()

  Return a message pertaining to what went wrong.
- customException (std::string m="custom exception occurred")

  Instantiate the object with a user specified message.

4.3.1 Detailed Description

Error reporting.

Common values, structures, etc.

Author

Nathan A. Mahynski

Date

11/17/13

Definition at line 14 of file common.h.

The documentation for this class was generated from the following file:

- /Users/nathanmahynski/CBEMDGPU/common.h

4.4 float3 Struct Reference

3 floating point numbers, same as defined for GPUs

`#include <dataTypes.h>`

Public Attributes

- float **x**
- float **y**
- float **z**

4.4.1 Detailed Description

3 floating point numbers, same as defined for GPUs

Requisite 'optimal' data types.

Author

Nathan A. Mahynski

Date

11/21/13

Definition at line 16 of file dataTypes.h.

The documentation for this struct was generated from the following file:

- /Users/nathanmahynski/CBEMDGPU/dataTypes.h

4.5 int3 Struct Reference

3 integers, same as defined for GPUs

`#include <dataTypes.h>`

Public Attributes

- int **x**
- int **y**
- int **z**

4.5.1 Detailed Description

3 integers, same as defined for GPUs

Definition at line 22 of file dataTypes.h.

The documentation for this struct was generated from the following file:

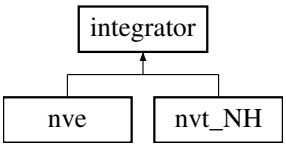
- /Users/nathanmahynski/CBEMDGPU/dataTypes.h

4.6 integrator Class Reference

Base class for integrators such as NVT (Nose-Hoover) or NVE ensembles.

`#include <integrator.h>`

Inheritance diagram for integrator:



Public Member Functions

- void setTimestep (const float dt)

  Set the integrator timestep.
- void calcForce (systemDefinition &sys)

  Calculate the forces on each atom.
- virtual void step (systemDefinition &sys)=0

  Move the system forward a step in time.

Protected Attributes

- cellList_cpu cl_

  Cell or neighbor list.
- std::vector< float3 > lastAccelerations_

  Acceleration of particles on previous timestep (useful for NVE integrator)
- float dt_

  Timestep size.

- int start_

  Flag for whether this object has been initialized or not.

4.6.1 Detailed Description

Base class for integrators such as NVT (Nose-Hoover) or NVE ensembles.

Integrator

Author

Nathan A. Mahynski

Date

11/17/13

Definition at line 15 of file integrator.h.

4.6.2 Member Function Documentation

4.6.2.1 void integrator::calcForce (systemDefinition & *sys*)

Calculate the forces on each atom.

Integration

Author

Nathan A. Mahynski

Date

11/19/13

Calculate the pairwise forces in a system. This also calculates the potential energy of a system. The kinetic energy is calculated during the verlet integration.

Parameters

in,out	*sys*	System definition

Calculate the pairwise forces in a system. This also calculates the potential energy of a system. The kinetic energy is calculated during the verlet integration.

Parameters

in,out	*sys*	System definition

Definition at line 23 of file integrator.cpp.

References systemDefinition::atoms, systemDefinition::box(), cellList_cpu::checkUpdate(), cl_, cellList_cpu::head(), cellList_cpu::list(), systemDefinition::mass(), cellList_cpu::nCells, cellList_cpu::neighbors(), systemDefinition::numAtoms(), systemDefinition::potential, systemDefinition::potentialArgs(), systemDefinition::rcut(), and systemDefinition::setPotE().

Referenced by nve::step(), and nvt_NH::step().

```
                                                {
    // For cache coeherency allocate new space for calculations
    float3 empty;
    empty.x = 0; empty.y = 0; empty.z = 0;
    std::vector <float3> acc (sys.numAtoms(), empty);
    const float rc = sys.rcut();

    // every time, check if the cell list needs to be updated first
    cl_.checkUpdate(sys);

    float Up = 0.0;
    const float3 box = sys.box();
    const float invMass = 1.0/sys.mass();

    // traverse cell list and calculate total system potential energy
    std::vector <float> args = sys.potentialArgs();
#pragma omp parallel for reduction(+:Up) schedule(dynamic, 1)
    for (unsigned int cellID = 0; cellID < cl_.nCells.x*cl_.nCells
      .y*cl_.nCells.z; ++cellID) {
        int atom1 = cl_.head(cellID);
        while (atom1 >= 0) {
            std::vector < int > neighbors = cl_.neighbors(cellID);
            for (int index = 0; index < neighbors.size(); ++index) {
                const int cellID2 = neighbors[index];
                int atom2 = cl_.head(cellID2);
                while (atom2 >= 0) {
                    if (atom1 > atom2) {
                        float3 pf;
                        Up += sys.potential (&sys.atoms[atom1].
      pos, &sys.atoms[atom2].pos, &pf, &box, &args[0], &rc);
                        acc[atom1].x -= pf.x*invMass;
                        acc[atom1].y -= pf.y*invMass;
                        acc[atom1].z -= pf.z*invMass;
                        acc[atom2].x += pf.x*invMass;
                        acc[atom2].y += pf.y*invMass;
                        acc[atom2].z += pf.z*invMass;
                    }
                    atom2 = cl_.list(atom2);
                }
            }
            atom1 = cl_.list(atom1);
        }
    }

    // save acceleration in array of atoms in system
    #pragma omp parallel for
    for (int i = 0; i < sys.numAtoms(); ++i) {
        sys.atoms[i].acc.x = -acc[i].x;
        sys.atoms[i].acc.y = -acc[i].y;
        sys.atoms[i].acc.z = -acc[i].z;
    }

    // set Up
    sys.setPotE(Up);
}
```

The documentation for this class was generated from the following files:

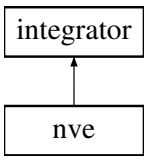
- /Users/nathanmahynski/CBEMDGPU/integrator.h
- /Users/nathanmahynski/CBEMDGPU/integrator.cpp
- /Users/nathanmahynski/CBEMDGPU/integrator.cu

4.7 nve Class Reference

Integration scheme that preserves total energy of the system.

`#include <nve.h>`

Inheritance diagram for nve:



Public Member Functions

- void step (systemDefinition &sys)
- void setTimestep (const float dt)

  Set the integrator timestep.
- void calcForce (systemDefinition &sys)

  Calculate the forces on each atom.

Protected Attributes

- cellList_cpu cl_

  Cell or neighbor list.
- std::vector< float3 > lastAccelerations_

  Acceleration of particles on previous timestep (useful for NVE integrator)
- float dt_

  Timestep size.
- int start_

  Flag for whether this object has been initialized or not.

4.7.1 Detailed Description

Integration scheme that preserves total energy of the system.

NVE integration.

Author

Nathan A. Mahynski

Date

11/18/13

Definition at line 14 of file nve.h.

4.7.2 Member Function Documentation

4.7.2.1 void integrator::calcForce (systemDefinition & *sys*) [inherited]

Calculate the forces on each atom.

Integration

Author

Nathan A. Mahynski

Date

11/19/13

Calculate the pairwise forces in a system. This also calculates the potential energy of a system. The kinetic energy is calculated during the verlet integration.

Parameters

in,out	*sys*	System definition

Calculate the pairwise forces in a system. This also calculates the potential energy of a system. The kinetic energy is calculated during the verlet integration.

Parameters

in,out	*sys*	System definition

Definition at line 23 of file integrator.cpp.

References systemDefinition::atoms, systemDefinition::box(), cellList_cpu::checkUpdate(), integrator::cl_, cellList_cpu::head(), cellList_cpu::list(), systemDefinition::mass(), cellList_cpu::nCells, cellList_cpu::neighbors(), systemDefinition::numAtoms(), systemDefinition::potential, systemDefinition::potentialArgs(), systemDefinition::rcut(), and systemDefinition::setPotE().

Referenced by step(), and nvt_NH::step().

```
                                                    {
    // For cache coeherency allocate new space for calculations
    float3 empty;
    empty.x = 0; empty.y = 0; empty.z = 0;
    std::vector <float3> acc (sys.numAtoms(), empty);
    const float rc = sys.rcut();

    // every time, check if the cell list needs to be updated first
    cl_.checkUpdate(sys);

    float Up = 0.0;
    const float3 box = sys.box();
    const float invMass = 1.0/sys.mass();

    // traverse cell list and calculate total system potential energy
    std::vector <float> args = sys.potentialArgs();
#pragma omp parallel for reduction(+:Up) schedule(dynamic, 1)
    for (unsigned int cellID = 0; cellID < cl_.nCells.x*cl_.nCells
      .y*cl_.nCells.z; ++cellID) {
        int atom1 = cl_.head(cellID);
        while (atom1 >= 0) {
            std::vector < int > neighbors = cl_.neighbors(cellID);
            for (int index = 0; index < neighbors.size(); ++index) {
                const int cellID2 = neighbors[index];
                int atom2 = cl_.head(cellID2);
                while (atom2 >= 0) {
                    if (atom1 > atom2) {
                        float3 pf;
                        Up += sys.potential (&sys.atoms[atom1].
      pos, &sys.atoms[atom2].pos, &pf, &box, &args[0], &rc);
                        acc[atom1].x -= pf.x*invMass;
                        acc[atom1].y -= pf.y*invMass;
                        acc[atom1].z -= pf.z*invMass;
                        acc[atom2].x += pf.x*invMass;
                        acc[atom2].y += pf.y*invMass;
                        acc[atom2].z += pf.z*invMass;
                    }
                    atom2 = cl_.list(atom2);
                }
            }
            atom1 = cl_.list(atom1);
        }
    }

    // save acceleration in array of atoms in system
    #pragma omp parallel for
    for (int i = 0; i < sys.numAtoms(); ++i) {
        sys.atoms[i].acc.x = -acc[i].x;
        sys.atoms[i].acc.y = -acc[i].y;
        sys.atoms[i].acc.z = -acc[i].z;
    }

    // set Up
    sys.setPotE(Up);
}
```

4.7.2.2 void nve::step (systemDefinition & *sys*) [virtual]

Do NVE integration

Author

Nathan A. Mahynski

Date

11/18/13

Integrate a single timestep forward using Velocity-Verlet integration scheme. Update the positions and velocities using velocity verlet integration. This uses the accelerations stored on each atom. Creates a cell list the first time it is called.

Parameters

in,out	*sys*	System definition

Implements integrator.

Definition at line 23 of file nve.cpp.

References systemDefinition::atoms, systemDefinition::box(), integrator::calcForce(), integrator::cl_, integrator::dt_, systemDefinition::mass(), systemDefinition::numAtoms(), systemDefinition::rcut(), systemDefinition::rskin(), systemDefinition::setKinE(), integrator::start_, and systemDefinition::updateInstantTemp().

```
                                    {
int chunk = OMP_CHUNK;
if (start_) {
    try {
        cellList_cpu tmpCL (sys.box(), sys.rcut(), sys.
  rskin());
        cl_ = tmpCL;
    } catch (std::exception &e) {
        std::cerr << e.what() << std:: endl;
        throw customException("Failed to integrate on first
   step");
    }

    // get initial temperature
    calcForce(sys);
    float tmp=0.0, Uk = 0.0;
    #pragma omp parallel for reduction(+:Uk)
    for (unsigned int i = 0; i < sys.numAtoms(); ++i) {
        Uk += (sys.atoms[i].vel.x*sys.atoms[i].vel.x)+(sys.atoms
  [i].vel.y*sys.atoms[i].vel.y)+(sys.atoms[i].vel.z*sys.atoms[i].
  vel.z);
    }
    Uk *= sys.mass();
    tmp = Uk;
    Uk *= 0.5;
    tmp /= (3.0*(sys.numAtoms()-1.0));
    sys.updateInstantTemp(tmp);
    sys.setKinE(Uk);
    start_ = 0;
}

// (1) evolve particle velocities
#pragma omp parallel shared(sys)
{
    #pragma omp for schedule(dynamic,OMP_CHUNK)
    for (unsigned int i = 0; i < sys.numAtoms(); ++i) {
        sys.atoms[i].vel.x += 0.5*dt_*(sys.atoms[i].acc.x);
        sys.atoms[i].vel.y += 0.5*dt_*(sys.atoms[i].acc.y);
        sys.atoms[i].vel.z += 0.5*dt_*(sys.atoms[i].acc.z);
    }

    // (2) evolve particle positions
    #pragma omp for schedule(dynamic,OMP_CHUNK)
    for (unsigned int i = 0; i < sys.numAtoms(); ++i) {
        sys.atoms[i].pos.x += sys.atoms[i].vel.x*dt_;
        sys.atoms[i].pos.y += sys.atoms[i].vel.y*dt_;
        sys.atoms[i].pos.z += sys.atoms[i].vel.z*dt_;
```

```
        }
    }

    // (3) calc force
    calcForce(sys);

    // (4) evolve particle velocities
    #pragma omp parallel shared(sys)
    {
    #pragma omp for schedule(dynamic,OMP_CHUNK)
    for (unsigned int i = 0; i < sys.numAtoms(); ++i) {
            sys.atoms[i].vel.x += 0.5*dt_*(sys.atoms[i].acc.x);
            sys.atoms[i].vel.y += 0.5*dt_*(sys.atoms[i].acc.y);
            sys.atoms[i].vel.z += 0.5*dt_*(sys.atoms[i].acc.z);
    }
    }
    float Uk = 0.0;
    float tmp = 0.0;

    #pragma omp parallel for reduction(+:Uk)
    // get temperature and kinetic energy
    for (unsigned int i = 0; i < sys.numAtoms(); ++i) {
        Uk += (sys.atoms[i].vel.x*sys.atoms[i].vel.x)+(sys.atoms
      [i].vel.y*sys.atoms[i].vel.y)+(sys.atoms[i].vel.z*sys.atoms[i].
      vel.z);
    }

    Uk *= sys.mass();
    tmp = Uk;
    Uk *= 0.5;
    tmp /= (3.0*(sys.numAtoms()-1.0));
    sys.updateInstantTemp(tmp);
    sys.setKinE(Uk);

}
```

The documentation for this class was generated from the following files:

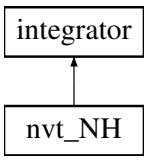
- /Users/nathanmahynski/CBEMDGPU/nve.h
- /Users/nathanmahynski/CBEMDGPU/nve.cpp

4.8 nvt_NH Class Reference

Uses Nose-Hover integration method to thermostat a system (constant T rather than E)

`#include <nvt.h>`

Inheritance diagram for nvt_NH:



Public Member Functions

- nvt_NH (const float Q)
- void step (systemDefinition &sys)
- void setTimestep (const float dt)

  Set the integrator timestep.
- void calcForce (systemDefinition &sys)

  Calculate the forces on each atom.

Protected Attributes

- cellList_cpu cl_

  Cell or neighbor list.
- std::vector< float3 > lastAccelerations_

  Acceleration of particles on previous timestep (useful for NVE integrator)
- float dt_

  Timestep size.
- int start_

  Flag for whether this object has been initialized or not.

4.8.1 Detailed Description

Uses Nose-Hover integration method to thermostat a system (constant T rather than E)

NVT integration with Nose-Hoover thermostat.

Author

Nathan A. Mahynski

Date

11/18/13

Definition at line 14 of file nvt.h.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 nvt_NH::nvt_NH (const float *Q*)

Do NVT integration with Nose-Hoover thermostat.

Date

11/18/13

Initialize integrator

Parameters

in	*Q*	Thermal mass for thermostat

Definition at line 20 of file nvt.cpp.

References integrator::start_.

```
                                {
    Q_ = Q;
    gamma_ = 0.0;
    start_ = 1;
}
```

4.8.3 Member Function Documentation

4.8.3.1 void integrator::calcForce (systemDefinition & *sys*) [inherited]

Calculate the forces on each atom.

Integration

Author

Nathan A. Mahynski

Date

11/19/13

Calculate the pairwise forces in a system. This also calculates the potential energy of a system. The kinetic energy is calculated during the verlet integration.

Parameters

in,out	*sys*	System definition

Calculate the pairwise forces in a system. This also calculates the potential energy of a system. The kinetic energy is calculated during the verlet integration.

Parameters

in,out	*sys*	System definition

Definition at line 23 of file integrator.cpp.

References systemDefinition::atoms, systemDefinition::box(), cellList_cpu::checkUpdate(), integrator::cl_, cellList_cpu::head(), cellList_cpu::list(), systemDefinition::mass(), cellList_cpu::nCells, cellList_cpu::neighbors(), systemDefinition::numAtoms(), systemDefinition::potential, systemDefinition::potentialArgs(), systemDefinition::rcut(), and systemDefinition::setPotE().

Referenced by nve::step(), and step().

```
                                                    {
    // For cache coeherency allocate new space for calculations
    float3 empty;
    empty.x = 0; empty.y = 0; empty.z = 0;
    std::vector <float3> acc (sys.numAtoms(), empty);
    const float rc = sys.rcut();

    // every time, check if the cell list needs to be updated first
    cl_.checkUpdate(sys);

    float Up = 0.0;
    const float3 box = sys.box();
    const float invMass = 1.0/sys.mass();

    // traverse cell list and calculate total system potential energy
    std::vector <float> args = sys.potentialArgs();
#pragma omp parallel for reduction(+:Up) schedule(dynamic, 1)
    for (unsigned int cellID = 0; cellID < cl_.nCells.x*cl_.nCells
      .y*cl_.nCells.z; ++cellID) {
        int atom1 = cl_.head(cellID);
        while (atom1 >= 0) {
            std::vector < int > neighbors = cl_.neighbors(cellID);
            for (int index = 0; index < neighbors.size(); ++index) {
                const int cellID2 = neighbors[index];
                int atom2 = cl_.head(cellID2);
                while (atom2 >= 0) {
                    if (atom1 > atom2) {
                        float3 pf;
                        Up += sys.potential (&sys.atoms[atom1].
      pos, &sys.atoms[atom2].pos, &pf, &box, &args[0], &rc);
                        acc[atom1].x -= pf.x*invMass;
                        acc[atom1].y -= pf.y*invMass;
                        acc[atom1].z -= pf.z*invMass;
                        acc[atom2].x += pf.x*invMass;
```

```
                    acc[atom2].y += pf.y*invMass;
                    acc[atom2].z += pf.z*invMass;
                }
                atom2 = cl_.list(atom2);
            }
        }
        atom1 = cl_.list(atom1);
    }
}

// save acceleration in array of atoms in system
#pragma omp parallel for
for (int i = 0; i < sys.numAtoms(); ++i) {
    sys.atoms[i].acc.x = -acc[i].x;
    sys.atoms[i].acc.y = -acc[i].y;
    sys.atoms[i].acc.z = -acc[i].z;
}

// set Up
sys.setPotE(Up);
}
```

4.8.3.2 void nvt_NH::step (systemDefinition & *sys*) [virtual]

Integrate a single timestep forward using Velocity-Verlet integration scheme. This is NOT identical since some intermediate bookkeeping needs to be handled for thermostat. Creates a cell list the first time it is called.

Parameters

in,out	*sys*	System definition

Implements integrator.

Definition at line 32 of file nvt.cpp.

References systemDefinition::atoms, systemDefinition::box(), integrator::calcForce(), integrator::cl_, integrator::dt_, systemDefinition::instantT(), systemDefinition::mass(), systemDefinition::numAtoms(), systemDefinition::rcut(), systemDefinition::rskin(), systemDefinition::setKinE(), integrator::start_, systemDefinition::targetT(), and systemDefinition::updateInstantTemp().

```
                                        {
    int chunk = OMP_CHUNK;
    if (start_) {
        try {
            cellList_cpu tmpCL (sys.box(), sys.rcut(), sys.
      rskin());
            cl_ = tmpCL;
        } catch (std::exception &e) {
            std::cerr << e.what() << std:: endl;
            throw customException("Failed to integrate on first
       step");
        }

        gamma_ = 0.0;
        for (unsigned int i = 0; i < sys.numAtoms(); ++i)  {
            gamma_ += (sys.atoms[i].vel.x*sys.atoms[i].vel.x)+(sys.
      atoms[i].vel.y*sys.atoms[i].vel.y)+(sys.atoms[i].vel.z*sys.atoms
      [i].vel.z);
        }
        gamma_ -= (3.0*(sys.numAtoms()-1.0))*sys.instantT();
        gamma_ /= Q_;

        // get initial temperature
    calcForce(sys);

        float tmp=0.0, Uk = 0.0;
        #pragma omp parallel for reduction(+:Uk)
        for (unsigned int i = 0; i < sys.numAtoms(); ++i) {
            Uk += (sys.atoms[i].vel.x*sys.atoms[i].vel.x)+(sys.atoms
      [i].vel.y*sys.atoms[i].vel.y)+(sys.atoms[i].vel.z*sys.atoms[i].
      vel.z);
        }
        Uk *= sys.mass();
        tmp = Uk;
        Uk *= 0.5;
        tmp /= (3.0*(sys.numAtoms()-1.0));
        sys.updateInstantTemp(tmp);
```

```
        sys.setKinE(Uk);
        start_ = 0;
        gammadot_ = 0.0;
        gammadd_ = 0.0;
    }

    tau2_ = Q_/ ((3.0*(sys.numAtoms()-1.0))*sys.targetT());
    gammadd_ = 1/tau2_*(sys.instantT()/sys.targetT()-1);

    // (1) update thermostat velocity and thermostat position
    // velocity half-step
    gammadot_ += dt_*0.5*gammadd_;

    // position step
    gamma_ += gammadot_*dt_;

    // (2) evolve particle velocities
    #pragma omp parallel shared(sys)
    {
        #pragma omp for schedule(dynamic,OMP_CHUNK)
        for (unsigned int i = 0; i < sys.numAtoms(); ++i) {
            sys.atoms[i].vel.x = sys.atoms[i].vel.x*exp(-gammadot_*
      dt_*0.5) + 0.5*dt_*(sys.atoms[i].acc.x);
            sys.atoms[i].vel.y = sys.atoms[i].vel.y*exp(-gammadot_*
      dt_*0.5) + 0.5*dt_*(sys.atoms[i].acc.y);
            sys.atoms[i].vel.z = sys.atoms[i].vel.z*exp(-gammadot_*
      dt_*0.5) + 0.5*dt_*(sys.atoms[i].acc.z);
        }

        // (3) evolve particle positions
        #pragma omp for schedule(dynamic,OMP_CHUNK)
        for (unsigned int i = 0; i < sys.numAtoms(); ++i) {
            sys.atoms[i].pos.x += sys.atoms[i].vel.x*dt_;
            sys.atoms[i].pos.y += sys.atoms[i].vel.y*dt_;
            sys.atoms[i].pos.z += sys.atoms[i].vel.z*dt_;
        }
    }

    // (4) calc force
    calcForce(sys);

    // (5) evolve particle velocities
    #pragma omp parallel shared(sys)
    {
    #pragma omp for schedule(dynamic,OMP_CHUNK)
    for (unsigned int i = 0; i < sys.numAtoms(); ++i) {
        sys.atoms[i].vel.x = (sys.atoms[i].vel.x+sys.atoms[i].
      acc.x*dt_*0.5)*exp(-gammadot_*dt_*0.5);
        sys.atoms[i].vel.y = (sys.atoms[i].vel.y+sys.atoms[i].
      acc.y*dt_*0.5)*exp(-gammadot_*dt_*0.5);
        sys.atoms[i].vel.z = (sys.atoms[i].vel.z+sys.atoms[i].
      acc.z*dt_*0.5)*exp(-gammadot_*dt_*0.5);
        }
    }
    float Uk = 0.0;
    float tmp = 0.0;

    #pragma omp parallel for reduction(+:Uk)
    // get temperature and kinetic energy
    for (unsigned int i = 0; i < sys.numAtoms(); ++i) {
        Uk += (sys.atoms[i].vel.x*sys.atoms[i].vel.x)+(sys.atoms
      [i].vel.y*sys.atoms[i].vel.y)+(sys.atoms[i].vel.z*sys.atoms[i].
      vel.z);
    }

    Uk *= sys.mass();
    tmp = Uk;
    Uk *= 0.5;
    tmp /= (3.0*(sys.numAtoms()-1.0));
    sys.updateInstantTemp(tmp);
    sys.setKinE(Uk);

    // (6) update thermostat velocity
    gammadd_ = 1/tau2_*(sys.instantT()/sys.targetT()-1);
    gammadot_ += dt_*0.5*gammadd_;
}
```

The documentation for this class was generated from the following files:

- /Users/nathanmahynski/CBEMDGPU/nvt.h
- /Users/nathanmahynski/CBEMDGPU/nvt.cpp

4.9 systemDefinition Class Reference

Contains all information pertaining to a system being simulated.

`#include <system.h>`

Public Member Functions

- void initRandom (const int N, const int rngSeed)
- void initThermal (const int N, const float Tset, const int rngSeed, const float dx)
- void updateInstantTemp (const float T)

  Manually set the instantaneous temperature.
- void setTemp (const float T)

  Assign the target temperature for NVT simulations.
- void setMass (const float m)

  Assign the mass of each particle.
- void setRcut (const float rc)

  Assign the cutoff radius of the pair potential.
- void setRskin (const float rs)

  Assign the skin radius as a buffer for the neighbor/cell lists.
- void setBox (const float x, const float y, const float z)

  Assign the simulation box size.
- void printBox ()

  Print the box dimesions to stdout.
- float3 box () const

  Report the box dimensions.
- float instantT () const

  Report the instantaneous temperature.
- float targetT () const

  Report the target temperature for NVT simulations.
- float mass () const

  Report the particle's mass.
- float PotE () const

  Report the instantaneous potential energy of the system.
- void setPotE (const float Up)

  Assign the potential energy.
- void setKinE (const float Uk)

  Assign the kinetic energy.
- float KinE () const

  Report the instantaneous kinetic energy of the system.
- float rskin () const

  Report the skin radius for the neighbor/cell lists.
- float rcut () const

  Report the pair potential function cutoff radius.
- void writeSnapshot ()

  Print a snapshot of the system.
- int numAtoms () const

  Report the pair potential function cutoff radius.
- void setPotentialArgs (const std::vector< float > args)

  Assign additional arguments to the pair potential function.
- std::vector< float > potentialArgs ()

  Report additional arguments to the pair potential function.
- void setPotential (pointFunction_t pp)

  Assign pair potential function.

Public Attributes

- pointFunction_t potential

  Pointer to pair potential function.
- std::vector< atom > atoms

  Vector of atoms in the system.

4.9.1 Detailed Description

Contains all information pertaining to a system being simulated.

System definition

Author

Nathan A. Mahynski

Date

11/17/13

Definition at line 17 of file system.h.

4.9.2 Member Function Documentation

4.9.2.1 void systemDefinition::initRandom (const int *N*, const int *rngSeed*)

Initialize a system of N atoms with random velocities and positions on a lattice. Net momeentum is automatically initialized to zero.

Parameters

in	*N*	Number of atoms to create
in	*rngSeed*	Random number generator seed

Definition at line 37 of file system.cpp.

References atoms.

```
                                                                    {
    float3 totMomentum;
    totMomentum.x = 0; totMomentum.y = 0; totMomentum.z = 0;

    if (N < 1) {
        throw customException ("N must be > 0");
        return;
    }

    srand(rngSeed);
    int chunk = OMP_CHUNK;
    unsigned int i;
    atoms.resize(N);
#pragma omp parallel private(i)
    {
#pragma omp for schedule(dynamic, chunk)
    for (i = 0; i < N; ++i) {
        if (i < N-1) {
            atoms[i].vel.x = (RNG-0.5);
            atoms[i].vel.y = (RNG-0.5);
            atoms[i].vel.z = (RNG-0.5);
            totMomentum.x += atoms[i].vel.x;
            totMomentum.y += atoms[i].vel.y;
            totMomentum.z += atoms[i].vel.z;
        } else {
            atoms[i].vel.x = -totMomentum.x;
            atoms[i].vel.y = -totMomentum.y;
```

```
            atoms[i].vel.z = -totMomentum.z;
        }
        atoms[i].pos.x = (RNG)*box_.x;
        atoms[i].pos.y = (RNG)*box_.y;
        atoms[i].pos.z = (RNG)*box_.z;
        atoms[i].acc.x = 0;
        atoms[i].acc.y = 0;
        atoms[i].acc.z = 0;
    }
    }
}
```

4.9.2.2 void systemDefinition::initThermal (const int *N*, const float *Tset*, const int *rngSeed*, const float *dx*)

Initialize a system of N atoms with random velocities to meet a desired temperature and positions on a lattice. Net momentum is automatically initialized to zero.

Parameters

in	*N*	Number of atoms to create
in	*rngSeed*	Random number generator seed

Definition at line 83 of file system.cpp.

References atoms.

```
                            {
    float3 totMomentum;
    totMomentum.x = 0; totMomentum.y = 0; totMomentum.z = 0;

    if (N < 1) {
    throw customException ("N must be > 0");
    return;
    }

    srand(rngSeed);
    atoms.resize(N);

    // maxwell boltzmann distribution has mean 0 stdev kT/m in each dimension
    typedef std::tr1::linear_congruential<int, 16807, 0, (int)((1U << 31) -1 )
      > Myceng;
    Myceng eng;
    float sig = sqrt(Tset/mass_);
    std::tr1::normal_distribution<float> distribution(0.0,sig);
    float rannum = 0.0;
    float tmpT = 0.0;
    int index = 0;
    const int xs = floor(box_.x/dx);
    const int ys = floor(box_.y/dx);
    const int zs = floor(box_.z/dx);

    // initialize particle positions on a simple cubic lattice
    for (unsigned int x = 0; x < xs; ++x) {
        for (unsigned int y = 0; y < ys; ++y) {
            for (unsigned int z = 0; z < zs; ++z) {
                if (index < N) {
                    atoms[index].pos.x = x*dx;
                    atoms[index].pos.y = y*dx;
                    atoms[index].pos.z = z*dx;
                } else {
                    x = xs;
                    y = ys;
                    z = zs;
                    break;
                }
                index++;
            }
        }
    }
    for (unsigned int i = 0; i < N; ++i) {
    atoms[i].acc.x = 0;
    atoms[i].acc.y = 0;
    atoms[i].acc.z = 0;
    if (i < N-1) {
        atoms[i].vel.x = distribution(eng);
        atoms[i].vel.y = distribution(eng);
        atoms[i].vel.z = distribution(eng);
```

```
        totMomentum.x += atoms[i].vel.x;
        totMomentum.y += atoms[i].vel.y;
        totMomentum.z += atoms[i].vel.z;
    } else {
        atoms[i].vel.x = -totMomentum.x;
        atoms[i].vel.y = -totMomentum.y;
        atoms[i].vel.z = -totMomentum.z;
    }
    tmpT += (atoms[i].vel.x*atoms[i].vel.x + atoms[i].vel.y*
      atoms[i].vel.y + atoms[i].vel.z*atoms[i].vel.z);
    }

    // do velocity rescaling to get exactly the right T
    tmpT *= mass_/(3.0*(N-1));
    tmpT /= Tset;
    tmpT = 1.0/tmpT;
    for (unsigned int i = 0; i < N; i ++ ) {
    atoms[i].vel.x = atoms[i].vel.x*sqrt(tmpT);
    atoms[i].vel.y = atoms[i].vel.y*sqrt(tmpT);
    atoms[i].vel.z = atoms[i].vel.z*sqrt(tmpT);
    }
}
```

4.9.2.3 void systemDefinition::setPotential (pointFunction_t *pp*)

Assign pair potential function.

Sets the "host" pair potential function which also sets the GPU equivalent in integrator.cu if using CUDA.

Parameters

in	*pp*	Pointer to pair potential function

Definition at line 21 of file system.cpp.

References potential.

```
                                                    {
/*#ifdef NVCC
    dev_potential = pp;
    cudaMemcpyFromSymbol(&potential, dev_potential, sizeof(pointFunction_t));
#else*/
    potential = pp;
//#endif
}
```

4.9.2.4 void systemDefinition::writeSnapshot ( )

Print a snapshot of the system.

Write instantaneous snapshot of the system to a file called "trajectory.xyz" This file is appended not overwritten consecutively.

Definition at line 159 of file system.cpp.

References atoms.

```
                                        {
    static int snapNum = 0;
    if (snapFile_ == NULL) {
        snapFile_ = fopen("trajectory.xyz", "w");
    }

    fprintf(snapFile_, "%d\nSnapshot #%d\n", atoms.size(), snapNum);

    for (unsigned int i = 0; i < atoms.size(); ++i) {
        fprintf(snapFile_, "%s\t%g\t%g\t%g\n", "A", atoms[i].pos.x, atoms
      [i].pos.y, atoms[i].pos.z);
    }

    snapNum++;
}
```

The documentation for this class was generated from the following files:

- /Users/nathanmahynski/CBEMDGPU/system.h
- /Users/nathanmahynski/CBEMDGPU/system.cpp

4.10 systemProps Class Reference

Holds all information about the CPU and GPU the simulation is being performed on.

`#include <cudaHelper.h>`

Public Member Functions

- **systemProps** (std::string fname)
- void displayAllProps ()

  Display cpu host and device (if using GPUs) properties.
- __host__ void displayCudaProps ()

  Display the properties of the GPU being used.
- void displayHost ()

  Display host name and other properties.
- int maxThreadsPerBlock (const int devID)

  Returns the maximum number of threads per block on the GPU.
- int maxGridDimX (const int devID)

  Returns the maximum number of blocks per grid on the GPU.
- int numDevices () const

  Returns the number of GPUs found attached to the CPU.

4.10.1 Detailed Description

Holds all information about the CPU and GPU the simulation is being performed on.

Functions to assist in using CUDA and/or GPUs

Author

Nathan A. Mahynski

Date

11/21/13

Definition at line 26 of file cudaHelper.h.

4.10.2 Member Function Documentation

4.10.2.1 __host__ void systemProps::displayCudaProps ( )

Display the properties of the GPU being used.

Display properties of all CUDA capable devices

Definition at line 23 of file cudaHelper.cu.

Referenced by displayAllProps().

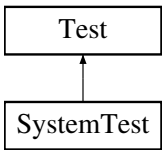
```
                                        {
    checkCudaDevices_();

    for (unsigned int i = 0; i < device_.size(); ++i) {
        *output_ << "# > Device Name: " <<  device_[i].name << endl;
        *output_ << "# ----------------------------------" << endl;
        *output_ << "# Total Global Memory: " << device_[i].totalGlobalMem/1024
       << " KB" << endl;
        *output_ << "# Shared Memory available per Block: " << device_[i].
      sharedMemPerBlock/1024 << " KB" << endl;
        *output_ << "# Registers per Thread Block: " << device_[i].regsPerBlock
       << endl;
        *output_ << "# Warp Size: " << device_[i].warpSize << endl;
        *output_ << "# Memory Pitch: " << device_[i].memPitch << endl;
        *output_ << "# Maximum Threads per Block: " << device_[i].
      maxThreadsPerBlock << endl;
        *output_ << "# Maximum Thread Dimensions (Block): " << device_[i].
      maxThreadsDim[0] << "x" << device_[i].maxThreadsDim[1] << "x" << device_[i].
      maxThreadsDim[2] << endl;
        *output_ << "# Maximum Thread Dimensions (Grid): " << device_[i].
      maxGridSize[0] << "x" << device_[i].maxGridSize[1] << "x" << device_[i].maxGridSize[2]
       << endl;
        *output_ << "# Total Constant Memory: " << device_[i].totalConstMem <<
      " bytes" << endl;
        *output_ << "# CUDA version: " << device_[i].major << "." << device_[i]
      .minor << endl;
        *output_ << "# Clock Rate: " << device_[i].clockRate << " kHz" << endl;
        *output_ << "# Texture Alignment: " << device_[i].textureAlignment << "
       bytes"<< endl;
        if (device_[i].deviceOverlap == 0) {
            *output_ << "# Device Overlap: Not Allowed" << endl;
        } else {
            *output_ << "# Device Overlap: Allowed" << endl;
        }
        *output_ << "# Number of Multiprocessors: " << device_[i].
      multiProcessorCount << endl;
        *output_ << "# ----------------------------------"<< endl;
    }
}
```

The documentation for this class was generated from the following files:

- /Users/nathanmahynski/CBEMDGPU/cudaHelper.h
- /Users/nathanmahynski/CBEMDGPU/cudaHelper.cu

4.11 SystemTest Class Reference

Inheritance diagram for SystemTest:



Protected Member Functions

- virtual void **SetUp** ()

Protected Attributes

- systemDefinition **a**
- int **natoms**
- float **mass**
- float **L**
- float **eps**

- float **sigma**
- nvt_NH **integrate**

4.11.1 Detailed Description

Definition at line 11 of file unittests.cpp.

The documentation for this class was generated from the following file:

- /Users/nathanmahynski/CBEMDGPU/unittests.cpp