N-Body Example Code

Code Optimization Workshop | 27.6.2022 | Jonathan Coles

- What is the time evolution of *N* bodies with non-zero mass under the influence of their own mutual (Newtonian) gravitational effect?
- Hugely relevant for astrophysical applications, even without considering effects from general relativity
- Used in simulations of star clusters, dark matter, and galaxy evolution, for example.
- Some of the largest simulations have over a trillion (10^{12}) particles (bodies).

The N-Body force equations

- Momentum is defined as mass times velocity.

$$p = mv$$

- Newton defined force as equal to the change in momentum *p* over time.

$$\vec{F} = \frac{dp}{dt} = \frac{d}{dt}(mv)$$

- For a fixed, unchanging mass, we get the famous equation

$$\vec{F} = ma$$

The N-Body force equations

- Newton also claimed that the force (vector) between two objects with masses m1 and m2 is proportional to the product of those masses and the square of the distance between them

$$\vec{F} = -G\frac{m_1 m_2}{r^2}\hat{r}$$

- Otherwise written as

$$\vec{F} = -G\frac{m_1 m_2}{r^3}\vec{r}$$

The N-Body force equations

- For *N* bodies (particles) we can write this as

$$F_{ij} = -G\frac{m_i m_j}{r_{ij}^3}\vec{r_{ij}}$$

- Gravity is additive, so for any given particle the total force on that particle is

$$\vec{F_i} = -G\sum_{i \neq j}\frac{m_i m_j}{r_{ij}^3}\vec{r_{ij}}$$

The N-Body force equations

- Substituting in our original definition of *F* we can derive a formula for the acceleration of a particle

$$\vec{F_i} = m_i \vec{a_i} = -G \sum_{i \neq j} \frac{m_i m_j}{r_{ij}^3} \vec{r_{ij}}$$

$$\vec{a_i} = -G \sum_{i \neq j} \frac{m_j}{r_{ij}^3} \vec{r_{ij}}$$

The N-Body force equations

- To avoid problems with close encounters that could lead to numerical instability, we will treat each particle as though it is a small cloud.
- We add a fixed "softening" parameter epsilon.

$$\vec{a}_i = -G \sum_{i \neq j} \frac{m_j}{(r_{ij}^2 + \epsilon^2)^{3/2}} \vec{r_{ij}}$$

The N-Body integration

- Now that we have an expression for acceleration, we can use it to evolve our particles with a simple first-order integrator.

$$\vec{a} = A(m_0, ..., m_{N-1}, \vec{r}_0, ..., \vec{r}_{N-1})$$

$$\vec{v}' = \vec{v} + \vec{a}\Delta t$$

$$\vec{x}' = \vec{x} + v'\Delta t$$

The N-Body integration

- The first order integrator unfortunately has terrible energy conserving properties.
- Many N-body codes use the second order Leap Frop integrator.
    - Symplectic - preserves a Hamiltonian and is time reversible if time step is fixed.
- There are several different formulations but here we will just look at the kick-drift-kick form

- A "kick" is when the velocity is updated

$$\vec{v}' = \vec{v} + \vec{a}\Delta t$$

- A "drift" is when the position is updated

$$\vec{x}' = \vec{x} + v'\Delta t$$

Kick-Drift-Kick Integration

- The name Leap Frog refers to how the calculation of kick and drift leap frog over each other at the time point at which they are evaluated.

- A single compete iteration looks like this:

Kick $$\vec{v}_{i,\tau+1/2} = \vec{v}_{i,\tau} + \vec{a}_{i,\tau}\frac{\Delta t}{2}$$

Drift $$\vec{x}_{i,\tau+1} = \vec{x}_{i,\tau} + v_{i,\tau+1/2}\Delta t$$

Acceleration $$\vec{a}_{i,\tau+1} = A(m_0, ..., m_{N-1}, \vec{r}_0, ..., \vec{r}_{N-1})$$

Kick $$\vec{v}_{i,\tau+1} = \vec{v}_{i,\tau+1/2} + \vec{a}_{i,\tau+1}\frac{\Delta t}{2}$$

NBody Example Code

N-body code example

- For this workshop, we will use a simple N-body code to demonstrate some optimization methods.
- The code provides a framework for running a simulation and collecting statistics.

```
$ ./nbody --help
Usage: nbody [OPTION...] ARG1 ARG2
A simple N-Body gravity simulator for the LRZ Code Optimization Course.

      --dt=REAL              The largest time step to take.
      --ic=NAME              The NAME of the initial conditions to generate:
                             circular (default), random.
  -N INT                     The number of particles.
  -o, --output=TAG           Write output to files with TAG prefix.
      --output-freq=INT      Write simulation output every INT steps.
      --overwrite[=0|1]      Overwrite all output files (=1) or simply append
                             (=0, default).
      --rsoft=REAL           Softening radius.
      --run-perf             Run performance measurements of the simulation
                             code.
      --run-sim[=small|medium|large|huge]
                             Run the simulation using the command line
                             parameters (default) or run one of the four
                             examples.
      --run-tests            Run the test suite before the simulation.
      --seed=REAL|time       Random number seed. Specify 'time' to use current
                             time (default).
      --status-freq=INT      Display a status report every INT steps.
      --status-output=NAME   Write status output to file NAME.
      --steps=INT            Number of steps to advance the simulation.
  -v, --verbose              Produce verbose output
  -?, --help                 Give this help list
      --usage                Give a short usage message
  -V, --version              Print program version
```

N-body code example

```
$ ./nbody-test --run-sim=small

==============================================================================

  N-Body Simulator for PRACE Code Optimization Course 2022

==============================================================================

 Initial Conditions:     Random
 Number of bodies:       256
 Timestep:               0.001
 Number of steps:        1000
 Total simulation time:  1
 Softening radius:       0.01
 Random number seed:     1


 Output tag:             -none-
 Status output:          -stdout-
 Log output:             -stdout-
 Status frequency:       1001 steps
 Output frequency:       1001 steps

 Real data type:         double  (8 bytes)

------------------------------------------------------------------------------
 STEP:         0
 E:      0.02348531  KE:       0.496491984605561  PE:      -0.473006675552114    E/E0:       1.000000000000000
Starting simulation.
Stopping simulation.

  Simulation statistics
------------------------------------------------------------------------------

       Total Time(s)     No. Calls     Time/Call(s)     % of Total
       -------------     ---------     ------------     ----------
 Kick:    7.20000e-04         2000      3.60000e-07           0.22
 Dift:    3.40000e-04         1000      3.40000e-07           0.11
 Accel:   3.22388e-01         1001      3.22066e-04          99.57
 I/O:     2.10000e-04         1001      2.09790e-07           0.06
 Total:   3.23780e-01
```

N-body code example

```
$ ./nbody-test --run-sim=small --compute-energy

===============================================================================

  N-Body Simulator for PRACE Code Optimization Course 2022

===============================================================================

 Initial Conditions:    Random
 Number of bodies:      256
 Timestep:              0.001
 Number of steps:       1000
 Total simulation time: 1
 Softening radius:      0.01
 Random number seed:    1


 Output tag:            -none-
 Status output:         -stdout-
 Log output:            -stdout-
 Status frequency:      1001 steps
 Output frequency:      1001 steps

 Real data type:        double  (8 bytes)

-------------------------------------------------------------------------------
 STEP:         0
 E:      0.02348531  KE:       0.496491984605561  PE:      -0.473006675552114    E/E0:       1.000000000000000
Starting simulation.
Stopping simulation.

  Simulation statistics
-------------------------------------------------------------------------------

       Total Time(s)     No. Calls     Time/Call(s)     % of Total
       -------------     ---------     ------------     ----------
 Kick:    7.20000e-04         2000      3.60000e-07           0.22
 Dift:    3.40000e-04         1000      3.40000e-07           0.11
 Accel:   3.22388e-01         1001      3.22066e-04          99.57
 I/O:     2.10000e-04         1001      2.09790e-07           0.06
 Total:   3.23780e-01
```

N-body code example

- We will focus on the routines that implement drift, kick, and acceleration.
- Drift and kick are both O(*N*).

```
struct vec3
{
    real x,y,z;
};
```

```
struct particle
{
    real m;
    struct vec3 r,v,a;
};
```

```
void kick(real dt, int N, struct particle *p)
{
    for (int i=0; i < N; i++)
    {
        p[i].v.x += p[i].a.x * dt;
        p[i].v.y += p[i].a.y * dt;
        p[i].v.z += p[i].a.z * dt;
    }
}
```

```
void drift(real dt, int N, struct particle *p)
{
    for (int i=0; i < N; i++)
    {
        p[i].r.x += p[i].v.x * dt;
        p[i].r.y += p[i].v.y * dt;
        p[i].r.z += p[i].v.z * dt;
    }
}
```

N-body code example

- Acceleration is implemented with an $O(N^2)$ algorithm. We will make many changes here.

```
void accel(int N, struct particle *p, real rsoft)
{
    for (int i=0; i < N; i++)
    {
        p[i].a.x = 0.0;
        p[i].a.y = 0.0;
        p[i].a.z = 0.0;

        for (int j=0; j < N; j++)
        {
            if (i==j) continue;

            real dx = p[i].r.x - p[j].r.x;
            real dy = p[i].r.y - p[j].r.y;
            real dz = p[i].r.z - p[j].r.z;
            real ir = RSQRT(dx*dx + dy*dy + dz*dz + rsoft);

            p[i].a.x -= p[j].m * dx * ir * ir * ir;
            p[i].a.y -= p[j].m * dy * ir * ir * ir;
            p[i].a.z -= p[j].m * dz * ir * ir * ir;
        }
    }
}
```

N-body code example

- in `nb-aos-support.c:run_simulation()`

```
real      dt = simopts->dt;
real half_dt = simopts->dt * 0.5;

accel(simopts->npart, sim->p, simopts->rsoft);

for (int istep = 1; istep <= simopts->nsteps; istep++)
{
    kick(half_dt, simopts->npart, sim->p);
    drift(dt, simopts->npart, sim->p));
    accel(simopts->npart, sim->p, simopts->rsoft);
    kick(half_dt, simopts->npart, sim->p);

    perform_requested_output(simopts, sim, istep);
}
```

Unpack NBody example code

```
$ tar xvzf COW-Code.tar.gz
```

N-body code course files

```
$ tar xzf COW-Code.tar.gz
$ ls -1 COW-Code
Makefile
nb-aos-alloc.c
nb-aos-data-layout.h
nb-aos-kda.c
nb-aos-support.c
nb-aos-tests.c
nb-aos.h
nb-main.c
nb-soa-alloc.c
nb-soa-data-layout.h
nb-soa-kda.c
nb-soa-support.c
nb-soa-tests.c
nb-soa.h
nb-support.h
nb.h
plot-perf.py
unity.c
unity.h
unity_internals.h
ver0
ver1
ver2
ver3
ver4
```

General code

Unit testing support

Array of Structures

Structure of Arrays

Plot performance measurements

Directories for successive optimizations

N-body code course files

```
$ ls -1 ver0/
compile_and_run
nb-aos-alloc.c
nb-aos-kda-ORIG.c
nb-aos-kda-TRI.c
nb-aos-kda.c
```

- `compile_and_run` will run `make` and submit a job to run the executable on a node.
  - Make sure to set `SLURM_RESERVATION` before using.
- Local source files will override those from the parent directory.

```
$ export SLURM_RESERVATION=hcow1s22
$ cd ver0
$ ./compile_and_run
icc -I.. -DUSE_AOS -O0 -DFLAGS=aos,double -I.. -fno-inline-functions -qopt-report=5 -qopt-report-phase=vec -qopt-report-phase=openmp -qopt-report-routine=kick,drift,accel -qopt-report-file=stdout  -DUNITY_OUTPUT_COLOR -DUNITY_INCLUDE_DOUBLE ../nb-aos-tests.c ../nb-main.c ../nb-aos-support.c nb-aos-alloc.c nb-aos-kda.c ../unity.c -o nbody-aos-ver0-dp-noopt > nbody-aos-ver0-dp-noopt.optrpt
icc -I.. -DUSE_AOS -O3 -xHost -DFLAGS=aos,O3,arch,double -I.. -fno-inline-functions -qopt-report=5 -qopt-report-phase=vec -qopt-report-phase=openmp -qopt-report-routine=kick,drift,accel -qopt-report-file=stdout  -DUNITY_OUTPUT_COLOR -DUNITY_INCLUDE_DOUBLE ../nb-aos-tests.c ../nb-main.c ../nb-aos-support.c nb-aos-alloc.c nb-aos-kda.c ../unity.c -o nbody-aos-ver0-dp > nbody-aos-ver0-dp.optrpt
icc -I.. -DUSE_AOS -O3 -xHost -DUSE_FLOAT -DFLAGS=aos,O3,arch,float -I.. -fno-inline-functions -qopt-report=5 -qopt-report-phase=vec -qopt-report-phase=openmp -qopt-report-routine=kick,drift,accel -qopt-report-file=stdout -DUNITY_OUTPUT_COLOR -DUNITY_INCLUDE_DOUBLE ../nb-aos-tests.c ../nb-main.c ../nb-aos-support.c nb-aos-alloc.c nb-aos-kda.c ../unity.c -o nbody-aos-ver0-fp > nbody-aos-ver0-fp.optrpt
srun: job 200303 queued and waiting for resources
srun: job 200303 has been allocated resources
$
```

Making a plot

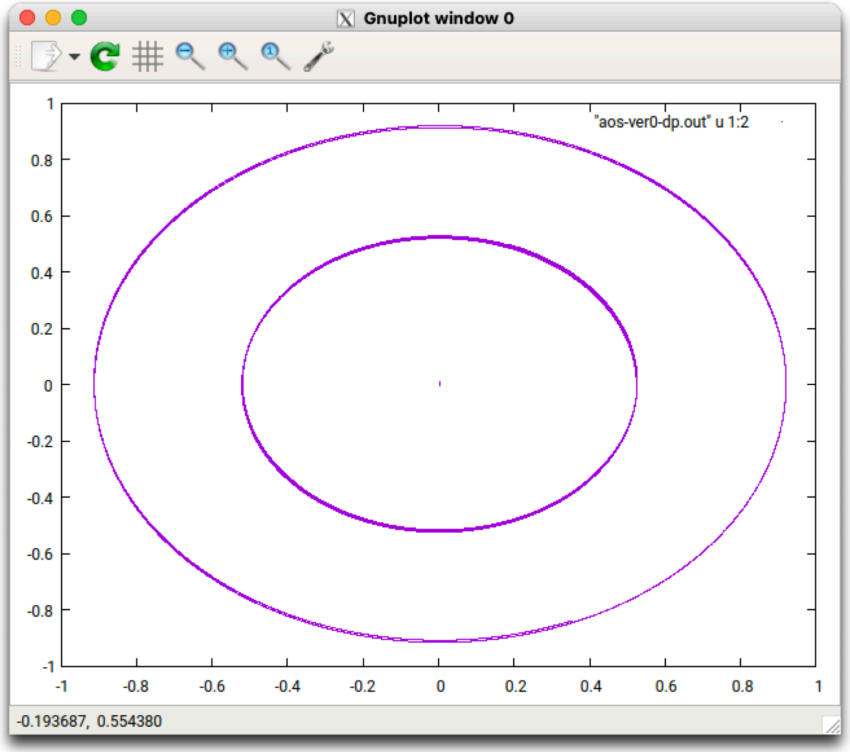
```
$ cd ver0
$ make -Bf ../Makefile nbody-aos-ver0-dp CFLAGS="-O3 -xHost -DFLAGS=aos,O3,arch,double"
$
$ ./nbody-aos-ver0-dp --run-sim=medium --run-perf --overwrite --output=aos-ver0-dp --output-freq=1 -N 3 --dt 0.001 --steps=10000
$
$ gnuplot

        G N U P L O T
        Version 5.2 patchlevel 2    last modified 2017-11-15

        Copyright (C) 1986-1993, 1998, 2004, 2007-2017
        Thomas Williams, Colin Kelley and many others

        gnuplot home:     http://www.gnuplot.info
        faq, bugs, etc:   type "help FAQ"
        immediate help:   type "help"  (plot window: hit 'h')

Terminal type is now 'qt'
gnuplot> plot "aos-ver0-dp.out" u 1:2 w p
```



Making a plot

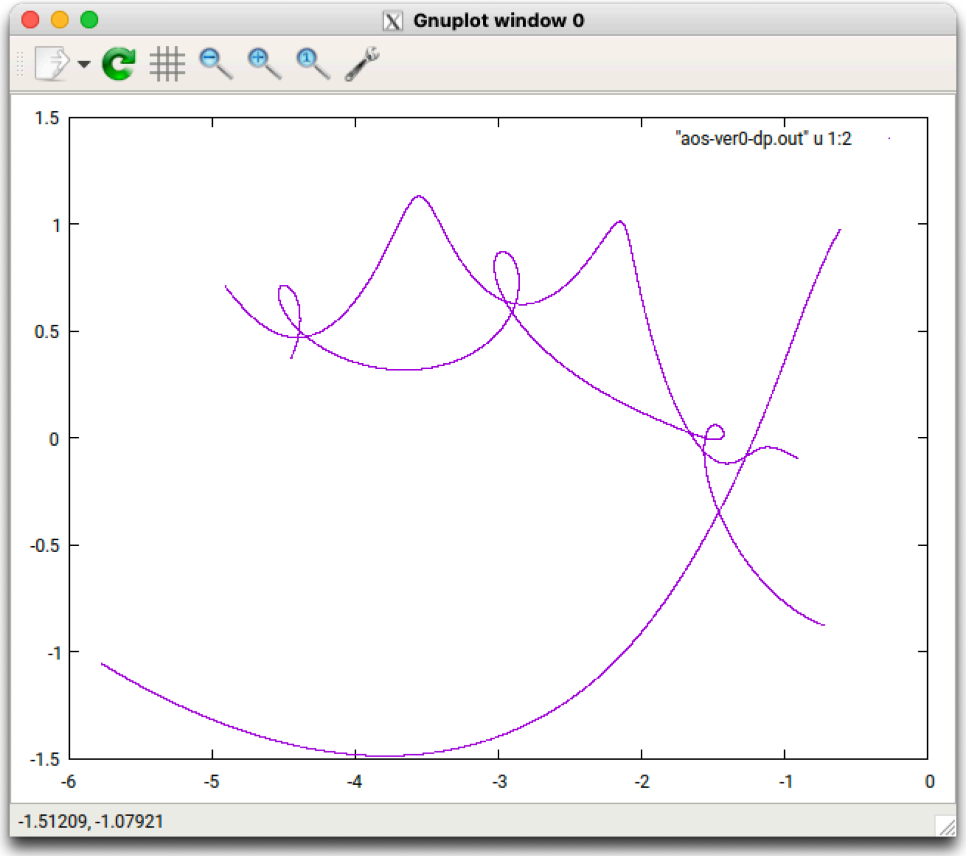
```
$ cd ver0
$ make -Bf ../Makefile nbody-aos-ver0-dp CFLAGS="-O3 -xHost -DFLAGS=aos,O3,arch,double"
$
$ ./nbody-aos-ver0-dp --run-sim=medium --ic=random --run-perf --overwrite --output=aos-ver0-dp --output-freq=1 -N 3 --dt 0.001 --
steps=10000 --status-freq=1
$
$ gnuplot

          G N U P L O T
          Version 5.2 patchlevel 2    last modified 2017-11-15

          Copyright (C) 1986-1993, 1998, 2004, 2007-2017
          Thomas Williams, Colin Kelley and many others

          gnuplot home:     http://www.gnuplot.info
          faq, bugs, etc:   type "help FAQ"
          immediate help:   type "help"  (plot window: hit 'h')

Terminal type is now 'qt'
gnuplot> plot "aos-ver0-dp.out" u 1:2 w p
```



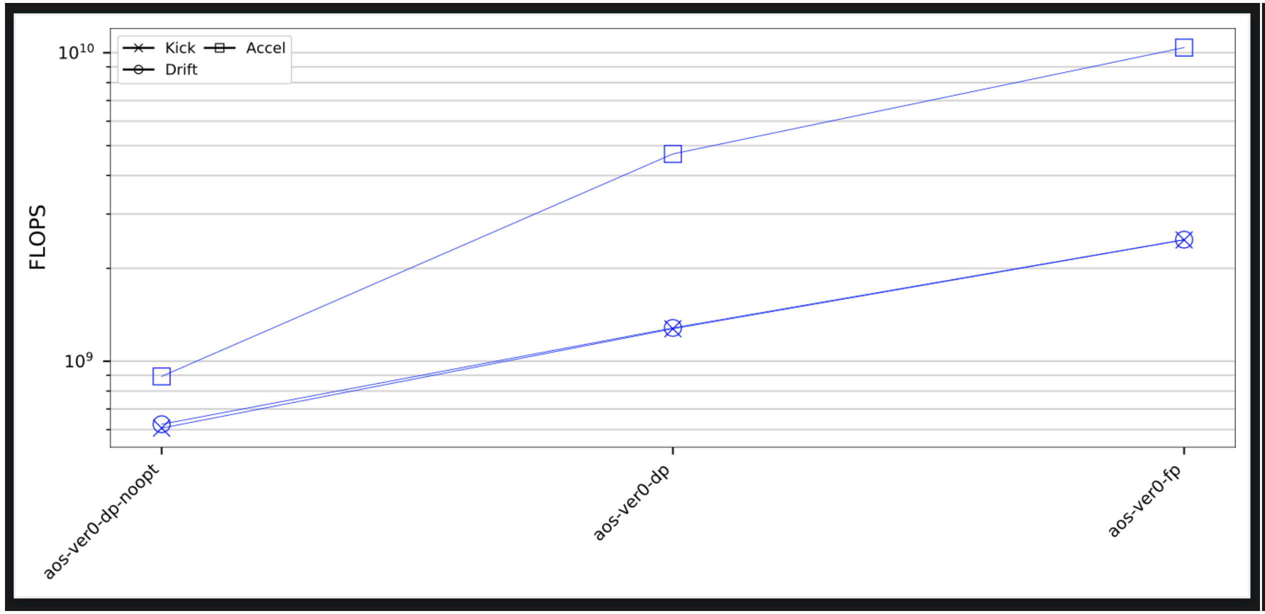
Generating performance data

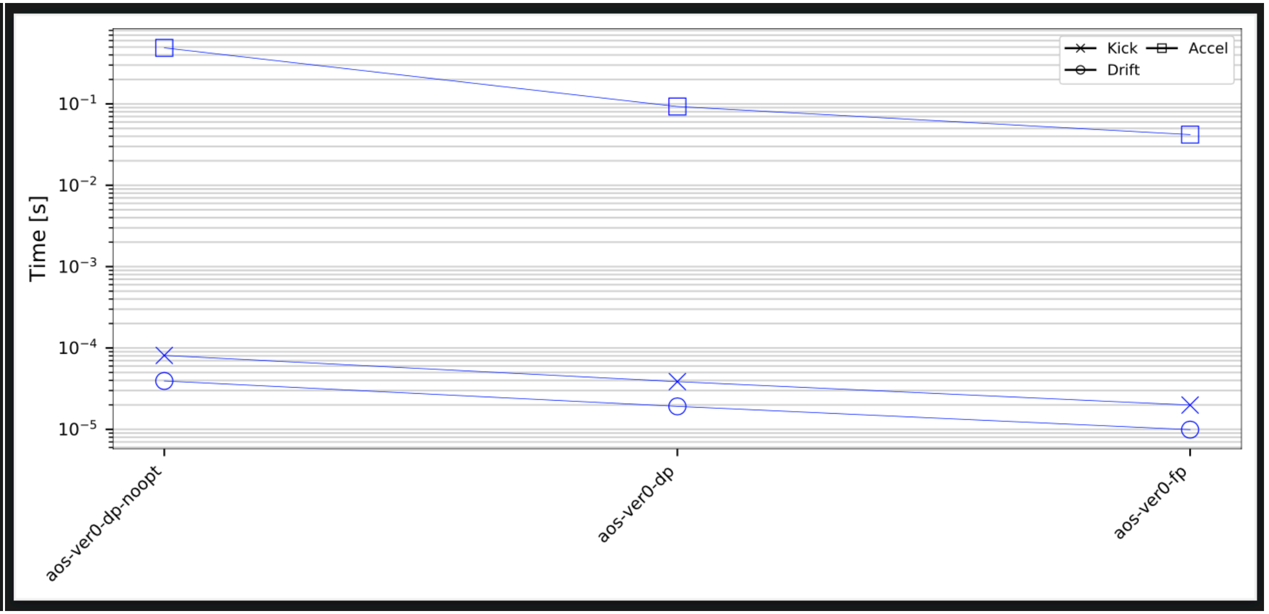
```
$ ./nbody-aos-ver0-dp --run-sim=large --steps=1 --run-perf --overwrite --output=aos-ver0-dp
$
$ cat aos-ver0-dp.perf

#------------------------------------------------------------------------------
META  NEW
META  Date: Sun Jun 26 15:00:19 2022
META  Compiler: Intel
META  Flags: aos,O3,arch,double
META  Npart: 4096
META  Nranks: 1
META  Nthreads: 1
META  TileSize: 1
META  Tag: aos-ver0-dp
HEAD  Kind,Time,Data Moved,Transfer rate,FLOPS,AI
UNIT  -text-, s, GB, GB/s, FLOP/s, FLOP/byte
DATA  Kick   3.70300e-05        5.49316e-04      1.48344e-01      1.32736e+09              0.08333
DATA  Drift  1.85800e-05        2.74658e-04      1.47825e-01      1.32271e+09              0.08333
DATA  Accel  9.28230e-02        1.87491e+00      2.01987e+01      4.69820e+09              0.21662
```

Generating performance data

```
$ cd ver0
$ ./compile_and_run
$
$ python ../plot-perf.py aos-ver0-dp-noopt.perf aos-ver0-dp.perf aos-ver0-fp.perf
```





Switching between float and double

`nb.h`

```
#ifdef USE_FLOAT
typedef float real;
#else
typedef double real;
#endif
```

```
#ifdef USE_FLOAT
#   define   INV(a)    (1.0f / (a))
#   define  SQRT(a)    sqrtf(a)
#   define RSQRT(a)    INV(SQRT(a))
#else
#   define   INV(a)    (1.0 / (a))
#   define  SQRT(a)    sqrt(a)
#   define RSQRT(a)    INV(SQRT(a))
#endif
```

Avoiding array dependencies and Allocating aligned memory

nb.h

```
-DUSE_RESTRICT
#define RESTRICT                        restrict

-DUSE_ASSUME_ALIGNED
#define NB_ASSUME_ALIGNED(var)  __assume_aligned((var), NB_ALIGN)

-DUSE_MM_MALLOC
#define NB_MALLOC(var,sz)       (var) = _mm_malloc((sz),NB_ALIGN)
```

Hands on Sessions

Version 0 - 20 minutes

- Array of Structures data layout

- Goals:
  - Understand compiling and running the nbody code
  - Understand generating plots and what they show
- Activities:
  - Compile with all compiler optimizations disabled (`-O0`)
  - Compile with all compiler optimizations enabled (`-O3 -xHost`)
  - Switch from using `double` as the basic datatype to `float`
  - Use `plot-perf.py` to create a `COW-perf-plot.pdf`
  - Download the file and view it

Version 1 - 15 minutes

- Structure of Arrays data layout

- Goals:
  - Understand the process of switch from AoS to SoA with the nbody code.
- Activities:
  - Compile with all compiler optimizations disabled (`-O0`)
  - Compile with all compiler optimizations enabled (`-O3 -xHost`)
  - Switch from using `double` as the basic datatype to `float`
  - Use `plot-perf.py` to create a `COW-perf-plot.pdf`
  - Download the file and view it

Version 2 - 20 minutes

- Enabling the compiler to vectorize effectively

- Goals:
  - Understand compiler optimization reports
  - Understand how to give hints to the compiler about how to optimize
- Activities:
  - Compile with previous best compiler flags
  - Use combinations of (`-restrict -DUSE_RESTRICT`), (`-no-vec -no-simd`), and (`-align -DUSE_MM_MALLOC -DUSE_ASSUME_ALIGNED`)
  - Use `plot-perf.py` to create a `COW-perf-plot.pdf`
  - Download the file and view it

Version 3 - 20 minutes

- Cache tiling

- Goals:
  - Understand compiler optimization reports
  - Understand how to give hints to the compiler about how to optimize
- Activities:
  - Compile with previous best compiler flags
  - Compile with `-DTILE_SIZE=8` or other sizes. Consider powers of two or other.
  - Use `plot-perf.py` to create a `COW-perf-plot.pdf`
  - Download the file and view it

Version 4 - 30 minutes

- OpenMP

- Goals:
  - Understand adding openmp annotations
- Activities:
  - Enable OpenMP (`-qopenmp`)
  - Enable first touch routine (`-DUSE_FIRST_TOUCH`)
  - Try different tile sizes again
  - Use `plot-perf.py` to create a `COW-perf-plot.pdf`
  - Download the file and view it

lrz

Leibniz Supercomputing Centre

of the Bavarian Academy of Sciences and Humanities