Matrix multiplication and L3 cache

From last week

\[
\begin{array}{c}
\text{\(A_{ij}\)} \\
\vdots \\
\text{\(C_{ij} = A_{ij} \cdot B_{ij}\)} \\
\end{array}
\]

Triply-nested loop: for each column of A, cycle through all the rows of B to compute a column of C.
A simple cache-eviction policy: oldest (i.e. least-recently used) data gets overwritten when you run out of space.
Two possibilities: if B fits in cache, read it from main memory once. If not, you are always always reading it from main memory.
Matrix multiplication and cache

Plot of performance as a function of (size of one matrix)/(size of L3). Here, the size of L3 is 3 MB.

This is actually pretty good for a simple cartoon of cache behavior. Theorem says that you will always have this behavior for triply-nested loops.
Homework 4

PIC methods for vorticity.

• Algorithmic issues
• Software / implementation.
Regularized particle methods

Particle method for vorticity transport in 2D.

\[ \frac{dx^k}{dt} = \vec{U}(x^k(t)) \quad \omega^k = \omega(x^k(0), 0)h^D \]

\[ \vec{U}(x) = \sum_k \omega^k \nabla_x \frac{1}{x} \int G(x - y) \zeta_\epsilon(y) dy \]

\[ = \sum_k \omega^k \vec{U}_\epsilon(x - x_k) \]

\[ U_\epsilon(z) = \nabla_z \left( \int G(z - y) \zeta_\epsilon(y) dy \right) \]

\[ G(z) = \frac{1}{2\pi} \log(|z|) \]

\[ \nabla_x \frac{1}{x} = \left( \frac{\partial}{\partial x_2}, -\frac{\partial}{\partial x_1} \right) \]

\[ \zeta_\epsilon(z) = \frac{1}{\epsilon D} \zeta \left( \frac{z}{\epsilon} \right) \]

For epsilon = 0, we get a point vortex method, with a singular velocity field, so some sort of regularization (epsilon > 0) is required.
Regularized particle methods

\[ \zeta(z) = \frac{(-140(1 - |z|)^3 + 420(1 - |z|)^4 - 252(1 - |z|)^5)}{2\pi} \]
Can think of this as regularizing the potential induced by a collection of delta functions.

\[
\omega(x) = \sum_k \omega_k \delta(x - x_k)
\]

\[
\Psi(x) = (G \ast \omega)(x) = \sum_k \omega_k G(x - x_k),
\]

\[
\Psi(x) = (G_\epsilon \ast \omega)(x) = \sum_k \omega_k G_\epsilon(x - x_k)
\]

\[
U = \nabla^\perp \Psi
\]
Particle-in-Cell (PIC)

Start with a collection of particles, each of which induces a contribution to the stream function in any point in space via convolution.

PIC: Replace continuous convolution with discrete convolution on a rectangular grid:
- Replace $\omega_k \delta(x - x_k) \rightarrow \omega_i^+ = \omega_k \delta^h(ih - x_k)$
- Compute stream function on the grid as discrete convolution.
- Interpolate finite-difference approximation to velocity on the grid to particles.
Given the particle locations, can compute a velocity field at those locations.

\[ \omega_i = \sum_k \omega^k \delta_h (i h - x^k), \quad \omega^k = \omega^k (x^k(0), 0) \]

\[ \psi_i = \sum_j G(i h - j h) \omega_j \]

\[ G(z) = \frac{1}{2\pi} \log(|z|) \]

\[ \vec{U}_i = D^\perp (\psi)_i \]

\[ (D_1 \psi) = \frac{\psi_{i+1,0} - \psi_{i-1,0}}{2h}, \quad (D_2 \psi) = \frac{\psi_{i+0,1} - \psi_{i-0,1}}{2h} \]

\[ U(x^k) = \sum_i U_i \delta_h (x^k - i h) \]

\[ \delta_h (z) = \delta \left( \frac{z_1}{h} \right) \delta \left( \frac{z_2}{h} \right) \]
Particle-in-Cell (PIC)

\[ U(x^k) = \sum_i U_i \delta_h(x^k - i h) \]

\[ \delta_h(z) = \delta \left( \frac{z_1}{h} \right) \delta \left( \frac{z_2}{h} \right) \]
Algorithmic design considerations

• Initialization: particles deposited on a grid with spacing $h_p$.

• What is the relationship between epsilon, $h$, $h_p$? Our choices: $h_p = h/n_p$, $n_p$ a positive integer, epsilon = $h$.

• There is a good convergence theory for vortex methods (i.e. no grid). The grid introduces its own short-range regularization. Can we eliminate that, while still retaining the (fast) grid representation of the long-range effects? (Answer: yes - PPPM).
We will provide an implementation of Hockney’s algorithm for computing

$$\psi_i = \sum_j G_\epsilon (ih - jh) \omega_j$$

as well as the templated RK4 class.

You will implement the rest of the particle method:
- Deposition of charge and calculation of velocity field (template class $F$).
- The template classes $x$, $dx$ appropriate to the PIC method.
template <class X, class F, class dX> class RK4
{
  public:  void advance(double a_time, double a_dt, X& a_state);
  protected:
    dX m_k;
    dX m_delta;
    F m_f;
};
Particle Classes

class DX // displacement.
{
public:

    DX()
    {
    ...}

    array<Real, DIM> m_x;

    inline void increment(double a_scale, const DX& a_rhs)
    {
    ...}

    inline void operator*=(double a_scale){...};
};
Particle Classes

class Particle
{
public:
    array<Real, DIM> m_x;
    Real strength;
    // (*this).m_x -> (*this).m_x + a_shift.m_x;
    inline void increment(const DX& a_shift)
    {
        ...
    }
};
Particle Classes

class ParticleSet; // Forward declaration.
    // Needed to avoid circular dependency.

class ParticleShift // Corresponds to dX in RK4.
{
public:
    vector<DX> m_particles;
    void init(const ParticleSet& a_particles);
    void increment(double a_scale, const ParticleShift& a_rhs);
    void operator*=(double a_scale);
    void setToZero();
};

All that the ParticleShift declaration is allowed to know about ParticleSet is that it is a class. Pointers, references, but not values (why?).
class ParticleSet

// Corresponds to template class X in RK4.
{
Public:

  ParticleSet(shared_ptr<ConvKernel>& a_kerptr,
               Box& a_box,
               double& a_h,
               array<Real, DIM>& a_lowCorner,
               int a_M);

  ParticleSet();

  vector<Particle> m_particles;
  double m_h;
  Box m_box;
  array<Real, DIM> m_lowCorner;
  Hockney m_hockney;

  void increment(const ParticleShift& a_shift);
};
Particle Classes

class ParticleVelocities // Implements template class F
{
    public:
        ParticleVelocities();
        void operator()(ParticleShift& a_k,
                        const Real& a_time, const Real& dt,
                        ParticleSet& a_state);
};

ParticleVelocities has no state, only a default constructor. All objects required to to evaluate the velocity must come in via a_state. This is one of the reasons why all of the member data of ParticleSet are public (alternative: friend classes).
Hockney Class

class Hockney
{
    public:
        Hockney();
        Hockney(shared_ptr<ConvKernel>& a_kerPtr, const double& a_h, int a_M);
        void define(shared_ptr<ConvKernel>& a_kerPtr, const double& a_h, int a_M);
        void convolve(RectMDArray<Real>& a_rhs);
        ~Hockney(){};

    protected:
        ...

};

You will have to define your Hockney in the ParticleSet constructor. a_kerptr, a_M, and a_h are passed through to you via the arguments. m_hockney.convolve(...) called in ParticleVelocities.
template <class X, class F, class dX>
void RK4<X,F,dX>::advance(double a_time, double a_dt, X& a_state) {
    m_delta.init(a_state);          // Initialize dx classes from a_state.
    m_k.init(a_state);
    m_f(m_k, a_time, a_dt, a_state, m_k);  // compute k1
    m_delta.increment(sixth, m_k);  m_k*=half;
    m_f(m_k, a_time+half*a_dt, a_dt, a_state, m_k); // compute k2
    m_delta.increment(third, m_k);  m_k*=half;
    m_f(m_k, a_time+half*a_dt, a_dt, a_state, m_k); // compute k3
    m_delta.increment(third, m_k);
    m_f(m_k, a_time+a_dt, a_dt, a_state, m_k); // compute k4
    m_delta.increment(sixth, m_k);
    a_state.increment(m_delta);
}
Test Problems

- $T = 0.00000 \quad Eff = 0.752$
- $T = 2.50000 \quad Eff = 0.794$
- $T = 5.00000 \quad Eff = 0.768$
- $T = 10.00000 \quad Eff = 0.852$
- $T = 15.00000 \quad Eff = 0.825$
- $T = 20.00000 \quad Eff = 0.775$
- $T = 27.50000 \quad Eff = 0.755$
- $T = 37.50000 \quad Eff = 0.823$
- $T = 50.00000 \quad Eff = 0.872$
Visit for Particles

From the directory you ran vortex2d.exe:

• Visit
  - open -> PART*.vtk (database)
  - controls -> view -> 2D -> set viewport, window to 0 1 0 1, apply, dismiss
  - add -> mesh -> mesh
  - draw
  - For small number of particles: click on mesh, change point type to sphere, pixels to 10
Top-level directory structure:

```
src/
  src/RectMDArray/
  src/fftTools/ - Contains only FFT1D, FFT1DW, FFTMD
  src/Particles/ - .H only - you will put your .cpp files here
  src/Hockney/
utils/
  utils/Writers/
  utils/timer/
exec/
```
exec/

    exec/GNUmakefile
    exec/vortexTest.cpp
    exec/o2d/
    exec/d2d/

Makefile will create o2d/, d2d/ directories.
- o2d/ will contain the .o files generated in the build process.
- d2d/ will contain the .d files (dependencies) generated in the build process.
GNUmakefile

HOME = ./..
WRITERS = $(HOME)/utils/Writers
RMDA = $(HOME)/src/RectMDArray
FFT = $(HOME)/src/fftTools
CONV = $(HOME)/src/Hockney
PARTICLES = $(HOME)/src/Particles
TIMER = $(HOME)/utils/timer

VPATH= . $(HOME) $(PARTICLES) $(RMDA) $(FFT) $(CONV) $(TIMER) $(WRITERS)  

Tells Gmake where to search for file dependencies.
DIM=2

CXX=clang++

FFTWDIR = /usr/local

CFLAGS = -O3
CFLAGS += -std=c++11 -I/ -I$(PARTICLES) -I$(RMDA) -I$(FFT) -I$(CONV) -I$(TIMER) -I$(WRITERS) -I$(FFTWDIR)/include

Tells the compiler where to look for #include files

CFLAGS += -D DIM=$(DIM)
GNUmakefile

odir = ./o.$(DIM)d
ddir = ./d.$(DIM)d

LIBS:= -L$(FFTWDIR)/lib -lfftw3

SRCFILES:= $(notdir $(wildcard $(TIMER)/*.cpp $(RMDA)/
*.cpp $(WRITERS)/*.cpp $(FFT)/*.cpp ..//*.
cpp $(CONV)/*.cpp
$(PARTICLES)/*.cpp))
This defines our .cpp sources. notdir strips off the directory part of each
element in the list.

OBJE=$($(patsubst %.cpp,$(odir)/%.o,$(SRCFILES))
This defines our object files. With this definition, make will look for them in
./$(odir)

DEPS:=$(patsubst $(odir)/%.o,$(ddir)/%.d, $(OBJE))
This defines our dependency files to be stored in $(ddir). More on them
shortly.
GNUmakefile

odir = ./o.$(DIM)d
ddir = ./d.$(DIM)d

LIBS:= -L$(FFTWDIR)/lib -lfftw3

SRCFILES:= $(notdir $(wildcard $(TIMER)/*.cpp $(RMDA)/*.cpp $(Writers)/*.cpp $(FFT)/*.cpp .//*.cpp $(CONV)/*.cpp $(Particles)//*.cpp))
This defines our .cpp sources. notdir strips off the directory part of each element in the list.

OBJ:= $(patsubst %.cpp, $(odir)/%.o, $(SRCFILES))
This defines our object files. With this definition, make will look for them in ./$(odir)

DEPS:=$(patsubst $(odir)/%.o, $(ddir)/%.d, $(OBJ))
This defines our dependency files to be stored in $(ddir). More on them shortly.
Rule for making `.o` files.

$(odir)/%.o: %.cpp GNUmakefile

```
<TAB>  mkdir -p $(odir); \\n  $(CXX) -c $(CFLAGS) $< -o $@
<TAB>  mkdir -p $(ddir); \
  $(CXX) -MM $(CFLAGS) $< \
      | sed '1s/^/o.$(DIM)d\//' > $*.d ;\n      mv $*.d $(ddir)
```

For each file of the form `%.cpp` accessible in VPATH, it makes a target of the form `$(odir)/%.o`.

```
$(CXX) -c : compile only, output object file to argument following `-o`.
```

In this case, it is `$@`.

This rule also makes a file of the form `$*.d` and moves it to `$(ddir)`. ($* is the stem of `%.cpp`).

```
mkdir -p creates a directory only if there isn't one.
```

Questions:

What do the `.d` files do for us?

What's that stuff in the middle?
Rule for making .o files.

-include $(DEPS) — includes all of the files in $(DEPS). In this case, that is all files of the form d.2d/*.*. In this case, that is all files of the form d.2d/Box.d. Placed at the end of the makefile. Each file is a makefile, with one rule in it.

Let’s look at d.2d/Box.d:

```
o.2d/DBox.o: ../src/RectMDArray/DBox.cpp ../src/RectMDArray/DBox.H \n    ../src/RectMDArray/Point.H ../src/RectMDArray/PointImplem.H
```

It is a rule for making o.2d/DBox.o. But it has no recipes. The effect of this is to add dependencies to already-existing ones (in this case, for o.2d/DBox.o).
Rule for making .o files.

<TAB> mkdir -p $(ddir);$(CXX) -MM $(CFLAGS) $< | sed '1s/^/o. $(DIM)d\///' > $*.d;mv $*.d $(ddir)

- | : take output from stdout and stream it to stdin for the next program.
- sed : streaming editor for Unix systems. Uses similar commands to vi.
- > : take output from stdout and send it to file on rhs.
- mv : move the file.
Rule for building vortex2D.exe

vortex2D: GNUmakefile $(OBJ)
   $(CXX) $(CFLAGS) $(OBJ) $(LIBS) -o vortex$(DIM)D.exe

- It depends on the makefile and everything in the $(OBJ) list.
- The rule is to run the compiler with no .cpp files, so all it is doing is linking the object files (so main had better be there).
Housekeeping, utilities.

clean:
<TAB> rm -r *.exe $(odir) $(ddir)

listsrrc:
<TAB> @echo $(SRCFILES)

listobj:
<TAB> @echo $(OBJSS)

listdep:
<TAB> @echo $(DEPS)
Results Demo